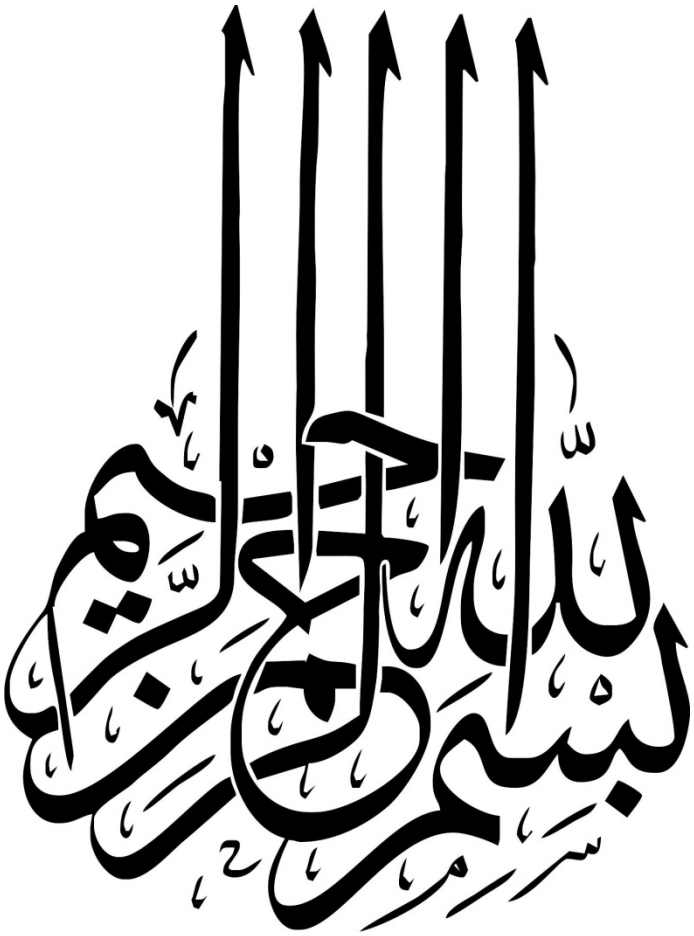

بسم الله الرحمن الرحيم

جاوا به زبان ساده

تالیف و ترجمه : یونس ابراهیمی

Younes.ebrahimi.1391@gmail.com

منبع : w3-farsi.com

تاریخ انتشار : 1395/01/08

این کتاب 900 صفحه A4 می باشد که 200 صفحه اول آن را تقدیم به همه دوستداران این زبان برنامه نویسی می کنم.

آپدیت های جدید را می توانید در سایت منبع آن (w3-farsi.com) مطالعه فرمایید.

Contents

جاوا چیست؟

جاوا یک زبان برنامه نویسی و همچنین یک پلتفرم می باشد. این زبان جزء زبان های سطح بالا و شیءگرا محسوب می شود. برای نخستین بار توسط جیمز گاسلینگ در شرکت سان مایکروسیستمز ایجاد گردید و در سال 1995 به عنوان بخشی ازسکوی جاوا منتشر شد. زبان جاوا شبیه به C++ است اما مدل شیءگرایی آسان‌تری دارد و از قابلیت‌های سطح پایین کمتری پشتیبانی می‌کند .یکی از قابلیت‌های بنیادین جاوا این است که مدیریت حافظه را بطور خودکار انجام می‌دهد. ضریب اطمینان عملکرد برنامه‌های نوشته شده به این زبان بالا است و وابسته به سیستم‌عامل خاصی نیست، به عبارت دیگر می‌توان آن را روی هر رایانه با هر نوع سیستم‌عاملی اجرا کرد. در زیر یک نمونه برنامه جاوا نشان داده شده است:

```
class Sample
{
    public static void main(String args[])
    {
        System.out.println("Hello Java");
    }
}
```

در باره جزییات برنامه بالا در درس های آینده بیشتر توضیح می دهیم. جاوا برای نوشتن انواع برنامه‌های کاربردی مناسب است. با جاوا می‌توان انواع برنامه‌های زیر را نوشت:

- برنامه‌های تحت وب : برنامه های تحت وب همانطور که از نام شان پیداست برنامه هایی هستند که در سمت سرور اجرا می شوند و صفحات وب داینامیک را به وجود می آورند Servlet ، jsp، struts، jsfتکنولوژی های مورد استفاده در جاوا برای ایجاد صفحات وب هستند.
- برنامه‌نویسی سیستم‌های کوچک : برنامه های موبایل در این دسته قرار می گیرند. از JAVA ME برای ساخت این نوع برنامه ها استفاده می شود.
- برنامه‌های کاربردی بزرگ :(Enterprise) از این نوع برنامه می توان به برنامه های بانکی اشاره کرد که در آنها به امنیت بسیار بالا نیاز است. از EJBدر ساخت این نوع برنامه ها در جاوا استفاده می شود.
- برنامه‌های رومیزی :(Desktop) این برنامه ها برنامه هایی هستند که بر روی ویندوز نصب می شوند. برنامه هایی مانند Media Player و آنتی ویروس ها جزو این دسته محسوب می شوند. در جاوا از AWT و Swing برای ساخت برنامه های ویندوزی استفاده می شود.

یکی از ویژگی‌های جاوا قابل حمل بودن آن است. یعنی برنامهٔ نوشته شده به زبان جاوا باید به طور مشابهی در کامپیوترهای مختلف با سخت‌افزارهای متفاوت اجرا شود؛ و باید این توانایی را داشته باشد که برنامه یک بار نوشته شود، یک بار کامپایل شود و در همه کامپیوترها اجرا گردد.

از زمان انتشار اولین نسخه جاوا (java 1.0) تا به امروز، شرکت Sun تقریبا هر دو سال یکبار نسخه ای جدید از زبان برنامه سازی جاوا را منتشر می نماید. در این نسخه تازه، معمولا قابلیت های جدیدی افزوده شده و ایرادهای نسخه قبل رفع می شوند.

نکته قابل توجه در مورد شماره گذاری نسخه های مختلف جاوا آن است که تا چهارمین نسخه آن شماره گذاری بصورت Java 1.xبود که x همان شماره نسخه مورد نظر می باشد. از نسخه پنجم به بعد شماره گذاری بصورت Java x تغییر یافت .یعنی بجای اینکه نسخه پنجم را بصورت Java 1.5 نامگذاری کنند، بصورت java 5.0 نامگذاری کردند.

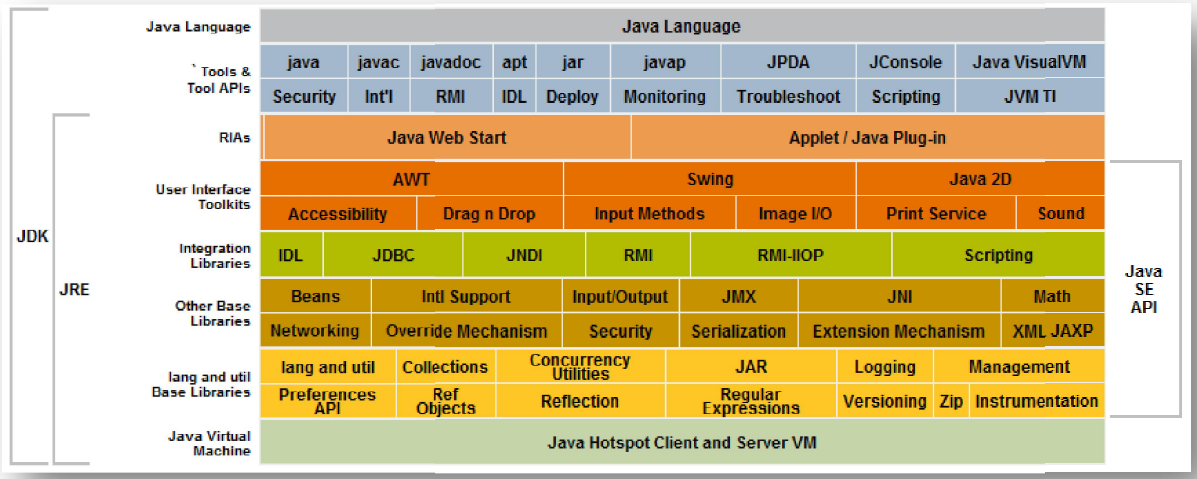
در ادامه به معرفی نسخه های مختلف جاوا بر اساس نسخه پایه ای آن یا همان نسخه استاندارد جاوا (Standard Edition(SE)می پردازیم. این نسخه شامل همه ملزومات مورد نیاز جهت Desktop Programming می باشد. در جدول زیر نسخه های مختلف جاوا و ویژگی های آنها ذکر شده است:

ویژگی ها	نام کد	نسخه	تاریخ پیدایش
• شامل 8 بسته (package) با 212 کلاس • مرورگر Netscape نسخه 2 تا 4 از java 1.0 پشتیبانی می کنند. • مایکروسافت و سایر کمپانی های بزرگ نرم افزاری جاوا را تایید می نمایند .		java 1.0	1995
• شامل 23 بسته با 504 کلاس • بهبود در مدیریت رویدادها(event handeling) ، کلاس های داخلی وJVM (Java Virtual Machine) • مایکروسافت قابلیت پشتیبانی از java 1.1 را در مرورگر IE اضافه می کند. در این زمان اکثر مرورگرها از جاوا پشتیبانی می کنند. • بسته swing با افزایش قابل توجهی در قابلیت های گرافیکی جاوا در این زمان بصورت مستقل از هسته مرکزی جاوا، منتشر گردید. • JavaBeans		java 1.1	1997
• شامل 59 بسته با 1520 کلاس • از این تاریخ به بعد نسخه های جاوا بصورت Java 2 platform نامیده می شوند. • تمامی کدها و ابزارهای تولید شده تا به این تاریخ بصورت متمرکز در یک بسته نرم افزاری متمرکز قرار گرفته و در واقع بصورت Software Development Kit به بازار عرضه گشت. • ایجاد JFC (Java Foundation Classes) که بر مبنای swing پایه ریزی شده و به جهت بهبود وضعیت گرافیکی مورد استفاده قرار می گیرد. • توجه JFC :از اصول Internet Foundation Classes محصول شرکت Netscape Communications استفاده می کند. با فراهم نمودن اجزاء رابط های گرافیکی جهت استفاده در تولید برنامه های کاربردی تجاری و اینترنتی جاوا، سبب افزایش قابلیت های AWT (Abstract Window Toolkit) شده است.	playground	J2SE 1.2	1999

• ایجاد یک IDL جهت پیاده سازی CORBA • افزودن مجموعه ای ازAPI ها جهت پشتیبانی از انواع List,Set,Hash maps و ...			
• شامل 76 بسته با 1842 کلاس • افزایش کارایی با افزوده شدنHotspot virtual machine • JavaSound • Java platform Debugger Architecture (JPDA) • قرارگیری Java Naming and Directory Interface (JNDI)در کتابخانه اصلی و مرکزی جاوا	Kestrel	J2SE 1.3	2000
• شامل 135 بسته با 2991 کلاس • پشتیبانی از(IPv6 (Internet Protocol version 6 • بهبود API مربوط به I/O بخصوص در بخش کار با تصاویر با فرمت های JPEG و PNG (خواندن و نوشتن) • JAXP (یک XML Parser متمرکز به همراه یک پردازشگرXSLT) • توسعه بخش امنیتی با متمرکز کردن و بهبود بخش امنیت و رمزنگاری (JCE, JSSE, JAAS)	Merlin	J2SE 1.4	2002
• شامل 165 بسته با 3000 کلاس • بهبود ساختار جاوا در جهت افزایش سرعت آغاز به کار و کاهش میزان فضای مورد نیاز از حافظه جهت کار(FootPoint) • بهبود زمان کامپایل(compile time) • بهبود وضعیت تبدیل انواع به یکدیگر(Type conversion) • تقویت کارایی حلقهfor ، در این نسخه ساختار حلقه For به گونه ای توسعه یافت که قادر بود فعالیت شمارش خود را بر روی اعضای ساختارهایی مثل مجموعه ها و دیگر ساختار های سلسله مراتبی انجام دهد . • افزوده شدن قابلیت تولید خودکار stub برایRMI	Tiger	J2SE 5.0	2004
• Sunاز این نسخه به بعد نام J2SE را به Java SE تغییر نام داد و "0." را از شماره نسخه های جدید خود حذف نمود. اما هنوز سیستم نام گذاری قدیمی نسخه های جاوا در بین توسعه دهندگان باقی مانده است.(1.6.0) • از این نسخه به بعد دیگر سیستم عامل های قدیمی مثل win9x یا win Me پشتیبانی نمی شود. آخرین نسخه ای که از سیستم عامل های گروه فوق پشتیبانی می کرد j2SE 5.0 update	Mustang	Java SE 6	2006

16بود. • بهبود وضعیت پشتیبانی از وب سرویس ها • JDBC 4.0 • ارتقاء JAXB به نسخه 2 • بهبود وضعیت GUI در جاوا، مانندAPI های Swing ، قابلیت sort وfiltering درtable ها و.. • Java Deployment ToolKit، یک مجموعه از توابع جاوا اسکریپتی برای راحتی بیشتر توسعه و کار با applet ها • کوچکتر کردن Kernel جاوا به منضور کم حجم تر کردن و سریع تر شدن جاوا در هنگام نصب و کم شدن مصرف حافظه . در چنین حالتی هرگاه به بسته های دیگری که بر روی سیستم نصب نشده است نیاز بود، کافیست آنها را دانلود کنید. • بهبود کارایی گرافیک در Java 2D و استفاده از Direct3D و Hardware Acceleration آخرین Update موجود Java SE 6 Update 14 می باشد.			
• پشتیبانی از تکنیک Dynamic Languages توسط JVM (Multi Language Virtual Machine) • ایجاد یک کتابخانه جدید برای پردازش موازی روی پردازنده های چند هسته ای	Dolphin	Java SE 7	2011

دیاگرام زبان برنامه نویسی جاوا، که از سوی شرکت Sun منتشر شده است را در شکل زیر مشاهده می کنید:



به علت تشابه بین این زبان و زبان های خانواده C و همچنین درخواست کاربران عزیز سایت w3-farsi.com مبنی بر آموزش این زبان قدرتمند، یک دوره کامل آموزشی از این زبان را در سایت قرار می دهم و امیدوارم که مورد استقبال شما عزیزان قرار گیرد.

JVM چیست ؟

برای اجرای برنامه‌های نوشته شده و کامپایل شده به زبان جاوا نیاز به سکویی یا برنامه‌ای است که به آن ماشین مجازی جاوا (Java Virtual Machine) یا به اختصار JVM گفته می‌شود. این ماشین کدهای کامپایل شده به زبان جاوا را گرفته و آنها را اجرا می‌کند. شاید این جمله را شنیده باشید که کدهای زبان جاوا بر روی هر ماشین قابل اجرا می‌باشند و اصطلاحا جاوا Multi Platform است.شخصی که دستگاهی با سیستم عامل ویندوز دارد، از سایت سان میکروسیستمز JVM مربوط به سیستم عامل ویندوز را نصب می‌کند. سپس برنامه‌ای را به زبان جاوا می‌نویسد و آن را کامپایل مینماید. پس از آن برنامه کامپایل شده را برای دوست خود که دستگاه دیگری با سیستم عامل لینوکس دارد ارسال می‌کند. این شخص قبلا JVM مخصوص سیستم عامل لینوکس را از سایت سان برداشته و بر روی دستگاه خود نصب نموده است. به همین دلیل هیچکدام از این دو نفر لازم نیست نگران باشد که سیستم عامل دستگاهایشان با یکدیگر متفاوت است. همانطور که از مثال مشخص است کدهای جاوا یکبار کامپایل می‌شوند و همه جا اجرا می‌شوند و این شعار جاوا است: "یک بار کامپایل کنید و همه جا اجرا کنید. "

NetBeans و JDK

Netbeans محیط توسعه یکپارچه ای است که دارای ابزارهایی برای کمک به شما برای توسعه برنامه های JAVA می باشد. توصیه می کنیم که از محیط Netbeans برای ساخت برنامه استفاده کنید، چون این محیط دارای ویژگی های زیادی برای کمک به شما جهت توسعه برنامه های JAVA می باشد. توسط Netbeans می توان در استانداردهای مختلف جاوا مانند J2SE ، J2EE و J2ME برنامه نویسی کرد. همچنین از محیط زبان هایPHP ، HTML، C و نیز Groovy پشتیبانی می کند .قبل از نصب Netbeans می بایست JDK را نصب نمایید، در غیر این صورت برای نصب دچار مشکل خواهید شد JDK .که مخفف عبارت Java Development Toolkit می باشد ترکیبی از کامپایلر زبان جاوا، کلاس های کتابخانه ای(Java Class Libraries) و JVM و فایل های راهنمای آنها می باشد. برای اینکه ما بتوانیم با استفاده از زبان برنامه نویسی جاوا، برنامه بنویسیم به این مجموعه نیاز داریم .تعداد زیادی از پردازش ها که وقت شما را هدر می دهند به صورت خودکار توسط NetBeans انجام می شوند. یکی از این ویژگی ها اینتل لایسنس (Intellisense) است که شما را در تایپ سریع کدهایتان کمک می کند NetBeans .برنامه شما را خطایابی می کند و حتی خطاهای کوچک (مانند بزرگ یا کوچک نوشتن حروف) را برطرف می کند .با این برنامه های قدرتمند بازدهی شما افزایش می یابد و در وقت شما با وجود این ویژگیهای شگفت انگیز صرفه جویی می شود NetBeans آزاد است و می توان ان را دانلود و از آن استفاده کرد.این برنامه ویژگیهای کافی را برای شروع برنامه نویسی JAVA در اختیار شما قرار می دهد.در آموزش ها از NetBeans نسخه 8.0.2 استفاده شده است و استفاده از این نسخه برای انجام تمرینات این سایت کافی می باشد. برای دانلود نرم افزارهای مورد نیاز بر روی لینک زیر کلیک کنید:

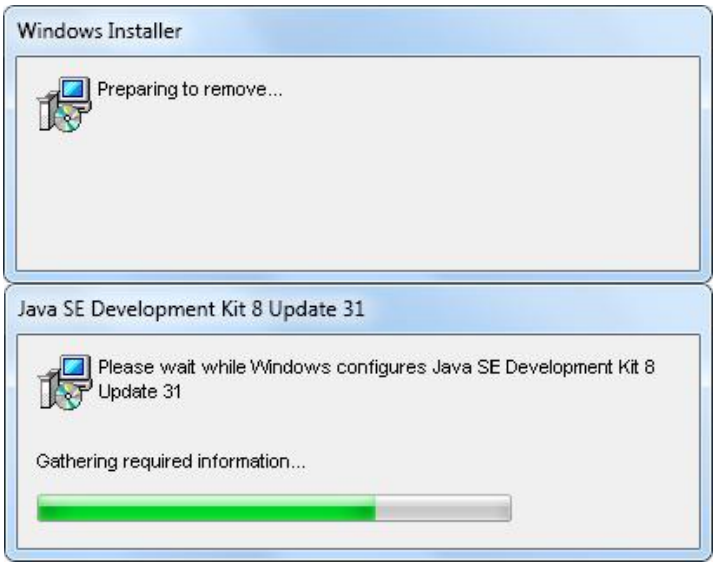
دانلود JDK وNetBeans

در درس آینده مراحل نصب و راه اندازی دو نرم افزار JDK و NetBeans را توضیح می دهیم.

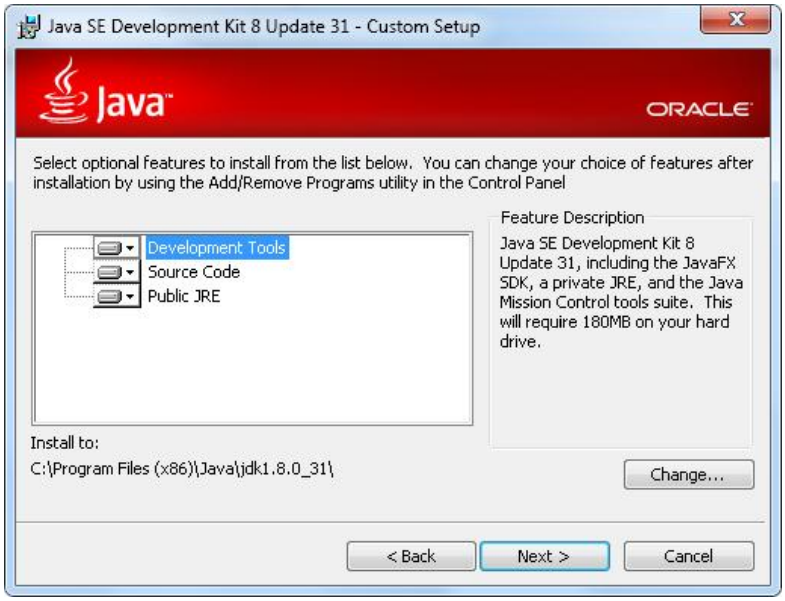
نصب NetBeans و JDK

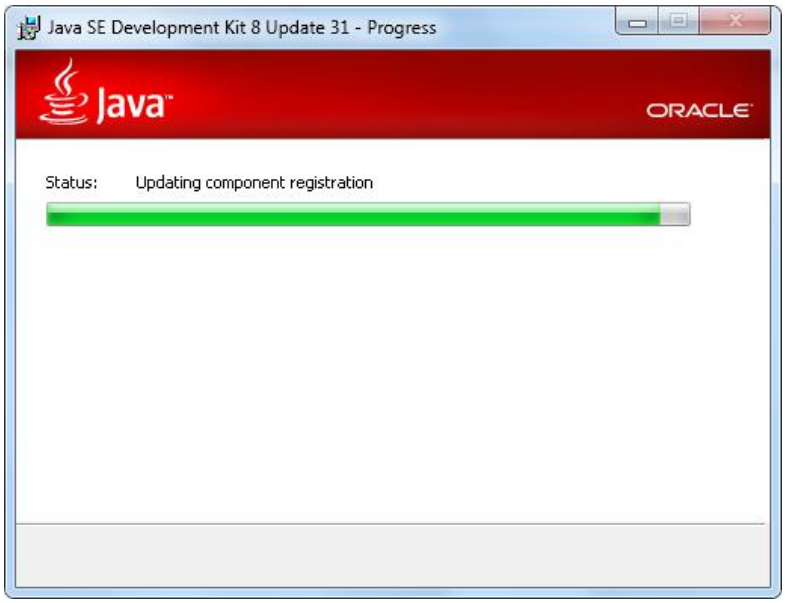
در درس قبل در مورد نرم افزارهای NetBeans و JDK توضیحات مختصری ارائه دادیم. در این درس می خواهیم شما را با نحوه نصب این دو نرم افزار آشنا کنیم. نصب این نرم افزارها مانند اکثر نرم افزارهای دیگر بسیار آسان بود و بعد از زدن چند دکمه Next نصب می شوند. در زیر مراحل تصویری نصب این دو نرم افزار نشان داده شده است.

نصبJDK


Windows Installer
Preparing to remove...
Java SE Development Kit 8 Update 31
Please wait while Windows configures Java SE Development Kit 8 Update 31
Gathering required information...


Java SE Development Kit 8 Update 31 - Setup
Java
ORACLE
Welcome to the Installation Wizard for Java SE Development Kit 8 Update 31
This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 31.
The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.
Next >
Cancel

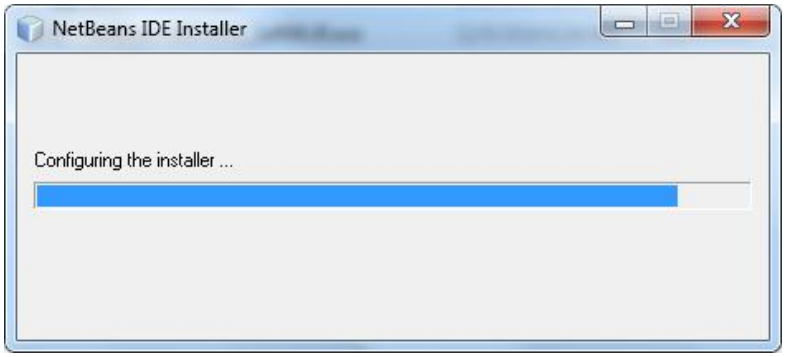

Java SE Development Kit 8 Update 31 - Custom Setup
Java
ORACLE
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel
Development Tools
Source Code
Public JRE
Feature Description
Java SE Development Kit 8 Update 31, including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.
Install to:
C:\Program Files (x86)\Java\jdk1.8.0_31\
Change...
< Back
Next >
Cancel

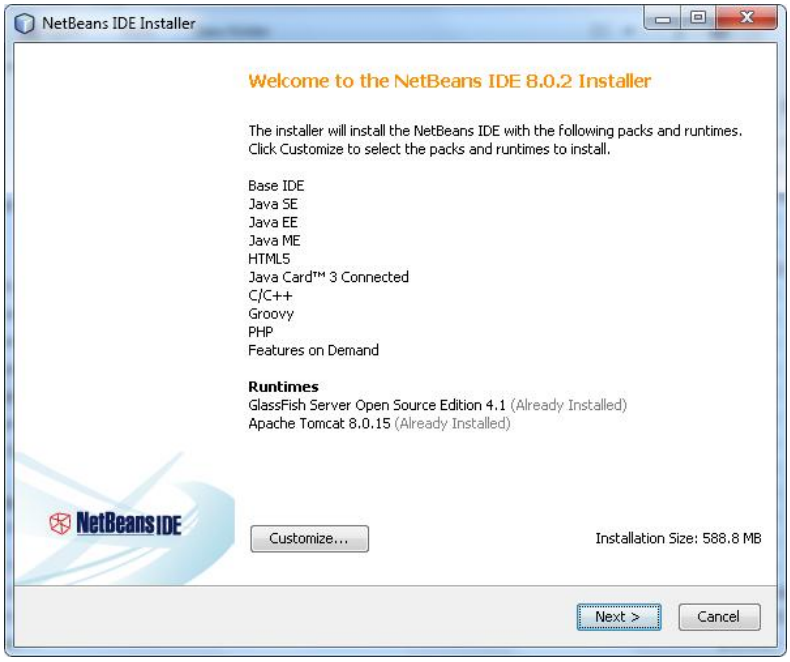

Java SE Development Kit 8 Update 31 - Progress
Java
ORACLE
Status: Updating component registration

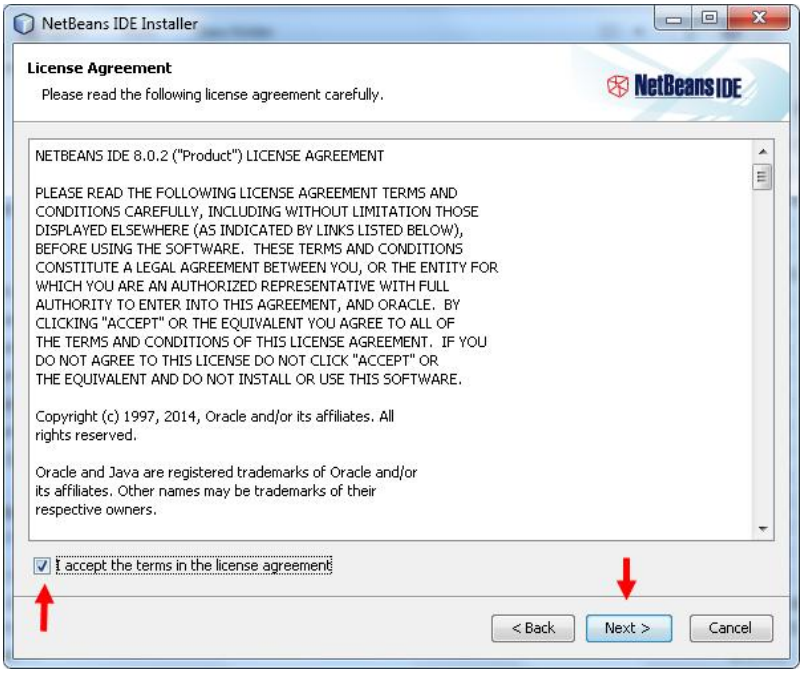

Java Setup - Destination Folder
Java
ORACLE
Destination Folder
Click "Change" to install Java to a different folder.
Install to:
C:\Program Files (x86)\Java\jre1.8.0_31\
Change...
Next >

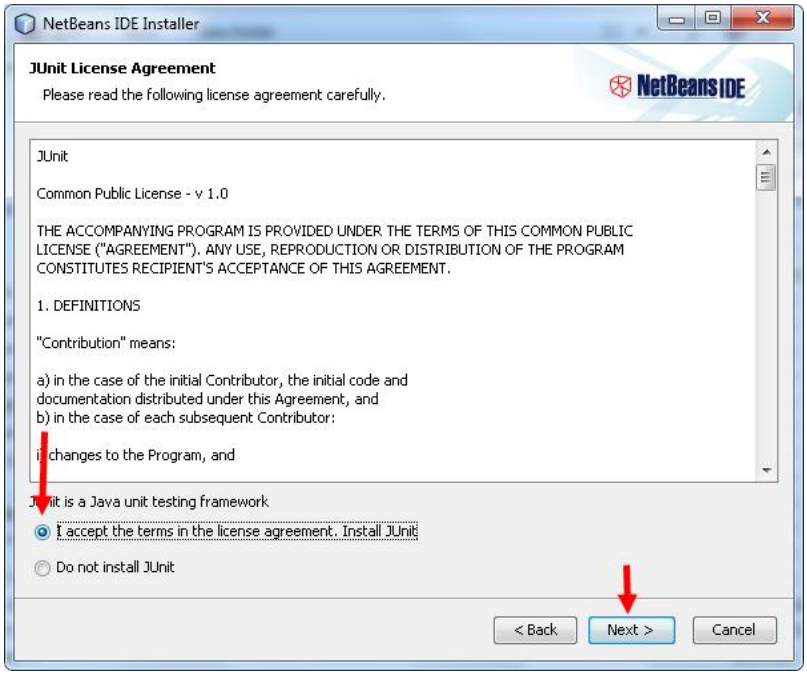

Java Setup - Progress
Java
ORACLE
Status: Installing Java
3 Billion Devices Run Java
Computers, Printers, Routers, Cell Phones, BlackBerry, Kindle, Parking Meters, Public Transportation Passes, ATMs, Credit Cards, Home Security Systems, Cable Boxes, TVs...
ORACLE

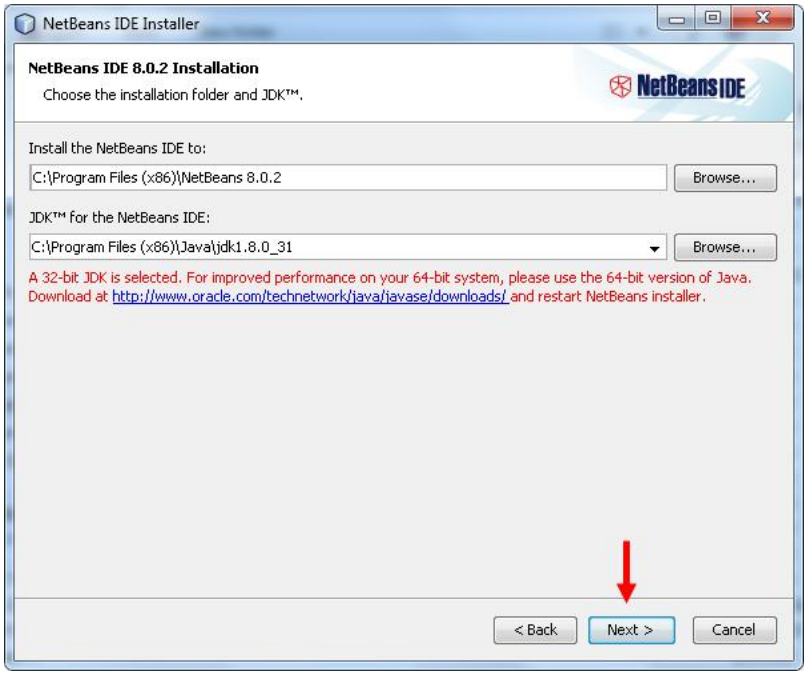
نصب NetBeans

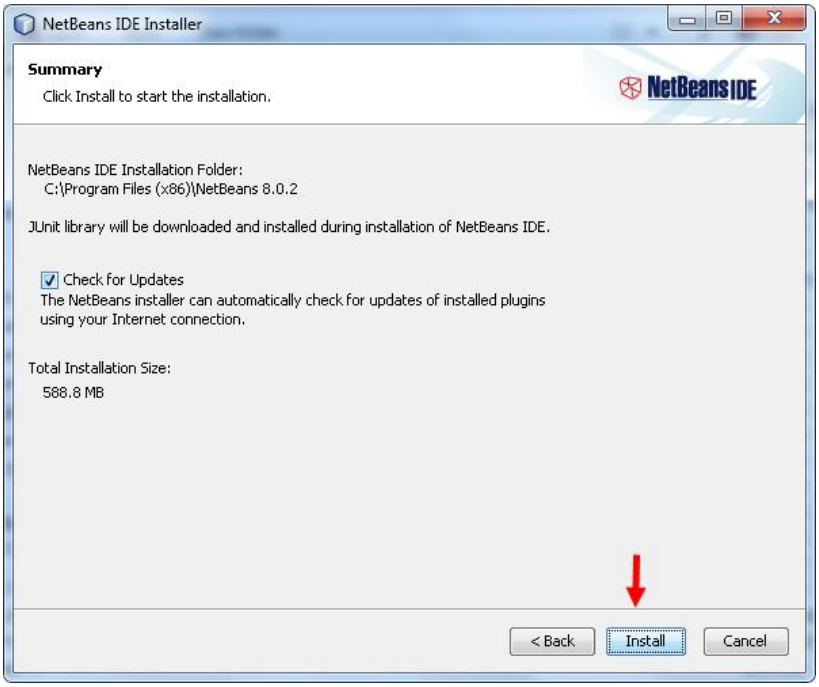


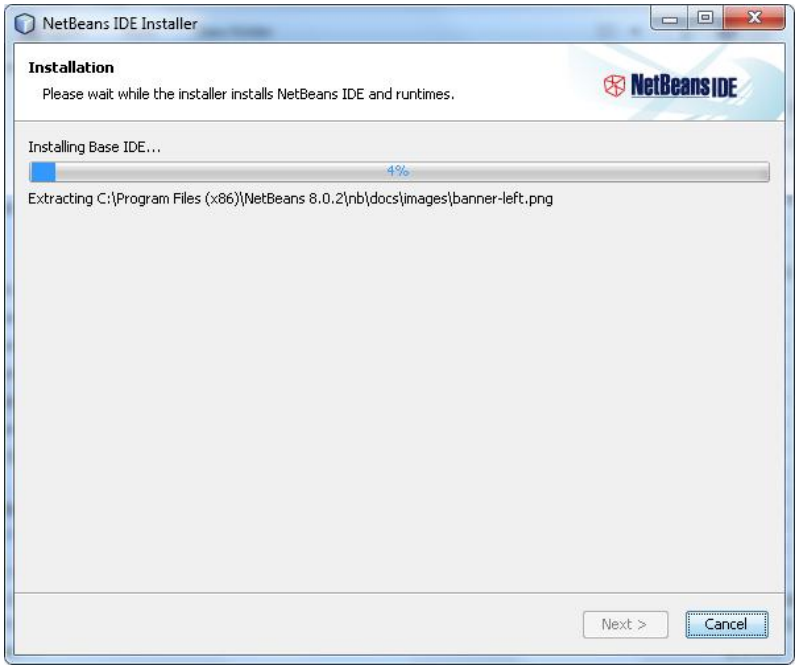

NetBeans IDE Installer
Welcome to the NetBeans IDE 8.0.2 Installer
The installer will install the NetBeans IDE with the following packs and runtimes.
Click Customize to select the packs and runtimes to install.
Base IDE
Java SE
Java EE
Java ME
HTML5
Java Card™ 3 Connected
C/C++
Groovy
PHP
Features on Demand
Runtimes
GlassFish Server Open Source Edition 4.1 (Already Installed)
Apache Tomcat 8.0.15 (Already Installed)
NetBeans IDE
Customize...
Installation Size: 588.8 MB
Next >
Cancel

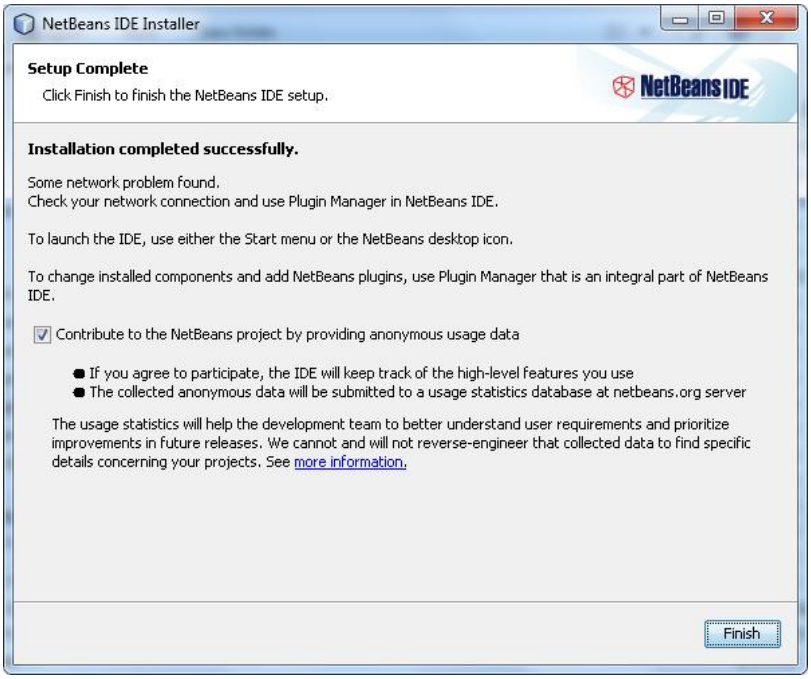

NetBeans IDE Installer
License Agreement
Please read the following license agreement carefully.
NetBeans IDE
NETBEANS IDE 8.0.2 ("Product") LICENSE AGREEMENT
PLEASE READ THE FOLLOWING LICENSE AGREEMENT TERMS AND
CONDITIONS CAREFULLY, INCLUDING WITHOUT LIMITATION THOSE
DISPLAYED ELSEWHERE (AS INDICATED BY LINKS LISTED BELOW),
BEFORE USING THE SOFTWARE. THESE TERMS AND CONDITIONS
CONSTITUTE A LEGAL AGREEMENT BETWEEN YOU, OR THE ENTITY FOR
WHICH YOU ARE AN AUTHORIZED REPRESENTATIVE WITH FULL
AUTHORITY TO ENTER INTO THIS AGREEMENT, AND ORACLE. BY
CLICKING "ACCEPT" OR THE EQUIVALENT YOU AGREE TO ALL OF
THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT. IF YOU
DO NOT AGREE TO THIS LICENSE DO NOT CLICK "ACCEPT" OR
THE EQUIVALENT AND DO NOT INSTALL OR USE THIS SOFTWARE.
Copyright (c) 1997, 2014, Oracle and/or its affiliates. All
rights reserved.
Oracle and Java are registered trademarks of Oracle and/or
its affiliates. Other names may be trademarks of their
respective owners.
I accept the terms in the license agreement
< Back
Next >
Cancel


NetBeans IDE Installer
JUnit License Agreement
Please read the following license agreement carefully.
NetBeans IDE
JUnit
Common Public License - v 1.0
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.
1. DEFINITIONS
"Contribution" means:
a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:
i) changes to the Program, and
JUnit is a Java unit testing framework
I accept the terms in the license agreement. Install JUnit
Do not install JUnit
< Back
Next >
Cancel

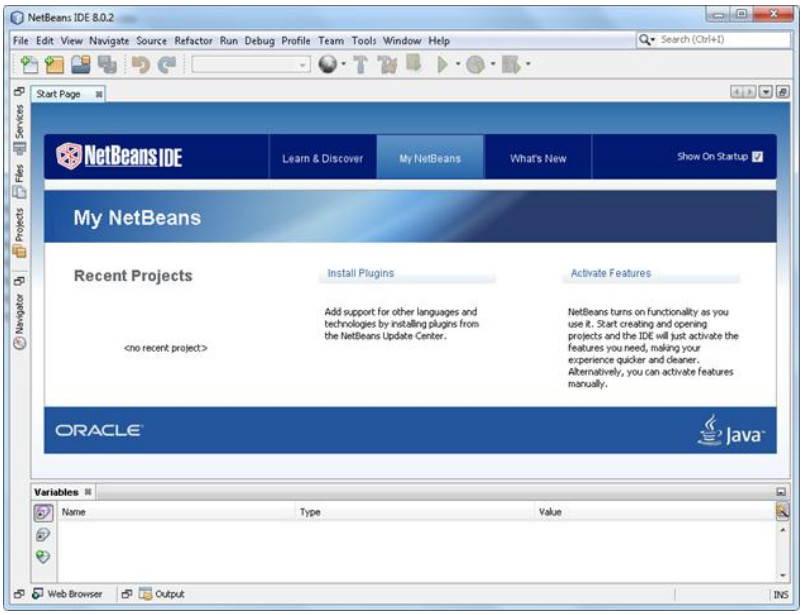

NetBeans IDE Installer
NetBeans IDE 8.0.2 Installation
Choose the installation folder and JDK™.
NetBeans IDE
Install the NetBeans IDE to:
C:\Program Files (x86)\NetBeans 8.0.2
Browse...
JDK™ for the NetBeans IDE:
C:\Program Files (x86)\Java\jdk1.8.0_31
Browse...
A 32-bit JDK is selected. For improved performance on your 64-bit system, please use the 64-bit version of Java.
Download at http://www.oracle.com/technetwork/java/javase/downloads/ and restart NetBeans installer.
< Back
Next >
Cancel


NetBeans IDE Installer
Summary
Click Install to start the installation.
NetBeans IDE
NetBeans IDE Installation Folder:
C:\Program Files (x86)\NetBeans 8.0.2
JUnit library will be downloaded and installed during installation of NetBeans IDE.
Check for Updates
The NetBeans installer can automatically check for updates of installed plugins using your Internet connection.
Total Installation Size:
588.8 MB
< Back
Install
Cancel


NetBeans IDE Installer
Installation
Please wait while the installer installs NetBeans IDE and runtimes.
NetBeans IDE
Installing Base IDE...
4%
Extracting C:\Program Files (x86)\NetBeans 8.0.2\nb\docs\images\banner-left.png
Next >
Cancel



وقتی برای اولین بار بر روی آیکون NetBeans بر روی دسکتاپ کلیک کرده و آن را اجرا می کنید، صفحه اول برنامه به صورت زیر نمایش داده می شود که نشان دهنده نصب کامل آن است:



در درس آینده درباره ایجاد پروژه در NetBeans توضیح می دهیم.

ساخت یک برنامه ساده درJAVA

اجازه بدهید یک برنامه بسیار ساده به زبان جاوا بنویسیم.این برنامه یک پیغام را نمایش می دهد.در این درس می خواهم ساختار و دستور زبان یک برنامه ساده جاوا را توضیح دهم.

قبل از ایجاد برنامه به این نکته توجه کنید که کدهای جاوا را می توان در داخل یک ویرایشگر متن ساده مانند NotePad نوشت و اجرا کرد. فقط کافیست که JDK بر روی سیستم شما نصب باشد. استفاده از نرم افزارهایی مانند NetBeans فقط برای راحتی در کدنویسی و کاهش خطا می باشد .

بدون استفاده ازNetBeans

همانطور که گفته شد، شما برای کامپایل و اجرای برنامه های جاوا به ابزاری به نام JDK نیاز دارید .JDK مخفف عبارت Java Development Kit است و شامل ابزارهای مورد نیاز شما برای اجرای برنامه های جاوا می شود. این مجموعه شامل ابزاری به نام JVM یا Java Virtual Machine است که ماشین مجازی جاوا نام دارد و وظیفه ی کامپایل و اجرای کدهای شما را برعهده دارد. خود JVM هم شامل ابزارهای دیگری است. مثلا javac یا java compiler اختصاصاً وظیفه ی کامپایل کردن برنامه ها را برعهده دارد. در درس قبل JDK را نصب کردیم و الان فرض می کنیم که شما هیچ IDE مانند netbeans و یا eclipse در اختیار ندارید و می خواهید یک برنامه جاوا بنویسید. در این برنامه میخواهیم پیغام Welcome to JAVA Tutorials چاپ شود. ابتدا یک ویرایشگر متن مانند Notepad++ را باز کرده و کدهای زیر را در داخل آن نوشته و با پسوند java ذخیره کنید:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to JAVA Tutorials!");
    }
}
```

نگران توضیح کدهای بالا نباشید، در ادامه در مورد آنها توضیح می دهیم .ما این فایل را در درایو D و با نام و پسوند MyFirstProgram.java ذخیره می کنیم. حال cmd را اجرا می کنیم و کدهای زیر را در داخل آن می نویسیم:

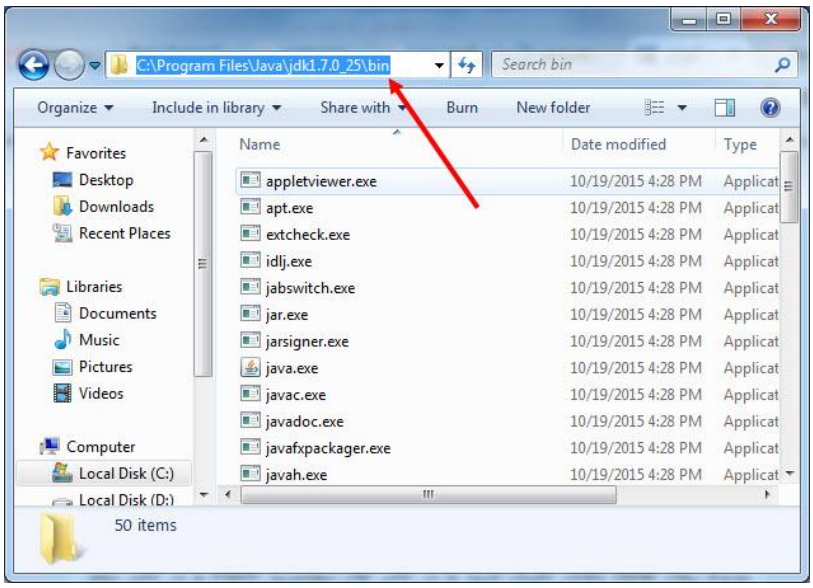
```
Java -version
Javac -version
```

با اجرای کدهای بالا ممکن است که به صورت زیر با خطا مواجه شوید (خطوط قرمز):

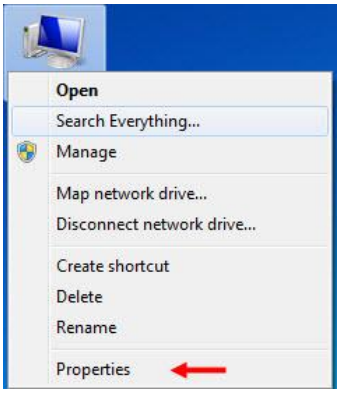
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
```

```
C:\Users\JavaTutorials>java -version
'java' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\JavaTutorials>javac -version
'javac' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\JavaTutorials>
```

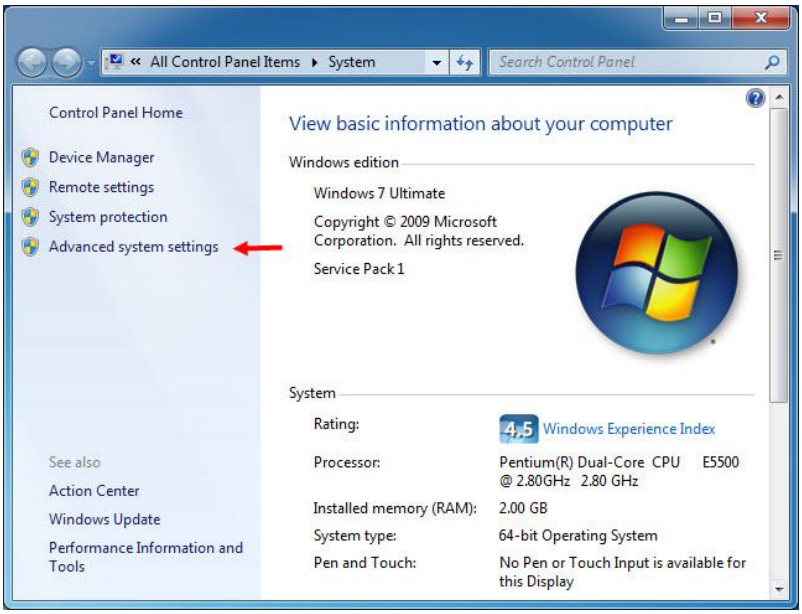
برای رفع خطاهای بالا مراحل زیر را طی کنید. ابتدا مسیر پوشه bin را کپی کنید:



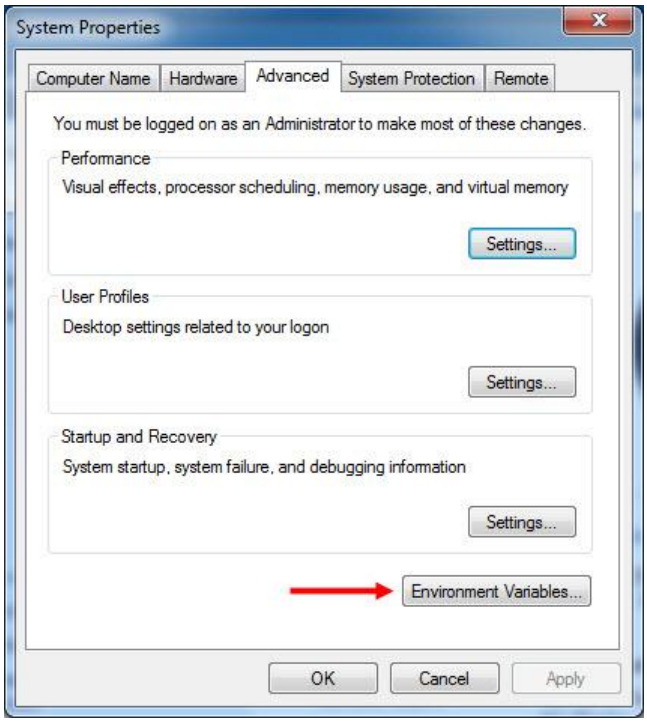
سپس بر روی MyComputer راست کلیک کرده و روی گزینه Properties کلیک کنید:



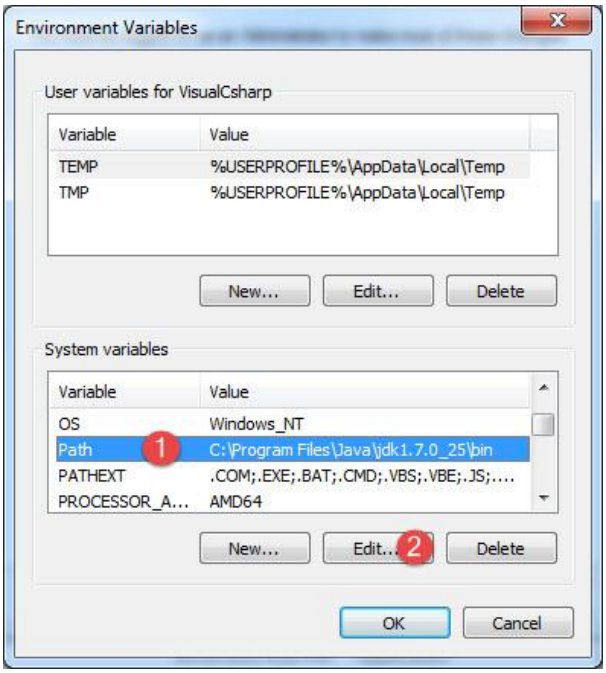
از پنل سمت چپ این صفحه Advanced system settings را باز کنید:



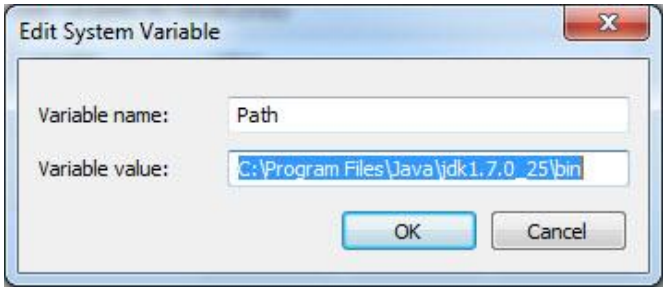
به تب Advanced روی Environment Variables... کلیک کنید.



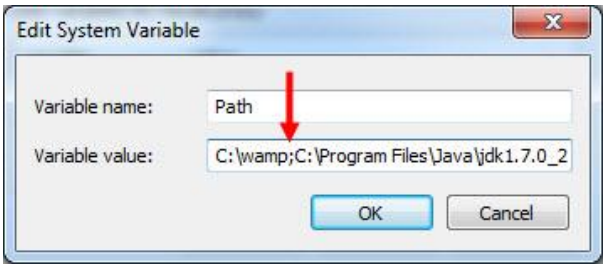
در قسمت پایین و بخش System Variables روی گزینه Path کلیک کرده و سپس گزینه Edit را بزنید:



در پنجره باز شده اگر قسمت Variable Value خالی بود مسیر پوشه bin را در آن کپی کنید:



و اگر از قبل مسیرهای دیگری وجود داشت ابتدا علامت سمیکالن (;) را در انتهای آنها گذاشته و سپس مسیر پوشه bin را کپی می کنید:



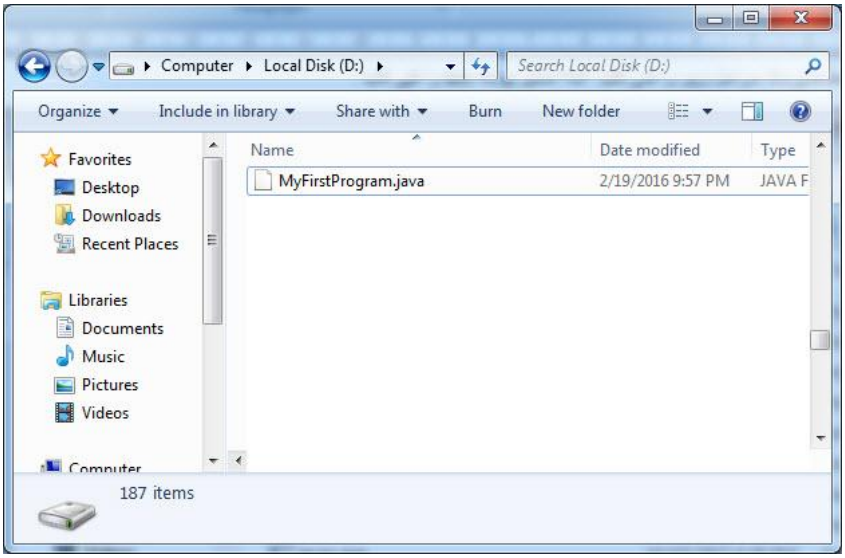
حال پنجره cmd را بسته و دوباره اجرا می کنیم و کد زیر را در داخل آن می نویسیم و دکمه Enter را می زنیم:

```
Javac –version
```

مشاهده می کنید که خطا برطرف شده و نسخه JDK نمایش داده می شود که نشان دهنده این است که مراحل را درست انجام داده اید:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\JavaTutorials>javac -version
javac 1.7.0_25

C:\Users\JavaTutorials>
```

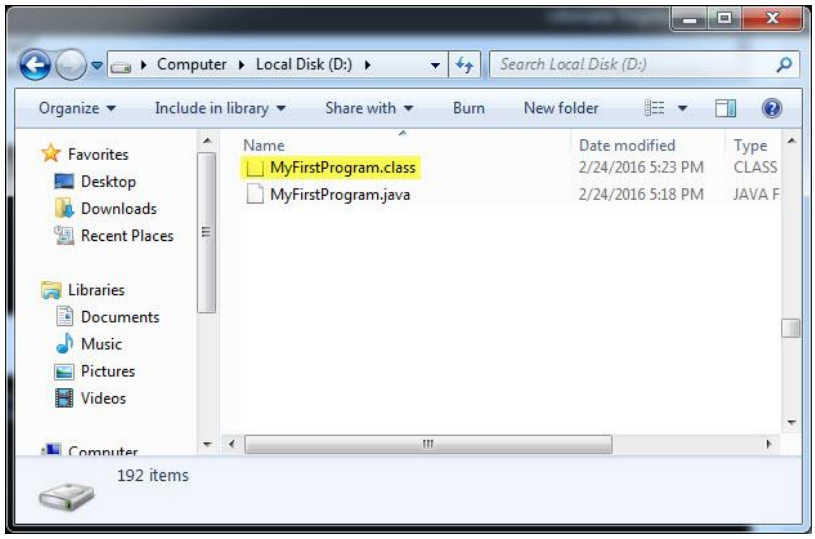
حال نوبت به اجرای برنامه می رسد:



فایل ما در درایو D قرار دارد. ابتدا cmd را باز کرده و کد زیر را در داخل آن نوشته و دکمه Enter را می زنید:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
C:\Users\JavaTutorials>d:

D:\>javac MyFirstProgram.java

D:\>
```

با اجرای کد بالا هیچ پیغامی چاپ نمی شود چون که دستور javac برنامه را کامپایل کرده و یک فایل همنام با کلاس MyFirstProgram و با پسوند .class ایجاد می کند:



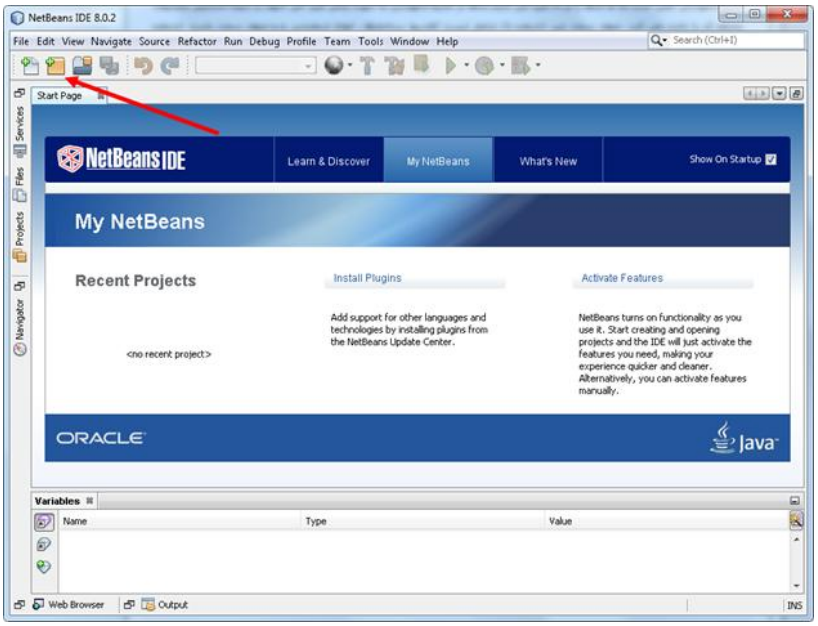
حال برای اجرای فایل MyFirstProgram.class باید دستور زیر را بنویسیم:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\JavaTutorials>d:

D:\>javac MyFirstProgram.java

D:\>java MyFirstProgram
Welcome to JAVA Tutorials!

D:\>
```

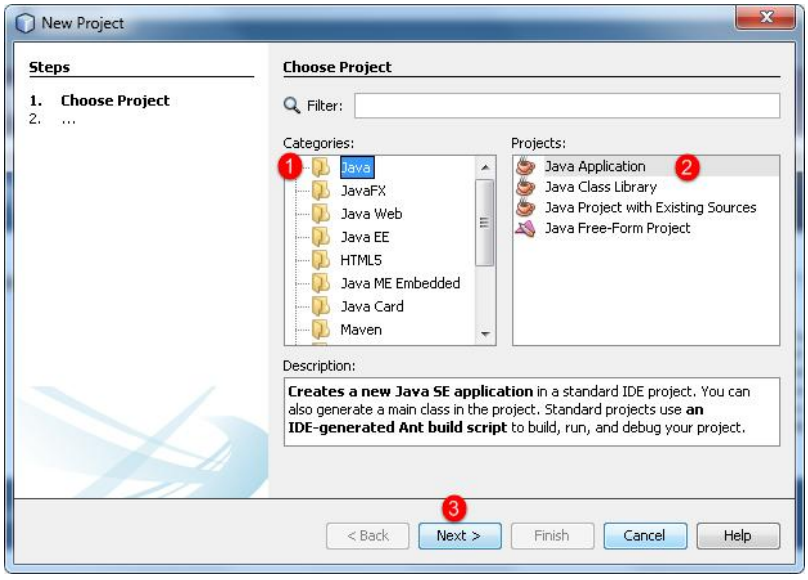
مشاهده می کنید که فایل جاوا اجرا و پیغام Welcome to JAVA Tutorials چاپ شد.

با استفاده ازNetBeans

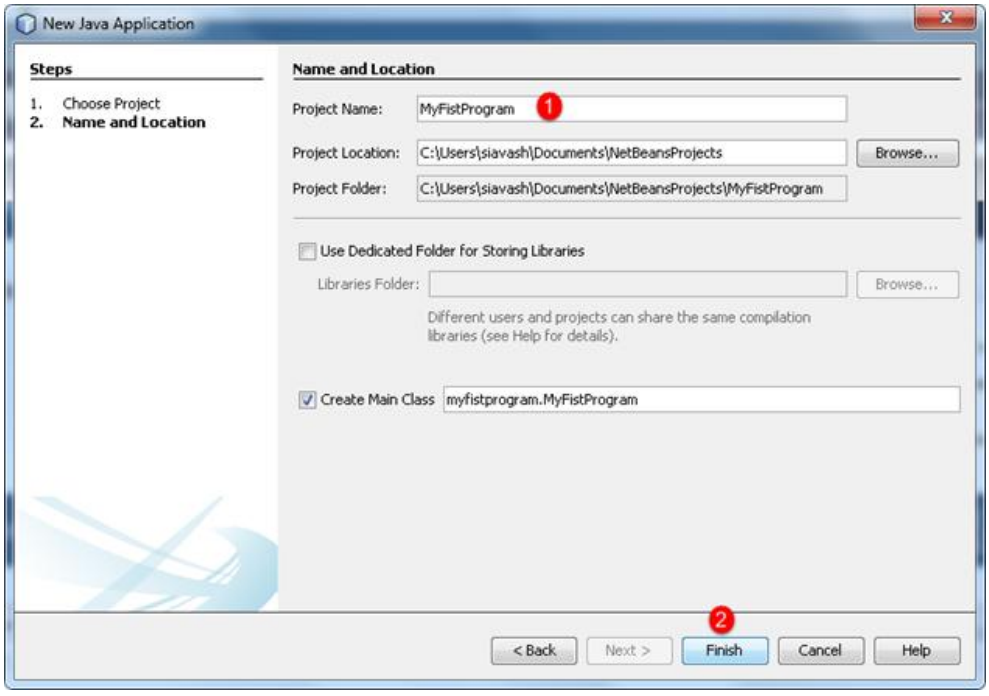
برنامه NetBeans را اجرا کنید. از مسیری که در شکل زیر نشان داده شده است یک پروژه جدید ایجاد کنید:



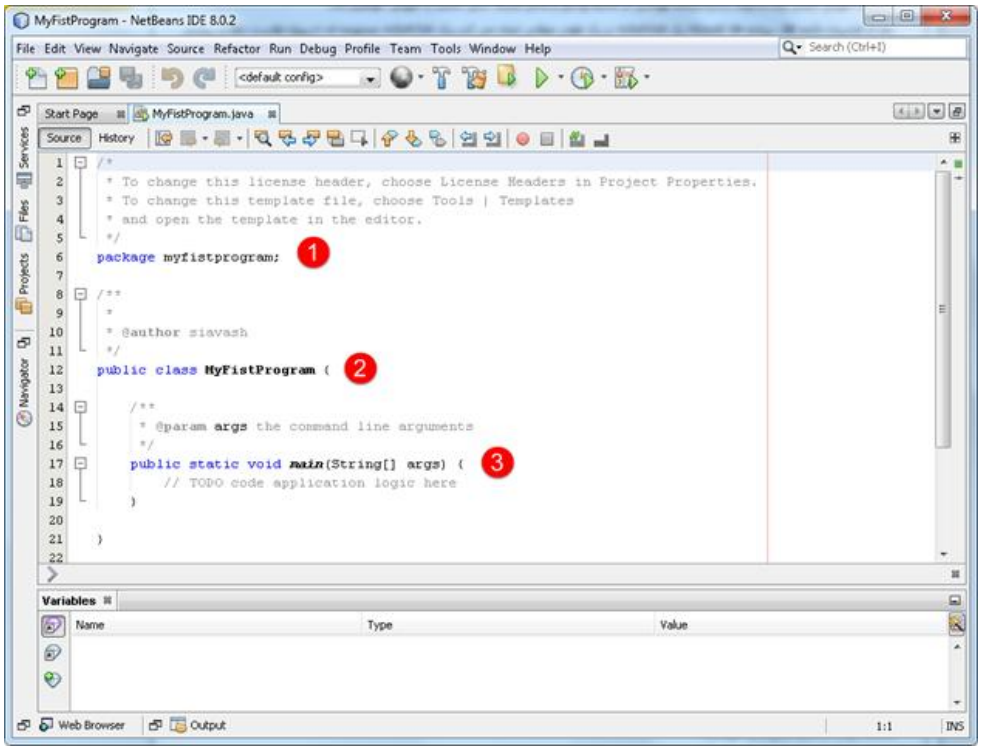
حال با یک صفحه مواجه می شوید. طبق شماره هایی که در شکل زیر نمایش داده شده اند گزینه ها را انتخاب کرده و به مرحله بعد بروید:



با زدن دکمه Next صفحه ای به صورت زیر نمایش داده می شود. در این پنجره نام پروژه تان (MyFirstProgram) را نوشته و سپس بر روی دکمه Finish کلیک کنید:



بعد از فشردن دکمه Finish ، وارد محیط کدنویسی برنامه به صورت زیر می شویم:



محیط کدنویسی یا IDE جایی است که ما کدها را در آن تایپ می کنیم.کدها در محیط کدنویسی به صورت رنگی تایپ می شوند در نتیجه تشخیص بخشهای مختلف کد را راحت می کند. همانطور که در شکل بالا مشاهده می کنید ما کدهای پیشفرض را به سه قسمت تقسیم کرده ایم. قسمت اولPackage ، قسمت دوم کلاس و قسمت سوم متدmain(). نگران اصطلاحاتی که به کار بردیم نباشید آنها را در فصول بعد توضیح خواهم داد. در محل کد نویسی کدهایی از قبل نوشته شده که برای شروع شما آنها را پاک کنید و کدهای زیر را در محل کدنویسی بنویسید:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to JAVA Tutorials!");
    }
}
```

ساختار یک برنامه در جاوا

مثال بالا ساده ترین برنامه ای است که شما می توانید در جاوا بنویسید .هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است. اجازه بدهید هر خط کد را در مثال بالا توضیح بدهیم. در خط اول Package تعریف شده است که شامل کدهای نوشته شده توسط شما است و از تداخل نامها جلوگیری می کند. در باره Package در درسهای آینده توضیح خواهیم داد. در خط دوم آکولاد ({) نوشته شده است.آکولاد برای تعریف یک بلوک کد به کار می رود. جاوا یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می باشد. هر آکولاد باز({) در جاوا باید دارای یک آکولاد بسته (}) نیز باشد.همه کدهای نوشته شده از خط 2 تا خط 6 یک بلوک کد است. در خط 2 یک کلاس تعریف شده است.در باره کلاسها در فصلهای آینده توضیح خواهیم داد.در مثال بالا کدهای شما باید در داخل یک کلاس نوشته شود.بدنه کلاس شامل کد های نوشته شده از خط 2 تا 6 می باشد. خط 3 متد main() یا متد اصلی نامیده می شود. هر متد شامل یک سری کد است که وقتی اجرا می شوند که متد را صدا بزنیم. در باره متد و نحوه صدا زدن آن در فصول بعدی توضیح خواهیم داد.متد main() نقطه آغاز اجرای برنامه است.این بدان معناست که ابتدا تمام کدهای داخل متد main() و سپس بقیه کدها اجرا می شود.در باره متد main() در فصول بعدی توضیح خواهیم داد.متد main() و سایر متدها دارای آکولاد و کدهایی در داخل آنها می باشند و وقتی کدها اجرا می شوند که متدها را صدا بزنیم.هر خط کد در جاوا به یک سیمیکولن (;) ختم می شود.اگر سیمیکولن در آخر خط فراموش شود برنامه با خطا مواجه می شود. مثالی از یک خط کد در جاوا به صورت زیر است:

```
System.out.println("Welcome to JAVA Tutorials!");
```

این خط کد پیغام Welcome to JAVA Tutorials! را در صفحه نمایش نشان می دهد.از متد println() برای چاپ یک رشته استفاده می شود.یک رشته گروهی از کاراکترها است که به وسیله دابل کوتیشن (") محصور شده است. مانند:

"Welcome to Visual C# Tutorials!"

یک کاراکتر می تواند یک حرف،عدد، علامت یا ... باشد. در کل مثال بالا نحوه استفاده از متد println() نشان داده شده است. این متد یک متد از کلاس PrintStream بوده و از آن برای چاپ مقدیر استفاده می شود .out یک فیلد استاتیک کلاس System و کلاس System هم هم یک کلاس از پیش تعریف شده در جاوا می باشد.جاوا فضای خالی و خطوط جدید را نادیده می گیرد.بنابراین شما می توانید همه برنامه را در یک خط بنویسید.اما اینکار خواندن و اشکال زدایی برنامه را مشکل می کند. یکی از خطاهای معمول در برنامه نویسی فراموش کردن سیمیکولن در پایان هر خط کد است.به مثال زیر توجه کنید:

```
System.out.println("Welcome to JAVA Tutorials!");
```

جاوا فضای خالی بالا را نادیده می گیرد و از کد بالا اشکال نمی گیرد. اما از کد زیر ایراد می گیرد.

```
System.out.println(;
        "Welcome to JAVA Tutorials!");
```

به سیمیکولن آخر خط اول توجه کنید.برنامه با خطای نحوی مواجه می شود چون دو خط کد مربوط به یک برنامه هستند و شما فقط باید یک سیمیکولن در آخر آن قرار دهید. همیشه به یاد داشته باشید که جاوا به بزرگی و کوچکی حروف حساس است.یعنی به طور مثال MAN و man در جاوا با هم فرق دارند. رشته ها و توضیحات از این قاعده مستثنی هستند که در درسهای آینده توضیح خواهیم داد .مثلا کدهای زیر با خطا مواجه می شوند و اجرا نمی شوند:

```
system.out.println("Welcome to JAVA Tutorials!");
SYSTEM.OUT.PRINTLN("Welcome to JAVA Tutorials!");
sYsTem.oUt.pRinTLn("Welcome to JAVA Tutorials!");
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می کند. اما کد زیر کاملا بدون خطا است:

```
System.out.println("Welcome to JAVA Tutorials!");
```

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
  statement1;
}
```

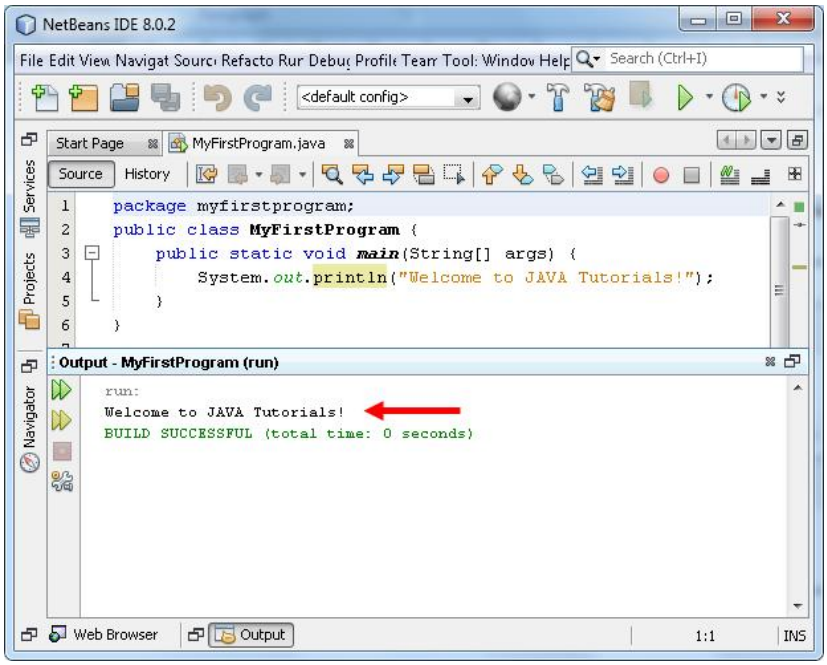
این کار باعث می شود که کدنویسی شما بهتر به چشم بیاید و تشخیص خطاها راحت تر باشد. یکی از ویژگیهای مهم جاوا نشان دادن کدها به صورت تو رفتگی است بدین معنی که کدها را به صورت تو رفتگی از هم تفکیک می کند و این در خوانایی برنامه بسیار موثر است.

ذخیره پروژه و اجرای برنامه

برای ذخیره پروژه و برنامه می توانید به مسیر File > Save All بروید یا از کلیدهای میانبر Ctrl+Shift+S استفاده کنید. همچنین می توانید از قسمت Toolbarبر روی شکل زیر کلیک کنید:

و برای اجرای برنامه هم از فلش سبزرنگ موجود در Toolbar و یا دکمه F6 استفاده کنید:

با اجرای برنامه بالا مشاهده می کنید که رشته Welcome to JAVA Tutorials! در خروجی برنامه به صورت زیر نمایش داده می شود:



وجود خط سبز در پایین فلش قرمز در شکل بالا نشان دهنده اجرای بدون نقص برنامه می باشد. حال که با خصوصیات و ساختار اولیه جاوا آشنا شدید در دسهای آینده مطالب بیشتری از این زبان برنامه نویسی قدرتمند خواهید آموخت.

استفاده ازPackage

برای دسته بندی کلاس ها و قرار دادن کلاس های مرتبط با هم در یک مکان، جاوا از مفهومی به نام بسته یا package استفاده می کند Package .معادل فضای نام در سی شارپ هستند.یک دلیل برای گروه بندی کلاس ها در package این است که امکان دارد دو برنامه نویس از دو کلاس هم نام استفاده کنند.با این کار از چنین برخوردهایی جلوگیری به عمل می آید.یعنی اگر دو کلاس هم نام در دو Package غیر همنام باشند مشکلی به وجود نمی آید. همانطور که در مثال بالا دیدید به طور پیشفرض

NetBeatns هنگام ایجاد برنامه یک Package با همنام یا اسمی که برای برنامه انتخاب کرده ایم با حروف کوچک و در داخل این Package هم کلاسی به همین اسم ایجاد می کند:

```
package myfirstprogram;

public class MyFirstProgram
{
    ...
}
```

برای وارد کردن کلاس یک Package در داخل Package دیگر از کلمه کلیدی import به صورت زیر استفاده می شود:

```
import PackageName.ClassName
```

همانطور که در مثال بالا مشاهده می کنید برای استفاده از کلاسی که در یک Package قرار دارد در Package دیگر ابتدا کلمه import سپس نام Package ، بعد علامت نقطه و در آخر نام کلاس را می نویسیم. مثلا برای استفاده از کلاس MyFirstProgram مربوط به پکیج myfirstprogram به صورت زیر عمل می شود:

```
import myfirstprogram.MyFirstProgram;
```

بسته ها را می توان به صورت تو در تو تعریف کرد. در این حالت در تعریف بسته یک کلاس، از بیرونی ترین بسته شروع کرده و هر بسته را با نقطه (.) به بسته بعدی متصل می کنیم:

```
import firstPackage.secondPackage.ClassName
```

ایجاد، نامگذاری و استفاده از Package ها

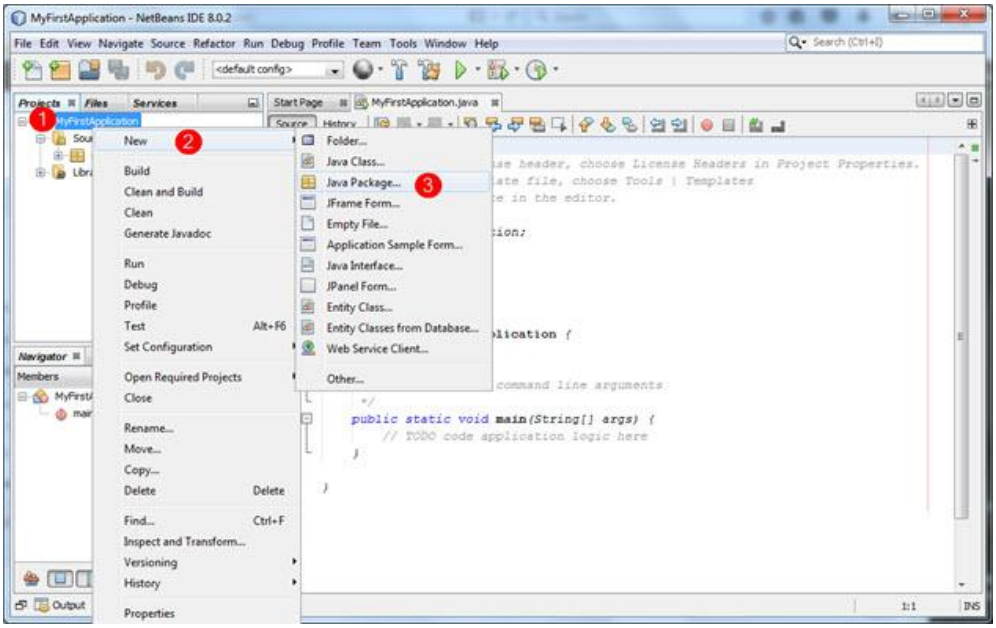
از انجاییکه Package در جاوا از اهمیت ویژه ای برخوردار است و باعث سازمان دهی برنامه می شوددر این درس به نحوه ایجاد، نامگذاری و استفاده از آنها می پردزیم. فرض کنید که شما با چند نفر به صورت گروهی و یا در یک شرکت کار می کنید. ممکن است که شما روی یک پروژه بزرگ کار کنید که دارای صدها کلاس باشد در چنین مواقعی برای سازماندهی و کنترل بیشتر بر کلاس های برنامه، بهتر است آنها را در داخل Package سازماندهی کنید. این کار باعث جلوگیری از درگیری در نامگذاری کلاس ها می شود بدین معنی که اگر شما و همکارتان بصورت اتفاقی یک کلاسی را فراخوانی کنید،اگر کلاس های شما در پکیج های مختلفی باشند هیچ مشکلی پیش نخواهد آمد. برای نامگذاری Packge ها معمولا به صورت قراردادی از نام دامنه سایت شرکت به صورت برعکس استفاده می کنند، چون که دامنه تکراری وجود ندارد و منحصر به فرد است:

```
com.DomainName.www
```

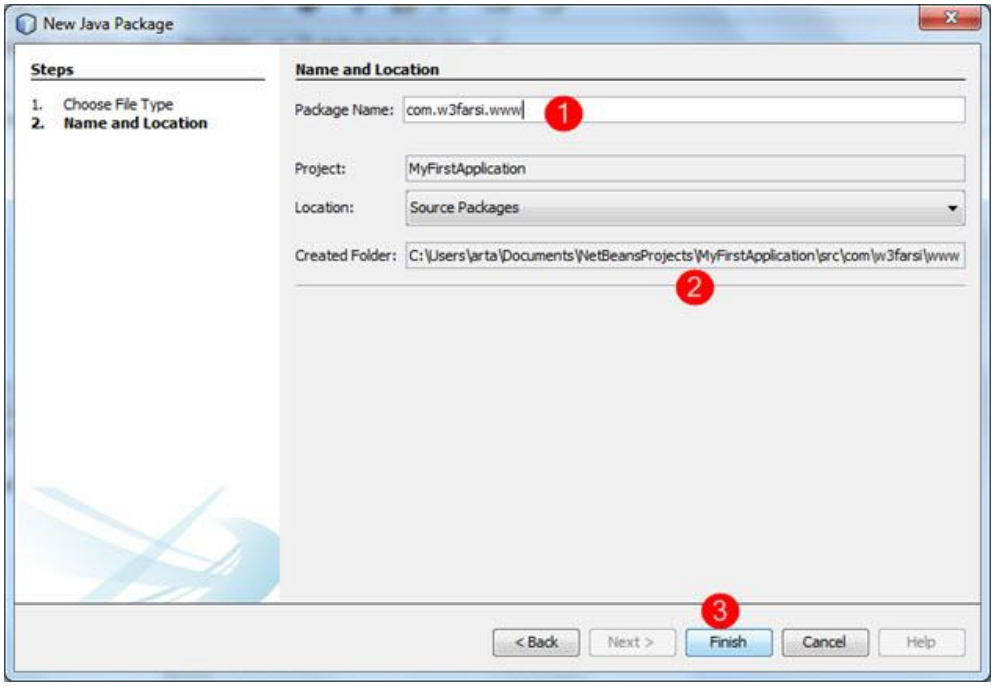
مثلا

```
com.w3farsi.com
```

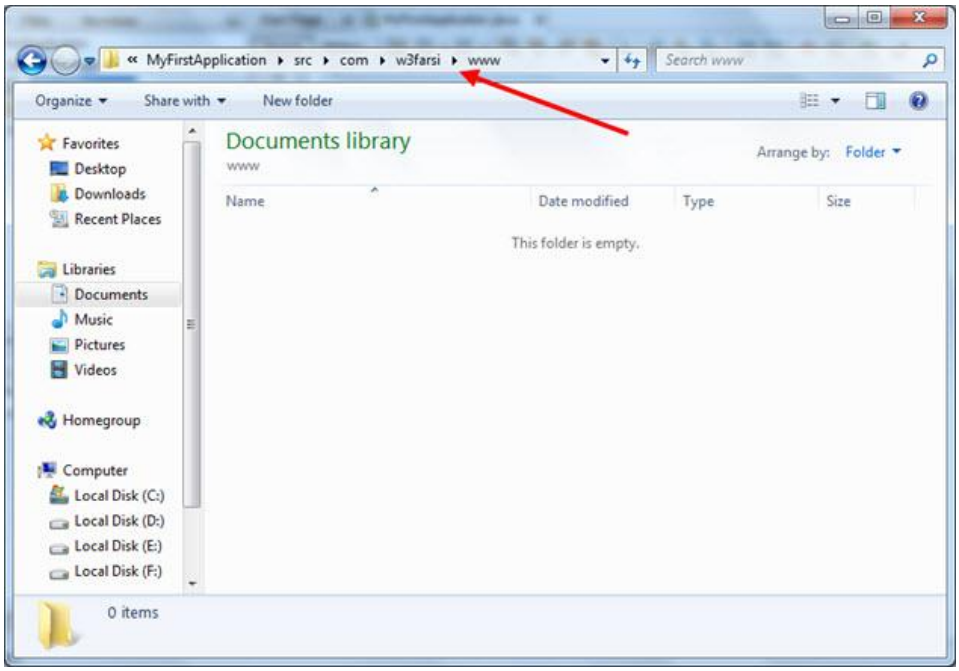
برای ایجاد Package در NetBeans به صورت زیر عمل می شود .بر روی نام پروژه تان کلیک راست کرده و یک Package ایجاد کنید:



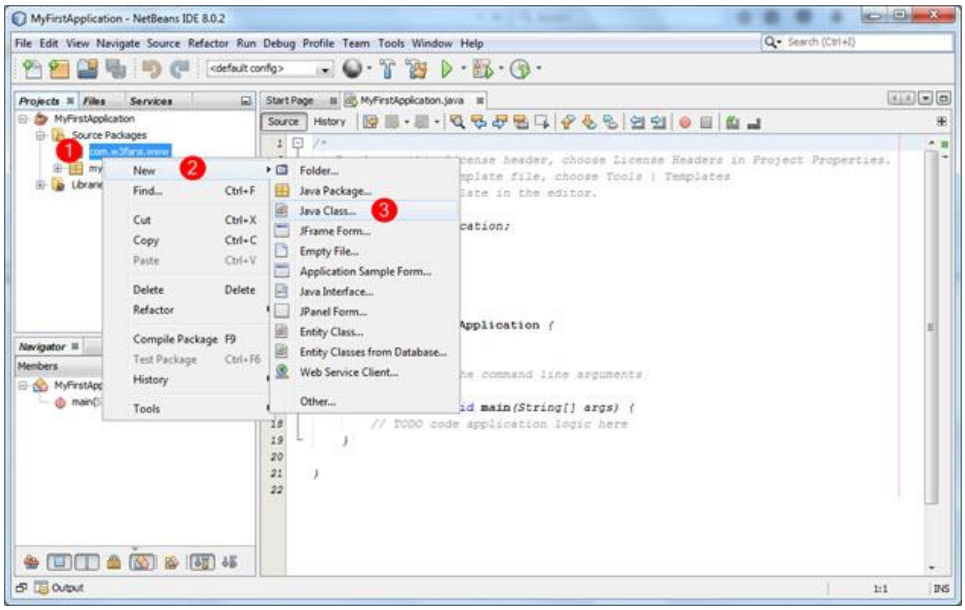
در مرحله بعد به صورت زیر یک نام برای آن انتخاب کرده و دکمه finish را بزنید. همانطور که در شکل زیر مشاهده می کنید Package در داخل پوشه src که در پوشه برنامه است ایجاد می شود:



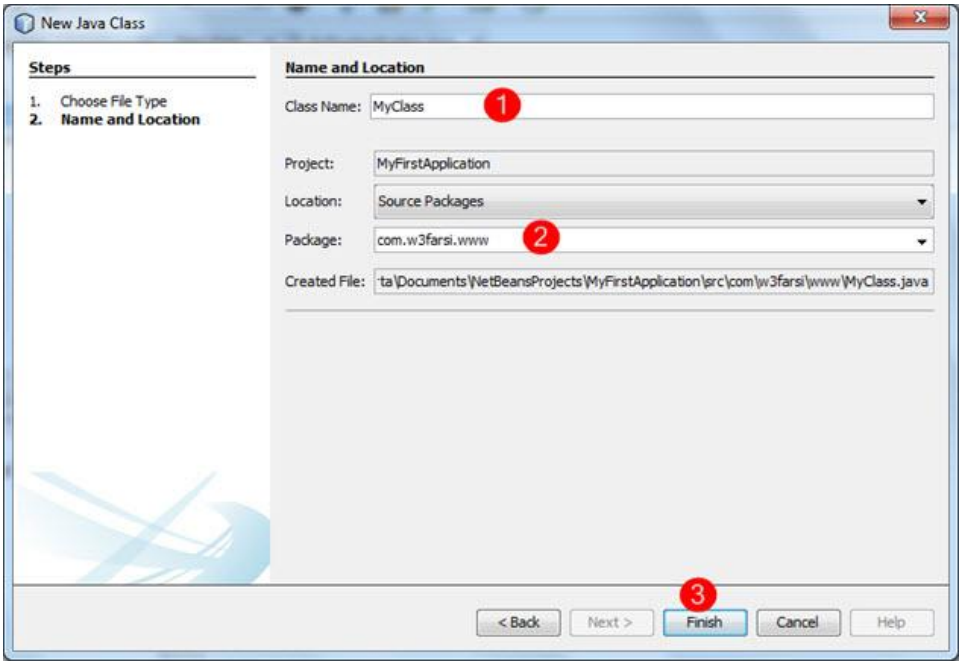
حال اگر به داخل پوشه src بروید مشاهده می کنید که چند پوشه تو در تو که هر پوشه نام یک قسمت از دامنه را دارد ایجاد شده است (یک پوشه با نام com و یکی با نام w3farsi و...) :



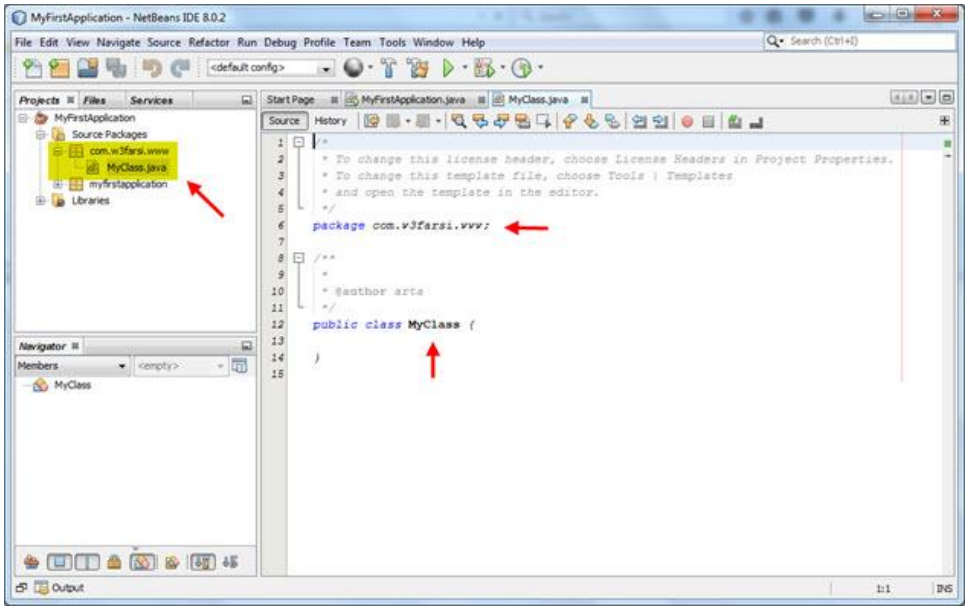
Package ایجاد شده ولی فاقد کلاس می باشد. برای ایجاد یک کلاس در داخل Package بر روی نام آن راست کلیک کرده و به صورت زیر یک کلاس ایجاد کنید:



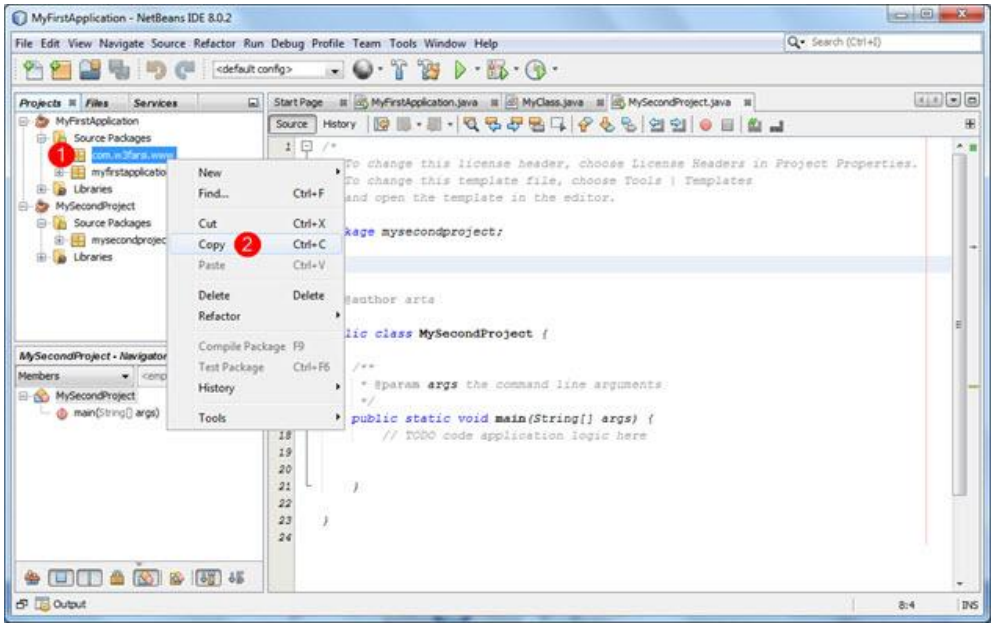
در مرحله بعد یک نام برای کلاس انتخاب کنید:



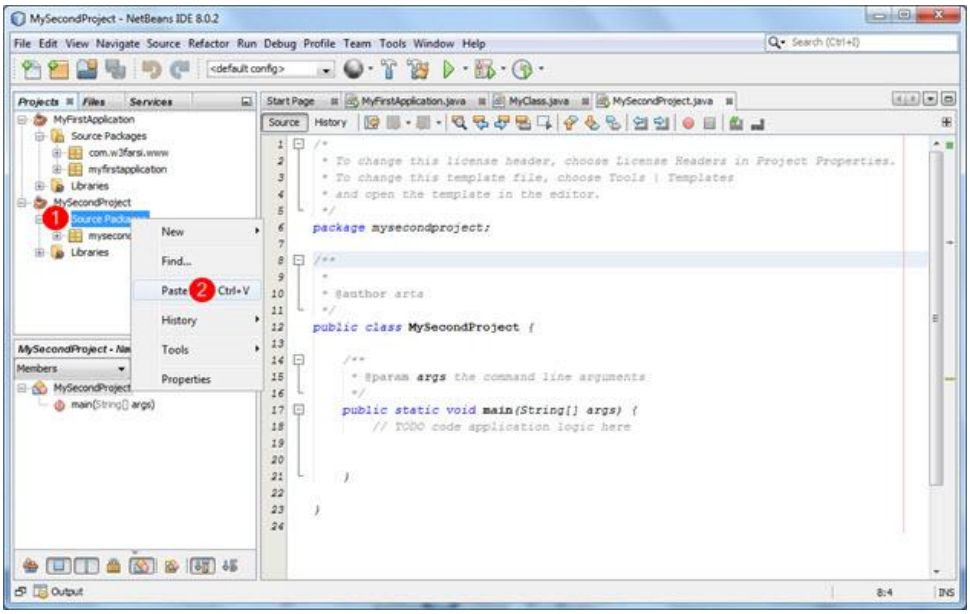
همانطور که مشاهده می کنید یک Package ایجاد شده که در داخل آن یک کلاس قرار دارد:



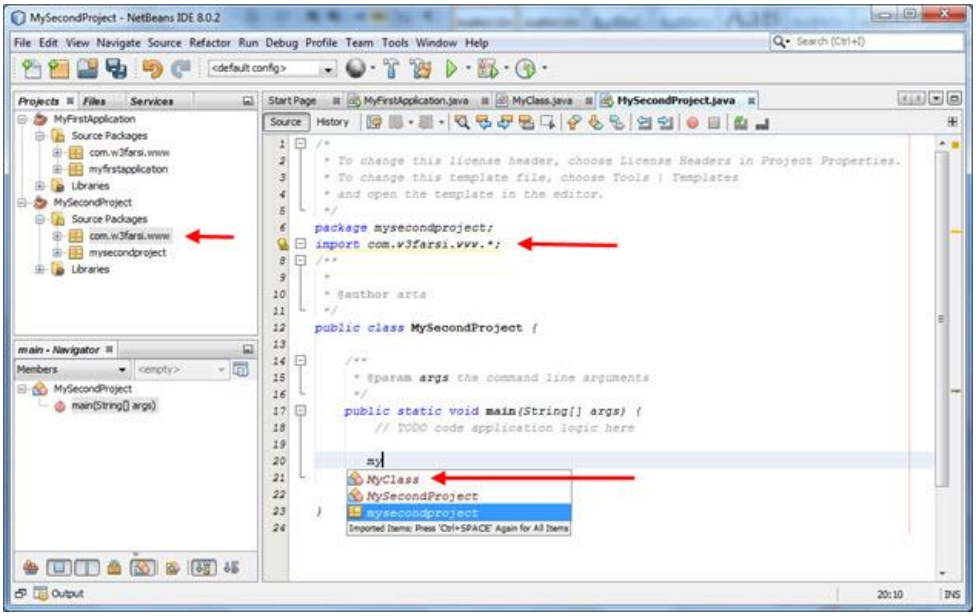
حال فرض کنید که یک برنامه دیگر ایجاد کرده ایم و می خواهیم از Packge و کلاس بالا در داخل آن استفاده کنیم . برای اینکار کافیست که بر رو ی نام Package راست کلیک کرده و بر روی گزینه copy کلیک کنیم:



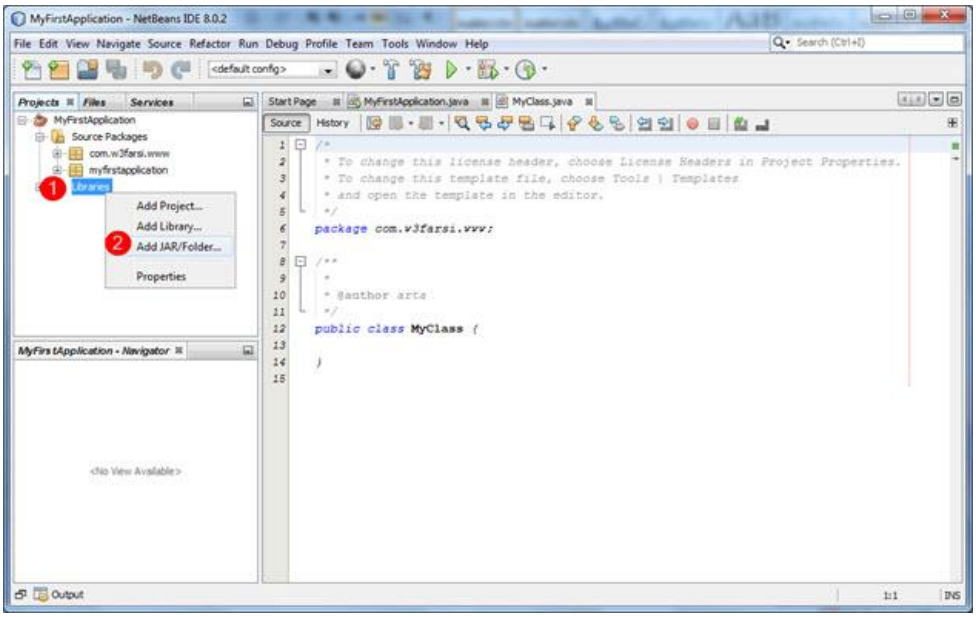
سپس در پروژه دوم بر روی Source Project راست کلیک کرده و گزینه Paste را بزنید:

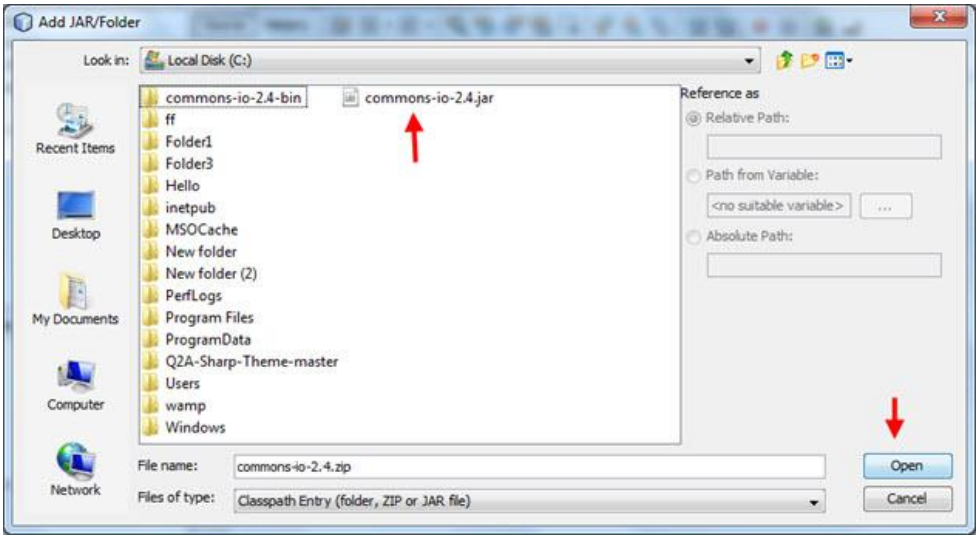


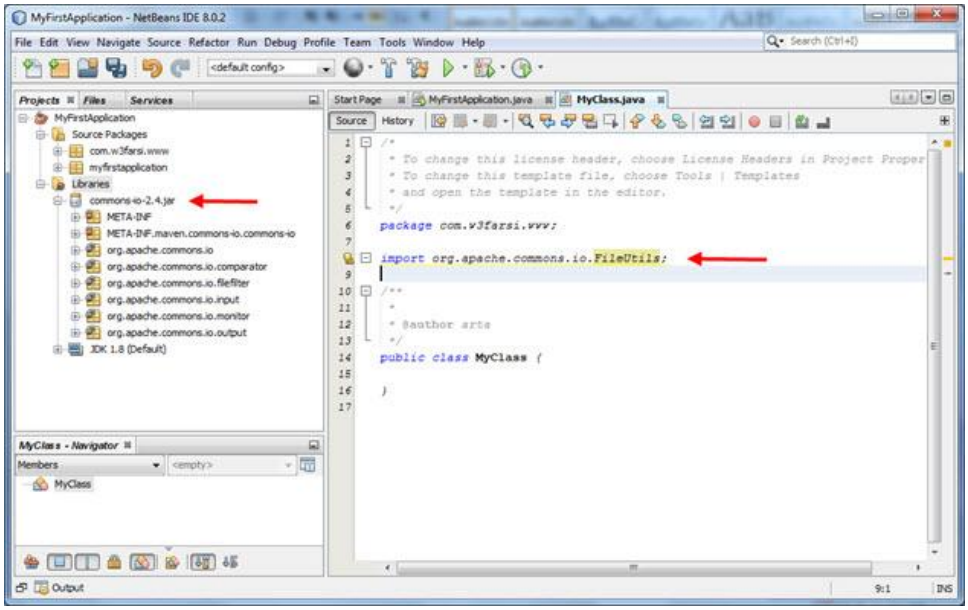
حال برای استفاده از کلاس ایجاد شده کافیست نام Package را import کنید تا کلاس قابل دسترسی باشد:



چون جاوا توسط چند شرکت بزرگ توسعه می یابد ، طبیعی است که این شرکت ها دارای Package ها و کتابخانه های کلاس مختص به خود باشند. مثلا یک کتابخانه کلاس برای کار با فایل ها و پوشه ها وجود دارد که محصول شرکت Apache بوده و دارای کلاس هایی می باشد که کار آنها انتقال و تغیر نام پوشه ها و فایل ها می باشد. برای وارد کردن این کتابخانه در برنامه و استفاده از کلاس های آنها به صورت زیر عمل می شود:


MyFirstApplication - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services
MyFirstApplication
Source Packages
com.w3farsi.www
myfirstapplication
Libraries
Add Project...
Add Library...
Add JAR/Folder...
Properties
Start Page
MyFirstApplication.java
MyClass.java
Source History
package com.w3farsi.www;
public class MyClass {
}
Output


Add JAR/Folder
Look in: Local Disk (C:)
commons-io-2.4-bin
commons-io-2.4.jar
ff
Folder1
Folder3
Hello
inetpub
MSOCache
New folder
New folder (2)
PerfLogs
Program Files
ProgramData
Q2A-Sharp-Theme-master
Users
wamp
Windows
Recent Items
Desktop
My Documents
Computer
Network
Reference as
Relative Path:
Path from Variable:
<no suitable variable>
Absolute Path:
File name: commons-io-2.4.zip
Files of type: Classpath Entry (folder, ZIP or JAR file)
Open
Cancel

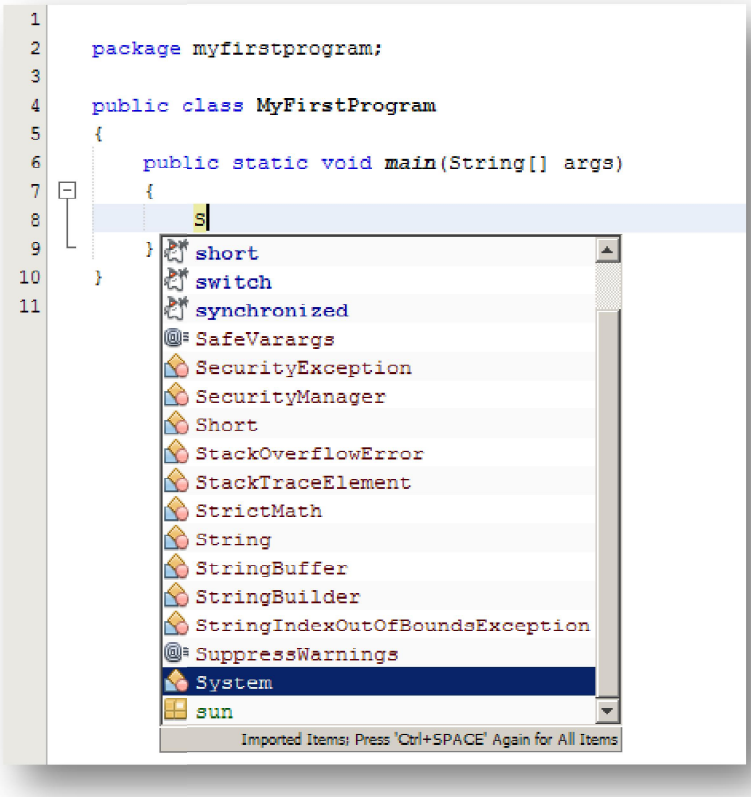


استفاده از IntelliSense در NetBeans

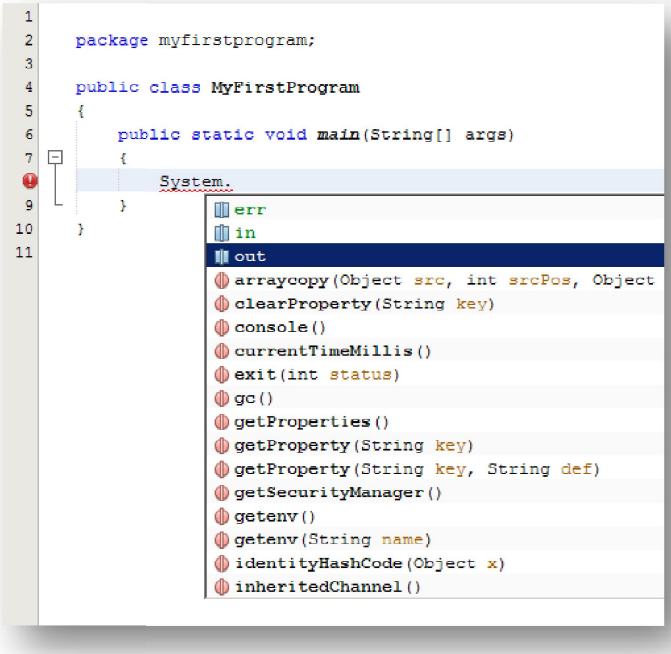
شاید یکی از ویژگیهای مهمNetBeans ، اینتل لایسنس باشد. IntelliSense ما را قادر می سازد که به سرعت به کلاسها و متدها و....دسترسی پیدا کنیم.وقتی که شما در محیط کدنویسی حرفی را تایپ کنید IntelliSense فورا فعال می شود. کد زیرا را در داخل متد main() بنویسید.

```
System.out.println("Welcome to JAVA Tutorials!");
```

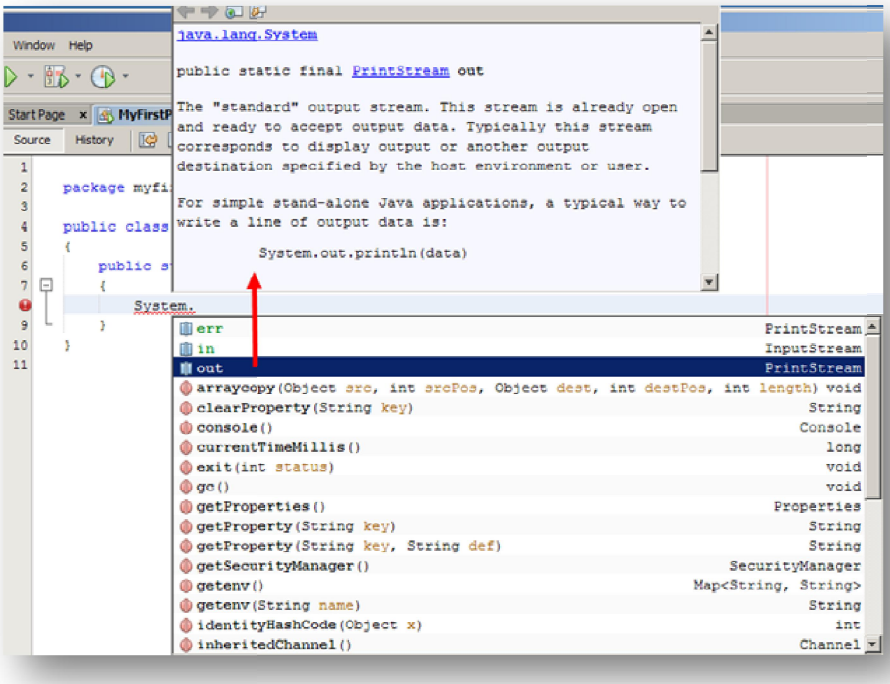
اولین حرف را تایپ کرده و سپس دکمه های ترکیبی Ctrl+Space را فشار دهید تا IntelliSense فعال شود:



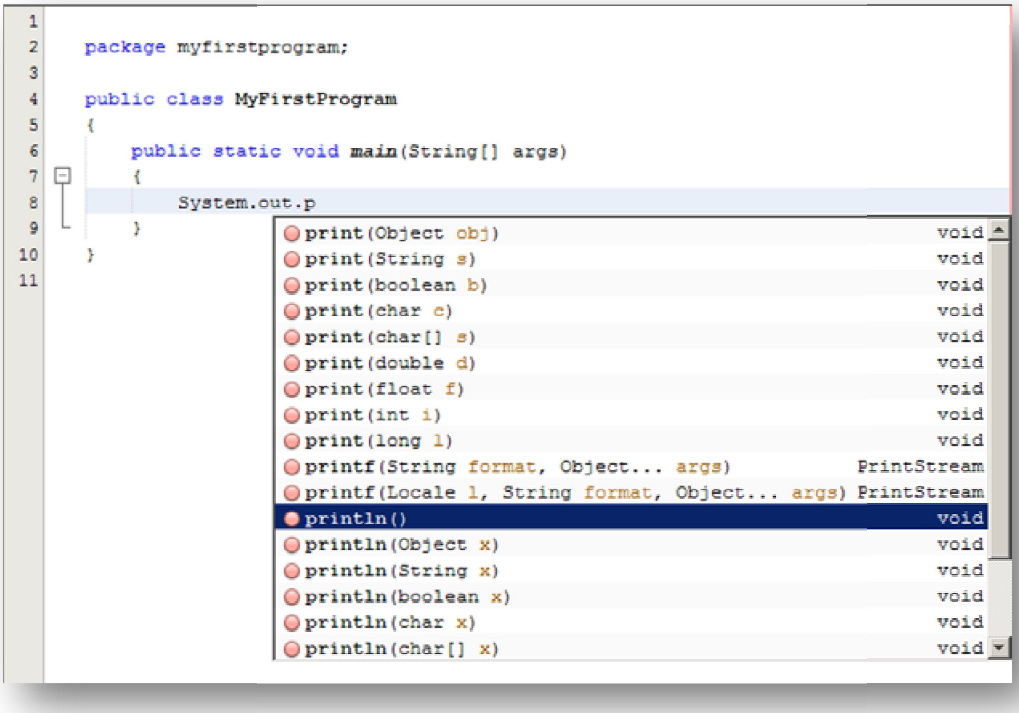
IntelliSense لیستی از کلمات به شما پیشنهاد می دهد که بیشترین تشابه را با نوشته شما دارند.شما می توانید با زدن دکمه tab گزینه مورد نظرتان را انتخاب کنید. با تایپ نقطه (.) شما با لیست پیشنهادی دیگری مواجه می شوید:



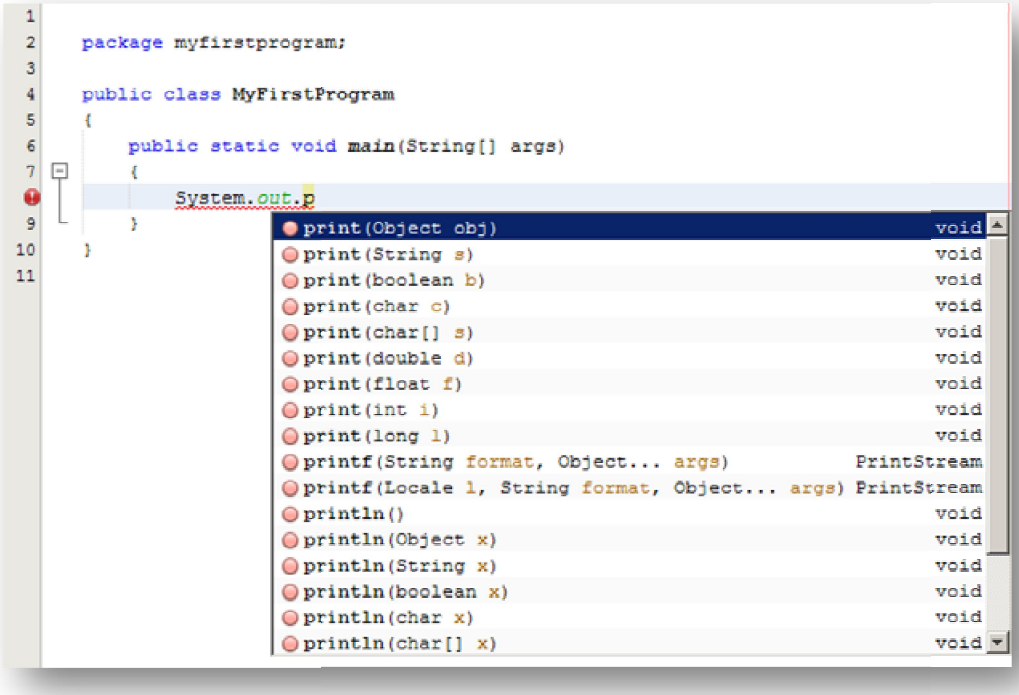
اگر بر روی گزینه ای که می خواهید انتخاب کنید لحظه ای مکث کنید توضیحی در رابطه با آن مشاهده خواهید کرد مانند شکل زیر:



هر چه که به پایان کد نزدیک می شوید لیست پیشنهادی محدود تر می شود.برای مثال با تایپ p ، اینتل لایسنس فقط کلماتی را که دارای حرف p هستند را نمایش می دهد:



با تایپ حرف های بیشتر لیست محدودتر می شود. اگر IntelliSense نتواند چیزی را که شما تایپ کرده اید پیدا کند هیچ چیزی را نمایش نمی دهد. برای ظاهر کردن IntelliSense کافیست دکمه ترکیبی Ctrl+Space را فشار دهید.برای انتخاب یکی از متدهایی که دارای چند حالت هستند، می توان با استفاده از دکمه های مکان نما (بالا و پایین) یکی از حالت ها را انتخاب کرد. مثلا متد println()همانطور که در شکل زیر مشاهده می کنید دارای چندین حالت نمایش پیغام در صفحه است:



IntelliSenseبه طور هوشمند کدهایی را به شما پیشنهاد می دهد و در نتیجه زمان نوشتن کد را کاهش می دهد.

رفع خطاها

بیشتر اوقات هنگام برنامه نویسی با خطا مواجه می شویم. تقریبا همه برنامه هایی که امروزه می بینید حداقل از داشتن یک خطا رنج می برند.خطا ها می توانند برنامه شما را با مشکل مواجه کنند .در جاوا سه نوع خطا وجود دارد:

خطای کامپایلری

این نوع خطا از اجرای برنامه شما جلوگیری می کند. این خطا ها شامل خطای دستور زبان می باشد.این بدین معنی است که شما قواعد کد نویسی را رعایت نکرده اید.یکی دیگر از موارد وقوع این خطا هنگامی است که شما از چیزی استفاده می کنید که نه وجود دارد و نه ساخته شده است. حذف فایلها یا اطلاعات ناقص در مورد پروژه ممکن است باعث به وجود آمدن خطای کامپایلری شود.استفاده از برنامه بوسیله برنامه دیگر نیز ممکن است باعث جلوگیری از اجرای برنامه و ایجاد خطای کامپایلری شود.

خطاهای منطقی

این نوع خطا در اثر تغییر در یک منطق موجود در برنامه به وجود می آید.رفع این نوع خطاها بسیار سخت است چون شما برای یافتن آنها باید کد را تست کنید.نمونه ای از یک خطای منطقی برنامه ای است که دو عدد را جمع می کند ولی حاصل تفریق دو عدد را نشان می دهد.در این حالت ممکن ات برنامه نویس علامت ریاضی را اشتباه تایپ کرده باشد.

استثناء

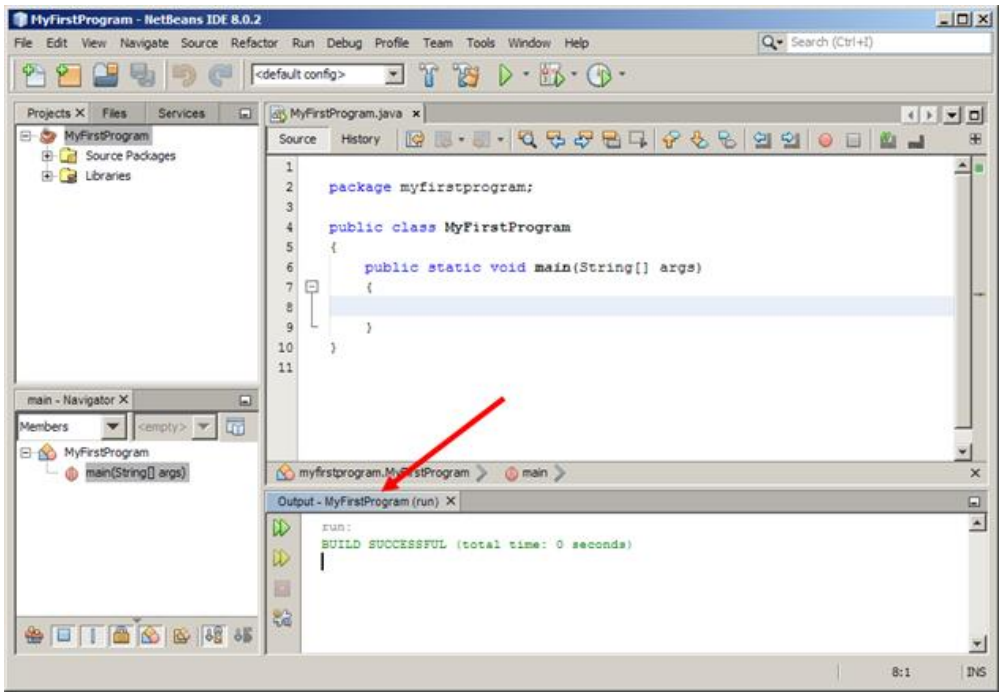
این نوع خطاها هنگامی رخ می دهند که برنامه در حال اجراست. این خطا هنگامی روی می دهد که کاربر یک ورودی نامعتبر به برنامه بدهد و برنامه نتواند آن را پردازش کند. NetBeans دارای ابزارهایی برای پیدا کردن و برطرف کردن خطاها هستند.وقتی در محیط کدنویسی در حال تایپ کد هستیم یکی از ویژگیهای NetBeans تشخیص خطاهای ممکن قبل از اجرای برنامه است.زیر کدهایی که دارای خطای کامپایلری هستند خط قرمز کشیده می شود.

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.oUt.println();
        System.out.println()
    }
}
```

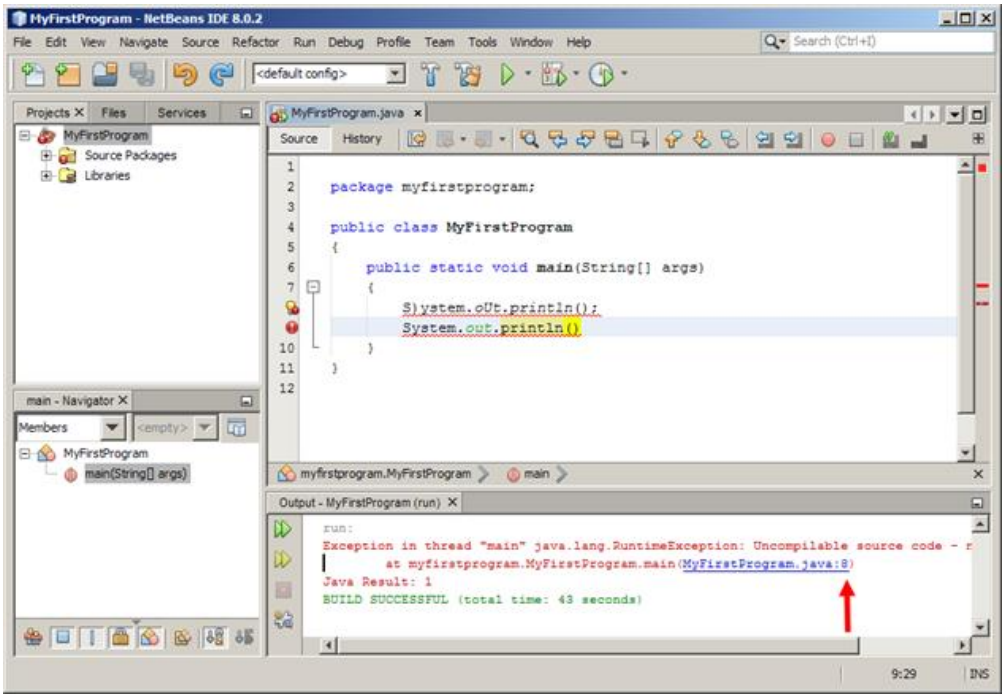
هنگامی که شما با موس روی این خطوط توقف کنید توضیحات خطا را مشاهده می کنید.شما ممکن است با خط سبز هم مواجه شوید که نشان دهنده اخطار در کد است ولی به شما اجازه اجرای برنامه را می دهند.به عنوان مثال ممکن است شما یک متغیر را تعریف کنید ولی در طول برنامه از آن استفاده نکنید. (در درس های آینده توضیح خواهیم داد).

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int number;
    }
}
```

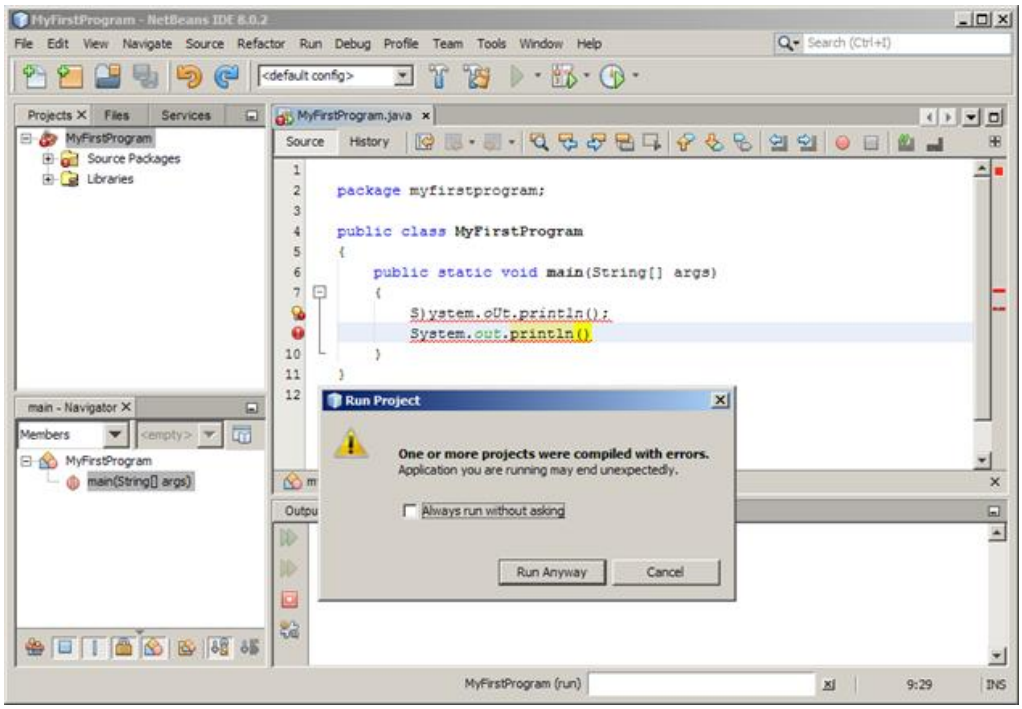
در باره رفع خطاها در آینده توضیح بیشتری می دهیم. پنجره Output که در شکل زیر با فلش قرمز نشان داده شده است به شما امکان مشاهده خطاها ، هشدارها و رفع آنها را می دهد.



همانطور که در شکل زیر مشاهده می کنید هرگاه برنامه شما با خطا مواجه شود لیست خطاها در پنجره Output نمایش داده می شود.



در شکل بالا علت به وجود آمدن خطا و شماره خطی که خطا در آن رخ داده است، نمایش داده شده است. اگر برنامه شما دارای خطا باشد و آن را اجرا کنید با پنجره زیر روبرو می شوید:



مربع کوچک داخل پنجره بالا را تیک زنید چون دفعات بعد که برنامه شما با خطا مواجه شود دیگر این پنجره به عنوان هشدار نشان داده نخواهد شد. با کلیک بر رو ی دکمه Run Anyway برنامه با وجود خطا نیز اجرا می شود.اما با کلیک بر روی دکمه Cancelاجرای برنامه متوقف می شود و شما باید خطاهای موجود در پنجره Output را بر طرف نمایید.

کاراکترهای کنترلی

کاراکترهای کنترلی کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می شوند و به دنبال آنها یک حرف یا عدد می آید و یک رشته را با فرمت خاص نمایش می دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می توان از کاراکتر کنترلی \n استفاده کرد:

```
System.out.println("Hello\nWorld!");
```

```
Hello
World
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \nنشانگر موس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می دهد. متد Println() هم مانند کاراکتر کنترلی \nیک خط جدید ایجاد می کند ، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می کند:

```
System.out.println("Hello World!");
```

کد بالا و کد زیر هیچ فرقی با هم ندارند:

```
System.out.print("Hello World!\n");
```

متد Print() کارکردی شبیه به Println() دارد با این تفاوت که نشان گر موس را در همان خط نگه می دارد و خط جدید ایجاد نمی کند. جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می دهد:

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	\f	چاپ کوتیشن	\’
خط جدید	\n	چاپ دابل کوتیشن	\”
سر سطر رفتن	\r	چاپ بک اسلش	\\
حرکت به صورت افقی	\t	حرکت به عقب	\b

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می کنیم.از آنجاییکه \معنای خاصی به رشته ها می دهد برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم:

```
System.out.println("We can print a \\ by using the \\\\ escape sequence.");
```
```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\،نشان دادن مسیر یک فایل در ویندوز است:

```
System.out.println("C:\\Program Files\\Some Directory\\SomeFile.txt");
```
```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (") برای نشان دادن رشته ها استفاده می کنیم برای چاپ آن از "\استفاده می کنیم:

```
System.out.println("I said, \"Motivate yourself!\".");
```
```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (') از '\استفاده می کنیم:

```
System.out.println("The programmer\'s heaven.");
```
```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از t\استفاده می شود:

```
System.out.println("Left\tRight");
```
```
Left    Right
```

هر تعداد کاراکتر که بعد از کاراکتر کنترلی r\بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می شوند:

```
System.out.println("Mitten\rK");
```
```
K
```

مثلا در مثال بالا کاراکتر K بعد از کاراکتر کنترلی r\ آمده است.کاراکتر کنترلی حرف Kرا به ابتدای سطر برده و جایگزین Mittenمی کند . برای مشاهده لیست مقادیر مبنای 16 برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید : http://www.ascii.cl/htmlcodes.htm

اگر کامپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می دهد.بیشترین خطا زمانی اتفاق می افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می کند.

متغیر

متغیر مکانی از حافظه است که شما می توانید مقادیری را در آن ذخیره کنید.می توان آن را به عنوان یک ظرف تصور کرد که داده های خود را در آن قرار داده اید.محتویات این ظرف می تواند پاک شود یا تغییر کند.هر متغیر دارای یک نام نیز هست.که از طریق آن می توان متغیر را از دیگر متغیر ها تشخیص داد و به مقدار آن دسترسی پیدا کرد.همچنین دارای یک مقدار می باشد که می تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد.مقدار متغیر می تواند تهی نیز باشد.متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده ای که در آن ذخیره می شود یکی است. متغیر دارای عمر نیز هست که از روی آن می توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد.و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است.ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می کنیم. هنگامی که یک برنامه ایجاد می کنیم احتیاج به یک مکان برای ذخیره داده ،مقادیر یا داده هایی که توسط کاربر وارد می شوند داریم.ایم مکان همان متغیر است. برای این از کلمه متغیر استفاده می شود چون ما می توانیم بسته به نوع شرایط هر جا که لازم باشد مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می گیرند که برنامه در حال اجراست و وقتی شما برنامه را می بندید محتویات متغیر ها نیز پاک می شود. قبلا ذکر شد که به وسیله نام متغیر می توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد:

1. نام متغیر باید با یک از حروف الفبا (a-z or A-Z) شروع شود.
2. نمی تواند شامل کاراکترهای غیرمجاز مانند .$,^ ,?, #باشد.
3. نمی توان از کلمات رزرو شده در جاوا برای نام متغیر استفاده کرد.
4. نام متغیر نباید دارای فضای خالی (spaces) باشد.
5. اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در جاوا دو حرف مانند a و Aدو کاراکتر مختلف به حساب می آیند.

دو متغیر با نامهای myNumberو MyNumberدو متغیر مختلف محسوب می شوند چون یکی از آنها با حرف کوچک mو دیگری با حرف بزرگ Mشروع می شود. شما نمی توانید دو متغیر را که دقیق شبیه هم هستند را در یک (scopeمحدوده) تعریف کنید Scope.به معنای یک بلوک کد است که متغیر در آن قابل دسترسی و استفاده است. در مورد Scopeدر فصلهای آینده بیشتر توضیح خواهیم داد. متغیر دارای نوع هست که نوع داده ای را که در خود ذخیره می کند را نشان می دهد. معمولترین انواع داده int, short, long, byte, double, float, char, Booleanمی باشند. برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع intاستفاده کنید.

انواع ساده

انواع ساده انواعی از داده ها هستند که شامل اعداد، کاراکترها و مقادیر بولی می باشند.به انواع ساده انواع اصلی نیز گفته می شود چون از آنها برای ساخت انواع پیچیده تری مانند کلاس ها و ساختارها استفاده می شود .انواع ساده دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می کنند.در جدول زیر انواع ساده و محدود آنها آمده است:

دامنه	نوع
اعداد صحیح بین 0 تا 255	byte
اعداد صحیح بین 32768- تا 32767	short
اعداد صحیح بین 2147483648- تا 2147483647	int
اعداد صحیح بین 9223372036854775808- تا 9223372036854775807	long

جدول زیر انواعی که مقادیر با ممیز اعشار را می توانند در خود ذخیره کنند را نشان می دهد:

دقت	دامنه تقریبی	نوع
7رقم	±1.5E-45 to ±3.4E38	float
16 – 15رقم	±5.0E-324 to ±1.7E308	double

برای به خاطر سپردن آنها باید از نماد علمی استفاده شود. نوع دیگری از انواع ساده برای ذخیره داده های غیر عددی به کار می روند و در جدول زیر نمایش داده شده اند:

مقادیر مجاز	نوع
کاراکترهای یونیکد	char
مقدار trueیا false	boolean

نوع char برای ذخیره کاراکترهای یونیکد استفاده می شود.کاراکترها باید داخل یک کوتیشن ساده قرار بگیرند مانند.('a')
نوع bool فقط می تواند مقادیر درست (true) یا نادرست (false) را در خود ذخیره کند و بیشتر در برنامه هایی که دارای ساختار تصمیم گیری هستند مورد استفاده قرار می گیرد.

استفاده از رشته ها

از رشته برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می شود.مقادیر ذخیره شده در یک رشته باید داخل دابل کوتیشن قرار گیرند تا توسط کامپایلر به عنوان یک رشته در نظر گرفته شوند، مانند .("massage"). جاوا دارای نوعی به نام رشته نیست، بلکه رشته ها اشیایی هستند که از روی کلاس (String حرف S به صورت بزرگ نوشته می شود) ساخته می شوند. با مفاهیم شیء و کلاس در درس های آینده آشنا می شوید. فقط در همین حد کافی است که بدانید که از رشته ها برای نمایش متن استفاده می شود مثلا برای نمایش متن Hello World می توان به صورت زیر عمل کرد :

```
String str ="Hello World";
```

دلیل اینکه در این قسمت درباره رشته ها مختصری توضیح دادیم این است که ممکن است در آموزش های بعدی با آنها سر و کار داشته باشیم. در آینده به طور مفصل در مورد رشته ها توضیح می دهیم.

استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهی متغیرها نمایش داده شده است:

```
 1: package myfirstprogram;
 2:
 3: import java.text.MessageFormat;
 4:
 6: public class MyFirstProgram
 7: {
 8:     public static void main(String[] args)
 9:     {
10:         //Declare variables
11:         int num1;
12:         int num2;
13:         double num3;
14:         double num4;
15:         boolean boolVal;
16:         char myChar;
17:
18:         //Assign values to variables
19:         num1 = 1;
20:         num2 = 2;
21:         num3 = 3.54;
22:         num4 = 4.12;
23:         boolVal = true;
24:         myChar = 'R';
25:
26:         //Show the values of the variables
27:         System.out.println(MessageFormat.format("num1 = {0}", num1));
29:         System.out.println(MessageFormat.format("num3 = {0}", num3));
30:         System.out.println(MessageFormat.format("num4 = {0}", num4));
31:         System.out.println(MessageFormat.format("boolVal = {0}", boolVal));
28:         System.out.println(MessageFormat.format("num2 = {0}", num2));
32:         System.out.println(MessageFormat.format("myChar = {0}", myChar));
33:     }
34: }
```

```
num1 = 1
num2 = 2
num3 = 3.54
num4 = 4.12
boolVal = true
myChar = R
```

تعریف متغیر

در خطوط 11-16 متغیرهایی با نوع و نام متفاوت تعریف شده اند. ابتدا باید نوع داده هایی را که این متغیر ها قرار است در خود ذخیره کنند را مشخص کنیم و سپس یک نام برای آنها در نظر بگیریم و در آخر سیمیکولن بگذاریم.همیشه به یاد داشته باشید که قبل از مقدار دهی و استفاده از متغیر باید آن را تعریف کرد.

```
int num1;
int num2;
double num3;
double num4;
bool boolVal;
char myChar;
```

نحوه تعریف متغیر به صورت زیر است:

```
data_type identifier;
```

date_type همان نوع داده است مانند int ، double و ... . Identifier نیز نام متغیر است که به ما امکان استفاده و دسترسی به مقدار متغیر را می دهد. برای تعریف چند متغیر از یک نوع می توان به صورت زیر عمل کرد:

```
data_type identifier1, identifier2, ... indentifierN;
```

مثال

```
int num1, num2, num3, num4, num5;
```

در مثال بالا 5 متغیر از نوع صحیح تعریف شده است.توجه داشته باشید که بین متغیر ها باید علامت کاما (,) باشد.

نامگذاری متغیرها

- نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود.
- نمی توان از کاراکترهای خاص مانند & ,% ,#یا عدد برای شروع نام متغیر استفاده کرد مانند .2numbers
- نام متغیر نباید دارای فاصله باشد.برای نام های چند حرفی میتوان به جای فاصله از علامت زیرخط یا _استفاده کرد.

نامهای مجاز:

```
num1  myNumber  studentCount  total       first_name    _minimum
num2  myChar    average      amountDue   last_name     _maximum
name  counter   sum          isLeapYear  color_of_car  _age
```

نامهای غیر مجاز:

```
123        #numbers#  #ofstudents  1abc2
```

```
123abc      $money      first name   ty.np
my number   this&that   last name    1:00
```

اگر به نامهای مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آنها خواهید شد.یکی از روشهای نامگذاری،نامگذاری کوهان شتری است.در این روش که برای متغیرهای دو کلمه ای به کار می رود،اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می شود مانند myNumber : توجه کنید که اولین حرف کلمه Numberبا حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudentsاست.اگز توجه کنید بعد از اولین کلمه حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

محدوده متغیر

متغیرها در داخل متد main()تعریف می شوند.این متغیرها فقط در داخل متد main()قابل دسترسی هستند. محدوده یک متغیر مشخص می کند که متغیر در کجای کد قابل دسترسی است. هنگامیکه برنامه به پایان متد main()می رسد متغیرها از محدوده خارج و بدون استفاده می شوند تا زمانی که برنامه در حال اجراست.محدوده متغیرها انواعی دارد که در درسهای بعدی با آنها آشنا می شوید. تشخیص محدوده متغیر بسیار مهم است چون به وسیله آن می فهمید که در کجای کد می توان از متغیر استفاده کرد. باید یاد آور شد که دو متغیر در یک محدوده نمی توانند دارای نام یکسان باشند. مثلا کد زیر در برنامه ایجاد خطا می کند:

```
int num1;
int num1;
```

از آنجاییکه جاوا به بزرگی و کوچک بودن حروف حساس است می توان از این خاصیت برای تعریف چند متغیر هم نام ولی با حروف متفاوت(از لحاظ بزرگی و کوچکی) برای تعریف چند متغیر از یک نوع استفاده کرد مانند:

```
int num1;
int Num1;
int NUM1;
```

مقداردهی متغیرها

می توان فورا بعد از تعریف متغیرها مقادیری را به آنها اختصاص داد.این عمل را مقداردهی می نامند. در زیر نحوه مقدار دهی متغیر ها نشان داده شده است:

```
data_type identifier = value;
```

به عنوان مثال:

```
int myNumber = 7;
```

همچنین می توان چندین متغیر را فقط با گذاشتن کاما بین آنها به سادگی مقدار دهی کرد:

```
data_type variable1 = value1, varaible2 = value2, ... variableN, valueN;

int num1 = 1, num2 = 2, num3 = 3;
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انخاب نوع و نام برای متغیر ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر.

اختصاص مقدار به متغیر

در زیر نحوه اختصاص مقادیر به متغیرها نشان داده شده است:

```
num1 = 1;
num2 = 2;
num3 = 3.54;
num4 = 4.12;
boolVal = true;
myChar = 'R';
```

به این نکته توجه کنید که شما به متغیری که هنوز تعریف نشده نمی توانید مقدار بدهید. شما فقط می توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند. مثلا متغیرهای بالا همه قابل استفاده هستند. در این مثال num1 و num2 هر دو تعریف شده اند و مقادیری از نوع صحیح به آنها اختصاص داده شده است. اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می دهد.

جانگهدار(Placeholders)

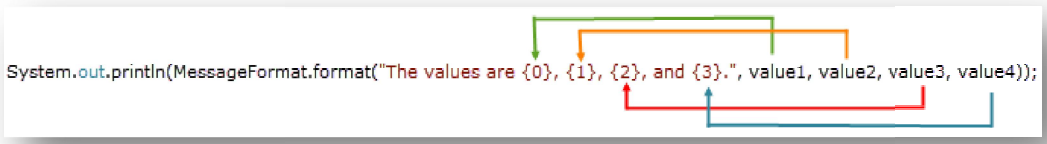
به متد format() از کلاس MessageFormat در خطوط (27-32) توجه کنید.برای استفاده از متد format() و کلاس MessageFormat ابتدا باید Package مربوط به آنها را در برنامه وارد کنید (خط 3:)

```
import java.text.MessageFormat;
```

این متد دو آرگومان قبول می کند. آرگومانها اطلاعاتی هستند که متد با استفاده از آنها کاری انجام می دهد. آرگومانها به وسیله کاما از هم جدا می شوند. آرگومان اول یک رشته قالب بندی شده است و آرگومان دوم مقداری است که توسط رشته قالب بندی شده مورد استفاده قرار می گیرد. اگر به دقت نگاه کنید رشته قالب بندی شده دارای عدد صفری است که در داخل دو آکولاد محصور شده است. البته عدد داخل دو آکولاد می تواند از صفر تا nباشد. به این اعداد جانگهدار می گویند. این اعداد بوسیله مقدار آرگومان بعد جایگزین می شوند. به عنوان مثال جانگهدار {0} به این معناست که اولین آرگومان (مقدار) بعد از رشته قالب بندی شده در آن قرار می گیرد. متد format() عملا می تواند هر تعداد آرگومان قبول کند اولین آرگومان همان رشته قالب بندی شده است که جانگهدار در آن قرار دارد و دومئین آرگومان مقداری است که جایگزین جانگهدار می شود. در مثال زیر از 4 جانگهدار استفاده شده است:

```
System.out.println(MessageFormat.format("The values are {0}, {1}, {2}, and {3}.", value1,
```

```
value2, value3, value4);
```



جانگهدارها از صفر شروع می شوند. تعداد جانگهدارها باید با تعداد آرگومانهای بعد از رشته قالب بندی شده برابر باشد. برای مثال اگر شما چهار جانگهدار مثل بالا داشته باشید باید چهار مقدار هم برای آنها بعد از رشته قالب بندی شده در نظر بگیرید. اولین جا نگهدار با دومین آرگومان و دومین جا نگهدار با سومین آرگومان جایگزین می شود. در ابتدا فهمیدن این مفهوم برای کسانی که تازه برنامه نویسی را شروع کرده اند سخت است اما در درسهای آینده مثالهای زیادی در این مورد مشاهده خواهید کرد.

ثابت

ثابت ها انواعی از متغیرها هستند که مقدار آنها در طول برنامه تغییر نمی کند. ثابت ها حتما باید مقدار دهی اولیه شوند و اگر مقدار دهی آنها فراموش شود در برنامه خطا به وجود می آید. بعد از این که به ثابت ها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمی توان آن را تغییر داد. برای تعریف ثابت ها باید از کلمه کلیدی **final** استفاده کرد.معمولا نام ثابت ها را طبق قرارداد با حروف بزرگ می نویسند تا تشخیص آنها در برنامه راحت باشد. نحوه تعریف ثابت در زیر آمده است:

```
final data_type identifier = initial_value;
```

مثال:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        final int NUMBER = 1;

        NUMBER = 10; //ERROR, Cant modify a constant
    }
}
```

در این مثال می بینید که مقدار دادن به یک ثابت ،که قبلا مقدار دهی شده برنامه را با خطا مواجه می کند.نکته ی دیگری که نباید فراموش شود این است که نباید مقدار ثابت را با مقدار دیگر متغیرهای تعریف شده در برنامه برابر قرار داد. مثال:

```
int someVariable;
final int MY_CONST = someVariable;
```

ممکن است این سوال برایتان پیش آمده باشد که دلیل استفاده از ثابت ها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی کنند بهتر است که آنها را به صورت ثابت تعریف کنید.این کار هر چند کوچک کیفیت برنامه شما را بالا می برد.

تبدیل ضمنی

تبدیل ضمنی یا تبدیل بزرگ کننده یا widening conversion یک نوع تبدیل است که به طور خودکار انجام می شود. در این نوع تبدیل در صورتی یک متغیر از یک نوع داده می تواند به یک نوع دیگر تبدیل شود که مقدار آن از مقدار داده ای که می خواهد به آن تبدیل شود کمتر باشد. به عنوان مثال نوع داده ای byte می تواند مقادیر 0 تا 255 را در خود ذخیره کند و نوع داده ای int مقادیر -2147483648 تا 2147483647 را شامل می شود. پس می توانید یک متغیر از نوع byte را به یک نوع int تبدیل کنید :

```
byte number1 = 5;

int number2 = number1;
```

در مثال بالا مقدار number1 برابر 5 است در نتیجه متغیر number2 که یک متغیر از نوع صحیح است می تواند مقدار number1را در خود ذخیره کند چون نوع صحیح از نوع بایت بزرگتر است. پس متغیر number1 که یک متغیر از نوع بایت است می تواند به طور ضمنی به number2 که یک متغیر از نوع صحیح است تبدیل شود. اما عکس مثال بالا صادق نیست.

```
int  number1 = 5;

byte number2 = number1;
```

در این مورد ما با خطا مواجه می شویم. اگر چه مقدار 5 متغیر number1 در محدوده مقادیر byte یعنی اعداد بین 0-255 قرار دارد اما متغیر ی از نوع بایت حافظه کمتری نسبت به متغیری از نوع صحیح اشغال می کند. نوع byte شامل 8 بیت یا 8 رقم دودویی است در حالی که نوع int شامل 32 بیت یا رقم باینری است.یک عدد باینری عددی متشکل از 0 و 1 است.برای مثال عدد 5 در کامپیوتر به عدد باینری 101 ترجمه می شود. بنابراین وقتی ما عدد 5 را در یک متغیر از نوع بایت ذخیره می کنیم عددی به صورت زیر نمایش داده می شود:

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می کنیم به صورت زیر نمایش داده می شود:

```
00000000000000000000000000000101
```

بنابراین قرار دادن یک مقدار int در یک متغیر byte درست مانند این است که ما سعی کنیم که یک توپ فوتبال را در یک سوراخ کوچک گلف جای دهیم. برای قرار دادن یک مقدار int در یک متغیر از نوع byte می توان از تبدیل صریح استفاه کرد که در درسهای آینده توضیح داده می شود. نکته دیگری که نباید فراموش شود این است که شما نمی توانید اعداد با ممیز اعشار را به یک نوع int تبدیل کنید چون این کار باعث از بین رفتن بخش اعشاری این اعداد می شود.

```
double number1 = 5.25;

int number2 = number1; //Error
```

تبدیلاتی که جاوا به صورت ضمنی می تواند انجام دهد در زیر آمده است:

تبدیل صریح

تبدیل صریح یا تبدیل کوچک کننده یا Narrowing Casting نوعی تبدیل است که برنامه را مجبور می کند که یک نوع داده را به نوعی دیگر تبدیل کند اگر این نوع تبدیل از طریق تبدیل ضمنی انجام نشود.در هنگام استفاده از این تبدیل باید دقت کرد چون در این نوع تبدیل ممکن است مقادیر اصلاح یا حذف شوند.ما می توانیم این عملیات را با استفاده از Cast انجام دهیم . فقط نام دیگر تبدیل صریح است و دستور آن به صورت زیر است:

```
datatypeA variableA = value;

datatypeB variableB = (datatypeB)variableA;
```

همانطور که قبلا مشاهده کردید نوع int را نتوانستیم به نوع byte تبدیل کنیم اما اکنون با استفاده از عمل Cast این تبدیل انجام خواهد شد:

```
int number1 = 5;

byte number2 = (byte)number1;
```

حال اگر برنامه را اجرا کنید با خطا مواجه نخواهید شد. همانطور که پیشتر اشاره شد ممکن است در هنگام تبدیلات مقادیر اصلی تغییر کنند.برای مثال وقتی که یک عدد با ممیز اعشار مثلا از نوع double را به یک نوع int تبدیل می کنیم مقدار اعداد بعد از ممیز از بین می روند:

```
double number1 = 5.25;

int number2 = (int)number1;

System.out.println(number2);
```

```
5
```

خروجی کد بالا عدد 5 است چون نوع داده ای int نمی تواند مقدار اعشار بگیرد. حالت دیگر را تصور کنید.اگر شما بخواهید یک متغیر را که دارای مقداری بیشتر از محدوده متغیر مقصد هست تبدیل کنید چه اتفاقی می افتد؟ مانند تبدیل زیر که می خواهیم متغیر number1 را که دارای مقدار 300 است را به نوع بایت تبدیل کنیم که محدود اعداد بین 255-0 را پوشش می دهد.

```
int number1 = 300;

byte number2 = (byte)number1;

System.out.println(MessageFormat.format("Value of number2 is {0}.", number2));
```

```
Value of number2 is 44.
```

خروجی کد بالا عدد 44 است Byte .فقط می تواند شامل اعداد 0 تا 255 باشد و نمی تواند مقدار 300 را در خود ذخیره کند.حال می خواهیم ببینیم که چرا به جای عدد 300 ما عدد 44 را در خروجی می گیریم. این کار به تعداد بیتها بستگی دارد.یک byte دارای 8 بیت است درحالی که int دارای 32 بیت است. حال اگر به مقدار باینری 2 عدد توجه کنید متوجه می شوید که چرا خروجی عدد 44 است.

```
300 = 00000000000000000000000100101100
255 =                         11111111
 44 =                         00101100
```

خروجی بالا نشان می دهد که بیشترین مقدار byte که عدد 255 است می تواند فقط شامل 8 بیت باشد (11111111) بنابراین فقط 8 بیت اول مقدار int به متغیر byteانتقال می یابد که شامل (00101100) یا عدد 44 در مبنای 10 است.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید:

- عملگر : نمادهایی هستند که اعمال خاص انجام می دهند.
- عملوند : مقادیری که عملگرها بر روی آنها عملی انجام می دهند.

مثلا X+Y: یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می آیند.

زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می آیند. جاوا دارای عملگرهای مختلفی از جمله عملگرهای ریاضی ،تخصیصی ، مقایسه ای ، منطقی و بیتی می باشد. از عملگرهای ساده ریاضی می توان به عملگر جمع و تفریق اشاره کرد. سه نوع عملگر در جاوا وجود دارد:

- یگانی – (Unary) به یک عملوند نیاز دارد
- دودویی – (Binary) به دو عملوند نیاز دارد
- سه تایی – (Ternary) به سه عملوند نیاز دارد

انواع مختلف عملگر که در ای بخش مورد بحث قرار می گیرند عبارتند از:

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه ای
- عملگرهای منطقی
- عملگرهای بیتی

عملگرهای ریاضی

جاوا از عملگرهای ریاضی برای انجام محاسبات استفاده می کند. جدول زیر عملگرهای ریاضی جاوا را نشان می دهد:

نتیجه	مثال	دسته	عملگر
Var1برابر است با حاصل جمع var2و var3	var1 = var2 + var3;	Binary	+
Var1برابر است با حاصل تفریق var2و var3	var1 = var2 – var3;	Binary	-
Var1برابر است با حاصلضرب var2در var3	var1 = var2 * var3;	Binary	*
Var1برابر است با حاصل تقسیم var2بر var3	var1 = var2 / var3;	Binary	/
Var1برابر است با باقیمانده تقسیم var2و var3	var1 = var2 % var3;	Binary	%
Var1برابر است با مقدار var2	var1 = +var2;	Unary	+
Var1برابر است با مقدار var2ضربدر 1 -	var1 = -var2;	Unary	-

دیگر عملگرهای جاوا عملگرهای کاهش و افزایش هستند.این عملگرها مقدار 1 را از متغیر ها کم یا به آنها اضافه می کنند. از این متغیر ها اغلب در حلقه ها استفاده می شود:

نتیجه	مثال	دسته	عملگر
مقدار var1 برابر است با var2 بعلاوه 1	var1 = ++var2;	Unary	++
مقدار var1 برابر است با var2 منهای 1	var1 = –var2;	Unary	–
مقدار var1 برابر است باvar2 به متغیر var2 یک واحد اضافه می شود	var1 = var2++;	Unary	++
مقدار var1 برابر است باvar2 از متغیر var2 یک واحد کم می شود	var1 = var2–;	Unary	–

به این نکته توجه داشته باشید که محل قرار گیری عملگر در نتیجه محاسبات تاثیر دارد.اگر عملگر قبل از متغیر var2 بیاید افزایش یا کاهش var1 اتفاق می افتد.چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2می شود و سپس متغیر var2 افزایش یا کاهش می یابد. به مثال های زیر توجه کنید:

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;

        x = ++y;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
    }
}
```

```
x=2
y=2
```

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;

        x = --y;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
```

```
    }
}
x=0
y=0
```

همانطور که در دو مثال بالا مشاهده می کنید،درج عملگرهای — و ++ قبل از عملوند y باعث می شود که ابتدا یک واحد از y کم و یا یک واحد به y اضافه شود و سپس نتیجه در عملوند x قرار بگیرد. حال به دو مثال زیر توجه کنید:

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;

        x = y--;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
    }
}
x=1
y=0
```

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;

        x = y++;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
    }
}
x=1
y=2
```

همانطور که در دو مثال بالا مشاهده می کنید،در عملگرهای — و ++ بعد از عملوند y باعث می شود که ابتدا مقدار y در داخل متغیر x قرار بگیرد و سپس یک واحد از y کم و یا یک واحد به ان اضافه شود. حال می توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در جاوا را یاد بگیریم:

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
       //Variable declarations
       int num1, num2;

       //Assign test values
       num1 = 5;
       num2 = 3;

       System.out.println(MessageFormat.format("The sum of {0} and {1} is {2}.", num1,
num2, (num1 + num2)));
       System.out.println(MessageFormat.format("The difference of {0} and {1} is {2}.",
num1, num2, (num1 - num2)));
       System.out.println(MessageFormat.format("The product of {0} and {1} is {2}.",
num1, num2, (num1 * num2)));
       System.out.println(MessageFormat.format("The quotient of {0} and {1} is {2}.",
num1, num2, ((double)num1 / num2)));
       System.out.println(MessageFormat.format("The remainder of {0} and {1} is {2}.",
num1, num2, (num1 % num2)));
    }
}
```

```
The sum of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product of 5 and 3 is 15.
The quotient of 5 and 3 is 1.67.
The remainder of 5 divided by 3 is 2
```

برنامه بالا نتیه هر عبارت را نشان می دهد. در این برنامه از متد println() برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. در این مثال با یک نکته عجیب مواجه می شویم و آن حاصل تقسیم دو عدد صحیح است.وقتی که دو عدد صحیح را بر هم تقسیم کنیم حاصل باید یک عدد صحیح و فاقد بخش کسری باشد.اما همانطور که مشاهده می کنید اگر فقط یکی از اعداد را به نوع اعشاری double تبدیل کنیم (در مثال می بینید) حاصل به صورت اعشار نشان داده می شود.

عملگرهای تخصیصی

نوع دیگر از عملگرهای جاوا عملگرهای جایگزینی نام دارند.این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می دهند. جدول زیر انواع عملگرهای تخصیصی در جاوا را نشان می دهد:

عملگر	مثال	نتیجه
=	var1 = var2;	مقدار var1 برابر است با مقدار var2
+=	var1 += var2;	مقدار var1 برابر است با حاصل جمع var1 و var2
-=	var1 -= var2;	مقدار var1 برابر است با حاصل تفریق var1 و var2
*=	var1 *= var2;	مقدار var1 برابر است با حاصل ضرب var1 در var2
/=	var1 /= var2;	مقدار var1 برابر است با حاصل تقسیم var1 بر var2
%=	var1 %= var2;	مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2

از عملگر += برای اتصال دو رشته نیز می توان استفاده کرد.استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلا شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می باشد. این حالت کدنویسی زمانی کارایی خود را نشان می دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تاثیر آنها را بر متغیر ها نشان می دهد.

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
       int number;

       System.out.println("Assigning 10 to number...");
       number = 10;
       System.out.println(MessageFormat.format("Number = {0}", number));

       System.out.println("Adding 10 to number...");
       number += 10;
       System.out.println(MessageFormat.format("Number = {0}", number));

       System.out.println("Subtracting 10 to number...");
       number -= 10;
       System.out.println(MessageFormat.format("Number = {0}", number));
    }
}
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از 3 عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار 10با استفاده از عملگر = به آن اختصاص داده شده است.سپس به آن با استفاده از عملگر =+ مقدار 10 اضافه شده است.و در آخر به وسیله عملگر =- عدد 10 از آن کم شده است.

عملگرهای مقایسه ای

از عملگرهای مقایسه ای برای مقایسه مقادیر استفاده می شود.نتیجه این مقادیر یک مقدار بولی(منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار true و اگر نتیجه مقایسه اشتباه باشد مقدار false را نشان می دهند. این عملگرها به طور معمول در دستورات شرطی به کار می روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می شوند. جدول زیر عملگرهای مقایسه ای در جاوا را نشان می دهد:

عملگر	دسته	مثال	نتیجه
==	Binary	var1 = var2 == var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت false است
!=	Binary	var1 = var2 != var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت false است
<	Binary	var1 = var2 < var3	var1 در صورتی true است که مقدار var2 کوچکتر از var3 مقدار باشد در غیر اینصورت false است
>	Binary	var1 = var2 > var3	var1 در صورتی true است که مقدارvar2 بزرگتر ازمقدار var3 باشد در غیر اینصورت false است
<=	Binary	var1 = var2 <= var3	var1 در صورتی true است که مقدار var2 کوچکتر یا مساوی مقدار var3 باشد در غیر اینصورت false است
>=	Binary	var1 = var2 >= var3	var1 در صورتی true است که مقدار var2 بزرگتر یا مساوی var3 مقدار باشد در غیر اینصورت false است

برنامه زیر نحوه عملکرد ای عملگرها را نشان می دهد:

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
       int num1 = 10;
       int num2 = 5;

       System.out.println(MessageFormat.format("{0} == {1} : {2}", num1, num2, num1 ==
num2));
       System.out.println(MessageFormat.format("{0} != {1} : {2}", num1, num2, num1 !=
num2));
```

```
        System.out.println(MessageFormat.format("{0}  < {1} : {2}", num1, num2, num1 <
num2));
        System.out.println(MessageFormat.format("{0}  > {1} : {2}", num1, num2, num1 >
num2));
        System.out.println(MessageFormat.format("{0} <= {1} : {2}", num1, num2, num1 <=
num2));
        System.out.println(MessageFormat.format("{0} >= {1} : {2}", num1, num2, num1 >=
num2));
    }
}
```

```
10 == 5 : False
10 != 5 : True
10 <  5 : False
10 >  5 : True
10 <= 5 : False
10 >= 5 : True
```

در مثال بالا ابتدا دو متغیر را که می خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می دهیم.سپس با استفاده از یک عملگر مقایسه ای آنها را با هم مقایسه کرده و نتیجه را چاپ می کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند x = y مقدار y را در به xاختصاص می دهد. عملگر == عملگر مقایسه ای است که دو مقدار را با هم مقایسه می کند مانند x==y و اینطور خوانده می شود x برابر است با. y

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرطهای پیچیده استفاده می شود. همانطور که قبلا یاد گرفتید مقادیر بولی می توانند false یا true باشند. فرض کنید که var2 و var3 دو مقدار بولی هستند.

مثال	دسته	نام	عملگر
var1 = var2 && var3;	Binary	منطقی AND	&&
var1 = var2 || var3;	Binary	منطقی OR	||
var1 = !var1;	Unary	منطقی NOT	!

عملگر منطقی AND(&&)

اگر مقادیر دو طرف عملگر AND ، trueباشند عملگر AND مقدار true را بر می گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها false باشند مقدار false را بر می گرداند. در زیر جدول درستی عملگر AND نشان داده شده است:

X	Y	X && Y
true	**true**	**true**
true	**false**	**false**
false	**true**	**false**
false	**false**	**false**

برای درک بهتر تاثیر عملگر AND یاد آوری می کنم که این عملگر فقط در صورتی مقدار true را نشان می دهد که هر دو عملوند مقدارشان true باشد.در غیر اینصورت نتیجه تمام ترکیبهای بعدی false خواهد شد. استفاده از عملگر ANDمانند استفاده از عملگرهای مقایسه ای است. به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگتر از 18 و salary کوچکتر از 1000 باشد.

```
result = (age > 18) && (salary < 1000);
```

عملگر AND زمانی کارامد است که ما با محدود خاصی از اعداد سرو کار داریم. مثلا عبارت 10 <= x <= 100بدین معنی است که x می تواند مقداری شامل اعداد 10 تا 100 را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می توان از عملگر منطقی ANDبه صورت زیر استفاده کرد.

```
inRange = (number <= 10) && (number >= 100);
```

عملگر منطقی OR(||)

اگر یکی یا هر دو مقدار دو طرف عملگر OR ،درست (true) باشد،عملگر OR مقدار true را بر می گرداند. جدول درستی عملگر ORدر زیر نشان داده شده است :

X	Y	X || Y
true	**true**	**true**
true	**false**	**true**
false	**true**	**true**
false	**false**	**false**

در جدول بالا مشاهده می کنید که عملگر OR در صورتی مقدار false را بر میگرداند که مقادیر دو طرف آن false باشند. کد زیر را در نظر بگیرید.نتیجه این کد در صورتی درست (true) است که رتبه نهایی دانش آموز (finalGrade) بزرگتر از 75 یا یا نمره نهایی امتحان آن 100 باشد.

```
isPassed = (finalGrade >= 75) || (finalExam == 100);
```

عملگر منطقی NOT(!)

برخلاف دو اپراتور OR و AND عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. این عملگر یک مقدار یا اصطلاح بولی را نفی می کند .مثلا اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می کند. جدول زیر عملکرد اپراتور NOT را نشان می دهد:

X	!X
true	**false**
false	**true**

نتیجه کد زیر در صورتی درست است که) age سن) بزرگتر یا مساوی 18 نباشد.

```
isMinor = !(age >= 18);
```

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می دهند که شکل باینری انواع داده ها را دستکاری کنید.برای درک بهتر این درس توصیه می شود که شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را یاد بگیرید.در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می کند وضعیت هر چیز یا خاموش است یا روشن.برای نشان دادن حالت روشن از عدد 1 و برای نشان دادن حالت خاموش از عدد 0 استفاده می شود.بنابراین اعداد باینری فقط می توانند صفر یا یک باشند.اعداد باینری را اعداد در مبنای 2 و اعداد اعشاری را اعداد در مبنای 10 می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده 8 بیت است. به عنوان مثال برای یک داده از نوع int به 32 بیت یا 8 بایت فضا برای ذخیره آن نیاز داریم،این بدین معناست که اعداد از 32 رقم 0 و 1 برای ذخیره استفاده می کنند. برای مثال عدد 100 وقتی به عنوان یک متغیر از نوع int ذخیره می شود در کامپیوتر به صورت زیر خوانده می شود:

```
00000000000000000000000001100100
```

عدد 100 در مبنای ده معادل عدد 1100100 در مبنای 2 است.در اینجا 7 رقم سمت راست نشان دهنده عدد 100 در مبنای 2 است و مابقی صفرهای سمت راست برای پر کردن بیتهایی است که عدد از نوع int نیاز دارد.به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می شوند. عملگرهای بیتی سی شارپ در جدول زیر نشان داده شده اند:

مثال	دسته	نام	عملگر
x = y & z;	Binary	بیتی AND	&
x = y | z;	Binary	بیتی OR	|
x = y ^ z;	Binary	بیتی XOR	^
x = ~y;	Unary	بیتی NOT	~
x &= y;	Binary	بیتی AND Assignment	&=
x |= y;	Binary	بیتی OR Assignment	|=
x ^= y;	Binary	بیتی XOR Assignment	^=

عملگر بیتی AND(&)

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام می دهد با این تفاوت که این عملگر بر روی بیتها کار می کند.اگر مقادیر دو طرف آن 1 باشد مقدار 1 را بر می گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می گرداند. جدول درستی عمگر بیتی AND در زیر آمده است:

X	Y	X AND Y
1	**1**	**1**
1	**0**	**0**
0	**1**	**0**
0	**0**	**0**

در زیر نحوه استفاده از عملگر بیتی AND آمده است:

```
int result = 5 & 3;

System.out.println(result);
```

```
1
```

همانطور که در مثال بالا مشاهده می کنید نتیجه عملکرد عملگر AND بر روی دو مقدار 5 و 3 عدد یک می شود.اجازه بدهید ببینیم که چطور این نتیجه را به دست می آید:

```
5:  00000000000000000000000000000101
3:  00000000000000000000000000000011
-----------------------------------
1:  00000000000000000000000000000001
```

ابتدا دو عدد 5 و 3 به معادل باینری شان تبدیل می شوند. از آنجاییکه هر عدد صحیح 32 (int)بیت است از صفر برای پر کردن بیتهای خالی استفاده می کنیم. با استفاده از جدول درستی عملگر بیتیAND می توان فهمید که چرا نتیجه عدد یک می شود.

عملگر بیتی OR(|)

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه صفر در غیر اینصورت 1 خواهد شد.جدول درستی این عملگر در زیر آمده است:

X	Y	X OR Y
1	**1**	**1**
1	**0**	**1**
0	**1**	**1**
0	**0**	**0**

نتیجه عملگر بیتی OR در صورتی صفر است که عملوند های دو طرف آن صفر باشند.اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید:

```
int result = 7 | 9;

System.out.println(result);
```

```
15
```

وقتی که از عملگر بیتی OR برای دو مقدار در مثال بالا (7 و 9) استفاده می کنیم نتیجه 15 می شود.حال بررسی می کنیم که چرا این نتیجه به دست آمده است؟

```
 7: 00000000000000000000000000000111
 9: 00000000000000000000000000001001
--------------------------------
15: 00000000000000000000000000001111
```

با استفاده از جدول درستی عملگر بیتی OR می توان نتیجه استفاده از این عملگر را تشخیص داد.عدد 1111 باینری معادل عدد 15 صحیح است.

عملگر بیتی XOR(^)

جدول درستی این عملگر در زیر آمده است:

X	Y	X XOR Y
1	**1**	**0**
1	**0**	**1**
0	**1**	**1**
0	**0**	**0**

در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک می شود.در مثال زیر تاثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده می کنید:

```
int result = 5 ^ 7;

System.out.println(result);
```

```
2
```

در زیر معادل باینری اعداد بالا (5 و 7) نشان داده شده است.

```
5: 00000000000000000000000000000101
7: 00000000000000000000000000000111
--------------------------------
2: 00000000000000000000000000000010
```

با نگاه کردن به جدول درستی عملگر بیتی XOR ، می توان فهمید که چرا نتیجه عدد 2 می شود.

عملگر بیتی NOT(~)

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد.در زیر جدول درستی این عملگر آمده است:

X	NOT X
1	**0**
0	**1**

عملگر بیتی NOT مقادیر بیتها را معکوس می کند.در زیر چگونگی استفاده از این عملگر آمده است:

```
int result = ~7;

System.out.println(result);
```

به نمایش باینری مثال بالا که در زیر نشان داده شده است توجه نمایید:

```
7: 00000000000000000000000000000111
-----------------------------------
-8: 11111111111111111111111111111000
```

عملگر بیتی تغییر مکان(shift)

این نوع عملگرها به شما اجازه می دهند که بیتها را به سمت چپ یا راست جا به جا کنید.دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می کنند.عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جاییبیت ها را نشان می دهد.

مثال	دسته	نام	عملگر
x = y << 2;	Binary	تغییر مکان به سمت چپ	<<
x = y >> 2;	Binary	تغییر مکان به سمت راست	>>

عملگر تغییر مکان به سمت چپ

این عملگر بیتهای عملوند سمت چپ را به تعداد n مکان مشخص شده توسط عملوند سمت راست، به سمت چپ منتقل می کند.
به عنوان مثال:

```
int result = 10 << 2;

System.out.println(result);
```

```
40
```

در مثال بالا ما بیتهای مقدار 10 را دو مکان به سمت چپ منتقل کرده ایم، حال بیایید تاثیر این انتقال را بررسی کنیم:

```
10: 00000000000000000000000000001010
--------------------------------
40: 00000000000000000000000000101000
```

مشاهده می کنید که همه بیت ها به اندازه دو واحد به سمت چپ منتقل شده اند.در این انتقال دو صفر از صفرهای سمت چپ کم می شود و در عوض دو صفر به سمت راست اضافه می شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عمگر تغییر مکان به سمت چپ است با این تفاوت که بیت ها را به سمت راست جا به جا می کند.
به عنوان مثال:

```
int result = 100 >> 4;

System.out.println(result);
```

```
6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت های مقدار 100 را به اندازه 4 واحد به سمت چپ جا به جا می کنیم. اجازه بدهید تاثیر این جا به جایی را مورد بررسی قرار دهیم:

```
100: 00000000000000000000000001100100
---------------------------------
  6: 00000000000000000000000000000110
```

هر بیت به اندازه 4 واحد به سمت راست منتقل می شود، بنابراین 4 بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می شود.

تقدم عملگرها

تقدم عملگرها مشخص می کند که در محاسباتی که بیش از دو عملوند دارند ابتدا کدام عملگر اثرش را اعمال کند.عملگر ها در جاوا در محاسبات دارای حق تقدم هستند.به عنوان مثال:

```
number = 1 + 2 * 3 / 1;
```

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه 9 خواهد شد (3=2+1 سپس 9=3×3 و در آخر 9=9/1). اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می دهد.برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد 2 ضربدر 3 و سپس نتیجه آنها تقسیم بر 1 می شود که نتیجه 6 به دست می آید.در آخر عدد 6 با 1 جمع می شود و عدد 7 حاصل می شود .در جدول زیر تقدم برخی از عملگرهای جاوا آمده است:

عملگر	تقدم
++, –, (used as prefixes); +, – (unary)	بالاترین
*, /, %	
+, -	
<<, >>	
<, >, <=, >=	
==, !=	
&	
^	
|	
&&	
||	
=, *=, /=, %=, +=, -=	
++, — (used as suffixes)	پایین ترین

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین ترین حق تقدم در محاسبات تاثیر می گذارند.به این نکته توجه کنید که تقدم عملگرها ++و – به مکان قرارگیری آنها بستگی دارد (در سمت چپ یا راست عملوند باشند). به عنوان مثال:

```
int number = 3;

number1 = 3 + ++number; //results to 7
number2 = 3 + number++; //results to 6
```

در عبارت اول ابتدا به مقدار number یک واحد اضافه شده و 4 می شود و سپس مقدار جدید با عدد 3 جمع می شود و در نهایت عدد 7 به دست می آید.در عبارت دوم مقدار عددی 3 به مقدار number اضافه می شود و عدد 6 به دست می آید.سپس این مقدار در متغیر number2 قرار می گیرد. و در نهایت مقدار number به 4 افزایش می یابد. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آنها از عملگرهای زیادی استفاده می شود از پرانتز استفاده می کنیم:

```
number = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ));
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می گیرند.به نکته ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید.در این عبارت ابتدا مقدار داخلی ترین پرانتز مورد محاسبه قرار می گیرد یعنی مقدار 6 ضربدر 7 شده و سپس از 5 کم می شود .اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می آیند مورد ارزیابی قرار دهید .به عنوان مثال:

```
number = 3 * 2 + 8 / 4;
```

هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابر این شما باید از چپ به راست آنها را در محاسبات تاثیر دهید.یعنی ابتدا 3 را ضربدر 2 می کنید و سپس عدد 8 را بر 4 تقسیم می کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می دهید.

گرفتن ورودی از کاربر

جاوا تعدادی متد برای گرفتن ورودی از کاربر در اختیار شما قرار می دهد. این متدها در کلاس Scanner قرار دارند. این کلاس در پکیج java.util قرار دارد و در نتیجه برای استفاده از آن باید آن را در برنامه به صورت زیر وارد کنید:

```
import java.util.Scanner;
```

از کلاس MessageFormat هم برای قالب بندی خروجی استفاده می کنیم. این دو کلاس را در خطوط 3 و 4 وارد کرده ایم. متدهای کلاس Scanner که مقادیر وارد شده توسط کاربر را از صفحه کلید می خوانند عبارتند از:

متد	توضیح
nextByte()	برای دریافت یک نوع داده از نوع byte به کار می رود.
nextShort()	برای دریافت یک نوع داده از نوع short به کار می رود.
nextInt()	برای دریافت یک نوع داده از نوع int به کار می رود.
nextLong()	برای دریافت یک نوع داده از نوع long به کار می رود.
next()	برای دریافت یک کلمه ساده به کار می رود.
nextLine()	برای دریافت یک خط رشته به کار می رود.
nextBoolean()	برای دریافت یک نوع داده از نوع boolean به کار می رود.
nextFloat()	برای دریافت یک نوع داده از نوع float به کار می رود.
nextDouble()	برای دریافت یک نوع داده از نوع double به کار می رود.

به برنامه زیر توجه کنید:

```
1:  package myfirstprogram;
```

```
 2:
 3:  import java.text.MessageFormat;
 4:  import java.util.Scanner;
 5:
 6:  public class MyFirstProgram
 7:  {
 8:      public static void main(String[] args)
 9:      {
10:          String name;
11:          int age;
12:          double height;
13:
14:          Scanner input = new Scanner(System.in);
15:
16:          System.out.print("Enter your Name: ");
17:          name = input.next();
18:
19:          System.out.print("Enter your Age: ");
20:          age = input.nextInt();
21:
22:          System.out.print("Enter your Height:");
23:          height = input.nextDouble();
24:
25:          System.out.println();
26:
27:          System.out.println(MessageFormat.format("Name is {0}.", name));
28:          System.out.println(MessageFormat.format("Age is {0}.", age));
29:          System.out.println(MessageFormat.format("Height is {0}.", height));
30:      }
31:  }
```

```
Enter your Name: john
Enter your Age: 18
Enter your Height:160.5

Name is john.
Age is 18.
Height is 160.5.
```

اجازه دهید که برنامه را تشریح کنیم. ابتدادر خطوط 3 و 4 برنامه، کلاس Scannerو MessageFormat را با استفاده از کلمه import به برنامه اضافه کرده ایم. در خطوط 10 و 11 و 12 یک شیء برای دریافت نام، یک متغیر از نوع صحیح به نام age برای دریافت سن و یک متغیر از نوع double برای دریافت قد شخص تعریف نموده ایم. درباره خط 14 زیاد توضیح نمی دهم، فقط کافیست که این رابدانید که وجود این خط برای دریافت ورودی از کاربر اجباری است. در درس های آینده با مفاهیم متد، شیء و کلاس آشنا خواهید شد. برنامه از کاربر می خواهد که نام خود را وارد کند (خط 16). در خط 17 شما به عنوان کاربر نام خود را وارد می کنید.مقدار متغیرname ، برابر مقداری است که توسط متد next() خوانده می شود.از آنجاییکه name از نوع رشته است باید از متد next() برای دریافت استفاده کنیم. در خط 19 برنامه از شما می خواهد که سن خود را وارد کند. در خط 20 شما سن خود را وارد می کنید.مقدار متغیرage ، برابر مقداری است که توسط متد nextInt() خوانده می شود.از آنجاییکه Ageاز نوع صحیح است باید از متد nextInt() برای دریافت استفاده کنیم. سپس برنامه از ما قد را سوال می کند.(خط 22) . چون height از نوع double است پس برای خواندن آن از متد nextDouble() استفاده کرده ایم. حال شما می توانید با اجرای برنامه و وارد کردن مقادیر نتیجه را مشاهده کنید.

ساختارهای تصمیم

تقریبا همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می دهند.حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود ان وقت با مشکل مواجه خواهید شد. جاوا راه های مختلفی برای رفع این نوع مشکلات ارائه می دهد. در این بخش با مطالب زیر آشنا خواهید شد:

- دستورif
- دستورif...else
- عملگر سه تایی
- دستور if چندگانه
- دستور if تو در تو
- عملگرهای منطقی
- دستورswitch

دستورif

می توان با استفاده از دستور if یک شرط خاص که باعث ایجاد یک کد می شود یک منطق به برنامه خود اضافه کنید.دستور if ساده ترین دستور شرطی است که برنامه می گوید اگر شرطی برقرار است کد معینی را انجام بده. ساختار دستور ifبه صورت زیر است:

```
if (condition)
   code to execute;
```

قبل از اجرای دستور ifابتدا شرط بررسی می شود.اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می شود.شرط یک عبارت مقایسه ای است. می توان از عملگرهای مقایسه ای برای تست درست یا اشتباه بودن شرط استفاده کرد.اجازه بدهید که نگاهی به نحوه استفاده از دستور ifدر داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از 10 و Goodbye World را اگر مقدار number از 10 بزرگتر باشد در صفحه نمایش می دهد.

```
1:  package myfirstprogram;
2:
3:  public class MyFirstProgram
4:  {
5:      public static void main(String[] args)
6:      {
```

```
 7:          //Declare a variable and set it a value less than 10
 8:          int number = 5;
 9:
10:          //If the value of number is less than 10
11:          if (number < 10)
12:                  System.out.println("Hello World.");
13:
14:          //Change the value of a number to a value which
15:          // is greater than 10
16:          number = 15;
17:
18:          //If the value of number is greater than 10
19:          if (number > 10)
20:                  System.out.println("Goodbye World.");
21:      }
22:  }
Hello World.
Goodbye World.
```

در خط 8 یک متغیر با نام number تعریف و مقدار 5 به آن اختصاص داده شده است. وقتی به اولین دستور if در خط 11 می رسیم برنامه تشخیص می دهد که مقدار number از 10 کمتر است یعنی 5 کوچکتر از 10 است.

منطقی است که نتیجه مقایسه درست می باشد بنابراین دستور if دستور را اجرا می کند(خط 12) و پیغام Hello World چاپ می شود. حال مقدار number را به 15 تغییر می دهیم (خط 16). وقتی به دومین دستور if در خط 19 می رسیم برنامه مقدار number را با 10 مقایسه می کند و چون مقدار number یعنی 15 از 10 بزرگتر است برنامه پیغام Goodbye World را چاپ می کند(خط 20). به این نکته توجه کنید که دستور if را می توان در یک خط نوشت:

```
if ( number > 10 ) System.out.println("Goodbye World.");
```

شما می توانید چندین دستور را در داخل دستور if بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جز بدنه دستور if هستند. نحوه تعریف چند دستور در داخل بدنه ifبه صورت زیر است:

```
if (condition)
{
    statement1;
    statement2;
    .
    .
    .
    statementN;
}
```

این هم یک مثال ساده:

```
if (x > 10)
{
   System.out.println("x is greater than 10.");
   System.out.println("This is still part of the if statement.");
}
```

در مثال بالا اگر مقدار x از 10 بزرگتر باشد دو پیغام چاپ می شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار x از 10 بزرگتر نباشد مانند کد زیر:

```
if (x > 10)
   System.out.println("x is greater than 10.");
   System.out.println("This is still part of the if statement. (Really?)");
```

کد بالا در صورتی بهتر خوانده می شود که بین دستورات فاصله بگذاریم :

```
if (x > 10)
   System.out.println("x is greater than 10.");

   System.out.println("This is still part of the if statement. (Really?)");
```

می بینِد که دستور دوم(خط 3) در مثال بالا جز دستور ifنیست.اینجاست که چون ما فرض را بر این گذاشته ایم که مقدار xاز 10 کوچکتر است پس خط This is still part of the if statement. (Really?)چاپ می شود. در نتیجه اهمیت وجود آکولاد مشخص می شود.به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه ifداشتید برای آن یک آکولاد بگذارید.فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطا شده و یافتن آن را سخت می کند. یکی از خطاهای معمول کسانی که برنامه نویسی را تازه شروع کرده اند قرار دادن سیمیکولن در سمت راست پرانتز if است. به عنوان مثال:

```
if (x > 10);
   System.out.println("x is greater than 10");
```

به یاد داشته باشید که ifیک مقایسه را انجام میدهد و دستور اجرایی نیست. بنابراین برنامه شما با یک خطای منطقی مواجه می شود. همیشه به یاد داشته باشید که قرار گرفتن سیمیکولن در سمت راست پرانتز ifبه منزله این است که بلوک کد در اینجا به پایان رسیده است. به این نکته توجه داشته باشید که شرطها مقادیر بولی هستند،بنابراین شما می توانید نتیجه یک عبارت را در داخل یک متغیر بولی ذخیره کنید و سپس از متغیر به عنوان شرط در دستور ifاستفاده کنید. اگر مقدار year برابر 2000 باشد سپس حاصل عبارت در متغیر isNewMillenium ذخیره می شود.می توان از متغیر برای تشخیص کد اجرایی بدنه دستور if استفاده کرد خواه مقدار متغیر درست باشد یا نادرست.

```
bool isNewMillenium = year == 2000;

if (isNewMillenium)
{
    System.out.println("Happy New Millenium!");
}
```

دستور if تو در تو

می توان از دستور if تو در تو در جاوا استفاده کرد. یک دستور ساده if در داخل دستور if دیگر.

```
if (condition)
{
    code to execute;

    if (condition)
    {
         code to execute;
    }
    else if (condition)
    {
        if (condition)
        {
           code to execute;
        }
    }
}
else
{
    if (condition)
    {
       code to execute;
    }
}
```

اجازه بدهید که نحوه استفاده از دستور if تو در تو را نشان دهیم:

```
 1: package myfirstprogram;
 2:
 3: public class MyFirstProgram
 4: {
 5:     public static void main(String[] args)
 6:     {
 7:         int age = 21;
 8:
 9:         if (age > 12)
10:         {
11:             if (age < 20)
12:             {
13:                 System.out.println("You are teenage");
14:             }
15:             else
16:             {
17:                 System.out.println("You are already an adult.");
18:             }
19:         }
20:         else
21:         {
22:             System.out.println("You are still too young.");
23:         }
24:     }
```

```
25:  }
You are already an adult.
```

اجازه بدهید که برنامه ا کالبد شکافی کنیم.ابتدا در خط 7 یک متغیر به نام age تعریف می کنیم و مقدار آن را برابر 21 قرار می دهیم.سپس به اولین دستور if می رسیم(خط 9). در این قسمت اگر سن شما بیشتر از 12 سال باشد برنامه وارد بدنه دستور if می شود در غیر اینصورت وارد بلوک else (خط 20) مربوط به همین دستور if می شود. حال فرض کنیم که ن شما بیشتر از 12 سال است و شما وارد بدنه اولین if شده اید.در بدنه اولین if یک دستور if دیگر را مشاهده می کنید.اگر سن کمتر 20 باشد دستور You are teenage چاپ می شود(خط 13) در غیر اینصورت دستور You are already an adult (خط 17) و چون مقدار متغیر تعریف شده در خط 7 بزرگتر از 20 است پس دستور مربوط به بخش else خط 17 چاپ می شود. حال فرض کنید که مقدار متغیر age کمتر از 12 بود، در این صورت دستور بخش else خط 20 یعنی You are still too youngچاپ می شد. پیشنهاد می شود که از if تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می آورد

عملگر شرطی

عملگر شرطی (:?) در جاوا مانند دستور شرطی if…else عمل می کند. در زیر نحوه استفاده از این عملگر آمده است:

```
<condition> ? <result if true> : <result if false>
```

عملگر شرطی تنها عملگر سه تایی جاوا است که نیاز ه سه عملوند دارد، شرط، یک مقدار زمانی که شرط درست باشد و یک مقدار زمانی که شرط نادرست باشد. اجازه بدهید که نحوه استفاده این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```
 1:  package myfirstprogram;
 2:
 3:  public class MyFirstProgram
 4:  {
 5:      public static void main(String[] args)
 6:      {
 7:          int number = -10;
 8:
 9:          int  ABS = (number > 0) ? (number) : -(number);
10:
11:          System.out.println("ABS      = " + ABS);
12:      }
13:  }
10
```

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می دهد. در این برنامه قصد ما به دست آوردن قدر مطلق یک عدد است. ابتدا در خط 7 یک متغیر از نوع int تعریف کرده و مقدار آن را 10- می گذاریم. در خط 9 یک متغیر از نوع صحیح تعریف کرده ایم تا نتیجه را در آن قرار دهیم. خط 9 به این صورت تعریف می شود : " اگر مقدار number از 10 بزرگتر باشد خود مقدار را در متغیر ABS قرار بده در غیر اینصورت آن را در منفی ضرب کرده و آن را در متغیر ABS قرار بده". حال برنامه بالا را با استفاده از دستور if else می نویسیم:

```
int number = -10;

if(number > 10)
{
    System.out.println(number);
}
else
{
    System.out.println(-(number));
}
```

هنگامی که چندین دستور در داخل یک بلوک if یا else دارید از عملگر شرطی استفاده نکنید چون خوانایی برنامه را پایین می آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می کنید؟ می توانید از چندین دستور ifاستفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید:

```
if (condition)
{
   code to execute;
}
else
{
   if (condition)
   {
      code to execute;
   }
   else
   {
      if (condition)
      {
         code to execute;
      }
      else
      {
         code to execute;
      }
   }
}
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می توانید کد بالا را ساده تر کنید:

```
if (condition)
{
```

```
   code to execute;
}
else if (condition)
{
   code to execute;
}
else if (condition)
{
   code to execute;
}
else
{
   code to execute;
}
```

حال که نحوه استفده از دستور else if را یاد گرفتید باید بدانید که مانند else ، else if نیز به دستور if وابسته است.دستور else if وقتی اجرا می شود که اولین دستور if اشتباه باشد حال اگر else if اشتباه باشد دستور else if بعدی اجرا می شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می شود. برنامه زیر نحوه استفاده از دستور if else را نشان می دهد:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {

        int choice = 2;


        if (choice == 1)
        {
            System.out.print("You might like my black t-shirt.");
        }
        else if (choice == 2)
        {
            System.out.print("You might be a clean and tidy person.");
        }
        else if (choice == 3)
        {
            System.out.print("You might be sad today.");
        }
        else
        {
            System.out.print("Sorry, your favorite color is " +
                             "not in the choices above.");
        }
    }
}
```

```
You might be a clean and tidy person
```

خروجی بنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می کنید پیغامهای مختلفی چاپ می شود.اگر عددی که شما تایپ می کنید در داخل حالتهای انتخاب نباشد کد مربوط به بلوک else اجرا می شود.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را در گیر می کنند و در آخر یک مقدار بولی را بر می گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است:

عملگر	تلفظ	مثال	تاثیر
&&	And	z = (x > 2) && (y < 10)	مقدار Z در صورتی true است که هر دو شرط دو طرف عملگر مقدارشان true باشد .اگر فقط مقدار یکی از شروط false باشد مقدار z برابر false خواهد شد.
||	Or	z = (x > 2) || (y < 10)	مقدار Z در صورتی true است که یکی از دو شرط دو طرف عملگر مقدارشان true باشد. اگر هر دو شرط مقدارشان false باشد مقدار z برابر false خواهد شد .
!	Not	z = !(x > 2)	مقدار Z در صورتی true است که مقدار شرط false باشد و در صورتی false است که مقدار شرط true باشد .

به عنوان مثال جمله z = (x > 2) && (y < 10) را به این صورت بخوانید : "در صورتی مقدار z برابر true است که مقدار x بزرگتر از 2 و مقدار y کوچکتر از 10 باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد. عملگر منطقی OR (||) تاثیر متفاوتی نسبت به عملگر منطقی AND (&&) دارد. نتیجه عملگر منطقی OR برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط true نباشد نتیجه false خواهد شد. می توان عملگرهای منطقی AND و OR را با هم ترکیب کرده و در یک عبارت به کار برد مانند:

```
if ( (x == 1) && ( (y > 3) || z < 10) ) )
{
    //do something here
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرطها استفاده می کنیم. در اینجا ابتدا عبارت (y > 3) || (z < 10)مورد بررسی قرار می گیرد.(به علت تقدم عملگرها) سپس نتیجه آن بوسیله عملگر AND با نتیجه (x == 1) مقایسه می شود. حال بیایید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم:

```
 1: package myfirstprogram;
 2:
 3: import java.util.Scanner;
 4:
 5: public class MyFirstProgram
 6: {
 7:     public static void main(String[] args)
 8:     {
 9:         int age;
10:         String gender;
```

```
11:
12:         Scanner input = new Scanner(System.in);
13:
14:         System.out.print("Enter your age: ");
15:         age = input.nextInt();
16:
17:         System.out.print("Enter your gender (male/female):");
18:         gender = input.next();
19:
20:         if (age > 12 && age < 20)
21:         {
22:             if (gender == "male")
23:             {
24:                 System.out.println("You are a teenage boy.");
25:             }
26:             else
27:             {
28:                System.out.println("You are not a teenage girl.");
29:             }
30:         }
31:         else
32:         {
33:             System.out.println("You are not a teenager.");
34:         }
35:     }
36: }
```

```
Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
Enter you age: 10
Enter your gender (male/female): male
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی AND را نشان می دهد (خط 20) .وقتی به دستور if می رسید (خط 20) برنامه سن شما را چک می کند.اگر سن شما بزرگتر از 12 و کوچکتر از 20 باشد (سن تان بین 12 و20 باشد) یعنی مقدار هر دو true باشد سپس کدهای داخل بلوک if اجرا می شوند.اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می شود. عملگر AND عملوند سمت چپ را مرود بررسی قرار می دهد.اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی کند و مقدار false را بر می گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می گیرد و مقدار true را بر می گرداند.

```
if (x == 2 & y == 3)
{
   //Some code here
}

if (x == 2 | y == 3)
{
   //Some code here
}
```

نکته مهم اینجاست که شما می توانید از عملگرهای & و | به عنوان عملگر بیتی استفاده کنید. تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد عملوند سمت چپ به وسیله عملگر بیتی AND (&) ارزیابی می شود. اگر شرطها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی AND (&&)و OR (||) به جای عملگرهای بیتی AND(&) و OR (|) بهتر خواهد بود. یکی دیگر از عملگرهای منطقی عملگر NOT (!) است که نتیجه یک عبارت را خنثی یا منفی می کند.به مثال زیر توجه کنید:

```
if (!(x == 2))
{
   System.out.println("x is not equal to 2.");
}
```

اگر نتیجه عبارت x == 2 برابر false باشد عملگر ! آن را True می کند.

دستور switch

در جاوا ساختاری به نام switch وجود دارد که به شما اجازه می دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور switch معادل دستور if تو در تو است با این تفاوت که در دستور switch متغیر فقط مقادیر ثابتی از اعداد ، رشته ها و یا کاراکترها را قبول می کند. مقادیر ثابت مقادیری هستند که قابل تغیر نیستند. در زیر نحوه استفاده از دستور switch آمده است:

```
switch (testVar)
{
   case compareVal1:
       code to execute if testVar == compareVa11;
       break;
   case compareVa12:
       code to execute if testVar == compareVa12;
       break;
   .
   .
   .
   case compareVa1N:
       code to execute if testVer == compareVa1N;
       break;
   default:
       code to execute if none of the values above match the testVar;
       break;
}
```

ابتدا یک مقدار در متغیر switch که در مثال بالا testVar است قرار می دهید.این مقدار با هر یک از عبارتهای caseداخل بلوک switch مقایسه می شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات caseبرابر بود کد مربوط به آن caseاجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور case از یکی بیشتر باشد نباید از آکولاد استفاده

کنیم. آخر هر دستور case با کلمه کلیدی break تشخیص داده می شود که باعث می شود برنامه از دستور switchخارج شده ودستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه می شود. دستور switchیک بخش defaultدارد. این دستور در صورتی اجرا می شود که مقدار متغیر با هیچ یک از مقادیر دستورات caseبرابر نباشد. دستور defaultاختیاری است و اگر از بدنه switchحذف شود هیچ اتفاقی نمی افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می نویسند. به مثالی در مورد دستور switchتوجه کنید:

```
1:  package myfirstprogram;
2:
3:  import java.util.Scanner;
4:
5:  public class MyFirstProgram
6:  {
7:      public static void main(String[] args)
8:      {
9:          Scanner input = new Scanner(System.in);
10:
11:         int choice;
12:
13:         System.out.println("What's your favorite pet?");
14:         System.out.println("[1] Dog");
15:         System.out.println("[2] Cat");
16:         System.out.println("[3] Rabbit");
17:         System.out.println("[4] Turtle");
18:         System.out.println("[5] Fish");
19:         System.out.println("[6] Not in the choices");
20:         System.out.print("Enter your choice: ");
21:
22:         choice = input.nextInt();
23:
24:         switch (choice)
25:         {
26:             case 1:
27:                 System.out.println("Your favorite pet is Dog.");
28:                 break;
29:             case 2:
30:                 System.out.println("Your favorite pet is Cat.");
31:                 break;
32:             case 3:
33:                 System.out.println("Your favorite pet is Rabbit.");
34:                 break;
35:             case 4:
36:                 System.out.println("Your favorite pet is Turtle.");
37:                 break;
38:             case 5:
39:                 System.out.println("Your favorite pet is Fish.");
40:                 break;
41:             case 6:
42:                 System.out.println("Your favorite pet is not in the choices.");
43:                 break;
44:             default:
45:                 System.out.println("You don't have a favorite pet.");
46:                 break;
47:         }
48:     }
```

```
49:  }
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه تان را می دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می کنید و این عدد در دستور switch با مقادیر caseمقایسه می شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر caseها برابر نبود دستور defaultاجرا می شود. یکی دیگر از ویژگیهای دستور switchاین است که شما می توانید از دو یا چند caseبرای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار number، 1، 2یا 3 باشد یک کد اجرا می شود. توجه کنید که caseها باید پشت سر هم نوشته شوند.

```
switch(number)
{
    case 1:
    case 2:
    case 3:
       System.out.println("This code is shared by three values.");
       break;
}
```

همانطور که قبلا ذکر شد دستور switchمعادل دستور ifتو در تو است. برنامه بالا را به صورت زیر نیز می توان نوشت:

```
if (choice == 1)
   System.out.println("Your favorite pet is Dog.");
else if (choice == 2)
   System.out.println("Your favorite pet is Cat.");
else if (choice == 3)
   System.out.println("Your favorite pet is Rabbit.");
else if (choice == 4)
   System.out.println("Your favorite pet is Turtle.");
else if (choice == 5)
   System.out.println("Your favorite pet is Fish.");
else if (choice == 6)
```

```
   System.out.println("Your favorite pet is not in the choices.");
else
   System.out.println("You don't have a favorite pet.");
```

کد بالا دقیقا نتیجه ای مانند دستور switch دارد.دستور default معادل دستور elseمی باشد. حال از بین این دو دستور (switch و if else) کدامیک را انتخاب کنیم. از دستور switch موقعی استفاده می کنیم که مقداری که می خواهیم با دیگر مقادیر مقایسه شود ثابت باشد. مثلا در مثال زیر هیچگاه از switch استفاده نکنید.

```
int myNumber = 5;
int x = 5;

switch (myNumber)
{
   case x:
      System.out.println("Error, you can't use variables as a value" +
                         " to be compared in a case statment.");
      break;
}
```

مشاهده می کنید که با اینکه مقدار x عدد 5 است و به طور واضح با متغیر myNumber مقایسه شده است برنامه خطا می دهد چون xیک ثابت نیست بلکه یک متغیر است یا به زبان ساده تر ،قابلیت تغییر را دارد. اگر بخواهید از x استفاده کنید و برنامه خطا ندهد باید از کلمه کلیدی final به صورت زیر استفاده کنید.

```
int myNumber = 5;
final int x = 5;

switch (myNumber)
{
   case x:
      System.out.println("Error has been fixed!");
      break;
}
```

از کلمه کلیدی final برای ایجاد ثابتها استفاده می شود.توجه کنید که بعد از تعریف یک ثابت نمی توان مقدار آن را در طول برنامه تغییر داد. به یاد داشته باشید که باید ثابتها را حتما مقداردهی کنید. دستور switch یک مقدار را با مقادیر case ها مقایسه می کند و شما لازم نیست که به شکل زیر مقادیر را با هم مقایسه کنید:

```
switch (myNumber)
{
   case x > myNumber:
      System.out.println("switch staments can't test if a value is less than " +
                         "or greater than the other value.");
      break;
}
```

تکرار

ساختارهای تکرار به شما اجازه می دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید.بدون ساختارهای تکرار شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است.مثلا شما مجبورید 10 بار جمله "Hello World."را تایپ کنید مانند مثال زیر:

```
System.out.println("Hello World.");
System.out.println("Hello World.");
System.out.println("Hello World.");
System.out.println("Hello World.");
System.out.println("Hello World.");
System.out.println("Hello World.");
System.out.println("Hello World.");
System.out.println("Hello World.");
System.out.println("Hello World.");
```

البته شما می توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می اورد.را بهتر برای نوشتن کدهای بالا استفاده از حلقه ها است. ساختارهای تکرار در جاوا عبارتند از:

- while
- do while
- for

حلقه While

ابتدایی ترین ساختار تکرار در جاوا حلقه While است.ابتدا یک شرط را مورد بررسی قرار می دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می شوند. ساختار حلقه Whileبه صورت زیر است:

```
while(condition)
{
    code to loop;
}
```

می بینید که ساختار Whileمانند ساختار if بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است مینویسیم اگر نتیجه درست یا trueباشد سپس کدهای داخل بلوک While اجرا می شوند. اگر شرط غلط یا false باشد وقتی که برنامه به حلقه While برسد هیچکدام از کدها را اجرا نمی کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه While اصلاح شوند.

به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم. در برنامه زیر نحوه استفاده از حلقه While آمده است:

```
1:  package myfirstprogram;
2:
3:  public class MyFirstProgram
4:  {
5:      public static void main(String[] args)
6:      {
7:          int counter = 1;
8:
9:          while (counter <= 10)
10:          {
11:              System.out.println("Hello World!");
12:              counter++;
13:          }
14:      }
15: }
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

برنامه بالا 10 بار پیغام Hello World! را چاپ می کند. اگر از حلقه در مثال بالا استفاده نمی کردیم مجبور بودیم تمام 10 خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط 7 یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است.سپس به آن مقدار 1 را اختصاص می دهیم چون اگر مقدار نداشته باشد نمی توان در شرط از آن استفاده کرد.

در خط 9 حلقه While را وارد می کنیم. در حلقه While ابتدا مقدار اولیه شمارنده با 10 مقایسه می شود که آیا از 10 کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه While و چاپ پیغام است .همانطور که مشاهده می کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می شود (خط 12). حلقه تا زمانی تکرار می شود که مقدار شمارنده از 10 کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می آید. به این نکته توجه کنید که در شرط بالا به جای علامت <= از < استفاده شده است. اگر از علامت < استفاده می کردیم کد ما 9 بار تکرار می شد چون مقدار اولیه 1 است و هنگامی که شرط به 10 برسد false می شود چون 10< 10 نیست. اگر می خواهید یک حلقه بی نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while(true)
{
   //code to loop
```

```
}
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات breakو returnکه در آینده توضیح خواهیم داد از حلقه خارج شوید.

حلقهdo While

حلقه do while یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه while است با این تفاوت که در این حلقه ابتدا کد اجرا می شودو سپس شرط مورد بررسی قرار می گیرد. ساختار حلقه do whileبه صورت زیر است:

```
do
{
   code to repeat;

} while (condition);
```

همانطور که مشاهده می کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می شوند.برخلاف حلقه whileکه اگر شرط نادرست باشد دستورات داخل بدنه اجرا نمی شوند.یکی از موراد برتری استفاده از حلقه do whileنسبت به حلقه whileزمانی است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. به مثال زیر توجه کنید:

استفاده ازwhile

```
//while version

System.out.print("Enter a number greater than 10: ");
number = input.nextInt();

while(number < 10)
{
   System.out.println("Enter a number greater than 10: ");
   number = input.nextInt();
}
```

استفاده ازdo while

```
//do while version

do
{
   System.out.println("Enter a number greater than 10: ");
   number = input.nextInt();
}
```

```
while(number < 10)
```

مشاهده می کنید که از کدهای کمتری در بدنه do while نسبت به whileاستفاده شده است.

حلقه for

یکی دیگر از ساختارهای تکرار حلقه for است.این حلقه عملی شبیه به حلقه while انجام می دهد و فقط دارای چند خصوصیت اضافی است. ساختار حلقه forبه صورت زیر است:

```
for(initialization; condition; operation)
{
   code to repeat;
}
```

مقدار دهی اولیه (initialization) اولین مقداری است که به شمارنده حلقه می دهیم.شمارنده فقط در داخل حلقه for قابل دسترسی است. شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می کند و تعیین می کند که حلقه ادامه یابد یا نه. عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می دهد. در زیر یک مثال از حلقه for آمده است:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        for(int i = 1; i <= 10; i++)
        {
            System.out.println("Number " + i);
        }
    }
}
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد 1 تا 10 را با استفاده از حلقه for می شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می کنیم و آن را با مقدار 1 مقدار دهی اولیه می کنیم. سپس با استفاده از شرط آن را با مقدار 10 مقایسه می کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (i++) فورا اجرا نمی شود. کد اجرا می شود و ابتدا رشته Number و سپس مقدار جاری i

یعنی 1 را چاپ می کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر 2 می شود و بار دیگر i با عدد 10 مقایسه می شود و این حلقه تا زمانی که مقدار شرط true شود ادامه می یابد. حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت زیر عمل کنید:

```
for (int i = 10; i > 0; i--)
{
   //code omitted
}
```

کد بالا اعداد را از 10 به 1 چاپ می کند (از بزرگ به کوچک). مقدار اولیه شمارنده را 10 می دهیم و با استفاده از عملگر کاهش (–) برنامه ای که شمارش معکوس را انجام می دهد ایجاد می کنیم. می توان قسمت شرط و عملگر را به صورت های دیگر نیز تغییر داد. به عنوان مثال می توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد. همچنین می توانید از چندین متغیر در ساختار حلقه for استفاده کنید.

```
for (int i = 1, y = 2; i < 10 && y > 20; i++, y -= 2)
{
   //some code here
}
```

به این نکته توجه کنید که اگر از چندین متغیر شمارنده یا عملگر در حلقه for استفاده می کنید باید آنها را با استفاده از کاما از هم جدا کنید. گاهی اوقات با وجود درست بودن شرط می خواهیم حلقه متوقف شود. سوال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از continue و breakرا نشان می دهد:

```
 1: package myfirstprogram;
 2:
 3: public class MyFirstProgram
 4: {
 5:     public static void main(String[] args)
 6:     {
 7:         System.out.println("Demonstrating the use of break\n");
 8:
 9:         for (int x = 1; x < 10; x++)
10:         {
11:             if (x == 5)
12:                 break;
13:
14:             System.out.println("Number " + x);
15:         }
16:
17:         System.out.println("\nDemonstrating the use of continue\n");
18:
19:         for (int x = 1; x < 10; x++)
20:         {
21:             if (x == 5)
22:                 continue;
23:
24:             System.out.println("Number "+ x);
25:         }
26:     }
```

```
27:  }
Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7
Number 8
Number 9
```

در این برنامه از حلقه for برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای for از حلقه های while و do…while استفاده می شد نتیجه یکسانی به دست می امد. همانطور که در شرط برنامه (خط 11) آمده است وقتی که مقدار x به عدد 5 رسید سپس دستور break اجرا شود(خط 12). حلقه بلافاصله متوقف می شود حتی اگر شرط x < 10 برقرار باشد.از طرف دیگر در خط 22 حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می یابد.(وقتی مقدار x برابر 5 شود حلقه از 5 رد شده و مقدار 5 را چاپ نمی کند و بقیه مقادیر چاپ می شوند.

آرایه ها

آرایه نوعی متغیر است که لیستی از آدرسهای مجموعه ای از داده های هم نوع را در خود ذخیره می کند.تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است.مثلا اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئنا تعریف این همه متغیر بسیار کسالت آور و خسته کننده است.اما با استفاده از آرایه می توان همه آنها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است:

```
datatype[] arrayName = new datatype[length];
```

Datatype نوع داده هایی را نشان می دهد که آرایه در خود ذخیره می کند.کروشه که بعد از نوع داده قرار می گیرد و نشان دهنده استفاده از آرایه است arrayName.که نام آرایه را نشان می دهد.هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد.به عنوان مثال برای نامگذاری آرایه ای که اعداد را در خود ذخیره می کند از کلمه number استفاده کنید.طول آرایه که به کامپایلر می گوید شما قصد دارید چه تعداد داده یا مقدار را در آرایه ذخیره کنید.از کلمه کلیدی new هم

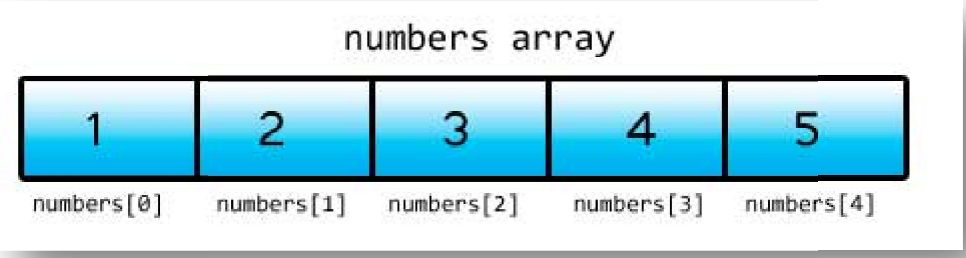
برای اختصاص فضای حافظه به اندازه طول آرایه استفاده می شود.برای تعریف یک آرایه که 5 مقدار از نوع اعداد صحیح در خود ذخیره می کند باید به صورت زیر عمل کنیم:

```
int[] numbers = new int[5];
```

در این مثال 5 آدرس از فضای حافظه کامپیوتر شما برای ذخیره 5 مقدار رزرو می شود. حال چطور مقادیرمان را در هر یک از این آدرسها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آنها استفاده می شود.

```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
```

اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می شود.به عنوان مثال شما یک آرایه 5 عضوی دارید،اندیس آرایه از 0 تا 4 می باشد چون طول آرایه 5 است پس 1-5 برابر است با 4.این بدان معناست که اندیس 0 نشان دهنده اولین عضو آرایه است و اندیس 1 نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید:



به هر یک از اجزاء آرایه و اندیسهای داخل کروشه توجه کنید.کسانی که تازه شروع به برنامه نویسی کرده اند معمولا در گذاشتن اندیس دچار اشتباه می شوند و مثلا ممکن است در مثال بالا اندیسها را از 1 شروع کنند. اگر بخواهید به یکی از اجزائ آرایه با استفاده از اندیسی دسترسی پیدا کنید که در محدوده اندیسهای آرایه شما نباشد با پیغام خطای ArrayIndexOutOfBoundsExceptionمواجه می شوید و بدین معنی است که شما آدرسی را می خواهید که وجود ندارد. یک راه بسیار ساده تر برای تعریف آرایه به صورت زیر است:

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

به سادگی و بدون احتیاج به کلمه کلیدی new می توان مقادیر را در داخل آکولاد قرار داد. کامپایلر به صورت اتوماتیک با شمارش مقادیر طول آرایه را تشخیص می دهد.

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه ها آمده است. در این برنامه 5 مقدار از کاربر گرفته شده و میانگین آنها حساب می شود:

```
1:  package myfirstprogram;
```

```
 2:
 3: import java.util.Scanner;
 4: import java.text.MessageFormat;
 5:
 6: public class MyFirstProgram
 7: {
 8:     public static void main(String[] args)
 9:     {
10:         Scanner input = new Scanner(System.in);
11:
12:         int[] numbers = new int[5];
13:         int total = 0;
14:         double average;
15:
16:         for (int i = 0; i < numbers.length; i++)
17:         {
18:             System.out.print("Enter a number: ");
19:             numbers[i] = input.nextInt();
20:         }
21:         for (int i = 0; i < numbers.length; i++)
22:         {
23:             total += numbers[i];
24:         }
25:
26:         average = total / (double)numbers.length;
27:
28:         System.out.println(MessageFormat.format("Average = {0}", average));
29:     }
30: }
```

```
Enter a number: 90
Enter a number: 85
Enter a number: 80
Enter a number: 87
Enter a number: 92
Average = 86
```

در خط 12 یک آرایه تعریف شده است که می تواند 5 عدد صحیح را در خود ذخیره کند.خطوط 13 و 14 متغیرهایی تعریف شده اند که از آنها برای محاسبه میانگین استفاده می شود.توجه کنید که مقدار اولیه total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود.در خطوط 16 تا 20 حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است.از خاصیت طول (length) ارایه برای تشخیص تعداد اجزای آرایه استفاده می شود.اگر چه می توانستیم به سادگی در حلقه for مقدار 5 را برای شرط قرار دهیم ولی استفاده از خاصیت طول آرایه کار راحت تری است و می توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ می شود.در خط 19 ورودی دریافت شده از کاربر با استفاده از متد nextInt() دریافت و در آرایه ذخیره می شود.اندیس استفاده شده در number (خط 19) مقدار i جاری در حلقه است.برای مثال در ابتدای حلقه مقدار i صفر است بنابراین وقتی در خط 19 اولین داده از کاربر گرفته می شود اندیس آن برابر صفر می شود. در تکرار بعدی i یک واحد اضافه می شود و در نتیجه در خط 19 و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می یابد. در خطوط 21- 24 از حلقه for دیگر برای دسترسی به مقدار هر یک از داده های آرایه استفاده شده است.در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می کنیم.

هر یک از اجزای عددی آرایه به متغیر total اضافه می شوند.بعد از پایان حلقه می توانیم میانگین اعداد را حساب کنیم (خط 26). مقدار total را بر تعداد اجزای آرایه (تعداد عدد ها) تقسیم می کنیم.برای دسترسی به تعداد اجزای آرایه می توان از خاصیت lengthآرایه استفاده کرد.توجه کنید که در اینجا ما مقدار خاصیت length را به نوع doubleتبدیل کرده ایم بنابراین نتیجه عبارت یک مقدار از نوع double خواهد شد و دارای بخش کسری می باشد .حال اگر عملوند های تقسیم را به نوع double تبدیل نکنیم نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط 28 مقدار میانگین را در صفحه نمایش چاپ می کند. طول آرایه بعد از مقدار دهی نمی تواند تغیر کند .به عنوان مثال اگر یک آرایه را که شامل 5 جز است مقدار دهی کنید دیگر نمی توانید آن را مثلا به 10 جز تغییر اندازه دهید.البته تعداد خاصی از کلاسها مانند آرایه ها عمل می کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند.آرایه ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آنها استفاده کنید بسیار مهم است.

حلقهforeach

حلقه foreach یکی دیگر از ساختارهای تکرار در جاوا می باشد که مخصوصا برای آرایه ها ،لیستها و مجموعه ها طراحی شده است.حلقه foreach با هر بار گردش در بین اجزاء ، مقادیر هر یک از آنها را در داخل یک متغیر موقتی قرار می دهد و شما می توانید بواسطه این متغیر به مقادیر دسترسی پیدا کنید. در زیر نحوه استفاده از حلقه foreach آمده است:

```
for (datatype temporaryVar : array)
{
    code to execute;
}
```

temporaryVar متغیری است که مقادیر اجزای آرایه را در خود نگهداری می کند.باید دارای نوع باشد تا بتواند مقادیر آرایه را در خود ذخیره کند.به عنوان مثال آرایه شما دارای اعدادی از نوع صحیح باشد باید نوع متغیر موقتی از نوع اعداد صحیح باشد یا هر نوع دیگری که بتواند اعداد صحیح را در خود ذخیره کند مانند double یا long. سپس علامت دو نقطه (:)و بعد از آن نام آرایه را می نویسیم. در زیر نحوه استفاده از حلقه foreach امده است:

```
 1: package myfirstprogram;
 2:
 3: public class MyFirstProgram
 4: {
 5:     public static void main(String[] args)
 6:     {
 7:         int[] numbers = { 1, 2, 3, 4, 5 };
 8:
 9:         for (int n : numbers)
10:         {
11:             System.out.println(n);
12:         }
13:     }
14: }
```

```
1
2
```

```
3
4
5
```

در برنامه آرایه ای با 5 جزء تعریف شده و مقادیر 1 تا 5 در آنها قرار داده شده است (خط 7). در خط 9 حلقه foreachشروع می شود.ما یک متغیر موقتی تعریف کرده ایم که اعداد آرایه را در خود ذخیره می کند. در هر بار تکرار از حلقه foreachمتغیر موقتی n، مقادیر عددی را از آرایه استخراج می کند. حلقه foreachمقادیر اولین تا آخرین جزء آرایه را در اختیار ما قرار می دهد.

حلقه foreach برای دریافت هر یک از مقادیر آرایه کاربرد دارد.بعد از گرفتن مقدار یکی از اجزای آرایه ،مقدار متغیر موقتی را چاپ می کنیم.حلقه foreach ما را قادر می سازد که به داده ها دسترسی یابیم و یا آنها را بخوانیم و اصلاح کنیم. برای درک این مطلب در مثال زیر مقدار هر یک از اجزا آرایه افزایش یافته است:

```
int[] numbers = { 1, 2, 3 };

for(int n : numbers)
{
   n++;
   System.out.println(n);
}
```

```
2
3
4
5
6
```

آرایه های چند بعدی

آرایه های چند بعدی آرایه هایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم.یک آرایه چند بعدی را می توان مانند یک جدول با تعدای ستون و ردیف تصور کنید.با افزایش اندیسها اندازه ابعاد آرایه نیز افزایش می یابد و آرایه های چند بعدی با بیش از دو اندیس به وجود می آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است:

```
datatype[][] arrayName = new datatype[lengthX][lengthY];
```

و یک آراه سه بعدی به صورت زیر ایجاد می شود:

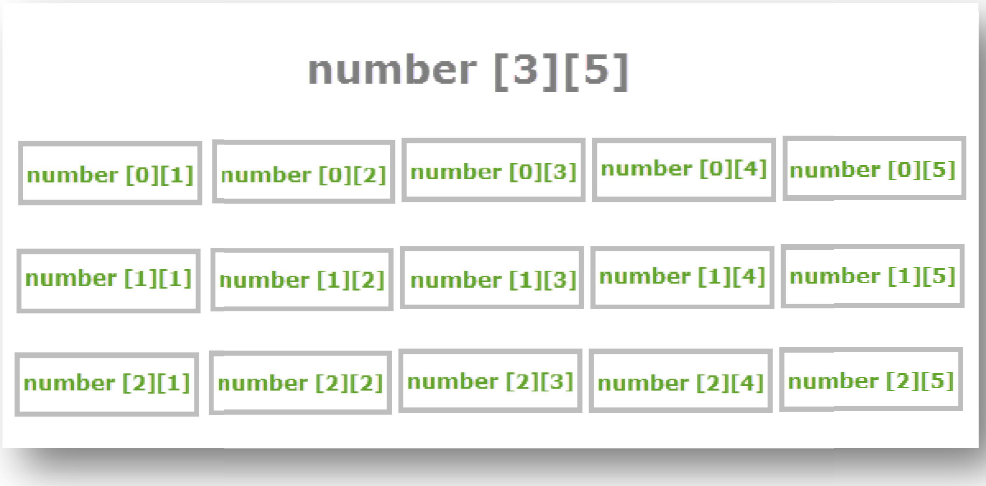
```
datatype[][][] arrayName = new datatype[lengthX][lengthY][lengthZ];
```

می تان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. به دلیل اینکه آرایه های سه بعدی یا آرایه های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می گیرند اجازه بدهید که در این درس بر روی آرایه های دو بعدی تمرکز کنیم.در تعریف این نوع آرایه ابتدا نوع آرایه یعنی اینکه آرایه چه نوعی از انواع داده را در خود ذخیره می کند را مشخص می کنیم.سپس دو جفت کروشه قرار می دهیم. به تعداد کروشه ها توجه کنید.اگر آرایه ما دو بعدی است باید 2 جفت

کروشه و اگر سه بعدی است باید 3 جفت کروشه قرار دهیم.سپس یک نام برای آرایه انتخاب کرده و بعد تعریف آنرا با گذاشتن کلمه new ، نوع داده و طول هر بعد آن کامل می کنیم.در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی مقدار X و دیگری مقدار Y که مقدار x نشان دهنده ردیف و مقدار Y نشان دهنده ستون آرایه است البته اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم.یک آرایه سه بعدی را می توان به صورت یک مکعب تصور کرد که دارای سه بعد است و x طول، Yعرض و z ارتفاع آن است. یک مثال از آرایه دو بعدی در زیر آمده است:

```
int[][] numbers = new int[3][5];
```

کد بالا به کامپایلر می گوید که فضای کافی به عناصر آرایه اختصاص بده)در این مثال 15 خانه). در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار 3 را به x اختصاص می دهیم چون 3 سطر و مقدار 5 را به Y چون 5 ستون داریم اختصاص می دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ چند راه برای مقدار دهی به آرایه ها وجود دارد. یک راه این است که مقادیر عناصر آرایه را در همان زمان تعریف آرایه، مشخص کنیم:

```
datatype[][] arrayName = { { r0c0, r0c1, ... r0cX },
                           { r1c0, r1c1, ... r1cX },
                                    .
                                    .
                                    .
                           { rYc0, rYc1, ... rYcX } };
```

به عنوان مثال:

```
 int[][] numbers = { { 1, 2, 3, 4, 5 },
                     { 6, 7, 8, 9, 10 },
                     { 11, 12, 13, 14, 15 } };
```

و یا می توان مقدار دهی به عناصر را به صورت دستی انجام داد مانند:

```
array[0][0] = value;
array[0][1] = value;
```

```
array[0][2] = value;
array[1][0] = value;
array[1][1] = value;
array[1][2] = value;
array[2][0] = value;
array[2][1] = value;
array[2][2] = value;
```

همانطور که مشاهده می کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می توان از اندیسهای X و Y و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه های چند بعدی

گردش در میان عناصر آرایه های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده از حلقه foreach و یا حلقه forتو در تو است. اجازه دهید ابتدا از حلقه foreach استفاده کنیم.

```
 1: package myfirstprogram;
 2:
 3: public class MyFirstProgram
 4: {
 5:     public static void main(String[] args)
 6:     {
 7:         int[][] numbers = { { 1, 2, 3, 4, 5 },
 8:                             { 6, 7, 8, 9, 10 },
 9:                             { 11, 12, 13, 14, 15 }
10:                           };
11:
12:         for (int[] number : numbers)
13:         {
14:             for (int num : number)
15:             {
16:                 System.out.print(num + " ");
17:             }
18:         }
19:     }
20: }
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

مشاهده کردید که گردش در میان مقادیر عناصر یک آرایه چند بعدی چقدر راحت است. به وسیله حلقه foreach نمی توانیم انتهای ردیفها را مشخص کنیم .برنامه زیر نشان می دهد که چطور از حلقه for برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف ها استفاده کنید.

```
 1: package myfirstprogram;
 2:
 3: public class MyFirstProgram
 4: {
 5: public static void main(String[] args)
 6: {
 7:     int[][] numbers = { { 1, 2, 3, 4, 5 },
```

```
 8:                 { 6, 7, 8, 9, 10 },
 9:                 { 11, 12, 13, 14, 15 }
10:                 };
11:
12:         for (int row = 0; row < numbers.length; row++)
13:         {
14:             for (int col = 0; col < numbers[row].length; col++)
15:             {
16:                 System.out.print(numbers[row][col] + " ");
17:             }
18:
19:             //Go to the next line
20:             System.out.println();
21:         }
22:     }
23: }
```

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

همانطور که درمثال بالا نشان داده شده است با استفاده از یک حلقه ساده forنمی توان به مقادیر دسترسی یافت بلکه به یک حلقه for تو در تونیاز داریم. در اولین حلقه for (خط 12) یک متغیر تعریف شده است که در میان ردیف های آرایه (row) گردش می کند. این حلقه تا زمانی ادامه می یابد که مقدار ردیف کمتر از طول اولین بعد باشد. در این مثال از خاصیت lenght کلاس Array استفاده کرده ایم.این خاصیت طول آرایه را در یک بعد خاص نشان می دهد. به عنوان مثال برای به دست آوردن طول اولین بعد آرایه که همان تعداد ردیف ها می باشد از دستور numbers.length استفاده کرده ایم.

در داخل اولین حلقه for حلقه for دیگری تعریف شده است (خط 14). در این حلقه یک شمارنده برای شمارش تعداد ستونهای (columns) هر ردیف تعریف شده است و در شرط داخل آن بار دیگر از خاصیت lenght استفاده شده است، ولی این بار به نوعی دیگر از آن استفاده می کنیم. همانطور که مشاهده می کنید ایتدا نام آرایه را نوشته ایم و سپس یک اندیس به آن اختصاص داده ایم و بعد از خاصیت lenght استفاده نموده ایم:

```
numbers[row].length
```

استفاده از row به عنوان اندیس باعث می شود که به عنوان مثال وقتی که مقدار ردیف (row) صفر باشد ، حلقه دوم از [0][0] تا [4][0] اجرا شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می دهیم، اگر مقدار ردیف (row)برابر 0 و مقدار ستون (col) برابر 0 باشد مقدار عنصری که در ستون 1و ردیف 1 (numbers[0][0]) قرار دارد نشان داده خواهد شد که در مثال بالا عدد 1 است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فورا دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور System.out.println()که به برنامه اطلاع می دهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار row این فرایند را دوباره تکرار می کند. سپس دومین حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می شود. این فرایند تا زمانی اجرا می شود که مقدار row کمتر از طول اولین بعد باشد. حال بیایید انچه را از قبل یاد گرفته ایم در یک برنامه به کار بریم.این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می گیرد و معدل سه دانش آموز را حساب می کند.

```
1: package myfirstprogram;
2:
```

```
 3: import java.util.Scanner;
 4: import java.text.MessageFormat;
 5:
 6: public class MyFirstProgram
 7: {
 8: public static void main(String[] args)
 9: {
10:         Scanner input = new Scanner(System.in);
11:
12:         double[][] studentGrades = new double[3][4];
13:         double total;
14:
15:         for (int student = 0; student < studentGrades.length; student++)
16:         {
17:             total = 0;
18:
19:             System.out.println(MessageFormat.format("Enter grades for Student {0}",
20: student + 1));
21:
22:             for (int grade = 0; grade < studentGrades[student].length; grade++)
23:             {
24:                 System.out.print(MessageFormat.format("Enter Grade #{0}: ", grade +
25: 1));
26:                 studentGrades[student][grade] = input.nextDouble();
27:                 total += studentGrades[student][grade];
28:             }
29:
30:             System.out.print(MessageFormat.format("Average is {0}", (total /
31: studentGrades[student].length)));
32:             System.out.println();
33:         }
34:     }
35: }
```

```
Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75

Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75

Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
Average is 90.00
```

در برنامه بالا یک آرایه چند بعدی از نوع double تعریف شده است (خط 12 .(همچنین یک متغیر به نام total تعریف می کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم. حال وارد حلقه for تو در تو می شویم (خط 15).در اولین حلقه forیک متغیر به نام sudent برای تشخیص پایه درسی هر دانش آموز تعریف کرده ایم. از خاصیت lenght هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه for می شویم.در خط 12 مقدار متغیر totalرا برابر صفر قرار می دهیم. بعدا مشاهده می کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می دهد و از شما می خواهد که شماره دانش آموز را وارد کنید .(student + 1) عدد 1 را به student اضافه کرده ایم تا به جای نمایش0 Student ، با 1 Student شروع شود، تا طبیعی تر به نظر برسد. سپس به دومین حلقه for در خط 21 می رسیم. در این حلقه یک متغیر شمارنده به نام gradeتعریف می کنیم که طول دومین بعد آرایه را با استفاده از studentGrades[student].length به دست می آورد. این طول تعداد نمراتی را که برنامه از سوال می کند را نشان می دهد. برنامه چهار نمره مربوط به دانش آموز را می گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می کند، نمره به متغیر total اضافه می شود.

وقتی همه نمره ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می دهد. در خط 28 معدل دانش آموز نشان داده می شود. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می آید. از studentGrades[student].length هم برای به دست آوردن تعداد نمرات استفاده می شود.

آرایه دندانه دار

آرایه دندانه دار یا jagged array آرایه ای چند بعدی است که دارای سطرهای با طول متغیر می باشد. نمونه ساده ای از آرایه های چند بعدی ، آرایه های مستطیلی است که تعداد ستون ها و سطرهای آنها برابر است. اما آرایه های دندانه دار دارای سطرهای (آرایه های) با طول متفاوت می باشند .بنابر این آرایه های دندانه دار را می توان آرایه ای از آرایه ها فرض کرد .دستور نوشتن این نوع آرایه ها به صورت زیر است:

```
datatype[][] arrayName;
```

ابتدا datatype که نوع آرایه است و سپس چهار کروشه باز و بسته و بعد از آن نام آرایه را می نویسیم. مقداردهی به این آرایه ها کمی گیج کننده است .به مثال زیر توجه کنید:

```
int[][] myArrays = new int[3][];

myArrays[0] = new int[3];
myArrays[1] = new int[5];
myArrays[2] = new int[2];
```

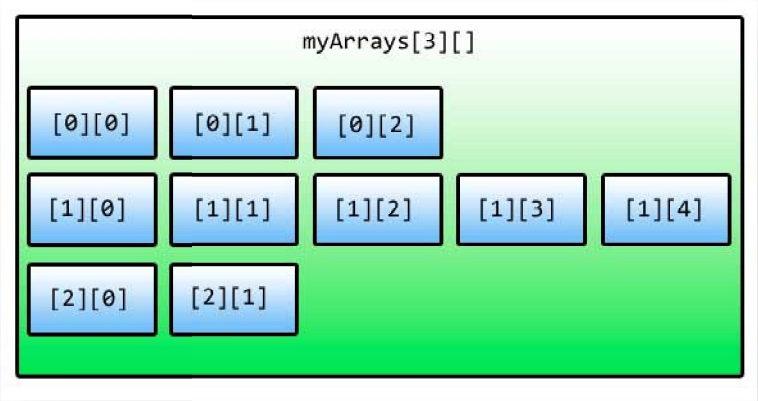
در کد بالا سه آرایه تعریف شده است که اندیس آنها از صفر شروع می شود .اعداد 3 و 5 و 2 هم به تعداد عناصری که هر کدام از آنها در خود می توانند جای دهند اشاره دارند. برای مقداردهی هر آرایه هم باید ابتدا اندیس آرایه و سپس اندیس عناصر آن را بنویسیم. مثلا مقداردهی اولین عنصر اولین آرایه مثال بالا به صورت زیر عمل می کنیم:

```
myArrays[0][0] = 1;
```

و برای مثلا دومین عنصر دومین آرایه هم به صورت زیر:

```
myArrays[1][1] = 4;
```

شکل زیر هم اندیس عناصر آرایه ای که در بالا تعریف کرده ایم را نشان می دهد:



با توجه به توضیحاتی که داده شد می توان عناصر آرایه ای که در ابتدای درس ایجاد کردیم را به صورت زیر مقداردهی کرد:

```
myArrays[0][0] = 1;
myArrays[0][1] = 2;
myArrays[0][2] = 3;

myArrays[1][0] = 5;
myArrays[1][1] = 4;
myArrays[1][2] = 3;
myArrays[1][3] = 2;
myArrays[1][4] = 1;

myArrays[2][0] =11;
myArrays[2][1] =22;
```

یک روش بهتر برای مقدار دهی آرایه های دندانه دار به صورت زیر است که در آن می توان طول سطرها را هم مشخص نکرد:

```
int[][] myArrays = {{1,2,3}, {5,4,3,2,1}, {11,22}};
```

برای دسترسی به مقدار عناصر یک آرایه دندانه دار باید اندیس سطر و ستون آن را در اختیار داشته باشیم:

```
array[row][column]

System.out.println(myArrays[1][2]);
```

نمی توان از حلقه foreach برای دسترسی به عناصر آرایه دندانه دار استفاده کرد:

```
for(int array : myArrays)
{
    System.out.println(array);
}
```

اگر از حلقه foreach استفاده کنیم با خطا مواجه می شویم چون عناصر این نوع آرایه ها ، آرایه هستند نه عدد یا رشته یا... .برای حل این مشکل باید نوع متغیر موقتی (array) را تغییر داده و از حلقه foreach دیگری برای دسترسی به مقادیر استفاده کرد.

```
for(int[] array : myArrays)
{
   for(int number : array)
   {
      System.out.println(number);
   }
}
```

همچنین می توان از یک حلقه for تو در تو به صورت زیر استفاده کرد:

```
for (int row = 0; row < myArrays.length; row++)
{
   for (int col = 0; col < myArrays[row].length; col++)
   {
      System.out.println(myArrays[row][col]);
   }
}
```

در اولین حلقه از length برای به دست آوردن تعداد سطرها (که همان آرایه های یک بعدی هستند)و در دومین حلقه از length برای به دست آوردن عناصر سطر جاری استفاده می شود.

متد

متدها به شما اجازه می دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه ای از کدها هستند که در هر جای برنامه می توان از آنها استفاده کرد.متدها دارای آرگومانهایی هستند که وظیفه متد را مشخص می کنند.متد در داخل کلاس تعریف می شود. نمی توان یک متد را در داخل متد دیگر تعریف کرد .وقتی که شما در برنامه یک متد را صدا می زنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می کند. در جاوا متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه ها نمی دانند با ید از کجا

شروع شوند ، این متد main() نام دارد. پارامتر ها همان چیزهایی هستند که متد منتظر دریافت آنها است. آرگومانها مقادیری هستند که به پارامترها ارسال می شوند .گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می روند. ساده ترین ساختار یک متد به صورت زیر است:

```
returnType MethodName()
{
   code to execute;
}
```

به برنامه ساده زیر توجه کنید.در این برنامه از یک متد برای چاپ یک پیغام در صفحه نمایش استفاده شده است:

```
1:  package myfirstprogram;
2:
3:  public class MyFirstProgram
4:  {
5:     static void PrintMessage()
6:     {
7:         System.out.println("Hello World!");
8:     }
9:
10:     public static void main(String[] args)
11:     {
12:         PrintMessage();
13:     }
14: }
Hello World!
```

در خطوط 8-5 یک متد تعریف کرده ایم. مکان تعریف آن در داخل کلاس مهم نیست. به عنوان مثال می توانید آن را زیر متد main()تعریف کنید. می توان این متد را در داخل متد دیگر صدا زد (فراخوانی کرد). متد دیگر ما در اینجا متد main() است که می توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده ایم (یعنی متد PrintMessage()) را صدا بزنیم. متد main()به صورت static تعریف شده است. برای اینکه بتوان از متد PrintMessage() در داخل متد main() استفاده کنیم باید آن را به صورت static تعریف کنیم. کلمه static به طور ساده به این معناست که می توان از متد استفاده کرد بدون اینکه از کلاس نمونه ای ساخته شود. متد main() همواره باید به صورت static تعریف شود چون برنامه فورا و بدون نمونه سازی از کلاس از آن استفاده می کند. وقتی به مبحث برنامه نویسی شی گرا رسیدید به طور دقیق کلمهstatic مورد بحث قرار می گیرد. برنامهclass (مثال بالا) زمانی اجرا می شود که برنامه دو متدی را که تعریف کرده ایم را اجرا کند و متد main() به صورت static تعریف شود. در باره این کلمه کلیدی در درسهای آینده مطالب بیشتری می آموزیم.

در تعریف متد بالا بعد از کلمه static کلمه کلیدی void آمده است که نشان دهنده آن است که متد مقدار برگشتی ندارد. در درس آینده در مورد مقدار برگشتی از یک متد و استفاده از آن برای اهداف مختلف توضیح داده خواهد شد .نام متد ما PrintMessage() است. به این نکته توجه کنید که در نامگذاری متد از روش پاسکال (حرف اول هر کلمه بزرگ نوشته می شود) استفاده کرده ایم. این روش نامگذاری قراردادی است و می توان از این روش استفاده نکرد، اما پیشنهاد می شود که از این روش برای تشخیص متدها استفاده کنید. بهتر است در نامگذاری متدها از کلماتی استفاده شود که کار ان متد را مشخص می کند مثلا نام هایی مانند GoToBed یا. OpenDoor همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد می توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید مانند IsLeapyear یا IsTeenager... ولی از گذاشتن علامت سوال در آخر اسم متد خودداری کنید. دو پرانتزی که بعد از نام می آید نشان دهنده آن است که نام متعلق به یک متد است. در این

مثال در داخل پرانتز ها هیچ چیزی نوشته نشده چون پارامتری ندارد.در درسهای آینده در مورد متدها بیشتر توضیح می دهیم. بعد از پرانتزها دو آکولاد قرار می دهیم که بدنه متد را تشکیل می دهد و کدهایی را که می خواهیم اجرا شوند را در داخل این آکولاد ها می نویسیم. در داخل متد main() متدی را که در خط 12 ایجاد کرده ایم را صدا می زنیم. برای صدا زدن یک متد کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم. اگر متد دارای پارامتر باشد باید شما آراگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درسهای آینده توضیح بیشتری می دهیم. با صدا زدن یک متد کدهای داخل بدنه آن اجرا می شوند. برای اجرای متد PrintMessage() برنامه از متد main() به محل تعریف متد PrintMessage() می رود. مثلا وقتی ما متد PrintMessage() را در خط 12 صدا می زنیم برنامه از خط 12 به خط 7 ، یعنی جایی که متد تعریف شده می رود. اکنون ما یک متد در برنامه class داریم و همه متدهای این برنامه می توانند آن را صدا بزنند.

مقدار برگشتی از یک متد

متدها می توانند مقدار برگشتی از هر نوع داده ای داشته باشند .این مقادیر می توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک متد است و شما او را صدا می زنید و از او می خواهید که کار یک سند را به پایان برساند. سپس از او میخواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی متد است. نکته مهم در مورد یک متد، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک متد آسان است. کافیست در تعریف متد به روش زیر عمل کنید:

```
returnType MethodName()
{
   return value;
}
```

returnType در اینجا نوع داده ای مقدار برگشتی را مشخص می کند (int، bool،...) .در داخل بدنه متد کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می نویسیم. نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و قبل از نام متد ذکر شود .اگر متد ما مقدار برگشتی نداشته باشد باید از کلمه void قبل از نام متد استفاده کنیم. مثال زیر یک متد که دارای مقدار برگشتی است را نشان می دهد.

```
1:  package myfirstprogram;
2:
3:  import java.text.MessageFormat;
4:
5:  public class MyFirstProgram
6:  {
7:      static int CalculateSum()
8:      {
9:          int firstNumber = 10;
10:         int secondNumber = 5;
11:
12:         int sum = firstNumber + secondNumber;
13:
14:         return sum;
```

```
15:     }
16:
17:     public static void main(String[] args)
18:     {
19:         int result = CalculateSum();
20:
21:         System.out.println(MessageFormat.format("Sum is {0}.", result));
22:     }
23: }
Sum is 15.
```

همانطور که در خط 7 مثال فوق مشاهده می کنید هنگام تعریف متد از کلمه int به جای void استفاده کرده ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط 9 و 10 دو متغیر تعریف و مقدار دهی شده اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر متدها مانند متد main() قابل دسترسی نیستند و فقط در متدی که در آن تعریف شده اند قابل استفاده هستند. در خط 12 جمع دو متغیر در متغیر sum قرار می گیرد. در خط 14 مقدار برگشتی sum توسط دستور return فراخوانی می شود. در داخل متد main() یک متغیربه نام resultدر خط 19 تعریف می کنیم و متد CalculateSum() را فراخوانی می کنیم.

متد CalculateSum() مقدار 15 را بر می گرداند که در داخل متغیر result ذخیره می شود. در خط 21 مقدار ذخیره شده در متغیر result چاپ می شود. متدی که در این مثال ذکر شد متد کاربردی و مفیدی نیست.با وجودیکه کدهای زیادی در متد بالا نوشته شده ولی همیشه مقدار برگشتی 15 است، در حالیکه می توانستیم به راحتی یک متغیر تعریف کرده و مقدار 15 را به آن اختصاص دهیم. این متد در صورتی کارامد است که پارامترهایی به آن اضافه شود که در درسهای آینده توضیح خواهیم داد . هنگامی که می خواهیم در داخل یک متد از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  import java.util.Scanner;
 4:  import java.text.MessageFormat;
 5:
 6:  public class MyFirstProgram
 7:  {
 8:      static int GetNumber()
 9:      {
10:          Scanner input = new Scanner(System.in);
11:
12:          int number;
13:
14:          System.out.print("Enter a number greater than 10: ");
15:          number = input.nextInt();
16:
17:          if (number > 10)
18:          {
19:              return number;
20:          }
21:          else
22:          {
23:              return 0;
24:          }
```

```
25:     }
26:
27:     public static void main(String[] args)
28:     {
29:     int result = GetNumber();
30:
31:     System.out.println(MessageFormat.format("Result = {0}.", result));
32:     }
33: }
```

```
Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط 25-8 یک متد با نام GetNumber() تعریف شده است که از کاربر یک عدد بزرگتر از 10 را می خواهد. اگر عدد وارد شده توسط کاربر درست نباشد متد مقدار صفر را بر می گرداند. و اگر قسمت else دستورif و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می شویم.

چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از 10 را وارد کند) برنامه به قسمت else می رود تا مقدار صفر را بر گرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می شود و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد پیغام خطا می دهد. و آخرین مطلبی که در این درس می خواهیم به شما آموزش دهیم این است که شما می توانید از یک متد که مقدار برگشتی ندارد خارج شوید. حتی اگر از نوع داده ای void در یک متد استفاده می کنید باز هم می توانید کلمه کلیدی return را در آن به کار ببرید. استفاده از return باعث خروج از بدنه متد و اجرای کدهای بعد از آن می شود.

```
1:  package myfirstprogram;
2:
3:  public class MyFirstProgram
4:  {
5:      static void TestReturnExit()
6:      {
7:          System.out.println("Line 1 inside the method TestReturnExit()");
8:          System.out.println("Line 2 inside the method TestReturnExit()");
9:
10:             return;
11:
12:         //The following lines will not execute
13:         System.out.println("Line 3 inside the method TestReturnExit()");
14:         System.out.println("Line 4 inside the method TestReturnExit()");
15:     }
16:
17:     public static void  main() (String[] args)
18:     {
19:         TestReturnExit();
20:         System.out.println("Hello World!");
21:     }
22: }
```

```
Line 1 inside the method TestReturnExit()
Line 2 inside the method TestReturnExit()
Hello World!
```

در برنامه بالا نحوه خروج از متد با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه متد تعریف شده (TestReturnExit()) در داخل متد main() فراخوانی و اجرا می شود.

پارامتر و آرگومان

پارامترها داده های خامی هستند که متد آنها را پردازش می کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می دهید که بر طبق آنها کارش را به پایان برساند. یک متد می تواند هر تعداد پارامتر داشته باشد. هر پارامتر می تواند از انواع مختلف داده باشد. در زیر یک متد با N پارامتر نشان داده شده است:

```
returnType MethodName(datatype param1, datatype param2, ... datatype paramN)
{
   code to execute;
}
```

پارامترها بعد از نام متد و بین پرانتزها قرار می گیرند. بر اساس کاری که متد انجام می دهد می توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومانهای آن را نیز تامین کنید. آرگومانها مقادیری هستند که به پارامترها اختصاص داده می شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می شود. اجازه بدهید که یک مثال بزنیم:

```
 1: package myfirstprogram;
 2:
 3: import java.util.Scanner;
 4: import java.text.MessageFormat;
 5:
 6: public class MyFirstProgram
 7: {
 8:     static int CalculateSum(int number1, int number2)
 9:     {
10:         return number1 + number2;
11:     }
12:
13:     public static void main(String[] args)
14:     {
15:         Scanner input = new Scanner(System.in);
16:         int num1, num2;
17:
18:         System.out.print("Enter the first number: ");
19:         num1 = input.nextInt();
20:         System.out.print("Enter the second number: ");
21:         num2 = input.nextInt();
22:
```

```
23:         System.out.println(MessageFormat.format("Sum = {0}", CalculateSum(num1,
24: num2)));
25:     }
26: }
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک متد به نام CalculateSum() (خطوط 8-11) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می کند پس نوع برگشتی ما نیز باید int باشد. متد دارای دو پارامتر است که اعداد را به آنها ارسال می کنیم. به نوع داده ای پارامترها توجه کنید. هر دو پارامتر یعنی number1 و number2 مقادیری از نوع اعداد صحیح (int) دریافت می کنند. در بدنه متد دستور return نتیجه جمع دو عدد را بر می گرداند. در داخل متد main() برنامه از کاربر دو مقدار را درخواست می کند و آنها را داخل متغیر ها قرار می دهد. حال متد را که آرگومانهای آن را آماده کرده ایم فراخوانی می کنیم. مقدار num1 به پارامتر اول و مقدار num2 به پارامتر دوم ارسال می شود. حال اگر مکان دو مقدار را هنگام ارسال به متد تغییر دهیم (یعنی مقدار num2 به پارامتر اول و مقدار num1 به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه متد ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومانها هنگام فراخوانی متد دقیقا با ترتیب قرار گیری پارامترها تعریف شده در متد مطابقت داشته باشد. بعد از ارسال مقادیر 10 و 5 به پارامترها ، پارامترها آنها را دریافت می کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (حرف اول دومین کلمه بزرگ نوشته می شود) نوشته می شود. در داخل بدنه متد (خط 10) دو مقدار با هم جمع می شوند و نتیجه به متد فراخوان (متدی که متد CalculateSum() را فراخوانی می کند) ارسال می شود.

در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می کنیم ولی در اینجا مشاهده می کنید که میتوان به سادگی نتیجه جمع را نشان داد (خط 10) .در داخل متد main() از ما دو عدد که قرار است با هم جمع شوند درخواست می شود.

در خط 23 متد CalculateSum() را فراخوانی می کنیم و دو مقدار صحیح به آن ارسال می کنیم. دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آنها برگردانده می شود. مقدار برگشت داده شده از متد به وسیله متد format() از کلاس MessageFormat نمایش داده می شود. (خط 23) در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده ای مختلف است:

```
 1: package myfirstprogram;
 2:
 3: import java.text.MessageFormat;
 4:
 5: public class MyFirstProgram
 6: {
 7:     static void ShowMessageAndNumber(string message, int number)
 8:     {
 9:         System.out.println(message);
10:         System.out.println(MessageFormat.format("Number = {0}", number));
11:     }
12:
13:     public static void main(String[] args)
14:     {
15:         ShowMessageAndNumber("Hello World!", 100);
```

```
16:     }
17: }
Hello World!
Number = 100
```

در مثال بالا یک متدی تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع int دریافت می کند. متد به سادگی دو مقداری که به آن ارسال شده است را نشان می دهد. در خط 15 متد را اول با یک رشته و سپس یک عدد خاص فراخوانی می کنیم . حال اگر متد به صورت زیر فراخوانی می شد:

```
ShowMessageAndNumber(100, "Welcome to Gimme C#!");
```

در برنامه خطا به وجود می آمد چون عدد 100 به پارامتری از نوع رشته و رشته Hello World! به پارامتری از نوع اعداد صحیح ارسال می شد.این نشان می دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی متد مهم است.

به مثال 1 توجه کنید در آن مثال دو عدد از نوع int به پارامترها ارسال کردیم که ترتیب ارسال آنها چون هردو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای متد دارای اهداف خاصی باشند ترتیب ارسال آرگومانها مهم است.

```
void ShowPersonStats(int age, int height)
{
    System.out.println(MessageFormat.format("Age = {0}", age));
    System.out.println(MessageFormat.format("Height = {0}", height));
}

//Using the proper order of arguments
ShowPersonStats(20, 160);

//Acceptable, but produces odd results
ShowPersonStats(160, 20);
```

در مثال بالا نشان داده شده است که حتی اگر متد دو آرگومان با یک نوع داده ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود .به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی آید چون سن شخص 20 و قد او 160 سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم سن شخص 160 و قد او 20 سانتی متر می شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می شود که شما متدهای کارامد تری تعریف کنید. تکه کد زیر نشان می دهد که شما حتی می توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

```
int MyMethod()
{
   return 5;
}

void AnotherMethod(int number)
{
   System.out.println(number);
}

// Codes skipped for demonstration

AnotherMethod(MyMethod());
```

چون مقدار برگشتی متد MyMethod() عدد 5 است و به عنوان آرگومان به متد AnotherMethod() ارسال می شود خروجی کد بالا هم عدد 5 است.

ارسال آرگومان به روش مقدار

ارسال آرگومانها به روش مقدار بدان معناست که شما یک کپی از مقدار متغیر را ارسال می کنید نه اصل متغیر یا ارجاع به آن را. در این حالت وقتی که آرگومان ارسال شده را در داخل متد اصلاح می کنیم مقدار اصلی آرگومان در خارج از متد تغییر نمی کند. اجازه دهید که ارسال با مقدار آرگومان را با یک مثال توضیح دهیم:

```
1:  package myfirstprogram;
2:
3:  import java.text.MessageFormat;
4:
5:  public class MyFirstProgram
6:  {
7:       static void ModifyNumberVal(int number)
8:       {
9:            number += 10;
10:           System.out.println(MessageFormat.format("Value of number inside method is
11: {0}.", number));
12:      }
13:
14:      public static void main(String[] args)
15:      {
16:           int num = 5;
17:           System.out.println(MessageFormat.format("num = {0}\n",num));
18:
19:           System.out.println("Passing num by value to method ModifyNumberVal() ...");
20:           ModifyNumberVal(num);
21:           System.out.println(MessageFormat.format("Value of num after exiting the
22: method is {0}", num));
23:      }
24: }
```

```
num = 5

Passing num by value to method ModifyNumberVal() ...
Value of number inside method is 15.
Value of num after exiting the method is 5.
```

در برنامه بالا متدی تعریف شده است که کار آن اضافه کردن عدد 10 به مقداری است که به آنها ارسال می شود (خطوط 11-7). این متد دارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع int) دارد. وقتی که متد را صدا می زنیم و آرگومانی به آن اختصاص می دهیم (خط 19) ،کپی آرگومان به پارامتر متد ارسال می شود. بنابراین مقدار اصلی متغیر خارج از متد هیچ ارتباطی به پارامتر متد ندارد. سپس مقدار 10 را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می کنیم.

برای اثبات اینکه متغیر numهیچ تغییری نکرده است مقدار آن را یکبار قبل از ارسال به متد (خط 16) و بار دیگر بعد از ارسال به متد (خط 20) چاپ کرده و مشاهده می کنیم که تغییر ی نکرده است .

ارسال آرایه به عنوان آرگومان

می توان آرایه ها را به عنوان آرگومان به متد ارسال کرد. ابتدا شما باید پارامترهای متد را طوری تعریف کنید که آرایه دریافت کنند. به مثال زیر توجه کنید.

```
1:  package myfirstprogram;
2:
3:  public class MyFirstProgram
4:  {
5:      static void TestArray(int[] numbers)
6:      {
7:          for(int number : numbers)
8:          {
9:             System.out.println(number);
10:         }
11:      }
12:
13:      public static void main(String[] args)
14:      {
15:          int[] array = { 1, 2, 3, 4, 5 };
16:          TestArray(array);
17:      }
18: }
```

```
1
2
3
4
5
```

مشاهده کردید که به سادگی می توان با گذاشتن کروشه بعد از نوع داده ای پارامتر یک متد ایجاد کرد که پارامتر آن ، آرایه دریافت می کند. وقتی متد در خط 16 فراخوانی می شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می کنیم. پس آرایه ها به روش ارجاع به متدها ارسال می شوند. در خطوط 10-7 از حلقه foreach برای دسترسی به اجزای اصلی آرایه که به عوان آرگومان به متد ارسال کرده ایم استفاده می کنیم .در زیر نحوه ارسال یک آرایه به روش ارجاع نشان داده شده است.

```
1:  package myfirstprogram;
2:
3:  public class MyFirstProgram
4:  {
5:      static void IncrementElements(int[] numbers)
6:      {
7:          for (int i = 0; i < numbers.length; i++)
```

```
8:          {
9:             numbers[i]++;
10:         }
11:     }
12:
13:     public static void main(String[] args)
14:     {
15:         int[] array = { 1, 2, 3, 4, 5 };
16:
17:         IncrementElements(array);
18:
19:         for (int num : array)
20:         {
21:             System.out.println(num);
22:         }
23:     }
24: }
2
3
4
5
6
```

برنامه بالا یک متد را نشان می دهد که یک آرایه را دریافت می کند و به هر یک از عناصر آن یک واحد اضافه می کند. به این نکته توجه کنید که از حلقه **foreach** نمی توان برای افزایش مقادیر آرایه استفاده کنیم چون این حلقه برای خواندن مقادیر آرایه مناسب است نه اصلاح آنها. در داخل متد ما مقادیر هر یک از اجزای آرایه را افزایش داده ایم..سپس از متد خارج شده و نتیجه را نشان می دهیم. مشاهده می کنید که هر یک از مقادیر اصلی متد هم اصلاح شده اند. راه دیگر برای ارسال آرایه به متد ، مقدار دهی مستقیم به متد فراخوانی شده است .به عنوان مثال:

```
IncrementElements( new int[] { 1, 2, 3, 4, 5 } );
```

در این روش ما آرایه ای تعریف نمی کنیم بلکه مجموعه ای از مقادیر را به پارامتر ارسال می کنیم که آنها را مانند آرایه قبول کند. از آنجاییکه در این روش آرایه ای تعریف نکرده ایم نمی توانیم در متد **Main**نتیجه را چاپ کنیم. اگر از چندین پارامتر در متد استفاده می کنید همیشه برای هر یک از پارامترهایی که آرایه قبول می کنند از یک جفت کروشه استفاده کنید. به عنوان مثال:

```
void MyMethod(int[] param1, int param2)
{
   //code here
}
```

به پارامترهای متد بالا توجه کنید ، پارامتر اول **(param1)** آرگومانی از جنس آرایه قبول می کند ولی پارامتر دوم **(param2)** یک عدد صحیح. حال اگر پارامتر دوم **(param2)**هم آرایه قبول می کرد باید برای آن هم از کروشه استفاده می کردیم:

```
void MyMethod(int[] param1, int[] param2)
{
   //code here
}
```

محدوده متغیر

متدها در جاوا دارای محدوده هستند. محدوده یک متغیر به شما می گوید که در کجای برنامه می توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متد تعریف می شود فقط در داخل بدنه متد قابل دسترسی است. می توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می کند:

```
1:  package myfirstprogram;
2:
3:  import java.text.MessageFormat;
4:
5:  public class MyFirstProgram
6:  {
7:       static void DemonstrateScope()
8:       {
9:           int number = 5;
10:
11:          System.out.println(MessageFormat.format("number inside method
12: DemonstrateScope() = {0}", number));
13:      }
14:
15:      public static void main(String[] args)
16:      {
17:          int number = 10;
18:
19:          DemonstrateScope();
20:
21:          System.out.println(MessageFormat.format("number inside the Main method() =
22: {0}", number));
23:      }
24: }
```

```
number inside method DemonstrateScope() = 5
number inside the Main method() = 10
```

مشاهده می کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده های متفاوتی هستند، می توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل متد Main()در خط 9 هیچ ارتباطی به متغیر داخل متد DemonstrateScope()در خط 16 ندارد. وقتی به مبحث کلاسها رسیدیم در این باره بیشتر توضیح خواهیم داد.

سربارگذاری متدها

سر بارگذاری متدها به شما اجازه می دهد که دو متد با نام یکسان تعریف کنید که دارای امضا و تعداد پارامترهای مختلف هستند.برنامه از روی آرگومانهایی که شما به متد ارسال می کنید به صورت خودکار تشخیص می دهد که کدام متد را فراخوانی کرده اید یا کدام متد مد نظر شماست. امضای یک متد نشان دهنده ترتیب و نوع پارامترهای آن است. به مثال زیر توجه کنید:

```
void MyMethod(int x, double y, string z)
```

که امضای متد بالا MyMethod(int, double, string) می باشد. به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضای متد نمی شوند. در مثال زیر نمونه ای از سربارگذاری متد ها آمده است.

```
1:  package myfirstprogram;
 2:
 3:  public class MyFirstProgram
 4:  {
 5:      static void ShowMessage(double number)
 6:      {
 7:          System.out.println("Double version of the method was called.");
 8:      }
 9:
10:      static void ShowMessage(int number)
11:      {
12:          System.out.println("Integer version of the method was called.");
13:      }
14:
15:      public static void main(String[] args)
16:      {
17:          ShowMessage(9.99);
18:          ShowMessage(9);
19:      }
20:  }
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو متد با نام مشابه تعریف شده اند. اگر سربارگذاری متد توسط سی شارپ پشتیبانی نمی شد برنامه زمان زیادی برای انتخاب یک متد از بین متدهایی که فراخوانی می شوند لازم داشت. رازی در نوع پارامترهای متد نهفته است. کامپایلر بین دو یا چند متد در صورتی فرق می گذارد که پارامترهای متفاوتی داشته باشند. وقتی یک متد را فراخوانی می کنیم ، متد نوع آرگومانها را تشخیص می دهد. در فراخوانی اول (خط 17) ما یک مقدار doubleرا به متد ShowMessage() ارسال کرده ایم در نتیجه متد ShowMessage() (خطوط 8-5) که دارای پارامتری از نوع double اجرا می شود. در بار دوم که متد فراخوانی می شود (خط 18) ما یک مقدار intرا به متد ShowMessage() ارسال می کنیم متد ShowMessage() (خطوط 13-10) که دارای پارامتری از نوع int است اجرا می شود. معنای اصلی سربارگذاری متد همین است که توضیح داده شد. هدف اصلی از سربارگذاری متدها این است که بتوان چندین متد که وظیفه یکسانی انجام می دهند را تعریف کرد تعداد زیادی از متدها در جاوا سربارگذاری می شوند مانند متد println() از کلاس out . قبلا مشاهده کردید که این متد می تواند یک آرگومان از نوع رشته دریافت کند و آن را نمایش دهد، و در حالت دیگر می تواند دو یا چند آرگومان قبول کند.

بازگشت(Recursion)

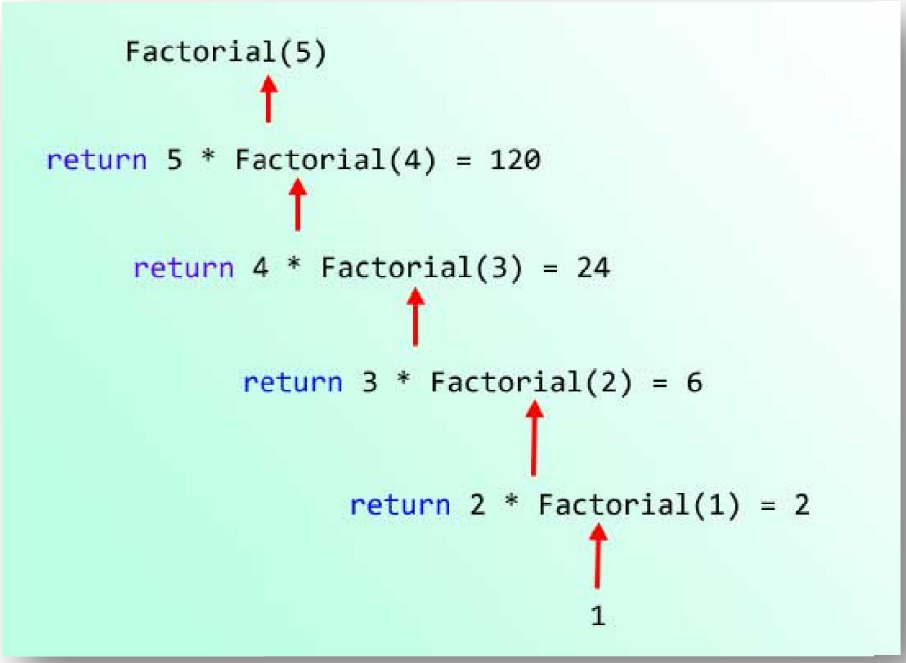
بازگشت فرایندی است که در آن متد مدام خود را فراخوانی می کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه نویسی است و تسط به آن کار را حتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای بی نهایت بار، متد، خود را فراخوانی می کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می دهیم. فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچکتر یا مساوی آن می باشد. به فاکتوریل عدد 5 توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

بنابراین برای ساخت یک متد بازگشتی باید به فکر توقف آن هم باشیم.بر اساس توضیح بازگشت ، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچکترین عدد صحیح مثبت 1 است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می کنیم.

```
 1:  package myfirstprogram;
 2:
 3:  public class MyFirstProgram
 4:  {
 5:      static long Factorial(int number)
 6:      {
 7:          if (number == 1)
 8:              return 1;
 9:
10:          return number * Factorial(number - 1);
11:      }
12:
13:      public static void main(String[] args)
14:      {
15:          System.out.println(Factorial(5));
16:      }
17:  }
120
```

متد مقدار بزرگی را بر می گرداند چون محاسبه فاکتوریل می تواند خیلی بزرگ باشد. متد یک آرگومان که یک عدد است و می تواند در محاسبه مورد استفاده قرار گیرد را می پذیرد. در داخل متد یک دستور if می نویسیم و در خط 7می گوییم که اگر آرگومان ارسال شده برابر 1 باشد سپس مقدار 1 را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می شود .در خط 10 مقدار جاری متغیر numberدر عددی یک واحد کمتر از خودش (number – 1) ضرب می شود.در این خط متد Factorialخود را فراخوانی می کند و آرگومان آن در این خط همان number – 1است. مثلا اگر مقدار جاری number 10باشد یعنی اگر ما بخواهیم فاکتوریل عدد 10 را به دست بیاوریم آرگومان متد Factorialدر اولین ضرب 9 خواهد بود. فرایند ضرب تا زمانی ادامه می یابد که آرگومان ارسال شده با عدد 1 برابر نشود. شکل زیر فاکتوریل عدد 5 را نشان می دهد.



کد بالا را به وسیله یک حلقه for نیز می توان نوشت.

```
factorial = 1;

for ( int counter = number; counter >= 1; counter-- )
   factorial *= counter
```

این کد از کد معادل بازگشتی آن آسان تر است. از بازگشت در زمینه های خاصی در علوم کامپیوتر استفاده می شود. استفاده از بازگشت حافظه زیادی اشغال می کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

برنامه نویسی شیء گرا(OOP)

برنامه نویسی شی گرا یاObject Oriented Programming ، شامل تعریف کلاسها و ساخت اشیاء مانند ساخت اشیاء در دنیای واقعی است. برای مثال یک ماشین را در نظر بگیرید.این ماشین دارای خواصی مانند رنگ ، سرعت ، مدل ، سازنده و برخی خواص دیگر است. همچنین دارای رفتارها و حرکاتی مانند شتاب و پیچش به چپ و راست و ترمز است.اشیاء در سی شارپ

تقلیدی از یک شی مانند ماشین در دنیای واقعی هستند.برنامه نویسی شی گرا با استفاده از کدهای دسته بندی شده کلاسها و اشیاء را بیشتر قابل کنترل می کند.

در ابتدا ما نیاز به تعریف یک کلاس برای ایجاد اشیاء مان داریم.شیء در برنامه نویسی شی گراء از روی کلاسی که شما تعریف کرده اید ایجاد می شود.برای مثال نقشه ساختمان شما یک کلاس است که ساختمان از روی آن ساخته شده است.کلاس شامل خواص یک ساختمان مانند مساحت،بلندی و مواد مورد استفاده در ساخت خانه می باشد. در دنیای واقعی ساختمان ها نیز بر اساس یک نقشه (کلاس) پایه گذاری(تعریف) شده اند. برنامه نویسی شیء گرا یک روش جدید در برنامه نویسی است که بوسیله برنامه نویسان مورد استفاده قرار می گیرد و به آنها کمک می کند که برنامه هایی با قابلیت استفاده مجدد، خوانا و راحت طراحی کنند.جاوا نیز یک برنامه شی گراست. در درس زیر به شما نحوه تعریف کلاس و استفاده از اشیاء آموزش داده خواهد شد.همچنین شما با دو مفهوم وراثت و چند ریختی که از مباحث مهم در برنامه نویسی شیء گرا هستند در آینده آشنا می شوید.

کلاس

کلاس به شما اجازه می دهد یک نوع داده ای که توسط کاربر تعریف می شود و شامل فیلدها و خواص (properties) و متدها است را ایجاد کنید. کلاس در حکم یک نقشه برای یک شی می باشد.

شی یک چیز واقعی است که از ساختار، خواص و یا رفتارهای کلاس پیروی می کند. وقتی یک شی می سازید یعنی اینکه یک نمونه از کلاس ساخته اید (در درس ممکن است از کلمات شی و نمونه به جای هم استفاده شود).

ابتدا ممکن است فکر کنید که کلاس ها و ساختارها شبیه هم هستند. تفاوت مهم بین این دو این است که کلاسها از نوع مرجع و ساختارها از نوع داده ای هستند. در درسهای آینده این موضوع شرح داده خواهد شد.

اگر یادتان باشد در بخشهای اولیه این آموزش کلاسی به نام MyFirstProgram تعریف کردیم که شامل متد main()بود و ذکر شد که این متد نقطه آغاز هر برنامه است. برای تعریف یک کلاس از کلمه کلیدی classبه صورت زیر استفاده می شود:

```
class ClassName
{
    field1;
    field2;
    ...
    fieldN;

    method1;
    method2;
    ...
    methodN;
}
```

این کلمه کلیدی را قبل از نامی که برای کلاسمان انتخاب می کنیم می نویسیم. در نامگذاری کلاسها هم از روش نامگذاری Pascalاستفاده می کنیم. در بدنه کلاس فیلدها و متدهای آن قرار داده می شوند .فیلدها اعضای داده ای خصوصی هستند که

کلاس از آنها برای رفتارها و ذخیره مقادیر خاصیت هایش (property) استفاده می کند. متدها رفتارها یا کارهایی هستند که یک کلاس می تواند انجام دهد. در زیر نحوه تعریف و استفاده از یک کلاس ساده به نام personنشان داده شده است.

```
1:  package myfirstprogram;
 2:
 3:  import java.text.MessageFormat;
 4:
 5:  class Person
 6:  {
 7:      public String name;
 8:      public int age;
 9:      public double height;
10:
11:      public void TellInformation()
12:      {
13:          System.out.println(MessageFormat.format("Name: {0}", name));
14:          System.out.println(MessageFormat.format("Age: {0} years old", age));
15:          System.out.println(MessageFormat.format("Height: {0}cm", height));
16:      }
17:  }
18:
19:  public class MyFirstProgram
20:  {
21:      public static void main(String[] args)
22:      {
23:          Person firstPerson = new Person();
24:          Person secondPerson = new Person();
25:
26:          firstPerson.name = "Jack";
27:          firstPerson.age = 21;
28:          firstPerson.height = 160;
29:          firstPerson.TellInformation();
30:
31:          System.out.println(); //Separator
32:
33:          secondPerson.name = "Mike";
34:          secondPerson.age = 23;
35:          secondPerson.height = 158;
36:          secondPerson.TellInformation();
37:      }
38:  }
```

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm
```

برنامه بالا شامل دو کلاس) Person خطوط 17-5) و) MyFirstProgramخطوط 38-19) می باشد. می دانیم که کلاس MyFirstProgramشامل متد main()است که برنامه برای اجرا به آن احتیاج دارد ولی اجازه دهید که بر روی کلاس

Personتمرکز کنیم. در خطوط 17-5 کلاس Personتعریف شده است. در خط 5 یک نام به کلاس اختصاص داده ایم تا به وسیله آن قابل دسترسی باشد. در داخل بدنه کلاس فیلدهای آن تعریف شده اند (خطوط 7-9.)

این سه فیلد تعریف شده خصوصیات واقعی یک فرد در دنیای واقعی را در خود ذخیره می کنند. یک فرد در دنیای واقعی دارای نام، سن، و قد می باشد. در خطوط 16-11 یک متد هم در داخل کلاس به نام TellInformation()تعریف شده است که رفتار کلاسمان است و مثلا اگر از فرد سوالی بپرسیم در مورد خودش چیزهایی می گوید. در داخل متد کدهایی برای نشان دادن مقادیر موجود در فیلدها نوشته شده است. نکته ای درباره فیلدها وجود دارد و این است که چون فیلدها در داخل کلاس تعریف و به عنوان اعضای کلاس در نظر گرفته شده اند محدوده آنها یک کلاس است.

این بدین معناست که فیلدها فقط می توانند در داخل کلاس یعنی جایی که به آن تعلق دارند و یا به وسیله نمونه ایجاد شده از کلاس مورد استفاده قرار بگیرند. در داخل متد main()و در خطوط 23 و 24 دو نمونه یا دو شی از کلاس Personایجاد می کنیم. برای ایجاد یک نمونه از یک کلاس باید از کلمه کلیدی newو به دنبال آن نام کلاس و یک جفت پرانتز قرار دهیم. وقتی نمونه کلاس ایجاد شد، سازنده را صدا می زنیم. یک سازنده متد خاصی است که برای مقداردهی اولیه به فیلدهای یک شی به کار می رود. وقتی هیچ آرگومانی در داخل پرانتزها قرار ندهید ،کلاس یک سازنده پیشفرض بدون پارامتر را فراخوانی می کند .درباره سازنده ها در درس های آینده توضیح خواهیم داد. در خطوط 29-26 مقادیری به فیلدهای اولین شی ایجاد شده از کلاس Person (first Person)اختصاص داده شده است. برای دسترسی به فیلدها یا متدهای یک شی از علامت نقطه (دات) استفاده می شود. به عنوان مثال کد firstPerson.nameنشان دهنده فیلد nameاز شی firstPersonمی باشد. برای چاپ مقادیر فیلدها باید متد TellInformation()شی firstPersonرا فراخوانی می کنیم.

در خطوط 36-33 نیز مقادیری به شی دومی که قبلا از کلاس ایجاد شده تخصیص می دهیم و سپس متد TellInformation() را فراخوانی می کنیم. به این نکته توجه کنید که firstPersonو secondPersonنسخه های متفاوتی از هر فیلد دارند بنابراین تعیین یک نام برای secondPersonهیچ تاثیری بر نام firstPersonندارد .در مورد اعضای کلاس در درسهای آینده توضیح خواهیم داد.

سازنده

سازنده ها متدهای خاصی هستند که وجود آنها برای ساخت اشیا لازم است. آنها به شما اجازه می دهند که مقادیری را به هر یک از اعضای داده ای یک آرایه اختصاص دهید و کدهایی که را که می خواهید هنگام ایجاد یک شی اجرا شوند را به برنامه اضافه کنید. اگر از هیچ سازنده ای در کلاس تان استفاده نکنید، کامپایلر از سازنده پیشفرض که یک سازنده بدون پارامتر است استفاده می کند. می توانید در برنامه تان از تعداد زیادی سازنده استفاده کنید که دارای پارامترهای متفاوتی باشند.در مثال زیر یک کلاس که شامل سازنده است را مشاهده می کنید:

```
1:  package myfirstprogram;
 2:
 3:  import java.text.MessageFormat;
 4:
 5:  class Person
 6:  {
 7:      public String name;
```

```
 8:     public int age;
 9:     public double height;
10:
11:     //Explicitly declare a default constructor
12:     public Person()
13:     {
14:     }
15:
16:     //Constructor that has 3 parameters
17:     public Person(String n, int a, double h)
18:     {
19:         name = n;
20:         age = a;
21:         height = h;
22:     }
23:
24:     public void ShowInformation()
25:     {
26:         System.out.println(MessageFormat.format("Name: {0}", name));
27:         System.out.println(MessageFormat.format("Age: {0} years old", age));
28:         System.out.println(MessageFormat.format("Height: {0}cm", height));
29:     }
30: }
31:
32: public class MyFirstProgram
33: {
34:     public static void main(String[] args)
35:     {
36:         Person firstPerson = new Person();
37:         Person secondPerson = new Person("Mike", 23, 158);
38:
39:         firstPerson.name = "Jack";
40:         firstPerson.age = 21;
41:         firstPerson.height = 160;
42:         firstPerson.ShowInformation();
43:
44:         System.out.println(); //Seperator
45:
46:         secondPerson.ShowInformation();
47:     }
48: }
```

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm
```

همانطور که مشاهده می کنید در مثال بالا دو سازنده را به کلاس Personاضافه کرده ایم. یکی از آنها سازنده پیشفرض (خطوط 12-14) و دیگری سازنده ای است که سه آرگومان قبول می کند (خطوط 19-21). به این نکته توجه کنید که سازنده درست شبیه به یک متد است با این تفاوت که

- نه مقدار برگشتی دارد و نه از نوع voidاست.
- نام سازنده باید دقیقا شبیه نام کلاس باشد.

سازنده پیشفرض در داخل بدنه اش هیچ چیزی ندارد و وقتی فراخوانی می شود که ما از هیچ سازنده ای در کلاس مان استفاده نکنیم. در آینده متوجه می شوید که چطور می توان مقادیر پیشفرضی به اعضای داده ای اختصاص داد، وقتی که از یک سازنده پیشفرض استفاده می کنید . به دومین سازنده توجه کنید. اولا که نام آن شبیه نام سازنده اول است. سازنده ها نیز مانند متدها می توانند سربارگذاری شوند. حال اجازه دهید که چطور می توانیم یک سازنده خاص را هنگام تعریف یک نمونه از کلاس فراخوانی کنیم.

```
Person firstPerson = new Person();
Person secondPerson = new Person("Mike", 23, 158);
```

در اولین نمونه ایجاد شده از کلاس Personاز سازنده پیشفرض استفاده کرده ایم چون پارامتری برای دریافت آرگومان ندارد. در دومین نمونه ایجاد شده، از سازنده ای استفاده می کنیم که دارای سه پارامتر است. کد زیر تاثیر استفاده از دو سازنده مختلف را نشان می دهد:

```
firstPerson.name = "Jack";
firstPerson.age = 21;
firstPerson.height = 160;
firstPerson.ShowInformation();

System.out.println(); //Seperator

secondPerson.ShowInformation();
```

همانطور که مشاهده می کنید لازم است که به فیلدهای شی ای که از سازنده پیشفرض استفاده می کند مقادیری اختصاص داده شود تا این شی نیز با فراخوانی متد ShowInformation()آنها را نمایش دهد. حال به شی دوم که از سازنده دارای پارامتر استفاده می کند توجه کنید، مشاهده می کنید که با فراخوانی متد ShowInformation()همه چیز همانطور که انتظار می رود اجرا می شود. این بدین دلیل است که شما هنگام تعریف نمونه و از قبل مقادیری به هر یک از فیلدها اختصاص داده اید بنابراین آنها نیاز به مقدار دهی مجدد ندارند مگر اینکه شما بخواهید این مقادیر را اصلاح کنید.

اختصاص مقادیر پیشفرض به سازنده پیشفرض

در مثالهای قبلی یک سازنده پیشفرض با بدنه خالی نشان داده شد. شما می توانید به بدنه این سازنده پیشفرض کدهایی اضافه کنید. همچنین می توانید مقادیر پیشفرضی به فیلدهای آن اختصاص دهید.

```
public Person()
{
   this.name = "No Name";
   this.age = 0;
   this.height = 0;
}
```

همانطور که در مثال بالا می بینید سازنده پیشفرض ما چیزی برای اجرا دارد. اگر نمونه ای ایجاد کنیم که از این سازنده پیشفرض استفاده کند ، نمونه ایجاد شده مقادیر پیشفرض سازنده پیشفرض را نشان می دهد.

```
Person person1 = new Person();

person1.ShowInformation();
Name: No Name
Age: 0 years old
Height: 0cm
```

استفاده از کلمه کلیدیthis

راهی دیگر برای ایجاد مقادیر پیشفرض استفاده از کلمه کلیدی thisاست. مثال زیر اصلاح شده مثال قبل است و نحوه استفاده از 4 سازنده با تعداد پارامترهای مختلف را نشان می دهد.

```
1:  package myfirstprogram;
 2:
 3:  import java.text.MessageFormat;
 4:
 5:  class Person
 6:  {
 7:      public String name;
 8:      public int age;
 9:      public double height;
10:
11:      public Person()
12:      {
13:          this.name = "No Name";
14:          this.age = 0;
15:          this.height = 0;
16:      }
17:
18:      public Person(String n)
19:      {
20:          this.name = n;
21:      }
22:
23:      public Person(String n, int a)
24:      {
25:          this.name = n;
26:          this.age = a;
27:      }
28:
29:      public Person(String n, int a, double h)
30:      {
31:          this.name = n;
32:          this.age = a;
33:          this.height = h;
34:      }
35:
36:      public void ShowInformation()
37:      {
38:          System.out.println(MessageFormat.format("Name: {0}"
39:          System.out.println(MessageFormat.format("Age: {0} y
40:          System.out.println(MessageFormat.format("Height: {0
```

```
41:     }
42: }
43:
44: public class MyFirstProgram
45: {
46:     public static void main(String[] args)
47:     {
48:         Person firstPerson  = new Person();
49:         Person secondPerson = new Person("Jack");
50:         Person thirdPerson  = new Person("Mike", 23);
51:         Person fourthPerson = new Person("Chris", 18, 152);
52:
53:         firstPerson.ShowInformation();
54:         secondPerson.ShowInformation();
55:         thirdPerson.ShowInformation();
56:         fourthPerson.ShowInformation();
57:     }
58: }
```

```
Name: No Name
Age: 0 years old
Height: 0cm

Name: Jack
Age: 0 years old
Height: 0cm

Name: Mike
Age: 23 years old
Height: 0cm

Name: Chris
Age: 18 years old
Height: 152cm
```

ما چهار سازنده بری اصلاح کلاسمان تعریف کرده ایم(خطوط 11، 18، 25، 32) .شما می توانید تعداد زیادی سازنده برای مواقع لزوم در کلاس داشته باشید .اولین سازنده یک سازنده پیشفرض است. دومین سازنده یک پارامتر از نوع رشته دریافت می کند. سومین سازنده دو پارامتر و چهارمین سازنده سه پارامتر می گیرد. به چارمین سازنده در خطوط 32-37 توجه کنید. سه سازنده دیگر به این سازنده وابسته هستند. در خطوط 11-16 یک سازنده پیشفرض بدون پارامتر تعریف شده است. به کلمه کلیدی this توجه کنید. این کلمه کلیدی به شما اجازه می دهد که یک سازنده دیگر موجود در داخل کلاس را فراخوانی کنید .مقادیر پیشفرضی به فیلدها از طریق سازنده پیشفرض اختصاص می دهیم. چون ما سه مقدار پیشفرض برای فیلدها بعد از کلمه کلیدی this سازنده پیشفرض (خطوط 13-15) در نظر گرفته ایم، در نتیجه سازنده ای که دارای سه پارامتر است (چهارمین سازنده) فراخوانی شده و سه آرگومان به پارامترهای آن ارسال می شود .کدهای داخل بدنه چهارمین سازنده اجرا می شوند و مقادیر پارامترهای آن به هر یک از اعضای داده ای یه همان فیلدهای تعریف شده در خطوط 7-9 اختصاص داده می شود. اگر کدی در داخل بدنه سازنده پیشفرض بنویسیم قبل از بقیه کدها اجرا می شود. دومین سازنده (خطوط 18-23) به یک آرگومان نیاز دارد که همان فیلد name کلاس Person است. وقتی این پارامتر با یک مقدار رشته ای پر شد، سپس به پارامترهای سازنده چهارم ارسال شده و در کنار دو مقدار پیشفرض دیگر (0 برای age و 0 برای height) قرار می گیرد. در خط 25-30 سومین سازنده

تعریف شده است که بسیار شبیه دومین سازنده است با این تفاوت که دو پارامتر دارد. مقدار دو پارامتر سومین سازنده به اضافه ی یک مقدار پیشفرض صفر برای سومین آرگومان ، به چهارمین سازنده با استفاده از کلمه کلیدی this ارسال می شود.

```
Person firstPerson  = new Person();
Person secondPerson = new Person("Jack");
Person thirdPerson  = new Person("Mike", 23);
Person fourthPerson = new Person("Chris", 18, 152);
```

همانطور که مشاهده می کنید با ایجاد چندین سازنده برای یک کلاس، چندین راه برای ایجاد یک شی بر اساس داده هایی که نیاز داریم به وجود می آید. در مثال بالا 4 نمونه از کلاس Personایجاد کرده ایم و چهار تغییر در سازنده آن به وجود آورده ایم. سپس مقادیر مربوط به فیلدهای هر نمونه را نمایش می دهیم. یکی از موراد استفاده از کلمه کلیدی thisبه صورت زیر است. فرض کنید نام پارامترهای متد کلاس شما یا سازنده، شبیه نام یکی از فیلدها باشد.

```
public Person(string name, int age, double height)
{
   name = name;
   age = age;
   height = height;
}
```

این نوع کدنویسی ابهام بر انگیز است و کامپایلر نمی تواند متغیر را تشخیص داده و مقداری به آن اختصاص دهد. اینجاست که از کلمه کلیدی thisاستفاده می کنیم.

```
public Person(string name, int age, double height)
{
   this.name = name;
   this.age = age;
   this.height = height;
}
```

قبل از هر فیلدی کلمه کلیدی thisرا می نویسیم و نشان می دهیم که این همان چیزی است که می خواهیم به آن مقداری اختصاص دهیم .کلمه کلیدی thisارجاع یک شی به خودش را نشان می دهد.

سطح دسترسی

سطح دسترسی مشخص می کند که متدها یک کلاس یا اعضای داده ای در چه جای برنامه قابل دسترسی هستند. در این درس می خواهیم به سطح دسترسی privateو public نگاهی بیندازیم. سطح دسترسی public زمانی مورد استفاده قرار می گیرد که شما بخواهید به یک متد یا فیلد در خارج از کلاس و حتی پروژه دسترسی یابید. به عنوان مثال به کد زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  import java.text.MessageFormat;
 4:
 5:  class Test
```

```
 6: {
 7:     public int number;
 8: }
 9:
10: public class MyFirstProgram
11: {
12:     public static void main(String[] args)
13:     {
14:        Test x = new Test();
15:
16:        x.number = 10;
17:     }
18: }
```

در این مثال یک کلاس به نام Test تعریف کرده ایم(خطوط 8-5). سپس یک فیلد یا عضو داده ای به صورت public در داخل کلاس Test تعریف می کنیم (خط 7) .با تعریف این عضو به صورت public می توانیم آن را در خارج از کلاس Test و در داخل متد main() کلاس MyFirstProgram مقدار دهی کنیم. حال سطح دسترسی publicرا به private تغییر می دهیم:

```
 1: package myfirstprogram;
 2:
 3: import java.text.MessageFormat;
 4:
 5: class Test
 6: {
 7:     private int number;
 8: }
 9:
10: public class MyFirstProgram
11: {
12:     public static void main(String[] args)
13:     {
14:        Test x = new Test();
15:
16:         x.number = 10;
17:     }
18: }
```

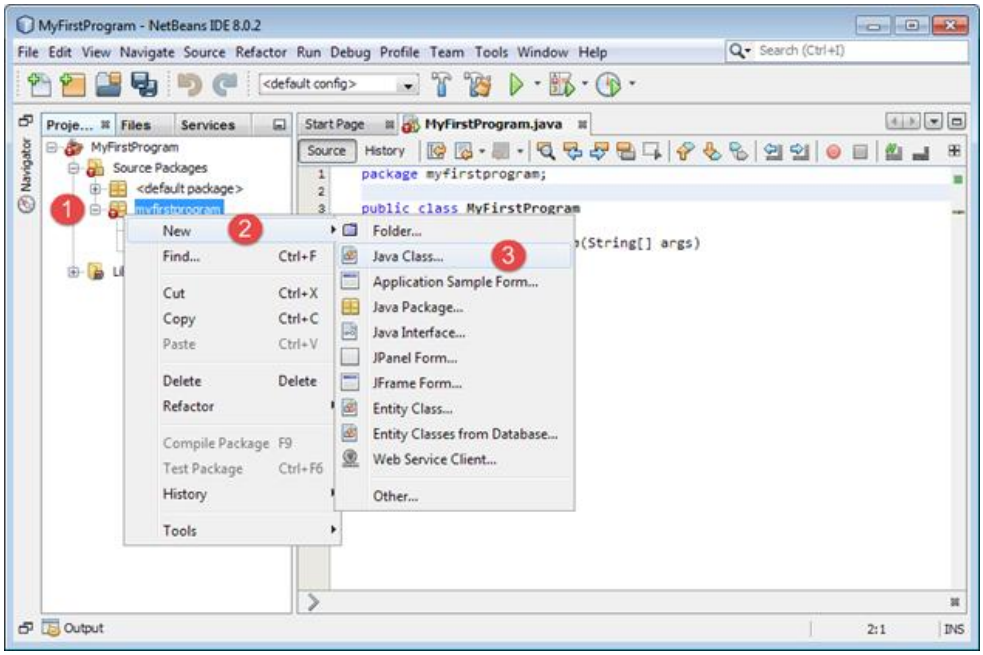
همانطور که در مثال بالا مشاهده می کنید این بار از کلمه private در تعریف فیلد number استفاده کرده ایم (خط 7). وقتی که برنامه را کامپایل می کنیم با خطا مواجه می شویم چون number در داخل کلاس MyFirstProgramو یا هر کلاس دیگر قابل دسترسی نیست.

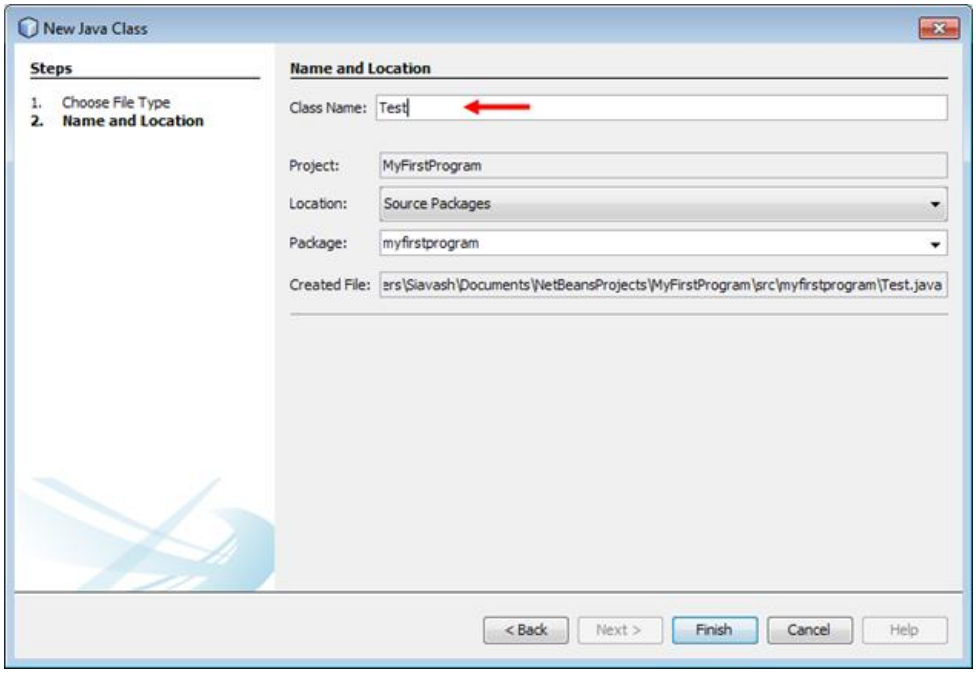
نکته دیگر اینکه اگر شما برای یک کلاس سطح دسترسی تعریف نکنید آن کلاس دارای سطح دسترسی داخلی (default modifier) می شود به این معنی که فقط کلاس های داخل پروژه ای که با آن کار می کنید و می توانند به آن کلاس دسترسی یابند. اگر یک کلاس را به صورت public و اعضای آن را به صورت private تعریف کنیم ، آنگاه می توان یک نمونه از کلاس را در داخل کلاس های دیگر ایجاد کرد ولی اعضای آن قابل دسترسی نیستند. اعضای داده ای private فقط به وسیله متد داخل کلاس Test قابل دسترسی هستند.

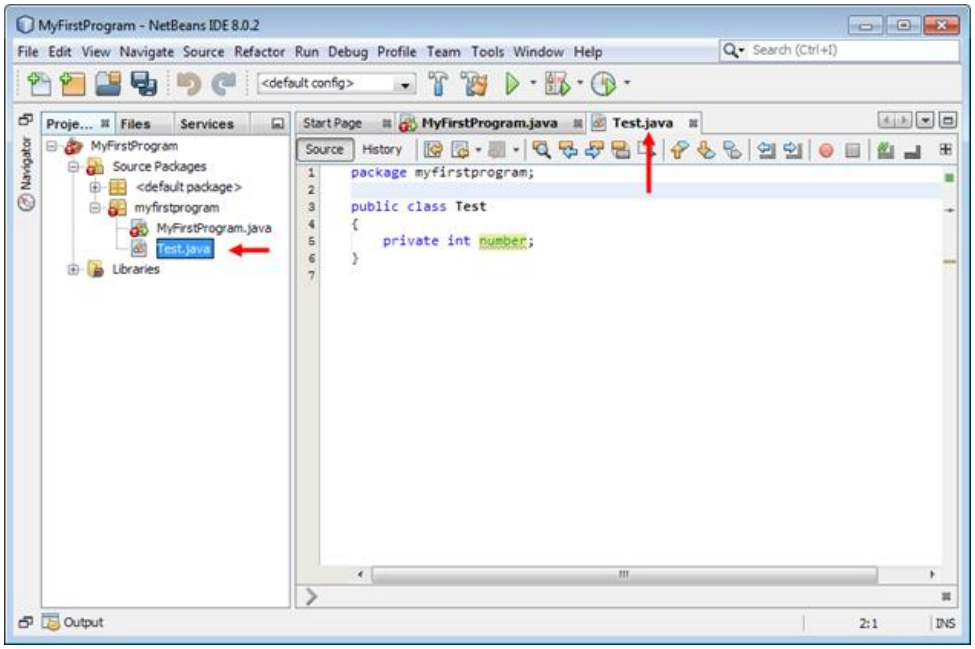
```
1: package myfirstprogram;
2:
3: import java.text.MessageFormat;
```

```
 4:
 5:  public class Test
 6:  {
 7:      private int number;
 8:  }
 9:
10:  public class MyFirstProgram
11:  {
12:      public static void main(String[] args)
13:      {
14:         Test x = new Test();
15:
16:          x.number = 10;
17:      }
18:  }
```

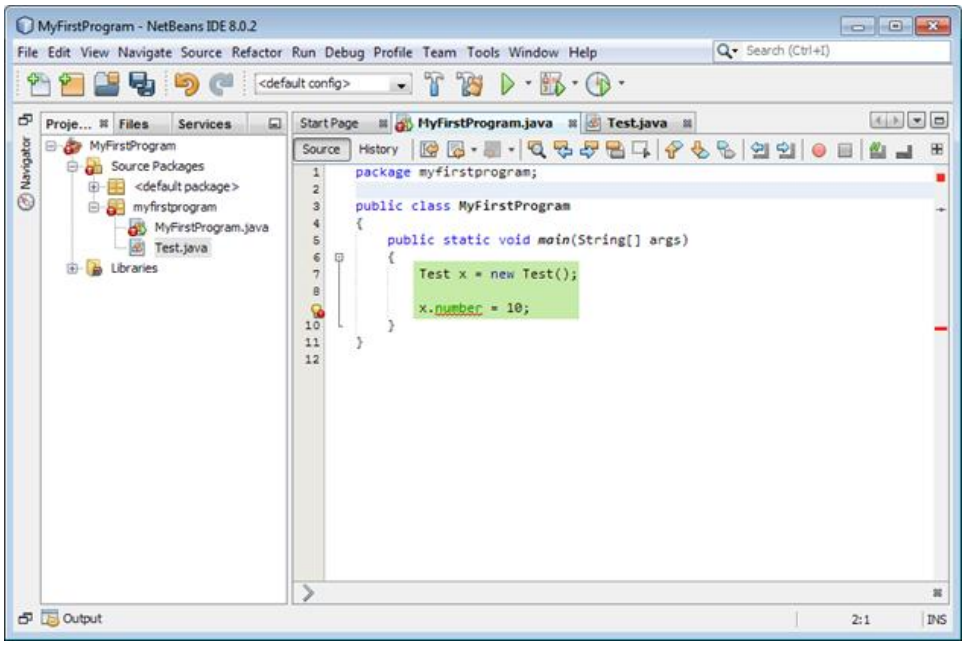
کد بالا کامپایل نمی شود، چون جاوا به شما اجازه استفاده از دو کلاس عمومی در یک فایل را نمی دهد. برای حل این مشکل یا باید کلمه public را مثلا از ابتدای کلاس Test حذف کنیم و یا اینکه چون ما می خواهیم حتما کلاس به صورت public تعریف شود و اعضای آن به صورت private آن را باید در یک فایل جدا به صورت زیر تعریف و با پسوند .java تعریف کنیم:







حال اگر از کلاس بالا در کلاس MyFirstProgram نمونه ای ایجاد کرده و بخواهیم از فیلد number استفاده کنیم با خطا مواجه می شویم:



سطوح دسترسی دیگری هم در جاوا وجود دارد که بعد از مبحث وراثت در درسهای آینده در مورد آنها توضیح خواهیم داد.

کپسوله سازی(Encapsulation)

کپسوله سازی (تلفیق داده ها با یکدیگر) یا مخفی کردن اطلاعات فرایندی است که طی آن اطلاعات حساس یک موضوع از دید کاربر مخفی می شود و فقط اطلاعاتی که لازم باشد برای او نشان داده می شود.

وقتی که یک کلاس تعریف می کنیم معمولا تعدادی اعضای داده ای (فیلد) برای ذخیره مقادیر مربوط به شی نیز تعریف می کنیم. برخی از این اعضای داده ای توسط خود کلاس برای عملکرد متدها و برخی دیگر از آنها به عنوان یک متغیر موقت به کار می روند. به این اعضای داده ای، اعضای مفید نیز می گویند چون فقط در عملکرد متدها تاثیر دارند و مانند یک داده قابل رویت کلاس نیستند .لازم نیست که کاربر به تمام اعضای داده ای یا متدهای کلاس دسترسی داشته باشد. اینکه فیلدها را طوری تعریف کنیم که در خارج از کلاس قابل دسترسی باشند بسیار خطرناک است چون ممکن است کاربر رفتار و نتیجه یک متد را تغییر دهد. به برنامه ساده زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  class Test
 4:  {
 5:      public int five = 5;
 6:
 7:      public int AddFive(int number)
 8:      {
```

```
 9:          number += five;
10:          return number;
11:      }
12:  }
13:
14:  public class MyFirstProgram
15:  {
16:      public static void main(String[] args)
17:      {
18:          Test x = new Test();
19:
20:          x.five = 10;
21:          System.out.println(x.AddFive(100));
22:      }
23:  }
110
```

متد داخل کلاس Testبه نام AddFiveدارای هدف ساده ای است و آن اضافه کردن مقدار 5 به هر عدد می باشد. در داخل متد main()یک نمونه از کلاس Test ایجاد کرده ایم و مقدار فیلد آن را از 5 به 10 تغییر می دهیم (در اصل نباید تغییر کند چون ما از برنامه خواسته ایم هر عدد را با 5 جمع کند ولی کاربر به راحتی آن را به 10 تغییر می دهد). همچنین متد AddFive()را فراخوانی و مقدار 100 را به آن ارسال می کنیم. مشاهده می کنید که قابلیت متد AddFive() به خوبی تغییر می کند و شما نتیجه متفاوتی مشاهده می کنید. اینجاست که اهمیت کپسوله سازی مشخص می شود. اینکه ما در درسهای قبلی فیلدها را به صورت public تعریف کردیم و به کاربر اجازه دادیم که در خارج از کلاس به آنها دسترسی داشته باشد کار اشتباهی بود. فیلدها باید همیشه به صورت privateتعریف شوند.

خواص(Properties)

property (خصوصیت) استانداردی در جاوا برای دسترسی به اعضای داده ای با سطح دسترسی private در داخل یک کلاس می باشد. هر property دارای دو بخش می باشد، یک بخش جهت مقدار دهی (بلوک set) و یک بخش برای دسترسی به مقدار (بلوک get) یک داده private می باشد property .ها باید به صورت publicتعریف شوند تا در کلاسهای دیگر نیز قابل دسترسی می باشند. در مثال زیر نحوه تعریف و استفاده از property آمده است:

```
1:  package myfirstprogram;
 2:
 3:  import java.text.MessageFormat;
 4:
 5:  class Person
 6:  {
 7:      private String name;
 8:      private int age;
 9:      private double height;
10:
11:      public void setName(String name)
12:      {
13:          this.name = name;
14:      }
15:
```

```
16:     public String getName()
17:     {
18:         return name;
19:     }
20:
21:     public void setAge(int age)
22:     {
23:         this.age = age;
24:     }
25:
26:     public int getAge()
27:     {
28:         return age;
29:     }
30:
31:     public void setHeight(double height)
32:     {
33:         this.height = height;
34:     }
35:
36:     public double getHeight()
37:     {
38:         return height;
39:     }
40:
41:
42:     public Person(String name, int age, double height)
43:     {
44:         this.name = name;
45:         this.age = age;
46:         this.height = height;
47:     }
48:
49: }
50:
51: public class MyFirstProgram
52: {
53:     public static void main(String[] args)
54:     {
55:         Person person1 = new Person("Jack", 21, 160);
56:         Person person2 = new Person("Mike", 23, 158);
57:
58:         System.out.println(MessageFormat.format("Name: {0}", person1.getName()));
59:         System.out.println(MessageFormat.format("Age: {0} years old",
60: person1.getAge()));
61:         System.out.println(MessageFormat.format("Height: {0}cm",
62: person1.getHeight()));
63:
64:         System.out.println(); //Seperator
65:
66:         System.out.println(MessageFormat.format("Name: {0}", person2.getName()));
67:         System.out.println(MessageFormat.format("Age: {0} years old",
68: person2.getAge()));
69:         System.out.println(MessageFormat.format("Height: {0}cm",
70: person2.getHeight()));
71:
72:
```

```
73:         person1.setName("Frank");
74:         person1.setAge(19);
75:         person1.setHeight(162);
76:
77:         person2.setName("Ronald");
78:         person2.setAge(25);
79:         person2.setHeight(174);
80:
81:         System.out.println(); //Seperator
82:
83:         System.out.println(MessageFormat.format("Name: {0}", person1.getName()));
84:         System.out.println(MessageFormat.format("Age: {0} years old",
85: person1.getAge()));
86:         System.out.println(MessageFormat.format("Height: {0}cm",
87: person1.getHeight()));
88:
89:         System.out.println(); //Seperator
90:
91:         System.out.println(MessageFormat.format("Name: {0}", person2.getName()));
92:         System.out.println(MessageFormat.format("Age: {0} years old",
93: person2.getAge()));
94:         System.out.println(MessageFormat.format("Height: {0}cm",
95: person2.getHeight()));
96:     }
97: }
```

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm

Name: Frank
Age: 19 years old
Height: 162cm

Name: Ronald
Age: 25 years old
Height: 174cm
```

در برنامه بالا نحوه استفاده از property آمده است.همانطور که مشاهده می کنید در این برنامه ما سه خصوصیت که هر کدام مربوط به اعضای داده ای هستند تعریف کرده ایم (سه فیلد با سطح دسترسیprivate).

```
private string name;
private int age;
private double height;
```

دسترسی به مقادیر این فیلدها فقط از طریق property های ارائه شده امکان پذیر است.

```
11: public void setName(String name)
12: {
```

```
13:     this.name = name;
14: }
15:
16: public String getName()
17: {
18:     return name;
19: }
20:
21: public void setAge(int age)
22: {
23:     this.age = age;
24: }
25:
26: public int getAge()
27: {
28:     return age;
29: }
30:
31: public void setHeight(double height)
32: {
33:     this.height = height;
34: }
35:
36: public double getHeight()
37: {
38:     return height;
39: }
```

وقتی یک خاصیت ایجاد می کنیم ، باید سطح دسترسی آن را public تعریف کرده و نوع داده ای را که بر می گرداند یا قبول می کند را مشخص کنیم. به این نکته توجه کنید که نام property ها همانند نام فیلدهای مربوطه می باشد با این تفاوت که حرف اول آنها بزرگ نوشته می شود. البته قبل از نام آنها کلمات setو get هم نوشته می شود. مثلا برای فیلد name (خط 7) دو متد با نامهای setName و getName (خطوط 19-11) ایجاد شده اند. البته یادآور می شویم که شباهت نام property ها و فیلدها اجبار نیست و یک قرارداد می باشد.

در داخل بدنه دو بخش می بینید ، یکی بخش set و دیگری بخش.get

- **بخش get ، که با کلمه کلیدی get نشان داده شده است به شما اجازه می دهد که یک مقدار را از فیلدها (اعضای داده ای) استخراج کنید.**
- **بخش set ، که با کلمه کلیدی set نشان داده شده است برای مقدار دهی به فیلدها (اعضای داده ای) به کار می رود.**

به عنوان مثال به عبارت زیر توجه کنید:

```
this.name = name;
```

این عبارت که در خط 13 کد بالا آمده است برای مقداردهی به فیلد name به کار رفته است this .به شئ جاری ایجاد شده از کلاس اشاره دارد، nameبعد آن همان فیلد تعریف شده در خط 7 کد ابتدای آموزش و name سمت راست علامت مساوی هم پارامتر تعریف شده در خط 11 است. برای دسترسی به یک خاصیت می توانید از علامت دات (.) استفاده کنید.

```
System.out.println(MessageFormat.format("Name: {0}", person1.getName()));
```

```
System.out.println(MessageFormat.format("Age: {0} years old", person1.getAge()));
System.out.println(MessageFormat.format("Height: {0}cm", person1.getHeight()));
```

فراخوانی یک خاصیت باعث اجرای کد داخل بدنه بلوک get آن می شود. سپس این بلوک مانند یک متد مقداری به فراخوان برگشت می دهد. مقدار دهی به یک property بسیار آسان است.

```
person1.setName("Frank");
person1.setAge(19);
person1.setHeight(162);
```

دستورات بالا بخش set مربوط به هر property را فراخوانی کرده و مقادیری به هر یک از فیلدها اختصاص می دهد. استفاده از propertyها کد نویسی را انعطاف پذیر می کند مخصوصا اگر بخواهید یک اعتبارسنجی برای اختصاص یک مقدار به فیلدها یا استخراج یک مقدار از آنها ایجاد کنید. مثلا شما می توانید یک محدودیت ایجاد کنید که فقط اعداد مثبت به فیلد age (سن) اختصاص داده شود .می توانید با تغییر بخش set خاصیت Age این کار را انجام دهید:

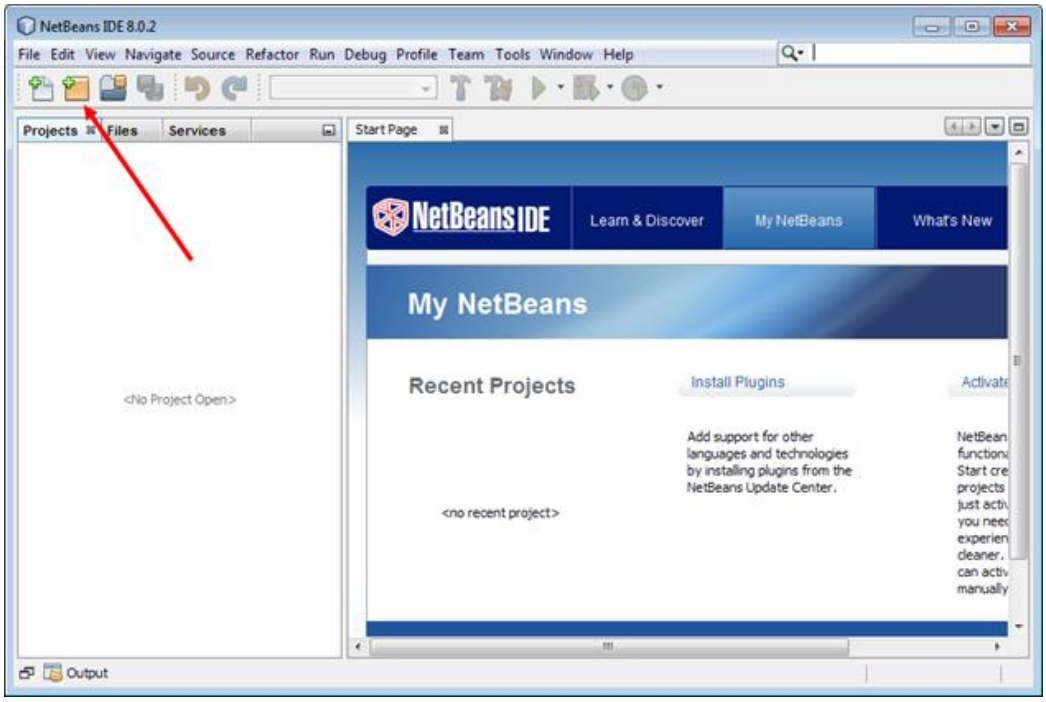
```
public void setAge(int age)
{
     if (age > 0)
     {
       this.age = age;
     }
     else
     {
       this.age = 0;
     }
}
```

حال اگر کاربر بخواهد یک مقدار منفی به فیلد age اختصاص دهد مقدار age صفر خواهد شد.

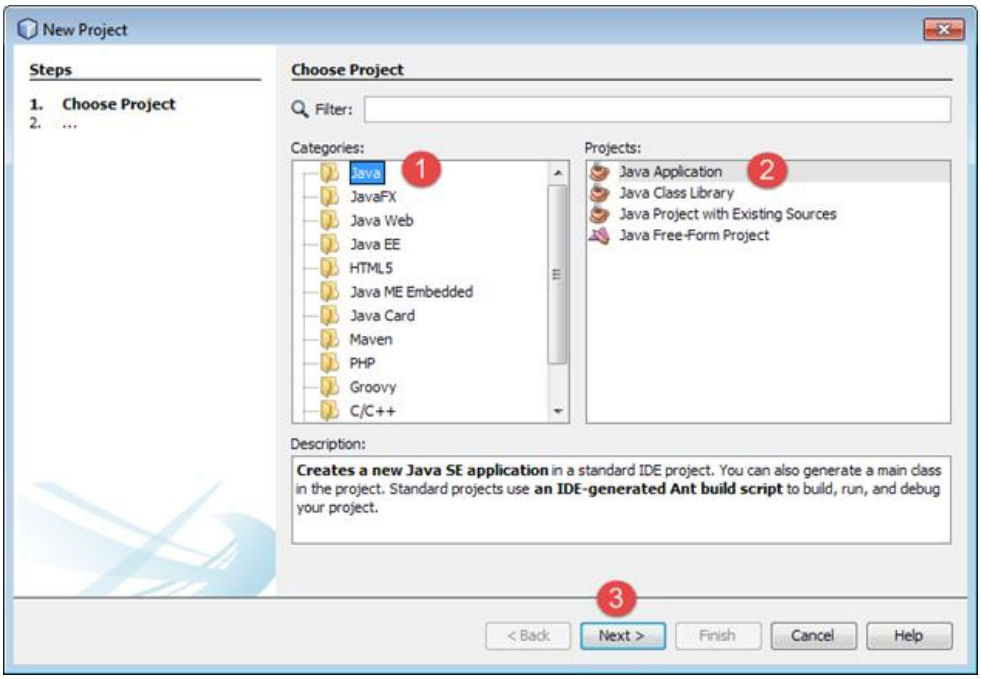
Package

Package (پکیج) راهی برای دسته بندی کدهای برنامه می باشد. هر چیز در جاوا حداقل در یک Package قرار دارد. وقتی برای یک کلاس اسمی انتخاب می کنید ممکن است برنامه نویسان دیگر به صورت اتفاقی اسمی شبیه به آن برای کلاسشان انتخاب کنند. وقتی شما از آن کلاسها در برنامه تان استفاده کنید از آنجاییکه از کلاسهای همنام استفاده می کنید در برنامه ممکن است خطا به وجود آید.

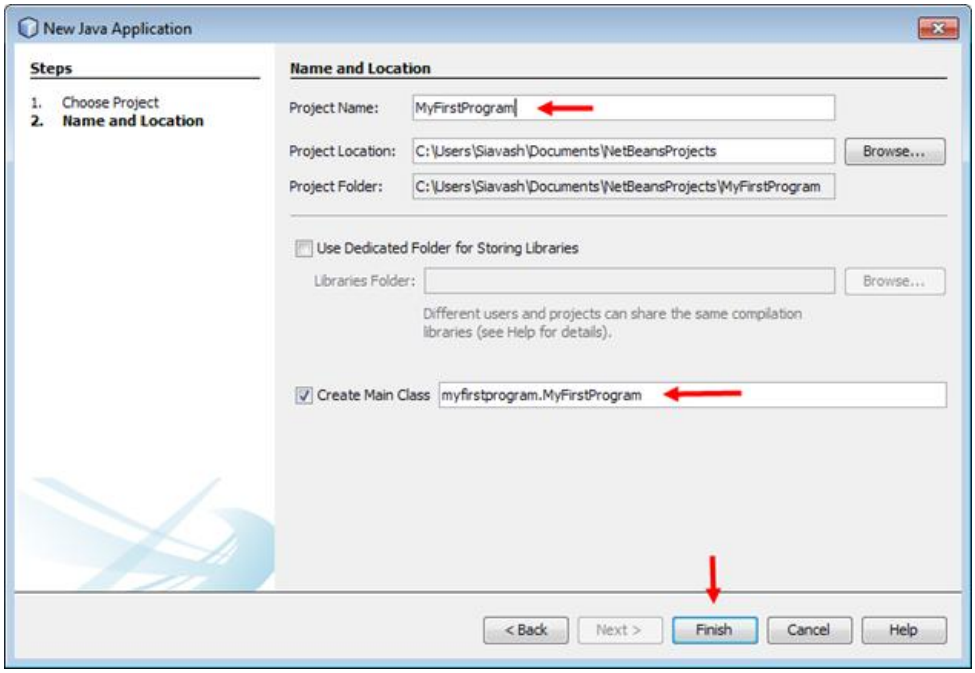
Packageاز وقوع این خطاها جلوگیری کرده یا آنها را کاهش می دهند .تاکنون و در درسهای قبلی ما فقط با یک پکیج آشنا شده ایم و آن پکیجی به نام myfirstprogramبود که کلاسی به همین نام (MyFirstProgram) و متد main() را در خود داشت. هنگامی که یک پروژه جدید ایجاد کنید به صورت پیشفرض یک فضای نام برای شما ایجاد خواهد شد که نام آن شبیه به نام پروژه تان می باشد. در این درس به شما نشان می دهیم که چگونه کلاسهایتان در در کدهای جداگانه بنویسید و سپس از آنها در فایلهای جدا استفاده کنید. برنامه NetBeansرا اجرا و یک پروژه جدید ایجاد کنید:



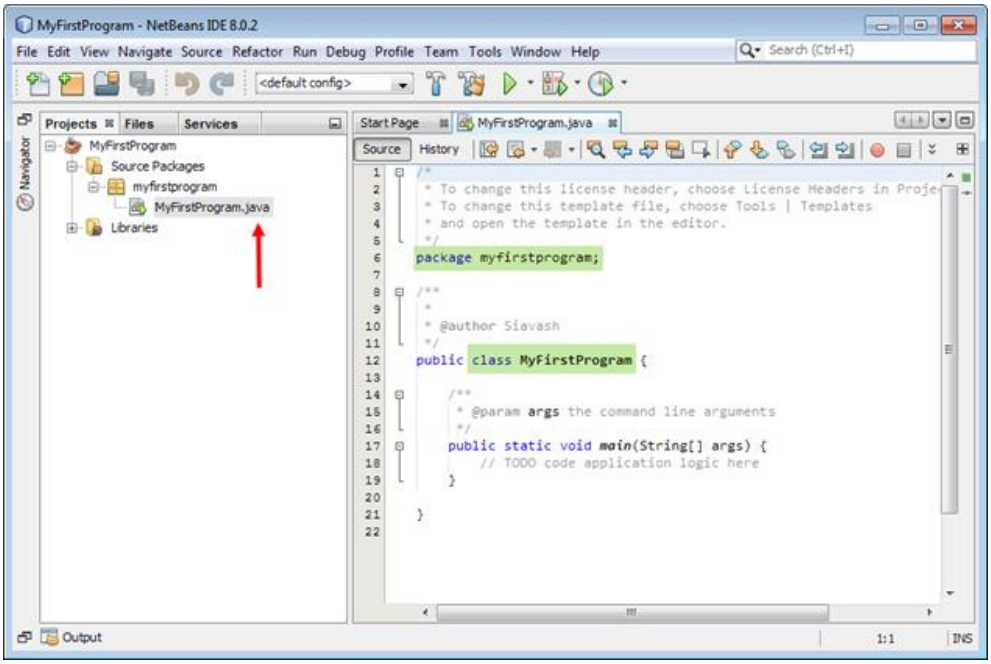
بعد از کلیک بر روی گزینه New Project و یا زدن دکمه های ترکیبی Ctrl+Shfit+Nپنجره ای به صورت زیر به نمایش در می آید که بر طبق شکل گزینه ها را نتخاب کنید:



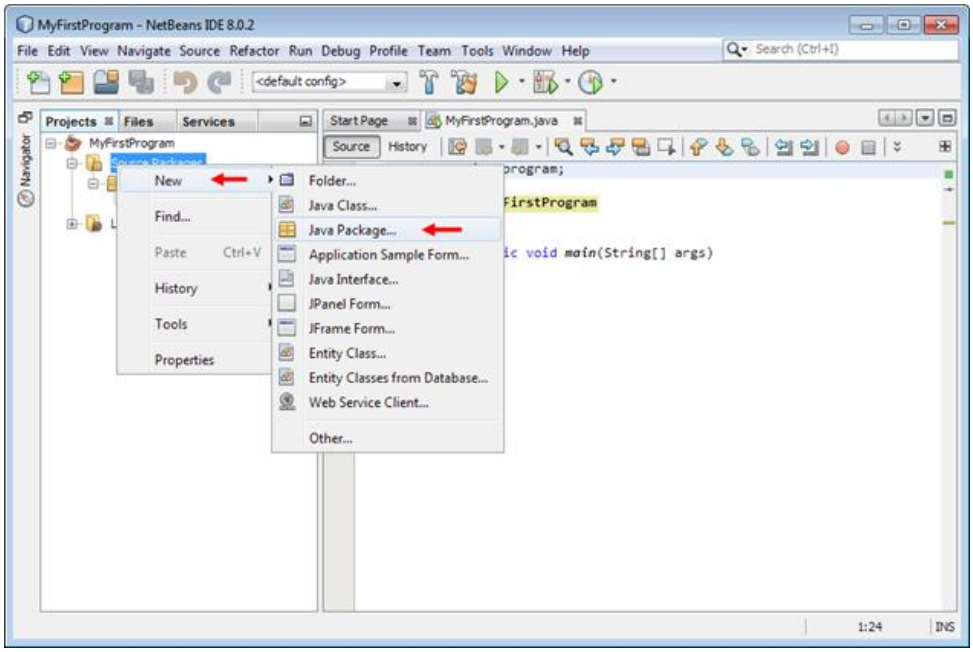
بعد از طی مراحل شکل بالا صفحه زیر به نمایش در می آید. در این صفحه و در قسمت Project Name نام پروژه تان را انتخاب کنید. مشاهده می کنید که در کادری پایین تر از آن بسته به نام پروژه یک Package به همراه نام کلاس ایجاد می کند:

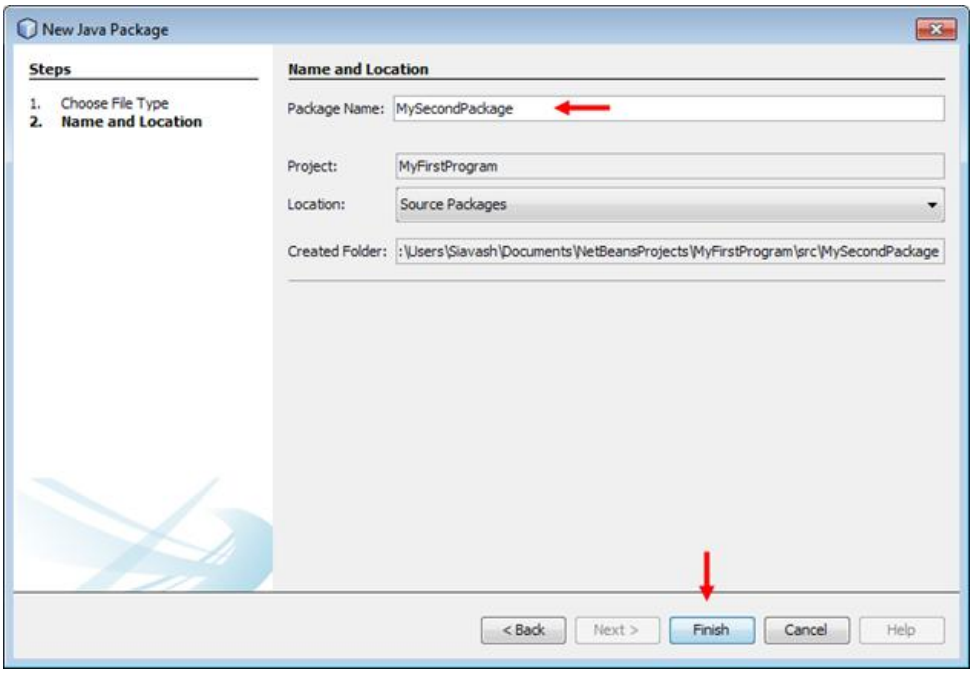


بعد از زدن دکمه finish در شکل بالا ، یک Package یک class به صورت زیر ایجاد می شود:

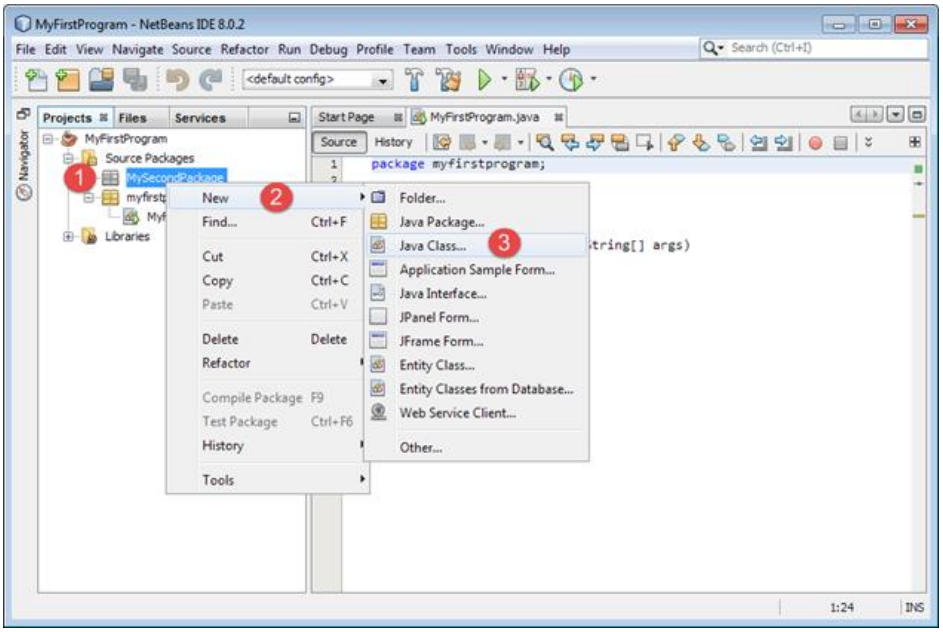


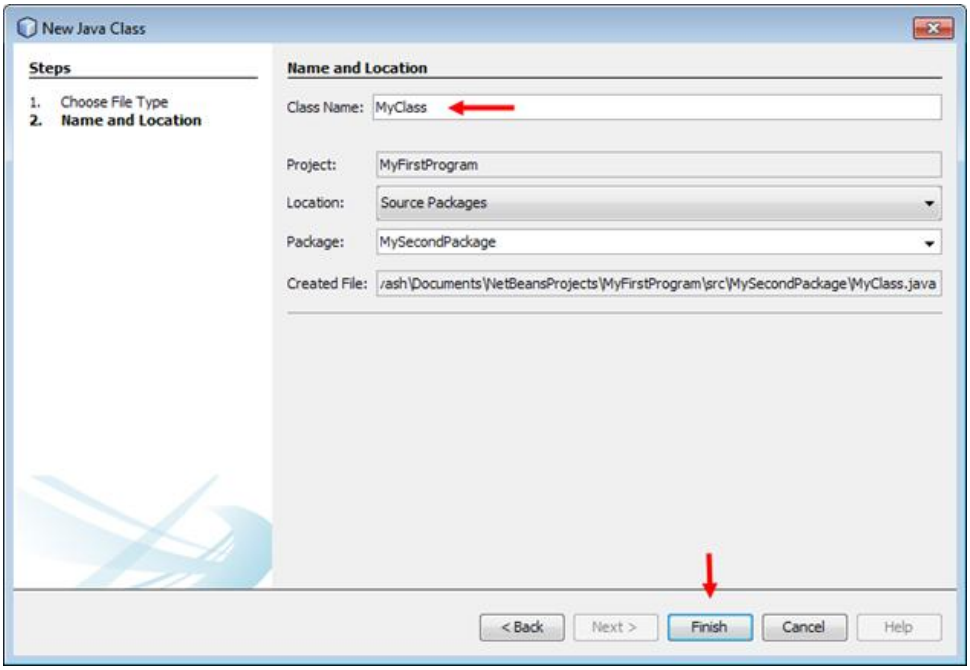
توجه کنید که پسوند کلاس در جاوا به صورت .java می باشد. پس تا اینجا ما یک Packageو یک کلاس و یک متد داریم. حال می خواهیم یک پکیج دیگر ایجاد و از کلاس ها و متدهای آن در داخل این پکیج استفاده کنیم. برای این کار بر روی گزینه Source Packageمانند شکل زیر راست کلیک کرده و یک پکیج جدید به نام MySecondPackageایجاد می کنیم:



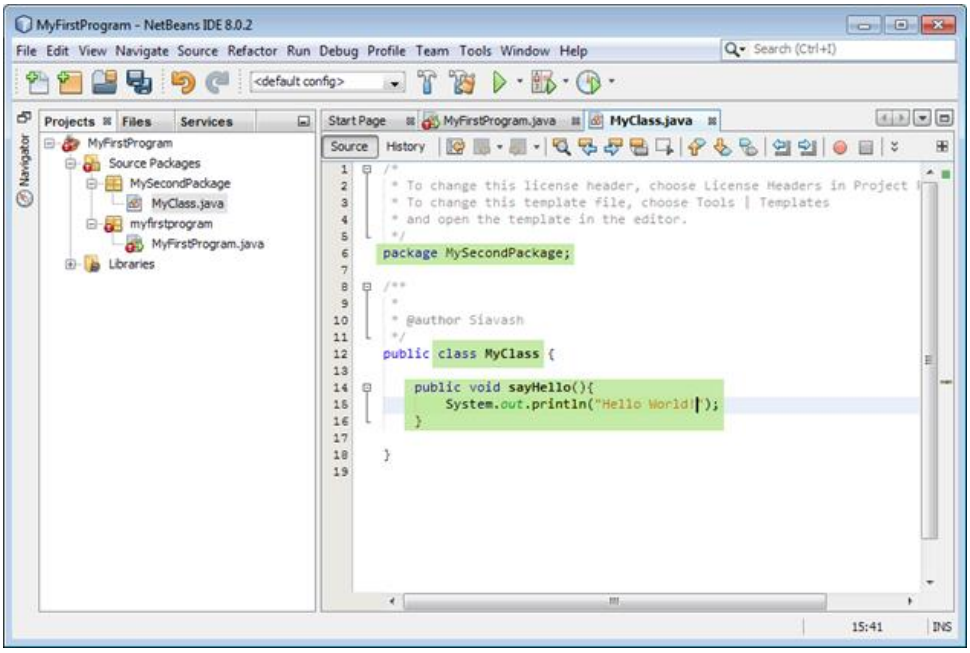


بعد از ایجاد این Package یک کلاس به نام MyClass به آن، به روش زیر اضافه می کنیم:

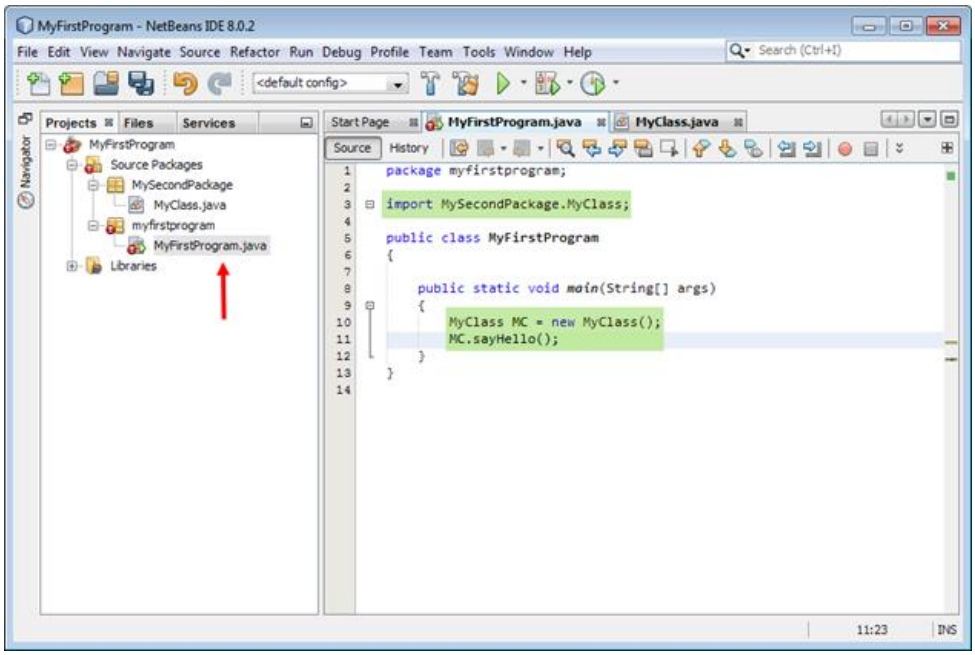




بعد از اضافه کردن کلاس یک متد به نام sayHello() به شکل زیر به کلاس اضافه نمایید:



حال فرض کنید که می خواهید از این کلاس و متد در کلاس MyFirstProgram استفاده کنید. برای این کار بر روی کلاس مذکور دو بار کلیک کرده و سپس مانند شکل زیر و با استفاده از کلمه import کلاس را در Package اولی وارد نمایید و سپس با ایجاد یک شی از کلاس متد مربوط به آن را فراخوانی کنید:



پس در کل می توان نتیجه گرفت که با استفاده از کلمه کلیدی import می توان همه محتویات یک پکیج را در داخل پکیج دیگر وارد کرد. اگر قصد وارد کردن فقط یک کلاس از یک پکیج را در داخل پکیج دیگر داشته باشیم به صورت زیر عمل می کنیم:

```
import Package.Class;
```

و اگر بخواهیم تمامی کلاس های یک Package را وارد Package دیگر کنیم به صورت زیر عمل می نماییم:

```
import Package.*;
```

اگر از کلمه import استفاده نکنیم مجبوریم که در ابتدای نام هر کلاس Package مربوط به آن را به صورت زیر ذکر کنیم:

```
MySecondPackage.MyClass MC = new MySecondPackage.MyClass();
```

شما محدود به دسته بندی کدهای کلاستان در داخل یک Package نیستید. می توانید یک Package تو در تو ایجاد کنید و کدهایتان را در درون آن بنویسید .برای دسترسی به کلاس Sample، مجبورید اول نام تمام Package هایی را که کلاس Sample در آنها قرار دارد بنویسید.

```
Package1.Package2.Sample
```

یا می توان از کلمه کلیدی importاستفاده کرد:

```
import Package1.Package2.Sample
```

وراثت

وراثت به یک کلاس اجازه می دهد که خصوصیات یا متدهایی را از کلاس دیگر به ارث برد. وراثت مانند رابطه پدر و پسری می ماند به طوریکه فرزند خصوصیاتی از قبیل قیافه و رفتار را از پدر خود به ارث برده باشد.

- **کلاس پایه یا کلاس والد کلاسی است که بقیه کلاسها از آن ارث می برند.**
- **کلاس مشتق یا کلاس فرزند کلاسی است که از کلاس پایه ارث بری می کند.**

همه متد و خصوصیات کلاس پایه می توانند در کلاس مشتق مورد استفاده قرار بگیرند به استثنای اعضا و متدهای با سطح دسترسی . privateهمه کلاس ها در جاوا از کلاس Object ارث بری می کنند. مفهوم اصلی وراثت در مثال زیر نشان داده شده است:

```
 1:  package myfirstprogram;
 2:
 3:  class Parent
 4:  {
 5:      private String message;
 6:
 7:      public void setMessage(String message)
 8:      {
 9:          this.message = message;
10:      }
11:
12:      public String getMessage()
13:      {
14:          return message;
15:      }
16:
17:      public void ShowMessage()
18:      {
19:          System.out.println(message);
20:      }
21:
22:      public Parent(String message)
23:      {
24:          this.message = message;
25:      }
26:  }
27:
28:  class Child extends Parent
29:  {
30:      public Child(String message)
31:      {
32:          super(message);
33:      }
```

```
34:  }
```

در این مثال دو کلاس با نامهای Parent و Childتعریف شده است. در این مثال یک فیلد را با سطح دسترسی private (خط 5)و خاصیت مربوط به آن را با سطح دسترسی public (خط 7–15) تعریف کرده ایم. سپس یک متد را برای نمایش پیام تعریف کرده ایم. یک سازنده در کلاس Parentتعریف شده است که یک آرگومان از نوع رشته قبول می کند و یک پیغام نمایش می دهد(خطوط 22–25). حال به کلاس Childتوجه کنید (خط 28–34) .این کلاس تمام متدها و خاصیت های کلاس Parentرا به ارث برده است.

نحوه ارث بری یک کلاس به صورت زیر است:

```
class DerivedClass extends BaseClass
```

براحتی می توان با قرار دادن کلمه کلیدی extends بعد از نام کلاس و سپس نوشتن نام کلاسی که از آن ارث بری می شود (کلاس پایه) این کار را انجام داد. در داخل کلاس Childهم یک سازنده ساده وجود دارد که یک آرگومان رشته ای قبول می کند .وقتی از وراثت در کلاسها استفاده می کنیم، هم سازنده کلاس مشتق و هم سازنده پیشفرض کلاس پایه هر دو اجرا می شوند .سازنده پیشفرض یک سازنده بدون پارامتر است. اگر برای یک کلاس سازنده ای تعریف نکنیم کامپایلر به صورت خودکار یک سازنده برای آن ایجاد می کند.

اگر هنگام صدا زدن سازنده کلاس مشتق بخواهیم سازنده کلاس پایه را صدا بزنیم باید از کلمه کلیدی superاستفاده کنیم. کلمه کلیدی superیک سازنده از کلاس پایه را صدا می زند .

در مثال بالا به وسیله تامین مقدار پارامتر messageسازنده کلاس مشتق و ارسال آن به داخل پرانتز کلمه کلیدی super، سازنده معادل آن در کلاس پایه فراخوانی شده و مقدار messageرا به آن ارسال می کند .سازنده کلاس Parentهم این مقدار (مقدار message) را در یک عضو داده ای (فیلد) private قرار می دهد. می توانید کدهایی را به داخل بدنه سازنده Child اضافه کنید تا بعد از سا زنده Parentاجرا شوند. اگر از کلمه کلیدی superاستفاده نشود به جای کلاس پایه سازنده پیشفرض فراخوانی می شود. اجازه بدهید که اشیایی از کلاسهای Parent و Childبسازیم تا نشان دهیم که چگونه کلاس Childمتدها و خواص کلاس Parentرا به ارث می برد.

```
 1:  public class MyFirstProgram
 2:  {
 3:
 4:      public static void main(String[] args)
 5:      {
 6:          Parent myParent = new Parent("Message from parent.");
 7:          Child myChild = new Child("Message from child.");
 8:
 9:          myParent.ShowMessage();
10:
11:          myChild.ShowMessage();
12:
13:          myParent.setMessage("Modified message of the parent.");
14:          myParent.ShowMessage();
15:
16:          myChild.setMessage("Modified message of the child.");
17:          myChild.ShowMessage();
18:
```

```
19:         //myChild.message; ERROR: can't access private members of base class
20:     }
21: }
```

```
Message from parent.
Message from child.
Modified message of the parent.
Modified message of the child.
```

هر دو شی را با ستفاده از سازنده های مربوط به خودشان مقدار دهی می کنیم.(خطوط 7-6) سپس با استفاده از ارث بری و از طریق شی Childبه اعضا و متدهای کلاس Parentدسترسی می یابیم. حتی اگر کلاس Childاز کلاس Parentارث ببرد باز هم اعضای با سطح دسترسی privateدر کلاس Childقابل دسترسی نیستند(خط 18). سطح دسترسی Protectکه در درس آینده توضیح داده خواهد شد به شما اجازه دسترسی به اعضا و متدهای کلاس پایه را می دهد. به نکته دیگر توجه کنید. اگر کلاس دیگری بخواهد از کلاس Childارث بری کند، باز هم تمام متدها و خواص کلاس Childکه از کلاس Parentبه ارث برده است را به ارث می برد.

```
class GrandChild extends Child
{
   //Empty Body
}
```

این کلاس هیچ چیزی در داخل بدنه ندارد. وقتی کلاس GrandChildرا ایجاد می کنید و یک خاصیت از کلاس Parentرا فراخوانی می کنید با خطا مواجه می شوید. چون هیچ سازنده ای که یک آرگومان رشته ای قبول کند در داخل بدنه GrandChildتعریف نشده است بنابراین شما می توانید فقط از سازنده پیشفرض یا بدون پارامتر استفاده کنید.

```
GrandChild myGrandChild = new GrandChild();

myGrandChild.setMessage("Hello my grandchild!");
myGrandChild.ShowMessage();
```

وقتی یک کلاس ایجاد می کنیم و سازنده GrandChildرا فراخوانی می کنیم ابتدا سازنده کلاس Parentفراخوانی می شود و سپس سازنده Childو در نهایت سازنده GrandChildاجرا می شود . برنامه زیر ترتیب اجرای سازنده ها را نشان می دهد. دوباره کلاسها را برای خوانایی بیشتر در داخل کدهای جدا قرار می دهیم.

```
1:  package myfirstprogram;
2:
3:  class Parent
4:  {
5:      public Parent()
6:      {
7:         System.out.println("Parent constructor was called!");
8:      }
9:  }
10:
11: class Child extends Parent
12: {
13:     public Child()
14:     {
15:        System.out.println("Child constructor was called!");
16:     }
```

```
17:  }
18:
19:  class GrandChild extends Child
20:  {
21:      public GrandChild()
22:      {
23:         System.out.println("GrandChild constructor was called!");
24:      }
25:  }
26:
27:  public class MyFirstProgram
28:  {
29:
30:      public static void main(String[] args)
31:      {
32:        GrandChild myGrandChild = new GrandChild();
33:      }
34:  }
```

```
Parent constructor was called!
Child constructor was called!
GrandChild constructor was called!
```

سطح دسترسی Protect

سطح دسترسی protectاجازه می دهد که اعضای کلاس، فقط در کلاسهای مشتق شده از کلاس پایه قابل دسترسی باشند. بدیهی است که خود کلاس پایه هم می تواند به این اعضا دسترسی داشته باشد. کلاسهایی که از کلاس پایه ارث بری نکرده اند نمی توانند به اعضای با سطح دسترسی protectیابند. در مورد سطوح دسترسی public و private قبلا توضیح دادیم. در جدول زیر نحوه دسترسی به سه سطح ذکر شده نشان داده شده است:

protected	private	public	قابل دسترسی در
true	**true**	**true**	داخل کلاس
false	**false**	**true**	خارج از کلاس
true	**false**	**true**	کلاس مشتق

مشاهده می کنید که publicبیشترین سطح دسترسی را داراست. صرف نظر از مکان، اعضای publicدر هر جا فراخوانی می شوند و قابل دسترسی هستند. اعضای privateفقط در داخل کلاسی که به آن تعلق دارند قابل دسترسی هستند. کد زیر رفتار اعضای دارای این سه سطح دسترسی را نشان می دهد:

```
1:  package myfirstprogram;
2:
3:   class Parent
4:   {
5:       protected int protectedMember = 10;
```

```
 6:      private   int privateMember = 10;
 7:      public    int publicMember = 10;
 8:  }
 9:
10:  class Child extends Parent
11:  {
12:      public Child()
13:      {
14:          protectedMember = 100;
15:          privateMember = 100;
16:          publicMember = 100;
17:      }
18:  }
19:
20:  public class MyFirstProgram
21:  {
22:      public static void main(String[] args)
23:      {
24:           Parent myParent = new Parent();
25:
26:           myParent.protectedMember = 100;
27:           myParent.privateMember = 100;
28:           myParent.publicMember = 100;
29:      }
30:  }
```

کدهایی که با خط قرمز نشان داده شده اند نشان دهنده وجود خطا هستند چون آنها اجازه دسترسی به فیلدهای protectedکلاس Parentرا ندارند. همانطور که در خط 15 مشاهده می کنید کلاس Child سعی می کند که به عضو private کلاس Parent دسترسی یابد. از آنجاییکه اعضای private در خارج از کلاس قابل دسترسی نیستند، حتی کلاس مشتق در خط 15 نیز ایجاد خطا می کند. اگر شما به خط 14 توجه کنید کلاس Child می تواند به عضو protect کلاس Parent دسترسی یابد چون کلاس Child از کلاس Parent مشتق شده است.

حال به خط 26 جاییکه می خواهیم در کلاس MyFirstProgram به فیلد protect کلاس Parent دسترسی یابیم نگاهی بیندازید. می بینید که برنامه پیغام خطا می دهد چون کلاس MyFirstProgram از کلاس Parent مشتق نشده است. همچنین کلاس MyFirstProgram به اعضای private کلاس Parent نیز نمی تواند دسترسی یابد.

اعضای static

اگر بخواهیم عضو داده ای (فیلد) یا خاصیتی ایجاد کنیم که در همه نمونه های کلاس قابل دسترسی باشد از کلمه کلیدی static استفاده می کنیم. کلمه کلیدی static برای اعضای داده ای و خاصیت هایی به کار می رود که می خواهند در همه نمونه های کلاس تقسیم شوند. وقتی که یک متد یا خاصیت به صورت staticتعریف شود، می توانید آنها را بدون ساختن نمونه ای از شی، فراخوانی کنید. به چند مثال توجه کنید:

```
1:  package myfirstprogram;
```

```
 2:
 3:  class SampleClass
 4:  {
 5:     public static String StaticMessage  = "This is the static message!";
 6:  }
 7:
 8:  public class MyFirstProgram
 9:  {
10:      public static void main(String[] args)
11:      {
12:          System.out.println(SampleClass.StaticMessage );
13:      }
14:  }
This is the static message!
```

در مثال بالا یک شی استاتیک به نام StaticMessage (خط 5) تعریف کرده ایم. مقدار شی StaticMessageدر همه نمونه های کلاس SampleClassقابل دسترسی است. برای فراخوانی یک متد،خاصیت و یا یک متغیر استاتیک ، به سادگی می توان نام کلاس و بعد از آن علامت دات (.) و در آخر نام متد یا خاصیت را نوشت. این موضوع را می توان در خط (12) مشاهده کرد. مشاهده می کنید که لازم نیست هیچ نمونه ای از کلاس ایجاد شود .یکی دیگر از کاربردهای این کلمه کلیدی در شمارش اشیاء است. به مثال زیر توجه کنید:

```
1:   package myfirstprogram;
 2:
 3:  class SampleClass
 4:  {
 5:     public static int number  = 1;
 6:
 7:     public SampleClass()
 8:     {
 9:          System.out.println("Number is : " + number++);
10:     }
11:  }
12:
13:  public class MyFirstProgram
14:  {
15:      public static void main(String[] args)
16:      {
17:          SampleClass Sample1 = new SampleClass ();
18:          SampleClass Sample2 = new SampleClass ();
19:          SampleClass Sample3 = new SampleClass ();
20:      }
21:  }
Number is : 1
Number is : 2
Number is : 3
```

همانطور که در خط 5 کد بالا مشاهده می کنید یک متغیر استاتیک با مقدار اولیه 1 ایجاد کرده ایم و در داخل سازنده کلاس در خط 9 ابتدا مقدار آن را چاپ و سپس یک واحد به آن اضافه کرده ایم. حال در خطوط 19-17 سه شیء از روی کلاس SampleClass ایجاد می کنیم. همانطور که در خروجی مشاهده می کنید با هر بار ایجاد شی یک بار سازنده کلاس فراخوانی و در نتیجه مقدار متغیر چاپ می شود. یکی از خواص متغیر های استاتیک این است که مقدار قبلی خود را حفظ می کنند. و از این خاصیت در مثال بالا برای شمارش اشیاء ساخته شده از کلاس استفاده کرده ایم.

عملگر instanceof

عملگر instanceofدر جاوا به شما اجازه می دهد که تست کنید که آیا یک شی یک نمونه از یک نوع خاص (کلاس، زیر کلاس، اینترفیس) است یا نه. عملگر instanceofبه دو عملوند نیاز دارد و یک مقدار بولی را برمی گرداند. به عنوان مثال، فرض کنید یک کلاس به نام Animelداریم، سپس یک نمونه از آن ایجاد می کنیم:

```
1:  package myfirstprogram;
 2:
 3:  class Animal
 4:  {
 5:
 6:  }
 7:
 8:  public class MyFirstProgram
 9:  {
10:      public static void main(String[] args)
11:      {
12:          Animal myAnimal = new Animal();
13:
14:          if (myAnimal instanceof Animal)
15:          {
16:                System.out.println("myAnimal is an Animal!");
17:          }
18:      }
19:  }
myAnimal is an Animal
```

رفتار عملگر instanceof را در این مثال مشاهده کردید. همانطور که می بینید از آن به عنوان شرط در عبارت ifاستفاده شده است. کاربرد آن در مثال بالا این است که چک می کند که آیا شی myAnimal یک نمونه از Animal است و چون نتیجه درست است کدهای داخل دستور if اجرا می شود. این عملگر همچنین می تواند چک کند که آیا یک شی خاص در سلسله مراتب وراثت یک نوع خاص است. به این مثال توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  class Animal
 4:  {
 5:
 6:  }
 7:
 8:  class Dog extends Animal
 9:  {
10:
11:  }
12:
13:  public class MyFirstProgram
14:  {
```

```
15:     public static void main(String[] args)
16:     {
17:         Dog myDog = new Dog();
18:
19:         if (myDog instanceof Animal)
20:         {
21:               System.out.println("myDog is an Animal!");
22:         }
23:     }
24: }
myDog is an Animal!
```

همانطور که در مثال بالا می بینید ما یک کلاس به نام Dog ایجاد کرده ایم که از کلاس Animal ارث می برد. سپس یک نمونه از این کلاس (Dog) ایجاد می کنیم و سپس با استفاده از عملگر instanceof تست می کنیم که آیا نمونه ایجاد شده جز کلاس Animal است یا یک کلاس مشتق شده از کلاس Animal می باشد. از آنجاییکه کلاس Dog از کلاس Animal ارث می برد (سگ من یک حیوان است.)، نتیجه عبارت درست (true) است. حال جمله بالا را تغییر دهیم : " حیوان من یک سگ است" . وقتی جمله برعکس می شود چه اتفاقی می افتد؟

```
Animal myAnimal = new Animal();

if (myAnimal instanceof Dog)
{
    System.out.println("myAnimal is a Dog!");
}
```

این باعث خطا نمی شود و عبارت فقط نتیجه falseرا بر می گرداند. می توان از کد بالا این را درک کرد که همه حیوانات سگ نیستند ولی همه سگها حیوان هستند .

Override

فرض کنید شما متد A را در کلاس A دارید و کلاس B از کلاس A ارث بری می کند، در این صورت متد A در کلاس B در دسترس خواهد بود. اما متد Aدقیق همان متدی است که از کلاس Aبه ارث برده شده است. حال اگر بخواهید که این متد رفتار متفاوتی از خود نشان دهد چکار می کنید؟ برای حل این مشکل باید متد کلاس پایه را Override کنید. به تکه کد زیر توجه کنید:

```
1:  package myfirstprogram;
2:
3:  class Parent
4:  {
5:      public void ShowMessage()
6:      {
7:          System.out.println("Message from Parent.")
8:      }
9:  }
10:
```

```
11:  class Child extends Parent
12:  {
13:      public void ShowMessage()
14:      {
15:          System.out.println("Message from Child.");
16:      }
17:  }
18:
19:  public class MyFirstProgram
20:  {
21:      public static void main(String[] args)
22:      {
23:          Parent myParent = new Parent();
24:          Child myChild = new Child();
25:
26:          myParent.ShowMessage();
27:          myChild.ShowMessage();
28:      }
29:  }
Message from Parent.
Message from Child.
```

همانطور که در کد بالا مشاهده می کنید دو کلاس به نام Parent (خطوط 9-3) و Child (خطوط 17-11) تعریف کرده ایم. کلاس Child که از کلاس Parent ارث می برد شامل متدی است که متد ()ShowMessage از کلاس پایه را overrideیا به صورت دیگری پیاده سازی می کند. همانطور که مشاهده می کنید این دو متد دقیقا شبیه به هم هستند و تنها اختلاف آنها در پیامی است که نشان می دهند. برای Override کردن یک متد قواعدی وجود دارد که در زیر به آنها اشاره شده است:

- تابع override شده یکی باشد.
- نوع مقدار بازگشتی تابع باید همانند یا فرزندی از نوع مقدار بازگشتی تابع override شده کلاس پدر باشد.
- سطح دسترسی تابع نمی تواند محدودتر از سطح دسترسی تابع override شده باشد. برای مثال: اگر تابع کلاس پدر به صورت public تعریف شده باشد، در این صورت تابع کلاس فرزند نمی تواند private یا protected باشد.
- تنها توابعی از کلاس پدر که توسط کلاس فرزند ارث بری شده اند می توانند override شوند.
- توابع static نمی توانند override شوند، ولی می توانند دوباره در کلاس فرزند تعریف شوند.
- اگر تابعی نمی تواند ارث برده شود، همانطور هم نمی تواند override شود.
- کلاس فرزند موجود در پکیج یکسان با کلاس پدر، می تواند تمامی توابعی از کلاس پدر را که به صورت private یا finalتعریف نشده باشند را override کند.
- کلاس فرزند در یک پکیج دیگر از کلاس پدر تنها می تواند توابع public یا protected کلاس پدر را که final نیستند را override کند.
- سازنده ها نمی توانند override شوند.

با استفاده از کلمه کلیدی super (خط 15) می توانید متد کلاس پایه را در داخل متد overrideشده فراخوانی کنید:

```
1:  package myfirstprogram;
2:
3:  class Parent
4:  {
5:      public void ShowMessage()
```

```
 6:      {
 7:          System.out.println("Message from Parent.")
 8:      }
 9:  }
10:
11:  class Child extends Parent
12:  {
13:      public void ShowMessage()
14:      {
15:          super.ShowMessage();
16:          System.out.println("Message from Child.");
17:      }
18:  }
19:
20:  public class MyFirstProgram
21:  {
22:      public static void main(String[] args)
23:      {
24:          Parent myParent = new Parent();
25:          Child myChild = new Child();
26:
27:          myParent.ShowMessage();
28:          myChild.ShowMessage();
29:      }
30:  }
Message from Parent.
Message from Parent.
Message from Child.
```

م توان یک کلاس دیگر که از کلاس Childارث بری می کند ایجاد کرده و دوباره متد ShowMessage()را overrideکرده و آنرا به صورت دیگر پیاده سازی کنیم. اگر بخواهید متدی را که ایجاد کرده اید به وسیله سایر کلاسها overrideنشود کافیست که از کلمه کلیدی finalبه صورت زیر استفاده کنید:

```
public final void ShowMessage()
```

حال اگر کلاس دیگری از کلاس Childارث ببرد نمی تواند متد ShowMessage()را overrideکند .

رابط(Interface)

اینترفیس ها شبیه به کلاسها هستند اما فقط شامل تعاریفی برای متدها و خواص (Property) می باشند. اینترفیس ها را می توان به عنوان پلاگین های کلاس ها در نظر گرفت. کلاسی که یک اینترفیس خاص را پیاده سازی می کند لازم است که کدهایی برای اجرا توسط اعضا و متدهای آن فراهم کند چون اعضا و متدهای اینترفیس هیچ کد اجرایی در بدنه خود ندارند. اجازه دهید که نحوه تعریف و استفاده از یک اینترفیس در کلاس را توضیح دهیم:

```
1:  package myfirstprogram;
```

```
 2:
 3: interface ISample
 4: {
 5:     public void ShowMessage(String message);
 6: }
 7:
 8: class Sample implements ISample
 9: {
10:     public void ShowMessage(String message)
11:     {
12:         System.out.println(message);
13:     }
14: }
15:
16: public class MyFirstProgram
17: {
18:     public static void main(String[] args)
19:     {
20:         Sample sample = new Sample();
21:
22:         sample.ShowMessage("Implemented the ISample Interface!");
23:     }
24: }
Implemented the ISample Interface!
```

در خطوط 6-3 یک اینترفیس به نام ISampleتعریف کرده ایم. بر طبق قراردادهای نامگذاری، اینترفیس ها به شیوه پاسکال نامگذاری می شوند و همه آنها باید با حرف I شروع شوند. همچنین در تعریف آنها باید از کلمه کلیدی interface استفاده شود. یک متد در داخل بدنه اینترفیس تعریف می کنیم (خط 5). به این نکته توجه کنید که متد تعریف شده فاقد بدنه است و در آخر ان باید از سیمیکولن استفاده شود. وقتی که متد را در داخل اینترفیس تعریف می کنید فقط لازم است که عنوان متد (نوع ، نام و پارامترهای آن) را بنویسید. به این نکته نیز توجه کنید که متدها و خواص تعریف شده در داخل اینترفیس سطح دسترسی ندارند چون باید همیشه هنگام اجرای کلاسها در دسترس باشند. برای پیاده سازی یک interface توسط یک کلاس از کلمه کلیدی implementsاستفاده می شود. کلاسی که اینترفیس را اجرا می کند کدهای واقعی را برای اعضای آن فراهم می کند. همانطور که در مثال بالا می بینید کلاس Sample، متد ShowMessage()اینترفیس ISampleرا اجرا و تغذیه می کند. می توان چند اینترفیس را در کلاس اجرا کرد.

```
class Sample implements ISample1, ISample2, ISample3
{
   //Implement all interfaces
}
```

به مثال زیر توجه کنید:

```
1: package myfirstprogram;
 2:
 3: interface IFirstinterface
 4: {
 5:     public void FirstMessage(String message);
 6: }
 7:
 8: interface ISecondinterface
 9: {
```

```
10:     public void SecondMessage(String message);
11: }
12:
13:
14: class Sample implements IFirstinterface , ISecondinterface
15: {
16:     @Override
17:     public void FirstMessage(String message)
18:     {
19:         System.out.println(message);
20:     }
21:
22:     @Override
23:     public void SecondMessage(String message)
24:     {
25:         System.out.println(message);
26:     }
27: }
28:
29: public class MyFirstProgram
30: {
31:     public static void main(String[] args)
32:     {
33:         Sample sample = new Sample();
34:
35:         sample.FirstMessage("Implemented the IFirstinterface Interface!");
36:         sample.SecondMessage("Implemented the ISecondinterface Interface!");
37:     }
38: }
```

```
Implemented the IFirstinterface Interface!
Implemented the ISecondinterface Interface!
```

درست است که می توان از چند اینترفیس در کلاس استفاده کرد ولی باید مطمئن شد که کلاس می تواند همه اعضای اینترفیسها را تغذیه کند. همانطور که در کد بالا مشاهده می کنید دو اینترفیس به نام های IFirstinterface و ISecondinterfaceدر خطوط 11-3 تعریف شده اند که به ترتیب دارای دو متد به نام های FirstMessage() و SecondMessage()می باشند. در خط 14 کلاس Sample این دو اینترفیس را پیاده سازی کرده است و درنتیجه همانطور که اشاره شد لازم است که کدهای بدنه دو متد موجود در این دو رابط را تغذیه کند. که این کار در خطوط 26-17 انجام شده است. عبارت @Override موجود در خطوط 16 و 22 به این دلیل قرار داده شده است که به کامپایلر اعلام کند که این دو متد Overrideشده اند و به نوعی یک اخطار به کامپایلر هست که می گوید این تایع مربوط به کلاس یا اینترفیسی است که کلاس جاری از آن مشتق شده و یا پیاده سازی کرده است.وجود @ هم به خاطر قرار داد خود زبان می باشد. اگر یک کلاس از کلاس پایه ارث ببرد و در عین حال از اینترفیس ها هم استفاده کند، در این صورت باید نام کلاس پایه قبل از نام اینترفیس ها ذکر شود . به شکل زیر:

```
class Sample extends BaseClass, ISample1, ISample2
{
}
```

نکته دیگر اینکه نمی توان از یک اینترفیس نمونه ای ایجاد کرد چون اینترفیس ها دارای سازنده نیستند، مثلا کد زیر اشتباه است :

```
ISample sample = new ISample();
```

اینترفیسها حتی می توانند از اینترفیسهای دیگر با استفاده از کلمه کلیدی extends ارث بری کنند. به مثال زیر توجه کنید:

```
1:  package myfirstprogram;
2:
3:  interface IBase
4:  {
5:      void BaseMethod();
6:  }
7:
8:  interface ISample extends IBase
9:  {
10:     void ShowMessage(String message);
11: }
12:
13: class Sample implements ISample
14: {
15:     @Override
16:     public void ShowMessage(String message)
17:     {
18:         System.out.println(message);
19:     }
20:
21:     @Override
22:     public void BaseMethod()
23:     {
24:         System.out.println("Method from base interface!");
25:     }
26: }
27:
28: public class MyFirstProgram
29: {
30:     public static void main(String[] args)
31:     {
32:         Sample sample = new Sample();
33:
34:         sample.ShowMessage("Implemented the ISample Interface!");
35:         sample.BaseMethod();
36:     }
37: }
```

```
Implemented the ISample Interface!
Method from base interface!
```

همانطور که در خط 8 کد بالا مشاهده می کنید رابط ISample از رابط IBase ارث بری کرده است پس حتی اگر کلاس Sampleفقط اینترفیس ISampleرا پیاده سازی کند، لازم است که همه اعضای IBaseرا هم پیاده سازی کند چون ISampleاز آن ارث بری می کند.

کلاسهای انتزاعی(Abstract Class)

کلاسهای مجرد (abstract) کلاسهایی هستند که کلاس پایه سایر کلاسها هستند.این نوع کلاسها می توانند مانند کلاسهای عادی دارای سازنده باشند. شما نمی توانید از کلاسهای انتزاعی نمونه ایجاد کنید چون که هدف اصلی از به کار بردن کلاسهای انتزاعی استفاده از آنها به عنوان کلاس پایه برای کلاسهای مشتق است. برای تعریف یک کلاس انتزاعی از کلمه کلیدی abstractاستفاده می شود. به مثال زیر در مورد استفاده از کلاسهای انتزاعی توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  abstract class Base
 4:  {
 5:      protected int number;
 6:      protected String name;
 7:
 8:      public abstract void ShowMessage();
 9:
10:      public Base(int number, String name)
11:      {
12:          this.number = number;
13:          this.name = name;
14:      }
15:  }
16:
17:  class Derived extends Base
18:  {
19:      @Override
20:      public void ShowMessage()
21:      {
22:           System.out.println("Hello World!");
23:      }
24:
25:      public Derived(int number, String name)
26:      {
27:          super(number, name);
28:      }
29:  }
```

در داخل کلاس انتزاعی دو فیلد محافظت شده (protected) تعریف کرده ایم (خطوط 6-5) که می خواهیم آنها را توسط سازنده کلاس مقدار دهی کنیم (خطوط 14-10). یک متد هم را به صورت انتزاعی (abstract) تعریف کرده ایم (خط 8). به این نکته توجه کنید که برای تعریف این متد کلمه کلیدی abstractرا به کار برده ایم.

این متد باید به وسیله کلاسهایی که از این کلاس ارث می برند overrideیا به صورت دیگر پیاده سازی شود، ولی از آن جاییکه به صورت abstractتعریف شده است فاقد بدنه می باشد. می بینید که کلاسهای abstractمی توانند شامل propertyهای معمولی مانند property، Nameمثال بالا باشند. کلاسهای abstractحداقل باید یک عضو abstractداشته باشند.

یک کلاس دیگر تعریف می کنید که از کلاس Baseارث بری کند. سپس در خطوط 23-19 متد abstractرا به صورت دیگر پیاده سازی می کنیم (overrideکنیم). همچنین یک سازنده تعریف می کنیم(خطوط 28-25) و با استفاده از کلمه کلیدی super

مقادیر پارامترها را به سازنده پایه ارسال می کنیم .نمی توان از یک کلاس abstract نمونه ایجاد کرد ولی از کلاس هایی که از این نوع کلاس ها مشتق می شوند، می توان نمونه ایجاد کرد.

کلاس final و متدfinal

کلاس final (کلاس نهایی)، کلاسی است که دیگر کلاس ها نمی توانند از آن ارث بری کنند و چون قابلیت ارث بری ندارد نمی تواند مجرد (abstract) هم باشد. مثال زیر یک کلاس final را نشان می دهد:

```
final class Base
{
   private int someField;

   public void SomeMethod()
   {
      //Do something here
   }

   //Constructor
   public Base()
   {
      //Do something here
   }
}

class Derived extends Base
{
   //This class cannot inherit the Base class
}
```

برای تعریف این کلاس ها از کلمه کلیدی final استفاده می شود .مشاهده می کنید که کلاس نهایی مانند کلاس های عادی، دارای فیلد، خواص، و متد می باشند. کلاس مشتق (Derived) در مثال بالا با خط قرمز نشان داده شده است چون نمی تواند از کلاس نهایی (Base) ارث بری کند. وقتی یک کلاس را نهایی می کنیم، تمام متدهای آن نیز نهایی می شوند. استفاده از این کلاسها همانطور که ذکر شد زمانی مفید است که بخواهید کلاسی ایجاد کنید که دیگر کلاسها نتوانند از آن ارث بری کنند.

متدFinal

متد final به متدی گفته می شود که هیچ زیر کلاسی نتواند آن را بازنویسی یا Override کند. به مثال زیر توجه کنید:

```
package program;

class Parent
{
```

```
    final void ShowMessage()
    {
        System.out.println("This is a final method!");
    }
}

class Child extends Parent
{
    @Override
    void ShowMessage()
    {
        System.out.println("This is a final method that Overriden!");
    }
}

public class Program
{
    public static void main(String[] args)
    {

    }
}
```

اگر به کدهای بالا توجه کنید و آن را در محیط NetBeans بنویسید مشاهده می کنید که در خط قرمز بالا خطا به وجود می آید. چون کلاس Child از کلاس Parentارث بری کرده است و زیر کلاس محسوب می شود و طبق تعریف هیچ زیر کلاسی نمی تواند متدهای final را بازنویسی کند.

چند ریختی(Polymorphism)

چند ریختی به کلاسهایی که در یک سلسله مراتب وراثتی مشابه هستند اجازه تغییر شکل و سازگاری مناسب می دهد و همچنین به برنامه نویس این امکان را می دهد که به جای ایجاد برنامه های خاص، برنامه های کلی و عمومی تری ایجاد کند. . به عنوان مثال در دنیای واقعی همه حیوانات غذا می خورند، اما روش های غذا خوردن آنها متفاوت است. در یک برنامه برای مثال ، یک کلاس به نام Animalایجاد می کنید. بعد از ایجاد این کلاس می توانید آن را چند ریخت (تبدیل) به کلاس Birdکنید و متد Fly()را فراخوانی کنید. به مثالی در باره چند ریختی توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  class Animal
 4:  {
 5:      public void Eat()
 6:      {
 7:          System.out.println("The animal ate
 8:      }
 9:  }
10:
11:  class Dog extends Animal
```

```
12:  {
13:      @Override
14:      public void Eat()
15:      {
16:          System.out.println("The dog ate!")
17:      }
18:  }
19:
20:  class Bird extends Animal
21:  {
22:      @Override
23:      public void Eat()
24:      {
25:          System.out.println("The bird ate!"
26:      }
27:  }
28:
29:  class Fish extends Animal
30:  {
31:      @Override
32:      public void Eat()
33:      {
34:          System.out.println("The fish ate!"
35:      }
36:  }
37:
38:  public class MyFirstProgram
39:  {
40:      public static void main(String[] args)
41:      {
42:          Dog myDog = new Dog();
43:          Bird myBird = new Bird();
44:          Fish myFish = new Fish();
45:          Animal myAnimal = new Animal();
46:
47:          myAnimal.Eat();
48:
49:          myAnimal = myDog;
50:          myAnimal.Eat();
51:          myAnimal = myBird;
52:          myAnimal.Eat();
53:          myAnimal = myFish;
54:          myAnimal.Eat();
55:      }
56:  }
```

```
The animal ate!
The dog ate!
The bird ate!
The fish ate!
```

همانطور که مشاده می کنید 4 کلاس مختلف تعریف کرده ایم Animal .کلاس پایه است و سه کلاس دیگر از آن مشتق می شوند. هر کلاس متد Eat()مربوط به خود را دارد. نمونه ای از هر کلاس ایجاد کرده ایم (45-42). حال متد Eat()را به وسیله نمونه ایجاد شده از کلاس Animalبه صورت زیر فراخوانی می کنیم:

```
Animal myAnimal = new Animal();

myAnimal.Eat();
```

در مرحله بعد چندریختی روی می دهد. همانطور که در مثال بالا مشاهده می کنید شی Dogرا برابر نمونه ایجاد شده از کلاس Animalقرار می دهیم (خط 49) و متد Eat()را بار دیگر فراخوانی می کنیم (خط 50). حال با وجود اینکه ما از نمونه کلاس Animalاستفاده کرده ایم ولی متد Eat()کلاس Dog فراخوانی می شود. این به دلیل تاثیر چند ریختی است.

سپس دو شی دیگر Bird) و Fish)را برابر نمونه ایجاد شده از کلاس Animalقرار می دهیم و متد Eat()مربوط به هر یک را فراخوانی می کنیم(خطوط 54-51). به این نکته توجه کنید که وقتی در مثال بالا اشیاء را برابر نمونه کلاس Animalقرار می دهیم ازعمل Castاستفاده نکرده ایم چون این کار (cast) وقتی که بخواهیم یک شی از کلاس مشتق (مثلا Dog) را در شیی از کلاس پایه (Animal) ذخیره کنیم لازم نیست. همچنین می توان کلاس Animalرا با سازنده هر کلاس مشتق دیگر مقدار دهی اولیه کرد:

```
Animal myDog = new Dog();
Animal myBird = new Bird();
Animal myFish = new Fish();

myDog.Eat();
myBird.Eat();
myFish.Eat();
```

اجازه دهید که برنامه بالا را اصلاح کنیم تا مفهوم چند ریختی را بهتر متوجه شوید:

```
 1:  package myfirstprogram;
 2:
 3:  class Animal
 4:  {
 5:      public void Eat()
 6:      {
 7:          System.out.println("The animal ate!")
 8:      }
 9:  }
10:
11:  class Dog extends Animal
12:  {
13:      @Override
14:      public void Eat()
15:      {
16:          System.out.println("The dog ate!");
17:      }
18:
19:      public void Run()
20:      {
21:          System.out.println("The dog ran!");
22:      }
23:  }
24:
25:  class Bird extends Animal
26:  {
27:      @Override
```

```
28:     public void Eat()
29:     {
30:         System.out.println("The bird ate!");
31:     }
32:
33:     public void Fly()
34:     {
35:         System.out.println("The bird flew!");
36:     }
37: }
38:
39: class Fish extends Animal
40: {
41:     @Override
42:     public void Eat()
43:     {
44:         System.out.println("The fish ate!");
45:     }
46:
47:     public void Swim()
48:     {
49:         System.out.println("The fish swam!");
50:     }
51: }
52:
53: public class MyFirstProgram
54: {
55:     public static void main(String[] args)
56:     {
57:         Animal animal1 = new Dog();
58:         Animal animal2 = new Bird();
59:         Animal animal3 = new Fish();
60:
61:         Dog  myDog  = (Dog)animal1;
62:         Bird myBird = (Bird)animal2;
63:         Fish myFish = (Fish)animal3;
64:
65:         myDog.Run();
66:         myBird.Fly();
67:         myFish.Swim();
68:     }
69: }
```

```
The dog ran!
The bird flew!
The fish swam!
```

در بالا سه شی از کلاس Animal ایجادو آنها را بوسیله سه سازنده از کلاسهای مشتق مقدار دهی اولیه کرده ایم (خطوط 59-57). سپس با استفاده از عمل cast اشیا ایجاد شده از کلاس Animal را در نمونه هایی از کلاس های مشتق ذخیره می کنیم (خطوط 63-61). وقتی این کار را انجام دادیم می توانیم متدهای مخصوص به هر یک از کلاسهای مشتق را فراخوانی کنیم (خطوط 67-65). یک را میانبر دیگر به وسیله کد زیر مشخص شده است ولی در این روش شما نمی توانید اشیا ایجاد شده از کلاس Animal را در نمونه هایی از کلاس های مشتق ذخیره می کنید.

```
((Dog)animal1).Run();
((Bird)animal2).Fly();
```

```
((Fish)animal3).Swim();
```

از چند ریختی می توان در رابط ها هم استفاده کرد. به کد زیر توجه کنید:

```
 1:  package myfirstprogram;
 2:
 3:  interface IAnimal
 4:  {
 5:      public void Eat();
 6:  }
 7:
 8:  class Dog implements IAnimal
 9:  {
10:      @Override
11:      public void Eat()
12:      {
13:          System.out.println("The dog ate!");
14:      }
15:
16:  }
17:
18:  class Bird implements IAnimal
19:  {
20:      @Override
21:      public void Eat()
22:      {
23:          System.out.println("The bird ate!");
24:      }
25:  }
26:
27:  class Fish implements IAnimal
28:  {
29:      @Override
30:     public void Eat()
31:      {
32:          System.out.println("The fish ate!");
33:      }
34:  }
35:
36:  public class MyFirstProgram
37:  {
38:      public static void main(String[] args)
39:      {
40:          IAnimal myDog  = new Dog();
41:          IAnimal myBird = new Bird();
42:          IAnimal myFish = new Fish();
43:
44:          myDog.Eat();
45:          myBird.Eat();
46:          myFish.Eat();
47:      }
48:  }
```

```
The dog ate!
The bird ate!
The fish ate!
```

تسلط کامل بر چند ریختی و وراثت برای درک بهتر شی گرایی ضروری است.

مدیریت استثناءها و خطایابی

بهترین برنامه نویسان در هنگام برنامه نویسی با خطاها و باگ ها در برنامه شان مواجه می شوند. درصد زیادی از برنامه ها هنگام تست برنامه با خطا مواجه می شوند. بهتر است برای از بین بردن یا به حداقل رساندن این خطاها، به کاربر در مورد دلایل به وجود آمدن آنها اخطار داده شود .خوشبختانه جاوا برای این مشکل راه حلی ارائه داده است. جاوا دارای مجموعه بزرگی از کلاسهایی است که برای برطرف کردن خطاهای خاص از آنها استفاده می کند. استثناء ها در جاوا راهی برای نشان دادن دلیل وقوع خطا در هنگام اجرای برنامه است.

جاوا دارای مجموعه بزرگی از کلاسهای استثناء است که شما می توانید با استفاده از آنها خطاهایی که در موقعیتهای مختلف روی می دهند را برطرف کنید. حتی می توانید یک کلاس استثناء شخصی ایجاد کنید .استثناء ها توسط برنامه به وجود می آیند و شما لازم است که آنها را اداره کنید. به عنوان مثال در دنیای کامپیوتر یک عدد صحیح هرگز نمی تواند بر صفر تقسیم شود. اگر بخواهید این کار را انجام دهید (یک عدد صحیح را بر صفر تقسیم کنید)، با خطا مواجه می شوید .اگر یک برنامه در جاوا با چنین خطایی مواجه شود پیغام خطای "java.lang.ArithmeticException: / by zero"نشان داده می شود که بدین معنا است که عدد را نمی توان بر صفر تقسیم کرد.

باگ(Bug) اصطلاحا خطا یا کدی است که رفتارهای ناخواسته ای در برنامه ایجاد می کند. خطایابی فرایند برطرف کردن باگها است ، بدین معنی که خطاها را از برنامه پاک کنیم Netbeans .دارای ابزارهایی برای خطایابی هستند ، که خطاها را یافته و به شما اجازه می دهند آنها را برطرف کنید. در درسهای آینده خواهید آموخت که چگونه از این ابزارهای کارامد جهت برطرف کردن باگها استفاده کنید. قبل از اینکه برنامه را به پایان برسانید لازم است که برنامه تان را اشکال زدایی کنید.

استثناء های اداره نشده

استثناءهای اداره نشده، استثناءهایی هستند که به درستی توسط برنامه اداره نشده اند و باعث می شوند که برنامه به پایان برسد. در اینجا می خواهیم به شما نشان دهیم که وقتی یک برنامه در زمان اجرا با یک استثناء مواجه می شود و آن را اداره نمی کند چه اتفاقی می افتد. در آینده خواهید دید که یک استثناء چگونه به صورت بالقوه باعث نابودی جریان و اجرای برنامه شما می شود. به برنامه زیر توجه کنید:

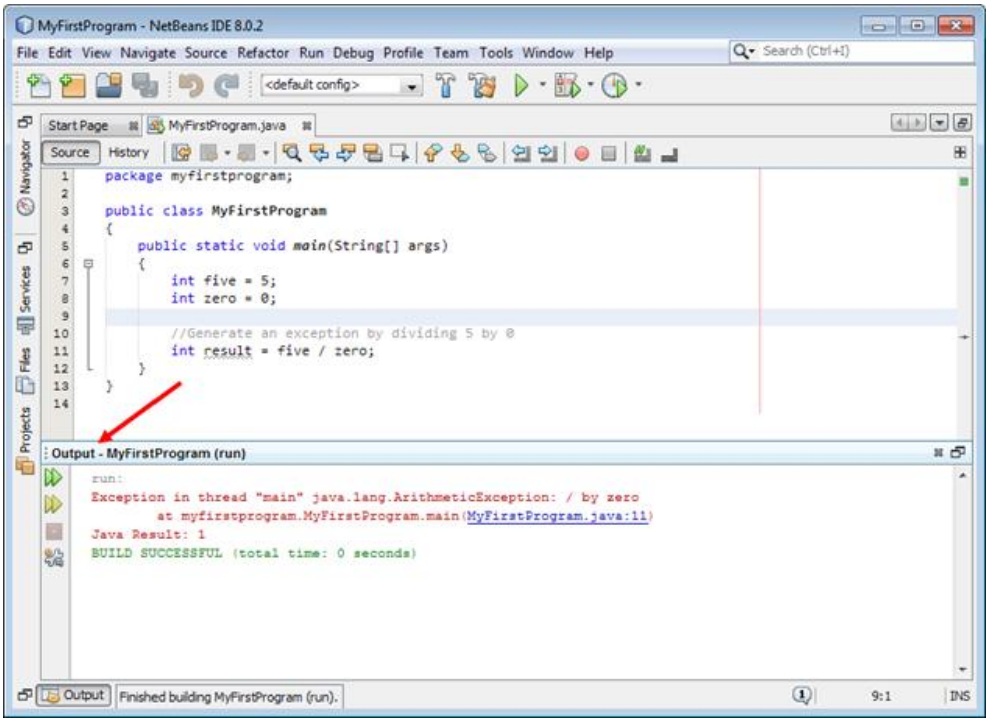
```
1:  package myfirstprogram;
 2:
 3:  public class MyFirstProgram
```

```
 4:  {
 5:      public static void main(String[] args)
 6:      {
 7:          int five = 5;
 8:          int zero = 0;
 9:
10:          //Generate an exception by dividing 5 by 0
11:          int result = five / zero;
12:      }
13:  }
```

همانطور که در مثال بالا مشاهده می کنید تقسیم یک عدد صحیح بر صفر غیر مجاز است و باعث ایجاد خطای java.lang.ArithmeticException: / by zero می شود. برنامه را با زدن دکمه F6 اجرا می کنیم. برنامه با موفقیت اجرا شده ولی با پیغام خطای زیر مواجه می شوید:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at myfirstprogram.MyFirstProgram.main(MyFirstProgram.java:11)
```

همانطور که مشاهده می کنید با اجرای برنامه اطلاعاتی در باره خروجی ها و یا خطاهای آن در پنجره Output نمایش داده می شود:



پنجره Output پنجره ای مفید است که در مورد استثناء اطلاعاتی در اختیار شما می گذارد و معمولا محل دقیق خطا را به شما نشان می دهد که در شکل بالا خط 11 (MyFirstProgram.java:11) نشان داده شده است.

دستور try وcatch

می توان خطاها را با استفاده از دستور try...catch اداره کرد. بدین صورت که کدی را که احتمال می دهید ایجاد خطا کند در داخل بلوک try قرار می دهید. بلوک catch هم شامل کدهایی است که وقتی اجرا می شوند که برنامه با خطا مواجه شود. تعریف ساده ی این دو بلوک به این صورت است که بلوک try سعی می کند که دستورات را اجرا کند و اگر در بین دستورات خطایی وجود داشته باشد برنامه دستورات مربوط به بخش catch را انجام می دهد. برنامه زیر نحوه استفاده از دستور try...catch را نمایش می دهد:

```
1:  package myfirstprogram;
 2:
 3:  public class MyFirstProgram
 4:  {
 5:      public static void main(String[] args)
 6:      {
 7:          int result;
 8:          int x = 5;
 9:          int y = 0;
10:
11:          try
12:          {
13:              result = x / y; //ERROR
14:          }
15:          catch(ArithmeticException e)
16:          {
17:              System.out.println("An attempt to divide by 0 was detected.");
18:          }
19:      }
20:  }
An attempt to divide by 0 was detected.
```

در دال بلوک try، مقدار x را که 5 است بر y که مقدار آن 0 است تقسیم کرده ایم. نتیجه محاسبه به وجود آمدن خطای by / zero (عدد تقسیم بر صفر) است. از آنجاییکه در برنامه بالا خطایی به وجود آمده است کدهای داخل بلوک catch اجرا می شوند. بنابراین:

```
try
{
    result = x / y; //Error: Jump to catch block
    System.out.println("This line will not be executed.");
}
catch(ArithmeticException e)
{
    System.out.println("An attempt to divide by 0 was detected.");
}
```

همانطور که در مثال بالا مشاهده می کنید از یک نوع استثناء مخصوص به یک خطا در داخل بلوک catch که کلاس ArithmeticException است استفاده کرده ایم. همچنین می توانید مقدار استثنا را در داخل یک متغیر قرار داده و سپس آن را نمایش دهید:

```
try
```

```
{
    result = x / y; //ERROR
}
catch(ArithmeticException error)
{
    System.out.println(error.getMessage());
}
/ by zero
```

متغیر دارای اطلاعات مفیدی در مورد استثناء به وجود آمده است. برای نمایش اطلاعاتی در مورد استثناء هم از متد getMessage()استفاده می کنیم. همه کلاسهای استثناء توضیحاتی در مورد خطاها می دهند. در درسهای آینده در مورد خصوصیات استثناء ها بیشتر توضیح می دهیم. اگر فکر می کنید که در بلوک tryممکن است با چندین خطا مواجه شوید می توانید از چندین بلوک catchاستفاده نمایید ولی به یاد داشته باشید که برای هر کدام از آن خطاها از کلاس استثناء مربوط به هر یک استفاده کنید:

```
int result;
int x = 5;
int y;

try
{
    y = input.nextInt();
    result = x / y; //ERROR
}
catch(ArithmeticException error)
{
    System.out.println(error.getMessage());
}
catch(InputMismatchException error)
{
    System.out.println(error.getMessage());
}
```

از انجاییکه مقدار y به وسیله ورودی که از کاربر گرفته می شود ،تعیین می شود، مقدار آن باید با توجه به مثال بالا غیر صفر باشد (عدد تقسیم بر صفر تعریف نشده است). اما یک مشکل وجود دارد.چون ممکن است که کاربر یک مقدار غیر عددی وارد کند (مثلا یک حرف) که در این صورت برنامه نمی تواند حرف را به عدد تبدیل کند و خطای نوع (InputMismatchException)اتفاق می افتد. وقتی استثناء اتفاق افتاد بلوک catchمربوط به این خطا اجرا می شود و محاسبه خارج قسمت تقسیم xبر yنادیده گرفته می شود. قسمت catch کد بالا را به صورت زیر هم می توان نوشت:

```
catch(ArithmeticException | InputMismatchException error)
{
    System.out.println(error.getMessage());
}
```

حال فرض کنید شما می خواهید تمام خطاهای احتمالی که ممکن است در داخل بلوک tryاتفاق می افتند را فهمیده و اداره کنید این کار چگونه امکانپذیر است؟ به راحتی و با استفاده از کلاس عمومی Exceptionمی توانید این کار را انجام داد. هر کلاس استثناء در جاوا از این کلاس ارث بری می کند بنابراین شما می توانید هر نوع استثناءیی را در شئی از کلاس Exceptionذخیره نمایید.

```
try
{
    //Put your codes to test here
}
catch (Exception error)
{
    System.out.println(error.Message);
}
```

با استفاده از این روش دیگر لازم نیست نگران اتفاق خطاهای احتمالی باشید چون بلوک catchبرای هر گونه خطایی که در داخل بلوک tryتشخیص داده شود پیغام مناسبی نشان می دهد. به این نکته توجه کنید که اگر بخواهید از کلاس پایه Exceptionهمراه با سایر کلاسهای استثناء دیگر که از آن مشتق می شوند در برنامه استفاده کنید باید کلاس پایه Exception در آخرین بلوک catchقرار گیرد.

```
try
{
    //Put your codes to test here
}
catch (ArithmeticException e)
{
    System.out.println("Division by zero is not allowed.");
}
catch (InputMismatchException e)
{
    System.out.println("Error on converting the data to proper type.");
}
catch (Exception e)
{
    System.out.println("An error occured.");
}
```

اگر کلاس پایه Exceptionرا در اولین بلوک catchقرار دهیم و خطایی در برنامه رخ دهد چون تمام کلاسهای استثناء از این کلاس مشتق می شوند در نتیجه اولین بلوک catchاجرا شده و سایر بلوک ها حتی با وجود اینکه خطای مورد نظر به آنها مربوط باشد اجرا نمی شوند .

بلوک finally

گاهی اوقات می خواهید برخی کدها همیشه اجرا شوند خواه استثنا رخ دهد، خواه رخ ندهد، در این صورت از بلوک finally استفاده می شود. قبلا یاد گرفتیم که اگر در بلوک tryیک استثناء رخ دهد همه کدهای موجود در این بلوک نادیده گرفته شده و برنامه به قسمت catchمی رود. کدهای نادیده گرفته شده ممکن است در برنامه نقش حیاتی داشته باشند.

هدف بلوک finallyهم حفظ نقش این کدها به صورت غیر مستقیم است. کدهایی را که فکر می کنید کدهای پایه ای هستند و برای اجرای برنامه لازم هستند را در داخل بلوک finallyقرار دهید. برنامه زیر نحوه استفاده از این بلوک را نشان می دهد:

```
1:  package myfirstprogram;
 2:
 3:  public class MyFirstProgram
 4:  {
 5:      public static void main(String[] args)
 6:      {
 7:          int result;
 8:          int x = 5;
 9:          int y = 0;
10:
11:          try
12:          {
13:              result = x / y; //ERROR
14:          }
15:          catch(ArithmeticException error)
16:          {
17:              System.out.println(error.getMessage());
18:          }
19:          finally
20:          {
21:              System.out.println("finally blocked was reached.");
22:          }
23:      }
24:  }
\ by zero.
finally blocked was reached.
```

بلوک finallyبعد از بلوک catchنوشته می شود. اگر از چندین بلوک catchدر برنامه استفاده می کنید بلوک finallyباید بعد از همه آنها قرار گیرد. می توان از بلوک tryو finallyدر صورتی که بلوک catchنداشته باشیم به صورت زیر استفاده کرد.

```
try
{
    //some code
}
finally
{
    //some code
}
```

از این بلوک معمولا برای بستن یک اتصال پایگاه داده یا بستن یک فایل استفاده می شود.

ایجاد استثناء

شما می توانید در هر جای برنامه یک خطای ساختگی ایجاد کنید .همچنین اگر پیغام پیشفرض استثناءها را دوست ندارید می توانید به دلخواه خودتان یک پیغام برای نمایش ایجاد کنید. به مثال زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  import java.util.Scanner;
```

```
 4:
 5: public class MyFirstProgram
 6: {
 7:     public static void main(String[] args)
 8:     {
 9:         Scanner input = new Scanner(System.in);
10:
11:         int firstNumber, secondNumber, result;
12:
13:         System.out.print("Enter the first number: ");
14:         firstNumber = input.nextInt();
15:
16:         System.out.print("Enter the second number: ");
17:         secondNumber = input.nextInt();
18:
19:         try
20:         {
21:             if (secondNumber == 0)
22:             {
23:                 throw new ArithmeticException();
24:             }
25:             else
26:             {
27:                 result = firstNumber / secondNumber;
28:             }
29:         }
30:         catch (ArithmeticException error)
31:         {
32:             System.out.println(error.getMessage());
33:         }
34:
35:     }
36: }
Enter the first number: 10
Enter the second number: 0
null
```

در خط 23 و درست قبل از یک نمونه ایجاد شده از کلاس exception از کلمه کلیدی throw استفاده کرده ایم. می توان مستقیما یک نمونه از کلاس exception ایجاد و یک خطا را به دام انداخت. به مثال زیر توجه کنید:

```
ArithmeticException error = new ArithmeticException();

throw error;
```

همچنین می توان یک پیغام خطای سفارشی را به وسیله یکی دیگر از سربارگذاری های کلاس Exception که یک رشته را دریافت و آن را به عنوان پیغام خطا نمایش می دهد، نمایش داد.

```
throw new ArithmeticException("Cannot divide by zero!");
```

در این حالت پیغام خطای پیشفرض تغییر کرده و در متد getMessag() ذخیره می شود. ایجاد استثناء بیشتر در مواقعی به کار می رود که یک کد در حالت عادی خطا ندارد ولی شما می خواهید در هر صورت به عنوان یک خطا در نظر گرفته شود.

تعریف یک استثناء توسط کاربر

در جاوا می توان یک استثناء سفارشی ایجاد کرد. استثناء سفارشی استثنایی است که توسط کاربر تعریف می شود و باید از کلاس پایه Exceptionارث بری کند. برای این کار یک کلاس جداگانه که از کلاس پایه Exceptionارث می برد ایجاد می کنیم. به کد زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  import java.util.Scanner;
 4:
 5:
 6:  class NegativeNumberException extends Exception
 7:  {
 8:      public NegativeNumberException()
 9:      {
10:          super("The operation will result to a negative number.");
11:      }
12:
13:      public NegativeNumberException(String message)
14:      {
15:          super(message);
16:      }
17:
18:      public NegativeNumberException(String message, Exception inner)
19:      {
20:          super(message, inner);
21:      }
22:  }
23:
24:  public class MyFirstProgram
25:  {
26:
27:      public static void main(String[] args)
28:      {
29:          Scanner input = new Scanner(System.in);
30:
31:          int firstNumber, secondNumber, difference;
32:
33:          System.out.print("Enter the first number: ");
34:          firstNumber = input.nextInt();
35:
36:          System.out.print("Enter the second number: ");
37:          secondNumber = input.nextInt();
38:
39:          difference = firstNumber - secondNumber;
40:
41:          try
42:          {
43:              if (difference < 0)
44:              {
45:                  throw new NegativeNumberException();
```

```
46:                 }
47:             }
48:             catch (NegativeNumberException error)
49:             {
50:                 System.out.println(error.getMessage());
51:             }
52:         }
53: }
```

```
Enter the first number: 10
Enter the second number: 11
The operation will result to a negative number.
```

در خط 6 مشاهده می کنید کلاس ایجاد شده توسط ما از کلاس Exceptionارث بری کرده است. به عنوان یک قرارداد باید به آخر نام کلاس های استثنایی که توسط کاربر تعریف می شوند کلمه Exceptionاضافه شده و 3 سازنده برای آنها تعریف شود.

- اولین سازنده بدون پارامتر می باشد.
- دومین سازنده یک آرگومان از نوع رشته برای نمایش پیغام خطا قبول می کند.
- سومین سازنده که دو آرگومان قبول می کند ، یکی پیغام خطا را نمایش داده و یکی بخش innerاست آن برای نشان دادن علت وقوع استثناء می باشد.

حال می خواهیم یک کلاس استثناء خیلی سفارشی ایجاد کنیم. به خطوط 51-29 توجه کنید .چون که قرار است که از کاربر ورودی دریافت کنیم از کلاس Scanner در خط 29 استفاده کرده ایم. از آنجاییکه تولید یک عدد منفی در هیچ برنامه ای یک استثناء محسوب نمی شود، ما به صورت دستی و برای خودمان یک استثناء ایجاد کرده ایم. ابتدا از کاربر می خواهیم که دو مقدار را وارد کند(خطوط 37-33). پس تفاوت دو عدد را محاسبه می کنیم (خط 39). در داخل بلوک tryتست می کنیم که آیا حاصل تفریق دو عدد، یک عدد منفی است (خط 47-41). اگر یک عدد منفی بود سپس یک نمونه از کلاس NegativeNumberExceptionایجاد می کنیم (خط 45) .بعد از ایجاد نمونه به وسیله بلوک catchو برای نشان داده پیغام خطا آن را اداره می کنیم (خطوط 51-48).

کلکسیون ها(Collections)

قبلا یاد گرفتیم که آرایه ها به ما اجازه ذخیره چندین مقدار از یک نوع را می دهند. آرایه ها از کلاس Java.util.Arrays ارث بری می کنند که این کلاس دارای خواص و متدهایی برای کار با داده های ساده ای مانند طول آرایه می باشد. آرایه های ساده در جاوا دارای طول ثابتی هستند که یک بار تعریف و مقدار دهی می شوند و شما نمی توانید طول یک آرایه خاص را افزایش یا کاهش دهید.

جاوا گزینه بهتری برای جایگزین کردن با آرایه ها پیشنهاد می دهد و بیشتر آنها کلاسها و رابطهایی هستند که در پکیج و کلاس java.util.ArrayListقرار دارند. به عنوان مثال کلاس ArrayList رفتاری شبیه به یک آرایه معمولی دارد با این تفاوت که به شما اجازه می دهد که طول آن را به صورت پویا تغییر داده یا یک عنصر را در طول اجرای برنامه به آن اضافه کرده و یا از آن حذف نمایید. دردرس بعد پی می برید که چگونه یک کلاس که شامل مجموعه ای از اشیاء است را به وسیله اجرا کردن و یا ارث بری از رابط ها و متدها ایجاد کنیم.

کلاسArrayList

کلاس ArrayList به شما اجازه ذخیره مقادیر انواع داده ای مختلف، و توانایی حذف و اضافه عناصر آرایه در هر لحظه را می دهد. در مثال زیر به سادگی کاربرد کلاس ArrayList آمده است.

```
package myfirstprogram;

import java.util.ArrayList;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayList myArray = new ArrayList();

        myArray.add("John");
        myArray.add(5);
        myArray.add(true);
        myArray.add(3.65);
        myArray.add('R');

        for (Object element : myArray)
        {
            System.out.println(element);
        }
    }
}
```

```
John
5
true
3.65
R
```

برای استفاده از این کلاس ابتدا باید در قسمت فضاهای نامی ، فضای نام java.util.ArrayListرا وارد کنیم(خط 3). همانطور که در مثال مشاهده می کنید یک نمونه از کلاس ArrayList ایجاد می کنیم. برای اضافه کردن یک عنصر به آرایه باید از متد () add استفاده کنیم. از آنجاییکه شی ایجاد شده از کلاس ArrayList آرگومانی از نوع Object قبول می کند بنابراین می توان مقادیری از هر نوع داده ای به آن ارسال کرد چون هر چیز در سی شارپ از Objectارث بری می کند.

حال برای نمایش توانایی این کلاس در نگهداری انواع داده ای مختلف پنج مقدار از پنج نوع مختلف داده را به آن اضافه می کنیم. سپس همه مقادیر را با استفاده از دستور foreach می خوانیم. چون کلاس ArrayList دارای انواع داده ای مختلفی است نمی توانیم از یک نوع داده ای خاص برای خواندن مقادیر استفاده کنیم . لذا برای این کار باید از نوع Object که می تواند هر نوع داده ای در خود ذخیره کند استفاده نمود. به این نکته توجه کنید که برای دسترسی به هر عنصر می توانید از طریق اندیس آن اقدام نمایید. کد زیر نحوه استفاده از حلقه for برای دسترسی به هر یک از اعضا را نشان می دهد.

```
for (int i = 0; i < myArray.size(); i++)
{
    System.out.println(myArray.get(i));
}
```

به متد size() در کد بالا توجه کنید. این متد درست شبیه به خاصیت length آرایه معمولی است و کار آن شمارش تعداد عناصر شی ArrayList می باشد.. در کد بالا همانطور که نشان داده شده است می توان به هر یک از عناصر با استفاده از اندیس شان دست یافت .(get(i)) نکته دیگر این است که شما می توانید به کلاس ArrayList یک ظرفیت ابتدایی بدهید. به عنوان مثال شما می توانید با استفاده از یک سازنده سربارگذاری شده نشان دهید که یک شی ArrayListمی تواند دارای 5 عنصر باشد.

```
ArrayList myArray = new ArrayList(5);
```

کد بالا 5 مکان خالی به وجود می آورد و شما می توانید با استفده از متد add()یکی دیگر به آنها اضافه کنید. می توان با استفاده از متد remove() کلاس ArrayList عناصر را پاک کرد. متد remove() یک شی که مطابق مقدار یک عنصر در آرایه است را قبول می کند. این متد به محض رسیدن به مقدار مورد نظر آن را حذف می کند. اگر عنصری را که مکانی غیر از مکان آخر آرایه باشد حذف کنید بقیه عناصر بعد از آن عنصر مکان خود را تنظیم می کنند به این معنی که فرض کنید آرایه ای دارای 5 عنصر است و شما عنصر 3 را حذف می کنید، در این صورت جای خالی این عنصر توسط عنصر 4 و جای عنصر 4 توسط عنصر 5 پر می شود .به تکه کد زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  import java.util.ArrayList;
 4:  import java.text.MessageFormat;
 5:
 6:  public class MyFirstProgram
 7:  {
 8:      public static void main(String[] args)
 9:      {
10:          ArrayList myArray = new ArrayList();
11:
12:          myArray.add("John");
13:          myArray.add(5);
14:          myArray.add(true);
15:          myArray.add(3.65);
16:          myArray.add('R');
17:
18:          for (int i = 0; i < myArray.size(); i++)
19:          {
20:              System.out.println(MessageFormat.format("myArray[{0}] = {1}", i,
21:  myArray.get(i)));
22:          }
23:
```

```
24:         myArray.remove(2);
25:
26:         System.out.println("\nAfter removing myArray[1] (The value true)...\n");
27:
28:         for (int i = 0; i < myArray.size(); i++)
29:         {
30:             System.out.println(MessageFormat.format("myArray[{0}] = {1}", i,
31: myArray.get(i)));
32:         }
33:     }
34: }
```

```
myArray[0] = John
myArray[1] = 5
myArray[2] = true
myArray[3] = 3.65
myArray[4] = R

After removing myArray[2] (The value 5)...

myArray[0] = John
myArray[1] = 5
myArray[2] = 3.65
myArray[3] = R
```

از آنجاییکه در مثال بالا مقدار عنصر myArray[2]را حذف کرده ایم (خط 23) همه عناصر متوالی در آرایه بالا مکان خود را تغییر می دهند. بنابراین عنصر myArray[3]جای myArray[2]، عنصر myArray[4]جای myArray[3]و ...را می گیرد.

جستجو، جایگزینی و به دست آوردن اندیس مقادیر

با استفاده از متد contains() می توان چک کرد که آیا یک مقدار خاص در داخل آرایه وجود دارد یا خیر. این متد یک آرگومان از نوع شی را قبول کرده و اگر یک مقدار را در داخل لیست عناصر پیدا کند true را بر می گرداند، از متدهای indexOf() و lastIndexOf()برای تشخیص اندیس یک مقدار خاص استفاده می شود.

متد IndexOf() اندیس اولین محل وقوع یک مقدار خاص را بر می گرداند.

متد LastIndexOf() اندیس آخرین محل وقوع یک مقدار خاص را بر می گرداند.

هر دو متد، در صورتیکه مقدار مورد نظر را پیدا نکنند مقدار 1- را بر می گردانند.

برای جایگزین کردن یک مقدار با یک مقدار موجود در ArrayList از متد set()استفاده می شود. فرض کنید که می خواهید عدد 15 را جایگزین عدد 5 در مثال بالا کنید برای اینکار باید به صورت زیر عمل کنید:

```
myArray.set(2,15);
```

عدد 2 در مثال بالا نشان دهنده اندیس مقداری از ArrayList است که ما می خواهیم مقداری دیگر را جایگزین آن کنیم. در این مثال اندیس 2 نشان دهنده مقدار 5 است. برای به دست آوردن تعداد عناصر در یک Arraylist از متد size()

```
myArray.size()
```

و برای پاک کردن همه عناصر از متد clear() استفاده می شود:

```
myArray.clear();
```

مرتب سازی مقادیرArrayList

مرتب سازی در ArrayList زمانی معنا دارد که همه آیتم ها از یک نوع مثلا عدد صحیح باشند. برای ایجاد یک کلاس ArrayListاز یک نوع خاص به صورت زیر عمل می شود:

```
ArrayList <type> CollectionName = new ArrayList();
```

که در آن type نوع داده ای مجموعه و CollectionName نامی است که برای مجموعه انتخاب کرده ایم. فرض کنید که می خواهیم یک مجموعه از نوع اعداد صحیح ایجاد و مرتب کنیم. به مثال زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  import java.util.ArrayList;
 4:  import java.util.Collections;
 5:  import java.text.MessageFormat;
 6:
 7:  public class MyFirstProgram
 8:  {
 9:      public static void main(String[] args)
10:      {
11:          ArrayList <Integer> myArray = new ArrayList();
12:
13:          myArray.add(1);
14:          myArray.add(5);
15:          myArray.add(3);
16:          myArray.add(2);
17:
18:          Collections.sort(myArray);
19:
20:          for (int i = 0; i < myArray.size(); i++)
21:          {
22:              System.out.println(MessageFormat.format("myArray[{0}] = {1}", i,
23:  myArray.get(i)));
24:          }
25:      }
26:  }
myArray[0] = 1
myArray[1] = 2
```

```
myArray[2] = 3
myArray[3] = 5
```

همانطور که در مثال بال مشاهده می کنید در خط 4 پکیج Collections را وارد برنامه کرده ایم. وجود این پکیج برای مرتب سازی الزامی است. در خط 11 یک مجموعه از نوع اعداد صحیح ایجاد و در خطوط 16-19 به صورت نامرتب چند آیتم به آن اضافه کرده ایم. سپس در خط 18 با استفاده از متد sort() کلاس Collectionsمجموعه را مرتب نموده ایم. با استفاده از متد sort() می توان مقادیر یک آرایه را مرتب نمود. اعداد از بزرگ به کوچک و رشته بر اساس حروف الفبا مرتب می شوند. اگر از این متد استفاده کنید همه اجزا با هم مقایسه می شوند. به عنوان مثال نمی توان یک رشته و یک عدد از نوع int را در داخل ArrayList قرار داد و آنها را با متد Sortمرتب نمود. در درس آینده یاد خواهید گرفت که چگونه از یک مقایسه گر سفارشی برای مرتب کردن عناصر استفاده نمود.

جنریک ها(Generics)

جنریک ها کلاسها، متدها یا رابط هایی هستند که بسته به نوع داده ای که به آنها اختصاص داده می شود رفتارشان را سازگار می کنند. به عنوان مثال می توان یک متد جنریک تعریف کرد که هر نوع داده ای را قبول کند .همچنین می توان یک متد ایجاد کرد که بسته به نوع دریافتی، مقادیری از انواع داده ای مانند int، doubleیا Stringرا نشان دهد. اگر از جنریک ها استفاده نکنید باید چند متد و یا حتی چندین متد سربارگذاری شده برای نمایش هر نوع ممکن ایجاد کنید.

```
public void Show(int number)
{
    System.out.println(number);
}

public void Show(double number)
{
    System.out.println(number);
}

public void Show(String message)
{
    System.out.println(message);
}
```

با استفاده از جنریک ها می توان متد جنریکی ایجاد کرد که هر نوع داده ای را قبول کند.

```
public <E> void Show(E item)
{
    System.out.println(item);
}
```

متدهای جنریک را در درسهای آینده توضیح خواهیم داد. حتما این سوال را از خودتان می پرسید که چرا نباید از نوع آبجکت که هر نوع داده ای را قبول می کند استفاده کنیم؟ در آینده مشاهده می کنید که با استفاده از جنریک ها نیاز به عمل cast (تبدیل صریح) ندارید. درباره جنریک ها در درسهای بعد مطالب بیشتری توضیح می دهیم.

متدهای جنریک

اگر بخواهید چندین متد با عملکرد مشابه ایجاد کنید و فقط تفاوت آنها در نوع داده ای باشد که قبول می کنند (مثلا یکی نوع int و دیگری نوع doubleرا قبول کند) می توان از متدهای جنریک برای صرفه جویی در کدنویسی استفاده کرد. ساختار عمومی یک متد جنریک به شکل زیر است:

```
<type> returnType methodName(type argument1)
{
    type someVariable;
}
```

مشاهده می کنید که قبل از نوع برگشتی متد یک نوع در داخل دو علامت بزرگتر و کوچکتر آمده است (<type>) که همه انواع در جاوا می توانند جایگزین آن شوند. برنامه زیر مثالی از نحوه استفاده از متد جنریک می باشد:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static <X> void Show(X val)
    {
        System.out.println(val);
    }

    public static void main(String[] args)
    {
        int intValue = 5;
        double doubleValue = 10.54;
        String stringValue = "Hello";
        boolean boolValue = true;

        Show(intValue);
        Show(doubleValue);
        Show(stringValue);
        Show(boolValue);
    }
```

```
5
10.54
Hello
true
```

یک متد جنریک ایجاد کرده ایم که هر نوع داده ای راقبول کرده و مقادیر آنها را نمایش می دهد (خطوط 8-5). سپس داده های مختلفی با وظایف یکسان به آن ارسال می کنیم. متد نیز نوع Xرا بسته به نوع داده ای که به عنوان آرگومان ارسال شده است تغییر می دهد. به عنوان مثال وقتی یک داده از نوع intارسال می کنیم، همه مکانهایی که Xدر آنها وجود دارد به intتبدیل می شوند و متد به صورت زیر در می آید:

```
public static void Show (int val)
{
```

```
    System.out.println(val);
}
```

به یک نکته در مورد استفاده از متدهای جنریک توجه کنید و آن این است که شما نمی توانید در داخل کدهای مربوط به متد محاسبات انجام دهید مثلا دو عدد را با هم جمع کنید چون کامپایلر نمی تواند نوع واقعی عملوندها را تشخیص دهد، ولی به سادگی می توان مقادیر را در داخل متد نشان داد چون کامپایلر هر نوع داده ای را که توسط متد System.out.println() استفاده می شود را می تواند تشخیص دهد.

```
public static <X> void Show(X val1, X val2)
{
    System.out.println(val1 + val2);
}
```

شما می توانید چندین نوع خاص را برای متد جنریک ارسال کنید، برای این کار هر نوع را به وسیله کاما از دیگری جدا کنید.

```
public static <X, Y> void Show(X val1, Y val2)
{
    System.out.println(val1);
    System.out.println(val2);
}
```

به مثال زیر که در آن دو مقدار مختلف به متد ارسال شده است توجه کنید:

```
Show(5, true);
```

مشاهده می کنید که X با نوع int و Y با نوع bool جایگزین می شود. این نکته را نیز یادآور شویم که شما می توانید دو آرگومان هم نوع را هم به متد ارسال کنید:

```
Show(5, 10);
```

کلاس جنریک

تعریف یک کلاس جنریک بسیار شبیه به تعریف یک متد جنریک است .کلاس جنریک دارای یک علامت بزرگتر و کوچک تر و یک نوع پارامتر خاص می باشد .برنامه زیر مثالی از یک کلاس جنریک می باشد:

```
1:  package myfirstprogram;
 2:
 3:  import java.text.MessageFormat;
 4:
 5:  class GenericClass
 6:  {
 7:      private T someField;
```

```
 8:
 9:      public GenericClass(T someVariable)
10:      {
11:          someField = someVariable;
12:      }
13:
14:      public void SetSomeProperty(T value)
15:      {
16:         this.someField = value;
17:      }
18:
19:      public T GetSomeProperty()
20:      {
21:        return someField;
22:      }
23:  }
24:
25:  public class MyFirstProgram
26:  {
27:      public static void main(String[] args)
28:      {
29:          GenericClass genericDouble = new GenericClass(30.50);
30:          GenericClass genericString = new GenericClass("Hello World!");
31:
32:          System.out.println(MessageFormat.format("Double Value : {0}",
33:              genericDouble.GetSomeProperty()));
34:          System.out.println(MessageFormat.format("String Value : {0}",
35:              genericString.GetSomeProperty()));
36:      }
37:  }
Double Value : 30.5
String Value : Hello World!
```

در مثال بالا یک کلاس جنریک (خطوط 23-5) که دارای یک فیلد (خط 7)، یک خاصیت (خطوط 22-14) و یک سازنده (خطوط 12-9) است را ایجاد می کنیم. تمام مکانهایی که ورودی Tدر آنها قرار دارد بعدا توسط انواعی که مد نظر شما است جایگزین می شوند. وقتی یک نمونه از کلاس جنریک تان ایجاد می کنید، یک نوع هم برای آن در نظر بگیرید .(<int>) مانند متدهای جنریک می توانید چندین نوع پارامتر به کلاسهای جنریک اختصاص دهید.

```
public class GenericClass<T1, T2, T3>
{
    private T1 someField1;
    private T2 someField2;
    private T3 someField3;
}
```

چون نمی دانید T1، T2و T3از چه نوعی هستند نمی توانید مانند مثال زیر از آنها نمونه جدید ایجاد کنید.

```
public GenericClass //Constructor
{
    someField1 = new T1();
    someField2 = new T2();
    someField3 = new T3();
}
```

کلاسهای غیر جنریک می توانند از کلاسهای جنریک ارث بری کنند، اما باید یک نوع برای پارامترکلاس پایه جنریک تعریف کنید.

```
public class MyClass extends GenericClass<Integer>
{

}
```

یک کلاس جنریک هم می تواند از یک کلاس غیر جنریک ارث بری کند.

Iterator و ListIterator

برای دسترسی، ویرایش و حذف هر یک از عناصر یک کلکسیون ابتدا باید عنصر مورد نظر را پیدا کنیم. برای این کار لازم است که در میان عناصر بگردیم. سه راه برای گردش در میان عناصر یک کلکسیون یا مجموعه وجود دارد:

1. با استفاده از رابطIterator
2. با استفاده از رابطListIterator
3. با استفاده از حلقهforeach

دسترسی به عناصر مجموعه با استفاده از رابطIterator

رابط Iterator یک مجموعه یا کلکسیون را رو به جلو پیمایش کرده و شما را قادر می سازد که عناصر را حذف و ویرایش کنید. هر کلکسیون برای ایجاد یک پیمایشگر دارای متدی به نام iterator() می باشد. وقتی یک شیء از Iterator می سازیم می توانیم به متدهای آن که در جدول زیر آمده اند نیز دست یابیم:

متد	توضیح
hasNext	چک می کند که آیا عنصری بعد از عنصر فعلی وجود دارد یا نه؟
next	عنصر بعدی را بر می گرداند.

به مثال زیر توجه کنید:

```
package myfirstprogram;

import java.util.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayList<String> arraylist = new ArrayList<>();
```

```
        arraylist.add("A");
        arraylist.add("B");
        arraylist.add("C");
        arraylist.add("D");

       Iterator iterator = arraylist.iterator();     //Declaring Iterator

        while(iterator.hasNext())
        {
          System.out.println(iterator.next());
        }
    }
}
```

```
A
B
C
D
```

مهمترین خط کد بالا خط زیر است:

```
Iterator iterator = arraylist.iterator();
```

در کد بالا با فراخوانی متد iterator() از کلسیونمان که در اینجا arraylist است آن را به مجموعه ای قابل پیمایش توسط Itratorمی کنیم و سپس با استفاده از متدهای این رابط آن را پیمایش می کنیم.

دسترسی به عناصر مجموعه با استفاده از رابطListIterator

رابط ListIterator یک مجموعه را با استفاده از متدهایی که در جدول زیر آمده اند، هم رو به جلو و هم رو به عقب پیمایش می کند:

متد	توضیح
hasNext	چک می کند که آیا عنصری بعد از عنصر فعلی وجود دارد یا نه؟
next	عنصر بعد از عنصر فعلی را بر می گرداند.
hasPrevious	چک می کند که آیا عنصری قبل از عنصر فعلی وجود دارد یا نه؟
previous	عنصر قبل از عنصر فعلی را بر می گرداند.

این رابط فقط در مجموعه هایی قابل دسترسی است که رابط List را پیاده سازی کرده باشند:

```
package myfirstprogram;

import java.util.*;
```

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayList<String> arraylist = new ArrayList<>();

        arraylist.add("A");
        arraylist.add("B");
        arraylist.add("C");
        arraylist.add("D");

        ListIterator listiterator = arraylist.listIterator();

        while(listiterator.hasNext())                              //In forward direction
        {
         System.out.println(listiterator.next());
        }

        System.out.println("\n\n");

        while(listiterator.hasPrevious())                         //In backward direction
        {
         System.out.println(listiterator.previous());
        }
    }
}
```

```
A
B
C
D


D
C
B
A
```

دسترسی به عناصر مجموعه با استفاده از حلقهforeach

Foreachنسخه ای از حلقه for است که می تواند در میان عناصر یک مجموعه گردش کند. از این حقله نمی توان برای ویرایش عناصر یک مجموعه استفاده کرد .حلقه foreach می تواند در میان هر مجموعه ای از اشیاء که رابط Iterable را پیاده سازی کرده باشد گردش کند:

```
package myfirstprogram;

import java.util.*;

public class MyFirstProgram
{
```

```
    public static void main(String[] args)
    {
        ArrayList<String> arraylist = new ArrayList<>();

        arraylist.add("A");
        arraylist.add("B");
        arraylist.add("C");
        arraylist.add("D");

        for(String str : arraylist)
        {
         System.out.println(str);
        }
    }
}
A
B
C
D
```

شمارش(Enumeration)

شمارش راهی برای تعریف داده هایی است که می توانند مقادیر محدودی که شما از قبل تعریف کرده اید را بپذیرند. به عنوان مثال شما می خواهید یک متغیر تعریف کنید که فقط مقادیر جهت (جغرافیایی)مانندeast ، west، north و south را در خود ذخیره کند. ابتدا یک enumeration تعریف می کنید و برای آن یک اسم انتخاب کرده و بعد از آن تمام مقادیر ممکن که می توانند در داخل بدنه آن قرار بگیرند تعریف می کنید. به نحوه تعریف یک enumerationتوجه کنید:

```
enum enumName
{
   value1,
   value2,
   value3,
   .
   .
   .
   valueN
}
```

ابتدا کلمه کلیدی enum و سپس نام آن را به کار می بریم. در جاوا برای نامگذاری enumeration از روش پاسکال استفاده کنید. در بدنه enum مقادیری وجود دارند که برای هر کدام یک نام در نظر گرفته شده است و به وسیله کاما از هم جدا شده اند. به یک مثال توجه کنید:

```
enum Direction
{
```

```
    North,
    East,
    South,
    West
}
```

به نحوه استفاده از enumeration در یک برنامه جاوا توجه کنید.

```
 1:  package myfirstprogram;
 2:
 3:  import java.text.MessageFormat;
 4:
 5:  enum Direction
 6:  {
 7:      North,
 8:      East,
 9:      South,
10:      West
11:  }
12:
13:  public class MyFirstProgram
14:  {
15:      public static void main(String[] args)
16:      {
17:          Direction myDirection;
18:
19:          myDirection = Direction.North;
20:
21:          System.out.println(MessageFormat.format("Direction: {0}", myDirection));
22:      }
23:  }
Direction: North
```

ابتدا enumeration را در خطوط 5-11 تعریف می کنیم. توجه کنید که enumerationرا خارج از کلاس قرار داده ایم. این کار باعث می شود که enumerationدر سراسر برنامه در دسترس باشد. می توان enumeration را در داخل کلاس هم تعریف کرد ولی در این صورت فقط در داخل کلاس قابل دسترس است.

```
public class MyFirstProgram
{
    enum Direction
    {
        //Code omitted
    }

    public static void main(String[] args)
    {
        //Code omitted
    }
}
```

برنامه را ادامه می دهیم. در داخل بدنه enumeration نام چهار جهت جغرافیایی وجود دارد (خطوط 7-10). در خط 17 یک متغیر تعریف شده است که مقدار یک جهت را در خود ذخیره می کند. نحوه تعریف آن به صورت زیر است:

```
enumType variableName;
```

در اینجا enuType نوع داده شمارشی (مثلا Direction یا مسیر) می باشد و variableNameنیز نامی است که برای آن انتخاب کرده ایم که در مثال قبل myDirectionاست. سپس یک مقدار به متغیر myDirection اختصاص می دهیم (خط 19) . برای اختصاص یک مقدار به صورت زیر عمل می کنیم:

```
variable = enumType.value;
```

ابدا نوع Enumeration سپس علامت نقطه و بعد مقدار آن (مثلاNorth) را می نویسیم. می توان یک متغیر را فورا ،به روش زیر مقدار دهی کرد:

```
Direction myDirection = Direction.North;
```

حال در خط 21 با استفاده از println() مقدار myDirection را چاپ م کنیم enum .ها مانند کلاس ها و اینترفیس ها از انواع ارجاعی به حساب می آیند و بنابر این می توانند دارای سازنده و فیلد و متد باشند. به هر عضو از یک enum می توان یک عدد اختصاص داد. به مثال زیر توجه کنید:

```
1:  package myfirstprogram;
 2:
 3:  enum Direction
 4:  {
 5:     North(3),
 6:     East(5),
 7:     South(7),
 8:     West(4);
 9:
10:     private final int directionindex;
11:
12:     Direction(int index)
13:     {
14:          this.directionindex = index;
15:     }
16:
17:     public int GetDirectionIndex()
18:     {
19:          return this.directionindex;
20:     }
21:  }
22:
23:  public class MyFirstProgram
24:  {
25:      public static void main(String[] args)
26:      {
27:          Direction myDirection = Direction.East;
28:          System.out.println(myDirection.GetDirectionIndex());
29:      }
30:  }
5
```

همانطور که در کد بالا مشاهده می کنید یک نوع شمارشی با نام Direction در خطوط 21-3 تعریف کرده ایم. به مقادیر این نوع شمارشی در خطوط 8-5 مقادیری را اختصاص داده ایم. در خط 10 یک فیلد با نام directionindex تعریف کرده ایم تا مقدار هر یک از عناصر شمارشی را که می خواهیم به وسیله سازنده ای که در خطوط 15-12 تعریف شده است در آن قرار دهیم. برای به دست آوردن این مقادیر هم از متد GetDirectionIndex() (خطوط 20-17) استفاده می کنیم .حال فرض کنید می خواهیم مقدار عددی که به East اختصاص داده شده است را به دست آوریم. برای این کار همانطور که در خط 27 مشاهده می کنید ابتدا یک نمونه از Direction ایجاد کرده و Direction.East را به آن اختصاص می دهیم، سپس در خط 28 با فراخوانی متد GetDirectionIndex()مقدار عددی East را به دست می آوریم. ار همین عدد 5 به دست آمده در مثال بالا می توان در ساختارهایی مانند if و switch استفاده کرد:

```
Direction myDirection = Direction.East;

switch(myDirection.GetDirectionIndex())
{
    case 1 : System.out.println("incorrect!");
        break;
    case 5 : System.out.println("That is Correct.");
        break;
}
```

کلاس های تو در تو(nested classes)

به کلاسی که در داخل کلاس دیگر تعریف شود کلاس تو در تو گفته می شود. از کلاس های تو در تو برای گروه بندی منطقی کلاس ها در یک مکان استفاده می شود، با این کار خوانایی کدها بیشتر و دستکاری آنها راحت تر می شود. کلاس تو در تو عضوی از کلاسی است که در داخل آن قرار دارد، بنابراین می تواند به صورتpublic ، protectو private تعریف شود. این کلاس ها می توانند توسط زیر کلاس (subclass) هم به ارث برده شوند. نحوه ایجاد یک کلاس تو در تو به صورت زیر می باشد.

```
class OuterClass
{
    class InnerClass
    {

    }
}
```

در جاوا چند نوع کلاس تو در تو وجود دارد که در زیر به آنها اشاره شده است:

- کلاس های داخلی استاتیک
- کلاس های داخلی غیر استایک

- کلاس های محلی
- کلاس های بی نام

در درس های آینده در باره این کلاس ها توضیح می دهیم.

کلاس داخلی استاتیک و غیر استاتیک

همانطور که در درس قبل اشاره شده در جاوا 4 نوع کلاس تو در تو وجود دارد که در این درس به دو نوع از آنها می پردازیم.

کلاس های داخلی استاتیک

این نوع کلاس ها به صورت زیر تعریف می شوند:

```
public class Outer
{
    public static class Nested
    {

    }
}
```

برای ایجاد شیء از این نوع کلاس ها ابتدا باید نام کلاس بیرونی و سپس علامت نقطه و بعد نام کلاس داخلی استاتیک را بنویسید. به کد زیر توجه کنید:

```
Outer.Nested instance = new Outer.Nested();
```

یک کلاس داخلی استاتیک یک کلاس عادی است که در داخل یک کلاس دیگر قرار دارد. کلاس های استاتیک داخلی فقط به اعضای استاتیک کلاسی که در آن قرار دارند ، دسترسی دارند. به مثال زیر توجه کنید:

```
package myfirstprogram;

class Outer
{
  static String  message = "Hello World!";

  public static class Nested
  {
      void ShowMessage()
      {
          System.out.println(message);
```

```
      }
  }
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Outer.Nested instance = new Outer.Nested();

        instance.ShowMessage();
    }
}
```

```
Hello World!
```

همانطور ک در مثال بالا متغیر message یک عضو استاتیک از کلاس Outer است و برای دسترسی به آن باید به صورتی که در خطوط مشاهده می کنید، عمل نمایید.

کلاس های داخلی غیر استاتیک یاInner Classes

کلاس های داخلی غیر استاتیک در جاوا به نام inner classes معروفند. برای ایجاد یک نمونه از این کلاس ها ابتدا باید یک نمونه از کلاس خارجی ایجاد کنید. نحوه تعریف این نوع کلاس ها به صورت زیر است:

```
public class Outer
{
    public class Inner
    {

    }
}
```

در زیر نحوه ایجاد نمونه از یک کلاس داخلی آمده است:

```
Outer outer = new Outer();

Outer.Inner inner = outer.new Inner();
```

به نحوه قرار داده کلمه new بعد از شیء ایجاد شده از کلاس خارجی در کد بالا توجه کنید. کلاس داخلی غیر استاتیک به فیلدهای کلاس خارجی دسترسی دارد ، حتی اگر این فیلدها به صورت private تعریف شده باشند. به کد زیر توجه کنید:

```
package myfirstprogram;

class Outer
{
    private String text = "I am private!";

    public class Inner
    {
```

```
        public void ShowMessage()
        {
            System.out.println(text);
        }
    }
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Outer outerClass = new Outer();
        Outer.Inner innerClass =  outerClass.new Inner();

        innerClass.ShowMessage();
    }
}
```

```
I am private!
```

کلاس های محلی(Local Classes)

کلاس های محلی در جاوا شبیه به کلاس های داخلی غیر استاتیک (inner class)هستند. این نوع کلاس ها در داخل یک متد یا محدوده ({...}) در داخل متد تعریف می شوند. به مثال زیر توجه کنید:

```
class Outer
{
    public void printText()
    {
        class Local
        {

        }

        Local local = new Local();
    }
}
```

کلاس های محلی تنها در داخل متد و یا بلوکی که در آن تعریف شده اند قابل دسترسی هستند. این کلاس ها همانند کلاس های غیر استاتیک می توانند به فیلدها و متدهای کلاسی که در داخل آن قرار دارند دسترسی داشته باشند. از نسخه 8 جاوا این کلاس ها می توانند به متغیرهای محلی و پارامترهای متدی که در آن تعریف شده اند دسترسی داشته باشند . پارامترها باید به صورت final تعریف شده باشند. کلاس های محلی می توانند در داخل متدهای استاتیک تعریف شوند که در این صورت فقط به قسمت های استاتیک کلاسی که در آن تعریف شده اند دسترسی خواهند داشت. این کلاس ها نمی توانند دارای

انواع استاتیک باشند (می توانند شامل ثابت ها باشند و متغیرهای آنها به صورت static final تعریف می شوند)، چون کلاس های محلی در حالت عادی غیر استاتیک هستند حتی اگر در داخل متد استاتیک تعریف شوند.