جاوا به زبان ساده

سر شناسه	: ابراهیمی، یونس - ۱۳۶۰
عنوان و نام پدید آورنده	: جاوا به زبان ساده / مؤلف: یونس ابراهیمی
مشخصات نشر	: نبض دانش، تهران - ۱۳۹۶
مشخصات ظاهری	: ۶۱۶ صفحه مصور
شابک	:
وضعیت فهرست نویسی	: فیپا
موضوع	: جاوا (زبان برنامه نویسی کامپیوتر)
رده بندی دیویی	:
رده بندی کنگره	:
شماره کتاب شناسی ملی	:

عنوان	: جاوا به زبان ساده
مؤلف	: یونس ابراهیمی
ناشر	: نبض دانش
سال چاپ	: ۱۳۹۶
نوبت چاپ	: دوم
تیراژ	: ۲۰۰
قیمت	: ۵۹۰۰۰ تومان

تقدیم به

همسر و پسر عزیزم

مبانی زبان جاوا

SWING

کار با تاریخ، فایل و رشته

کار با بانک اطلاعاتی

مقدمه

همگام با پیشرفت فناوری‌های دیگر، زبان‌های برنامه نویسی نیز ارتقا پیدا کردند. وقتی زبان JAVA طراحی و پیاده سازی شد، تحول بزرگی در دنیای برنامه نویسی به وجود آمد. این زبان برنامه نویسی موفق که در سال ۱۹۹۵ به طور رسمی به بازار معرفی شد، توانست چنان محبوبیتی در جهان پیدا کند که در حال حاضر در بیش از ۳ میلیارد سیستم مورد استفاده قرار گرفته و تاکنون بیش از ۱۰۰۰ جلد کتاب پیرامون آن به رشته تحریر درآمده است. خوشبختانه JAVA این روزها به عنوان یکی از دروس رشته‌های کامپیوتر در دانشگاه‌های کشور تدریس می‌شود و این نشان از توانایی‌ها و اهمیت این زبان است. منابع متعددی برای معرفی و به کارگیری زبان JAVA عرضه شده است که جای تقدیر و تشکر دارد. اما کتاب حاضر دارای ویژگی‌های بارزی از جمله، بیان ساده مطالب، ارائه مثال‌های متنوع، بررسی دقیق و موشکافانه موضوعات و سلسله مراتب آموزشی است. مثال‌هایی که در کتاب ارائه شده‌اند همگی دارای هدف خاصی هستند به طوریکه هر کدام، یک یا چند نکته زبان جاوا را به خواننده آموزش می‌دهند. در این کتاب ما به شما نحوه برنامه نویسی به زبان جاوا را به صورت تصویری آموزش می‌دهیم. سعی کنید حتماً بعد از خواندن مباحث، آنها را به صورت عملی تمرین کنید و اینکه قابلیت و مفهوم کدها را بفهمید نه آنها را حفظ کنید.

برای دریافت فایل‌ها و آپدیت‌های جدید این کتاب به سایت **www.w3-farsi.com** مراجعه فرمایید.

راه‌های ارتباط با نویسنده

وب سایت	: www.w3-farsi.com
لینک تلگرام	: https://telegram.me/ebrahimi_younes
ID تلگرام	: @ebrahimi_younes
پست الکترونیکی	: younes.ebrahimi.1391@gmail.com

فصل اول

مبانی زبان جاوا

جاوا چیست؟

جاوا (JAVA) یک زبان برنامه نویسی شیءگراست که نخستین بار توسط James Gosling (جیمز گاسلینگ) در شرکت Sun Microsystems ایجاد گردید. در سال ۱۹۹۰ شرکت Sun Microsystems در حال توسعه نرم‌افزاری برای استفاده ابزارهای الکترونیکی بود که مسئولیت تیم، که آن را، تیم پروژه Green نامیدند، جیمز گاسلینگ بر عهده گرفت.

در سال ۱۹۹۱ تیم تصمیم گرفت که زبان جدید را OAK (بلوط) بنامند. علت این نام گذاری وجود درختان بلوط در محوطه اطراف ساختمان محل کار اعضای تیم Green بود. در سال ۱۹۹۲ تیم پروژه Green زبان جدیدی را معرفی کرد که با ابزارهای مختلف خانگی و لمسی کار می‌کرد. در سال ۱۹۹۳ وب جهانی توسعه یافت و زبان OAK با معرفی Applet که قابلیت های زیادی به کامپیوترهای متصل به وب می‌افزود، مشهور شد .

در سال ۱۹۹۵ زبان OAK به JAVA تغییر نام پیدا کرد و توسط Microsoft و Netscape پشتیبانی شد. از آنجا که مراسم تغییر نام در کافی شاپ برگزار شده بود و همچنین علاقه اعضای تیم Green به قهوه، یک فنجان قهوه داغ به عنوان نماد جاوا در نظر گرفته شد. معتبرترین داستان درباره دلیل این نامگذاری این است که، جیمز گاسلینگ به نوعی قهوه علاقه داشت، که در جزیره‌ای به نام جاوا، که در اندونزی در جنوب شرقی آسیا است، می روید. کلا زبان برنامه نویسی جاوا به سه دسته کلی تقسیم می شود:

- JAVA SE یا JAVA Standard Edition برای نوشتن برنامه های کوچک دسکتاپی کاربرد دارد.
- JAVA ME یا JAVA Micro Edition برای برنامه نویسی برای منابع سخت افزاری (CPU، MEMORY) محدود، مثل موبایل و لوازم خانگی کاربرد دارد.
- JAVA EE یا JAVA Enterprised Edition برای برنامه های بزرگ که معمولا بر روی شبکه های بزرگ مخصوصا اینترنت نصب و اجرا می شوند کاربرد دارد.

تاریخچه جاوا

از زمان انتشار اولین نسخه جاوا (java 1.0) تا به امروز، شرکت Sun تقریبا هر دو سال یکبار نسخه ای جدیدی از این زبان 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.x بود 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 (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:

نسخه	نام کد	تاریخ پیدایش
java 1.0	Oak	January 1996
java 1.1		February 1997
J2SE 1.2	playground	December 1998
J2SE 1.3	Kestrel	May 2000
J2SE 1.4	Merlin	February 2002
J2SE 5.0	Tiger	September 2004
Java SE 6	Mustang	December 2006
Java SE 7	Dolphin	July 2011
Java SE 8		March 2014

برای آشنایی بیشتر با این زبان به لینک های زیر مراجعه کنید:

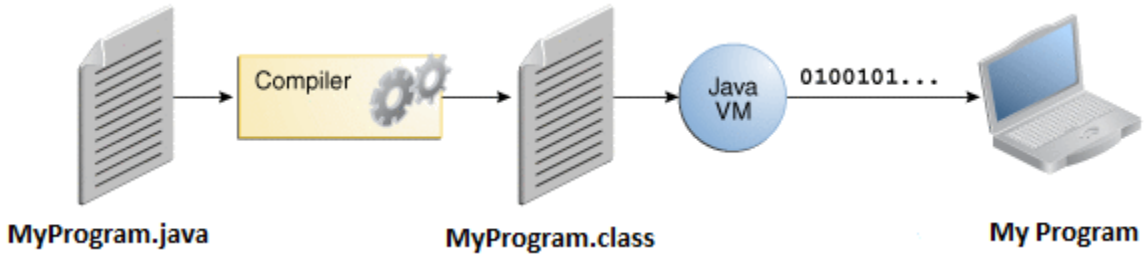
https://en.wikipedia.org/wiki/Java_(programming_language)

https://en.wikipedia.org/wiki/Java_version_history

ضریب اطمینان عملکرد برنامه‌های نوشته شده به این زبان بالا است و وابسته به سیستم عامل خاصی نیست، به عبارت دیگر می‌توان آن را روی هر رایانه با هر نوع سیستم عاملی اجرا کرد و این، همان شعار جاوا است: "یک بار بنویس، همه جا اجرا کن".

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 Platformاست.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. می توان نحوه اجرای کدهای جاوا را به صورت زیر خلاصه کرد:



همانطور که در شکل بالا مشاهده می کنید:

- برنامه‌نویس کدهای خود را درون فایلی با پسوند .java می‌نویسید.
- وقتی برنامه نویس برنامه خود را اجرا می کند، کدهای برنامه توسط کمپایلر جاوا به bytecode تبدیل می‌شوند و درون فایلی با همان نام قبلی اما این بار با پسوند .class ذخیره می‌شوند.

- ماشین مجازی جاوا (Java Virtual Machine) فایل class. را اجرا می‌کند.

ماشین مجازی جاوا یا JVM بر روی تمام سیستم‌عامل‌های مطرح (ویندوز، مکینتاش و لینوکس) قابل نصب است. به همین دلیل فایل class. برنامه شما در تمام این سیستم عامل‌ها می‌تواند اجرا شود و به همین دلیل است که به جاوا زبان مستقل از سیستم عامل گفته می‌شود. شعار جاوا این است: «یک بار بنویس، همه جا اجرا کن» !

NetBeans و JDK

Netbeans محیط توسعه یکپارچه ای است که دارای ابزارهایی برای کمک به شما برای توسعه برنامه‌های JAVA می‌باشد. توصیه می‌کنیم که از محیط Netbeans برای ساخت برنامه استفاده کنید، چون این محیط دارای ویژگی‌های زیادی برای کمک به شما جهت توسعه برنامه‌های JAVA می‌باشد. توسط Netbeans می‌توان در استانداردهای مختلف جاوا مانند J2SE، J2EE و J2ME برنامه نویسی کرد. همچنین از محیط زبان‌های PHP، HTML، C و نیز Groovy پشتیبانی می‌کند. قبل از نصب Netbeans می‌بایست JDK را نصب نمایید، در غیر این صورت برای نصب دچار مشکل خواهید شد. JDK که مخفف عبارت Java Development Toolkit می‌باشد ترکیبی از کمپایلر زبان جاوا، کلاس‌های کتابخانه ای (Java Class Libraries) و JVM و فایل‌های راهنمای آن‌ها می‌باشد. برای اینکه ما بتوانیم با استفاده از زبان برنامه نویسی جاوا، برنامه بنویسیم به این مجموعه نیاز داریم. تعداد زیادی از پردازش‌ها که وقت شما را هدر می‌دهند به صورت خودکار توسط NetBeans انجام می‌شوند. یکی از این ویژگی‌ها اینتلی سنس (Intellisense) است که شما را در تایپ سریع کدهایتان کمک می‌کند. NetBeans برنامه شما را خطایابی می‌کند و حتی خطاهای کوچک (مانند بزرگ یا کوچک نوشتن حروف) را برطرف می‌کند. با این برنامه‌های قدرتمند بازدهی شما افزایش می‌یابد و در وقت شما با وجود این ویژگیهای شگفت انگیز صرفه جویی می‌شود NetBeans آزاد است و می‌توان آن را دانلود و از آن استفاده کرد. این برنامه ویژگیهای کافی را برای شروع برنامه نویسی JAVA در اختیار شما قرار می‌دهد.در آموزش‌ها از NetBeans نسخه ۸/۰/۲ استفاده شده است و استفاده از این نسخه برای انجام تمرینات این سایت کافی می‌باشد. برای دانلود نرم افزارهای مورد نیاز به سایت **w3-farsi.com** و لینک زیر مراجعه کنید:

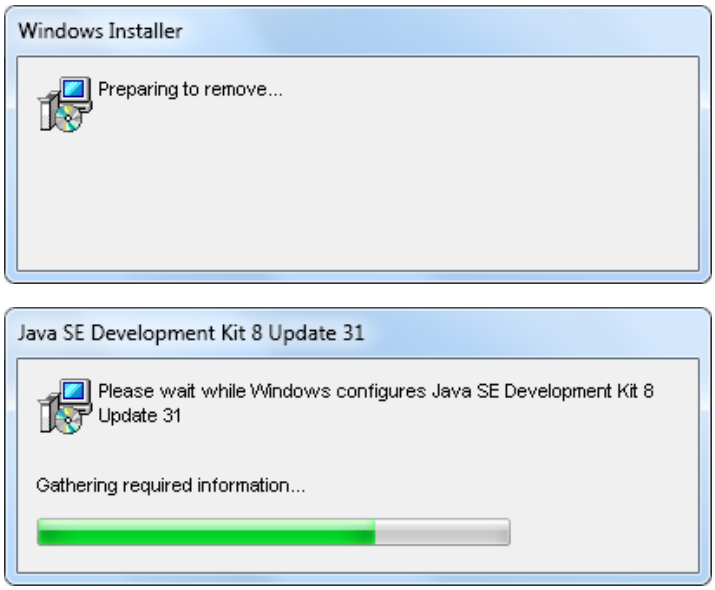
```
http://www.w3-farsi.com/?p=7529
```

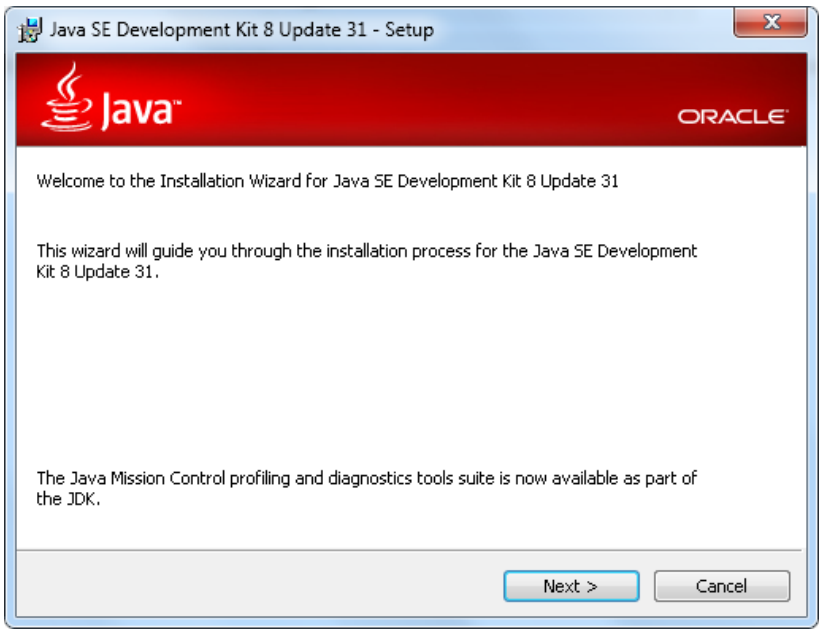
در درس آینده مراحل نصب و راه اندازی دو نرم افزار JDK و NetBeans را توضیح می‌دهیم.

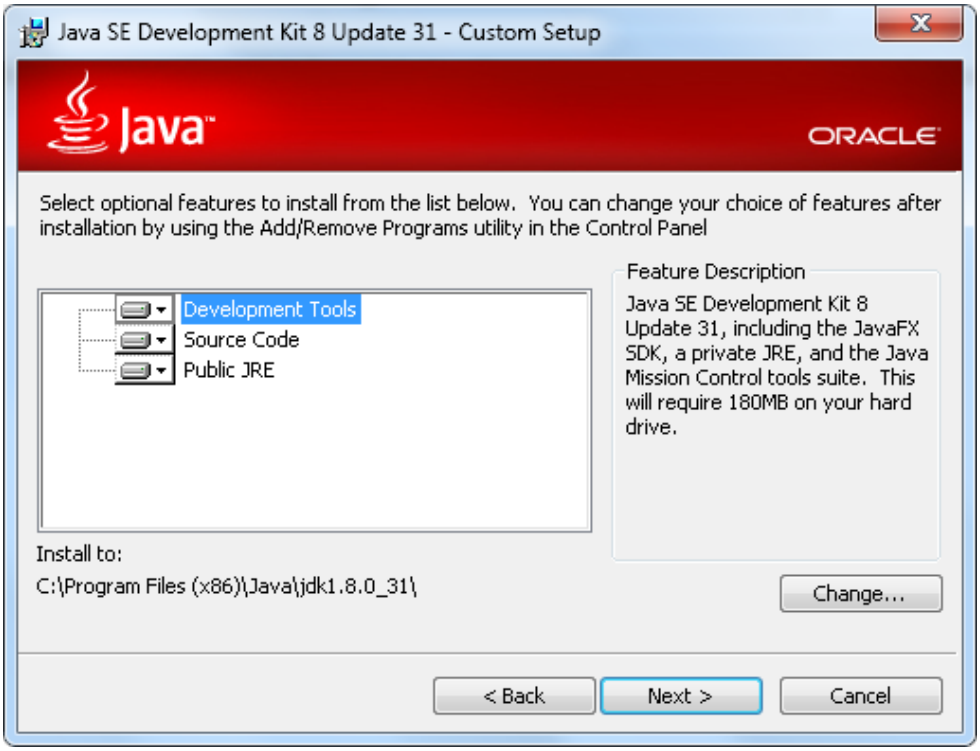
نصب NetBeans و JDK

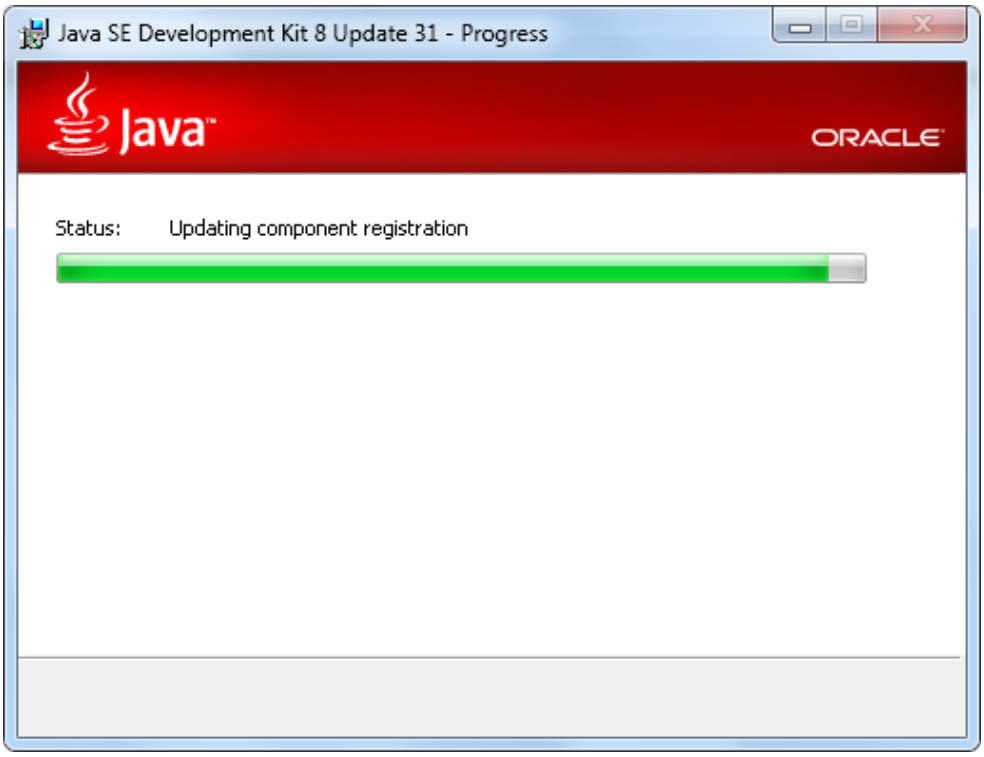
در درس قبل در مورد نرم افزارهای NetBeans و JDK توضیحات مختصری ارائه دادیم. در این درس می‌خواهیم شما را با نحوه نصب این دو نرم افزار آشنا کنیم. نصب این نرم افزارها مانند اکثر نرم افزارهای دیگر بسیار آسان بود و بعد از زدن چند دکمه Next نصب می‌شوند. در زیر مراحل تصویری نصب این دو نرم افزار نشان داده شده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

نصب JDK




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

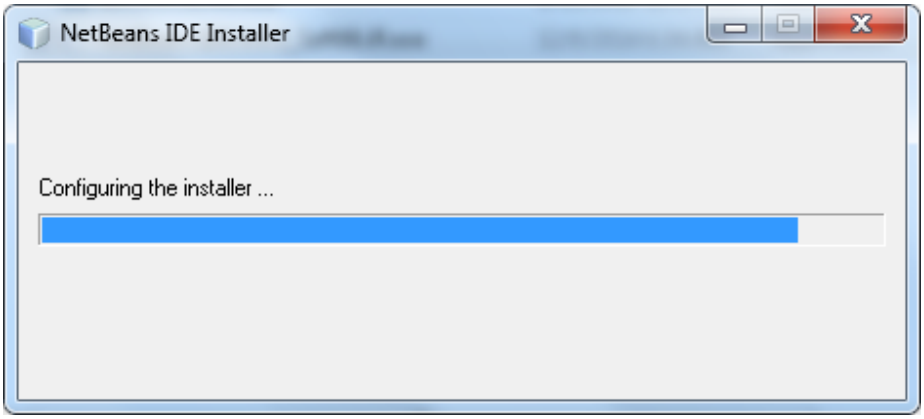

Java SE Development Kit 8 Update 31 - Progress
Java
ORACLE
Status:
Updating component registration

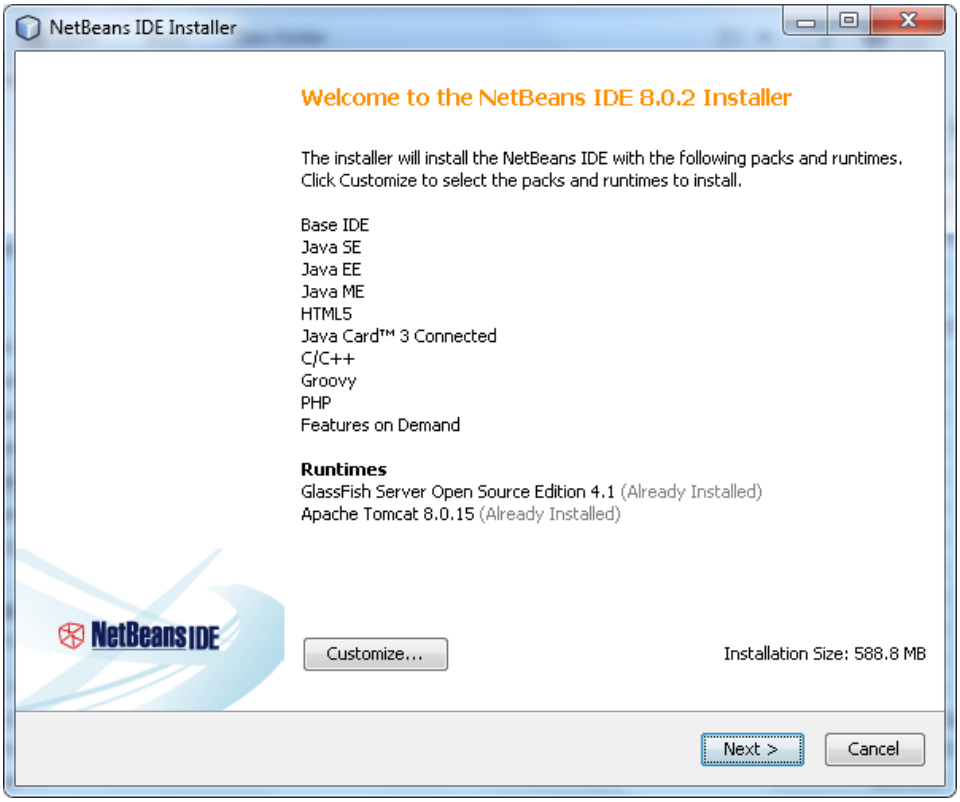

Java Setup - Destination Folder
Java
ORACLE
Destination Folder
Click "Change" to install Java to a different folder.
Install to:
C:\Program Files (x86)\Java\jre1.8.0_31\
Change...
Next >

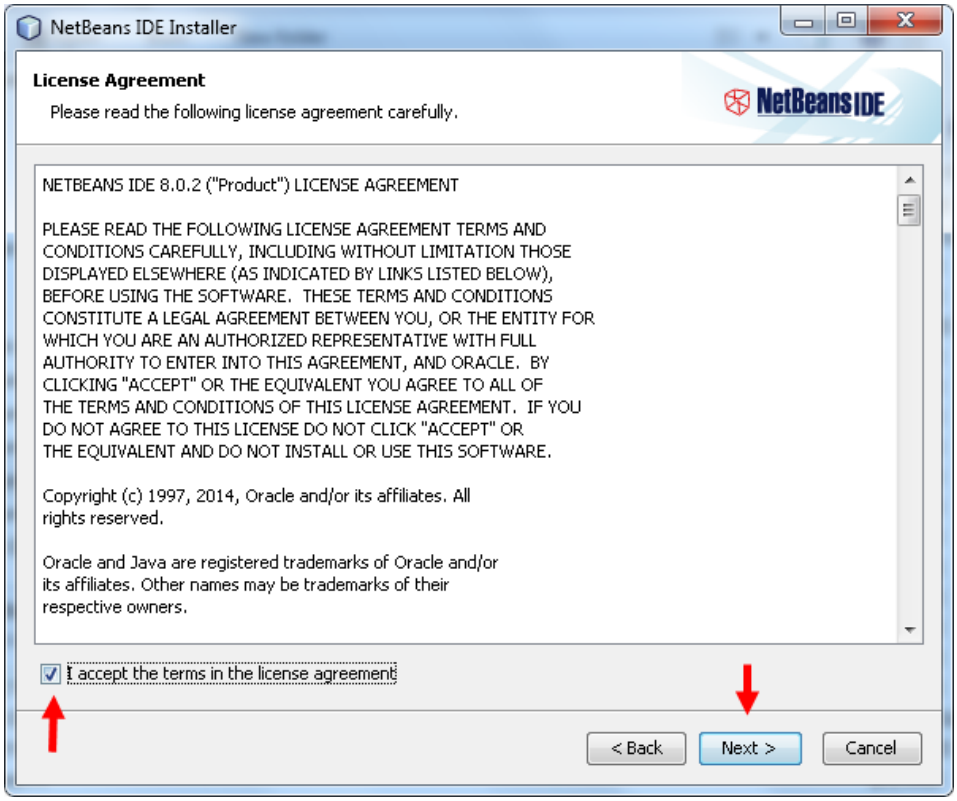

Java Setup - Progress
Java
ORACLE
Status: Installing Java
3 Billion Devices Run Java
Computers, Printers, Routers, Cell Phones, BlackBerry, Kindle, Parking Meters, Public Transportation Passes, ATMs, Credit Cards, Home Security Systems, Cable Boxes, TVs...
ORACLE

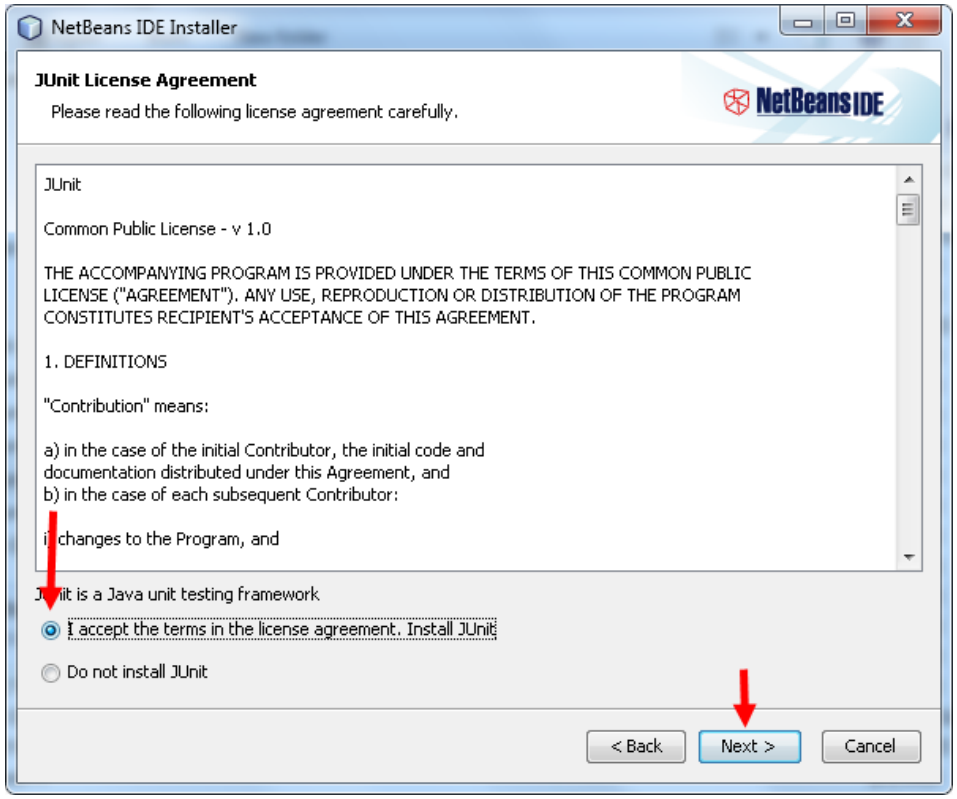

Java SE Development Kit 8 Update 31 - Complete
Java
ORACLE
Java SE Development Kit 8 Update 31 Successfully Installed
Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.
Next Steps
Close

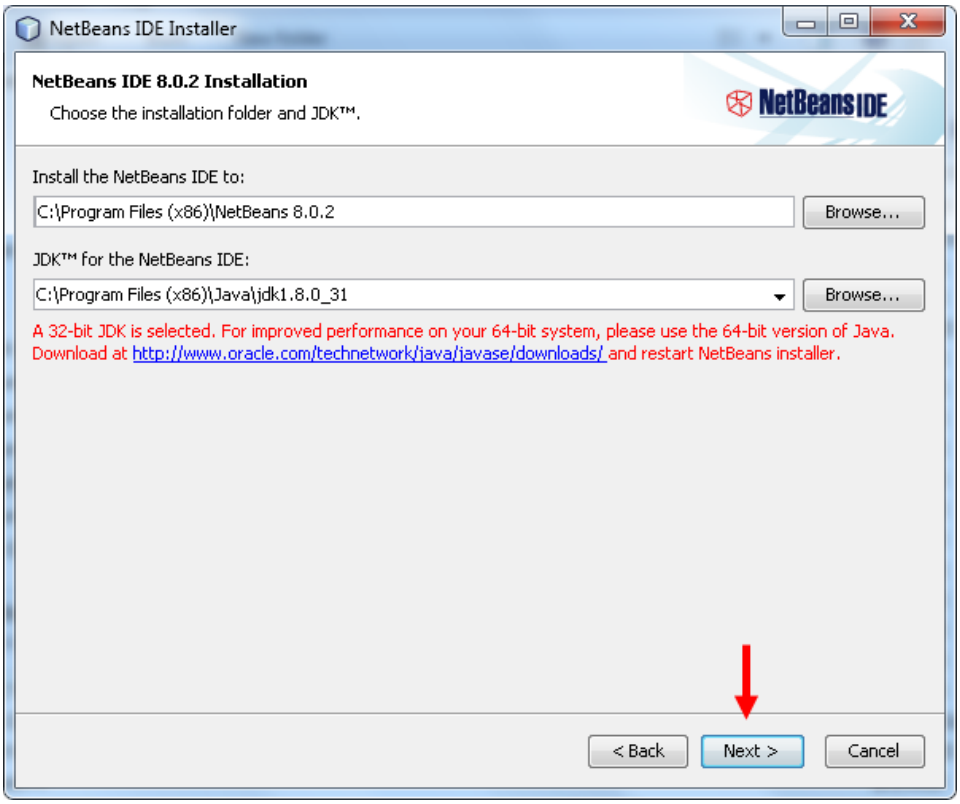
نصب NetBeans

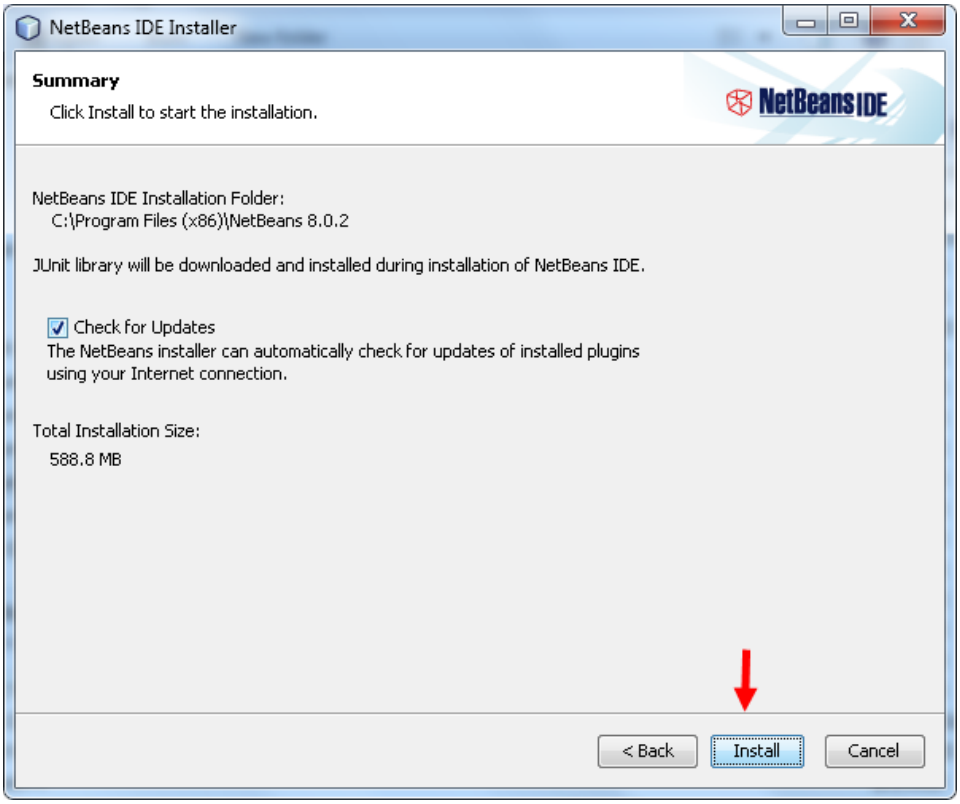


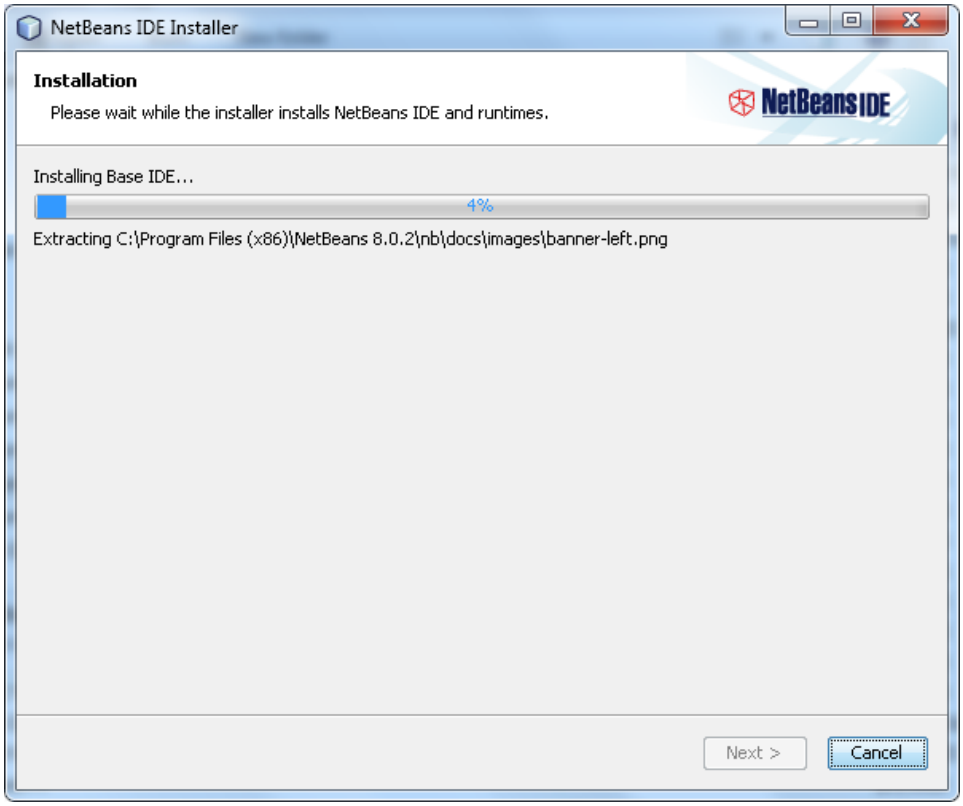


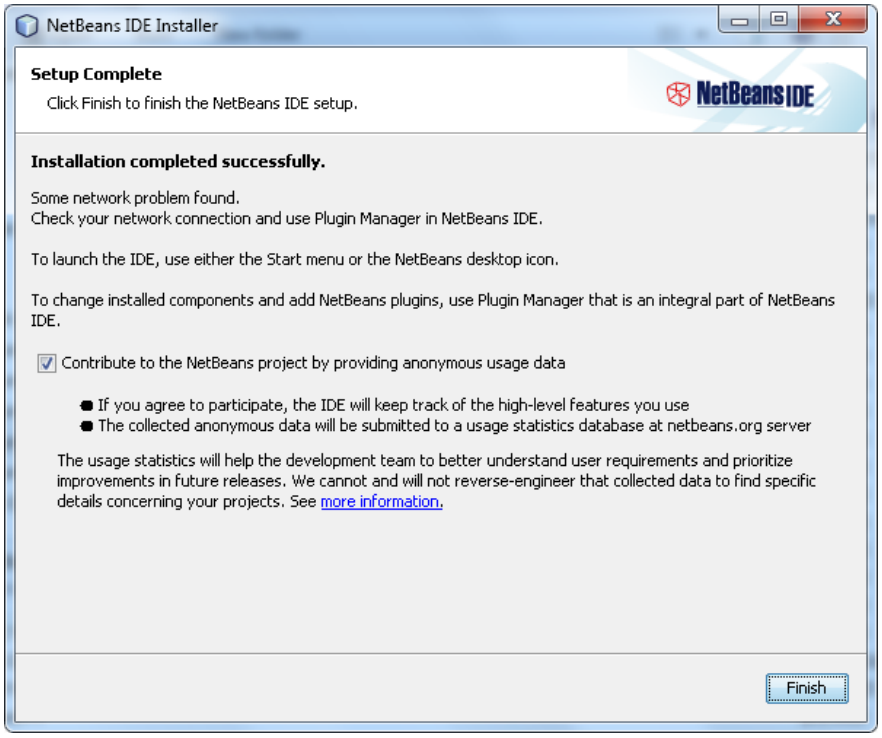

NetBeans IDE Installer
License Agreement
Please read the following license agreement carefully.
NetBeans IDE
NETBEANS IDE 8.0.2 ("Product") LICENSE AGREEMENT
PLEASE READ THE FOLLOWING LICENSE AGREEMENT TERMS AND CONDITIONS CAREFULLY, INCLUDING WITHOUT LIMITATION THOSE DISPLAYED ELSEWHERE (AS INDICATED 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

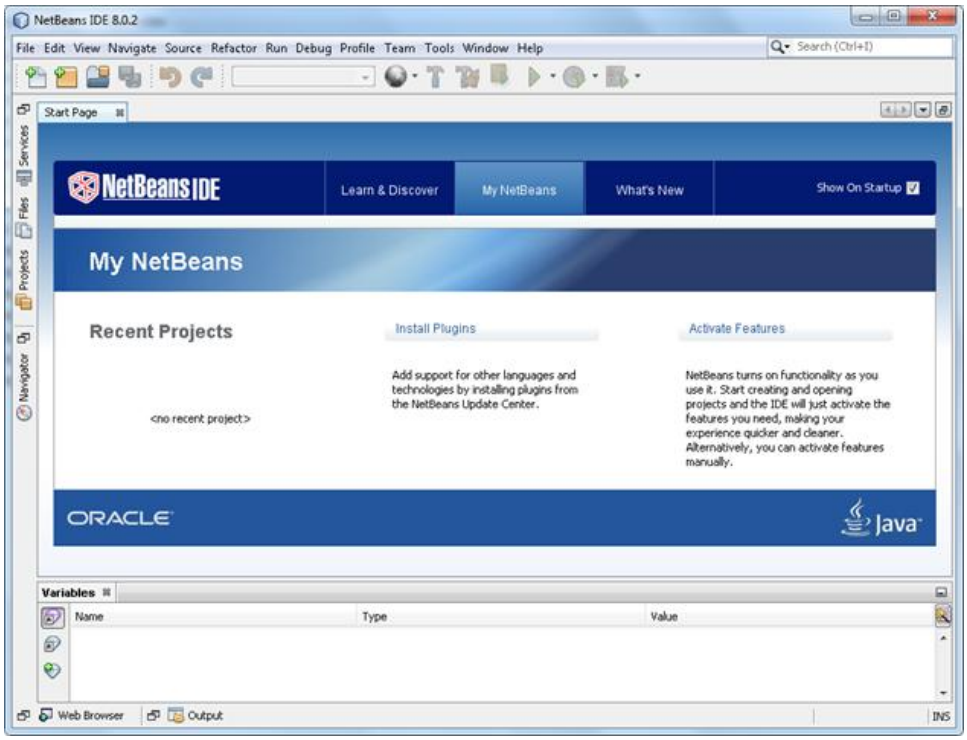

NetBeans IDE Installer
NetBeans IDE 8.0.2 Installation
Choose the installation folder and JDK™.
NetBeans IDE
Install the NetBeans IDE to:
C:\Program Files (x86)\NetBeans 8.0.2
Browse...
JDK™ for the NetBeans IDE:
C:\Program Files (x86)\Java\jdk1.8.0_31
Browse...
A 32-bit JDK is selected. For improved 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





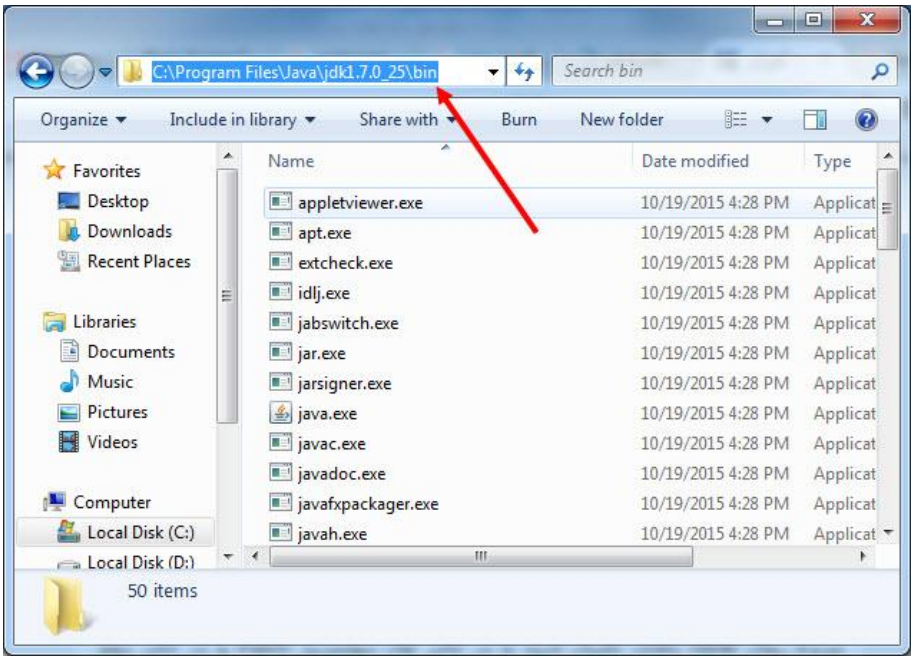
وقتی برای اولین بار بر روی آیکون NetBeans بر روی دسکتاپ کلیک کرده و آن را اجرا می‌کنید، صفحه اول برنامه به صورت زیر نمایش داده می‌شود که نشان دهنده نصب کامل آن است:



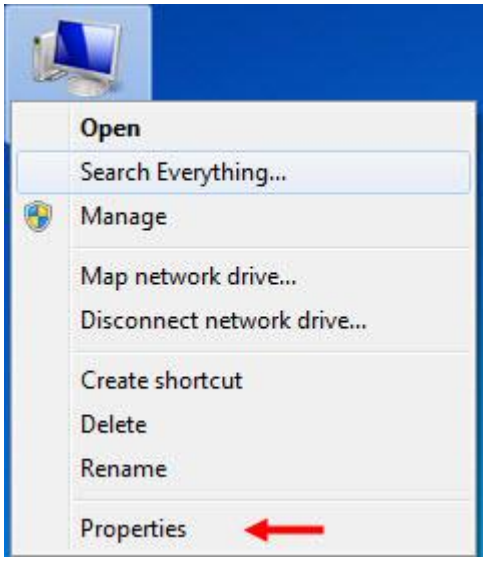
در درس آینده درباره ایجاد پروژه در NetBeans توضیح می‌دهیم.

پیکربندی JDK

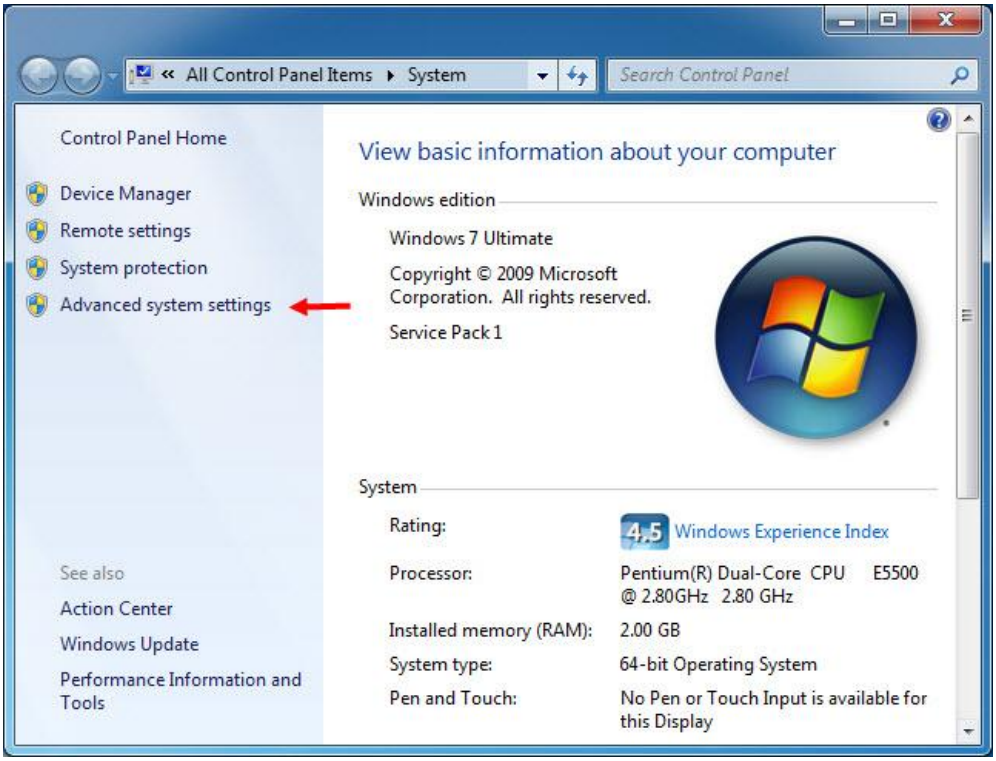
در درس قبل ما نسخه نهایی JDK یا Java Development Kit را بر روی سیستم عامل مان نصب کردیم. ولی این پایان کار نیست. برای اجرای برنامه های جاوا لازم است که مسیر پوشه bin این نرم افزار را در متغیر path معرفی کنیم. برای این کار ابتدا مسیر پوشه bin را کپی کنید:



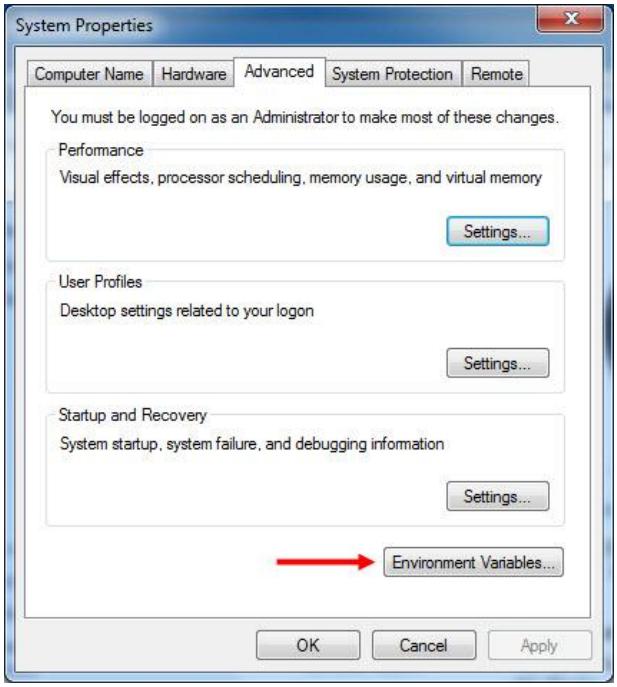
سپس بر روی `MyComputer` راست کلیک کرده و روی گزینه `Properties` کلیک کنید:



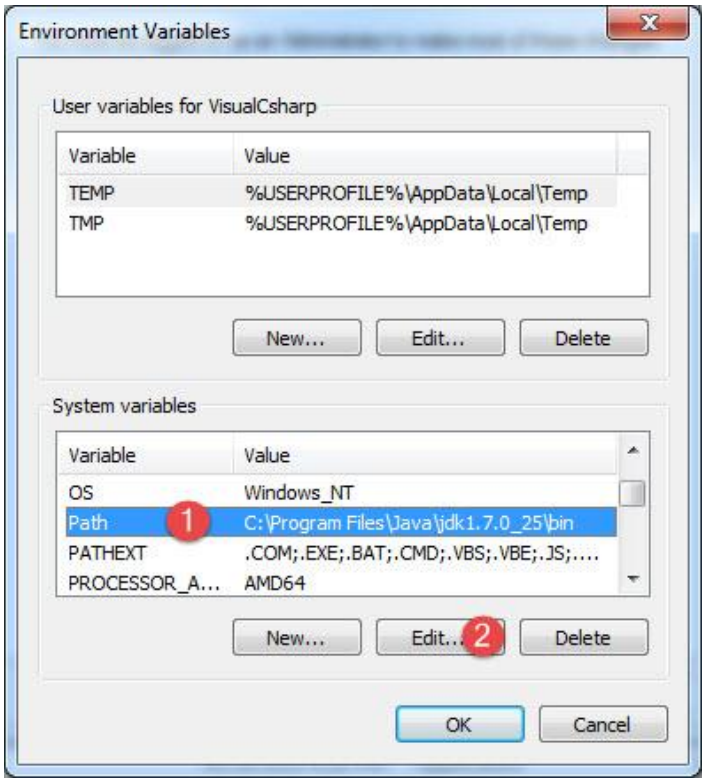
از پنل سمت چپ این صفحه `Advanced system settings` را باز کنید:



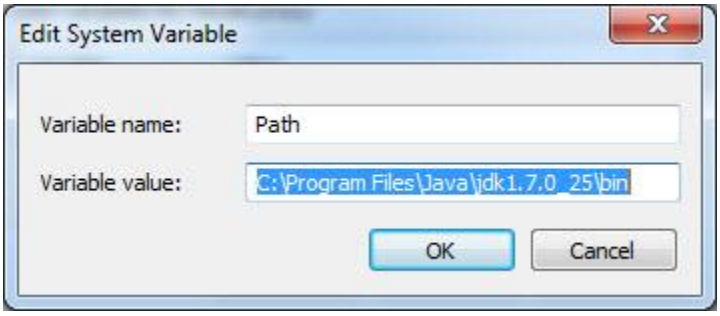
به تب Advanced روی Environment Variables… کلیک کنید.



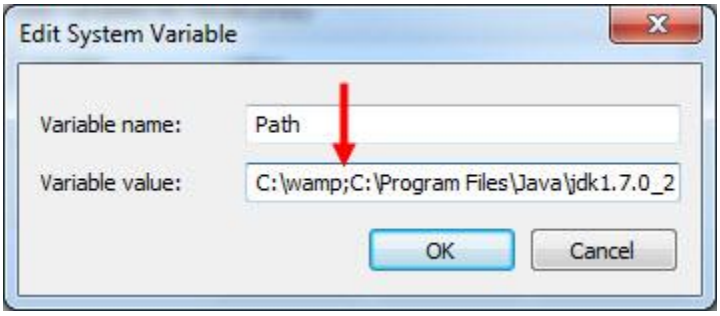
در قسمت پایین و بخش System Variables روی گزینه Path کلیک کرده و سپس گزینه Edit را بزنید:



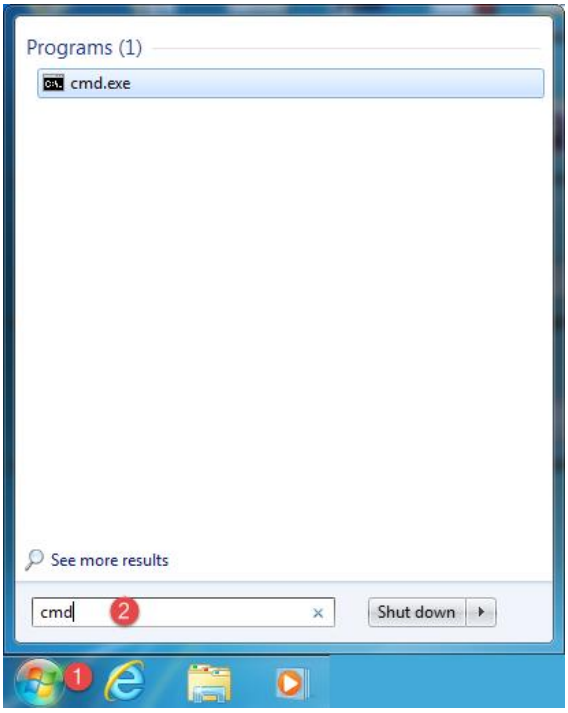
در پنجره باز شده اگر قسمت `Variable Value` خالی بود مسیر پوشه `bin` را در آن کپی کنید:



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 cmd را 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>
```

ساخت یک برنامه ساده در JAVA

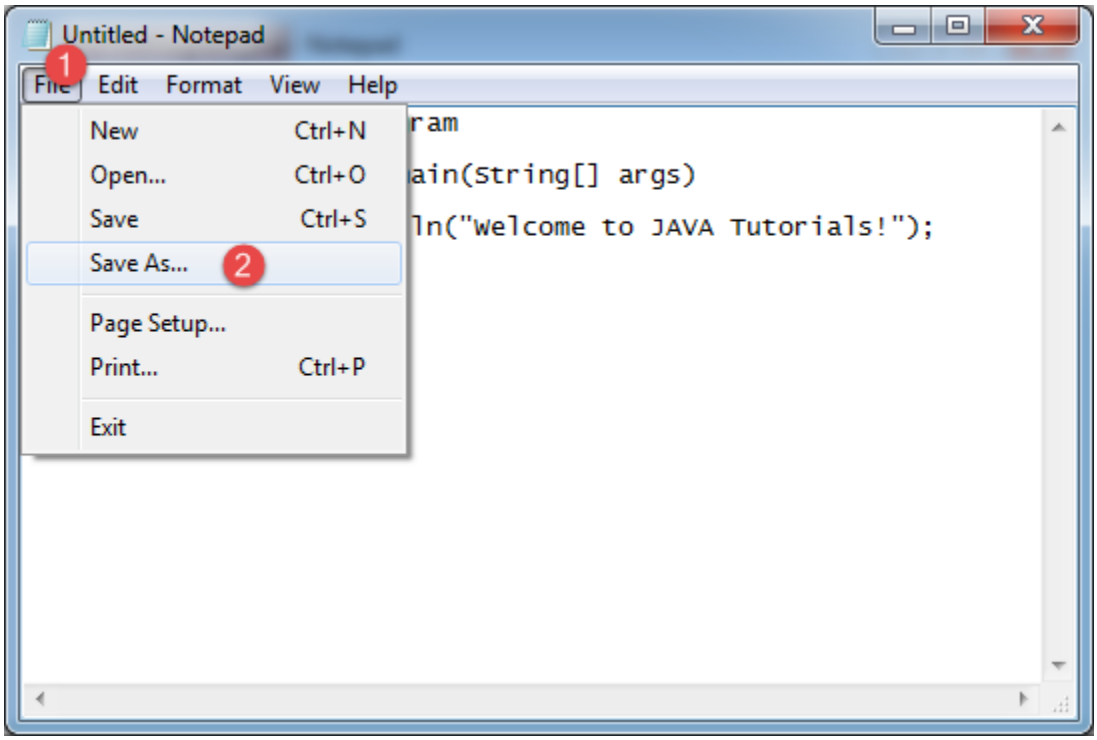
اجازه بدهید یک برنامه بسیار ساده به زبان جاوا بنویسیم. این برنامه یک پیغام را نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده جاوا را توضیح دهم. قبل از ایجاد برنامه به این نکته توجه کنید که کدهای جاوا را می‌توان در داخل یک ویرایشگر متن ساده مانند NotePad نوشت و اجرا کرد. فقط کافیست که JDK بر روی سیستم شما نصب باشد. استفاده از نرم افزارهایی مانند NetBeans فقط برای راحتی در کدنویسی و کاهش خطا می‌باشد.

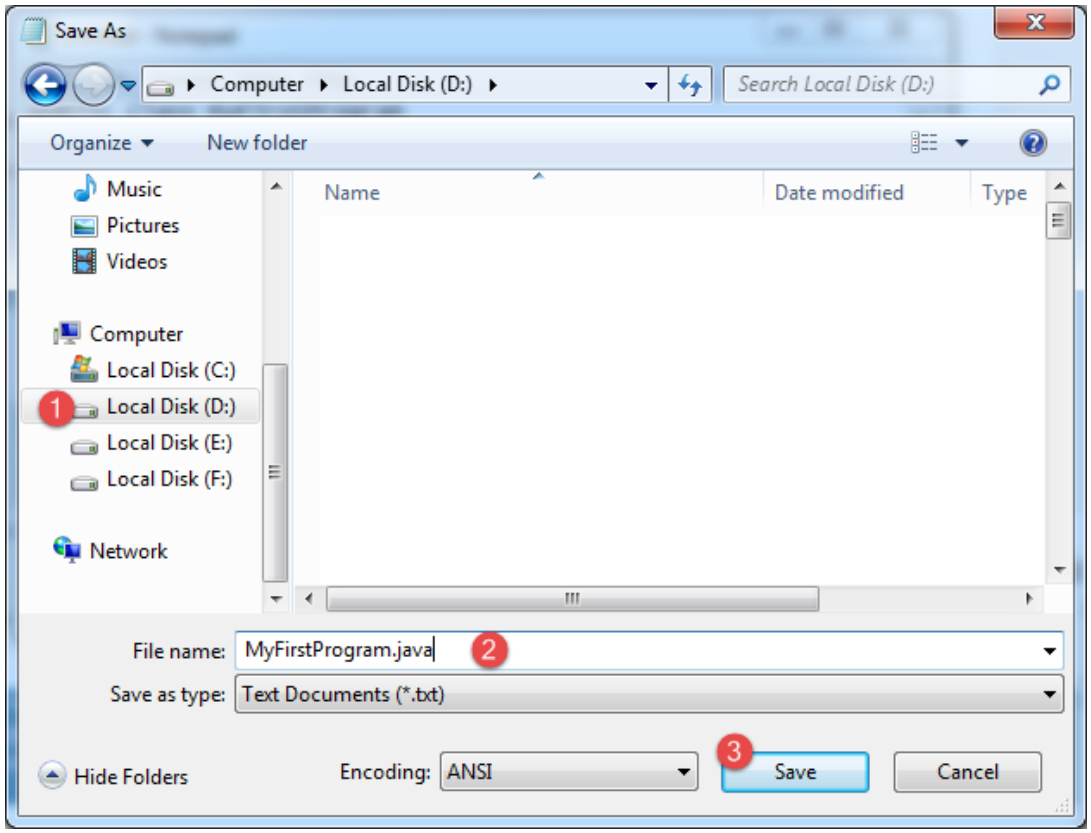
بدون استفاده از NetBeans

همانطور که گفته شد، شما برای کامپایل و اجرای برنامه‌های جاوا به ابزاری به نام JDK نیاز دارید. JDK مخفف عبارت Java Development Kit است و شامل ابزارهای مورد نیاز شما برای اجرای برنامه‌های جاوا می‌شود. این مجموعه شامل ابزاری به نام JVM یا Java Virtual Machine است که ماشین مجازی جاوا نام دارد و وظیفه‌ی کامپایل و اجرای کدهای شما را برعهده دارد. خود JVM هم شامل ابزارهای دیگری است. مثلاً javac یا java compiler اختصاصاً وظیفه‌ی کامپایل کردن برنامه‌ها را برعهده دارد. در درس قبل JDK را نصب کردیم و الان فرض می‌کنیم که شما هیچ IDE مانند netbeans و یا eclipse در اختیار ندارید و می‌خواهید 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 :

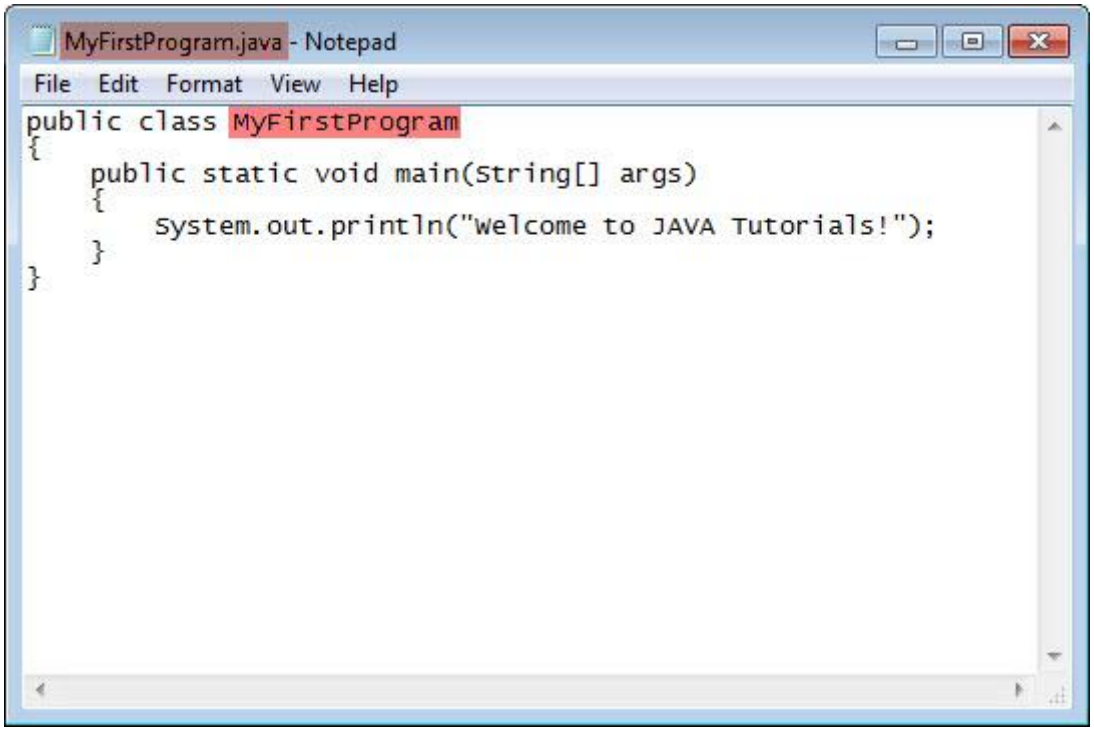
```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to JAVA Tutorials!");
    }
}
```

و در درایو D و با نام و پسوند MyFirstProgram.java ذخیره می کنیم :

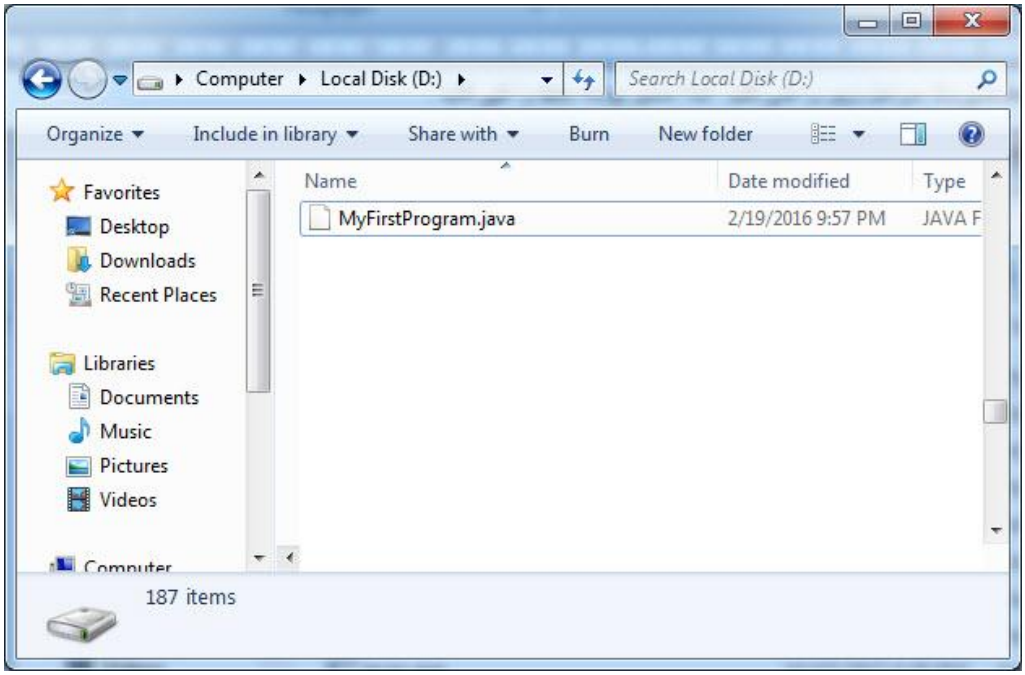




نگران توضیح کدهای بالا نباشید، در ادامه در مورد آنها توضیح می دهیم. پس شکل نهایی برنامه، باید به صورت زیر باشد:



حال نوبت به اجرای برنامه می رسد:

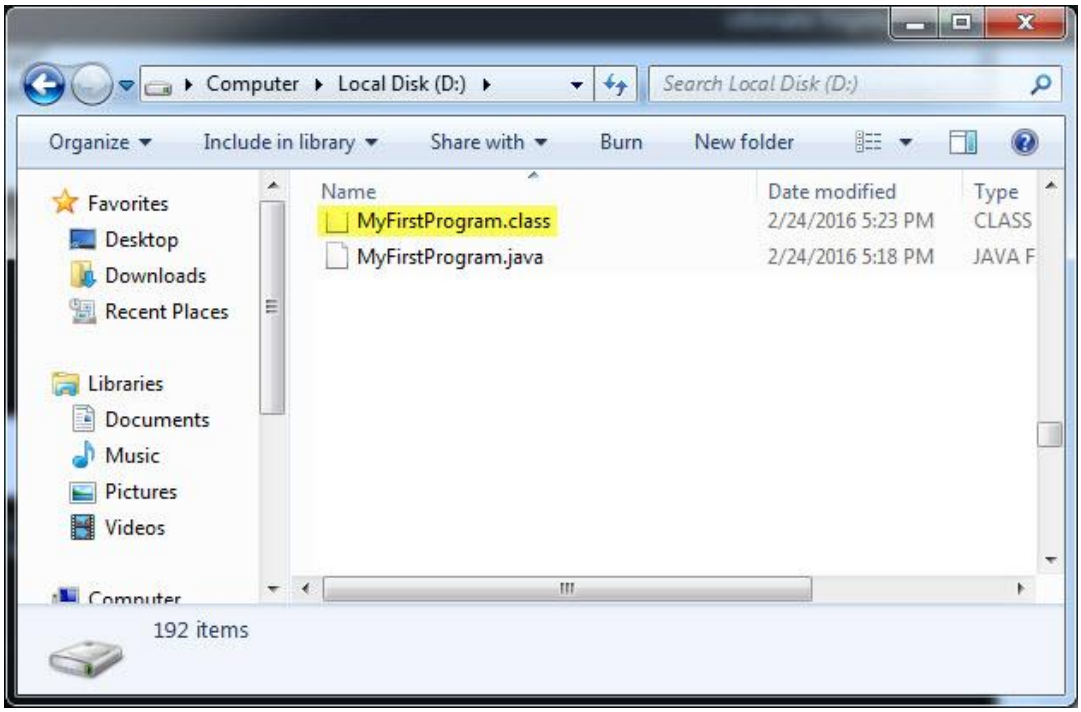


فایل ما در درایو `D` قرار دارد. ابتدا `cmd` را باز کرده و کد زیر را در داخل آن نوشته و دکمه `Enter` را می زنید:

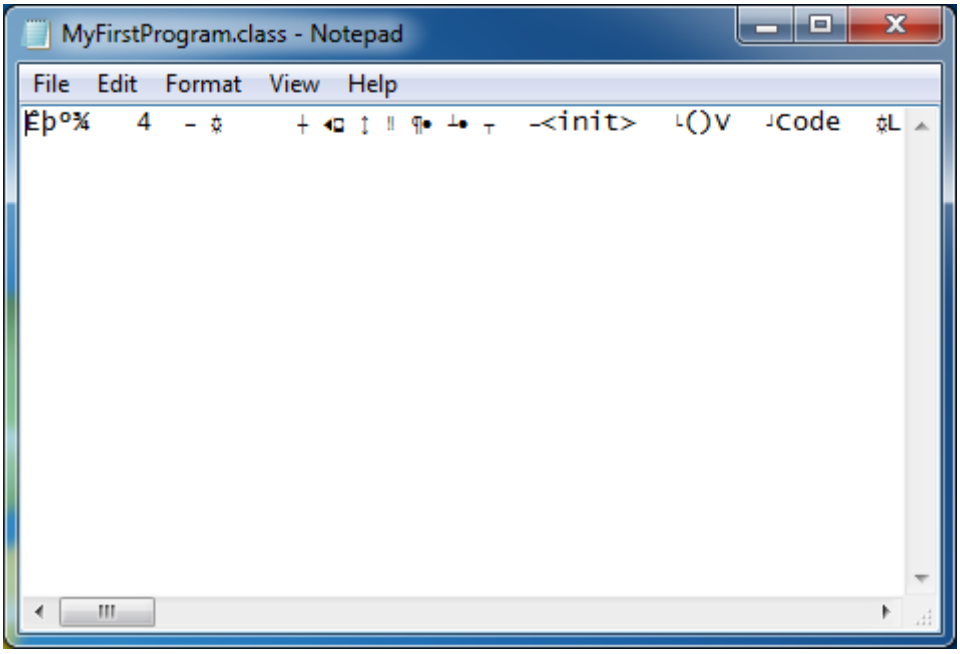
```
Microsoft Windows [Version 6.1.7601]
```

```
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:



اگر فایل ایجاد شده یعنی `MyFirstProgram.class` را با برنامه `NotePad` باز کنید مشاهده می کنید که شامل کدهایی نا مفهوم می باشد:



به این کدهای نامفهوم که برای ماشین مجازی جاوا (JVM) قابل فهم می باشد، ByteCode گفته می شود. 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.

یک نکته را متذکر می شویم و آن این است که نام فایل دارای پسوند java باید با نام کلاسی که در داخل آن است، یکی باشد،

در غیر اینصورت با خطا مواجه می شوید. فرض کنید ما نام فایل بالا را به Program.java تغییر داده و آن را کامپایل می کنیم.

در اینصورت خطای زیر به ما نمایش داده می شود:

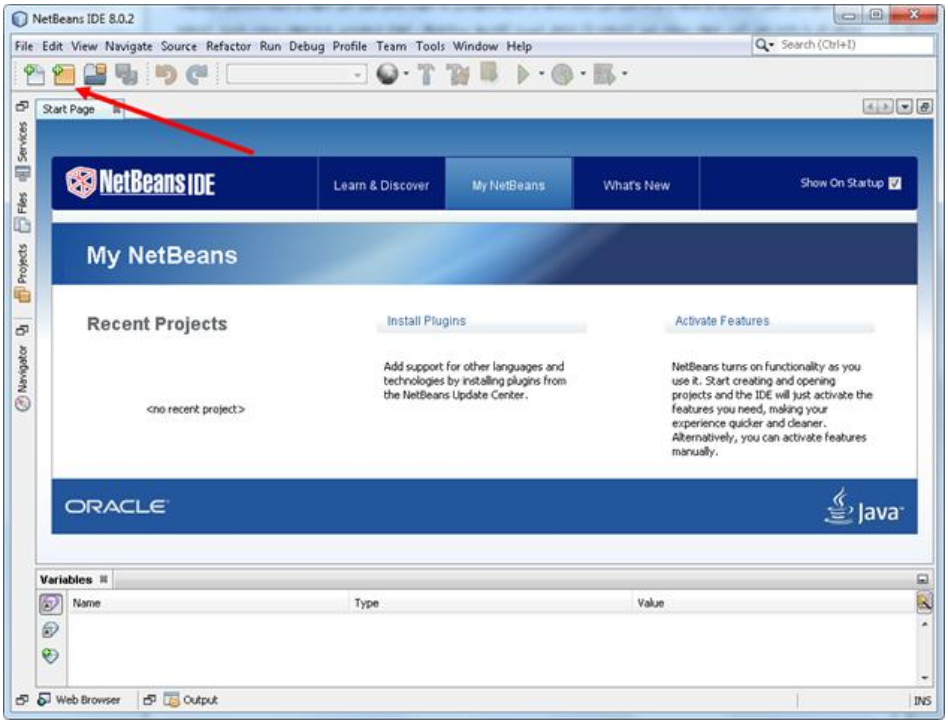
```
Program.java:3: error: class MyFirstProgram is public,
should be declared in a file named MyFirstProgram.java
public class MyFirstProgram
       ^
```

```
1 error
```

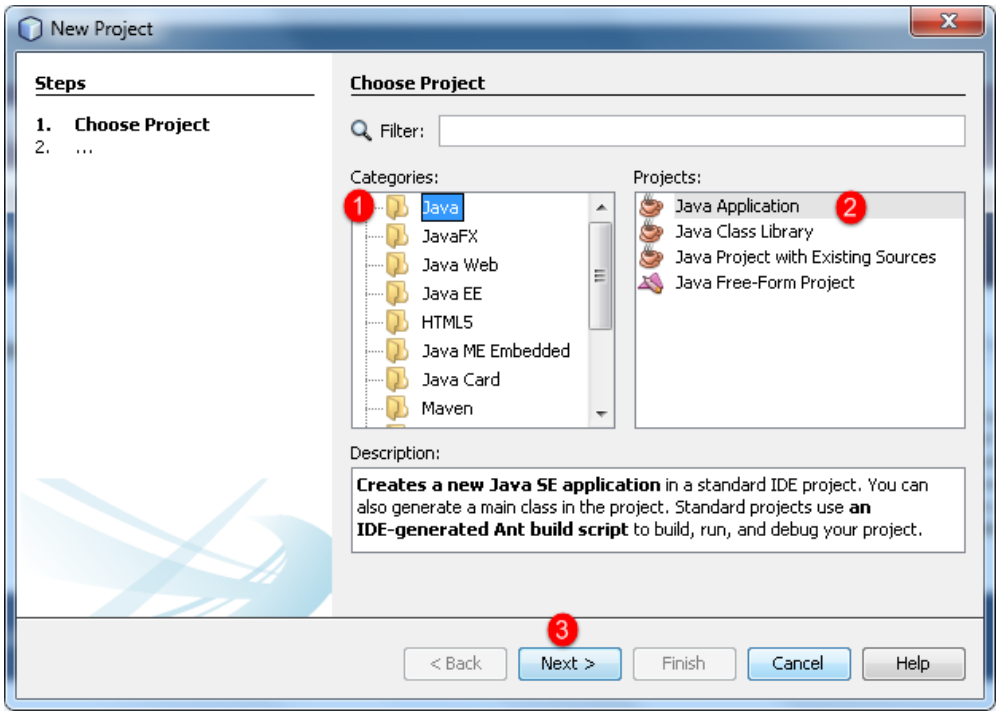
خطای بالا به این معنی است که کلاسی با نام MyFirstProgram باید در فایلی با نام MyFirstProgram.java تعریف شود.

با استفاده از NetBeans

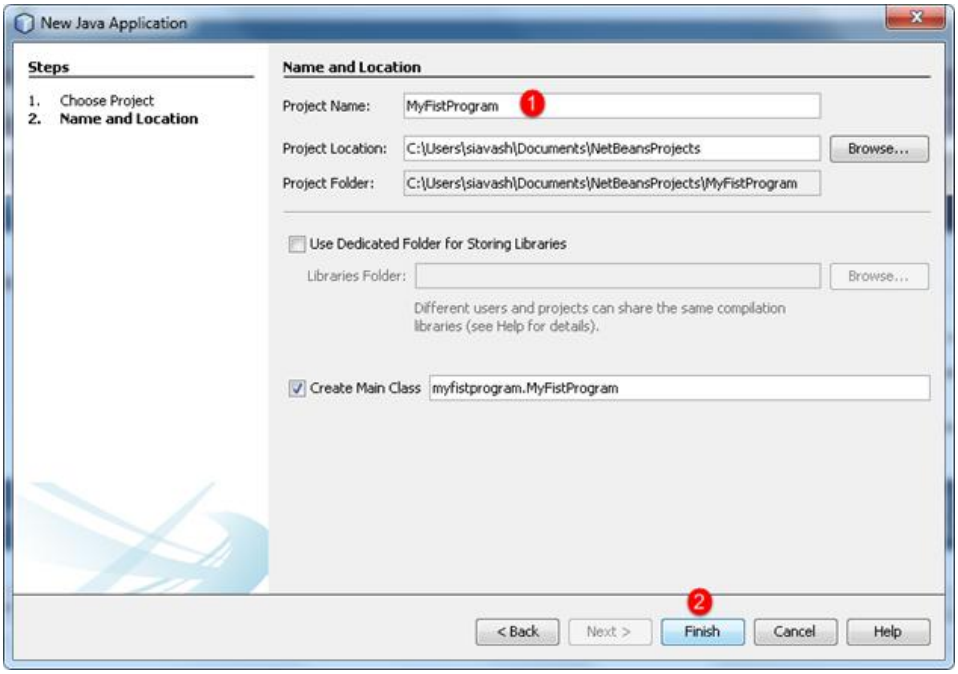
برنامه NetBeans را اجرا کنید. از مسیری که در شکل زیر نشان داده شده است یک پروژه جدید ایجاد کنید:



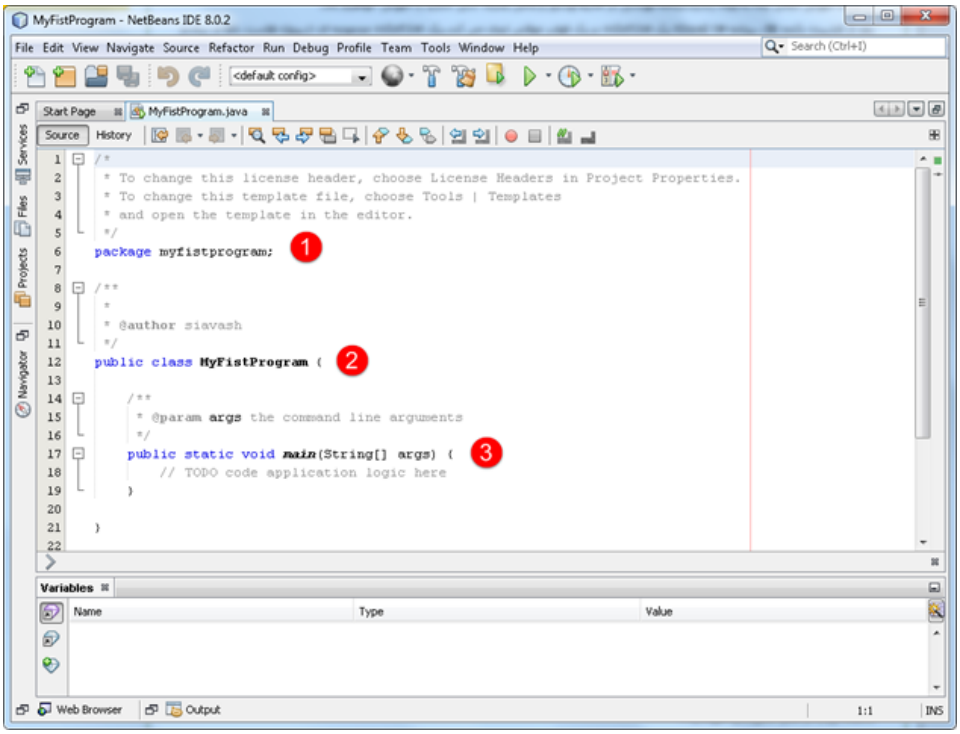
حال با یک صفحه مواجه می‌شوید. طبق شماره‌هایی که در شکل زیر نمایش داده شده‌اند گزینه‌ها را انتخاب کرده و به مرحله بعد بروید:



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



بعد از فشردن دکمه `Finish`، وارد محیط کدنویسی برنامه به صورت زیر می‌شویم:



محیط کدنویسی یا IDE جایی است که ما کدها را در آن تایپ می‌کنیم.کدها در محیط کدنویسی به صورت رنگی تایپ می‌شوند در نتیجه تشخیص بخشهای مختلف کد را راحت می‌کند. همانطور که در شکل بالا مشاهده می‌کنید ما کدهای پیشفرض را به سه قسمت تقسیم کرده‌ایم. قسمت اول Package، قسمت دوم کلاس و قسمت سوم متد ()main. 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:

```
1  package myfirstprogram;
2
3  public class MyFirstProgram
4  {
5      public static void main(String[] args)
6      {
7          System.out.println("Welcome to JAVA Tutorials!");
8      }
9  }
```

ساختار یک برنامه در جاوا

مثال بالا ساده‌ترین برنامه ای است که شما می‌توانید 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 می‌کند. درباره Package در درس‌های آینده توضیح خواهیم داد. در خط ۴، آکولاد ({) نوشته شده است. آکولاد برای تعریف یک بلوک کد به کار می‌رود. جاوا یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می‌باشد. هر آکولاد باز ({) در جاوا باید دارای یک آکولاد بسته (}) نیز باشد. همه کدهای نوشته شده از خط ۴ تا خط ۹ یک بلوک کد است.

در خط ۴ یک کلاس تعریف شده است.درباره کلاس‌ها در فصل‌های آینده توضیح خواهیم داد. در مثال بالا کدهای شما باید در داخل یک کلاس نوشته شود.بدنه کلاس شامل کدهای نوشته شده از خط ۴ تا ۹ می‌باشد. خط ۵ متد ()main یا متد اصلی نامیده می‌شود. هر متد شامل یک سری کد است که وقتی اجرا می‌شوند 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 می‌شود.درباره متد ()main در فصول بعدی توضیح خواهیم داد. متد ()main و سایر متدها دارای آکولاد و کدهایی در داخل آن‌ها می‌باشند و وقتی کدها اجرا می‌شوند که متدها را صدا بزنیم. هر خط کد در جاوا به یک سیمیکولن (;) ختم می‌شود. اگر سیمیکولن در آخر خط فراموش شود برنامه با خطا مواجه می‌شود. 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 می‌دهد.از متد println() برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است که به وسیله دابل کوتیشن (") محصور شده است، مانند :

. "Welcome to Visual C# Tutorials!"

یک کاراکتر می‌تواند 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 می‌شود. out یک فیلد استاتیک کلاس System

و کلاس System هم یک کلاس از پیش تعریف شده در جاوا می‌باشد. جاوا فضای خالی و خطوط جدید را نادیده می‌گیرد. بنابراین شما می‌توانید همه برنامه را در یک خط بنویسید. اما اینکار خواندن و اشکال زدایی برنامه را مشکل می‌کند. 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 می‌گیرد و از کد بالا اشکال نمی‌گیرد. اما از کد زیر ایراد می‌گیرد :

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

به سیمیکولن آخر خط اول توجه کنید.برنامه با خطای نحوی مواجه می‌شود 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. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
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 می‌کند. اما کد زیر کاملاً بدون خطا است:

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

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
  statement1;
}
```

این کار باعث می‌شود 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 می‌کند و این در خوانایی برنامه بسیار مؤثر است.

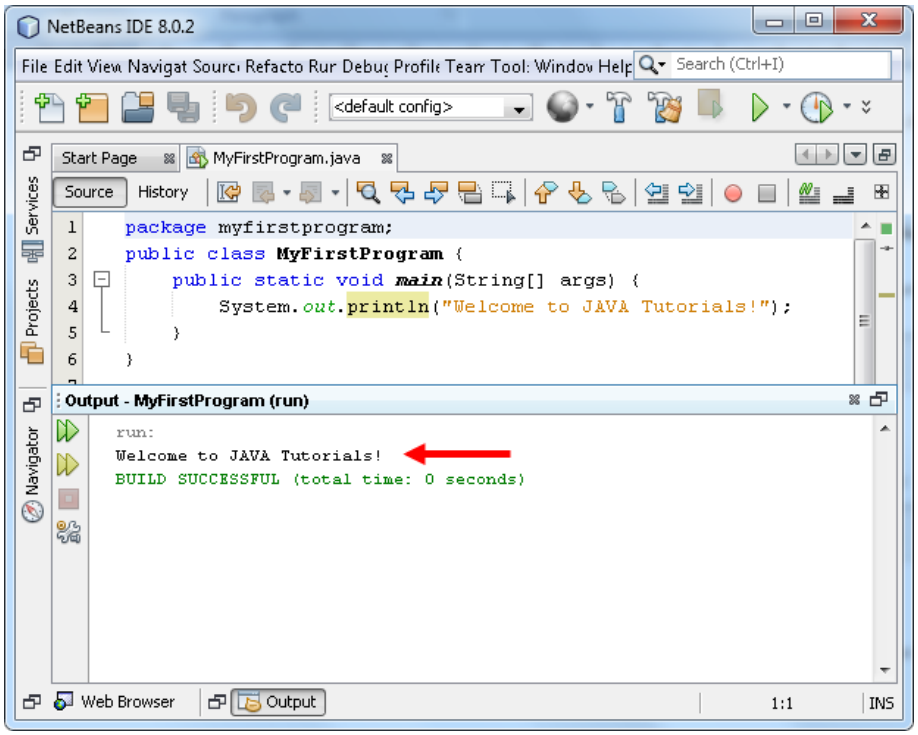
ذخیره پروژه و اجرای برنامه

برای ذخیره پروژه و برنامه می‌توانید به مسیر File > Save All بروید یا از کلیدهای میانبر Ctrl+Shift+S استفاده کنید.

همچنین می‌توانید 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 می‌کنید که رشته Welcome to JAVA Tutorials! در خروجی برنامه به صورت زیر نمایش داده می‌شود:



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

استفاده از Package

برای دسته بندی کلاس‌ها و قرار دادن کلاس‌های مرتبط با هم در یک مکان، جاوا از مفهومی به نام بسته یا package استفاده می‌کند. پکیج معادل فضای نام در سی شارپ می باشد. یک دلیل برای گروه بندی کلاس‌ها 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 می‌آید. یعنی اگر دو کلاس هم نام در دو Package غیر همنام باشند مشکلی به وجود نمی‌آید. همانطور که در مثال بالا دیدید به طور پیشفرض NetBeatns هنگام ایجاد برنامه یک Package همنام با اسمی که برای برنامه انتخاب کرده‌ایم با حروف کوچک و در داخل این Package هم کلاسی به همین اسم ایجاد می‌کند:

```
package myfirstprogram;

public class MyFirstProgram
{
    ...
}
```

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

```
import PackageName.ClassName
```

همانطور که در مثال بالا مشاهده می‌کنید 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 می‌نویسیم. مثلاً برای استفاده از کلاس MyFirstProgram مربوط به پکیج myfirstprogram به صورت زیر عمل می‌شود:

```
import myfirstprogram.MyFirstProgram;
```

بسته‌ها را می‌توان به صورت تو در تو تعریف کرد. در این حالت در تعریف بسته یک کلاس، از بیرونی‌ترین بسته شروع کرده و هر بسته را با نقطه (.) به بسته بعدی متصل می‌کنیم:

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

نکته ای که بهتر است در همین جا به آن اشاره کنم این است که اگر بخواهید کد زیر را بدون استفاده از NetBeans کامپایل و اجرا کنید، کامپایل می شود ولی در زمان اجرا با خطای Error: Could not find or load main class MyFirstProgram مواجه می شوید:

```
package myfirstprogram;

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

کد بالا را با نام و پسوند MyFirstProgram.java و در درایو D ذخیره کنید. حال cmd را اجرا کرده و سعی کنید کد بالا را کامپایل و اجرا نمایید:

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

C:\Users\YounesJava>d:

D:\>javac MyFirstProgram.java

D:\>java MyFirstProgram
Error: Could not find or load main class MyFirstProgram

D:\>java -cp . myfirstprogram.MyFirstProgram
Welcome to JAVA Tutorials!
```

همانطور که مشاهده می کنید برای کامپایل کد بالا به صورت زیر عمل کرده ایم:

```
javac MyFirstProgram.java
```

اما برای اجرای کد، اگر از دستور زیر استفاده کنیم با خطا مواجه می شویم:

```
java MyFirstProgram
```

دلیل وجود خطا، خط اول کد، یعنی package myfirstprogram; می باشد. برای رفع این مشکل یا باید خط مذکور را از برنامه حذف و برنامه را مجددا کامپایل و اجرا کرد یا اگر آن را پاک نکنید باید از کد زیر برای اجرای کد استفاده کنید:

```
java -cp . myfirstprogram.MyFirstProgram
```

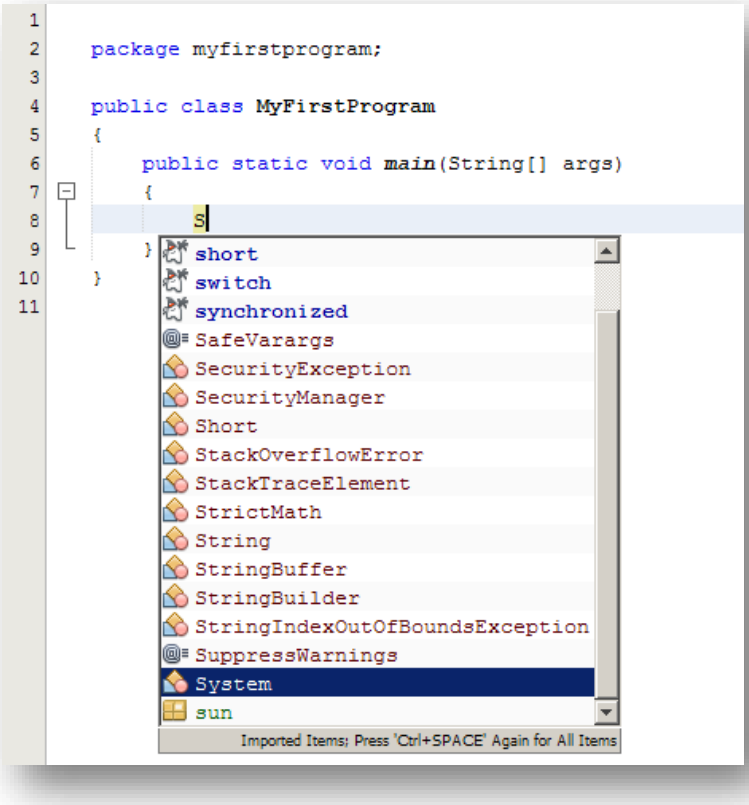
که در این صورت برنامه اجرا و پیغام Welcome to JAVA Tutorials! چاپ می شود. در درس های آینده توصیه می کنیم که اگر می خواهید از NotePad و cmd برای اجرای کدها استفاده کنید این خط را از اول هم فایل ها حذف و سپس برنامه را کامپایل و اجرا کنید.

استفاده از IntelliSense در NetBeans

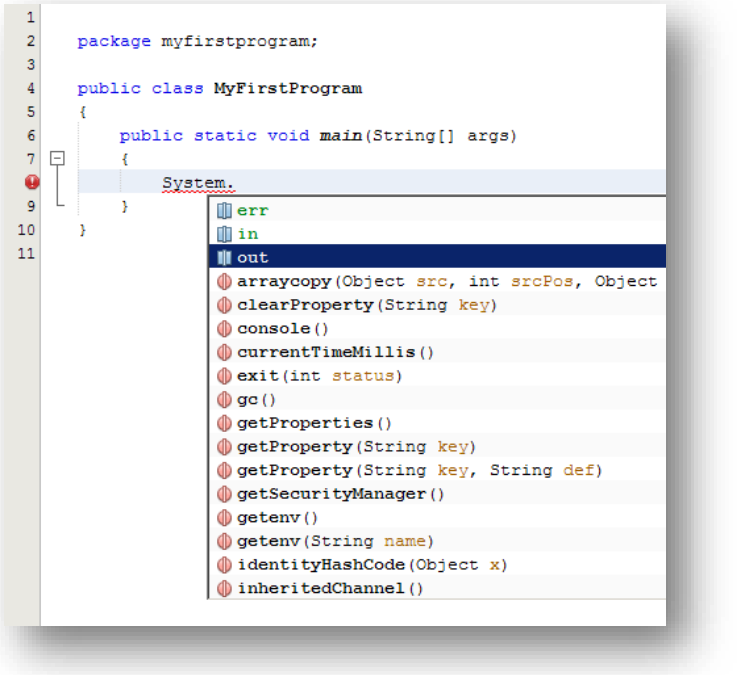
شاید یکی از ویژگیهای مهم NetBeans، اینتلی سنس باشد. IntelliSense ما را قادر می‌سازد که به سرعت به کلاس‌ها و متدها و....دسترسی پیدا کنیم. وقتی که شما در محیط کدنویسی حرفی را تایپ کنید IntelliSense فوراً فعال می‌شود. کد زیرا را در داخل متد main() بنویسید.

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

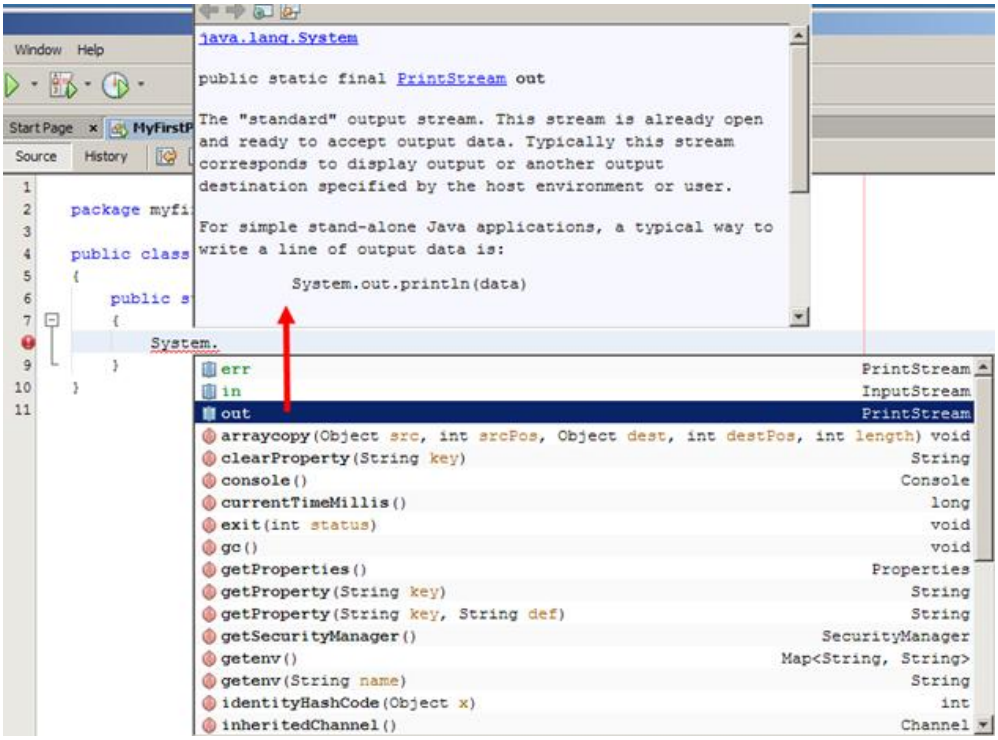
اولین حرف را تایپ کرده و سپس دکمه‌های ترکیبی Ctrl+Space را فشار دهید تا IntelliSense فعال شود:



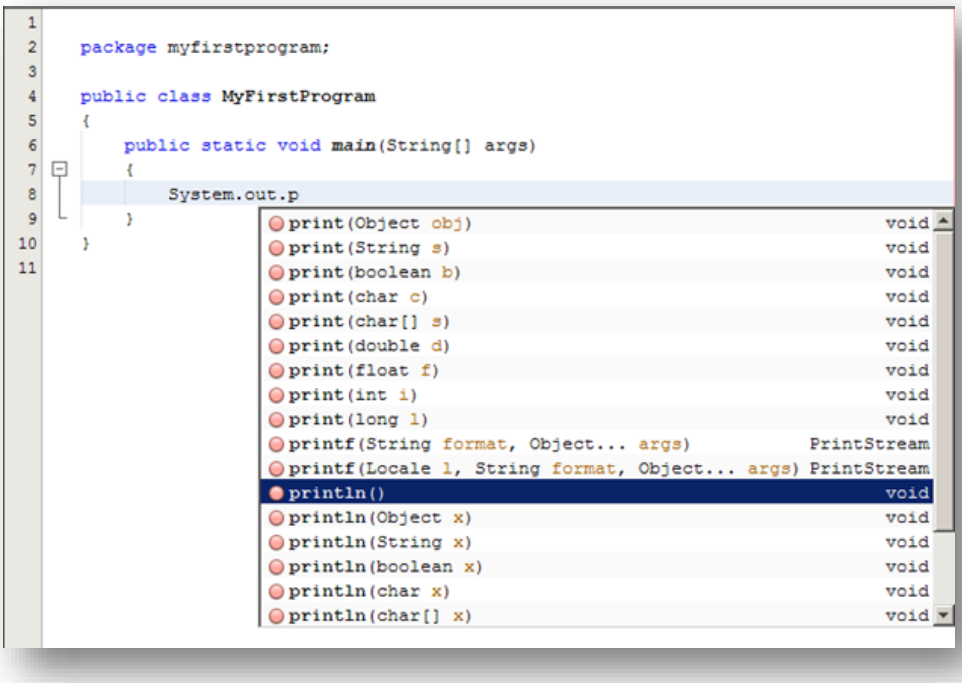
IntelliSense لیستی از کلمات به شما پیشنهاد می‌دهد که بیشترین تشابه را با نوشته شما دارند. شما می‌توانید با زدن دکمه tab گزینه مورد نظرتان را انتخاب کنید. با تایپ نقطه (.) شما با لیست پیشنهادی دیگری مواجه می‌شوید:



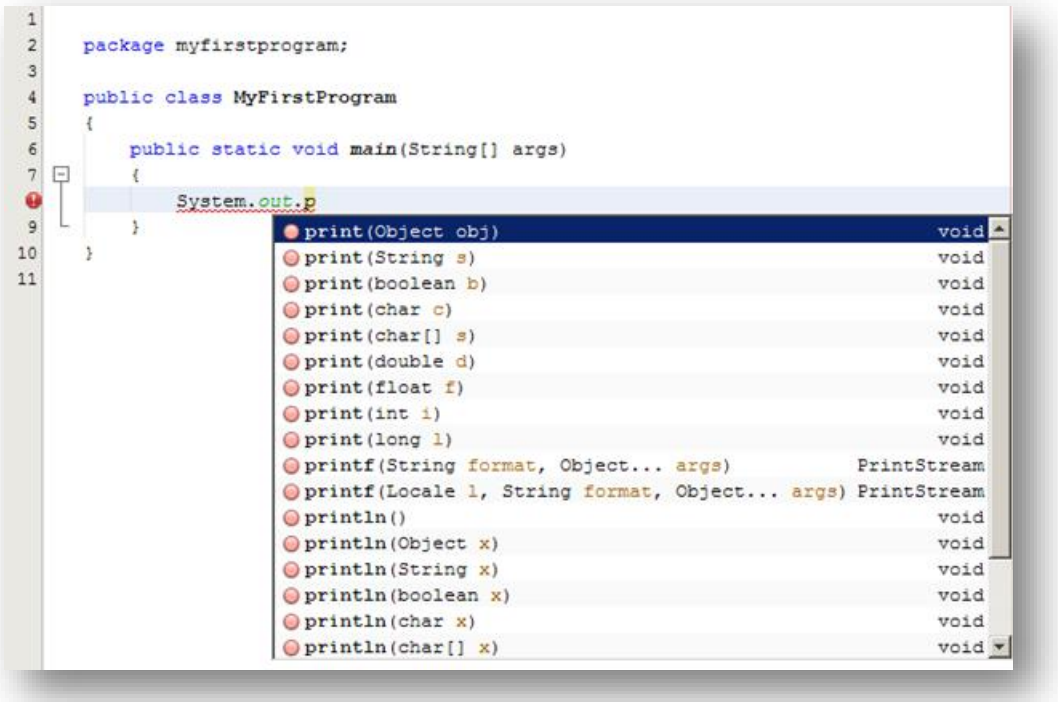
اگر بر روی گزینه ای که می‌خواهید انتخاب کنید لحظه ای مکث کنید توضیحی در رابطه با آن مشاهده خواهید کرد مانند شکل زیر:



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



با تایپ حرف‌های بیشتر لیست محدودتر می‌شود. اگر IntelliSense نتواند چیزی را که شما تایپ کرده‌اید پیدا کند هیچ چیزی را نمایش نمی‌دهد. برای ظاهر کردن IntelliSense کافیست دکمه ترکیبی Ctrl+Space را فشار دهید. برای انتخاب یکی از متدهایی که دارای چند حالت هستند، می‌توان با استفاده از دکمه‌های مکان نما (بالا و پایین) یکی از حالت‌ها را انتخاب کرد. مثلاً متد println() همانطور که در شکل زیر مشاهده می‌کنید دارای چندین حالت نمایش پیغام در صفحه است:



IntelliSense به طور هوشمند کدهایی را به شما پیشنهاد می‌دهد و در نتیجه زمان نوشتن کد را کاهش می‌دهد.

رفع خطاها

بیشتر اوقات هنگام برنامه نویسی با خطا مواجه می‌شویم. تقریباً همه برنامه‌هایی که امروزه می‌بینید حداقل از داشتن یک خطا رنج می‌برند. خطاها می‌توانند برنامه شما را با مشکل مواجه کنند. در جاوا سه نوع خطا وجود دارد:

خطای کمپایلری

این نوع خطا از اجرای برنامه شما جلوگیری می‌کند. این خطاها شامل خطای دستور زبان می‌باشد. این بدین معنی است که شما قواعد کد نویسی را رعایت نکرده‌اید. یکی دیگر از موارد وقوع این خطا هنگامی است که شما از چیزی استفاده می‌کنید که نه وجود دارد و نه ساخته شده است. حذف فایل‌ها یا اطلاعات ناقص در مورد پروژه ممکن است باعث به وجود آمدن خطای کمپایلری شود.استفاده از برنامه بوسیله برنامه دیگر نیز ممکن است باعث جلوگیری از اجرای برنامه و ایجاد خطای کمپایلری شود.

خطاهای منطقی

این نوع خطا در اثر تغییر در یک منطق موجود در برنامه به وجود می‌آید. 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 می‌کند ولی حاصل تفریق دو عدد را نشان می‌دهد.در این حالت ممکن است برنامه نویس علامت ریاضی را اشتباه تایپ کرده باشد.

استثناء

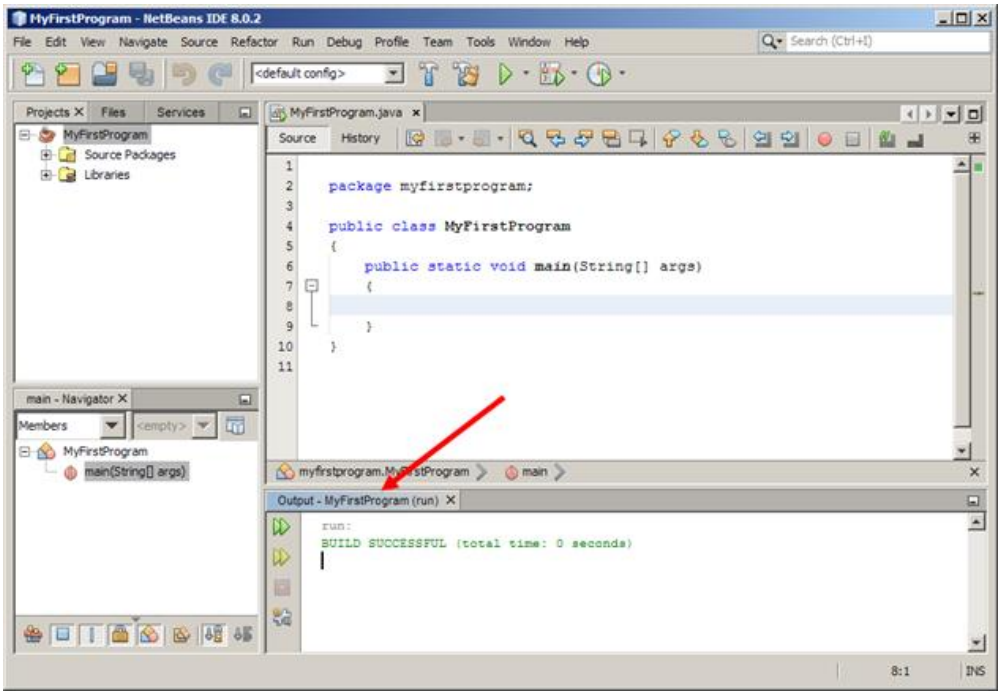
این نوع خطاها هنگامی رخ می‌دهند که برنامه در حال اجراست. این خطا هنگامی روی می‌دهد که کاربر یک ورودی نامعتبر به برنامه بدهد و برنامه نتواند آن را پردازش کند. NetBeans دارای ابزارهایی برای پیدا کردن و برطرف کردن خطاها هستند. وقتی در محیط کدنویسی در حال تایپ کد هستیم یکی از ویژگیهای NetBeans تشخیص خطاهای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

```
package myfirstprogram;

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

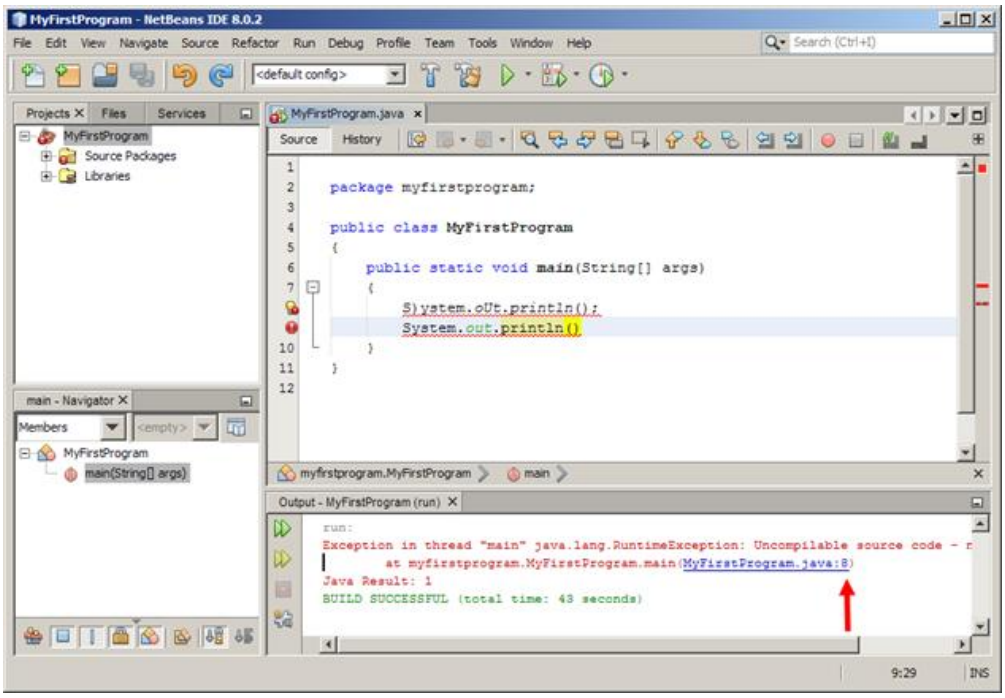
هنگامی که شما با ماوس روی این خطوط توقف کنید توضیحات خطا را مشاهده می‌کنید. شما ممکن است با خط سبز هم مواجه شوید که نشان دهنده اخطار در کد است ولی به شما اجازه اجرای برنامه را می‌دهند. به عنوان مثال ممکن است شما یک متغیر را تعریف کنید ولی در طول برنامه از آن استفاده نکنید. (در درس‌های آینده توضیح خواهیم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

```
package myfirstprogram;

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

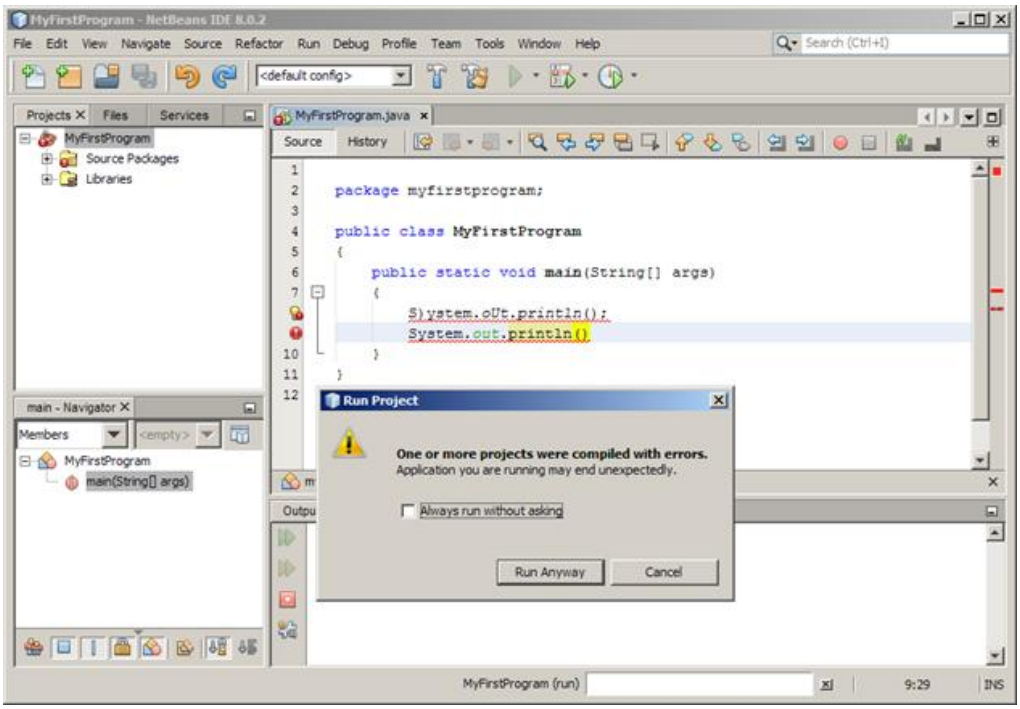
در باره رفع خطاها در آینده توضیح بیشتری می‌دهیم. پنجره Output که در شکل زیر با فلش قرمز نشان داده شده است به شما امکان مشاهده خطاها، هشدارها و رفع آن‌ها را می‌دهد.



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



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 می‌شوید:



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 روی دکمه Run Anyway برنامه با وجود خطا نیز اجرا می‌شود. اما با کلیک بر روی دکمه Cancel اجرای برنامه متوقف می‌شود 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 می‌شوند و به دنبال آن‌ها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد:

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

```
Hello
World
```

مشاهده کردید که کمپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. متد Println() هم مانند کاراکتر کنترلی \n یک خط جدید ایجاد می‌کند، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می‌کند:

```
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 ماوس را در همان خط نگه می‌دارد و خط جدید ایجاد نمی‌کند. جدول زیر لیست کاراکترهای کنترلی و کارکرد آن‌ها را نشان می‌دهد:

کاراکتر کنترلی	عملکرد	کاراکتر کنترلی	**عملکرد**
\f	Form Feed	\'	چاپ کوتیشن

\n	خط جدید	\”	چاپ دابل کوتیشن
\r	سر سطر رفتن	\\	چاپ بک اسلش
\t	حرکت به صورت افقی	\b	حرکت به عقب

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم.از آنجاییکه \ معنای خاصی به رشته‌ها می‌دهد 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 رشته‌ها استفاده می‌کنیم برای چاپ آن از \” استفاده می‌کنیم:

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

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (‘) از \’ استفاده می‌کنیم:

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

```
The programmer's heaven.
```

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

```
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 می‌شوند:

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

```
K
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی \r۳ آمده است.کاراکتر کنترلی حرف K را به ابتدای سطر برده و جایگزین Mitten می‌کند. برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید :

```
http://www.ascii.cl/htmlcodes.htm
```

اگر کمپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد.بیشترین خطا زمانی اتفاق می‌افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند.

توضیحات

وقتی که کدی تایپ می کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در جاوا (و بیشتر زبانهای برنامه نویسی) می توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کمپایلر نادیده گرفته می شوند و به عنوان بخشی از کد محسوب نمی شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می خواهید در مورد یک کد خاص، توضیح بدهید، می توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می شود. در برنامه زیر نقش توضیحات نشان داده شده است:

```java
package myfirstprogram;
{
    public class MyFirstProgram
    {
        public static void main(String[] args)
        {
            // This line will print the message hello world
            System.out.println("Hello World!");
        }
    }
}
```

در کد بالا، خط ۷ یک توضیح درباره خط ۸ است که به کاربر اعلام می کند که وظیفه خط ۸ چیست ؟ با اجرای کد بالا فقط جمله Hello Worldچاپ شده و خط ۷ در خروجی نمایش داده نمی شود چون کمپایلر توضیحات را نادیده می گیرد .توضیحات بر سه نوعند:

توضیحات تک خطی

```
// single line comment
```

توضیحات چند خطی

```
/* multi
   line
   comment */
```

توضیحات مستند سازی

```
/**
Documentation Comments
*/
```

توضیحات تک خطی همانگونه که از نامش پیداست، برای توضیحاتی در حد یک خط به کار می روند. این توضیحات با علامت // شروع می شوند و هر نوشته ای که در سمت راست آن قرار بگیرد جز توضیحات به حساب می آید. این نوع توضیحات معمولا در بالا یا کنار کد قرار می گیرند. اگر توضیح درباره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می شود. توضیحات چند خطی با */ شروع و با /* پایان می یابند. هر نوشته ای که بین این دو علامت قرار بگیرد جز توضیحات محسوب می شود. نوع دیگری از توضیحات ، توضیحات مستند سازی نامیده می شوند. این نوع با **/ شروع و به /* ختم می شوند. از این نوع برای مستند سازی برنامه استفاده می شود و در درس های آینده در مورد آنها توضیح خواهیم داد.

متغیر

متغیر مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست.که از طریق آن

می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده ای که در آن ذخیره می‌شود یکی است. متغیر دارای عمر نیز هست که از روی آن می‌توان 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 می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند داریم. این مکان همان متغیر است. برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد:

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

دو متغیر با نامهای myNumber و MyNumber دو متغیر مختلف محسوب می‌شوند چون یکی از آن‌ها با حرف کوچک m و دیگری با حرف بزرگ M شروع می‌شود. شما نمی‌توانید دو متغیر را که دقیق شبیه هم هستند را در یک scope (محدوده) تعریف کنید. Scope به معنای یک بلوک کد است که متغیر در آن قابل دسترسی و استفاده است. در مورد Scope در فصل‌های آینده بیشتر توضیح خواهیم داد. متغیر دارای نوع هست که نوع داده ای را که در خود ذخیره می‌کند را نشان می‌دهد. معمول‌ترین انواع داده int، short، long، byte، double، float، char، Boolean می‌باشند. برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع int استفاده کنید.

انواع ساده

انواع ساده انواعی از داده‌ها هستند که شامل اعداد، کاراکترها و مقادیر بولی می‌باشند. به انواع ساده انواع اصلی نیز گفته می‌شود چون از آن‌ها برای ساخت انواع پیچیده تری مانند کلاس‌ها و ساختارها استفاده می‌شود. انواع ساده دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می‌کنند. در جدول زیر انواع ساده و محدوده آن‌ها آمده است:

نوع	دامنه
byte	اعداد صحیح بین ۱۲۸- تا ۱۲۷
short	اعداد صحیح بین ۳۲۷۶۸- تا ۳۲۷۶۷
int	اعداد صحیح بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷
long	اعداد صحیح بین ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸- تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۸۰۷

جدول زیر انواعی که مقادیر با ممیز اعشار را می‌توانند در خود ذخیره کنند را نشان می‌دهد:

نوع	دامنه تقریبی	دقت
float	±1.5E-45 to ±3.4E38	۷ رقم
double	±5.0E-324 to ±1.7E308	۱۵-۱۶ رقم

برای به خاطر سپردن آن‌ها باید از نماد علمی استفاده شود. نوع دیگری از انواع ساده برای ذخیره داده‌های غیر عددی به کار می‌روند و در جدول زیر نمایش داده شده‌اند:

نوع	مقادیر مجاز
char	کاراکترهای یونیکد
boolean	مقدار true یا false

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

استفاده از رشته‌ها

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

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

دلیل اینکه در این قسمت درباره رشته‌ها مختصری توضیح دادیم این است که ممکن است در آموزش‌های بعدی با آن‌ها سر و کار داشته باشیم. در آینده به طور مفصل در مورد رشته‌ها توضیح می‌دهیم.

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
5  public class MyFirstProgram
6  {
7      public static void main(String[] args)
8      {
9          //Declare variables
10         int num1;
11         int num2;
12         double num3;
13         double num4;
14         boolean boolVal;
15         char myChar;
16
```

```
        //Assign values to variables
        num1 = 1;
        num2 = 2;
        num3 = 3.54;
        num4 = 4.12;
        boolVal = true;
        myChar = 'R';

        //Show the values of the variables
        System.out.println(MessageFormat.format("num1 = {0}", num1));
        System.out.println(MessageFormat.format("num3 = {0}", num3));
        System.out.println(MessageFormat.format("num4 = {0}", num4));
        System.out.println(MessageFormat.format("boolVal = {0}", boolVal));
        System.out.println(MessageFormat.format("num2 = {0}", num2));
        System.out.println(MessageFormat.format("myChar = {0}", myChar));
    }
}
```

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

تعریف متغیر

در خطوط ۱۶-۱۱ متغیرهایی با نوع و نام متفاوت تعریف شده‌اند. ابتدا باید نوع داده‌هایی را که این متغیرها 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;
boolean 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 می‌دهد. برای تعریف چند متغیر از یک نوع می‌توان به صورت زیر عمل کرد:

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

مثال

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

در مثال بالا ۵ متغیر از نوع صحیح تعریف شده است. توجه داشته باشید که بین متغیرها باید علامت کاما (,) باشد.

نامگذاری متغیرها

- نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود.
- نمی‌توان از کاراکترهای خاص مانند #، %، & یا عدد برای شروع نام متغیر استفاده کرد مانند 2numbers.
- نام متغیر نباید دارای فاصله باشد. برای نام‌های 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 می‌رود، اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می‌شود مانند : myNumber. توجه کنید که اولین حرف کلمه Number با حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudents است. اگر توجه کنید بعد از اولین کلمه حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

محدوده متغیر

در کد ابتدای درس، متغیرها در داخل متد main() تعریف شده اند. در نتیجه، این متغیرها فقط در داخل متد main() قابل دسترسی و استفاده هستند. محدوده یک متغیر مشخص می‌کند که متغیر در کجای کد قابل دسترسی است. هنگامیکه برنامه به پایان متد main() می‌رسد متغیرها از محدوده خارج و بدون استفاده می‌شوند تا زمانی که برنامه در حال اجراست.محدوده متغیرها انواعی دارد که در درس‌های بعدی با آن‌ها آشنا می‌شوید. تشخیص محدوده متغیر بسیار مهم است چون به وسیله آن می‌فهمید که در کجای کد می‌توان از متغیر استفاده کرد. باید یاد آور شد که دو متغیر در یک محدوده نمی‌توانند دارای نام یکسان باشند. مثلاً کد زیر در برنامه ایجاد خطا می‌کند:

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

از آنجاییکه جاوا به بزرگی و کوچک بودن حروف حساس است می‌توان 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

می‌توان فوراً بعد از تعریف متغیرها مقادیری را به آن‌ها اختصاص داد. این عمل را مقداردهی می‌نامند. در زیر نحوه مقدار دهی متغیرها نشان داده شده است:

```
data_type identifier = value;
```

به عنوان مثال:

```
int myNumber = 7;
```

همچنین می‌توان 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 نمی‌توانید مقدار بدهید. شما فقط می‌توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند. مثلاً متغیرهای بالا همه قابل استفاده هستند. در این مثال num1 و num2 هر دو تعریف شده‌اند و مقادیری از نوع صحیح به آن‌ها اختصاص داده شده است. اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می‌دهد.

جانگهدار(Placeholders)

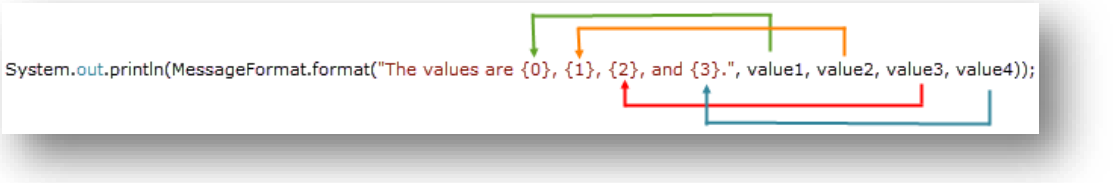
به متد format() از کلاس MessageFormat در خطوط (۳۲-۲۷) توجه کنید. برای استفاده از متد format() و کلاس MessageFormat ابتدا باید Package مربوط به آن‌ها را در برنامه وارد کنید (خط ۳):

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

این متد دو آرگومان قبول می‌کند. آرگومان‌ها اطلاعاتی هستند که متد با استفاده از آن‌ها کاری انجام می‌دهد. آرگومان‌ها به وسیله کاما از هم جدا می‌شوند. آرگومان اول یک رشته قالب بندی شده است و آرگومان دوم مقداری است که توسط رشته قالب بندی شده مورد استفاده قرار می‌گیرد. 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 می‌تواند از صفر تا n باشد. به این اعداد جانگهدار می گویند. این اعداد بوسیله مقدار آرگومان بعد جایگزین می‌شوند. به عنوان مثال جانگهدار {0} به این معناست که اولین آرگومان (مقدار) بعد از رشته قالب بندی شده در آن

قرار می‌گیرد. متد format() عملاً می‌تواند هر تعداد آرگومان قبول کند اولین آرگومان همان رشته قالب بندی شده است که جانگهدار در آن قرار دارد و دومین آرگومان مقداری است که جایگزین جانگهدار می‌شود. در مثال زیر از ۴ 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 می‌شوند. 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 می‌شود. در ابتدا فهمیدن این مفهوم برای کسانی که تازه برنامه نویسی را شروع کرده‌اند سخت است. اما در درس‌های آینده مثال‌های زیادی در این مورد مشاهده خواهید کرد.

ثابت

ثابت‌ها انواعی از متغیرها هستند که مقدار آن‌ها در طول برنامه تغییر نمی‌کند. ثابت‌ها حتماً باید مقدار دهی اولیه شوند و اگر مقدار دهی آن‌ها فراموش شود در برنامه خطا به وجود می‌آید. بعد از این که به ثابت‌ها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمی‌توان آن را تغییر داد. برای تعریف ثابت‌ها باید از کلمه کلیدی final استفاده کرد. معمولاً نام ثابت‌ها را طبق قرارداد با حروف بزرگ می‌نویسند 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 می‌بینید که مقدار دادن به یک ثابت، که قبلاً مقدار دهی شده برنامه را با خطا مواجه می‌کند. نکته‌ی 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 سؤال برایتان پیش آمده باشد که دلیل استفاده از ثابت‌ها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی‌کنند بهتر است که آن‌ها را به صورت ثابت تعریف کنید. این کار هر چند کم، کیفیت برنامه شما را بالا می‌برد.

تبدیل ضمنی

تبدیل ضمنی یا تبدیل بزرگ کننده یا widening conversion یک نوع تبدیل است که به طور خودکار انجام می‌شود. در این نوع تبدیل در صورتی یک متغیر از یک نوع داده می‌تواند به یک نوع دیگر تبدیل شود که مقدار آن از مقدار داده ای که می‌خواهد به آن تبدیل شود کمتر باشد. به عنوان مثال نوع داده ای byte می‌تواند مقادیر ۰ تا ۲۵۵ را در خود ذخیره کند و نوع داده ای int مقادیر ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را شامل می‌شود. پس می‌توانید یک متغیر از نوع byte را به یک نوع int تبدیل کنید :

```
byte number1 = 5;

int number2 = number1;
```

در مثال بالا مقدار number1 برابر ۵ است در نتیجه متغیر number2 که یک متغیر از نوع صحیح است می‌تواند مقدار number1 را در خود ذخیره کند چون نوع صحیح از نوع بایت بزرگ‌تر است. پس متغیر number1 که یک متغیر از نوع بایت است می‌تواند 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 می‌شویم. اگر چه مقدار ۵ متغیر number1 در محدوده مقادیر byte یعنی اعداد بین ۲۵۵-۰ قرار دارد اما متغیری از نوع بایت حافظه کمتری نسبت به متغیری از نوع صحیح اشغال می‌کند. نوع byte شامل ۸ بیت یا ۸ رقم دودویی است در حالی که نوع int شامل ۳۲ بیت یا رقم باینری است.یک عدد باینری عددی متشکل از ۰ و ۱ است. برای مثال عدد ۵ در کامپیوتر به عدد باینری ۱۰۱ ترجمه می‌شود. بنابراین وقتی ما عدد ۵ را در یک متغیر از نوع بایت ذخیره می‌کنیم، عددی به صورت زیر نمایش داده می‌شود:

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می‌کنیم، به صورت زیر نمایش داده می‌شود:

```
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 می‌توان از تبدیل صریح استفاه کرد که در درس‌های آینده توضیح داده می‌شود. نکته دیگری که نباید فراموش شود این است که شما نمی‌توانید اعداد با ممیز اعشار را به یک نوع int تبدیل کنید چون این کار باعث از بین رفتن بخش اعشاری این اعداد می‌شود.

```
double number1 = 5.25;
int number2 = number1; //Error
```

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

```
byte > short > int > long > float > double
```

تبدیل صریح

تبدیل صریح یا تبدیل کوچک کننده یا Narrowing Casting نوعی تبدیل است که برنامه را مجبور می‌کند 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 می‌توانیم این عملیات را با استفاده از Cast انجام دهیم. Cast فقط نام دیگر تبدیل صریح است و دستور آن به صورت زیر است:

```
datatypeA variableA = value;
datatypeB variableB = (datatypeB)variableA;
```

همانطور که قبلاً 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 مثلاً از نوع double را به یک نوع int تبدیل می‌کنیم مقدار اعداد بعد از ممیز از بین می‌روند:

```
double number1 = 5.25;
int number2 = (int)number1;
System.out.println(number2);
```

```
5
```

خروجی کد بالا عدد ۵ است چون نوع داده ای int نمی‌تواند 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 می‌افتد؟ مانند تبدیل زیر که می‌خواهیم متغیر number1 را که دارای مقدار ۳۰۰ است را به نوع بایت تبدیل کنیم که محدود اعداد بین ۲۵۵-۰ را پوشش می‌دهد.

```
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 ۴۴ است. Byte فقط می‌تواند شامل اعداد ۰ تا ۲۵۵ باشد و نمی‌تواند مقدار ۳۰۰ را در خود ذخیره کند. حال می‌خواهیم ببینیم که چرا به جای عدد ۳۰۰ ما عدد ۴۴ را در خروجی می‌گیریم. این کار به تعداد بیت‌ها بستگی دارد. یک byte دارای ۸ بیت است در حالی که int دارای ۳۲ بیت است. حال اگر به مقدار باینری ۲ عدد توجه کنید متوجه می‌شوید که چرا خروجی عدد ۴۴ است.

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

خروجی بالا نشان می‌دهد که بیشترین مقدار byte که عدد ۲۵۵ است می‌تواند فقط شامل ۸ بیت باشد (۱۱۱۱۱۱۱۱) بنابراین فقط ۸ بیت اول مقدار int به متغیر byte انتقال می‌یابد که شامل (۰۰۱۰۱۱۰۰) یا عدد ۴۴ در مبنای ۱۰ است.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید:

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

مثلاً X+Y یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می‌آیند. زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیند. جاوا دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسه ای، منطقی و بیتی می‌باشد. از عملگرهای ساده ریاضی می‌توان 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 می‌گیرند، عبارت‌اند از:

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

- عملگرهای بیتی

عملگرهای ریاضی

جاوا از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی جاوا را نشان می‌دهد:

عملگر	دسته	مثال	نتیجه
+	Binary	var1 = var2 + var3;	Var1 برابر است با حاصل جمع var2 و var3
-	Binary	var1 = var2 – var3;	Var1 برابر است با حاصل تفریق var2 و var3
*	Binary	var1 = var2 * var3;	Var1 برابر است با حاصلضرب var2 در var3
/	Binary	var1 = var2 / var3;	Var1 برابر است با حاصل تقسیم var2 بر var3
%	Binary	var1 = var2 % var3;	Var1 برابر است با باقیمانده تقسیم var2 و var3
+	Unary	var1 = +var2;	Var1 برابر است با مقدار var2
-	Unary	var1 = -var2;	Var1 برابر است با مقدار var2 ضربدر ۱ -

دیگر عملگرهای جاوا عملگرهای کاهش و افزایش هستند.این عملگرها مقدار ۱ را از متغیرها کم یا به آن‌ها اضافه می‌کنند. از این متغیرها اغلب در حلقه‌ها استفاده می‌شود:

عملگر	دسته	مثال	نتیجه
++	Unary	var1 = ++var2;	مقدار var1 برابر است با var2 بعلاوه ۱
--	Unary	var1 = --var2;	مقدار var1 برابر است با var2 منهای ۱
++	Unary	var1 = var2++;	مقدار var1 برابر است با var2. به متغیر var2 یک واحد اضافه می‌شود .
--	Unary	var1 = var2--;	مقدار var1 برابر است با var2. از متغیر var2 یک واحد کم می‌شود .

به این نکته توجه داشته باشید که محل قرار گیری عملگر در نتیجه محاسبات تأثیر دارد. اگر عملگر قبل از متغیر var2 بیاید افزایش یا کاهش var1 اتفاق می‌افتد. چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2 می‌شود و سپس متغیر var2افزایش یا کاهش می‌یابد. به مثال‌های 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 می‌کنید،درج عملگرهای -- و ++ قبل از عملوند y باعث می‌شود 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 می‌کنید،در عملگرهای -- و ++ بعد از عملوند y باعث می‌شود که ابتدا مقدار y در داخل متغیر x قرار بگیرد و سپس یک واحد از y کم و یا یک واحد به آن اضافه شود. حال می‌توانیم 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 نتیجه هر عبارت را نشان می‌دهد. در این برنامه از متد println() برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. در این مثال با یک نکته عجیب مواجه می‌شویم 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 می‌کنید اگر فقط یکی از اعداد را به نوع اعشاری double تبدیل کنیم (در مثال می‌بینید) حاصل به صورت اعشار نشان داده می‌شود.

عملگرهای تخصیصی

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

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 می‌توان استفاده کرد.استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می‌باشد. این حالت کدنویسی زمانی کارایی خود را نشان می‌دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آن‌ها را بر متغیرها نشان می‌دهد.

```
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 ۳ عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار ۱۰ با استفاده از عملگر = به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر =+ مقدار ۱۰ اضافه شده است.و در آخر به وسیله عملگر =- عدد ۱۰ از آن کم شده است.

عملگرهای مقایسه ای

از عملگرهای مقایسه ای برای مقایسه مقادیر استفاده می‌شود. 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 می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه ای در جاوا را نشان می‌دهد:

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 مقدار var2 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 این عملگرها را نشان می‌دهد:

```
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 می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آن‌ها مقادیری اختصاص می‌دهیم.سپس با استفاده از یک عملگر مقایسه ای آن‌ها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. 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 x اختصاص می‌دهد. عملگر == عملگر مقایسه ای است که دو مقدار را با هم مقایسه می‌کند مانند x==y و اینطور خوانده می‌شود x برابر است با y .

عملگرهای منطقی

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

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

عملگر منطقی (&&)AND

اگر مقادیر دو طرف عملگر AND، true باشند عملگر AND مقدار true را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آن‌ها false باشند مقدار false را بر می‌گرداند. 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 تأثیر عملگر AND یادآوری می‌کنم که این عملگر فقط در صورتی مقدار true را نشان می‌دهد که هر دو عملوند مقدارشان true باشد.در غیر اینصورت نتیجه تمام ترکیب‌های بعدی false خواهد شد. استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه ای است. به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگ‌تر از ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

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

عملگر AND زمانی کارامد است که ما با محدود خاصی از اعداد سرو کار داریم. مثلاً عبارت 10 <= x <= 100 بدین معنی است که x می‌تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان از عملگر منطقی AND به 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 می‌گرداند. جدول درستی عملگر OR در 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 می‌کنید 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 ۷۵ یا نمره نهایی امتحان آن ۱۰۰ 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 می‌کند. مثلاً اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد، آنرا true می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد :

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که age (سن) بزرگ‌تر یا مساوی ۱۸ نباشد.

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

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می‌دهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را از لینک زیر یاد بگیرید :

```
http://www.w3-farsi.com/?p=5698
```

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند، وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود.بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع int به ۳۲ بیت یا ۸ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع intذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود:

```
00000000000000000000000001100100
```

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ 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 می‌شوند. عملگرهای بیتی جاوا در جدول زیر نشان داده شده‌اند:

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

^=	بیتی تخصیصی XOR	Binary	x ^= y;

عملگر بیتی (&)AND

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیت‌ها کار می‌کند.اگر مقادیر دو طرف آن ۱ باشد مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. 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 می‌کنید نتیجه عملکرد عملگر AND بر روی دو مقدار ۵ و ۳ عدد یک می‌شود.اجازه بدهید ببینیم که چطور این نتیجه را به دست می‌آید:

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

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح ۳۲ بیت است از صفر برای پر کردن بیت‌های خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی AND می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی (|)OR

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه صفر در غیر اینصورت ۱ 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 عملوندهای 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 (۷ و ۹) استفاده می‌کنیم نتیجه ۱۵ می‌شود.حال بررسی می‌کنیم که چرا این نتیجه به دست آمده است؟

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

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد.عدد ۱۱۱۱ باینری معادل عدد ۱۵ 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 می‌شود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاده می‌کنید:

```
int result = 5 ^ 7;

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

```
2
```

در زیر معادل باینری اعداد بالا (۵ و ۷) نشان داده شده است.

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

با نگاه کردن به جدول درستی عملگر بیتیXOR، می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

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 می‌کند.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 می‌دهند 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 می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت‌ها را نشان می‌دهد.

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

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 می‌کند.

به عنوان مثال:

```
int result = 10 << 2;

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

```
40
```

در مثال بالا ما بیت‌های مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیایید تأثیر این انتقال را بررسی کنیم:

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

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عمگر تغییر مکان به سمت چپ است با این تفاوت که بیت‌ها را به سمت راست جا به جا می‌کند. به عنوان مثال:

```
int result = 100 >> 4;

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

```
6
```

با استفاده از عملگر تغییر مکان به سمت راست بیت‌های مقدار ۱۰۰ را به اندازه ۴ واحد به سمت چپ جا به جا می‌کنیم. اجازه بدهید تأثیر این جا به جایی را مورد بررسی قرار دهیم:

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

هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

تقدم عملگرها

تقدم عملگرها مشخص می‌کند که در محاسباتی که بیش از دو عملوند دارند ابتدا کدام عملگر اثرش را اعمال کند. عملگرها 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 ۹ خواهد شد (۳=۲+۱ سپس ۹=۳×۳ و در آخر ۹=۹/۱). اما کمپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد.برای مثال عمل ضرب و تقسیم نسبت به

جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد ۲ ضربدر ۳ و سپس نتیجه آن‌ها تقسیم بر ۱ می‌شود که نتیجه ۶ به دست می‌آید.در آخر عدد ۶ با ۱ جمع می‌شود و عدد ۷ حاصل می‌شود. در جدول زیر تقدم برخی از عملگرهای جاوا آمده است:

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

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین‌ترین حق تقدم در محاسبات تأثیر می‌گذارند.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 ۴ می‌شود و سپس مقدار جدید با عدد ۳ جمع می‌شود و در نهایت عدد ۷ به دست می‌آید. در عبارت دوم مقدار عددی ۳ به مقدار number اضافه می‌شود و عدد ۶ به دست می‌آید. سپس این مقدار

در متغیر number2 قرار می‌گیرد. و در نهایت مقدار number به ۴ افزایش می‌یابد. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آن‌ها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم:

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

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد، توجه کنید.در این عبارت ابتدا مقدار داخلی‌ترین پرانتز مورد محاسبه قرار می‌گیرد. یعنی مقدار ۶ ضربدر ۷ شده و سپس از ۵ کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند 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 تأثیر دهید. یعنی ابتدا ۳ را ضربدر ۲ می‌کنید و سپس عدد ۸ را بر ۴ تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می‌دهید.

گرفتن ورودی از کاربر

جاوا تعدادی متد برای گرفتن ورودی از کاربر در اختیار شما قرار می‌دهد. 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 می‌کنیم. این دو کلاس را در خطوط ۳ و ۴ وارد کرده‌ایم. متدهای کلاس Scanner که مقادیر وارد شده توسط کاربر را از صفحه کلید می‌خوانند عبارت‌اند از:

متد	توضیح
nextByte()	برای دریافت یک نوع داده از نوع byte به کار می‌رود.
nextShort()	برای دریافت یک نوع داده از نوع short به کار می‌رود.

nextInt()	برای دریافت یک نوع داده از نوع int به کار می‌رود.
nextLong()	برای دریافت یک نوع داده از نوع long به کار می‌رود.
next()	برای دریافت یک کلمه ساده به کار می‌رود.
nextLine()	برای دریافت یک خط رشته به کار می‌رود.
nextBoolean()	برای دریافت یک نوع داده از نوع boolean به کار می‌رود.
nextFloat()	برای دریافت یک نوع داده از نوع float به کار می‌رود.
nextDouble()	برای دریافت یک نوع داده از نوع double به کار می‌رود.

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 ۳ و ۴ برنامه، کلاس Scanner و MessageFormat را با استفاده از کلمه import به برنامه اضافه کرده‌ایم. در خطوط ۱۰ و ۱۱ و ۱۲ 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 نموده‌ایم. درباره خط ۱۴ زیاد توضیح نمی‌دهم، فقط کافیست که این رابدانید که وجود این خط برای دریافت ورودی از کاربر اجباری است. در درس‌های آینده با مفاهیم متد، شیء و کلاس آشنا خواهید شد. برنامه از کاربر می‌خواهد که نام خود را وارد کند (خط ۱۶). در خط ۱۷ شما به عنوان کاربر نام خود را وارد می‌کنید. مقدار متغیر name، برابر مقداری است که توسط متد ()next خوانده می‌شود.از آنجاییکه name از نوع رشته است باید از متد ()next برای دریافت استفاده کنیم. در خط ۱۹ برنامه از شما می‌خواهد که سن خود را وارد کند. در خط ۲۰ شما سن خود را وارد می‌کنید. مقدار متغیر age، برابر مقداری است که توسط متد ()nextInt خوانده می‌شود.از آنجاییکه Age از نوع صحیح است باید از متد ()nextInt برای دریافت استفاده کنیم. سپس برنامه از ما قد را سؤال می‌کند (خط ۲۲). چون height از نوع double است پس برای خواندن آن از متد ()nextDouble استفاده کرده‌ایم. حال شما می‌توانید با اجرای برنامه و وارد کردن مقادیر نتیجه را مشاهده کنید.

ساختارهای تصمیم

تقریباً همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. 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 آن وقت با مشکل مواجه خواهید شد. جاوا راه‌های مختلفی برای رفع این نوع مشکلات ارائه می‌دهد. در این بخش با مطالب زیر آشنا خواهید شد:

- دستور if
- دستور if…else
- عملگر سه تایی

- دستور if چندگانه
- دستور if تو در تو
- عملگرهای منطقی
- دستور switch

دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید.دستور if ساده‌ترین دستور شرطی است که به برنامه می‌گوید، اگر شرطی برقرار است کد معینی را انجام بده. ساختار دستور if به صورت زیر است:

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

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود.اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه ای است. می‌توان از عملگرهای مقایسه ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```
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 ۸ یک متغیر با نام number تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور if در خط ۱۱ می‌رسیم برنامه تشخیص می‌دهد که مقدار number از ۱۰ کمتر است یعنی ۵ کوچک‌تر از ۱۰ است.

منطقی است که نتیجه مقایسه درست می‌باشد بنابراین دستور if دستور را اجرا می‌کند(خط ۱۲) و پیغام Hello World چاپ می‌شود. حال مقدار number را به ۱۵ تغییر می‌دهیم (خط ۱۶). وقتی به دومین دستور if در خط ۱۹ می‌رسیم برنامه مقدار number را با ۱۰ مقایسه می‌کند و چون مقدار number یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند(خط ۲۰).

به این نکته توجه کنید که دستور if را می‌توان در یک خط نوشت:

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

شما می‌توانید 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 if به 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 ۱۰ بزرگ‌تر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار x از ۱۰ 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 می‌شود 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?)");
```

می‌بیند که دستور دوم (خط ۳) در مثال بالا جز دستور if نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار x از ۱۰ کوچک‌تر است پس خط `This is still part of the if statement. (Really?)` چاپ می‌شود. در نتیجه اهمیت وجود آکولاد مشخص می‌شود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه if داشتید برای آن یک آکولاد بگذارید. فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطا شده و یافتن آن را سخت می‌کند. یکی از خطاهای معمول کسانی که برنامه نویسی را تازه شروع کرده‌اند 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 if یک مقایسه را انجام می‌دهد و دستور اجرایی نیست. بنابراین برنامه شما با یک خطای منطقی مواجه می‌شود. همیشه به یاد داشته باشید که قرار گرفتن سیمیکولن در سمت راست پرانتز if به 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 می‌توانید نتیجه یک عبارت را در داخل یک متغیر بولی ذخیره کنید و سپس از متغیر به عنوان شرط در دستور if استفاده کنید. اگر مقدار year برابر ۲۰۰۰ باشد سپس حاصل عبارت در متغیر isNewMillenium ذخیره می‌شود.می‌توان از متغیر برای تشخیص کد اجرایی بدنه دستور if استفاده کرد خواه مقدار متغیر درست باشد یا نادرست.

```
boolean isNewMillenium = year == 2000;

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

دستور if else

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود. یعنی اگر حالتی برقرار بود کارخاصی انجام شود. اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود، دستور دیگر اجرا شود، باید از دستور if else استفاده کنید.

ساختار دستور if else در زیر آمده است:

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

از کلمه کلیدی else نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با if به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه if و بدنه else دارید استفاده از آکولاد اختیاری است. کد داخل بلوک else فقط در صورتی اجرا می‌شود که شرط داخل دستور ifنادرست باشد. در زیر نحوه استفاده از دستور if…else آمده است.

```
1  package myfirstprogram;
2
3  public class MyFirstProgram
4  {
5      public static void main(String[] args)
6      {
7          int number = 5;
8
9          //Test the condition
10         if (number < 10)
11         {
12             System.out.println("The number is less than 10.");
13         }
14         else
15         {
16             System.out.println("The number is either greater than or equal to 10.");
17         }
18
19         //Modify value of number
20         number = 15;
21
22         //Repeat the test to yield a different result
23         if (number < 10)
24         {
25             System.out.println("The number is less than 10.");
26         }
27         else
28         {
29             System.out.println("The number is either greater than or equal to 10.");
30         }
31     }
```

```
32  }
```

در خط ۷ یک متغیر به نام number تعریف کرده‌ایم و در خط ۱۰ تست می‌کنیم که آیا مقدار متغیر number از ۱۰ کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک if اجرا می‌شود (خط ۱۲) و اگر مقدار number را تغییر دهیم و به مقداری بزرگ‌تر از ۱۰ تغییر دهیم (خط ۲۰)، شرط نادرست می‌شود (خط ۲۳) و کد داخل بلوک else اجرا می‌شود (خط ۲۹). مانند بلوک if نباید به آخر کلمه کلیدی else سیمیکولن اضافه شود.

دستور if تو در تو

می‌توان 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)
```

```
            {
                System.out.println("You are teenage");
            }
            else
            {
                System.out.println("You are already an adult.");
            }
        }
        else
        {
            System.out.println("You are still too young.");
        }
    }
}
```

```
You are already an adult.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا در خط ۷ یک متغیر به نام age تعریف می‌کنیم و مقدار آن را برابر ۲۱ قرار می‌دهیم.سپس به اولین دستور if می‌رسیم (خط ۹). در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنه دستور if می‌شود در غیر اینصورت وارد بلوک else (خط ۲۰) مربوط به همین دستور if می‌شود. حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین if شده‌اید. در بدنه اولین if یک دستور if دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد دستور You are teenage چاپ می‌شود (خط ۱۳) در غیر اینصورت دستور You are already an adult (خط ۱۷) و چون مقدار متغیر تعریف شده در خط ۷ بزرگ‌تر از ۲۰ است پس دستور مربوط به بخش else خط ۱۷ چاپ می‌شود. حال فرض کنید که مقدار متغیر age کمتر از ۱۲ بود، در این صورت دستور بخش else خط ۲۰ یعنی You are still too young چاپ می‌شد. پیشنهاد می‌شود که از if تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد

عملگر شرطی

عملگر شرطی (?:) در جاوا مانند دستور شرطی if…else عمل می‌کند. 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 به 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.

```
package myfirstprogram;

```

```
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 می‌دهد. در این برنامه قصد ما به دست آوردن قدر مطلق یک عدد است. ابتدا در خط ۷ یک متغیر از نوع int تعریف کرده و مقدار آن را ۱۰- می‌گذاریم. در خط ۹ یک متغیر از نوع صحیح تعریف کرده‌ایم تا نتیجه را در آن قرار دهیم. خط ۹ به این صورت تعریف می‌شود : " اگر مقدار number از ۱۰ 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 می‌نویسیم:

```
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 می‌آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده 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. می‌توانید 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 می‌شود که اولین دستور if اشتباه باشد حال اگر else if اشتباه باشد دستور else if بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می‌شود. برنامه زیر نحوه استفاده از دستور if else را نشان می‌دهد:

```
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 می‌کنید پیغام‌های مختلفی چاپ می‌شود.

اگر عددی که شما تایپ می‌کنید در داخل حالت‌های انتخاب نباشد کد مربوط به بلوک else اجرا می‌شود.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را درگیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است:

عملگر	تلفظ	مثال	تأثیر
&&	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 false، z 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 false، z 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 ۲ و مقدارy کوچک‌تر از ۱۰ 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 تأثیر 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. می‌توان 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 می‌کنیم. در اینجا ابتدا عبارت (y > 3) || (z < 10) مورد بررسی قرار می‌گیرد (به علت تقدم عملگرها). سپس نتیجه آن بوسیله عملگر AND با نتیجه (x == 1) مقایسه می‌شود.

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 می‌دهد (خط ۲۰). وقتی به دستور if می‌رسید (خط ۲۰) برنامه سن شما را چک می‌کند. اگر سن شما بزرگ‌تر از ۱۲ و کوچک‌تر از ۲۰ باشد (سنتان بین ۱۲ و ۲۰ باشد) یعنی مقدار هر دو true باشد، سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می‌شود. عملگر AND عملوند سمت چپ را مرود بررسی قرار می‌دهد. اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار false را بر می‌گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار true را بر می‌گرداند.

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

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

نکته مهم اینجاست که شما می‌توانید از عملگرهای & و | به عنوان عملگر بیتی استفاده کنید. تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می‌روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد عملوند سمت چپ به وسیله عملگر بیتی (&) AND ارزیابی می‌شود. 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 می‌کند. 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 می‌کند.

دستور switch

در جاوا ساختاری به نام switch وجود دارد که به شما اجازه می‌دهد 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، رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقادیری هستند که قابل تغییر 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 می‌دهید. این مقدار با هر یک از عبارت‌های case داخل بلوک switch مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات case برابر بود کد مربوط به آن case اجرا 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 می‌شود که باعث می‌شود برنامه از دستور switch خارج شده و دستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه می‌شود. دستور switch یک بخش default دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر دستورات case برابر نباشد. دستور default

اختیاری است و اگر از بدنه switch حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور switch توجه کنید:

```java
package myfirstprogram;

import java.util.Scanner;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        int choice;

        System.out.println("What's your favorite pet?");
        System.out.println("[1] Dog");
        System.out.println("[2] Cat");
        System.out.println("[3] Rabbit");
        System.out.println("[4] Turtle");
        System.out.println("[5] Fish");
        System.out.println("[6] Not in the choices");
        System.out.print("Enter your choice: ");

        choice = input.nextInt();

         switch  (choice)
        {
            case 1:
                System.out.println("Your favorite pet is Dog.");
                break;
            case 2:
                System.out.println("Your favorite pet is Cat.");
                break;
            case 3:
                System.out.println("Your favorite pet is Rabbit.");
                break;
            case 4:
                System.out.println("Your favorite pet is Turtle.");
                break;
            case 5:
                System.out.println("Your favorite pet is Fish.");
                break;
            case 6:
                System.out.println("Your favorite pet is not in the choices.");
                break;
            default:
                System.out.println("You don't have a favorite pet.");
                break;
        }
    }
}
```

```
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 علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور switch با مقادیر case مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر case ها برابر نبود دستور default اجرا می‌شود. یکی دیگر از ویژگیهای دستور switch این است که شما می‌توانید از دو یا چند case برای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار number، ۱، ۲ یا ۳ باشد یک کد اجرا می‌شود. توجه کنید که case ها 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 قبلاً ذکر شد دستور switch معادل دستور if تو در تو است. برنامه بالا را به صورت زیر نیز می‌توان 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 دقیقاً نتیجه ای مانند دستور switch دارد. دستور default معادل دستور else می‌باشد. حال از بین این دو دستور (switch و if else) کدامیک را انتخاب کنیم. از دستور switch موقعی استفاده می‌کنیم که مقداری که می‌خواهیم با دیگر مقادیر مقایسه شود ثابت باشد. مثلاً 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 می‌کنید که با اینکه مقدار x عدد ۵ است و به طور واضح با متغیر myNumber مقایسه شده است برنامه خطا می‌دهد چون x یک 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 می‌شود.توجه کنید که بعد از تعریف یک ثابت نمی‌توان مقدار آن را در طول برنامه تغییر داد. به یاد داشته باشید که باید ثابت‌ها را حتماً مقداردهی کنید. دستور switch یک مقدار را با مقادیر case ها مقایسه می‌کند 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 می‌دهند 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.مثلاً شما مجبورید ۱۰ 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.");
System.out.println("Hello World.");
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. راه بهتر برای نوشتن کدهای بالا استفاده از حلقه‌ها است. ساختارهای تکرار در جاوا عبارت‌اند از:

- while
- do while
- for

حلقه While

ابتدایی‌ترین ساختار تکرار در جاوا حلقه while است.ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار حلقه while به صورت زیر است:

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

می‌بینید که ساختار while مانند ساختار if بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است می‌نویسیم اگر نتیجه درست یا true باشد سپس کدهای داخل بلوک while اجرا می‌شوند. اگر شرط غلط یا false باشد وقتی که برنامه به حلقه while برسد هیچکدام از کدها را اجرا نمی‌کند. 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 می‌گیرد 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 ۱۰ بار پیغام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم مجبور بودیم تمام ۱۰ خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط ۷ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است.سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد نمی‌توان در شرط از آن استفاده کرد.

در خط ۹ حلقه while را وارد می‌کنیم. در حلقه while ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه while و چاپ پیغام است. همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۱۲). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بین‌هایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت > از => استفاده شده است. اگر از علامت > استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ برسد false می‌شود چون ۱۰ > ۱۰ نیست. اگر می‌خواهید 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 break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

حلقه do 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 می‌گیرد. ساختار حلقه do while به صورت زیر است:

```
do
{
   code to repeat;

} while (condition);
```

همانطور که مشاهده می‌کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می‌شوند. برخلاف حلقه while که اگر شرط نادرست باشد دستورات داخل بدنه اجرا نمی‌شوند. یکی از موراد برتری استفاده از حلقه do while نسبت به حلقه while زمانی است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. به مثال زیر توجه کنید:

استفاده از while

```
//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 می‌کنید که از کدهای کمتری در بدنه do while نسبت به while استفاده شده است.

حلقه for

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

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

مقدار دهی اولیه (initialization) اولین مقداری است که به شمارنده حلقه می‌دهیم.شمارنده فقط در داخل حلقه for قابل دسترسی است. شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می‌کند و تعیین می‌کند که حلقه ادامه یابد یا نه. عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می‌دهد. 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 ۱ تا ۱۰ را با استفاده از حلقه for می‌شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می‌کنیم و آن را با مقدار ۱ مقدار دهی اولیه می‌کنیم. سپس با استفاده از شرط آن را با مقدار ۱۰ مقایسه می‌کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (i++) فوراً اجرا نمی‌شود. کد اجرا می‌شود و ابتدا رشته Number و سپس مقدار جاری i یعنی ۱ را چاپ می‌کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر ۲ می‌شود و بار دیگر i با عدد ۱۰ مقایسه می‌شود و این حلقه تا زمانی که مقدار شرط true شود ادامه می‌یابد. 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 ۱۰ به ۱ چاپ می‌کند (از بزرگ به کوچک). مقدار اولیه شمارنده را ۱۰ می‌دهیم و با استفاده از عملگر کاهش (--) برنامه ای که شمارش معکوس را انجام می‌دهد ایجاد می‌کنیم. می‌توان قسمت شرط و عملگر را به صورت‌های دیگر نیز تغییر داد. به عنوان مثال می‌توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد. همچنین می‌توانید 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 می‌کنید باید آن‌ها را با استفاده از کاما از هم جدا کنید. گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال 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 می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد:

```
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 حلقه‌های while و do…while استفاده می‌شد نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط ۱۱) آمده است وقتی که مقدار x به عدد ۵ رسید سپس دستور break اجرا شود(خط ۱۲). حلقه بلافاصله متوقف می‌شود حتی اگر شرط x < 10 برقرار باشد.از طرف

دیگر در خط ۲۲ حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد. وقتی مقدار x برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند.

آرایه‌ها

آرایه نوعی متغیر است که لیستی از آدرس‌های مجموعه ای از داده‌های هم نوع را در خود ذخیره می‌کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آن‌ها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می‌توان 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];
```

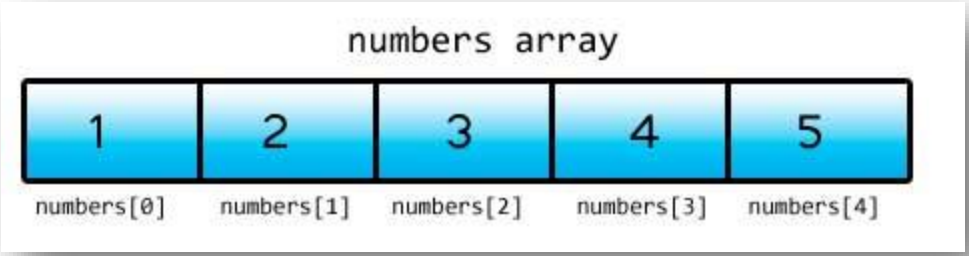
Datatype نوع داده‌هایی را نشان می‌دهد که آرایه در خود ذخیره می‌کند. کروشه که بعد از نوع داده قرار می‌گیرد و نشان دهنده استفاده از آرایه است. arrayName که نام آرایه را نشان می‌دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه ای که اعداد را در خود ذخیره می‌کند از کلمه number استفاده کنید. طول آرایه که به کمپایلر می‌گوید 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 می‌شود. برای تعریف یک آرایه که ۵ مقدار از نوع اعداد صحیح در خود ذخیره می‌کند باید به صورت زیر عمل کنیم:

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

در این مثال ۵ آدرس از فضای حافظه کامپیوتر شما برای ذخیره ۵ مقدار رزرو می‌شود. حال چطور مقادیرمان را در هر یک از این آدرس‌ها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آن‌ها استفاده می‌شود.

```
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 می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است پس ۱-۵ برابر است با ۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ 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 کرده‌اند معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیس‌ها را از ۱ 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 ArrayIndexOutOfBoundsException مواجه می‌شوید و بدین معنی است که شما آدرسی را می‌خواهید 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 می‌توان مقادیر را در داخل آکولاد قرار داد. کمپایلر به صورت اتوماتیک با شمارش مقادیر طول آرایه را تشخیص می‌دهد.

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آن‌ها حساب می‌شود:

```
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 ۱۲ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. خطوط ۱۳ و ۱۴ متغیرهایی تعریف شده‌اند که از آن‌ها برای محاسبه میانگین استفاده می‌شود. توجه کنید که مقدار اولیه `total` صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۱۶ تا ۲۰ حلقه `for` برای تکرار و گرفتن ورودی از کاربر تعریف شده است.از خاصیت طول (`length`) آرایه برای تشخیص تعداد اجزای آرایه استفاده می‌شود.اگر چه می‌توانستیم به سادگی در حلقه `for` مقدار ۵ را برای شرط قرار دهیم ولی استفاده از خاصیت طول آرایه کار راحت تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه `for` با تغییر جدید هماهنگ می‌شود. در خط ۱۹ ورودی دریافت شده از کاربر با استفاده از متد `nextInt()` دریافت و در آرایه ذخیره می‌شود. اندیس استفاده شده در `number` (خط ۱۹) مقدار `i` جاری در حلقه است. برای مثال در ابتدای حلقه مقدار `i` صفر است بنابراین وقتی در خط ۱۹ اولین داده از کاربر گرفته می‌شود، اندیس آن برابر صفر می‌شود. در تکرار بعدی `i` یک واحد اضافه می‌شود و در نتیجه در خط ۱۹ و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می‌شود. این حالت تا زمانی که شرط در حلقه `for` برقرار است ادامه می‌یابد. در خطوط ۲۴-۲۱ از حلقه `for` دیگر برای دسترسی به مقدار هر یک از داده‌های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می‌کنیم.

هر یک از اجزای عددی آرایه به متغیر total اضافه می‌شوند. بعد از پایان حلقه می‌توانیم میانگین اعداد را حساب کنیم (خط ۲۶). مقدار total را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می‌کنیم. برای دسترسی به تعداد اجزای آرایه می‌توان از خاصیت length آرایه استفاده کرد. توجه کنید که در اینجا ما مقدار خاصیت length را به نوع double تبدیل کرده‌ایم بنابراین نتیجه عبارت یک مقدار از نوع double خواهد شد و دارای بخش کسری می‌باشد. حال اگر عملوندهای تقسیم را به نوع double تبدیل نکنیم نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط ۲۸ مقدار میانگین را در صفحه نمایش چاپ می‌کند. طول آرایه بعد از مقدار دهی نمی‌تواند تغیر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جز است مقدار دهی کنید دیگر نمی‌توانید آن را مثلاً به ۱۰ جز تغییر اندازه دهید. البته تعداد خاصی از کلاس‌ها مانند آرایه‌ها عمل می‌کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه‌ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آن‌ها استفاده کنید بسیار مهم است.

حلقه foreach

حلقه foreach یکی دیگر از ساختارهای تکرار در جاوا می‌باشد که مخصوصاً برای آرایه‌ها، لیست‌ها و مجموعه‌ها طراحی شده است. حلقه foreach با هر بار گردش در بین اجزاء، مقادیر هر یک از آن‌ها را در داخل یک متغیر موقتی قرار می‌دهد و شما می‌توانید 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 می‌کند. temporaryVar 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 می‌نویسیم. در زیر نحوه استفاده از حلقه foreach آمده 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 ۵ جزء تعریف شده و مقادیر ۱ تا ۵ در آن‌ها قرار داده شده است (خط ۷). در خط ۹ حلقه foreach شروع می‌شود. ما یک متغیر موقتی تعریف کرده‌ایم که اعداد آرایه را در خود ذخیره می‌کند. در هر بار تکرار از حلقه foreach متغیر موقتی n، مقادیر عددی را از آرایه استخراج می‌کند. حلقه foreach مقادیر اولین تا آخرین جزء آرایه را در اختیار ما قرار می‌دهد.

حلقه foreach برای دریافت هر یک از مقادیر آرایه کاربرد دارد. بعد از گرفتن مقدار یکی از اجزای آرایه، مقدار متغیر موقتی را چاپ می‌کنیم. حلقه foreach ما را قادر می‌سازد که به داده‌ها 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
```

آرایه‌های چند بعدی

آرایه‌های چند بعدی، آرایه‌هایی هستند که برای دسترسی به هر یک از عناصر آن‌ها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می‌توان مانند یک جدول با تعدای ستون و ردیف تصور کنید. با افزایش اندیس‌ها اندازه ابعاد آرایه نیز افزایش می‌یابد و آرایه‌های چند بعدی با بیش از دو اندیس به وجود می‌آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است:

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

و یک آرایه سه بعدی به صورت زیر ایجاد می‌شود:

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

می توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. به دلیل اینکه آرایه‌های سه بعدی یا آرایه‌های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می‌گیرند. اجازه بدهید که در این درس بر روی آرایه‌های دو بعدی تمرکز کنیم. در تعریف این نوع آرایه ابتدا نوع آرایه یعنی اینکه آرایه چه نوعی از انواع داده را در خود ذخیره می‌کند را، مشخص می‌کنیم. سپس دو جفت کروشه قرار می‌دهیم. به تعداد کروشه‌ها توجه کنید. اگر آرایه ما دو بعدی است باید ۲ جفت کروشه و اگر سه بعدی است باید ۳ 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 می‌کنیم. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی مقدار Xو 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 می‌توان به صورت یک مکعب تصور کرد که دارای سه بعد است و x طول، Y عرض و z ارتفاع آن است. یک مثال از آرایه دو بعدی در زیر آمده است:

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

کد بالا به کمپایلر می‌گوید که فضای کافی به عناصر آرایه اختصاص بده (در این مثال ۱۵ خانه). در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.

مقدار ۳ را به x اختصاص می‌دهیم چون ۳ سطر و مقدار ۵ را به ۷ چون ۵ ستون داریم اختصاص می‌دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ چند راه برای مقدار دهی به آرایه‌ها 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 می‌توان 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 می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیس‌های X و Y و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه‌های چند بعدی

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

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int[][] numbers = { { 1, 2, 3, 4, 5 },
                            { 6, 7, 8, 9, 10 },
                            { 11, 12, 13, 14, 15 }
                          };

        for (int[] number : numbers)
        {
            for (int num : number)
            {
                System.out.print(num + " ");
            }
        }
    }
}
```

```
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 نمی‌توانیم انتهای ردیف‌ها را مشخص کنیم. برنامه زیر نشان می‌دهد که چطور از حلقه for برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف‌ها استفاده کنید.

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int[][] numbers = { { 1, 2, 3, 4, 5 },
                            { 6, 7, 8, 9, 10 },
                            { 11, 12, 13, 14, 15 }
                          };

        for (int row = 0; row < numbers.length; row++)
        {
            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 for نمی‌توان به مقادیر دسترسی یافت بلکه به یک حلقه for تو در تو نیاز داریم. در اولین حلقه for (خط ۱۲) یک متغیر تعریف شده است که در میان ردیف‌های آرایه (row) گردش می‌کند. این حلقه تا زمانی ادامه می‌یابد که مقدار ردیف کمتر از طول اولین بعد باشد. در این مثال از خاصیت lenght کلاس Array استفاده کرده‌ایم. این خاصیت طول آرایه را در یک بعد خاص نشان می‌دهد. به عنوان مثال برای به دست آوردن طول اولین بعد آرایه که همان تعداد ردیف‌ها می‌باشد از دستور numbers.length استفاده کرده‌ایم.

در داخل اولین حلقه for حلقه for دیگری تعریف شده است (خط ۱۴). 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 می‌کنیم. همانطور که مشاهده می‌کنید ابتدا نام آرایه را نوشته‌ایم و سپس یک اندیس به آن اختصاص داده‌ایم و بعد از خاصیت lenght استفاده نموده‌ایم:

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

استفاده از row به عنوان اندیس باعث می‌شود که به عنوان مثال وقتی که مقدار ردیف (row) صفر باشد، حلقه دوم از [۰][۰] تا [۴][۰] اجرا شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (row) برابر ۰ و مقدار ستون (col) برابر ۰ باشد مقدار عنصری که در ستون ۱ و ردیف ۱ (numbers[0][0]) قرار دارد، نشان داده خواهد شد که در مثال بالا عدد ۱ است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور System.out.println() که به برنامه اطلاع می‌دهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار row

این فرایند را دوباره تکرار می‌کند. سپس دومین حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. این فرایند تا زمانی اجرا می‌شود که مقدار row کمتر از طول اولین بعد باشد. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار بریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```java
package myfirstprogram;

import java.util.Scanner;
import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        double[][] studentGrades = new double[3][4];
        double total;

        for (int student = 0; student < studentGrades.length; student++)
        {
            total = 0;

            System.out.println(MessageFormat.format("Enter grades for Student {0}", student + 1));

            for (int grade = 0; grade < studentGrades[student].length; grade++)
            {
                System.out.print(MessageFormat.format("Enter Grade #{0}: ", grade + 1));
                studentGrades[student][grade] = input.nextDouble();
                total += studentGrades[student][grade];
            }

            System.out.print(
              MessageFormat.format("Average is {0}", (total / studentGrades[student].length)));
            System.out.println();
        }
    }
}
```

```
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 ۱۲). همچنین یک متغیر به نام total تعریف می‌کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم. حال وارد حلقه for تو در تو می‌شویم (خط ۱۵). در اولین حلقه for یک متغیر به نام sudent برای تشخیص پایه درسی هر دانش آموز تعریف کرده‌ایم. از خاصیت lenght هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه for می‌شویم.در خط ۱۷ مقدار متغیر total را برابر صفر قرار می‌دهیم. بعداً مشاهده می‌کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که شماره دانش آموز را وارد کنید (student + 1). عدد ۱ را به student اضافه کرده‌ایم تا به جای نمایش Student 0، با Student 1 شروع شود، تا طبیعی تر به نظر برسد. سپس به دومین حلقه for در خط ۲۱ می‌رسیم. در این حلقه یک متغیر شمارنده به نام grade تعریف می‌کنیم که طول دومین بعد آرایه را با استفاده از studentGrades[student].length به دست می‌آورد. این طول تعداد نمراتی را که برنامه از سؤال می‌کند را نشان می‌دهد. برنامه چهار نمره مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود.

وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خط ۲۸ معدل دانش آموز نشان داده می‌شود. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید. از studentGrades[student].length هم برای به دست آوردن تعداد نمرات استفاده می‌شود.

آرایه دندانه دار

آرایه دندانه دار یا jagged array آرایه ای چند بعدی است که دارای سطرهای با طول متغیر می‌باشد. نمونه ساده ای از آرایه‌های چند بعدی، آرایه‌های مستطیلی است که تعداد ستون‌ها و سطرهای آن‌ها برابر است. اما آرایه‌های دندانه دار دارای سطرهای (آرایه‌های) با طول متفاوت می‌باشند. بنابر این آرایه‌های دندانه دار را می‌توان آرایه ای از آرایه‌ها فرض کرد. دستور نوشتن این نوع آرایه‌ها 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 می‌نویسیم. مقداردهی به این آرایه‌ها 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];
```

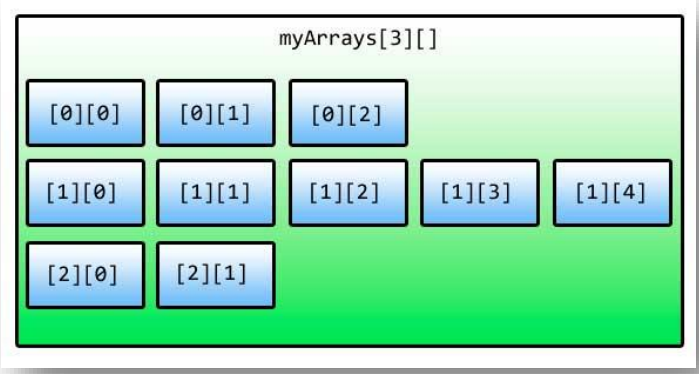
در کد بالا سه آرایه تعریف شده است که اندیس آن‌ها از صفر شروع می‌شود. اعداد ۳ و ۵ و ۲ هم به تعداد عناصری که هر کدام از آن‌ها در خود می‌توانند جای دهند اشاره دارند. برای مقداردهی هر آرایه هم باید ابتدا اندیس آرایه و سپس اندیس عناصر آن را بنویسیم. مثلاً مقداردهی اولین عنصر اولین آرایه مثال بالا به صورت زیر عمل می‌کنیم:

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

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

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

شکل زیر هم اندیس عناصر آرایه ای که در بالا تعریف کرده‌ایم را نشان می‌دهد:



با توجه به توضیحاتی که داده شد می‌توان 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 آرایه‌های دندانه دار به صورت زیر است که در آن می‌توان 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]);
```

نمی‌توان 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 می‌شویم چون عناصر این نوع آرایه‌ها، 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 می‌توان 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 آرایه‌های یک بعدی هستند) و در دومین حلقه از lengthبرای به دست آوردن عناصر سطر جاری استفاده می‌شود.

متد

متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه ای از کدها هستند که در هر جای برنامه می‌توان از آن‌ها استفاده کرد.متدها دارای آرگومان‌هایی هستند که وظیفه متد را مشخص می‌کنند. متد در داخل کلاس تعریف می‌شود. نمی‌توان یک متد را در داخل متد دیگر تعریف کرد. وقتی که شما در برنامه یک متد را صدا می‌زنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می‌کند. در جاوا متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند با ید از کجا شروع شوند، این متد main() نام دارد. پارامترها همان چیزهایی هستند که متد منتظر دریافت آن‌ها است. آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند. گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین 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 ۸-۵ یک متد تعریف کرده‌ایم. مکان تعریف آن در داخل کلاس مهم نیست. به عنوان مثال می‌توانید آن را زیر متد main() تعریف کنید. می‌توان این متد را در داخل متد دیگر صدا زد (فراخوانی کرد). متد دیگر ما در اینجا متد main() است که می‌توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده‌ایم (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 می‌توان از متد استفاده کرد بدون اینکه از کلاس نمونه ای ساخته شود. متد ()main همواره باید به صورت static تعریف شود چون برنامه فوراً و بدون نمونه سازی از کلاس از آن استفاده می‌کند. وقتی به مبحث برنامه نویسی شیء گرا رسیدید به طور دقیق کلمه static مورد بحث قرار می‌گیرد. برنامه class (مثال بالا) زمانی اجرا می‌شود که برنامه دو متدی را که تعریف کرده‌ایم را اجرا کند و متد ()main به صورت static تعریف شود. در باره این کلمه کلیدی در درس‌های آینده مطالب بیشتری می‌آموزیم.

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 می‌شود) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص متدها استفاده کنید.

بهتر است در نامگذاری متدها از کلماتی استفاده شود که کاران متد را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor . همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد می‌توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید، مانند IsLeapyear یا IsTeenager، ولی از گذاشتن علامت سؤال در آخر اسم متد خودداری کنید. دو پرانتزی که بعد از نام می‌آید نشان دهنده آن است که نام متعلق به یک متد است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درس‌های آینده در مورد متدها بیشتر توضیح می‌دهیم.

بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه متد را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم. در داخل متد ()main متدی را که در خط ۱۲ ایجاد کرده‌ایم را صدا می‌زنیم. 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 می‌دهیم. با صدا زدن یک متد کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای متد ()PrintMessage برنامه از متد ()main به محل تعریف متد ()PrintMessage می‌رود. مثلاً وقتی ما متد ()PrintMessage را در خط ۱۲ صدا می‌زنیم برنامه از خط ۱۲ به خط ۷، یعنی جایی که متد تعریف شده می‌رود. اکنون ما یک متد در برنامه class داریم و همه متدهای این برنامه می‌توانند آن را صدا بزنند.

مقدار برگشتی از یک متد

متدها می‌توانند مقدار برگشتی از هر نوع داده ای داشته باشند. این مقادیر می‌توانند 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 می‌زنید و از او می‌خواهید 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 می‌کند (int، bool،...). در داخل بدنه متد کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می‌نویسیم. 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 می‌دهد.

```
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 ۷ مثال فوق مشاهده می‌کنید هنگام تعریف متد از کلمه int به جای void استفاده کرده‌ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۹ و ۱۰ دو متغیر تعریف و مقدار دهی شده‌اند.

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 شده‌اند قابل استفاده هستند. در خط ۱۲ جمع دو متغیر در متغیر sum قرار می‌گیرد. در خط ۱۴ مقدار برگشتی sum توسط دستور return فراخوانی می‌شود. در داخل متد main() یک متغیر به نام result در خط ۱۹ تعریف می‌کنیم و متد CalculateSum() را فراخوانی می‌کنیم.

متد CalculateSum() مقدار ۱۵ را بر می‌گرداند که در داخل متغیر result ذخیره می‌شود. در خط ۲۱ مقدار ذخیره شده در متغیر result چاپ می‌شود. 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 ۱۵ است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این متد در صورتی کارآمد است که پارامترهایی به آن اضافه شود که در درس‌های آینده توضیح خواهیم داد. هنگامی که می‌خواهیم 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 ۲۵-۸ یک متد با نام GetNumber() تعریف شده است که از کاربر یک عدد بزرگ‌تر از ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد متد مقدار صفر را بر می‌گرداند. و اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می‌شویم.

چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت else می‌رود تا مقدار صفر را برگرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می‌شود و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد پیغام خطا می‌دهد.