

جاوا به زبان فوق العاده ساده

ترجمه و تاليف: هادی خداپناه

## بسم تعالی

زبان برنامه نویسی جاوا در دنیا روز به روز با اقبال بیشتری مواجه می شود و عده‌ی زیادی به برنامه نویسی با این زبان روی می آورد. دلیل این هم روشن است: امن بودن و در عین حال قدرت زبان جاوا باعث گردیده، تعداد برنامه نویس هایی که با جاوا کار می کنند از تعداد برنامه نویس هایی که با سایر زبان های برنامه نویسی کار می کنند به مراتب بیشتر باشد. در کنار این موضوع شاهد رواج گستردگی گوشی های هوشمند بین تمام مردم دنیا هستیم، که اکثرا از سیستم عامل اندروید استفاده میکنند و حال آنکه ما برای ایجاد و نوشتگان اپلیکیشن ها و برنامه های جانبی روی اندروید بایستی از جاوا استفاده کنیم. باید توجه داشت که بخش رابط کابری اندروید با زبان جاوا نوشته شده است و برنامه های اندرویدی میتوانند برای ارتباط با لایه های زیرین سیستم عامل اندروید از کتابخانه های جاوای اندروید بهره ببرند. همچنین برای کار با پایگاه داده هایی همچون اوراکل، بدون دانستن زبان جاوا امری غیر ممکن است. زیرا که برای کاربا پایگاه داده ای اوراکل می بایستی حتما، با جاوا آشنایی داشته باشیم. ما با استفاده از جاوا قادر خواهیم بود تا برنامه های کاربردی، برای لینوکس، اندروید و ویندوز بنویسیم.

هدف بندۀ از نگارش این کتاب ارتقای سطح علمی جاوا در ایران بوده و معیار اصلی من هنگام تالیف کتاب مورد توجه قراردادن خوانندگان مبتدی و کاربرانی که اصلا برنامه نویسی نکرده اند بوده است. در عین حال از کابران حرفه ای نیز غافل نشده ام، و این کتاب شامل نکات ارزنده ای برای متخصصین حرفه ای کامپیوتر است.

رویه آموزشی کتاب هم از مطالب مقدماتی شروع شده و تا سطح پیشرفته پیش میرود. سعی من در تالیف این کتاب به صورت خود آموز بوده به صورتی که شما دیگر نیازی به کلاس آموزشی یا استاد ندارید.

و اما وجه تسمیه این کتاب با سایر کتاب های فارسی جاوا موجود در ایران، در این است که بندۀ تمام مطالب فصل های کتاب را از آخرين ويرايش های کتاب های معتبر انگلیسي اخذ کرده و به جرات می توانم بگویم حتی از یک منبع فارسی (که معمولا پر از اشتباه اند) استفاده نکرده ام.

بیش فرض من در هنگام نوشتگان این کتاب در سطح مقدماتی و پایین قرار دارند و قدم به قدم توسط مطالعه ای این کتاب به سطح خوبی در جاوا خواهند رسید.

هادی خداپناه

۱۳۹۵/۶/۱۳

## فهرست مطالب

### فصل ۱

۶.....	نصب JAVA
۱۶.....	نصب eclipse

### فصل ۲

۲۸.....	متغیر
۲۸.....	انواع داده ها در جاوا
۳۰.....	تخصیص مقدار به متغیر های دارای مقدار اولیه
۳۰.....	عبارات تخصیص (Assignment Statements)
۳۴.....	نوع داده ی char
۳۳.....	نوع boolean
۳۵.....	نوع داده های اعشاری در جاوا
۳۶.....	نوع داده ای صحیح در جاوا
۳۷.....	نوع مرجع (Reference Types)

۴۱.....	ایجاد مقادیر جدید با اعمال عملگرها (Creating New Values by Applying Operators)
۴۳.....	عملگر های افزایش (increment) و کاهش (decrement)
۴۵.....	عملگر های تخصیص (Assignment Operators)

### فصل ۳

۴۹.....	Making Decisions (Java if Statements)
۵۱.....	کنترل کلید های فشار داده شده از کیبورد
۵۳.....	ایجاد کردن مقادیر تصادفی
۵۴.....	دستورات انتخابی
۵۵.....	علامت == (The double equal sign)
۵۶.....	بدون if کاربرد else
۵۷.....	ایجاد شرط با عملگر های مقایسه ای و منطقی
۵۸.....	مقایسه ی اشیاء (Comparing objects)

۶۰ ..... عملگر های منطقی در جاوا

۶۴ ..... ساخت if های تودرتو

۶۶ ..... در جاوا Switch

۶۹ ..... جدید و بهبود یافته Switch

## فصل ۴

۷۳ ..... تکرار دستور العمل ها بارها و بارها ( دستور while در جاوا )

۷۵ ..... تکرار تا تعدادی معین(حلقه i for در جاوا )

۷۸ ..... اجرا تا زمانی که آنچه میخواهید را به دست آورید ( حلقه i do در جاوا )

۸۰ ..... ذخیره i یک کارکتر تنها

۸۱ ..... کارکردن با فایل ها در جاوا

۸۲ ..... محدوده i متغیر ها

## فصل ۵

۸۵ ..... تعریف کلاس

۸۷ ..... تعریف متغیر ها و تعریف اشیاء

۸۹ ..... کلاس های عمومی ( Public classes )

۹۱ ..... تعریف متدها داخل کلاس

۹۰ ..... یک Account که خودش را نمایش میدهد

۹۲ ..... فرستادن و گرفتن مقدار به Method ها :

۹۸ ..... خطاب

۱۰۲ ..... برنامه نویسی خوب (Good programming)

۱۰۳ ..... Accessor Methods فراخوانی

## فصل ۶

۱۰۹.....	ذخیره کردن داده ها در فایل
۱۱۰.....	کردن کد Past و Copy
۱۱۲.....	خواندن یک خط در هر دفعه
۱۱۵.....	ایجاد یک زیر کلاس
۱۱۸.....	ایجاد زیر کلاس ها شکل گیری عادت هاست
۱۱۹.....	استفاده از زیر کلاس ها
۱۲۰.....	تطابق انواع (types match)
۱۲۱.....	تغییر دادن متدهای قبلی
۱۲۴.....	استفاده از متدها از درون زیر کلاس ها و کلاس ها

## فصل ۷

۱۲۷.....	مقیاس دما (temperature scale) چیست
۱۲۷.....	(انواع enum) جاوا
۱۲۸.....	Temperature چیست؟
۱۲۹.....	با سازنده (statements) چه کاری میتوان انجام داد
۱۳۱.....	دو راه برای انجام یک کار
۱۳۲.....	ساخت temperatures بهتر
۱۳۳.....	Using all this stuff
۱۳۴.....	یک سازنده ی پیش فرض
۱۳۵.....	سازنده ای که کارهای بیشتری انجام میدهد
۱۳۸.....	منابع

# فصل 1

وَالْمُؤْمِنُونَ

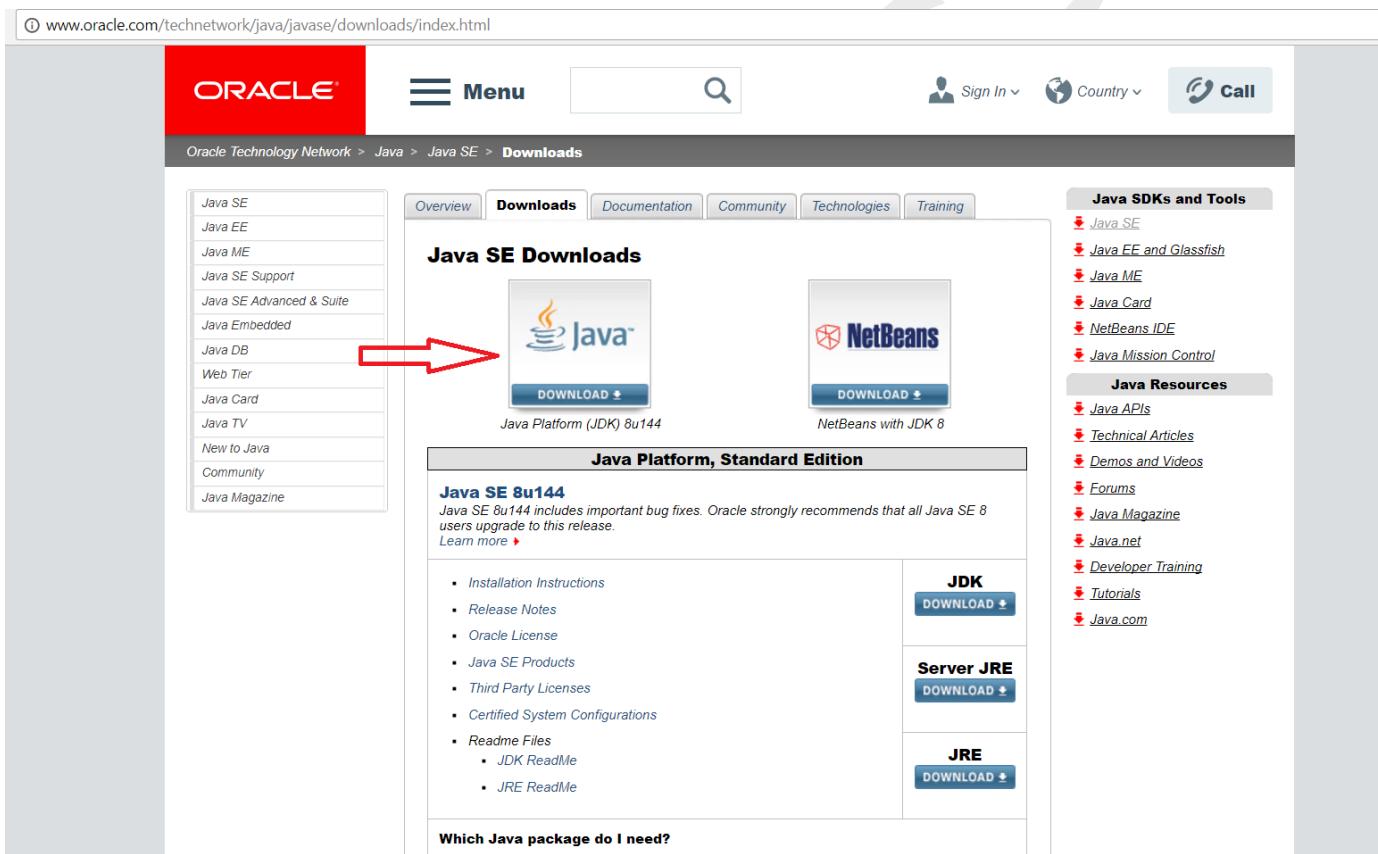
# نصب JAVA

در ابتدا فایل **JRE: (Java Runtime)** و نیز فایل **JDK (Java SE Development Kit)** را از سایت رسمی آن به آدرس **Environment**)

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

دانلود کنید و هر دو را نصب کنید.

توجه کنید که در زمانی که این مطلب رو می نویسم، گویا ایران تحریم شده است و باید از **VPN** (همون فیلتر شکن خودمون) استفاده کنیم از سایت بالا این دو فایل رو دانلود کنید. با اینکه از سایت های ایرانی آنها را تهییه نمایید.



بعد کلیک روی دکمه **ی مشخص شده در تصویر زیر** رو به خواهد گشت.

سپس گزینه **Accept License Agreement** را تیک دار کنید. و از داخل کنترل قرمز رنگ **JDK** متناسب با سیستم عامل خود را انتخاب کنید.

The screenshot shows the Oracle Java SE Development Kit 8u144 Downloads page. At the top, there's a navigation bar with links for Oracle Technology Network, Java, Java SE, Downloads, Overview, Downloads (which is highlighted), Documentation, Community, Technologies, and Training. On the right side, there are links for Sign In, Country, and Call. Below the navigation, a sidebar on the left lists categories like Java SE, Java EE, Java ME, etc. The main content area features a heading 'Java SE Development Kit 8 Downloads' with a note about accepting the Oracle Binary Code License Agreement. It also includes a 'See also:' section with links to developer newsletters, day workshops, and magazine. A table provides download links for various platforms. A large red arrow is drawn on the left side of the page, pointing towards the 'Downloads' section.

**Java SE Development Kit 8 Downloads**

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u144 checksum

**Java SE Development Kit 8u144**

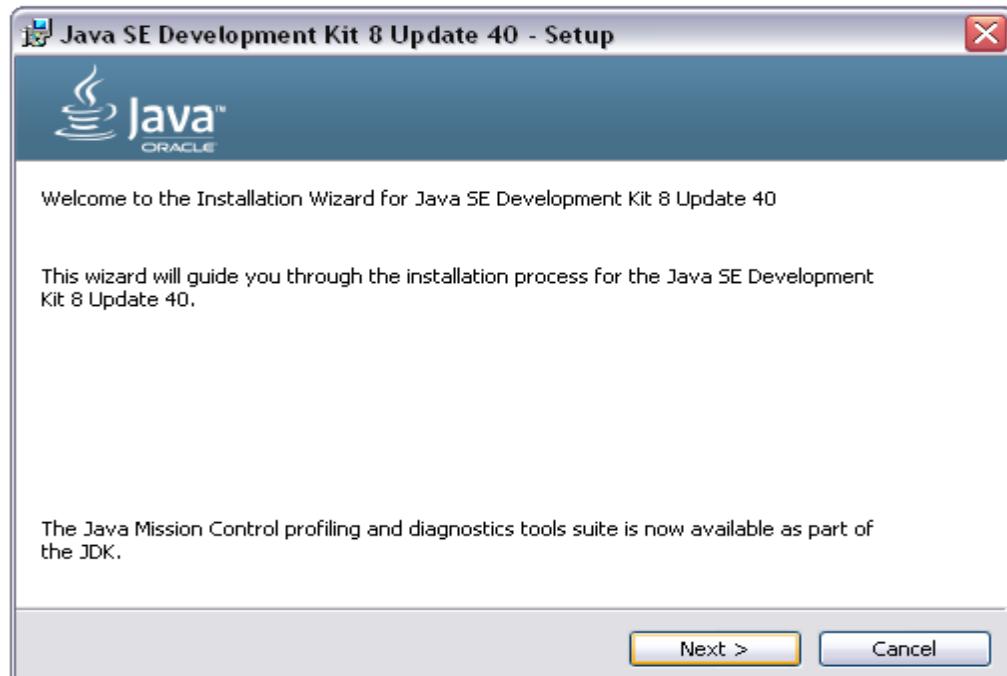
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.89 MB	<a href="#">jdk-8u144-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	74.83 MB	<a href="#">jdk-8u144-linux-arm64-vfp-hflt.tar.gz</a>
Linux x66	164.65 MB	<a href="#">jdk-8u144-linux-i586.rpm</a>
Linux x66	179.44 MB	<a href="#">jdk-8u144-linux-i586.tar.gz</a>
Linux x64	162.1 MB	<a href="#">jdk-8u144-linux-x64.rpm</a>
Linux x64	176.92 MB	<a href="#">jdk-8u144-linux-x64.tar.gz</a>
Mac OS X	226.6 MB	<a href="#">jdk-8u144-macosx-x64.dmg</a>
Solaris SPARC 64-bit	139.87 MB	<a href="#">jdk-8u144-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	99.18 MB	<a href="#">jdk-8u144-solaris-sparcv9.tar.gz</a>
Solaris x64	140.51 MB	<a href="#">jdk-8u144-solaris-x64.tar.Z</a>
Solaris x64	96.99 MB	<a href="#">jdk-8u144-solaris-x64.tar.gz</a>
Windows x86	190.94 MB	<a href="#">jdk-8u144-windows-i586.exe</a>
Windows x64	197.76 MB	<a href="#">jdk-8u144-windows-x64.exe</a>

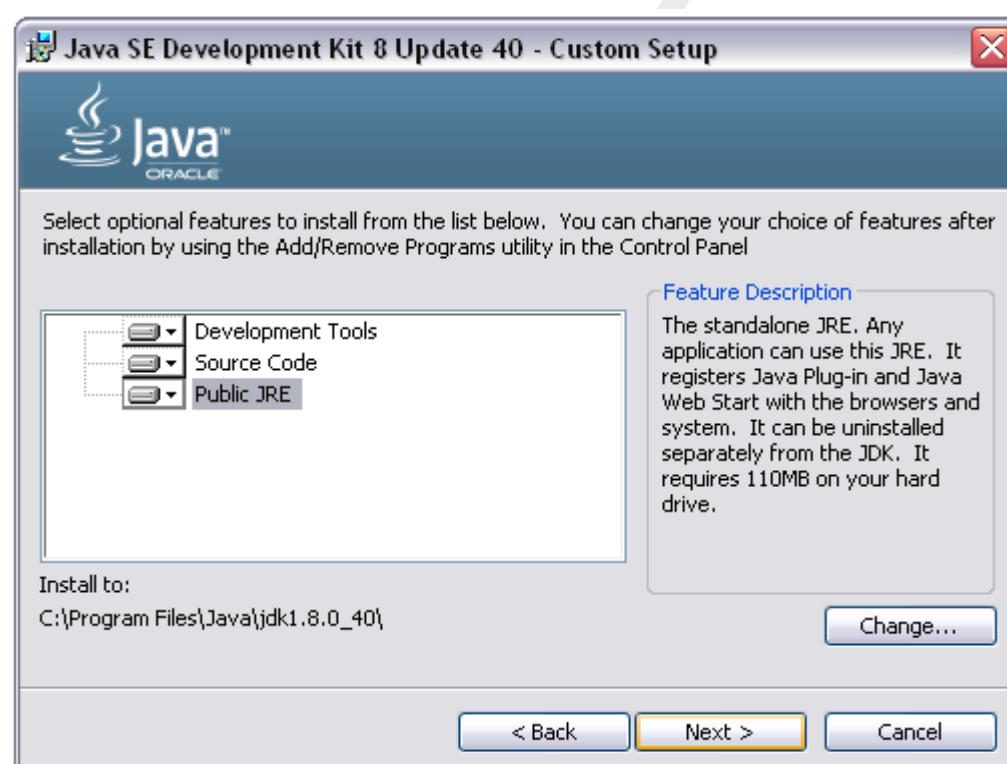
**Java SE Development Kit 8u144 Demos and**

بعد از دانلود فایل JDK نوبت به نصب آن میرسد

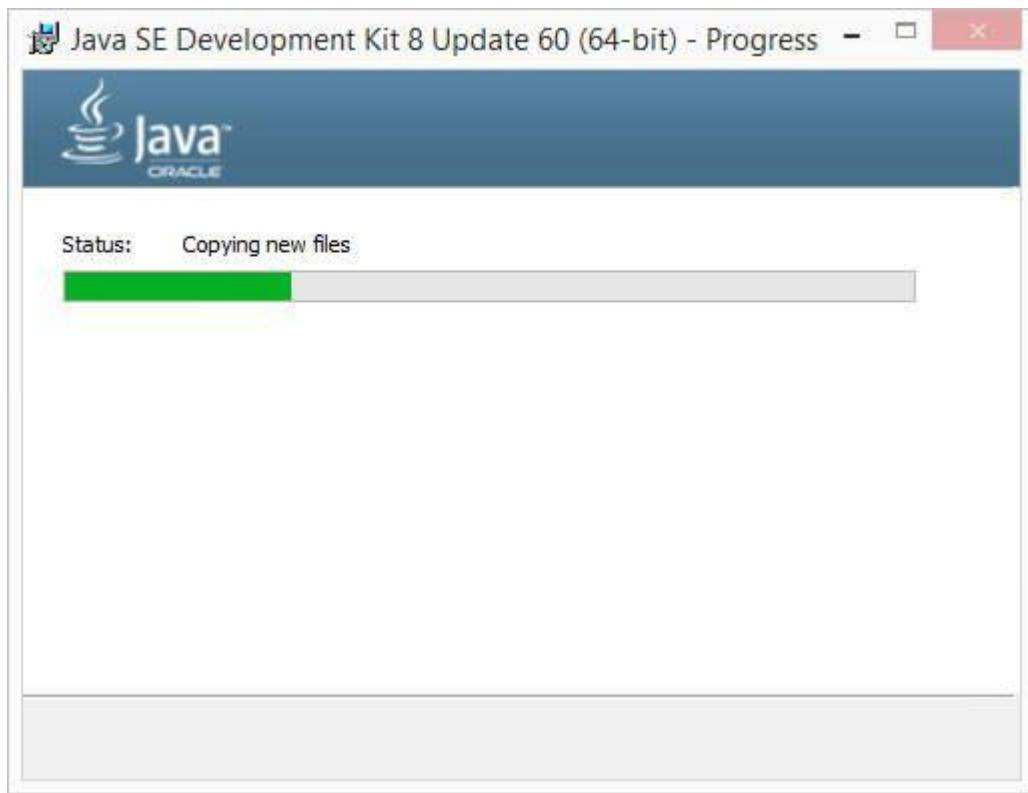


فایل اجرایی جاوا (jdk) را اجرا کنید.

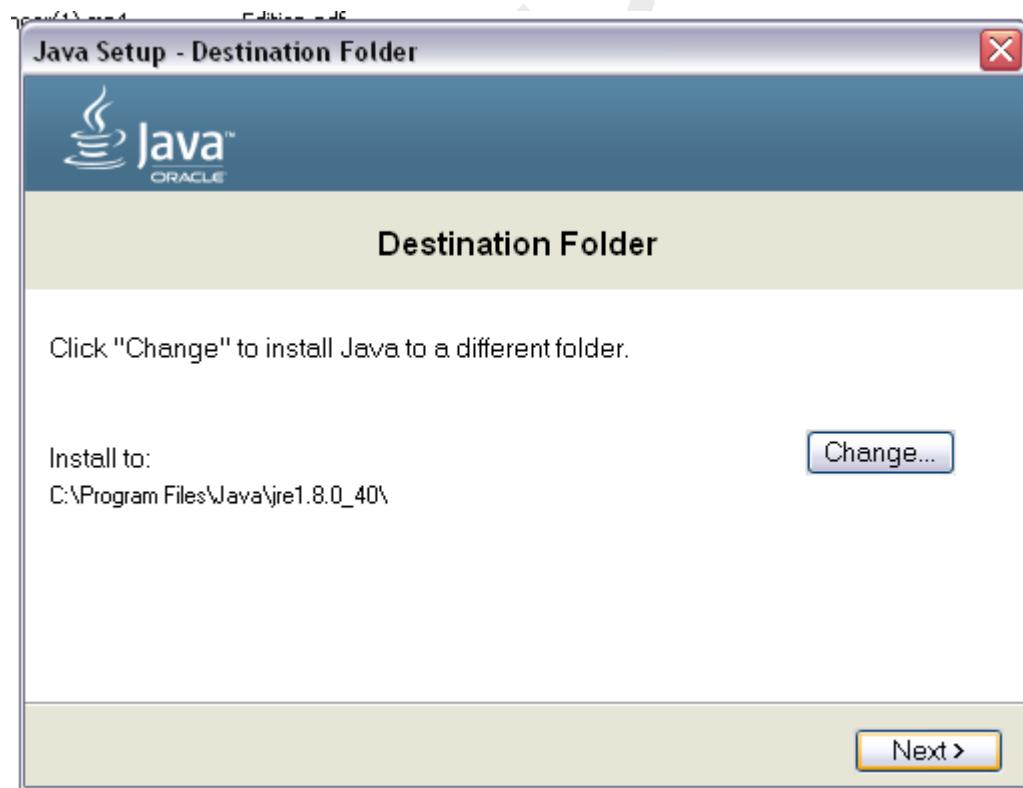
در کادر اول زیر روی کلیک کنید.



در کادر دوم نیز بعد از مشخص کردن مسیر نصب جاوا روی کلیک کنید Next



در اینجا بعد از مشخص کردن پوشه‌ی مقصد برای نصب جاوا روی Next کلیک کنید.

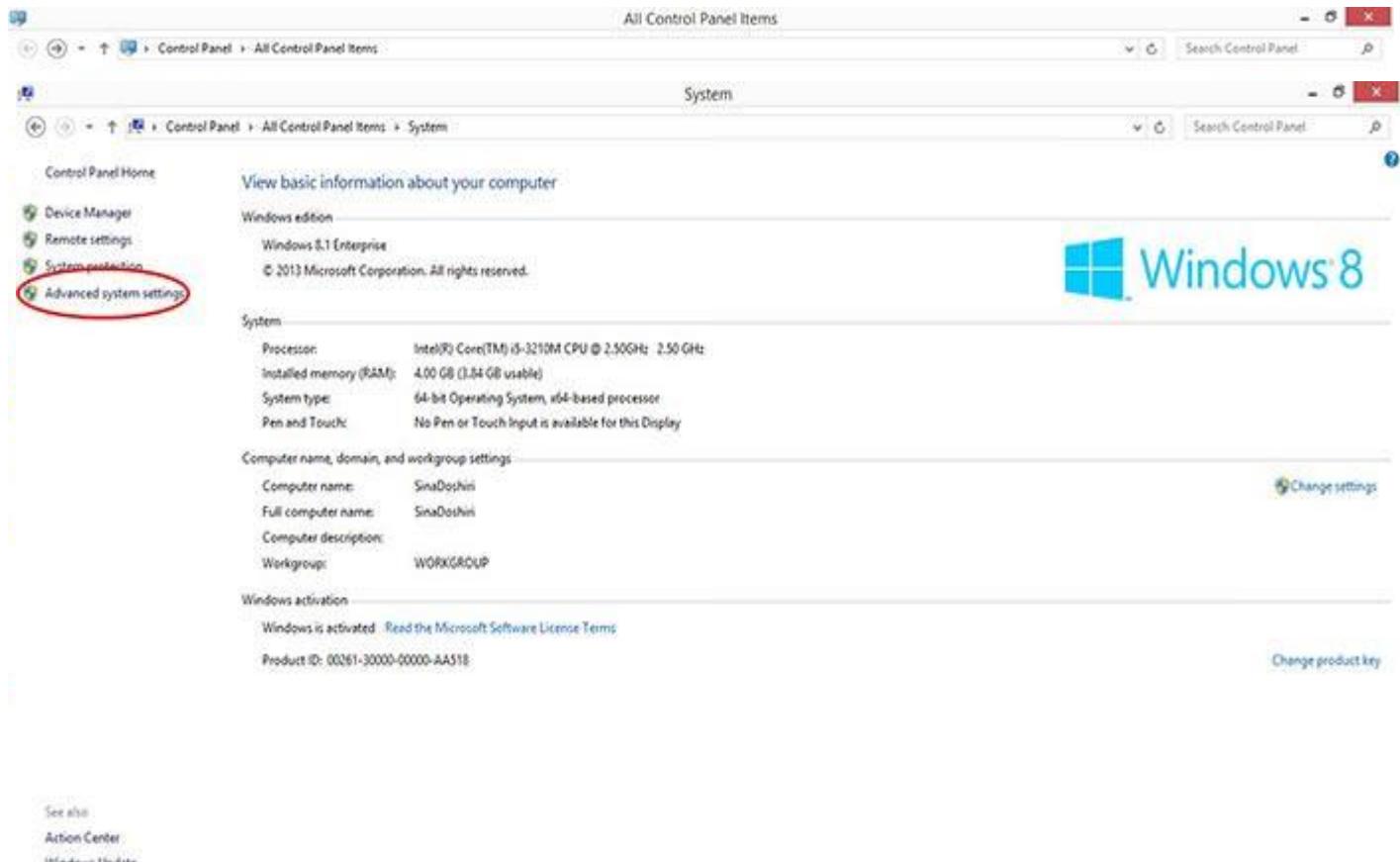




در کادر پایین اظهار میکند که جاوا با موفقیت نصب شد... روی Close کلیک کنید...

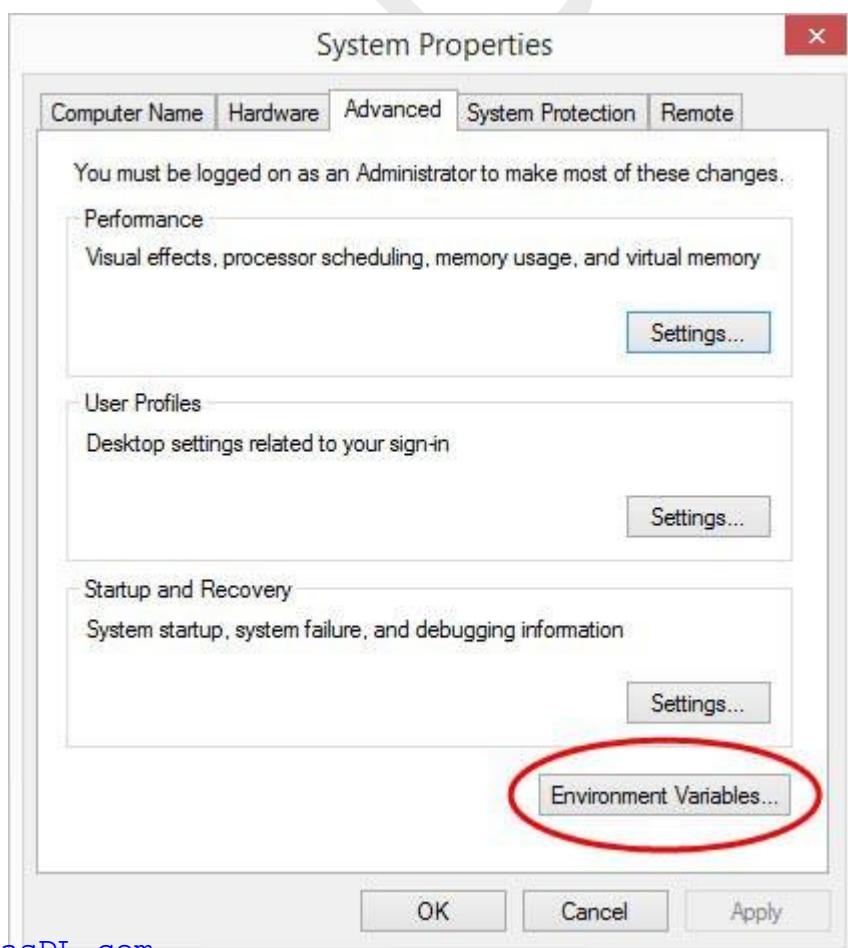


حال که ما جاوا را نصب کردیم، بایستی جاوا رو به سیستم عامل خود معرفی کنیم. برای اینکار وارد کنترل پنل شوید و روی System کلیک کنید

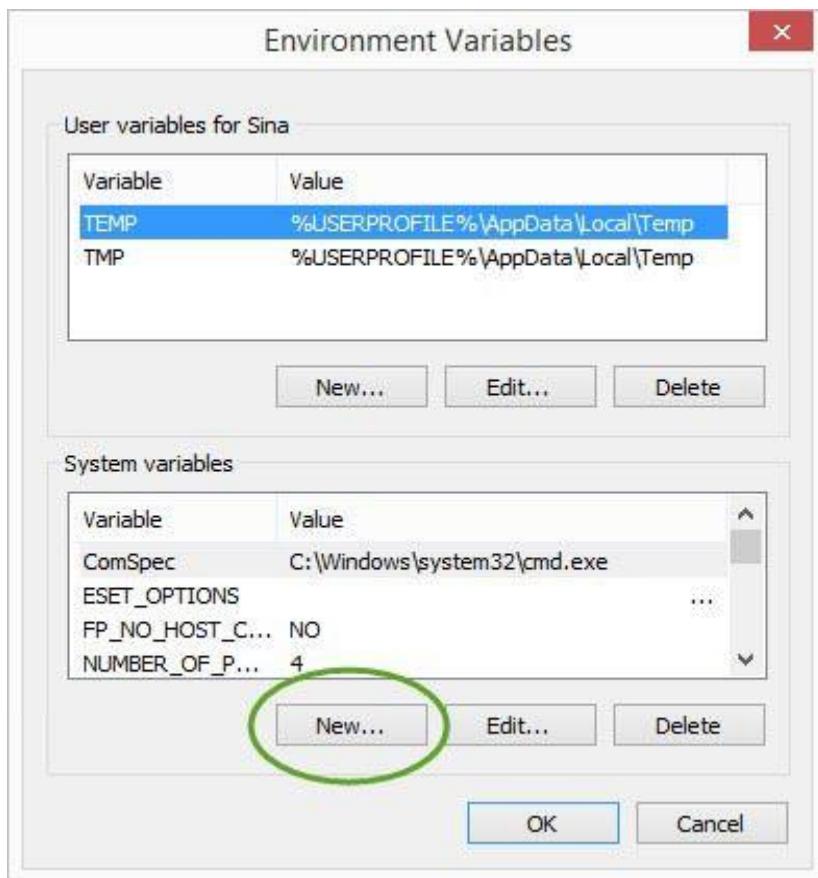


آنگه پنجره ای ظاهر خواهد گشت که روی Advanced system settings کلیک کنید

حال پنجره ای ظاهر میشود که باید روی Environment Variables کلیک کنید



دوباره پنجره‌ی جدیدی ظاهر خواهد گشت که در قسمت **System variables** روی دکمه‌ی **New** کلیک کنید



با این کادر یک متغیر جدید ایجاد می‌شود  
که بایستی مقدار دهی کنید

New System Variable

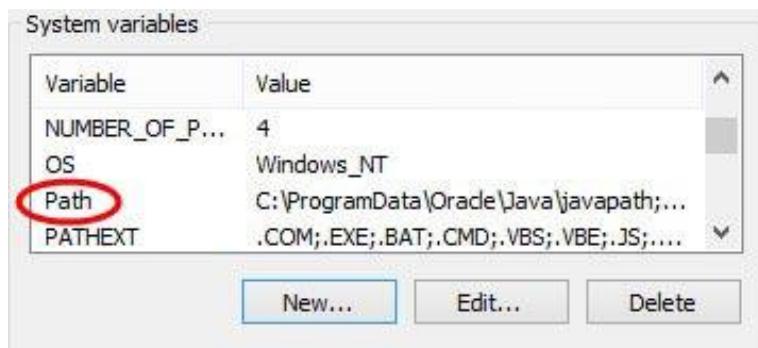
Variable name: JAVA\_HOME

Variable value: C:\Program Files\Java\jdk1.8.0\_60\

OK

Cancel

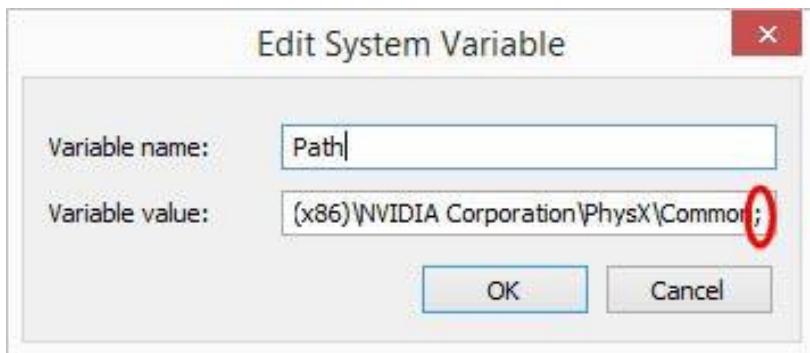
پنجره‌ی ظاهر شده را مانند رو به رو پر کنید (با این کار متغیر را مقدار دهی می‌کنید). در قسمت **Value** مسیری که قبل مشخص کرده اید را بنویسید



بعد از این مرحله باید مسیر دایرکتوری bin را در متغیر سیستمی PATH قرار دهیم. برای این کار

دوباره در قسمت system variables به دنبال متغیر path بگردید.

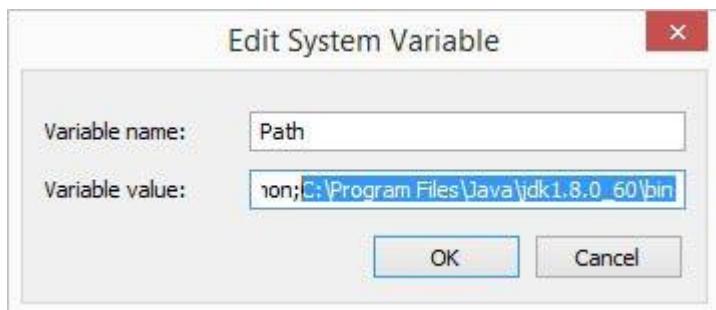




را انتخاب کرده و روی **Edit** کلیک کنید.

در قسمت **value** به آخر خط رفته و یک سمی کالن قرار دهید.

بعد از قرار دادن سمی کالن، نشانگر ماوس خود را بعد از سمی کالن قرار دهید و بعد مسیر دایرکتوری **bin** را در این قسمت **paste** کنید.



در نهایت رو **OK** کلیک کنید. و کامپیوتر خود را رستارت کنید.

## نصب eclipse

در آخر نوبت به نصب برنامه‌ی eclipse می‌رسه که می‌توانید اون رو از سایت زیر دانلود و نصب کنید .آموزش دانلود از سایت را در زیر قرار دادم.

<https://eclipse.org/downloads/>

دانلود  
از سایت



GETTING STARTED

MEMBERS

PROJECTS

MORE ▾

[Create account](#) [Log in](#)

Google Custom Search

[DOWNLOAD](#)

## Eclipse Is...

An amazing open source community of **Tools**, **Projects** and **Collaborative Working Groups**. Discover what we have to offer and join us.

[DISCOVER ▾](#)

IDE &amp; Tools



Community of Projects



Collaborative Working Groups



GETTING STARTED

MEMBERS

PROJECTS

MORE ▾

[Create account](#) [Log in](#)

Google Custom Search

[SEARCH](#)

HOME / DOWNLOADS / ECLIPSE DOWNLOADS

[Packages](#) [Developer Builds](#)

Eclipse Mars.1 (4.5.1) Release for Windows



### Try the Eclipse Installer NEW

The easiest way to install and update your Eclipse Development Environment.

[FIND OUT MORE ▾](#)3,074,790  
DownloadsMac OS X  
▲ 64 bitWindows  
▲ 32 bit | 64 bitLinux  
▲ 32 bit | 64 bit

...or download an Eclipse Package

#### Eclipse IDE for Java EE Developers



275 MB | 2,249,274 DOWNLOADS

Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...

Windows

32 bit | 64 bit



#### Run your Apps on the Cloud

Want Eclipse in the cloud? IBM Bluemix makes it easy. Sign up to begin building today!

Promoted  
Downloads

#### Eclipse IDE for Java Developers



166 MB | 1,117,108 DOWNLOADS

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven integration and WindowBuilder...

Windows

32 bit | 64 bit



#### Eclipse IDE for C/C++ Developers



176 MB | 429,919 DOWNLOADS

An IDE for C/C++ developers with Mylyn integration.

Windows

32 bit | 64 bit



## Squish

### RELATED LINKS

- Compare & Combine Packages
- New and Noteworthy
- Install Guide
- Documentation
- Updating Eclipse
- Forums

### MORE DOWNLOADS

- Other builds
- Eclipse Mars (4.5)
- Eclipse Luna (4.4)
- Eclipse Kepler (4.3)
- Eclipse Juno (4.2)
- Older Versions

### Hint

You will need a **Java runtime environment (JRE)** to use Eclipse (Java SE 7 or greater is recommended). All



GETTING STARTED MEMBERS PROJECTS MORE ▾

Google Custom Search



DOWNLOAD

HOME / DOWNLOADS / PACKAGES / ECLIPSE IDE FOR JAVA DEVELOPERS

## RELEASES

Neon Packages  
Mars Packages  
Luna Packages  
Kepler Packages  
Juno Packages  
Indigo Packages  
Helios Packages  
Galileo Packages  
Ganymede Packages  
Europa Packages  
All Releases



## Eclipse IDE for Java Developers

### Package Description

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven integration and WindowBuilder

#### This package includes:

- Eclipse Git Team Provider
- Eclipse Java Development Tools
- Maven Integration for Eclipse
- Mylyn Task List
- Code Recommenders Tools for Java Developers
- WindowBuilder Core
- Eclipse XML Editors and Tools

### Download Links

Windows 32-bit  
Windows 64-bit  
Mac OS X (Cocoa) 64-bit  
Linux 32-bit  
Linux 64-bit

Downloaded 1,117,108 Times

▶ Checksums...

Bugzilla

Create account Log in



GETTING STARTED

MEMBERS

PROJECTS

MORE ▾

Google Custom Search



HOME / DOWNLOADS / ECLIPSE DOWNLOADS - SELECT A MIRROR

## Eclipse downloads - Select a mirror

All downloads are provided under the terms and conditions of the [Eclipse Foundation Software User Agreement](#) unless otherwise specified.

Download from: Canada - Rafal Rzeczkowski (<http://>)

OR >

Get It Faster from our Members

File: [eclipse-java-mars-1-win32-x86\\_64.zip](http://eclipse-java-mars-1-win32-x86_64.zip)

Checksums: MD5 SHA1

SHA-S12

DOWNLOAD



IBM

Blazingly fast downloads hosted by IBM Bluemix.

[DOWNLOAD](#)



Obeo

Download the latest Eclipse easily and



The Easiest Way to Get Your Own Modeling Tool

[DOWNLOAD NOW](#)

### OTHER OPTIONS FOR THIS FILE

- All mirrors (xml)
- Direct link to file (download starts immediately from best mirror)



GETTING STARTED

MEMBERS

PROJECTS

MORE ▾

Create account Log in

Google Custom Search



DOWNLOAD

HOME / DOWNLOADS / THANK YOU FOR DOWNLOADING ECLIPSE

## Thank you for downloading Eclipse

If the download doesn't start in a few seconds, please [click here](#) to start the download.

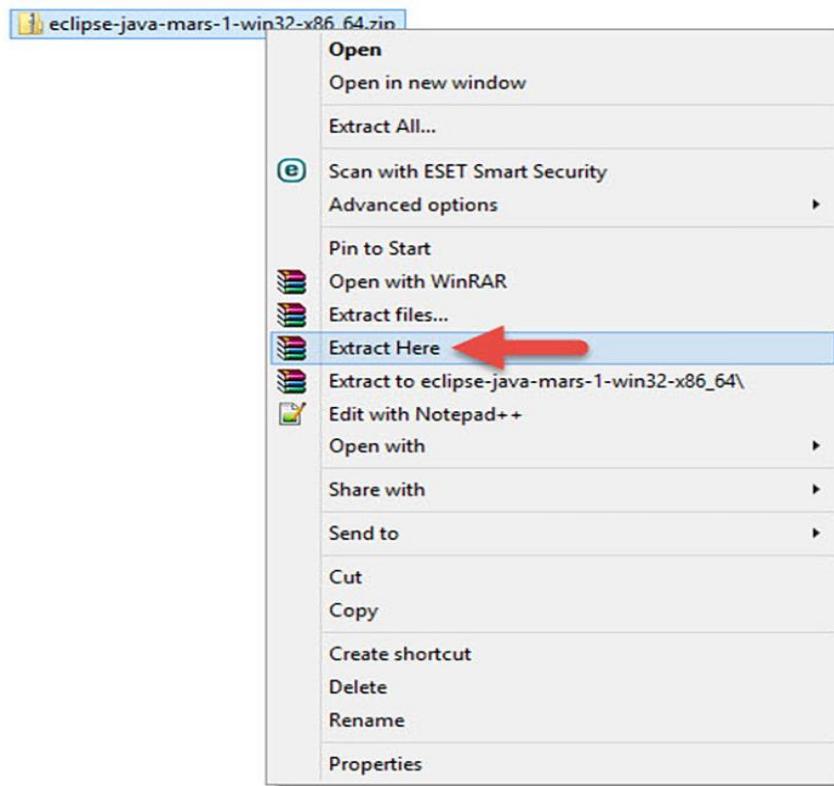
Stay Up to Date  
Subscribe to the Eclipse Newsletter



در تصویر شماره ی سه در بالا ، باید با توجه به سیستم عامل خود یک گزینه را انتخاب کنید.

بعد از دانلود برنامه نوبت به نصب و اجرای آن میرسد.

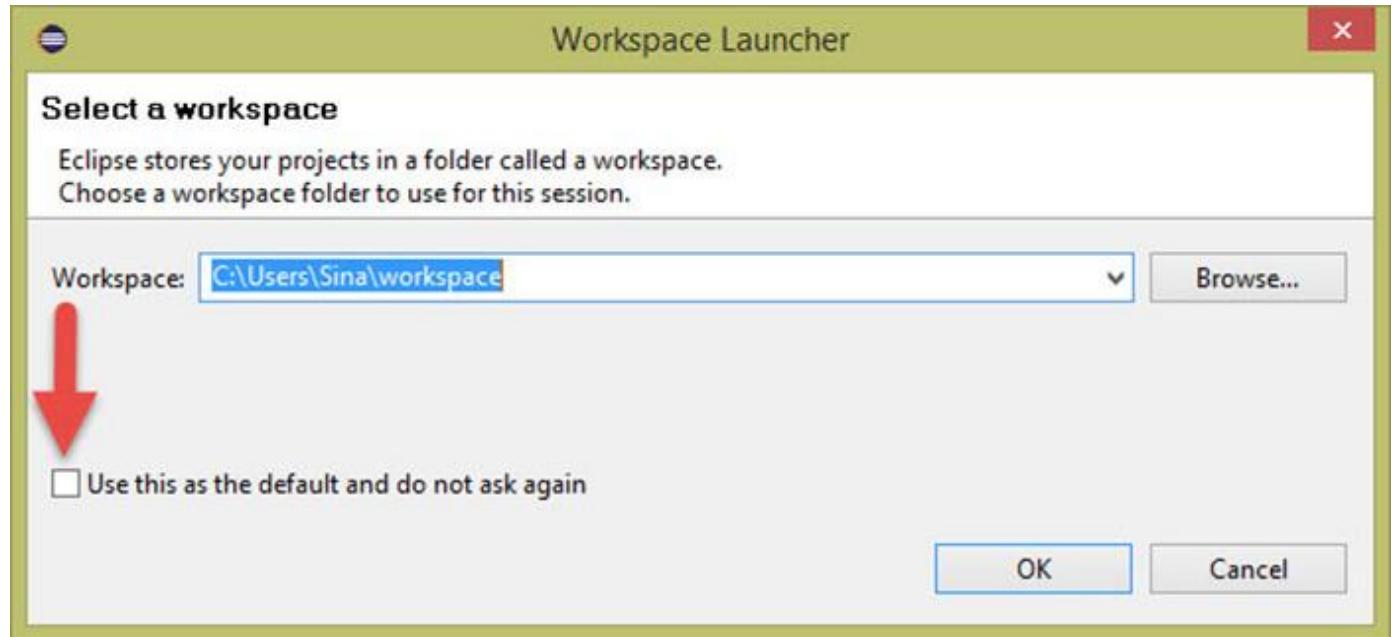
ما با استفاده از این برنامه ایکلیپس برنامه های نوشته شده ی جاوا را اجرا و تست خواهیم کرد



ابتدا فایل فشرده رو از حالت فشرده خارج کنید.

فایل eclipse رو اجرا کنید.

configuration	Date modified: 9/24/2015 6:36 AM
dropins	Date modified: 9/24/2015 6:36 AM
features	Date modified: 9/24/2015 6:36 AM
p2	Date modified: 9/24/2015 6:35 AM
plugins	Date modified: 9/24/2015 6:36 AM
readme	Date modified: 9/24/2015 6:36 AM
.eclipseproduct Type: ECLIPSEPRODUCT File	Date modified: 9/2/2015 10:05 AM Size: 60 bytes
artifacts.xml	Date modified: 9/24/2015 6:36 AM Size: 132 KB
eclipse.exe Type: Application	Date modified: 9/24/2015 6:37 AM Size: 312 KB
eclipse.ini Type: Configuration settings	Date modified: 9/24/2015 6:36 AM Size: 460 bytes
eclipsesec.exe Type: Application	Date modified: 9/24/2015 6:37 AM Size: 24.9 KB

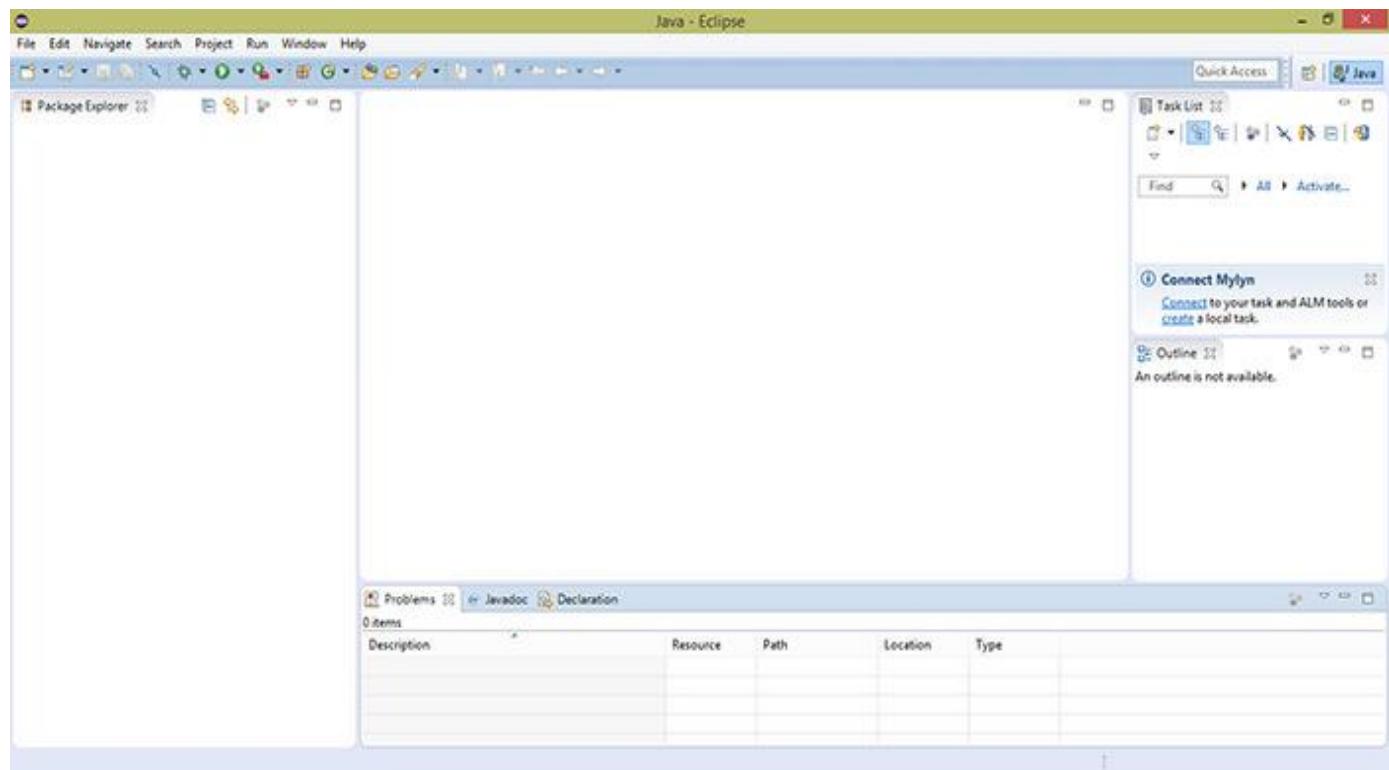


در کادر بالا مسیری که پروژه های جاوا ذخیره خواهد شد را مشخص کنید.

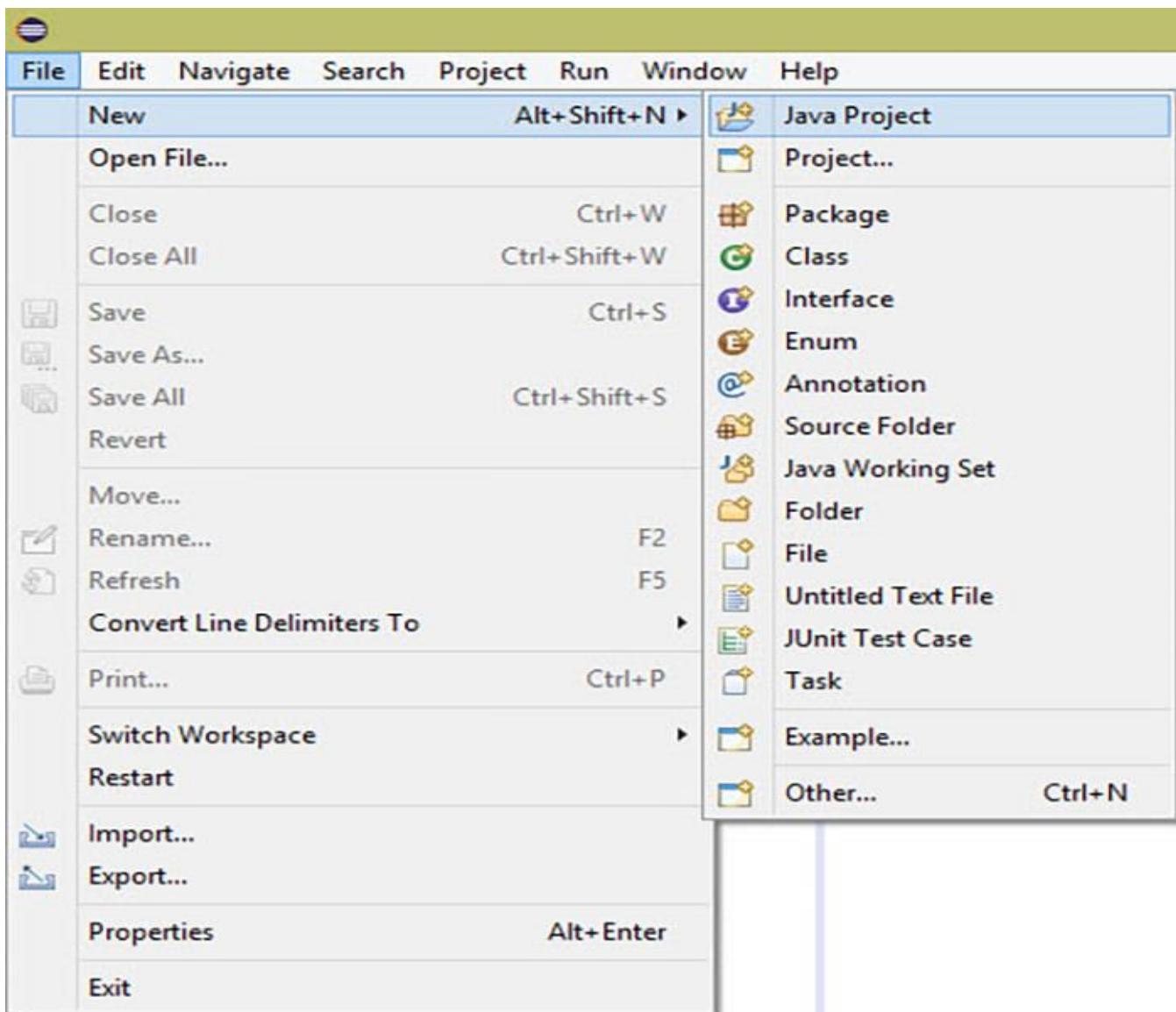
همچنین اگر گزینه‌ی پایینی را تیک بزنید آنگاه دیگر این کادر نمایش داده نخواهد شد و مسیری که برای ذخیره‌ی برنامه‌ها مشخص کرده اید به عنوان پیش فرض تعیین می‌گردد.



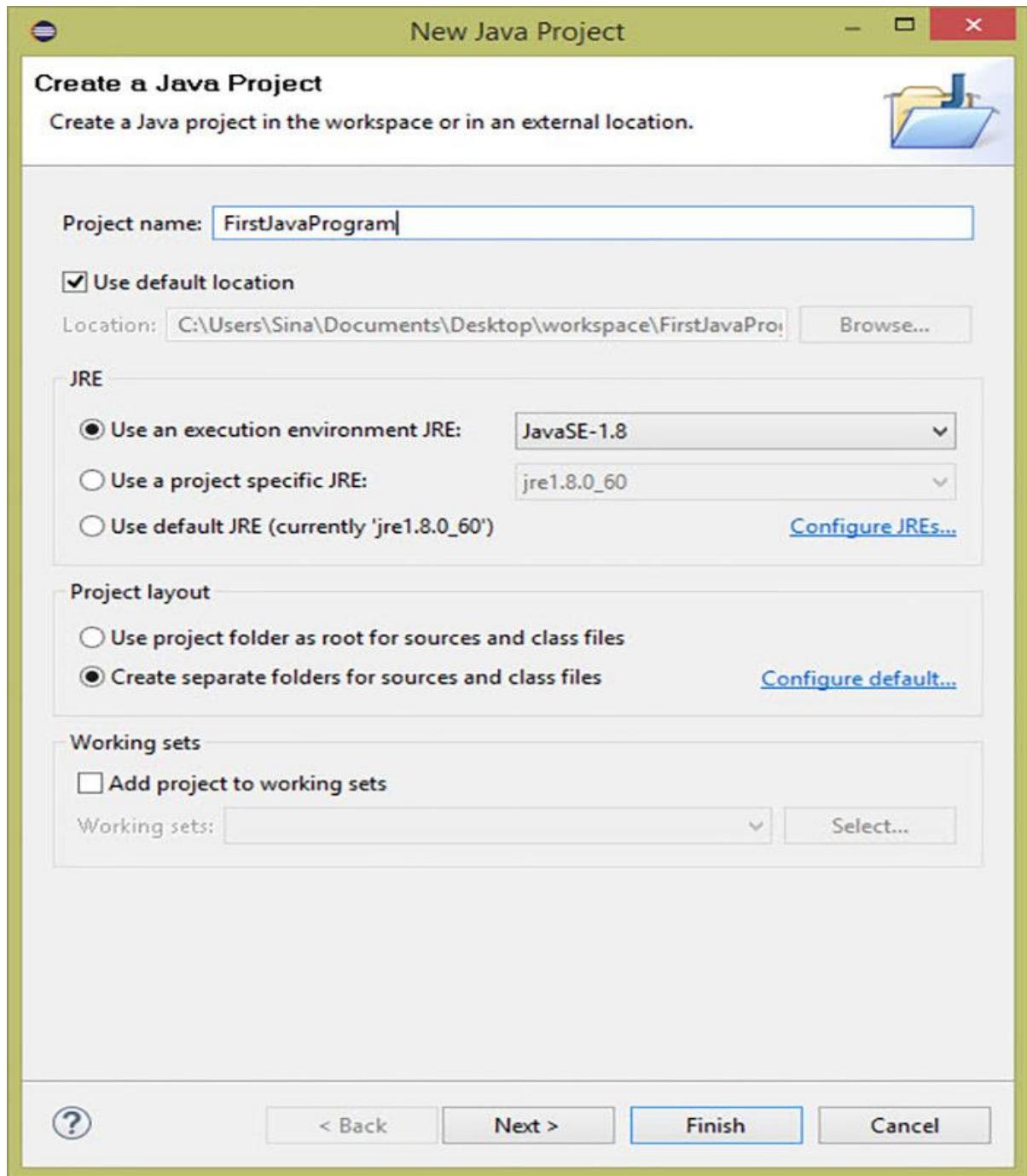
زمانی که فایل eclipse رو اجرا کردید صفحه خوشامد گویی مانند بالا ظاهر خواهد گشت. رو دکمه‌ی Workbench که با فلش سبز مشخص شده است کلیک کنید.



حال نوبت آن است که اولین پروژه جاوای خودمان را ایجاد کنیم. برای این من مانند تصویر زیر عمل کنید.

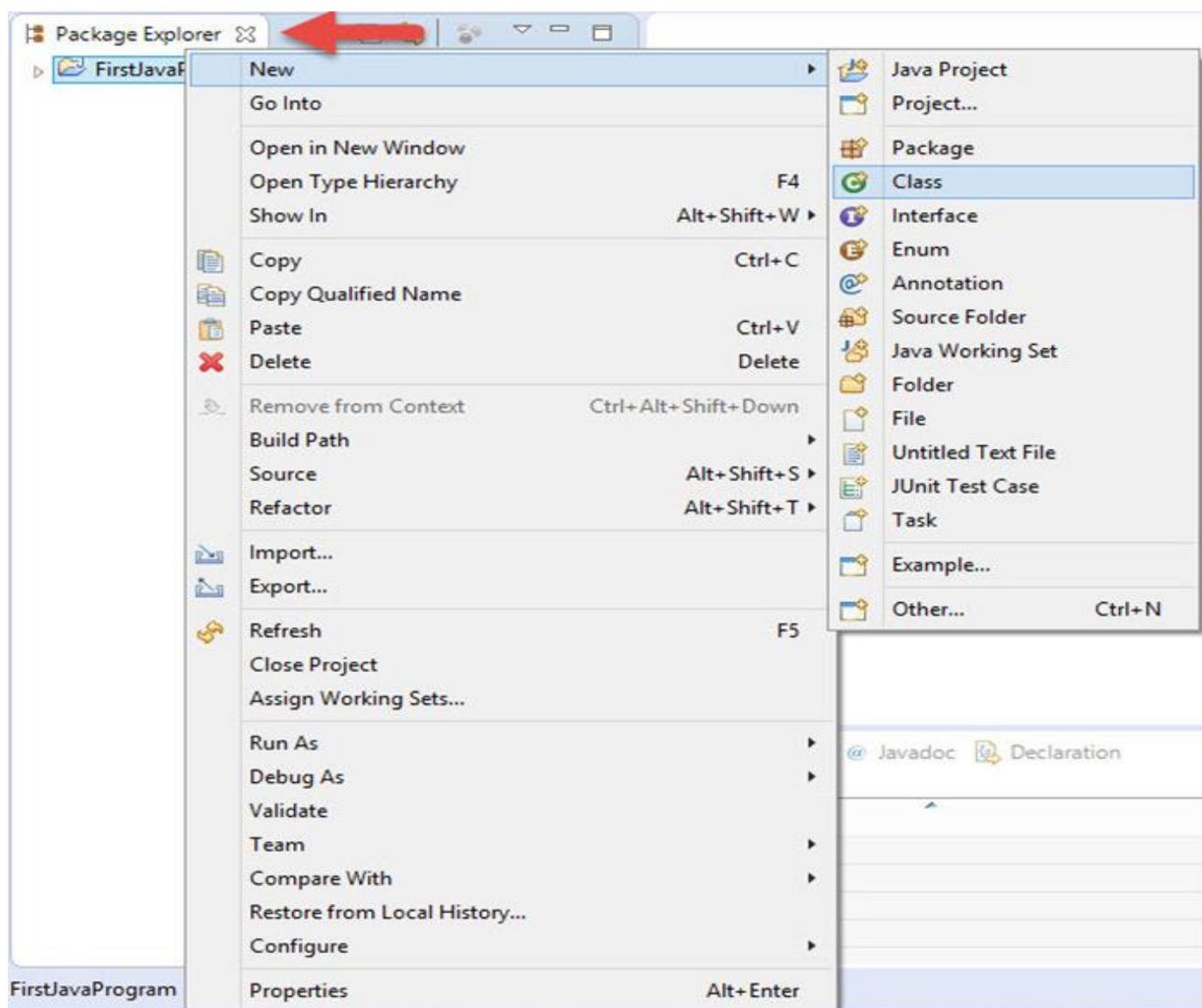


بعد از انتخاب Java Project پنجره‌ی زیر نمایش داده خواهد شد . در کادر بالا نام پروژه‌ی خود را بنویسید که من FirstJavaProgram را برای نام برنامه نوشتم



سپس روی **Finish** کلیک کنید.

حال باید یک کلاس برای پروژه‌ی خود ایجاد کنیم. برای اینکار روی پروژه‌ی خود یعنی راست کلیک کنید و مانند تصویر زیر عمل کنید



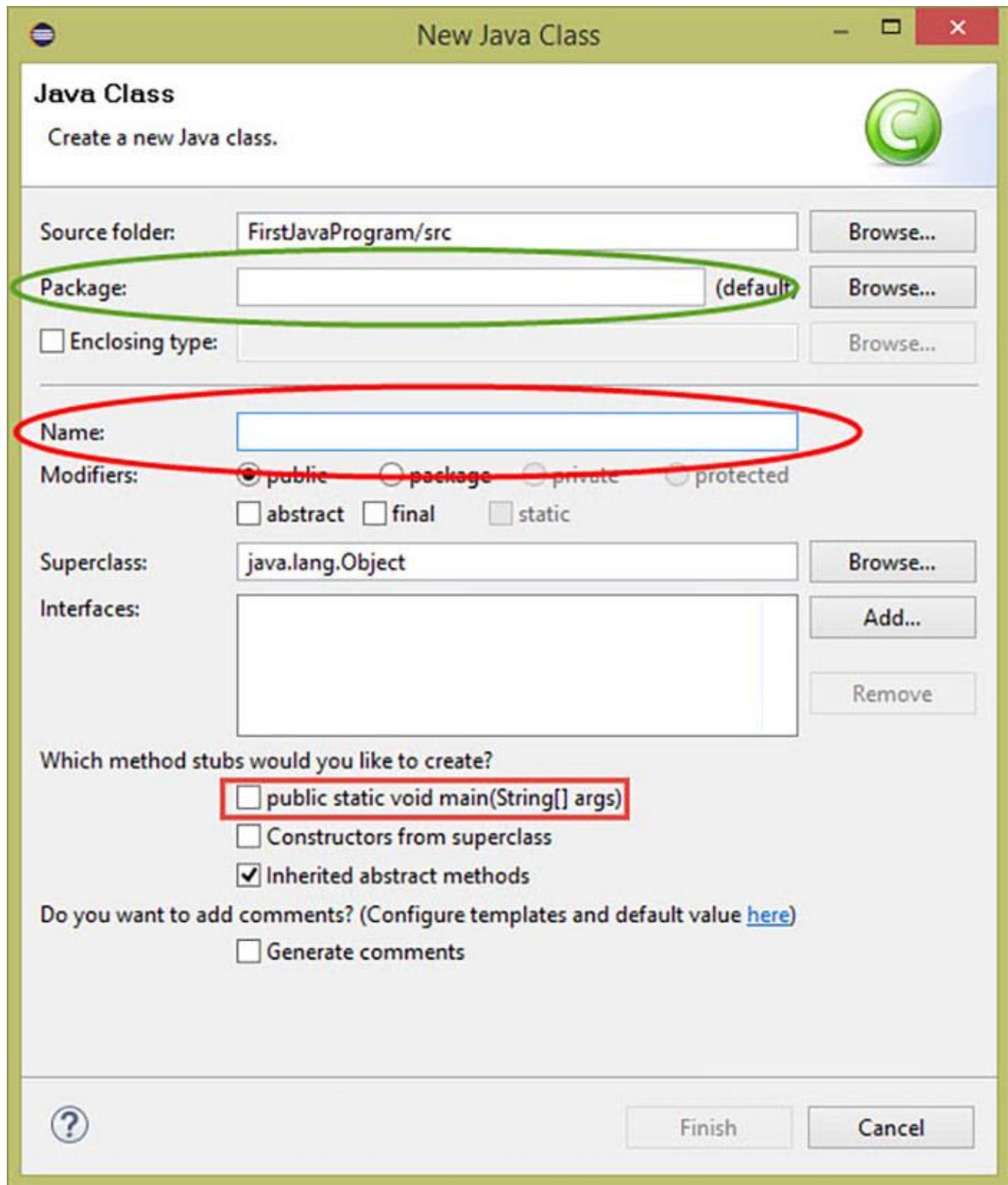
توجه: قسمتی که با فلش قرمز مشخص شده را **Package Explorer** نامیده میشود. تمام پروژه هایی که ما ایجاد می کنیم در این قسمت قرار خواهد گرفت.

بعد از کلیک روی **class** با پنجره‌ی زیر روبه رو خواهیم شد.

در بیضی سبز رنگ **Package** پکیج مشخص میشود. که بعدا در این مورد توضیح خواهم داد.

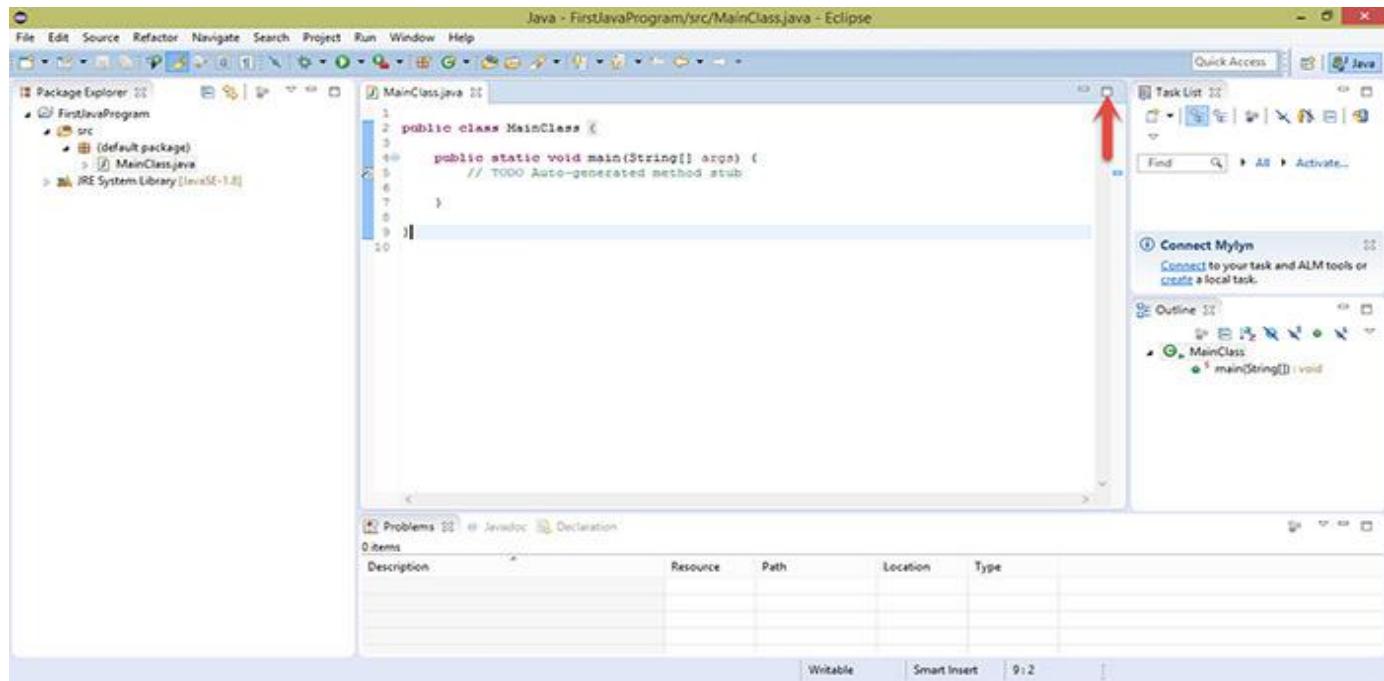
در قسمت **Name** که ب بیضی قرمز مشخص شده نام کلاس خود را می نویسیم.

در قسمت آخر که با مستطیل قرمز مشخص شده است اگر تیک دار باشد **main** به ظور خودکار متدهای **main** را در کلاس های ما پیاده سازی میکند.



روی Finish کلیک کنید.

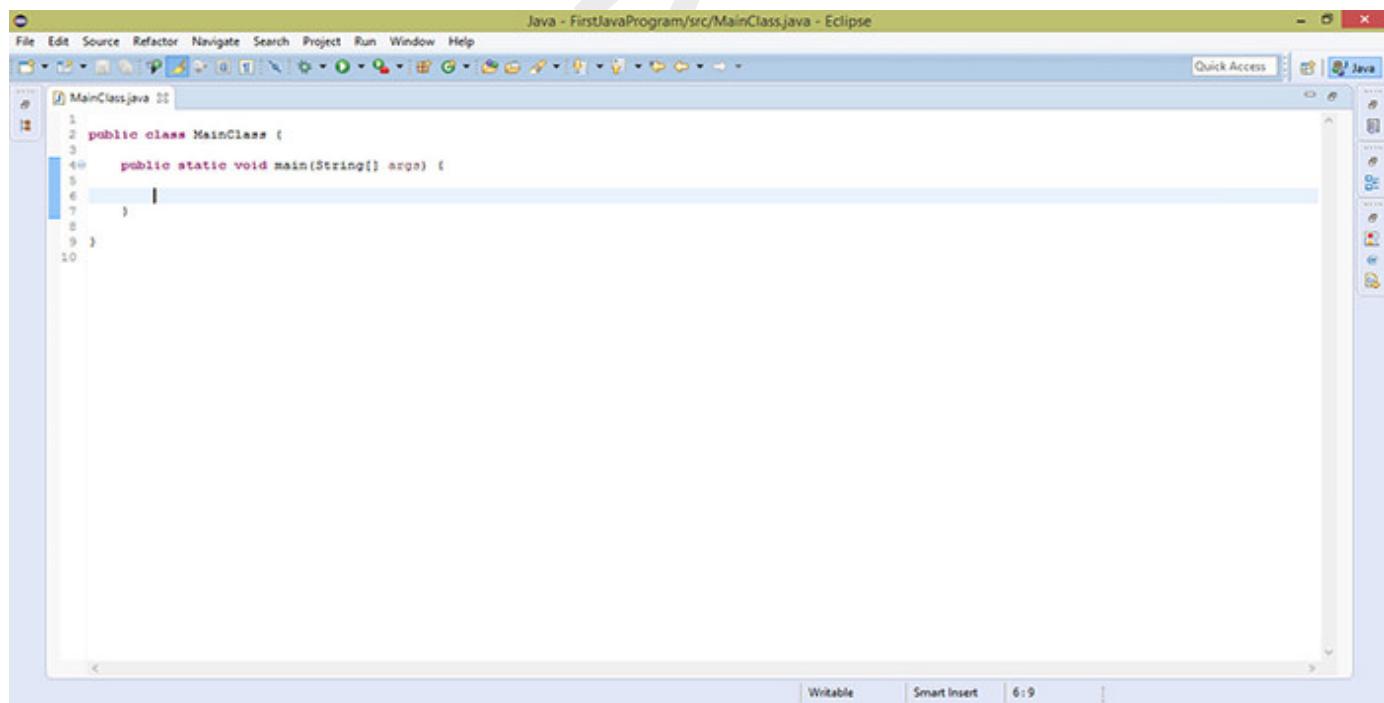
بعد از ساخته شدن کلاس با تصویر زیر مواجه خواهید شد



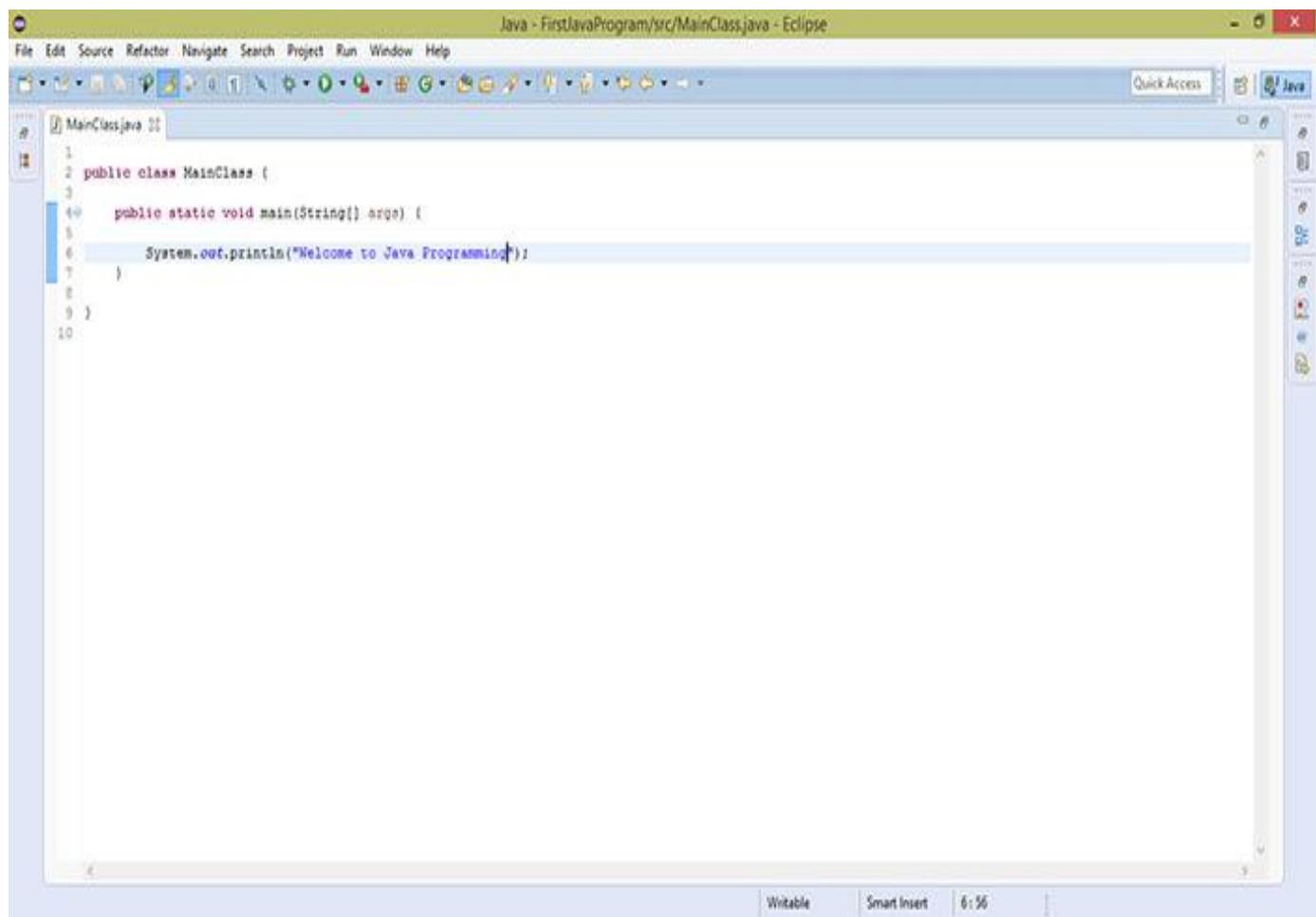
که با فلش قرمز مشخص شده کلیک کنید تا فضای بیشتری برای کد نویسی داشته باشیم.

روی دکمه **Maximize**

حال یک برنامه‌ی نمونه ایجاد می‌کنیم تا مطلب کاملاً جا بیافتد. البته نگران نباشید این برنامه فقط برای آشنایی شما با چگونگی ایجاد و اجرای یک برنامه‌ی جاوا در ایکلیپس است. و اگر کدهای نوشته شده رو نمی‌فهمید اصلاً مهم نیست!!!!!! چون در فصلهای بعدی همه‌ی کدها را ریز به ریز توضیح خواهم داد.



به عنوان مثال میخواهیم برنامه‌ی ما عبارت Welcome to Java Programming رو چاپ کند بنابرین باید از دستور System.out.println() در جاوا استفاده کنیم.

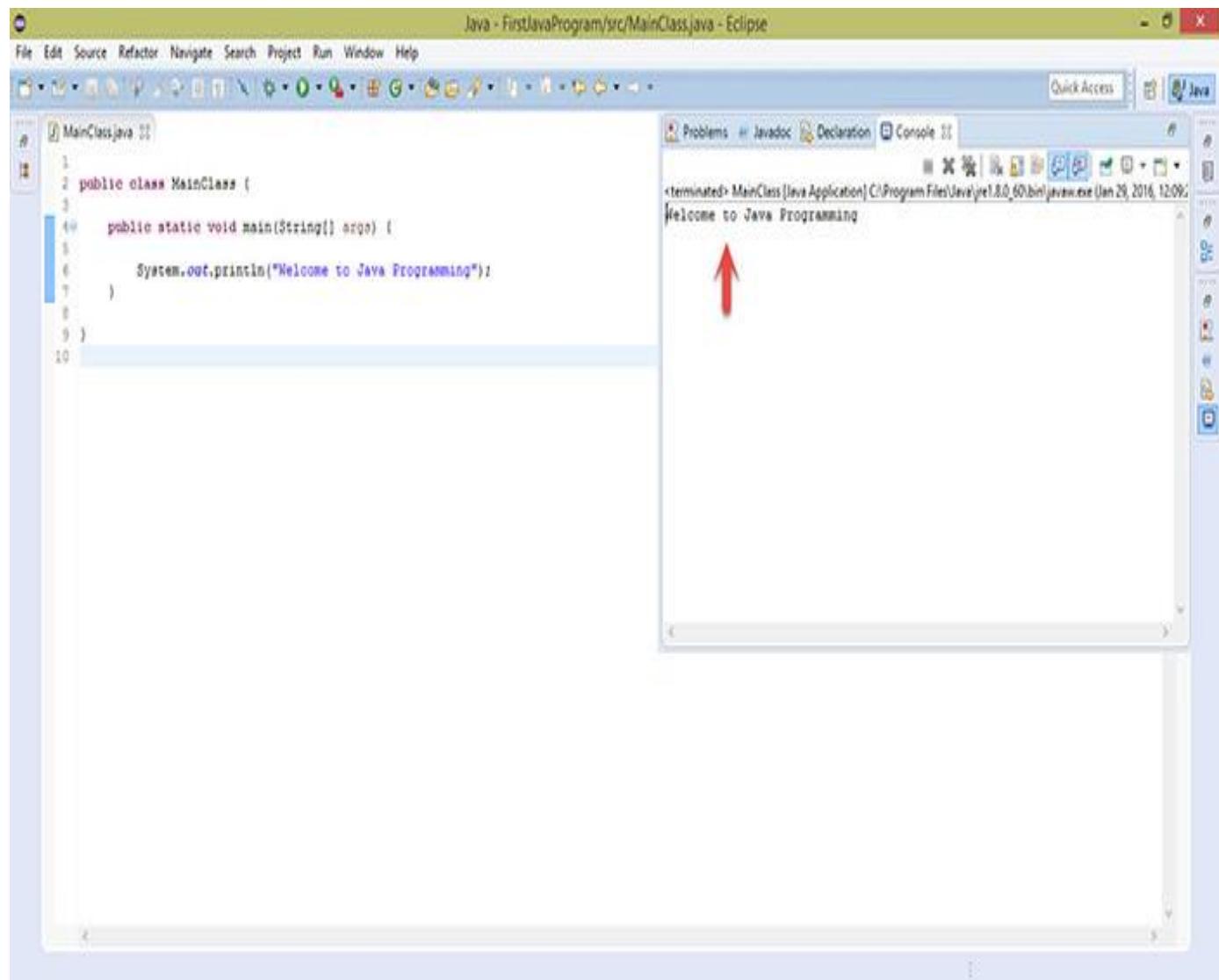


The screenshot shows the Eclipse IDE interface with the title bar "Java - FirstJavaProgram/src/MainClass.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The main editor window displays the following Java code:

```
1 public class MainClass {  
2     public static void main(String[] args) {  
3         System.out.println("Welcome to Java Programming");  
4     }  
5 }  
6  
7 }  
8  
9 }  
10 }
```

The line "System.out.println("Welcome to Java Programming");" is highlighted in blue, indicating it is selected or being edited. The status bar at the bottom shows "Writable", "Smart Insert", and "6:56".

در نهایت کلیدهای ترکیبی **CTRL + S** را بزنید تا برنامه ذخیره شود. در دست آخر کلید های **CTRL + F11** را نگه دارید تا برنامه اجرا گردد. حاصل اجرای برنامه رو در تصویر پایین می بینیم.



# فصل ۲

از برآیند

وار

## متغیر:

به طور خلاصه میتوان متغیر را اینگونه تعریف نمود :

"متغیر، اسمی برای مکانی از حافظه‌ی کامپیوتر است، که داده‌ها در آنجا ذخیره میشوند."

## انواع داده‌ها در جاوا :

به طور دقیق، جاوا دارای هشت نوع داده‌ی اصلی می‌باشد، که جدول ۱-۴ لیست کامل انواع اصلی در جاوا

را نمایش میدهد

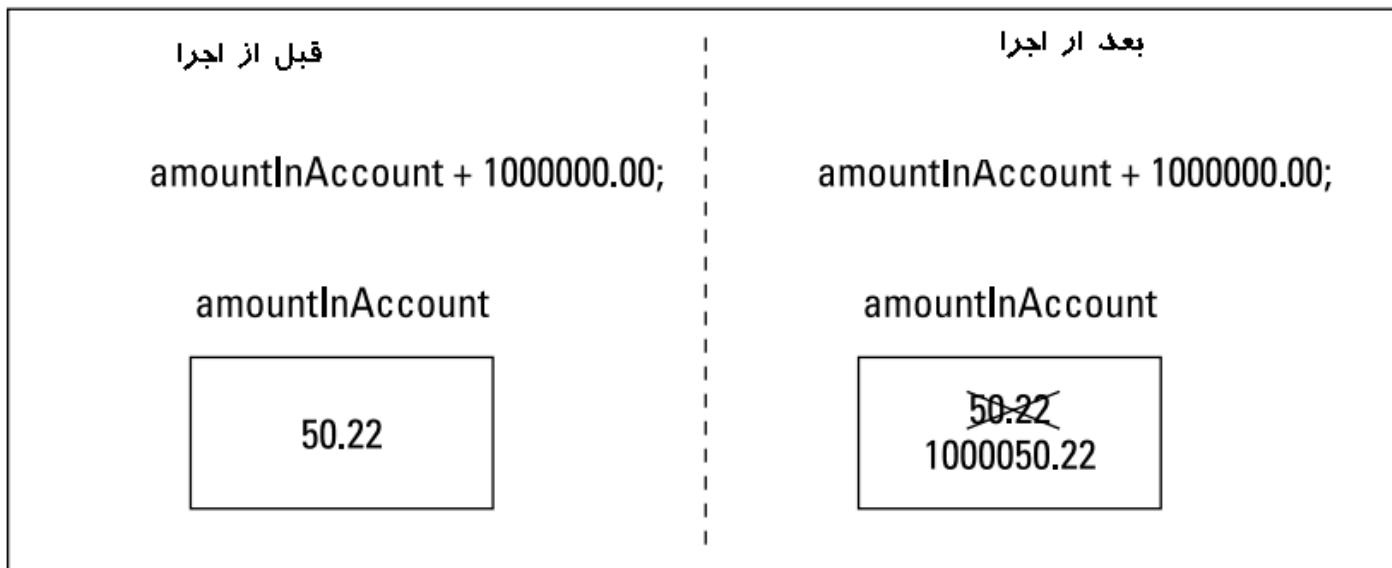
جدول ۱-۴

### انواع داده‌های اصلی جاوا

نام نوع داده	لیترال نمونه	محدوده‌ی مقادیر
انواع عددی صحیح		
byte	(byte) 42	-128 to 127
short	(short) 42	-32768 to 32767
int	42	-2147483648 to 2147483647
long	42L	-9223372036854775808 to 9223372036854775807
انواع عددی دیسیمال		
float	42.0F	$-3.4 \times 10^{38}$ to $3.4 \times 10^{38}$
double	42.0	$-1.8 \times 10^{308}$ to $1.8 \times 10^{308}$
انواع کاراکتری		
char	'A'	Thousands of characters, glyphs, and symbols
انواع منطقی		
boolean	true	true, false

## تخصیص مقدار به متغیر های دارای مقدار اولیه :

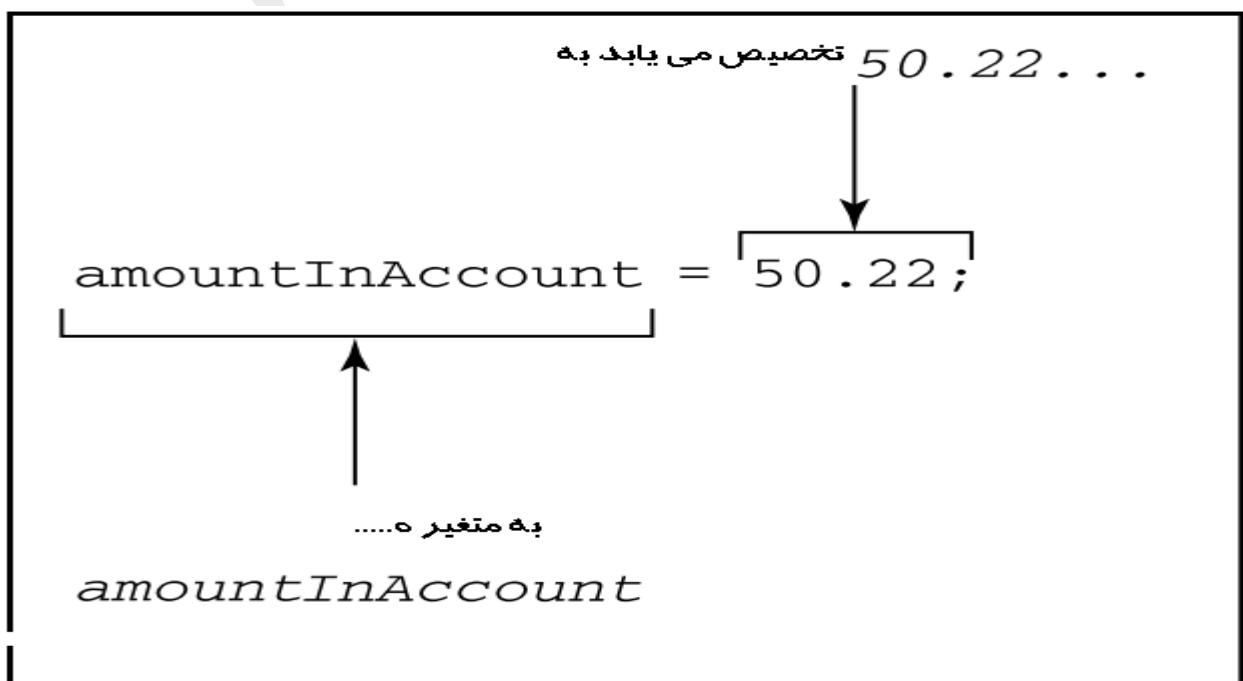
برای اینکه کاملا درک نمایید که چه انفاقی برای متغیری که دارای مقدار اولیه میباشد هنگام تخصیص مقدار جدید می افتد به شکل ۱-۴ دقیق نمایید:



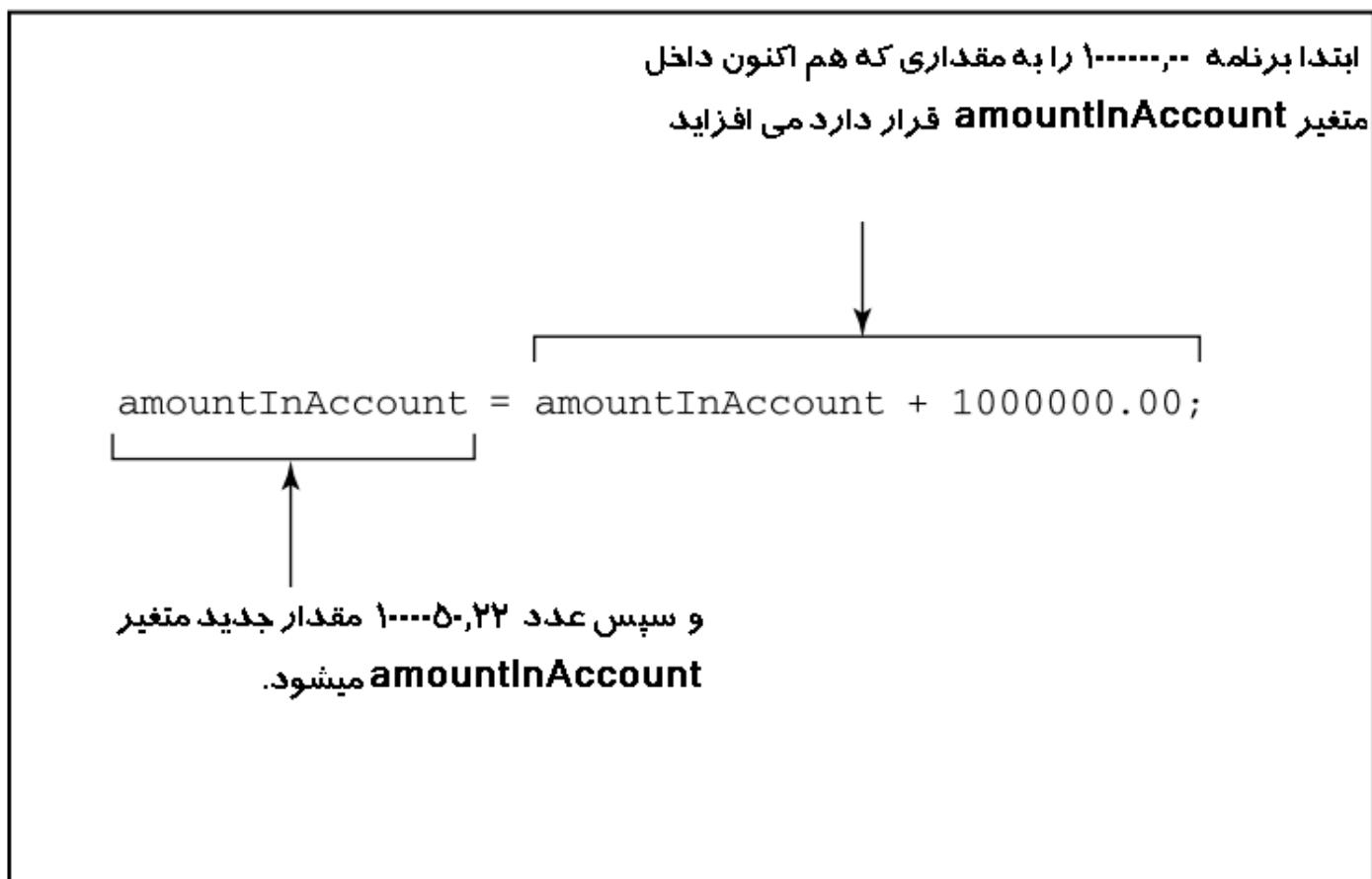
**توجه :** متغیر تغییر میکند ولی یک لیترال تغییر ناپذیر است.

## عبارات تخصیص (Assignment Statements)

در تخصیص، شما یک مقدار را به چیزی که در بیشتر مواقع متغیر است اختصاص میدهید در موقع تخصیص شما باید از عملگر = استفاده نمایید. بشکل ۲-نماینده دهنده ی همین موضوع است:



شكل ۳-۴ چگونگی افزایش مقدار متغیر `amountInAccount` را توضیح میدهد، که مقدار قبلی متغیر را به اندازه ۱ واحد افزایش می‌دهد.



(با توجه به شکل ۳-۴ مقدار اولیه متغیر `amountInAccount` مقدار ۵۰,۲۲ بوده است).

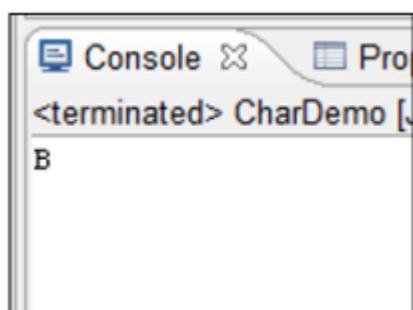
## نوع داده‌ی **char**

نوع داده‌ای که در جاوا برای ذخیره سازی کاراکتر‌ها استفاده می‌شود **char** نام دارد.

مثال ۴-۴ یک برنامه ساده که از نوع داده‌ی **char** استفاده می‌کند را نشان میدهد:

```
public class CharDemo {  
    public static void main(String args[]) {  
        char myLittleChar = 'b';  
        char myBigChar = Character.toUpperCase(myLittleChar);  
        System.out.println(myBigChar);  
    }  
}
```

خروجی حاصل از اجرای مثال ۴-۴ :



در خط ۳ متغیر **myLittleChar** تعریف شده و در ضمن با مقدار **b** مقدار

دهی شده است، دقت کنید که **b** میان دو تا تک کوتیشن قرار گرفته

است. در جاوا همیشه لیترال‌های **char** میان تک کوتیشن قرار می‌گیرند.

در خط ۴ متغیر **myBigChar** تعریف می‌گردد، و در مقدار دهی به آن ،

برنامه یک متدهای **Character.toUpperCase** را دارد فراخوانی می‌کند. متدهای

**UpperCase** همانطور که از نامش میتوان حدس زد. حرف را می‌گیرد و معادل **Character.toUpperCase**

(حروف-بزرگ) آن را باز می‌گرداند. که در اینجا متغیر **myLittleChar** که حاوی لیترال **b** می‌باشد را

گرفته ، و معادل حرف بزرگ(**UpperCase**) آن را که می‌باشد باز می‌گرداند. این **B** بازگردانده شده در

ذخیره می‌گردد.

```
char mychar='Hadi'; //Error
```

اگر سعی کنید کدی مانند بالا بنویسید ، با خطأ روبرو خواهید شد شما نمیتوانید، در یک زمان بیش از یک حرف در در متغیری از نوع *char* ذخیره کنید. اگر میخواهید کلمه یا جمله ای را در جاوا ذخیره کنید به چیزی به نام *String* نیاز خواهید داشت. (در مورد *string* در فصل های بعدی توضیح داده شده است).

## نوع **boolean**

یک متغیر نوع Boolean فقط دو مقدار را ذخیره میکند : `false` یا `true`

مثال ۵-۴ طریقه‌ی استفاده از متغیر بولین را روشن میسازد::

```
public class ElevatorFitter2 {
    public static void main(String args[]) {
        System.out.println("True or False?");
        System.out.println("You can fit all ten of the");
        System.out.println("Brickenchickerdectuplets");
        System.out.println("on the elevator:");
        System.out.println();
        int weightOfAPerson = 150;
        int elevatorWeightLimit = 1400;
        int numberOfPeople =
            elevatorWeightLimit / weightOfAPerson;

        boolean allTenOkay = numberOfPeople >= 10;
        System.out.println(allTenOkay);
    }
}
```

True or False?  
You can fit all ten of the  
Brickenchicker dectuplets  
on the elevator:

false

خروجی حاصل از اجرای مثال ۵-۴ :

در مثال بالا در خط یازدهم، متغیر `allTenOkay` از نوع `boolean` می باشد. برای یافتن ارزش (مقدار)

`allTenOkay` برنامه‌چک میکند که آیا `numberOfPeople` بزرگتر تا مساوی ده است یا نه ؟

(سمبل `>` نماد بزرگتر یا مساوی در جاوا میباشد).

در خط یازدهم `numberOfPeople >= 10` یک عبارت است. ارزش این عبارت به مقدار ذخیره شده در `numberOfPeople` بستگی دارد. اما شما با نگاه کردن به کد برنامه میتوانید بفهمید مه مقدار `numberOfPeople >= 10` بزرگتر مساوی ده نیست. بنابراین ارزش عبارت `numberOfPeople >= 10` برابر با `false` خواهد بود.

نکته : هر قسمت از برنامه های جاوا که ارزش (value) داشته باشد یک عبارت است.

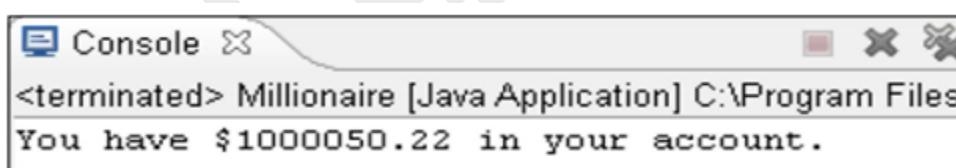
در خط هفتم، از `()` استفاده شده ، درحالی که هیچ چیزی درون پارانائز های آن وجود ندارد، این باعث میشود جاوا یک خط خالی ایجاد کند.

## نوع داده های اعشاری در جاوا:

جاوا داری دو نوع داده اعشاری اصلی می باشد: `double` و `float`. تفاوت اساسی این دونوع این در این است که `double` دارای دقیق مضاعف نسبت به `float` می باشد و میتوانید اعداد اعشاری بزرگتری را در خود جا دهد. برای مشاهده محدوده مقادیر به جدول ۱-۴ رجوع کنید. برای روشن شدن مطلب به مثال ۲-۴

```
public class Millionaire {  
    public static void main(String args[]) {  
        double amountInAccount;  
        amountInAccount = 50.22;  
        amountInAccount = amountInAccount + 1000000.00;  
        System.out.print("You have $");  
        System.out.print(amountInAccount);  
        System.out.println("in your account.");  
    }  
}
```

در زیر توجه کنید:



خروجی مثال ۲-۴ :

در خط سوم، متغیر `amountInAccount` از نوع `double` تعریف شده است. در خط پنجم مقدار قبلی `amountInAccount` با **1000000** جمع شده و حاصل داخل `amountInAccount` قرار گرفته است.

توجه کنید که F موجود در انتهای لیترال های ، برای مشخص کردن اینکه اینها از نوع float هستند به کار رفته اند.. و این مقادیر از نوع هگزادیسیمال نمی باشند.

## نوع داده ای صحیح در جاوا :

جاوا دارای چهار نوع داده ای نوع صحیح اصلی را پشتیبانی میکند. محدوده مقادیر این انواع داده ای را میتوانید در جدول ۱-۴ مشاهده نمایید. توجه داشته باشید که در مواردی که مقدار متغیر در طول برنامه به طور مکرر تغییر میکند و یا از مقدار دقیق مقادیر این متغیر ها در طول اطلاع ندارید ، بهتر است از نوع داده ای را استفاده کنید که به اندازه‌ی کافی بزرگ باشد تا احتمال سریز ( overflow ) را به حداقل برسانید. جهت فهم بهتر به مثال ۳-۴ دقت نمایید

```
public class ElevatorFitter{  
    public static void main(String args){  
        int weightOfAPerson;  
        int elevatorWeightLimit;  
        int numberOfPeople;  
        weightOfAPerson = 150;  
        elevatorWeightLimit = 1400;  
        numberOfPeople =  
            elevatorWeightLimit / weightOfAPerson;  
        System.out.print("You can fit");  
        System.out.print(numberOfPeople);  
        System.out.println("people on the elevator.");  
    }  
}
```

خروجی حاصل از اجرای برنامه‌ی بالا:

```
<terminated> ElevatorFitter [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe
You can fit 9 people on the elevator.
```

در خط سوم متغیر `elevatorWeightLimit` از نوع `int` تعریف گردیده. سپس متغیر های `weightOfAPerson` و `numberOfPeople` در خطوط بعدی تعریف شده است.

در خط هشتم ابتدا داخل پارانتز حساب می‌شود، یعنی اول عبارت `elevatorWeightLimit/PersonWeight` محاسبه شده و سپس نتیجه‌ی نهایی در متغیر `numberOfPeople` ذخیره می‌گردد.

## انواع مرجع (Reference Types)

با ترکیب چیزی‌های ساده شما عناصر پیچیده‌تری را به دست می‌آورید. به طور مشابه چند نوع اصلی در

```
import javax.swing.JFrame;

public class ShowAFrame {

    public static void main(String args[]) {
        JFrame myFrame = new JFrame();

        String myTitle = "Blank Frame";

        myFrame.setTitle(myTitle);
        myFrame.setSize(300, 200);
        myFrame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
        myFrame.setVisible(true);
    }
}
```

جاوا را میتوان با یکدیگر ترکیب کرد و انواع مرجع را به دست آورد. برای روشن سازی بهتر مطلب به مثال

۴-۶ را که در زیر آورده به دقت توجه کنید:



خروجی حاصل از برنامه ۴-۶ :

در مثال ۴-۶ دو نوع مرجع تعریف گردیده است که هر دوی آنها در API جاوا تعریف شده اند. (اولی را که شما در بیشتر اوقات استفاده خواهید کرد) String نام دارد. نوع دیگر (که شما از آن در ساخت GUI ها استفاده خواهید کرد) JFrame نام دارد.

گروهی از کاراکتر هاست، آن شبیه آن است که چندین مقدار char را در یک سطر داشته باشد. متغیر myTitle از نوع String اعلان گردیده و "Blank Frame" به آن تخصیص داده شده است. توجه داشته باشید که کلاس String در API جاوا اعلان شده است.

در برنامه های جاوا، علامت دابل کوتیشن در ابتدا و انتهای لیترال های String باید وجود داشته باشد.

JFrame در جاوا بسیار شبیه به window میباشد. (با این تفاوت که شما JFrame را بجای window فراخوانی میکنید).

سعی نکنید تمام خطوط مثال ۴-۶ تفسیر کنید. تنها چیزی که باید از این مثال یادبگیرید، اعلان دو متغیر مرجع، یعنی myFrame از نوع JFrame و myTitle از نوع String میباشد. (در فصول بعدی در مورد دیگر خطوط این مثال توضیح داده خواهد شد).

نکته‌ی مهم دیگر آن هست که `JFrame` و `String` نام کلاس میباشند و در حالت کلی هر کلاس در جاوا یک نوع مرجع میباشد. (در مورد کلاس‌های جاوا دو فصل‌های بعدی توضیح داده خواهد شد).

شما میتوانید متغیر یازنوع `double` را به صورت زیر اعلان کنید:

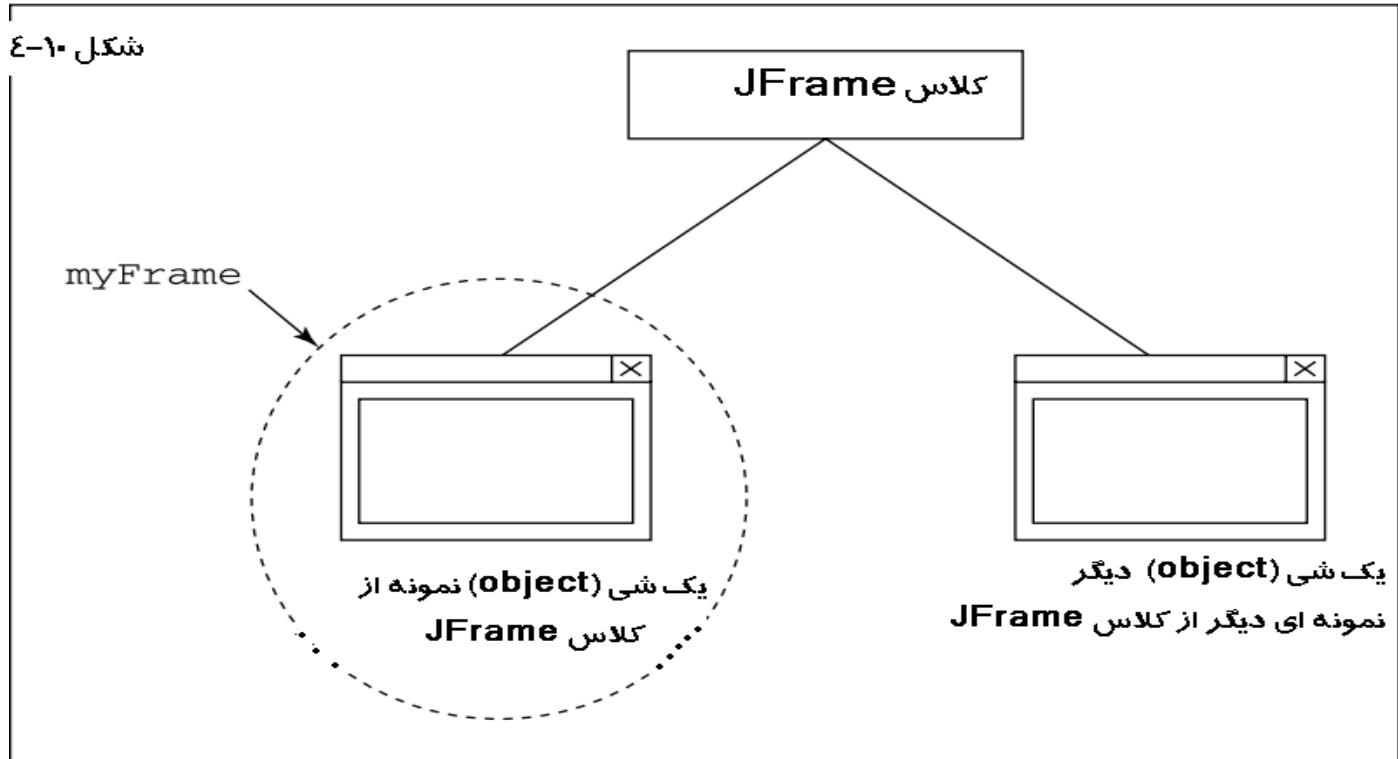
```
double amountInAccount;  
double amountInAccount = 50.22;
```

```
JFrame myFrame;  
JFrame myFrame = new JFrame();
```

شما همچنین میتوانید `myframe` را از نوع `JFrame` اختصاص دهید:

در شکل ۱۰-۴ متغیر `myframe` به مثال و نمونه‌ای از کلاس `JFrame` ارجاع میکند. (در فصول بعدی در مورد کلاس و شی به طور کامل توضیح داده خواهد شد).

شکل ۶-۱۰



در خط اول مثال ۶-۴ شما اعلان میکنید که در این برنامه از `javax.swing.JFrame` در طول برنامه استفاده

خواهید کرد. اگر خط یک را از برنامه ۶-۴ حذف کنید مجبور هستید هر جا که از `JFrame` استفاده میکنید.

نام کامل `javax.swing.JFrame` را تکرار کنید. به درک بهتر برنامه زیر را با حذف خط اول از مثال ۶-۴ باید

```
public class ShowAFrame {  
    public static void main(String args[]) {  
        javax.swing.JFrame myFrame =  
            new javax.swing.JFrame();  
        String myTitle = "Blank Frame";  
        myFrame.setTitle(myTitle);  
        myFrame.setSize(3200, 200);  
        myFrame.setDefaultCloseOperation  
            (javax.swing.JFrame.EXIT_ON_CLOSE);  
        myFrame.setVisible(true);  
    }  
}
```

برنامه را به شکل زیر تغییر دهید.

### ایجاد مقادیر جدید با اعمال عملگرها (Creating New Values by Applying Operators)

عملیات روی داده ها: مقدار یک متغیر ممکن است نتیجه ی یک عبارت محاسباتی و یا سایر عملیات روی داده ها باشد. برای انجام عملیات مختلف بر روی داده ها از **عملگر ها** استفاده میشود.

ابتدا برای توضیح کار کرد عملگرها محاسباتی در جاوا و اینکه از آن ها چگونه در برنامه هایمان استفاده کنیم به جدول زیر توجه نمایید

ناتیجه	مثال	نام عملگر	عملگر	نوع عملگر
۲۰	$۵*۴$	ضرب	*	محاسباتی
۲	$۱۰/۴$	تقسیم	/	
۷	$۲+۵$	جمع	+	
۴	$۹-۵$	منها	-	
۳	$۵٪۲۳$	باقیمانده	%	

عملگر های حسابی در جدول شرح داده شده اند نکته ای که نباید فراموش کرد این است حاصل تقسیم عدد صحیح بر عدد صحیح دیگری عدد صحیح خواهد بود . البته این نکته از جدول بالا هم قابل مشاهده و است. حال برای روشن شدن هر چه بیتر عملگر های حسابی با مثال ۷-۴ که در زیر آورده شده توجه نمایید:

```
import static java.lang.System.out;  
  
public class MakeChange {  
  
    public static void main(String args[]) {  
  
        int total = 248;  
  
        int quarters = total / 25;  
  
        int whatsLeft = total % 25;  
  
        int dimes = whatsLeft / 10;  
  
        int whatsLeft = whatsLeft % 10;  
  
        int nickels = whatsLeft / 5;  
  
        int whatsLeft = whatsLeft % 5;  
  
        int cents = whatsLeft;  
  
        out.println("From " + total + " cents you get");  
  
        out.println(quarters + " quarters");  
  
        out.println(dimes + " dimes");  
  
        out.println(nickels + " nickels");  
  
        out.println(cents + " cents");  
    }  
}
```

```
From 248 cents you get  
9 quarters  
2 dimes  
0 nickels  
3 cents
```

به خط اول برنامه‌ی بالا توجه نمایید؛ آن را با خط اول مثال ۶-۴

```
import javax.swing.JFrame;
```

با اضافه نمودن `import static java.lang.System.out;` به مثال ۶-۴ کد برنامه کمی برای نوشتن آسانتر

شده و حتی خواندن آن را نیز سهیل می‌شود. بدین صورت که شما بجای `System.out.println` فقط از عبارت کوتاه‌تر `out.println` استفاده کرده‌اید.

باید توجه نمود شما می‌توانید خط اول برنامه `out` را حذف نموده و

به جای عبارت `System.out.println` عبارت `out.println` را جایگزین آن نمایید.

ولی اما در مثال ۶-۴ کاری که برنامه انجام میدهد: خرد کردن پول ۲۴۸ سنتی به سکه‌های رایج در ایالات متحده یعنی: ۱ سنتی، ۵ سنتی (nickel)، ۱۰ سنتی (dime)، ۲۵ سنتی (quarter) می‌باشد. کلاس `MakeChange` کمترین تعداد سکه‌های که با آن‌ها می‌توان یک ۲۴۸ سنتی را خرد کرد ارائه میدهد.

### عملگر های افزایش (increment) و کاهش (decrement) :

جاوا عملگر های بسیار مرتبی دارد که کار را برای ما ساده می‌کند. عملگر افزایشی یک واحد افزایش و عملگر کاهشی یک واحد کم می‌کند. عملگر افزایشی را با دو تا بعلاوه (`++`) و عملگر کاهش را با دو تا منهای (`--`) نشان خواهیم داد. برای آنکه بفهمید آن‌ها چگونه کار می‌کنند به چند تا مثال نیاز دارید.

```

import static java.lang.System.out;
public class preIncrementDemo {
    public static void main(String args[]) {
        int numberOfBunnies = 27;

        ++numberOfBunnies;
        out.println(numberOfBunnies); ←
        out.println(++numberOfBunnies); ←
        out.println(numberOfBunnies); ←
    }
}

```

numberOfBunnies

۲۸ میشود

چاپ میشود ۲۸

numberOfBunnies

۲۹ میشود و سپس چاپ میگردد

دوباره ۲۹ چاپ میشود

کار کرد این عملگر ها بستگی دارد که شما آن ها را کجای متغیر به کار برد ه باشید..اگر آن را قبل از یک

متغیر به کار ببرید آن **preincrement** نامیده میشود. به مثال ۱۲-۴ توجه نمایید:

شما `++` را قبل از متغیر قرار میدهید و کامپیووتر قبل از اینکه از متغیر مورد نظر در یک عبارت دیگر استفاده

کند مقدار آن را یک واحد افزایش میدهد.

برای مثال در خط هفتم برنامه ابتدا `numberOfBunnies` یک واحد افزایش یافته و سپس چاپ میگردد.

ولی اگر عملگر افزایش یعد از متغیر قرار گیرد (`postincrement`), نمایده میشود و ابتدا متغیر با همان

مقدار دز یک عبارت شرکت کرده و سپس افزایش صورت میگیرد. برای درک بهتر به مثال ۱۴-۴ توجه

نمایید:

```

import static java.lang.System.out;
public class postIncrementDemo {
    public static void main(String args[]) {
        int numberOfBunnies = 27;

        numberOfBunnies++; <-- چاپ میشود ۲۸
        out.println(numberOfBunnies); <-- چاپ میشود و سپس
        out.println(numberOfBunnies++); <-- چاپ میگردد ۲۹
        out.println(numberOfBunnies); <-- چاپ میگردد ۲۹
    }
}

```

numberOfBunnies

۲۸ میشود

۲۸ چاپ میگردد

۲۸ چاپ میشود و سپس

**numberOfBunnies**

۲۹ میگردد

۲۹ چاپ میگردد

نکته ای که ممکن است بی دققی به آن موجب فاجعه شود نادیده گرفتن تفاوت مابین preincrement و postincrement میباشد و این به برنامه‌ی شما بستگی دارد که از کدام یک استفاده کنید.

نکته ۱ : در `--numberOfBunnies` کامپیوتر ابتدا یک واحد از متغیر کم کرده و بعد آن را در بقیه‌ی عبارت استفاده میکند.

نکته ۲ : در `numberOfBunnies--` کامپیوتر ابتدا متغیر را با مقدار قبلی در عبارت حساب کرده و سپس از مقدار آن یک واحد میکاهد.

## عملگر های تخصیص (Assignment Operators)

جاوا عملگر های تخصیص گوناگونی دارد که شما میتوانید عملیات های جمع و تفریق و ضرب تقسیم و ... را انجام دهید. برای مثال در مثال پایین توجه کنید چگونه `numberOfBunnies` به مقدار متغیر `5+=` می‌افزاید. و یا چگونه `2=*` مقدار `numberOfBunnies` را دوباره میکند..... برای آموختن عملکرد این عملگرها به مثال ۸-۴ توجه نمایید :

```

public class UseAssignmentOperators {
    public static void main(String args[]) {
        int numberOfBunnies = 27;
        int numberExtra = 53;
        numberOfBunnies += 1;
        System.out.println(numberOfBunnies);
        numberOfBunnies += 5;
        System.out.println(numberOfBunnies);
        numberOfBunnies += numberExtra;
        System.out.println(numberOfBunnies);
        numberOfBunnies *= 2;
        System.out.println(numberOfBunnies);
        System.out.println(numberOfBunnies -= 7);
        System.out.println(numberOfBunnies = 100);
    }
}

```

```

28
33
86
172
165
100

```

خروجی ::

دو خط آخر مثال ۸-۴ ویژگی های خاص عملگر هاس جاوا میباشند، شما میتوانید عملیات تخصیص در به عنوان بخشی از یک عبارت در جاوا انجام دهید. در خط سیزدم عملگر ۷ واحد از `numberOfBunnies` میکاهد و مقدار

System.out.println از ۱۷۳ به ۱۶۵ کاهش پیدا میکند. و بعد با این مقدار جدید فراخوانی میشود. و در خط چهاردهم نشان میدهد که تخصیص مقدار به یک متغیر میتواند خود بخشی از یک عبارت بزرگتر باشد.

# فصل ۳

فایل دانلود

## Making Decisions (Java if Statements)

زمانی که شما در حال نوشتن برنامه های کامپیوتری هستید دائما در معرض انتخاب مسیر برنامه قرار دارید - (آیا کاربر پسوردش را به درستی وارد کرده است؟ اگر بله، پس اجازه بده کاربر وارد سیستم شده و با آن کار کند. اگر خیر، پس به کاربر اجازه‌ی ورود نده). پس شما در برنامه های جاوا به راهی برای شاخه شاخه کردن برنامه و انتخاب بین شاخه ها احتیاج دارید. خوشبختانه جاوا برای این منظور راه حلی دارد که عبارات `if` نامیده می‌شوند.

ابتدا یک مثال زده و سپس از روی مثال زیر عبارات `if` و چند مبحث دیگر را توضیح خواهم داد.

```
import static java.lang.System.out;
import java.util.Scanner;
import java.util.Random;
public class GuessingGame {
    public static void main(String args[]) {
        Scanner keyboard = new Scanner(System.in);
        out.print("Enter an int from 1 to 10: ");
        int inputNumber = keyboard.nextInt();
        int randomNumber = new Random().nextInt(10) + 1;
        if (inputNumber == randomNumber) {
            out.println("*****");
            out.println("*You win.*");
            out.println("*****");
        } else {
            out.println("You lose.");
            out.print("The random number was ");
            out.println(randomNumber + ".");
        }
        out.println("Thank you for playing.");
        keyboard.close();
    }
}
```

خروجی مثال ۱-۵ :

```
Enter an int from 1 to 10: 2
*****
*You win.*
*****
Thank you for playing.
```

```
Enter an int from 1 to 10: 4
You lose.
The random number was 10.
Thank you for playing.
```

برنامه ۱-۵ یک بازی ساده است. یک عدد حدسى را از کاربر گرفته و سپس یک عدد تصادفى مابین ۱ تا ۱۰ تولید میکند. اگر عدد وارد شده توسط کاربر با عدد تصادفى تولید شده توسط برنامه یکسان باشد، کاربر برنده شده و در غیر این صورت بازنده است و برنامه به کاربر میگوید که عدد تصادفى چه بوده است.

### کنترل کلید های فشار داده شده از کیبورد:

به این تکه کد که برشی از مثال ۱-۵ است دقت نمایید :

```
importjava.util.Scanner;

Scanner keyboard = new Scanner(System.in);

intinputNumber = keyboard.nextInt();
```

این شبه کد هر عددی که کاربر از طریق کیبورد وارد کند را گرفته و در خط سوم آن را در متغیر inputNumber قرار میدهد. زمانی که شما میخواهید از کیبورد داده ای را دریافت کنید دو خط اول را فقط یکبار در برنامه قرار میدهید. بعده در برنامه تان، هر زمان که نیاز باشد کاربر داده ای int ای را تایپ

کند باید کدی همانند خط سوم از شبه کد بالا که شامل فرآخوانی `nextInt` باشد را بنویسید . تا داده ای

```
importjava.util.Scanner;  
  
Scanner readingThingie = new Scanner(System.in);  
  
intvalueTypedIn = readingThingie.nextInt();
```

وارد شده از کیبورد گرفته شده و در متغیر مورد نظر ذخیره شود.

تمام کلمات موجود در شبه کد سه خطی بالا بجز دوتای آن ها جزئی از جاوا بوده و در جاوا موجود میباشند. آن دو نامی که من از خودم انتخاب کردم `inputNumber` و `keyboard` میباشند. پس برای روشن تر ساختن موضوع شبه کد بالا را فقط در از جنبه‌ی نام‌هایی که خودم انتخاب کردم تغییر میدهم :

من نمیخواهم تمام مفاهیم و جزئیات مثال ۱-۵ را توضیح بدهم ولی ذکر چند نکته ضروری ست :

**نکته ۱ :** وقتی `import java.util.Scanner` را به کار بردید از کلمه‌ی کلیدی `static` استفاده نمیکنید. وقتی `import java.lang.System.out` را به کار بردید از کلمه‌ی کلیدی `static` استفاده کردید. این امر به این خاطر است که `Scanner` نام کلاس است. در حالی که `System.out` نام هیچ کلاسی در جاوا نیست.

**نکته ۲ :** معمولاً (روی کامپیوتر های رومیزی و لب تاب ها) کلمه‌ی `System.in` برای استفاده کیبورد به کار میروند.

**نکته ۳ :** زمانی که شما انتظار دارید کاربر مقدار عددی صحیحی را وارد نماید از `(nextInt)` استفاده کنید. اگر انتظار دارید مقداری اعشاری وارد نماید از `(nextDouble)` استفاده نمایید. اگر انتظار دارید یکی از مقادیر درست یا غلط را وارد نماید از `(nextBoolean)` استفاده نمایید. اگر انتظار دارید که کاربر کلماتی نظیر `Hadi, Java, Computer` و... را وارد نماید از `(next)` استفاده نمایید.

**نکته ۵:** شما میتوانید چندین مقدار را از طریق کیبورد یکی پی از دیگری دریافت نمایید. برای این کار از `keyboard.nextInt()` چندین بار استفاده نمایید.

**نکته ۶:** زمانی که شما از `Scanner` جاوا استفاده میکنید، شما باید بعد از آخرین فراخوانی `nextInt()` را فراخوانی نمایید. مانند خط بیست و سوم از مثال ۱-۵.

## ایجاد کردن مقادیر تصادفی

رسیدن به مقادیر واقعی تصادفی به طور شگفت آوری مشکل است. یک کامپیوتر در تولید مقادیر تصادفی ممکن است به نظر رسد که واقعاً دارد مقادیر تصادفی تولید میکند. ولی در انتهای تنها چیزی که به آن گفته شده را انجام میدهد و این یک فشن و روش کاملاً قطعی است نه تصادفی.

پس در مثال ۱-۵ زمانی که کامپیوتر کد زیر را اجرا میکند:

```
import java.util.Random;  
  
int randomNumber = new Random().nextInt(10) + 1;
```

به نظر مقادیر وتصادفی مایین ۱ تا ۱۰ تولید میکند ولی این مقادیر در اصل، واقعاً تصادفی نیستند. کامپیوتر فقط از دستور العمل های داد شده به آن پیروی میکند.

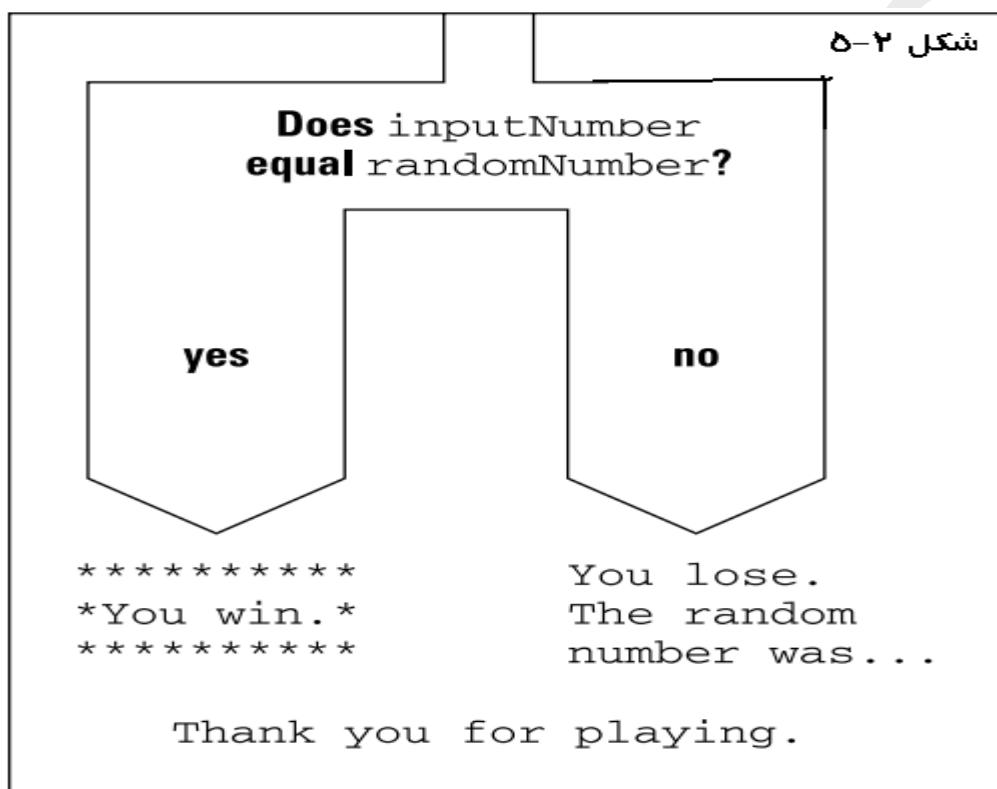
نگران نباشید که `new Random().nextInt()` چه معنی میدهد. هدف از مثال ۱-۵ یادگیری یک سری مباحث مورد نظر این فصل میباشد. اگر معنی برخی کد ها در مثال هایی که می آوریم جای نگرانی نیست. چون برخی از این کدها و مفاهیم و فصول بعدی توضیح داده خواهد شد.

دستورات انتخابی:

همان طور که در مثال ۱-۵ مشاهده میکنید عبارت if یک انشعاب در برنامه را نشان میدهد. کامپیوتر فقط یکی از دو شاخه را انتخاب خواهد کرد شاخه win یا شاخه lose را. کامپیوتر یکی از دو شاخه را با تست کردن درست یا غلط بودن شرط ، انتخاب خواهد کرد. شرطی که تست خواهد شد :

inputNumber == randomNumber

شکل ۲-۵ دستور if را به صورت گرافیکی نشان میدهد:



همانطور که از شکل پیداست.

آیا `inputNumber` با `randomNumber` برابر است؟ اگر شرط درست باشد، برنامه بلوک مایین شرط و کلمه `else` را اجرا خواهد کرد. اما اگر شرط نادرست باشد کامپیوتر بلوک بعد از کلمه `else` را اجرا

خواهد نمود. به هر حال کامپیوتر آخرین فرآخوانی `println` را اجرا خواهد کرد و را نمایش خواهد داد.

### **: (The double equal sign) == علامت**

علامت مساوی = برای تخصیص یک لیترال به یک متغیر یا تخصیص مقدار یک متغیر به یک متغیر دیگر و .... به کار میروند. در حال که علامت دو تا مساوی == برای مقایسه دو مقدار به کار برده میشود تا بفهمیم که آیا آنها دارای مقداری یکسان اند یا خیر ؟

## کاربرد if بدون else

به مثال ۱-۵ دوباره نگاهی بیاندازید، شاید نخواهید به کاربر بگویید که او باخته است. در این صورا باید مثال

۱-۵ را به صورت زیر ویرایش کنید مثال ۲ :

```
import static java.lang.System.in;
import static java.lang.System.out;
import java.util.Scanner;
import java.util.Random;
public class DontTellThemTheyLost {
    public static void main(String args[]) {
        Scanner keyboard = new Scanner(in);
        out.print("Enter an int from 1 to 10: ");
        int inputNumber = keyboard.nextInt();
        int randomNumber = new Random().nextInt(10) + 1;
        if (inputNumber == randomNumber) {
            out.println("*You win.*");
        }
        out.println("That was a very good guess :-)");
        out.print("The random number was ");
        out.println(randomNumber + ".");
        out.println("Thank you for playing.");
        keyboard.close();
    }
}
```

خروجی مثال ۵-۲ :

```
Enter an int from 1 to 10: 4
*You win.*
That was a very good guess :-
The random number was 4.
Thank you for playing.
```

```
Enter an int from 1 to 10: 4
That was a very good guess :-
The random number was 6.
Thank you for playing.
```

مثال ۵-۲ بخش else ندارد اگر inputNumber برابر باشد برنامه You win را چاپ خواهد نمود در غیر این صورت برنامه You win را چاپ نخواهد کرد.

### ایجاد شرط با عملگر های مقایسه ای و منطقی :

جاوا عملگر ها و ابزار های بسیار فراوانی برای نیاز های متنوع شما در ایجاد شرط داراست. در این بخش قصد داریم به این موضوع پردازیم.

جدول ۱-۵ به شما نشان میدهد که چگونه ی میتوانید از عملگر های مقایسه ای برای مقایسه یک چیز با دیگری استفاده کنید.

جدول ۱-۵

عملگر های مقایسه ای

نماد عملگر	معنا	مثال
==	برابر است با	numberOfCows == 5
!=	برابر نیست با	buttonClicked != panicButton
<	کوچک تر است از	numberOfCows < 5
>	بزرگتر است از	myInitial > 'B'
<=	کوچکتر مساوی است	numberOfCows <= 5
>=	بزرگتر مساوی است	myInitial >= 'B'

## مقایسه اشیاء (Comparing objects)

زمانی که شما کار کردن با اشیاء را شروع کردید، شما خواهید دانست که میتوانید از `==` و `!=` برای مقاسه اشیاء با یکدیگر استفاده کنید. برای مثال یک دکمه که شما روی نمایشگر کامپیوتر می بینید یک شیء است. شما میتوانید با نوشتن یک برنامه جویا شوید که آیا یک چپ کلیک روی دکمه خاص روی نمایشگر کامپیوتر اتفاق است یا نه؟ شما میتوانید این کار را با استفاده از عملگر های کیفیت جاوا (Java's equality operator) انجام دهید.

```
if (e.getSource() == bCopy) {  
    clipboard.setText(which.getText());
```

اما برای مقایسه دو `String` باید توجه داشته باشد که، هنگام این مقایسه استفاده از `==` بدین معنی است که "آیا این `String` دقیقا در محلی از حافظه که `String` دیگر قرار دارد، ذخیره شده است". معمولا شما چنین خواسته اس ندارید. حال برای پرسیدن اینکه "آیا این `String` دقیقا همان کاراکترهایی را دارد که `String` دیگر داراء میباشد" یک متده به اسم `equals` وجود دارد که در API (Application Programming Interface) `String` تعریف گردیده است. `equals` دو `String` را مقایسه می مند که ببیند که آیا آن ها کاراکتر های یکسانی دارند یا خیر؟ به مثال ۳-۵ توجه نمایید.

در مثال ۳-۵ (`keyboard.next()`) فراخوانی شده و هر چیزی که کاربر روی کیبورد کامپیوتر تایپ کند را گردآوری میکند. سپس کلمه `password` را در متغیری به نام `password` ذخیره میکند و در نهایت با استفاده از `if` تفاوت `==` و متده `equals` نشان داده می شود.

در فراخوانی متده `equals` مهم نیست که کدام `String` داخل پارانائز و کدام یک قبل از نقطه باشد ، برای مثال میتوانستیم بخط بیست و دمدم را به شکل زیر نیز بنویسیم:

```
if ("swordfish".equals(password))
```

که در این صورت معنی و کارکرد برنامه ۳-۵ هیچ نغیری نمیکرد.

```
import static java.lang.System.*;
import java.util.Scanner;
public class CheckPassword {
    public static void main(String args[]) {
        out.print("What's the password?");
        Scanner keyboard = new Scanner(in);
        String password = keyboard.next();
        out.println("You typed >>" + password + "<<");
        out.println();
        if (password == "swordfish") {
            out.println("The word you typed is stored");
            out.println("in the same place as the real");
            out.println("password. You must be a");
            out.println("hacker.");
        } else {
            out.println("The word you typed is not");
            out.println("stored in the same place as");
            out.println("the real password, but that's");
            out.println("no big deal.");
        }
        out.println();
        if (password.equals("swordfish")) {
            out.println("The word you typed has the");
            out.println("same characters as the real");
            out.println("password. You can use our");
            out.println("precious system.");
        } else {
            out.println("The word you typed doesn't");
            out.println("have the same characters as");
            out.println("the real password. You can't");
            out.println("use our precious system.");
        }
        keyboard.close();
    }
}
```

خروچی مثال ۵-۳ :

```
What's the password? swordfish
You typed >>swordfish<<

The word you typed is not
stored in the same place as
the real password, but that's
no big deal.

The word you typed has the
same characters as the real
password. You can use our
precious system.
```

در اولین خط برنامه ۵-۳ یک راه تبلی را نشان میدهد شما با نوشتن `*` یک راه سهل و آسان را انتخاب کرده اید ، این یک خط تقریباً معادل ۳۰ عدد `import` جداگانه‌ی دیگر مانند `System.out` ، `System.err`، `System.nanoTime` و `System.in`: کاربردن `(*ستاره)` در برنامه‌های بزرگ است که باعث کوتاه‌تر شدن کد برنامه خواهد شد.

## عملگر های منطقی در جاوا

جاوا تعدادی عملگر برای کارئ تست مقادیر منطقی دارد که در جدول زیر نشان داده شده‌اند :

عملگر های منطقی		
نام عملگر	معنا	مثال
<code>&amp;&amp;</code>	و	<code>5 &lt; x &amp;&amp; x &lt; 10</code>
<code>  </code>	یا	<code>x &lt; 5    10 &lt; x</code>
<code>!</code>	نفیض(نه منطقی)	<code>!password.equals("swordfish")</code>

شما میتوانید از این عملگرها در شرط‌ها به طور مهرانه‌ای استفاده کنید. به مثال ۴-۵ دقیق نمایید:

حاصل اجرای چندین بار مثال ۴-۵ در زیر نشان داده شده است ، زمانی که یوزرنیم bburd و پسورد swordfish می باشد و یا زمانی که یوزرنیم hritter و پسورد preakston می باشد. کاربر پیام مناسب را که نشان می دهد یوزرنیم و پسورد صحیح هستند را دریافت میکند، در غیر این صورت پیامی مبنی در نامعتبر بودن یوزرنیم و پسورد دریافت میکند.

```
import javax.swing.JOptionPane;
public class Authenticator {
    public static void main(String args[]) {
        String username =
        JOptionPane.showInputDialog("Username:");
        String password =
        JOptionPane.showInputDialog("Password:");
        if (
            username != null && password != null &&
            (
                (username.equals("bburd") &&
                password.equals("swordfish")) ||
                (username.equals("hritter") &&
                password.equals("preakston")))
            )
        )
        {
            JOptionPane.showMessageDialog
                (null, "You're in.");
        } else {
            JOptionPane.showMessageDialog
                (null, "You're suspicious.");
        }
    }
}
```



مثال ۴-۵ یک روش تازه را برای گرفتن ورودی از کاربر نشان میدهد. روس صفحه نمایش یک کادر پیام (دیالوگ) مبنی در درخواست از کاربر برای وارد کردن ورودی، نشان داده می شود.

```
String password = JOptionPane.showInputDialog("Password:");
```

درمثال ۴-۵ دوبار از `JOptionPane.showInputDialog` استفاده شده است. یک بار برای گرفتن یوزرنیم و دیگر بار برای گرفتن پسورد. برای مثال `JOptionPane.showInputDialog("Username")` یک کادر محاوره ای (دیالوگ) شامل فیلدی برای وارد کردن متن و نیز دکمه های `OK` و `Cancel` به نمایش در می

آورد. زمانی که کاربر بروی OK کلیک می نماید. کامپیوتر هر آنچه که در فیلد متن وارد شده را داخل یک متغیر قرار می دهد.

حال به قسمت دوم ، خط یازدهم برنامه دقیق نمایید:

```
JOptionPane.showMessageDialog (null, "You're in.");
```

این خط نیز یک کادر محاوره ای(دیالوگ) نمایش می دهد با این تفاوت که هیچ فیلد متنی برای وترد کردن اطلاعات وجود ندارد. نام API در javax.swing.JOptionPane لدر جاوا درون چیزی که نامیده میشود، تعریف شده است. دلیل وجود خط اول نیز همین موضوع است.

در مثال بالا JOptionPane.showInputDialog به زیبایی کار میکند زیرا کاربر یوزرنیم و پسورد را به صورت رشته ای از کاراکتر ها وارد میکند. اگر شما میخواهد کاربر مقدار int را وارد کند باید چیزی شبیه

```
int numberOfCows = Integer.parseInt(JOptionPane.showInputDialog("How many cows?"))
```

تایپ کنید. و یا برای دریافت کردن مقدار double از کار بر باید چیزی شبیه به زیر را تایپ کنید.

```
double fractionOfHolsteins = Double.parseDouble(JOptionPane.showInputDialog("Holsteins:"))
```

توجه داشته باشید که username != null به این معنی است که یوزرنیم تهی نیست، یا اینکه یوزنیم دارای یک مقدار است.

توجه : دو عبارات `username != null` و `password != null` اختیاری نیستند. اگر شما آن ها را از برنامه حذف کنید و سپس برنامه را اجرا کنید و در کادر ظاهر شده روی Cancel کلیک کنید برنامه مقابل چشمانتان خطای `NullPointerException` را میگیرد. برای تمرین این خطوط را حذف کرده و دوباره برنامه را اجرا کنید تا عملا با این نوع خطا آشنا شوید.

کاربرد کلمه `if` کلیدی `null` در خطوط دهم و یازدهم کاملاً متفاوت است، فراخوانی `showMessageDialog` به جاوا میگوید که یک کادر محاوره ای (دیالوگ) جدید ایجاد کند. کلمه `null` بر این موضوع اشاره میکند که کادر محاوره ای جدید بر روی هیچ یک از کادر های محاوره ای قبلی سبز نشود.

## ساخت `if` های تودرتو

در جاوا می توان از یک `if` داخل `if` دیگر استفاده کرد. به مثال ۵-۵ دقیق:

```
import static java.lang.System.out;
import java.util.Scanner;
public class Authenticator2 {
    public static void main(String args[]) {
        Scanner keyboard = new Scanner(System.in);
        out.print("Username: ");
        String username = keyboard.next();
        if (username.equals("bburd")) {
            out.print("Password: ");
            String password = keyboard.next();
            if (password.equals("swordfish")) {
                out.println("You're in.");
            } else {
                out.println("Incorrect password");
            }
        } else {
            out.println("Unknown user");
        }
        keyboard.close();
    }
}
```

خروجی حاصل از چندین بار اجرای مثال ۵-۵ :

```
Username: bburd  
Password: swordfish  
You're in.
```

```
Username: bburd  
Password: catfish  
Incorrect password
```

```
Username: jschmoe  
Unknown user
```

شما برای وارد شدن باید هر دوی یوزرنیم و پسورد را درست وارد کنید.)

به عبارتی دیگر هر دو شرط if باید درست باشد). شرط اول درستی

یوزرنیم را بررسی میکند. شرط دوم درستی پسورد را بررسی میکند. اگر شما

اولین شرط(یوزنیم) را درست وارد کنید. شما به سوی if دیگری که شرط

دوم(پسورد) را بررسی میکند پیش روی میکنید. در اولین if شرط(یوزنیم) را نادرست وارد کنید، هرگز

شرط if دوم (پسورد) از شما پرسیده نخواهد شد.

آیا "bburd" برابر "username" است

خیر

بله

Unknown user

آیا "swordfish" برابر "password" است

خیر

بله

Incorrect password

You're in

برای درک بهتر  
موضوع به نمودار  
گرافیکی ۵-۷ دقیق کنید :

البته این مثال فقط برای یادگیری if های تو در تو آورده شده و شاید در کاربرد آن در عمل باید کمی تغییرات روی آن انجام دهیم: ۱- پسورد وارد شده توسط کاربر به صورت ستاره دار نشان داده نمیشود. این امنیت را پایین می آورد ۲- هیچ گونه عمل کد گذاری(encrypting) روی پسورد انجام نمیشود. ۳- که کابران غیر قانونی که سعی در وارد شدن به سیستم دارند نشان داده میشود که کدام یک از username یا psword را اشتباه وارد میکنند.....هدف از گفتن این مسائل فقط آشنا کردن شما بود و قصد پیاده کردن این مسائل را در این فصل نداریم.

## در جاوا Switch

وقتی که قصد داریم از تعداد زیادی گزینه فقط یک مورد آنها را انتخاب کنیم دستور switch کاربرد دارد:  
ابتدا به مثال ۵-۶ دقت نمایید :

```
import static java.lang.System.out;
import java.util.Scanner;
public class JustSwitchIt {
    public static void main(String args[]) {
        Scanner keyboard = new Scanner(System.in);
        out.print("Which verse? ");
        int verse = keyboard.nextInt();
        switch (verse) {
            case 1:
                out.println("That's because he has no brain.");
                break;
            case 2:
                out.println("That's because he is a pain.");
                break;
            case 3:
                out.println("'Cause this is the last refrain.");
                break;
            default:
                out.println("No such verse. Please try again.");
                break;
        }
        out.println("Ohhhhhh . . .");
        keyboard.close();
    }
}
```

```
Which verse? 2  
That's because he is a pain.  
Ohhhhhh. . . .
```

```
Which verse? 6  
No such verse. Please try again.  
Ohhhhhh. . . .
```

. ابتدا کاربر یک عدد مانند عدد ۲ را وارد میکند. سپس

برنامه به ابتدای عبارت switch یعنی خط هشتم میرسد.

کامپیوتر مقدار متغیر verse را بررسی میکند. بعد از آنکه

کامپیوتر تشخیص داد که مقدار متغیر verse عدد ۲ است. هر یک از case های عبارت switch را بررسی

میکند. عدد ۲ با اولین case مطابقت ندارد. پس کامپیوتر به case دوم حرکت میکند. مقدار متغیر verse

با مقدار case دوم مطابقت دارد پس کامپیوتر دستوری که بلافاصله بعد از case دوم میآید را اجرا

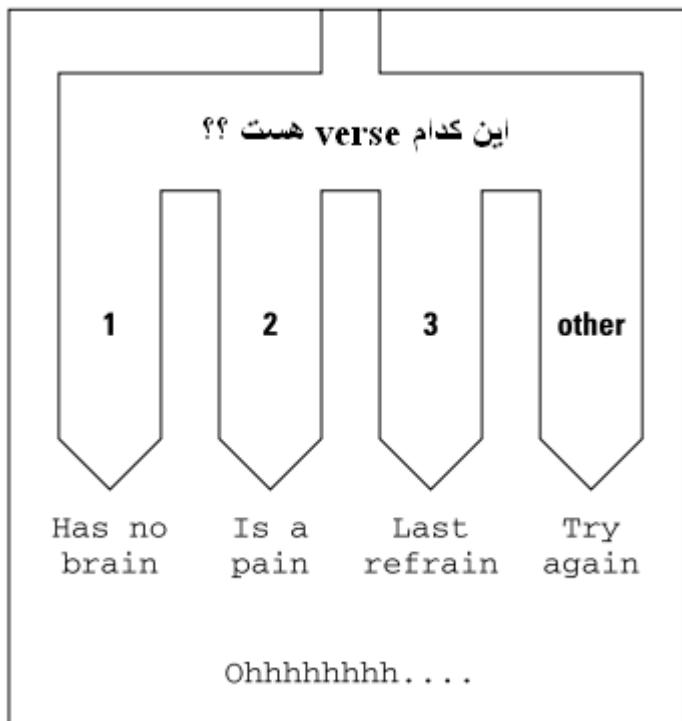
میکند.

حال دوم دارای 2 دستور است که اولی That's because he is a pain را نمایش میدهد. و دومی

دستور break است. زمانی که کامپیوتر به دستور break میرسد، برنامه از switch خارج شده و به

اولین دستور بعد switch که در اینجا (\*\*\*\*\*\*) out.println("\*\*\*\*\*") میباشد میرسد. شکل

: ۹-۵



حال اگر دستور break را حذف کنیم چه اتفاقی

می افتد؟؟ در این صورت دستورات تمام

های بعدی نیز اجرا خواهند شد.

مثالاً اگر break را از case دوم حذف کنیم بعد از نمایش برنامه از خارج نشده و Cause this is the last refrain نیز نمایش داده خواهد شد. خوب با توجه با این نکته موجود در default احیاری نیست. اما فراموش کردن break در انتهای هر case یک خطای رایج در برنامه نویسی است. برای که شما با عواقب ناخوشایند این فراموشی آشنا شوید به مثال ۷-۵ دقت کنید.

```
import static java.lang.System.out;
import java.util.Scanner;
public class FallingForYou {
    public static void main(String args[]) {
        Scanner keyboard = new Scanner(System.in);
        out.print("Which verse? ");
        int verse = keyboard.nextInt();
        switch (verse) {
            case 3:
                out.print("Last refrain, ");
                out.println("last refrain,");
            case 2:
                out.print("He's a pain, ");
                out.println("he's a pain,");
            case 1:
                out.print("Has no brain, ");
                out.println("has no brain,");
        }
        out.println("In the rain, in the rain.");
        out.println("Ohhhhhhhh... ");
        out.println();
        keyboard.close();
    }
}
```

خروجی حاصل :

```
Which verse? 1  
Has no brain, has no brain,  
In the rain, in the rain.  
Ohhhhhhhh...
```

```
Which verse? 2  
He's a pain, he's a pain,  
Has no brain, has no brain,  
In the rain, in the rain.  
Ohhhhhhhh...
```

```
Which verse? 3  
Last refrain, last refrain,  
He's a pain, he's a pain,  
Has no brain, has no brain,  
In the rain, in the rain.  
Ohhhhhhhh...
```

```
Which verse? 6  
In the rain, in the rain.  
Ohhhhhhhh...
```

همانطور که مشاهده میکنید. ما ۳ را وارد کردیم ، که عدد ۳ با اولین case مطابقت دارد و دستورات اولین case اجرا میشود ، ولی چون break را فراموش کردیم، دستورات case های بعدی هم به ترتیب اجرا شده اند.

## جديد و بپرورد یافته Switch

در مثال های ۶-۵ و ۷-۵ ، متغیر verse ( از نوع int ) دستور switch را به یکی از case ها هدایت میکند.. یک متغیر int داخل دستور switch در همه ورژن های جاوا کار میکند- در ورژن های قدیمی جاوا، فقط انواع char و تعداد کمی از انواع دیگر داخل دستور switch کار میکرنند- اما اگر شما از جاوا ۷ با نسخه های جدید تر استفاده میکنید، میتوانید از مقادیر String داخل switch برای انتخاب و اجرای یکی از case ها استفاده کنید. اگر شما از جاوا برای ایجاد اپلیکیشن های Android استفاده میکنید توجه کنید که در جاوا ۶ و کلیه ورژن های قدیمی جاوا ، شما نمی توانید برای پرسش میان case ها داخل دستور switch از مقادیر string استفاده نمایید. به مثال ۸ دقیق کنید

```
import static java.lang.System.out;
import java.util.Scanner;
public class SwitchIt7 {
    public static void main(String args[]) {
        Scanner keyboard = new Scanner(System.in);
        out.print("Which verse (one, two or three)? ");
        String verse = keyboard.next();
        switch (verse) {
            case "one":
                out.println("That's because he has no brain.");
                break;
            case "two":
                out.println("That's because he is a pain.");
                break;
            case "three":
                out.println("Cause this is the last refrain.");
                break;
            default:
                out.println("No such verse. Please try again.");
                break;
        }
        out.println("Ohhhhhhhh. . .");
        keyboard.close();
    }
}
```

Which verse (one, two or three)? two  
That's because he is a pain.  
Ohhhhhhhh. . .

خروجی :

# فصل از خود بپنداش

## تکرار دستور العمل ها بازهای و بارهای ( دستور while در جاوا)

در اینجا یک بازی حدس زدن برای شما مثال زده شده است. کامپیوتر اعداد تصادفی از یک تا ده تولید میکند. و سپس کامپیوتر از شما میخواهد که اعداد را حدس بزنید اگر حدس شما اشتباه بود بازی ادامه می‌یابد، و چنانچه حدس شما درست باشد بازی خاتمه می‌یابد. سپس با استفاده از این مثال به توضیح حلقه `while` خواهیم پرداخت. مثال ۶-۱ :

```
import static java.lang.System.out;
import java.util.Scanner;
import java.util.Random;
public class GuessAgain {
    public static void main(String args[]) {
        Scanner keyboard = new Scanner(System.in);
        int numGuesses = 0;
        int randomNumber = new Random().nextInt(10) + 1;
        out.println("*****");
        out.println("Welcome to the Guessing Game");
        out.println("*****");
        out.println();
        out.print("Enter an int from 1 to 10: ");
        int inputNumber = keyboard.nextInt();
        numGuesses++;
        while (inputNumber != randomNumber) {
            out.println();
            out.println("Try again...");
            out.print("Enter an int from 1 to 10: ");
            inputNumber = keyboard.nextInt();
            numGuesses++;
        }
        out.print("You win after ");
        out.println(numGuesses + " guesses.");
        keyboard.close();
    }
}
```

## خروجی مثال ۶-۱:

```
*****  
Welcome to the Guessing Game  
*****  
  
Enter an int from 1 to 10: 2  
  
Try again...  
Enter an int from 1 to 10: 5  
  
Try again...  
Enter an int from 1 to 10: 8  
  
Try again...  
Enter an int from 1 to 10: 3  
You win after 4 guesses.
```

در این خروجی کامپیوتر یک عدد تصادفی تولید کرده و از کاربر خواسته و عدد تولید شده را حدس بزنند، کاربر در اولین حدس خود اشتباه میکند. ولی در تلاش دوم عدد تصادفی را به درستی

حدس میزند. و پیام .... You Win را دریافت میکند، توجه کنید که به ازای هر حدس متغیر numGuesses را درست میزند. و پیام .... You Win را دریافت میکند، توجه کنید که به ازای هر حدس متغیر

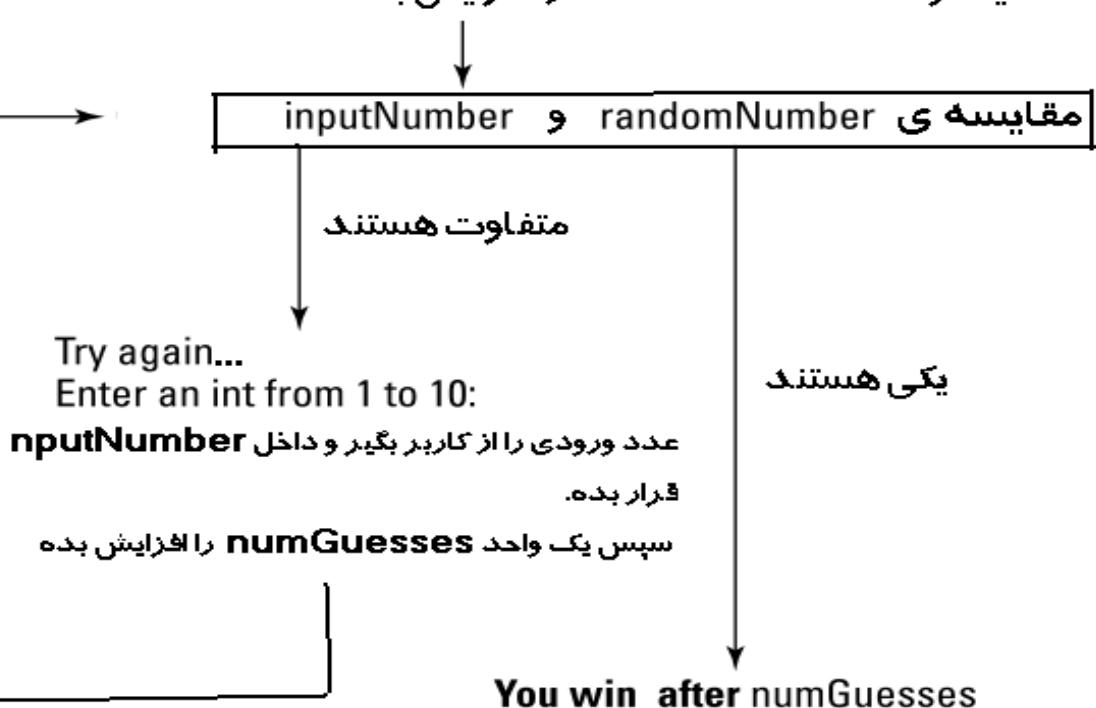
یک واحد افزایش می یابد.

به خط شانزدهم دقیق کنید کاری که این دستور انجام میدهد این است که تا زمانی که متغیر randomNumber ( عدد تصادفی تولید شده توسط کامپیوتر) و متغیر inputNumber ( عدد وارد شده توسط کاربر) نامساوی باشند. دستورات خطوط ۱۷ الی ۲۱ را تکرار نماید. ۶-۲ کارکرد این مثال را

روشن

میکند:

Welcome to the Guessing Game  
Enter an int from 1 to 10:  
عدد ورودی را بگیر و داخل **inputNumber** قرار بده.  
یک واحد افزایش **numGuesses** را افزایش بده.



اگر یکبار دیگر کد مثال بالا را مرور کنید. خواهید دانست که تنها کاری که حلقه `while` انجام میدهد این است که تا زمانی که شرط اش درست باشد، دستورات موجود در اولین بلوک بعدی اش را تکرار میکند..

## تکرار تا تعدادی معین(حلقه `for` در جاوا)

ابتدا به مثال ۶-۲ توجه نمایید، سپس حلقه `for` از روی این مثال توضیح داده خواهد شد.

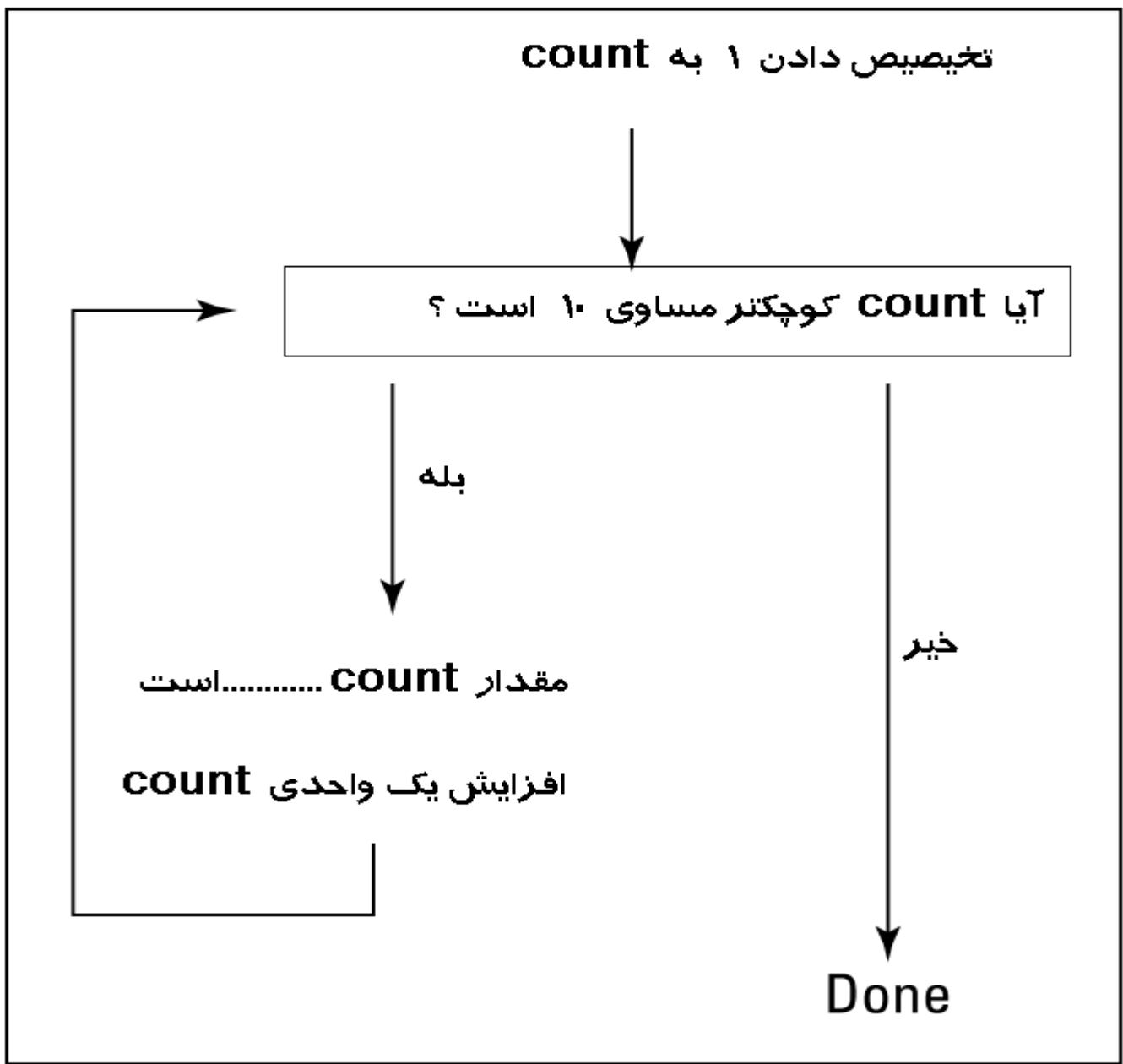
```
import static java.lang.System.out;
public class Yawn {
    public static void main(String args[]) {
        for (int count = 1; count <= 10; count++) {
            out.print("The value of count is ");
            out.print(count);
            out.println(".");
        }
        out.println("Done!");
    }
}
```

```
The value of count is 1.
The value of count is 2.
The value of count is 3.
The value of count is 4.
The value of count is 5.
The value of count is 6.
The value of count is 7.
The value of count is 8.
The value of count is 9.
The value of count is 10.
Done!
```

خروجی :

زمانی که برنامه را اجرا می کنید، حلقه `for` با تخصیص دادن یک به متغیر `count` شروع می شود. سپس مقدار `count` بررسی می شود تا اطمینان حاصل شود که کوچکتر مساوی ده می باشد. اگر درست بود برنامه پیش می رود، و دستورات مابین دو آکولاد را اجرا می کند، و در آخر یک واحد به `count` می افزاید.

شکل ۶-۲ کارکرد حلقه for در این مثال را توضیح میدهد:



اجرای حلقه for ادامه می‌یابد تا اینکه مقدار count به ۱۱ می‌رسد. در این هنگام شرط حلقه (کوچکتر مساوی بودن مقدار count از ۱۰) نادرست می‌شود. پس اجرای حلقه به پایان میرسد. و برنامه از اولین دستوری که بلافاصله بعد از for آمده است، ادامه پیدا می‌کند.

حالت کلی دستور for به صورت زیر است:

for ( initialization ; expression ; update)

(initialization) مقدار دهی اولیه : زمانی که برنامه برای اولین بار به عبارت `for` میرسد فقط یکبار اجرا میشود.

(Expression شرط) : چندین بار ارزیابی میشود (قبل از هر تکرار).  
(Update به روز رسانی متغیر حلقه) : چندین بار اعمال میشود (بعد از هر تکرار).

```
import static java.lang.System.out;
public class AlIsAllWet {
    public static void main(String args[]) {
        for (int verse = 1; verse <= 3; verse++) {
            out.print("Al's all wet. ");
            out.println("Oh, why is Al all wet? Oh,");
            out.print("Al's all wet 'cause ");
            out.println("he's standing in the rain.");
            out.println("Why is Al out in the rain?");
            switch (verse) {
                case 1:
                    out.println("That's because he has no brain.");
                    break;
                case 2:
                    out.println("That's because he is a pain.");
                    break;
                case 3:
                    out.println("Cause this is the last refrain.");
                    break;
            }
            switch (verse) {
                case 3:
                    out.println("Last refrain, last refrain,");
                case 2:
                    out.println("He's a pain, he's a pain,");
                case 1:
                    out.println("Has no brain, has no brain,");
            }
            out.println("In the rain, in the rain.");
            out.println("Ohhhhhhhh... ");
            out.println();
        }
        out.print("Al's all wet. ");
        out.println("Oh, why is Al all wet? Oh,");
        out.print("Al's all wet 'cause ");
        out.println("he's standing in the rain.");
    }
}
```

مثال بالا مربوطی هم به مطالب فصل قبلی میکند. دو عبارت switch داخل حلقه i for به کار رفته اند. یکی از این switch ها از دستور break استفاده میکند و دیگری نه.

متغیر verse به عنوان متغیر حلقه i for عمل میکند ابتدا مقدار ۱ و سپس ۲ و درنهایت مقدار ۳ را به خود میگیرد که در هر یک از این حالات case های مناسب در عبارات switch اجرا میشوند. زمانی که شرط حلقه نادرست میشود (مقدار count بزرگتر از سه میشود) برنامه بلا فاصله از حلقه خارج شده و چهار دستور انتهای برنامه را اجرا میکند.

### اجرا تا زمانی که آنچه میخواهید را به دست آورید (حلقه i do در جاوا)

اولین سوالی که ممکن از خود پرسید این است که این حلقه چه تفاوتی با حلقه i while دارد. جواب این است که حلقه i while حداقل یکبار اجرا خواهد شد. چون شرط حلقه در انتهای حلقه بررسی میشود.

بعد مثال ۴-۶ توضیحات بیشتری داده خواهد شد.

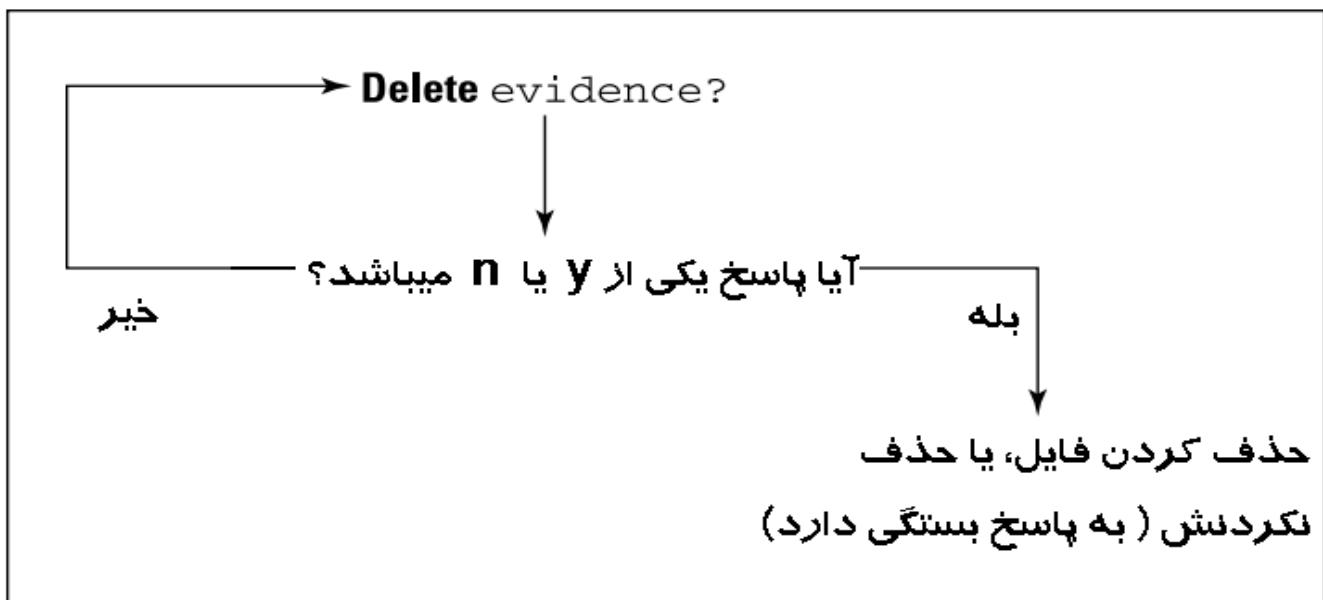
```
import java.io.File;
import static java.lang.System.out;
import java.util.Scanner;
public class DeleteEvidence {
    public static void main(String args[]) {
        File evidence = new File("cookedBooks.txt");
        Scanner keyboard = new Scanner(System.in);
        char reply;
        do {
            out.print("Delete evidence? (y/n) ");
            reply =
                keyboard.findWithinHorizon(".", 0).charAt(0);
            } while (reply != 'y' && reply != 'n');
        if (reply == 'y') {
            out.println("Okay, here goes...");
            evidence.delete();
            out.println("The evidence has been deleted.");
            } else {
            out.println("Sorry, buddy. Just asking.");
            }
        keyboard.close();
    }
}
```

خروجی حاصل از دوبار اجرای مثال بالا :

```
Delete evidence? (y/n) n  
Sorry, buddy. Just asking.  
  
Delete evidence? (y/n) u  
Delete evidence? (y/n) Y  
Delete evidence? (y/n) L  
Delete evidence? (y/n) S  
Delete evidence? (y/n) .  
Delete evidence? (y/n) y  
Okay, here goes...  
The evidence has been deleted.
```

قبل از حذف فایل برنامه از کاربر می پرسد که فایل را حذف کند یا نه ؟؟ اگر کاربر **Y** یا **ن** جواب دهد برنامه همانطور که کاربر انتظار دارد پیش میرود. اما اگر کاربر از هرگونه کارکتر دیگر (

مانند اعداد ، حروف بزرگ، سymbل ها.....) استفاده کند. برنامه از کاربر دوباره سوال خواهد کرد. حال سوالی که مطرح میشود این است که استفاده از حلقه **d0** در اینجا چه لزومی دارد؟؟؟ در پاسخ باید گفت که در اینجا برنامه برای اینکه از کاربر سوالی پرسد نیازی به بررسی شرط حلقه ندارد ( البته قبل از کاربر پاسخی بدهد، چیزی برای بررسی وجود ندارد) و بعد اینکه کاربر به سوال مربوطه جواب داد شرط حلقه بررسی خواهد شد.



توجه: در مورد کاردن با فایل ها در فصول توضیح داده خواهد شد. هدف این مثال فقط آشنایی با حلقه `do` میباشد نه آموزش کار با فایل ها.

شما میتوانید کیفیت برنامه را حتی بهتر هم کنید چون کاربر ممکن است به بزرگ یا کوچک بودن حروف بی توجه باشد شما میتوانید شرط حلقه را به صورت زیر گسترش دهید:

```
do {  
    out.print("Delete evidence? (y/n) ");  
    reply = keyboard.findWithinHorizon(".", 0).charAt(0);  
} while (reply != 'y' && reply != 'Y' &&  
        reply != 'n' && reply != 'N');  
  
if (reply == 'y' || reply == 'Y') {
```

## ذخیره ی یک کارکتر تنها

در مثال ۳-۵ کاربر یک کلمه ای را در کیبورد تایپ می کرد و متده `keyboard.next` کلمه تایپ شده را گرفته و داخل متغیر `String` به نام `password` ذخیره میکرد. در آن مثال همه چیز به خوبی به پیش میرفت چون `String` می تواند تعداد زیادی کارکتر را در خود جاء دهد. اما در مثال بالا شما انتظار ندارید که کاربر چندین کاراکتر وارد کند، شما انتظار دارید که کاربر فقط یک حرف (۷ یا ۸) وارد نماید، به همین دلیل متغیری از نوع `char` به کار نمی برد. در عوض متغیر از نوع `String` به کار می برد، متغیر که فقط یک سимвل را در یک زمان می تواند نگه دارد. API جاوا چیزی به نام `nextChar` ندارد. پس برای خواندن چیزی که با متغیر نوع `char` هم خوانی داشته باشد باید چیزی شبیه تکه کد زیر را به کار ببرید:

```
keyboard.findWithinHorizon(".", 0).charAt(0)
```

شما میتوانید دقیقا از کد بالا هر جا که نیاز داشید تنها یک کاراکتر تنها را بخوانید، استفاده کنید.

## کار کردن با فایل ها در جاوا

جزئیات این موضوع در فصول بعدی ارائه خواهد شد، اما برای آشنایی با مطلب در اینجا توضیحاتی ارائه میشود. شما میتوانید در API جاوا یک کلاس به نام `java.io.File` پیدا کنید. عبارت :

```
File evidence = new File("cookedBooks.txt");
```

درون حافظه‌ی کامپیوتر یک شی جدید ایجاد می‌کند. این شی از کلاس `java.io.File` شکل گرفته است. این شی هر چیزی را که برنامه نیا دارد در مورد فایل `cookedBooks.txt` بداند توصیف میکند. با توجه ب این نکته متغیر `evidence` به فابل روی دیسک `cookedBooks.txt` رجوع میکند.

شی `evidence` یک مثال از کلاس `java.io.File` است که دارای متد `delete` می‌باشد. زمانی که شما `evidence.delete` را فراخوانی میکنید، کامپیوتر فایل مربوطه را پاک میکند. البته شما نمی‌توانید چیزی را که وجود ندارد را پاک کنید، برای مثال در دستور

```
File evidence = new File("cookedBooks.txt");
```

جاوا بررسی نمی‌کند که آیا فایلی به نام `cookedBooks.txt` وجود دارد یا نه؟ برای اینکه جاوا را مجبور کنید که این بررسی را انجام دهد چنین راه دارید، ساده‌ترین راه فراخوانی متد `exists` میباشد. زمانی که شما `(evidence.exists()` را فراخوانی می‌کنید. متد به فولدری که جاوا انتظار دارد `cookedBooks.txt` را پیدا کند، نگاه میکند فراخوانی `(evidence.exists())` را بر می‌گرداند اگر `cookedBooks.txt` را پیدا کند و در غیر این صورت مقدار `false` بر می‌گرداند. در اینجا نسخه‌ی دیگر از مثال ۴-۶ را باقیلیت چک کردن وجود فایل آورده می‌شود :

```

import java.io.File;
import static java.lang.System.out;
import java.util.Scanner;
public class DeleteEvidence {
    public static void main(String args[]) {
        File evidence = new File("cookedBooks.txt");
        if (evidence.exists()) {
            Scanner keyboard = new Scanner(System.in);
            char reply;
            do {
                out.print("Delete evidence? (y/n) ");
                reply =
                    keyboard.findWithinHorizon(".", 0).charAt(0);
            } while (reply != 'y' && reply != 'n');
            if (reply == 'y') {
                out.println("Okay, here goes...");
                evidence.delete();
                out.println("The evidence has been deleted.");
            } else {
                out.println("Sorry, buddy. Just asking.");
            }
            keyboard.close();
        }
    }
}

```

## محدوده‌ی متغیر‌ها

مجموعه از عبارات که درون دو آکولاد {} قرار گرفته‌اند، یک بلوک را ایجاد می‌کنند. اگر شما متغیری را داخل یک بلوک تعریف کنید نمی‌توانید خارج از آن بلوک از آن متغیر استفاده کنید. برای نمونه اگر در مثال برای نمونه اگر مثال ۴-۶ را به صورت زیر تغییر می‌دادید خطای دریافت می‌کردید:

```

do {
    out.print("Delete evidence? (y/n) ");
    char reply =
        keyboard.findWithinHorizon(".", 0).charAt(0);
} while (reply != 'y' && reply != 'n');
if (reply == 'y')

```

زیرا متغیر reply داخل بلوک مربوط به حلقه `i` اعلان گردیده و فقط در همانجا قابل استفاده است، ولی در بالا خارج از بلوک مربوطه نیز از `reply` سه مرتبه استفاده شده که این باعث ایجاد خطای خواهد گردید.

فایل آنلاین

# فَصْلٌ هُ

وَالْمُهَاجِرُونَ  
بِإِيمَانِهِمْ بِاللهِ

## تعریف کلاس

چه چیزی دو حساب بانکی را در یک بانک مشترک متمایز میکند؟؟ اگر از یک بانک دار این سوال را بپرسید، پاسخ طولانی خواهد شنید، ولی منظور اصلی ما از تفاوت حساب های بانکی در برنامه‌ی بانکداری و چگونگی برنامه نویسی برای ایجاد این تفاوت است. برای مثال احتمالاً در حساب های بانکی یک متغیر به نام balance وجود دارد، که برای حساب من دارای مقدار ۲۴,۰۲ و برای حساب شما دارای مقدار ۵۵,۶۳ می باشد. حال سوال این است: چگونه در برنامه‌ی بانکداری مابین متغیر balance حساب من، و متغیر balance حساب شما تفاوت قائل شده و آنها را از هم تشخیص دهیم.

تفاوت ایجاد دو شی (object) جداگانه است. متغیر balance اولی را داخل شی اول، و متغیر balance دوم را داخل شی دوم قرار دهید. شما همچنین میتوانید متغیر های name و address درون هریک قرار دهید، پس شما دو شی خواهید داشت، و هر شی یک حساب بانکی را نشان میدهد. حال در اینجا هر شی مثالی و نمونه‌ای از کلاس Account است.

نمونه‌ای از کلاس Account

name	Hadi
address	222 Cyberspace Lane
balance	24.02

نمونه‌ی دیگری از کلاس Account

name	Ali
address	111 Consumer Street
balance	55.63

ولی با این راحل مشکل به طور کامل حل نشده است. در واقع تا به اینجا شما دو شی در برنامه‌ی کامپیوتری خود دارید. حال شما میتوانید دو متغیر برای رجوع به این دو شی داشته باشید، فرض کنید اولین متغیر را به نام myAccount و متغیر دوم را با نام yourAccount ایجاد کنید. که اولین متغیر به شی اول (حساب بانکی من) که و دومین متغیر به شی دوم (حساب بانکی شما) اشاره میکند. برای مثال:

```
1 myAccount.balance // به balance حساب من رجوع میکند
2 myAccount.name // به name حساب من رجوع میکند
3 myAccount.balance = 24.02; // حساب من را برابر ۲۴,۰۲ قرار میدهد
4 out.println(yourAccount.name); // حساب شما را چاپ میکند
```

در بالا دو شی که مثالی از یک class به نام Account بودند آورده شده بود. ولی بر استفاده از این موضوع شما باید کلاس Account را از قبل تعریف کنید برای روشن شدن موضوع به تعریف کلاس Account در جاوا دقت نمایید:

```
public class Account {
    String name;
    String address;
    double balance;
}
```

در واقع این تعریف به برنامه میگوید که هر مثال و نمونه ای از کلاس Account ( یعنی اشیا ) سه متغیر خواهند داشت - name و address و balance - در برنامه نویسی جاوا متغیر هایی که داخل کلاس ( نه داخل ) method تعریف شده اند ، فیلد ( field ) نامیده میشوند.

## تعریف متغیر ها و تعریف اشیاء

ابتدا به مثال ۷-۲ دقیق کنید، سپس از روی این مثال به توضیح مطالب خواهیم پرداخت.

```
import static java.lang.System.out;
public class UseAccount {
    public static void main(String args[]) {
        Account myAccount;
        Account yourAccount;
        myAccount = new Account();
        yourAccount = new Account();
        myAccount.name = "HadiKhodapanah";
        myAccount.address = "222 Cyberspace Lane";
        myAccount.balance = 24.02;
        yourAccount.name = "Ali Alizadeh";
        yourAccount.address = "111 Consumer Street";
        yourAccount.balance = 55.63;
        out.print(myAccount.name);
        out.print(" (");
        out.print(myAccount.address);
        out.print(") has $");
        out.print(myAccount.balance);
        out.println();
        out.print(yourAccount.name);
        out.print(" (");
        out.print(yourAccount.address);
        out.print(") has $");
        out.print(yourAccount.balance);
    }
}
```

در اینجا دو کلاس Account و UseAccount وجود دارد که یک برنامه‌ی بالا را شکل میدهد. مثال مثال ۷-۲ کلاس UseAccount را تعریف میکند و کلاس UseAccount یک متدهای اصلی (main method) دارد. متد main متغیرهای مخصوص خود را داراست (یعنی متغیرهای myAccount و yourAccount).

دو خط اول داخل متد main (خطوط چهارم و پنجم) کمی گمراه کننده هستند به معنی این خطوط دقیق کنید: خط Account yourAccount بدین معنی است که: اگر و زمانی که من متغیر yourAccount را ایجاد میکنم، این متغیر به چیزی رجوع میکند که آن چیز مثال و نمونه‌ای از کلاس Account خواهد بود.

از لحاظ فنی زمانی که کامپیوتر کد new Account() را اجرا میکند، یک شی به وسیله‌ی فراخوانی کلاس سازنده‌ی Account ساخته میشود..

بعد از اجرای

Account yourAccount;

yourAccount

بعد از اجرای

yourAccount =  
new Account();

yourAccount

name

address

balance

زمانی که کامپیوتر کد `yourAccount = new Account()` را اجرا می کند. کامپیوتر یک شی جدید ساخته و متغیر `yourAccount` به آن شی ارجاع میکند. برای روشن شدن موضوع ، به شکل بالا رجوع کنید.

حال به خروجی حاصل از کمپایل و اجرای برنامه های ۷-۱ و ۷-۲ دقت کنید:

Hadi Khodapanah (222 Cyberspace Lane) has \$24.02

Ali Alizadeh (111 Consumer Street) has \$55.63

## کلاس های عمومی ( Public classes )

کلاس Account به صورت public است، یک کلاس public برای همه‌ی کاربران به وسیله‌ی کلاس های دیگر در دسترس است، برای نمونه کلاس UseAccount نیز به صورت public هست. وقتی که کلاسی شامل متدهای main است، برنامه نویسان جاوا تمایل دارند که کلاس را به صورت public تعریف کنند، بدون اینکه زیاد در مورد کسی که از کلاس استفاده خواهد کرد فکر کنند !!!!

زمانی که شما کلاسی را به صورت public اعلام میکنید، شما باید کلاس را در فایلی که هم نام کلاس است، معرفی و ذخیره کنید. برای مثال زمانی که شما public class MyImportantCode، شما باید کدهای کلاس را در فایلی به نام MyImportantCode.java قرار دهید. اگر کد شما دارای دو کلاس public باشد شما باید دست کم دو فایل java. را در نظر بگیرید، به عبارتی دیگر شما نمی‌توانید دو کلاس public را در یک فایل java. ذخیره کنید.

### تعریف متدهای داخل کلاس

محفویات اطلاعات موجود در دو حساب بانک را در نظر بگیرید :

بدون برنامه نویسی شیء گرا		
Name	Address	Balance
Hadi Khodapanah	222 Cyberspace Lane	24.02
Ali Alizadeh	111 Consumer Street	55.63

جدول بالا نشان میدهد که هر حساب سه چیز دارد – name ، address ، balance – این شیوه‌ی بود که قبل از اینکه برنامه نویسی شیء گرا وارد عرصه شود مورد استفاده قرار میگرفت. اما برنامه نویسی شیء گرا تغییر بزرگی را در تفکر برنامه نویسی ایجاد کرد. با برنامه نویسی شیء گرا هر حساب (account) می‌تواند یک name و یک address و یک balance داشته. و به طریقی نمایش داده شوند. به طوری که در برنامه نویسی شیء گرا هر شیء یک سری عملگر یا functionality توکار دارد. هر حساب (account) میداند که چگونه خود را نمایش دهد. یک string میتواند به شما بگوید که آیا کاراکتر‌های یکسانی با یک string دیگر دارد. به عنوان مثال یک PrintStream.out میداند که چگونه println دارد.

در برنامه نویسی شیء گرا هر شی متد های خودش را دارد. متد ها زیر برنامه های کوچکی هستند که شما میتوانید آن ها را برای انجام کارهای مورد نظرتان فراخوانی کنید.

حال یک جدول تسهیل شده را تصور کنید، در جدول جدید هر شیء یک سری functionality توکار (in) دارد.. هر account میداند که چگونه خود را روی مانیتور، نمایش دهد. هر سطر از جدول یک کپی از متد display را برای خود دارد.

جدول ۷-۲	روش شیء گرابی		
Name	Address	Balance	Display
Hadi Khodapanah	222 Cyberspace Lane	24.02	out.print....
Ali Alizadeh	111 Consumer Street	55.63	out.print....

## یک Account که خودش را نمایش میدهد

در جدول 7-2 هر شیء چهار چیز دارد: name , address , balance , -- ، روشهایی برای نمایش خودش — وقتی شما از برنامه نویسی سنتی به برنامه نویسی شیء گرا کوچ میکنید دیگر هیچ وقت به گذشته و برنامه نویسی سنتی برخواهید گشت (چون نیازی به این کار نخواهید داشت) : به مثال ۷-۳ دقت کنید :

```
import static java.lang.System.out;
public class Account {
    String name;
    String address;
    double balance;
    public void display() {
        out.print(name);
        out.print(" (" );
        out.print(address);
        out.print(" ) has $" );
        out.print(balance);
    }
}
```

## استفاده از کلاس Account ببود یافته : در مثال ۴-۷ در زیر :

```
public class UseAccount {  
    public static void main(String args[]) {  
        Account myAccount = new Account();  
        Account yourAccount = new Account();  
        myAccount.name = " HadiKhodapanah";  
        myAccount.address = "222 Cyberspace Lane";  
        myAccount.balance = 24.02;  
        yourAccount.name = "Ali Alizadeh ";  
        yourAccount.address = "111 Consumer Street";  
        yourAccount.balance = 55.63;  
        myAccount.display();  
        System.out.println();  
        yourAccount.display();  
    }  
}
```

در مثال ۴-۳ کلاس Account چهار چیز دارد — *name*, *address*, *balance*, متدهای *display()* و *toString()*. هر مثالی از کلاس Account یک name یک address و یک balance دارد. راهی که شما برای صدا زدن و فراخوانی این چیزها استفاده می‌کنید بسیار زیبا و منحصر به فرد است. برای رجوع به ذخیره شده در myAccount میتوانید چنین بنویسید:

```
myAccount.name
```

یا برای آنکه myAccount خودش را روی صفحه نمایش چاپ کند میتوانید بنویسید:

```
myAccount.display()
```

توجه: زمانی که شما یک method را فراخوانی می‌کنید، بعد از نام متدها یک جفت پارانتز قرار می‌دهید.

## فرستادن و گرفتن مقدار به Method ها :

هنگامی که شما هنگام فراخوانی متدها، مقادیری را داخل پارامتر قرار میدهید و با این کار آن مقادیر را به متدهای مورد نظر ارسال میکنید. و به این مقادیری هنگام فراخوانی متدها داخل پارامترهای متدهای قرار می دهید پارامتر (parameters) نامیده میشوند.

و به مقادیر که به شما برگشت داده میشوند مقادیر برگشتهای return value نامیده میشوند، و عموماً به انواعی که توسط متدهای برگردانده میشوند انواع برگشتهای return type گفته میشود. مثال ۷-۵ :

```
import static java.lang.System.out;
public class Account {
    String name;
    String address;
    double balance;
    public void display() {
        out.print(name);
        out.print(" (" );
        out.print(address);
        out.print(" ) has $" );
        out.print(balance);
    }
    public double getInterest(double percentageRate) {
        return balance * percentageRate / 100.00;
    }
}
```

در ادامه همین مبحث „ به مثال ۷-۶ توجه نمایید.

```

import static java.lang.System.out;
public class UseAccount {
public static void main(String args[]) {
    Account myAccount = new Account();
    Account yourAccount = new Account();
    myAccount.name = " HadiKhodapanah";
    myAccount.address = "222 Cyberspace Lane";
    myAccount.balance = 24.02;
    yourAccount.name = " AliAlizadeh";
    yourAccount.address = "111 Consumer Street";
    yourAccount.balance = 55.63;
    myAccount.display();

    out.print(" plus $");
    out.print(myAccount.getInterest(5.00));
    out.println(" interest ");
    yourAccount.display();

    double yourInterestRate = 7.00;
    out.print(" plus $");
    double yourInterestAmount =
    yourAccount.getInterest(yourInterestRate);
    out.print(yourInterestAmount);
    out.println(" interest ");
}
}

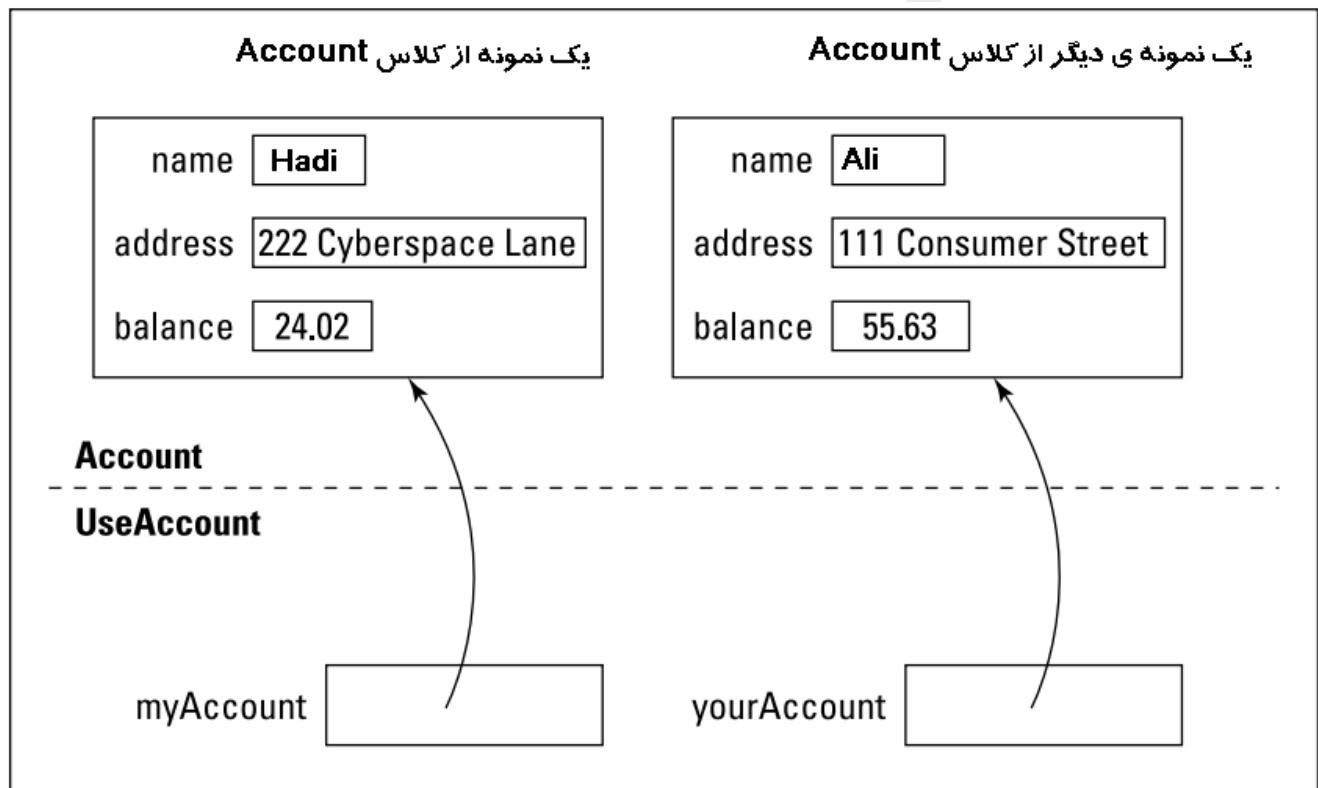
```

در مثال ۷-۵ ، کلاس Account دارای متدهای getInterest و getBalance است. این متدهای getInterest و getBalance در مثال ۷-۶ فراخوانی شده‌اند. که اولین فراخوانی لیترال ۰.۰۵ را عنوان پارامتر دارد، و دومین فراخوانی نیز متغیر yourInterestRate را به عنوان پارامتر دارد. خروجی حاصل از اجرای مثالهای ۷-۵ و ۷-۶ :

```
Hadi Khodapanah (222 Cyberspace Lane) has $24.02 plus $1.2009999999999998 interest
Ali Alizadeh (111 Consumer Street) has $55.63 plus $3.8941000000000003 interest
```

در اولین فراخوانی، نرخ interest برابر با ۰.۰۵ می‌باشد. و در دومین فراخوانی نرخ interest برابر با ۰.۰۷ می‌باشد.

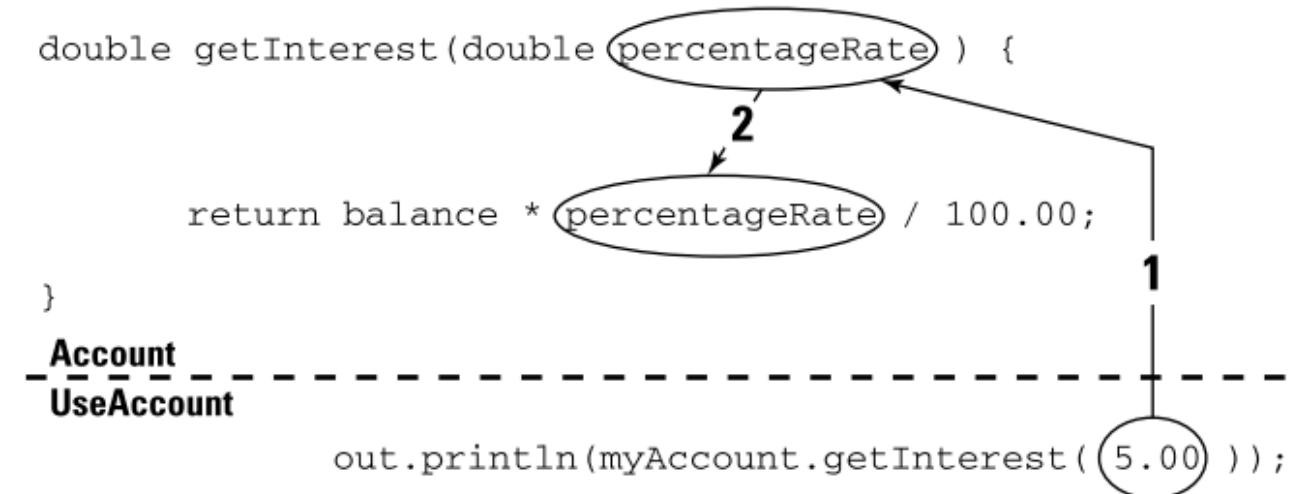
شکل زیر مطالب گفته شده را به صورت گرافیکی نشان می‌دهد.



کلمه‌ی `getInterest` نام یک متدهای است و پارامترهای آن شامل تمام چیزهایی می‌شود که شما هنگام فراخوانی متدهای دارید به آن ارسال کنید. و توجه کنید که، کلمه‌ی `double` به کامپیوتر می‌گوید که

زمانی که متدهای `getInterest` یک مقدار `double` فراخوانی شد. متدهای `getInterest` به محلی که فراخوانی شده است، بر میگردند.

در فرستادن یک مقدار به یک متدهای `getInterest` در زیر آورده شده دقت کنید.



زمانی که شما مثال های ۷-۵ و ۷-۶ را اجرا می کنید، جریان اجزا شدن برنامه از بالا به طرف پایین نیست، بلکه به این صورت است که برنامه ابتدا از `main` به `getInterest` میرود و سپس به `main` برگشته و باز به `getInterest` میرود و در نهایت به `main` بر میگردد. شکل ۷-۷ این موضوع را بهتر نشان میدهد. و اما در مورد برگشت یک مقدار از یک متدهای کاری که انجام میشود پایین شکل ۷-۸ توضیح داده شده :

```

double getInterest(double percentageRate ) {

    return balance * percentageRate / 100.00;
}

Account
-----
UseAccount

        out.println(myAccount.getInterest(5.00));

```

مثلا وقتی متد getInterest فراخوانی میشود، متد دستور return که داخل بدنه ی متد هست را اجرا میکند. ابتدا مقدار عبارت balance \* percentageRate / 100.00 را محاسبه شده و سپس دستور return اجرا میشود، کاری که دستور return انجام میدهد این است که مقدار محاسبه شده را به جایی از main که در آنجا getInterest فراخوانی شده بود بر میگرداند. دوباره به شکل بالا دقต کنید.

```

public class Account {
    Yada, yada, yada...

    double getInterest(double percentageRate) {
        return balance * percentageRate / 100.00;
    }
}

public class UseAccount {
    public static void main(String args[]) {
        Account myAccount = new Account();
        Account yourAccount = new Account();
        1
        myAccount.name = "Hadi Khodapanah";
        myAccount.address = "222 Cyberspace Lane";
        myAccount.balance = 24.02;
        4
        yourAccount.name = "Ali Alizadeh";
        yourAccount.address = "111 Consumer Street";
        yourAccount.balance = 55.63;

        myAccount.display();

        out.print(" plus $");
        2
        out.print( myAccount.getInterest(5.00) );
        3
        out.println(" interest ");
        yourAccount.display();

        double yourInterestRate = 7.00;
        out.print(" plus $");
        double yourInterestAmount =
        5
        [ yourAccount.getInterest(yourInterestRate) ];
        out.print(yourInterestAmount);
        out.println(" interest ");
    }
}

```

The diagram illustrates the execution flow between the `Account` and `UseAccount` classes. It shows the following sequence of events:

- Line 1:** The `getInterest` method is called from the `UseAccount` class.
- Line 2:** The `getInterest` method returns its value to the `UseAccount` class.
- Line 3:** The `getInterest` method is called again from the `UseAccount` class.
- Line 4:** The `getInterest` method returns its value to the `UseAccount` class.
- Line 5:** The `getInterest` method is called again from the `UseAccount` class.

دوباره به خروجی حاصل از اجرای مثال های ۷-۵ و ۷-۶ نگاه کنید: مقدار interest حساب من فقط \$1,۲۰,۹۹۹۹۹۹۹۹۹۹۸ میباشد. در حالی که این مقدار ممکن است ۱,۲۰۱ میشود، واضح است که خطای رخ داده است. این خطابه این خاطر ایجاد شده است که کامپیوتر از صفر و یک استفاده میکند و فضای بینهایت برای انجام محاسبات در اختیار ندارد. سریع ترین راحل برای به دست آوردن اعدادی بدون این گونه خطاهای استفاده از روش و طریقه‌ی حساس تری است. شما می‌توانید اعداد را گرد کنید و فقط دو رقم بعد از ممیز را نشان دهید، یا اینکه از برخی ابزارهای موجود در Application Programming API (API) می‌توانید استفاده کنید مثال ۷-۷ موضوع گفته شده را روشن تر می‌سازد:

```

import static java.lang.System.out;
public class UseAccount {
public static void main(String args[]) {
    Account myAccount = new Account();
    Account yourAccount = new Account();
myAccount.balance = 24.02;
yourAccount.balance = 55.63;
double myInterest = myAccount.getInterest(5.00);
double yourInterest = yourAccount.getInterest(7.00);
out.printf("$%4.2f\n", myInterest);
out.printf("$%5.2f\n", myInterest);
out.printf("$%.2f\n", myInterest);
out.printf("$%3.2f\n", myInterest);
out.printf("%.2f %.2f", myInterest, yourInterest);
}
}

```

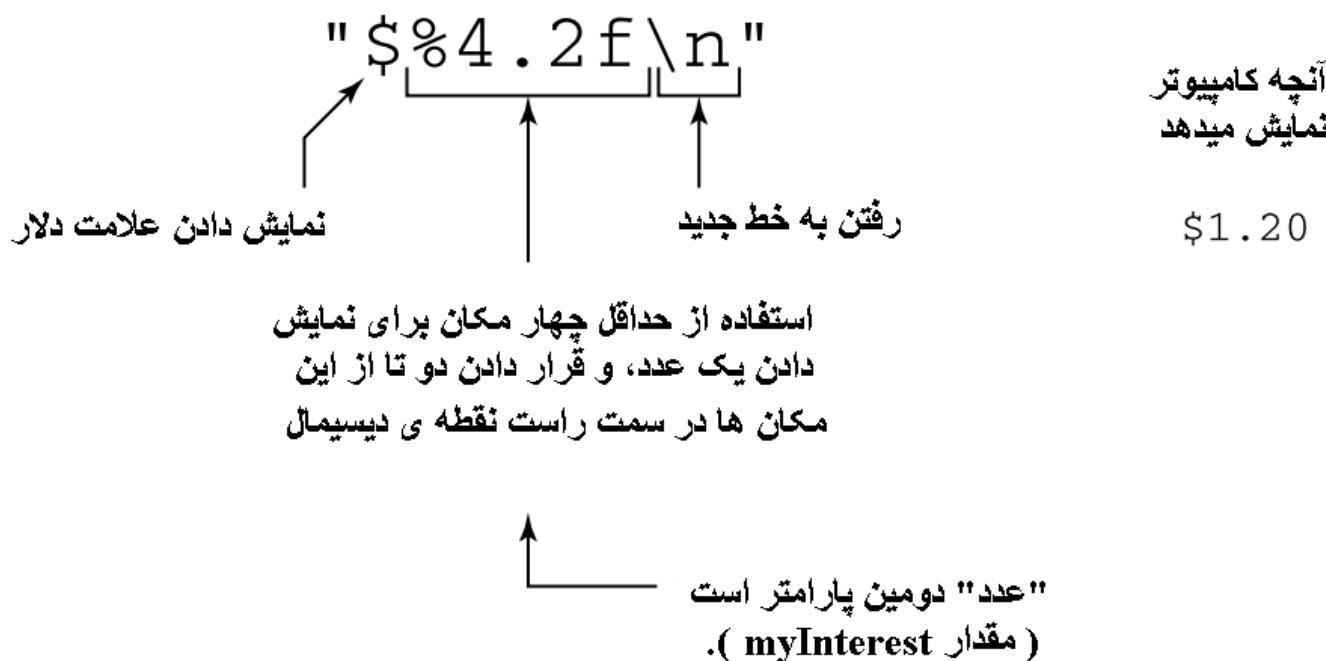
\$1.20  
 \$ 1.20  
 \$1.20  
 \$1.20  
 \$1.20 \$3.89

خروجی :

در مثال بال شما از متده استفاده کردید، وقتی که شما `printf` را فراخوانی میکنید، همیشه باید حداقل دو پارامتر داخل پرانتزهای آن قرار دهید. پارامتر اول فرمت رشته (`format string`) است. فرمت رشته دقیقا چگونگی نمایش پارامتر های دیگر را تعیین میکند. همه ی پارامتر های دیگر (بعد از اولین پارامتر) مقادیری هستند که نمایش داده میشوند.

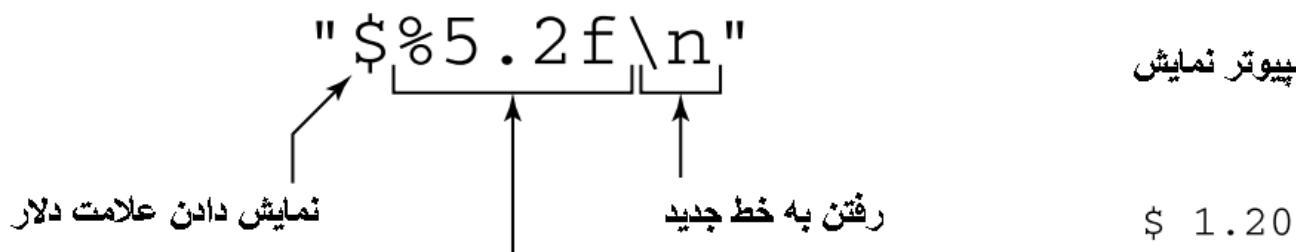
به آخرین فرآخوانی printf دا مثال ۷-۷ دقیق است. اولین پارامتر دارای دو مکان نگه دارنده myInterest placeholder (placeholder) نمایش دادن را توصیف میکند. و دومین yourInterest placeholder (placeholder) نیز نمایش دادن را توصیف میکند. برای اینکه بفهمید این فرمت رشته ها دقیقاً چگونه کار میکنند به شکل ۷-۱۰ تا ۷-۱۴ دقیق است:

: format string استفاده از



اضافه کردن فضای اضافی برای نمایش مقدار :

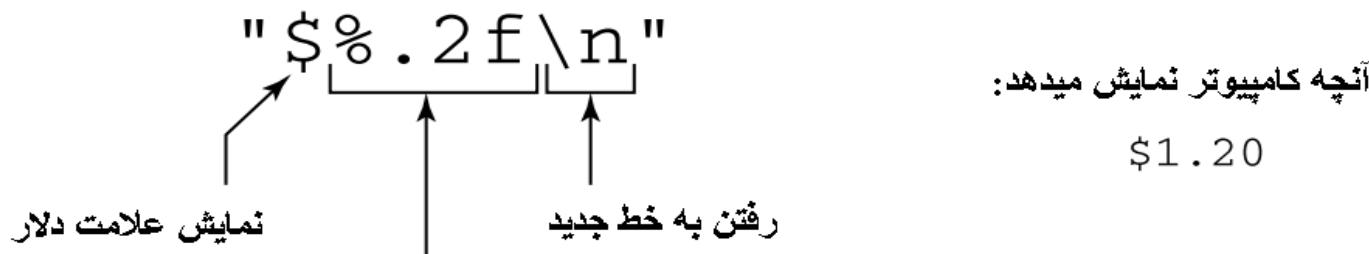
آنچه کامپیوتر نمایش میدهد:



استفاده از حداقل پنج مکان برای یک عدد، و قرار دادن دو تا از این مکان ها در سمت راست نقطهٔ دیسیمال

چونکه myInterest تنها چهار مکان اشغال میکند، myInterest با یک فضای خالی نمایش داده میشود

نمایش دادن یک مقدار، بدون مشخص کردن مقدار دقیق مکان های تخصیص داده شده به آن :



استفاده از مکان های زیاد به طور مناسب و مقتضی، برای نمایش دادن یک عدد، و قرار دادن دو تا از این مکان ها در سمت راست نقطهٔ دیسیمال

آنچه کامپیوتر نمایش میدهد:

اختصاص دادن دو مکان کمتر، برای نمایش یک مقدار :

آنچه کامپیوتر نمایش میدهد :

\$1.20

نمایش دادن علامت دلار

رفن به خط جدید

استفاده از حداقل چهار مکان برای  
نمایش یک عدد، و قرار دادن دو تا  
از این مکان ها در سمت راست  
علامت دیسیمال

کافی نیست، پس کامپیوتر از چهار  
مکان استفاده میکند

نمایش دادن بیش از یک مقدار :

آنچه کامپیوتر نمایش میدهد :

\$1.20    \$3.89

نمایش دادن  
علامت دلار

نمایش دادن مقدار دومین  
پارامتر(*myInterest*) با دو تا  
از این مکان ها در سمت راست  
 نقطه ی دیسیمال

نمایش دادن مقدار پارامتر سوم  
(*yourInterest*) با دو تا از  
این مکان ها در سمت راست  
 نقطه ی دیسیمال

نمایش دادن یک مکان خالی  
و یک علامت دلار

توجه داشته باشید که یک فراخوانی `printf` شیوه ای که اعداد برای محاسبات ذخیره شده اند را تغییر نمیدهد، کاری که در تصاویر بالا آمده است، این است که دسته ای از کاراکتر ها عددی را را زیبا تر(`nice-looking`) کنیم، تا بتوانند رو مانیتور نمایش داده شوند.

## برنامه نویسی خوب (Good programming):

فرض کنید شما در حال نوشتن کد برای کلاس `UseAccount` هستید. آیا شما بدون نوشتن `yourAccount.balance` یا `myAccount.name` چیزی به نام متدهای دسترسی دارنده (`accessor methods`) میباشد. مثال های ۷-۸ و ۷-۹ این متدها را شرح میدهند:

مخفی کردن فیلد ها:

```
public class Account {  
    private String name;  
    private String address;  
    private double balance;  
    public void setName(String n) {  
        name = n;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setAddress(String a) {  
        address = a;  
    }  
    public String getAddress() {  
        return address;  
    }  
    public void setBalance(double b) {  
        balance = b;  
    }  
    public double getBalance() {  
        return balance;  
    }  
}
```

## فراخوانی : Accessor Methods

به نظر میرسد حاصل اجرای مثال های ۷-۸ و ۷-۹ که هیچ تفاوتی با اجرای مثال های ۷-۱ و ۷-۲ ندارد. حاصل اجرای هر دو برنامه یکی ست و تفاوت بزرگ این است که در مثال ۷-۸ کلاس Account به دقت اجرای فیلدهای name و address و balance را کنترل می کند.

دوباره به مثال ۷-۸ رجوع کرده و به متدهای setName و getName توجه کنید که عبارت تخصیص داخل متدهای set و get را درون یک if قرار دهیم :

```
1 public void setName(String n) {
2     if (!n.equals("")) {
3         name = n;
4     }
5 }
```

حال برنامه نویس متصدی نوشتن کلاس UseAccount کدی مانند (") myAccount.setName(") را مینویسد، فراخوانی setName هیچ تأثیر با نتیجه ای ندارد ندارد. از این گذشته، چون فیلد name یک فیلد خصوصی (private) است، عبارت زیر در کلاس UseAccount مجاز نیست::

```
import static java.lang.System.out;
public class UseAccount {
public static void main(String args[]) {
    Account myAccount = new Account();
    Account yourAccount = new Account();
    myAccount.setName("HadiKhodapanah");
    myAccount.setAddress("222 Cyberspace Lane");
    myAccount.setBalance(24.02);
    yourAccount.setName("Ali Alizadeh");
    yourAccount.setAddress("111 Consumer Street");
    yourAccount.setBalance(55.63);
    out.print(myAccount.getName());
    out.print(" ");
    out.print(myAccount.getAddress());
    out.print(" has $");
    out.print(myAccount.getBalance());
    out.println();
    out.print(yourAccount.getName());
    out.print(" ");
    out.print(yourAccount.getAddress());
    out.print(" has $");
    out.print(yourAccount.getBalance());
}
```

البته فراخوانی هایی همچون myAccount.setName("Joe Schmoe") هنوز کار میکنند زیرا "Joe Schmoe" برابر با رشته‌ی خالی "" نیست. با استفاده از فیلد‌های خصوصی(private) و متدهای accessor شما قادر خواهید بود از اینکه کسی یک رشته‌ی خالی برای فیلد name تخصیص دهد جلوگیری کنید. یا حتی میتوانید با استفاده از دستورات if بیشتر، قواعد بیشتری را روی آنچه کاربر وارد میکند اعمال کنید.

# فصل ٤

رواية  
البناء

با فکر کردن درباره ی داده ها(data) ، شما شروع به نوشتمن برنامه ای شنگرا میکنید. وقتی درباره ی حساب ها(accounts) کدی مینویسید، سوالی مطرح میشود که: اساسا یک account چیست؟؟ شما کدی برای مدیریت کلیک روی دکمه ها(button) مینویسید، سوال این است که : دکمه چیست؟؟ زمانی که شما برنامه ای در مورد فرستادن لیست حقوق و دستمزد به کارکنان(employees) مینویسید، کارمند در برنامه چیست؟؟

در مثال ۱-۸ یک کارمند کسی هست با یک نام و یک عنوان شغلی. یقینا یک کارمند مشخصات و خصوصیات دیگری هم دارد، ولی ما برای ساده شدن کار از آنها صرف نظر میکنیم، مثال ۱-۸ معین میکند که یک کارمند بودن چه معنی ای میدهد:

```
import static java.lang.System.out;
public class Employee {
    private String name;
    private String jobTitle;
    public void setName(String nameIn) {
        name = nameIn;
    }
    public String getName() {
        return name;
    }
    public void setJobTitle(String jobTitleIn) {
        jobTitle = jobTitleIn;
    }
    public String getJobTitle() {
        return jobTitle;
    }
    public void cutCheck(double amountPaid) {
        out.printf("Pay to the order of %s ", name);
        out.printf("(%s) ***$ ", jobTitle);
        out.printf("%,.2f\n", amountPaid);
    }
}
```

هر کارمند(employee) خصیصه هایی دارد، که دو تا از این خصیصه ها بدون تردید بسیار ساده اند. هر کارمند یک نام(name) و یک عنوان شغلی(jobTitle) دارد. در مثال ۱-۸ کلاس Employee یک فیلد name و یک فیلد jobTitle دارد.

یک کارمند (employee) چهار متده برای مدیریت مقادیر نام کارمند و عنوان شغلی دارد، که این متدها عبارت اند از : getName و setName و getJobTitle و setJobTitle .

بعلاوه هر کارمند (employee) یک متده cutCheck دارد. مقصود این است که متده که حقوق و دستمزد کارمندان را مینویسد بایستی به یک کلاس تعلق داشته باشد. دلیل این امر آن است که ، بیشتر اطلاعاتی که داخل لیست حقوق و دستمزد نوشته میشود، برای هر کارمند متفاوت است. جزئیات این مثال در ادامه توضیح داده خواهد شد.

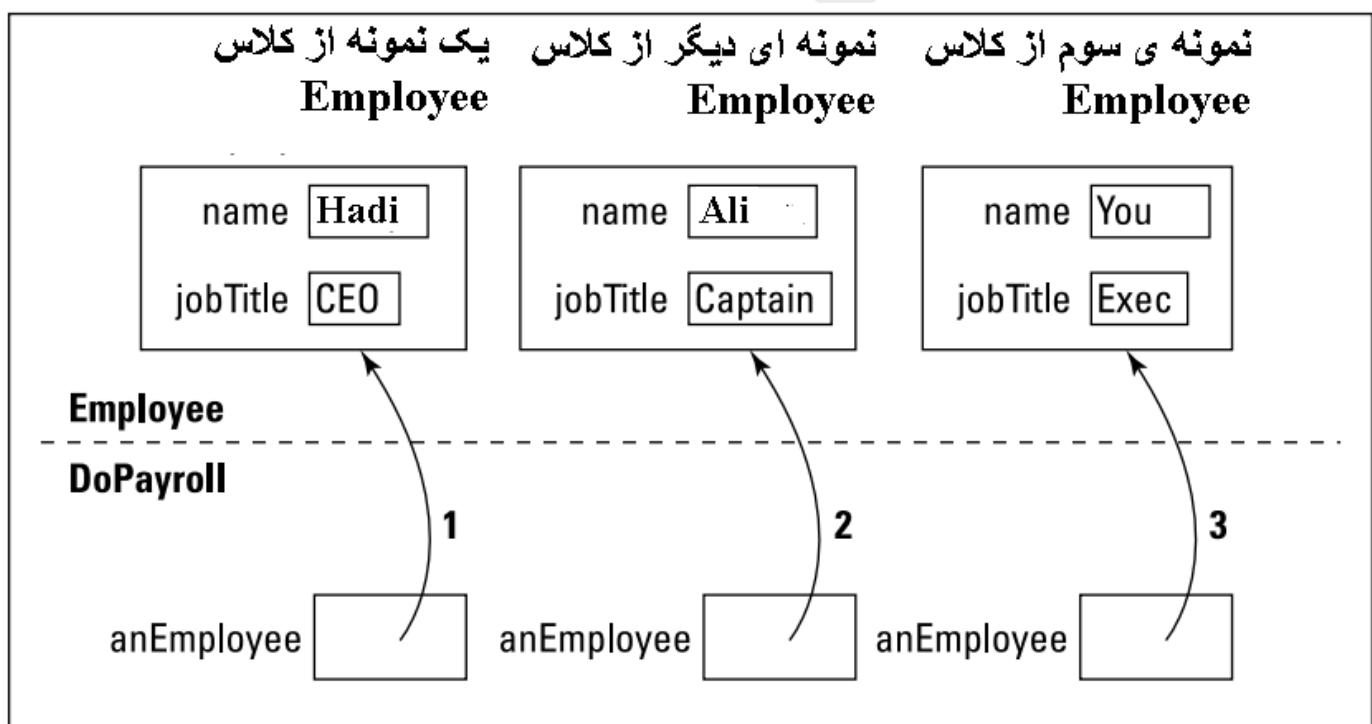
کلاس Employee در مثال ۱-۸ هیچ متده main ای نداشت. پس هیچ نقطه‌ی شروعی برای اجرای کد وجود ندارد. برای برطرف کردن این نقص برنامه نویس یک برنامه‌ی جداگانه با یک متده main مینویسد، و از آن برنامه برای ایجاد یک شیء نمونه از Employee استفاده میکند. مثال ۲-۸ یک کلاس دارای متده main را نشان میدهد.

```
importjava.util.Scanner;
importjava.io.File;
importjava.io.IOException;
public class DoPayroll {
    public static void main(String args[])
        throws IOException {
        Scanner diskScanner =
        new Scanner(new File("EmployeeInfo.txt"));
        for (int empNum = 1; empNum <= 3; empNum++) {
            payOneEmployee(diskScanner);
        }
        diskScanner.close();
    }
    static void payOneEmployee(Scanner aScanner) {
        Employee anEmployee = new Employee();
        anEmployee.setName(aScanner.nextLine());
        anEmployee.setJobTitle(aScanner.nextLine());
        anEmployee.cutCheck(aScanner.nextDouble());
        aScanner.nextLine();
    }
}
```

برای اجرا کردن مثال ۸-۲ هارد دیسک شما باستی دارای فایلی به نام EmployeeInfo.txt داشته باشد.

کلاس DoPayroll در مثال ۸-۲ دارای دو متده است. یکی از متدها (main) متده دیگر (payOneEmployee) را سه بار فراخوانی میکند. در هر بار متده payOneEmployee یک سری داده‌ی اولیه از EmployeeInfo.txt دریافت کرده و آن‌ها را به متدهای کلاس Employee میفرستد.

در زمان اولین فراخوانی anEmployee = new Employee() عبارت payOneEmployee باعث میشود که anEmployee به شی جدیدی منسوب شده و به آن اشاره کند. زمانی که payOneEmployee برای دومین بار فراخوانی میشود، کامپیوتر همان عبارت را دوباره اجرا میکند. این اجرای دوباره باعث ایجاد یک تجسم خارجی جدیدی از متغیر anEmployee شده، که به شی کاملاً جدیدی منسوب و به آن شی اشاره میکند. در اجرای سوم همه چیز مثل قبل دوباره تکرار میشود. یک متغیر anEmployee جدید به شی سوم منسوب شده و به آن رجوع میکند. ممکن است این موضوع ممکن است کمی گیج کننده به نظر برسد ولی شکل ۸-۱ همه چیز به خوبی روشن میسازد:



مثال ۸-۱ سه فراخوانی printf دارد. هر printf فرمات رشته‌ای (format string) خود را صدا میزند (مانند "%.\*s") و یک متغیر (مانند jobTitle) . هر فرمات رشته‌ای یک نگه دارنده مکان (placeholder) مانند %s دارد که مشخص میکند که کجا و چگونه مقادی ر متغیرها نمایش داده شوند.

برای مثال دومین فراخوانی `printf` ، فرمات رشته دارای یک نگه دارنده ی جاء `%s` میباشد. این `%s`

```
Pay to the order of Hadi Khodapanah (CEO) ***$5,000.00  
Pay to the order of Ali Alizadeh (Captain) ***$7,000.00  
Pay to the order of Your Name Here (Honorary Exec of the Day) ***$10,000.00
```

مکانی را برای مقدار متغیر `jobTitle` نگه میدارد. در مثال ۱-۸ بر اساس قواعد جاوا، نماد `%s` همیشه مکان ای را برای رشته نگه میدارد و میتوان بقین کرد که متغیر `String jobTitle` از نوع `String` اعلام شده است. پارانتز ها در اطراف هر عنوان شغلی در خروجی برنامه وجود دارد. شکل ۱-۸ :

در مثال ۱-۸ به ویرگول داخل نگه دارنده ی مکان `%,20.` توجه کنید. ویرگول به برنامه می گوید که از جداکننده های گروه بند (grouping separators) استفاده کند. به همین دلیل است که شما در شکل ۱-۸ `$5,000,00` و `$7,000,00` را به جای `$5000.00` و `$7000.00` میبینید.

جداکننده های گروه بند در کشور های مختلف متفاوت اند. برای مثال در فرانسه برای نوشتن یک هزار (مايل) شما `,000,000` را مینویسید. جاوا میتوانید به طور اتوماتیک اعداد شما را فرانسوی کند البته با کمک عباراتی همچون :

```
out.print(new,(java.util.Formatter().format(java.util.Locale.FRANCE,"%,.2f", 1000.00 ))
```

برای جزئیات بیشتر به (Application Programming Interface API) رجوع کنید. `Locale` و `Formatter`

## ذخیره کردن داده ها در فایل

کد های در مثال ۱-۸ اجرا نمیشوند مگر آنکه شما بعضی اطلاعات کارمند (`employee`) را داخل یک فایل داشته باشید. مثال ۱-۸ میگوید این فایل `EmployeeInfo.txt` نام دارد. پس قبل از اجرای مثال ۱-۸ یک فایل کوچک `EmployeeInfo.txt` ایجاد کنید. شکل ۱-۹ :

```
Hadi Khodapanah  
CEO  
5000.00  
Ali Alizadeh  
Captain  
7000.00  
Your Name Here  
Honorary Exec of the Day  
10000.00
```

در مثال ۱-۹، من بر این نکته تکیه کرده ام که، زمانی که شما کاراکتر ها را در (مانند) شکل ۱-۹ تایپ میکنید، شما فایل را با تایپ کردن `.....1` به پایان میبرید.

و سپس Enter را فشار میدهید) دوباره به شکل ۸-۳ نگاه کنید چگونه مکان نما در ابتدای خط جدید قرار دارد). اگر شما فشار دادن Enter را فراموش کنید، آنگاه زمانی که تلاش کنید مثال ۸-۲ را اجرا کنید با مشکلاتی مواجه خواهید شد.

گروه بندی جدا کننده ها ، در کشور های مختلف، متفاوت است. فایل نشان داده شده در شکل ۸-۳ روی یک کامپیوتری که در کشور ایالات متحده پیکره بندی شده است، کار میکند. جایی که ۵۰۰۰،۰۰۰ پنج هزار است. ولی در بعضی کشورها که ۵۰۰۰،۰۰۰ پنج هزار است، به درستی کار نخواهد کرد). دقیت کنید که در عدد اول نقطه . ما بین صفر ها قرار گرفته درحالی که در عدد دوم ویرگول).

اگر شما در چنین کشوری زندگی میکنید. و از فایلی که در شکل ۸-۳ نشان داده شده است استفاده میکنید، زمانی که سعی در اجرای مثال این بخش داشته باشید احتمالا یک پیام خطا دریافت خواهید کرد) یک InputMismatchException (برای حل این مشکل اعداد موجود در فایل داده را ، به فرمتی رایج در کشور خود تغییر دهید ، تا خطا مورد بحث را رفع نمایید.

## Copy و Past کردن کد :

تقریبا در بیشتر زبان های برنامه نویسی نوشتن و خواندن داده ها روی فایل می تواند دشوار و نیازمند مهارت باشد. شما خطوط کد اضافی را به برنامه اضافه میکنید تا به کامپیوتر بگویید چه کاری را انجام دهد. در برخی مواقع شما میتوانید این کد ها از کدهای برنامه های دیگر مردم کپی کنید. برای مثال میتوانید از الگوی ارائه شده در مثال ۸-۲ پیروی کنید :

شما میخواهید از روی فایل اطلاعاتی را بخوانید. با تصور اینکه دارید از روی کیبورد اطلاعاتی را میخوانید شروع کنید. کد دستورات Scanner ها و نیز کد دستورات next معمول را داخل برنامه ی خود قرار دهید. سپس آیتم های اضافی را به الگوی مثال ۸-۲ اضافه کنید:

دو اعلان import جدید اضافه کنید. یکی برای java.io.IOException و دیگری برای java.io.File

در هدر(header) متده خود ، throws IOException را تایپ کنید.

```

/*
* الگوی استفاده شده در مثال 8-2
*/
importjava.util.Scanner;
importjava.io.File;
importjava.io.IOException;
classSomeClassName {
public static void main(String args[])
throwsIOException {
    Scanner scannerName =
new Scanner(new File("SomeFileName"));
    محل قرار دادن قسمتی از کد ها //
scannerName.nextInt();
scannerName.nextDouble();
scannerName.next();
scannerName.nextLine();
    محل قرار دادن قسمت دیگری از کدها //
scannerName.close();
}
1

```

در فراخوانی new Scanner عبارت new File("") را تایپ کنید.

فایلی که هم اکنون داخل کامپیوتر شما وجود دارد را در نظر بگیرید و نام آن را داخل علامت های کوتیشن بنویسید.

کلمه ای که شما برای اسم اسکنر scanner ( ) تان به کار بردید را به خاطر داشته باشید. از همان کلمه برای فراخوانی next و nextDouble و nextInt استفاده کنید..در ضمن از همان کلمه برای فراخوانی close استفاده کنید.

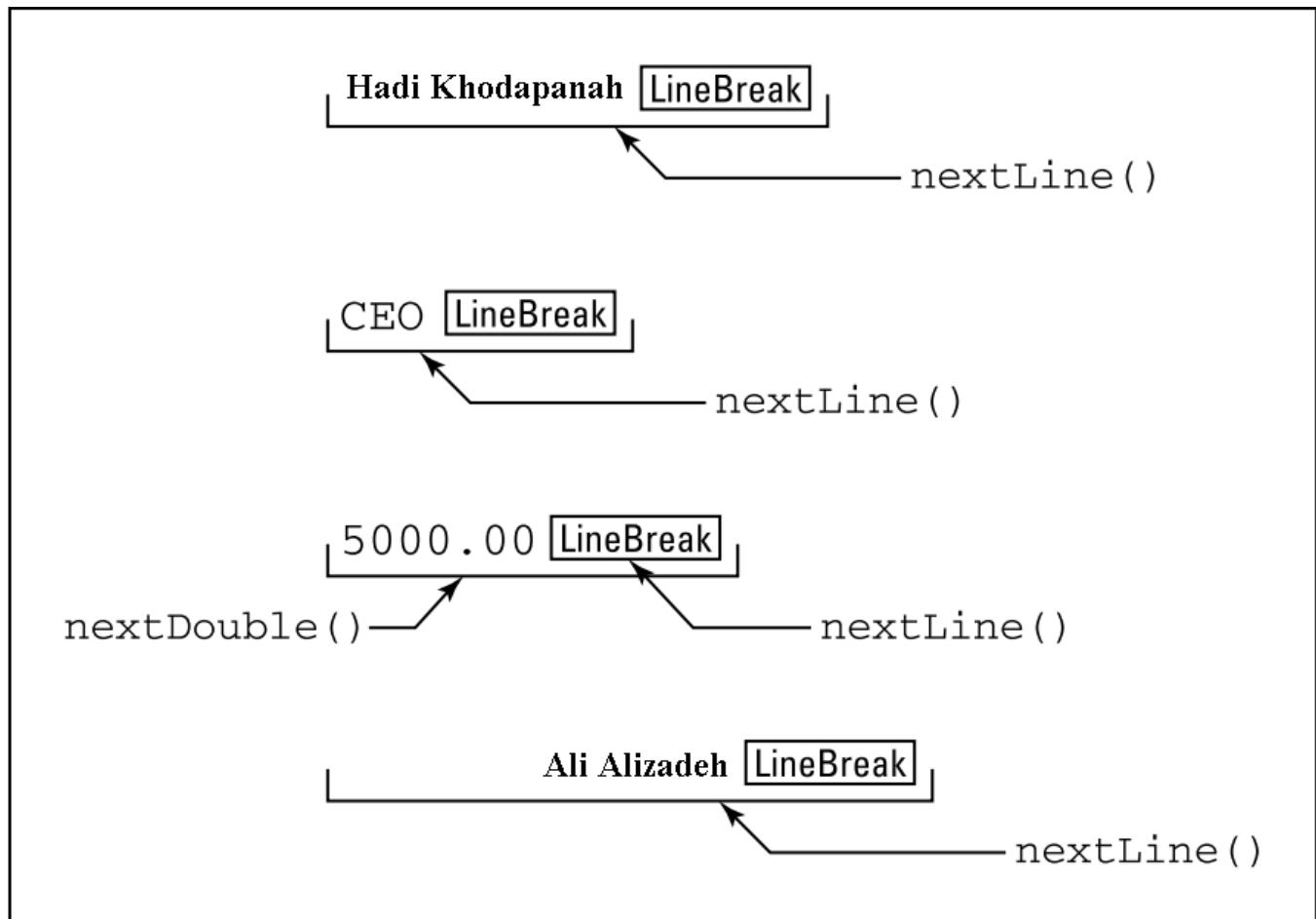
البته بعضی اوقات، کپی کردن کدها و جاگذاری آن ها در برنامه‌ی خودتان میتواند شما را به دردسر بیاندازد.شاید شما دارید برنامه‌ای میتویسید که با الگوی مثال ۲-۸ مطابقت ندارد و شما نیاز دارید که کمی در الگوی برنانه را دستکاری کرده و تغییر دهید.اما برای تغییر الگو شما باید ایده‌ی پشت الگو را دقیقاً درک کرده باشید.

## خواندن یک خط در هر دفعه

در مثال ۸-۲ متدهای `nextLine` و `next` را توضیح میدهد. در عمل، هر اسکنری که شما ایجاد کرده اید یک متدهای `nextLine` دارد. شما ممکن است از این متدهای `nextLine` استفاده نکنید، ولی به هر حال این متدهای در دسترس است. زمانی که شما متدهای `nextLine` اسکنر را فراخوانی میکنید، این متدهای تا تمام شدن خط کنونی، هر چیزی را گرفته و نگه میدارند. در مثال ۸-۲ یه فراخوانی به `nextLine` میتواند کل یک خط را از فایل `EmployeeInfo.txt` بخواند. (در برنامه‌ی دیگر، فراخوانی `nextLine` اسکنر، امکان دارد که هر چیزی که کاریر روی کیبورد تایپ میکند را تا فشار دادن دکمه‌ی `Enter` بخواند). دوباره به کلمه‌ی `nextLine` دقت کنید: هر چیزی را تا انتهای خط کنونی میخواند. متأسفانه آن چیزی که خواندن تا انتهای خط کنونی همیشه آن چیزی که شما فکر میکنید، معنی نمی‌دهد.

با آمیختن فراخوانی‌های `nextInt` و `nextDouble` و `nextLine` ممکن است برنامه کمی شلوغ و آشفته به نظر آید، پس شما باید آن چیزی که در برنامه انجام میدهید را زیر نظر داشته باشید. و خروجی برنامه را به دقت چک کنید.

برای فهمیدن همه‌ی اینها، شما باید از چگونگی جدا شدن خطوط فایل داده آگاهی داشته باشید. جدا کننده‌ی یک خط از یک خط دیگر را به عنوان یک کاراتر اضافی در نظر بگیرید که بین دو خط جدا قرار میگیرد. در این صورت فراخوانی `nextLine` به معنی خواندن هر چیزی در خط کنونی فایل داده، تا انتهای خط (یعنی تا رسیدن به کاراکتر جداکننده است) میباشد.



اگر فراخوانی `nextLine` را بخواند فراخوانی `Hadi Khodapanah[LineBreak]`، `nextLine` بعدی اگر فراخوانی `CEO[LineBreak]` را خواهد خواند.

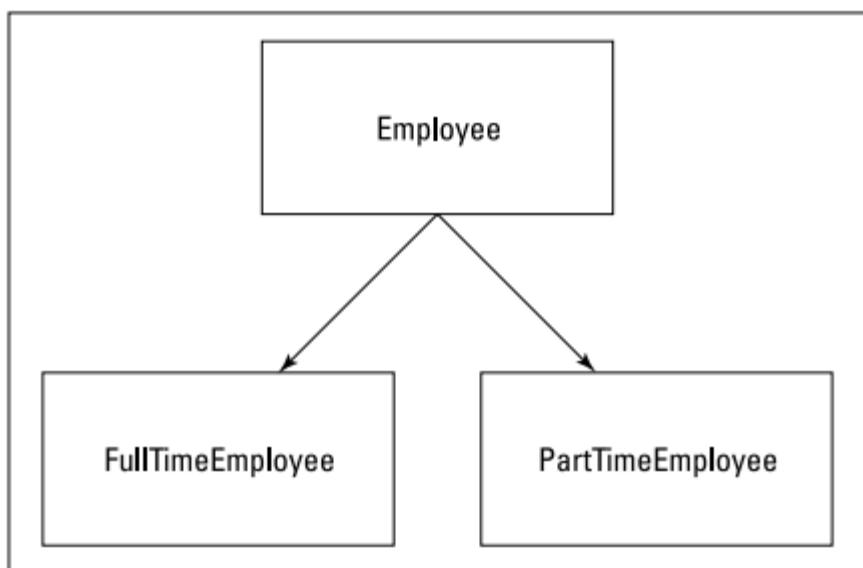
اگر فراخوانی `nextDouble` عدد `5000.00` را بخواند، فراخوانی بعدی `nextLine` را خواهد خواند.

اگر فراخوانی `nextLine`، `nextLine[LineBreak]` را بخواند. بعد از عدد `5000.00` فراخوانی بعدی اگر فراخوانی `Ali Alizadeh[LineBreak]` را خواهد خواند.

پس بعد از خواندن عدد ۵ شما به دو فراخوانی `nextLine`، به منظور جمع کردن نام Ali Alizadeh نیاز دارید، اشتباهی که معمولاً دانشجویان مرتكب می‌شوند این است که اولین فراخوانی از این دو فراخوانی را فراموش می‌کنند.

دوباره به شکل ۳-۸ نگاهی بیاندازید، برای مثال به درستی کار کند شما نیاز دارید که بعد از آخرین یک جدا کننده‌ی خطوط (line break) داشته باشید. از آخرین فراخوانی nextLine را انجام ندهید ممکن است برنامه‌ی شما خراب شده و پیام خطایی به مضمون NoSuchElementException: No line found دریافت نماید.

توجه به این نکته الزامی است که ، اولین nextLine که از فایل شکل ۸-۳ ، Hadi Khodapanah[LineBreak] را خوانده ، اما همین فرآخوانی Hadi Khodapanah را بدون هیچگونه line break به کد اجرایی تحویل میدهد. بنابراین nextLine در جستجوی یک line break بوده و پس از پیدا کردن ، آن از دست میدهد. توجه نکردن به این نکته میتواند برنامه شما را با مشکلات جدی مواجه کند.



## ایجاد یک زیر کلاس

در مثال ۱-۸ یک کلاس Employee تعریف شده است. شما میتوانید از آنچه که در مثال ۱-۸ تعریف شده استفاده کرده و یا آن را گسترش دهید. با این کار میتوانید کلاس های جدیدتر و مخصوص تری را ایجاد کنید. بر این اساس در مثال ۱-۳ تصمیم گرفتن یک کلاس جدید به اسم FullTimeEmployee تعریف کنم:

```
public class FullTimeEmployee extends Employee {  
    private double weeklySalary;  
    private double benefitDeduction;  
    public void setWeeklySalary(double weeklySalaryIn) {  
        weeklySalary = weeklySalaryIn;  
    }  
    public double getWeeklySalary() {  
        return weeklySalary;  
    }  
    public void setBenefitDeduction(double benefitDedIn) {  
        benefitDeduction = benefitDedIn;  
    }  
    public double getBenefitDeduction() {  
        return benefitDeduction;  
    }  
    public double findPaymentAmount() {  
        return weeklySalary - benefitDeduction;  
    }  
}
```

به مثال ۱-۳ نگاه کنید. شما میتوانید ببینید که هر نمونه از کلاس FullTimeEmployee دارای دو فیلد benefitDeduction و weeklySalary میباشد. اما آیا آن ها تنها فیلد هایی هستند که هر نمونه از کلاس FullTimeEmployee توسعه دارد؟ نه... خط اول از برنامه ۱-۳ میگوید که کلاس FullTimeEmployee یافته‌ی کلاس Employee میباشد. این بدین معنی است که علاوه بر داشتن دو فیلد weeklySalary و jobTitle دارای دو فیلد دیگر name و benefitDeduction هر نمونه از کلاس FullTimeEmployee حاصل میشوند.

زمانی که یک کلاس بر اساس گسترش دادن یک کلاسی که از قبل موجود است ایجاد میکنید، کلاس گسترش یافته به طور اتوماتیک (وظایف اساسی کارکردی) functionality ای در کلاس موجود قبلی

تعریف شده را به ارث میبرد. بنابراین کلاس FullTimeEmployee فیلد های name و jobTitle را به ارث میبرد. کلاس FullTimeEmployee همچنین تمام متدهای در کلاس Employee اعلان گردیده اند را نیز به ارث می برد. در (setJobTitle, getJobTitle, and cutCheck, setName, getName).... واقع کلاس FullTimeEmployee بک زیر کلاس از کلاس Employee میباشد این بدان معنی است که کلاس FullTimeEmployee یک سوپر کلاس (superclass) برای کلاس Employee میباشد. شما همچنین میتوانید بگویید که کلاس FullTimeEmployee فرزند کلاس Employee میباشد و کلاس Employee والد کلاس FullTimeEmployee است. مثال ۴-۸ تقلیبی و جعلی است اما میتواند آموزنده باشد.

ولی چرا کد مثال ۴-۸ تقلیبی و معرفی شد. در واقع تفاوت اصلی مابین مثال ۴-۸ و وضعیت اراثت در مثال های ۱ و ۳-۸ این است که : کلاس فرزند نمیتواند مستقیم به فیلد های خصوصی کلاس والد رجوع داشته باشد. برای اینکه کلاس فرزند به فیلد های خصوصی کلاس والد دسترسی داشته باشد، کلاس فرزند بایستی ابتدا متدهای دسترسی دارنده (accessor) کلاس والد را فراخوانی کند. به مثال ۳-۸ برگردید فراخوانی name="Rufus" setName("Rufus") مجاز و قانونی است ولی دستور name="Rufus" مجاز نخواهد بود.

نکته : شما به فایل Employee.java روی هارد خود برای نوشتن کلاسی که توسعه یافته ای باشد نیاز ندارید. تمام نیاز شما فایل Employee.class می باشد.

```

import static java.lang.System.out;
public class FullTimeEmployee {
    private String name;
    private String jobTitle;
    private double weeklySalary;
    private double benefitDeduction;
    public void setName(String nameIn) {
        name = nameIn;
    }
    public String getName() {
        return name;
    }
    public void setJobTitle(String jobTitleIn) {
        jobTitle = jobTitleIn;
    }
    public String getJobTitle() {
        return jobTitle;
    }
    public void setWeeklySalary(double weeklySalaryIn) {
        weeklySalary = weeklySalaryIn;
    }
    public double getWeeklySalary() {
        return weeklySalary;
    }
    public void setBenefitDeduction(double benefitDedIn) {
        benefitDeduction = benefitDedIn;
    }
    public double getBenefitDeduction() {
        return benefitDeduction;
    }
    public double findPaymentAmount() {
        return weeklySalary - benefitDeduction;
    }
    public void cutCheck(double amountPaid) {
        out.printf("Pay to the order of %s ", name);
        out.printf("(%s) ***$", jobTitle);
        out.printf("%,.2f\n", amountPaid);
    }
}

```

## ایجاد زیر کلاس ها شکل گیری عادت هاست

زمانی که شما با ایجاد زیر کلاس ها(کلاس های توسعه یافته) خو گرفتید. خلق زیر کلاس باعث ایجاد راحتی در برنامه هایی که می نویسید خواهد شد. اگر شما کلاس FullTimeEmployee را ایجاد کرده اید. پس میتوانید کلاس دیگری به اسم PartTimeEmployee را نیز ایجاد کنید که در شکل ۸-۵ ایجاد شده است.

```
public class PartTimeEmployee extends Employee {  
    private double hourlyRate;  
    public void setHourlyRate(double rateIn) {  
        hourlyRate = rateIn;  
    }  
    public double getHourlyRate() {  
        return hourlyRate;  
    }  
    public double findPaymentAmount(int hours) {  
        return hourlyRate * hours;  
    }  
}
```

بر خلاف کلاس FullTimeEmployee ، کلاس PartTimeEmployee هیچ حقوق و دستمزد(salary) و یا کسر (deduction) ندارد. در عوض PartTimeEmployee یک فیلد hourlyRate دارد. بعلاوه یک فیلد numberOfWorked که ممکن است در برنامه قرار داده شود.. البته توجه داشته باشید که تعداد ساعت کاری یک کارمند پاره وقت از یک هفته به هفته ی دیگر (بسته به زیادی کار موجود) ممکن است متفاوت باشد.

## استفاده از زیر کلاس ها

بخش های قبلی نحوه ایجاد زیر کلاس ها را توضیح می دهد، البته ای نکته ای مهم این است : ایجاد زیر کلاس ها به خودی خود مزیتی ندارد مگر آنکه شما کدی برای استفاده از آنها بنویسید. در این بخش کدی که از این زیر کلاس ها (subclasses) را توضیح خواهیم داد.

مثال ۸-۶ شما یه برنامه‌ی ساده، که از زیر کلاس های PartTimeEmployee و FullTimeEmployee استفاده می‌کند، را نشان میدهد. و شکل ۸-۶ نیز خروجی این برنامه را نشان میدهد.

```
public class DoPayrollTypeF {  
    public static void main(String args[]) {  
        FullTimeEmployee ftEmployee = new FullTimeEmployee();  
        ftEmployee.setName("HadiKhodapanah");  
        ftEmployee.setJobTitle("CEO");  
        ftEmployee.setWeeklySalary(5000.00);  
        ftEmployee.setBenefitDeduction(500.00);  
        ftEmployee.cutCheck(ftEmployee.findPaymentAmount());  
        System.out.println();  
        PartTimeEmployee ptEmployee = new PartTimeEmployee();  
        ptEmployee.setName("Ali Alizadeh");  
        ptEmployee.setJobTitle("Driver");  
        ptEmployee.setHourlyRate(7.53);  
        ptEmployee.cutCheck(ptEmployee.findPaymentAmount(10));  
    }  
}
```

Pay to the order of **Hadi Khodapanah**(CEO) \*\*\*\$4,500.00

Pay to the order of **Ali Alizadeh** (Driver) \*\*\*\$75.30

برای اینکه کاملاً مثال ۸-۶ را متوجه شوید بایستی دوباره به زیرکلاس های Employee و PartTimeEmployee و FullTimeEmployee نگاهی بیاندازید.

نیمه‌ی نخست مثال ۸-۶ به یک کارمند تمام وقت می‌پردازد. توجه کنید که چگونه متدهای زیادی با متغیر ftEmployee برای استفاده در دسترس هستند. برای مثال شما میتوانید FullTimeEmployee را فراخوانی کنیدزیرا که ftEmployee.setWeeklySalary نوع ftEmployee.setName را فراخوانی کنید، چونکه کلاس داراست. شما همچنین میتوانید ftEmployee کلاس Employee را گسترش میدهد.

برای اینکه cutCheck داخل کلاس Employee اعلان شده است شما میتوانید ftEmployee.findPaymentAmount را فراخوانی کنید. شما همچنین میتوانید ftEmployee.cutCheck را نیز فراخوانی کنید زیرا که متدهای findPaymentAmount داخل کلاس FullTimeEmployee میباشد.

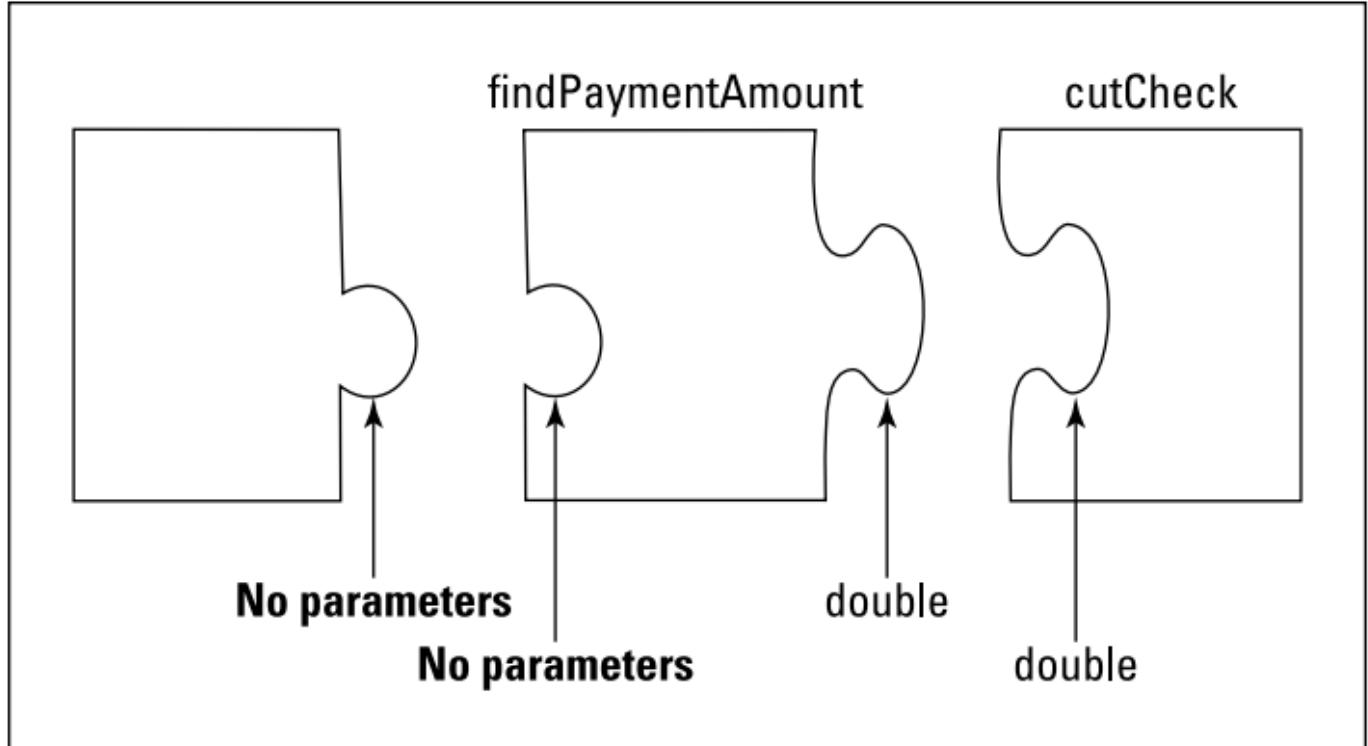
## تطابق انواع (types match)

دوباره به نیمه‌ی نخست مثال ۸-۶ نگاهی اندادته و به آخرین عبارت توجه توجه کنید:

متدهای ftEmployee.findPaymentAmount با لیست پارامتر خالی فراخوانی میشود. برای اینکه متدهای findPaymentAmount هیچ پارامتری را نمیپذیرد.(مثال ۳-۸).  
متدهای findPaymentAmount نوع داده‌ای double را برمیگرداند.

مقدار double بر میگرداند به متدهای ftEmployee.cutCheck که ای double به پاس داده میشود. الیته متدهای cutCheck یک پارامتر از نوع double میگیرد.(مثال ۱-۸).

به شکل زیر دقต کنید:



## تغییر دادن متدهای قبلی

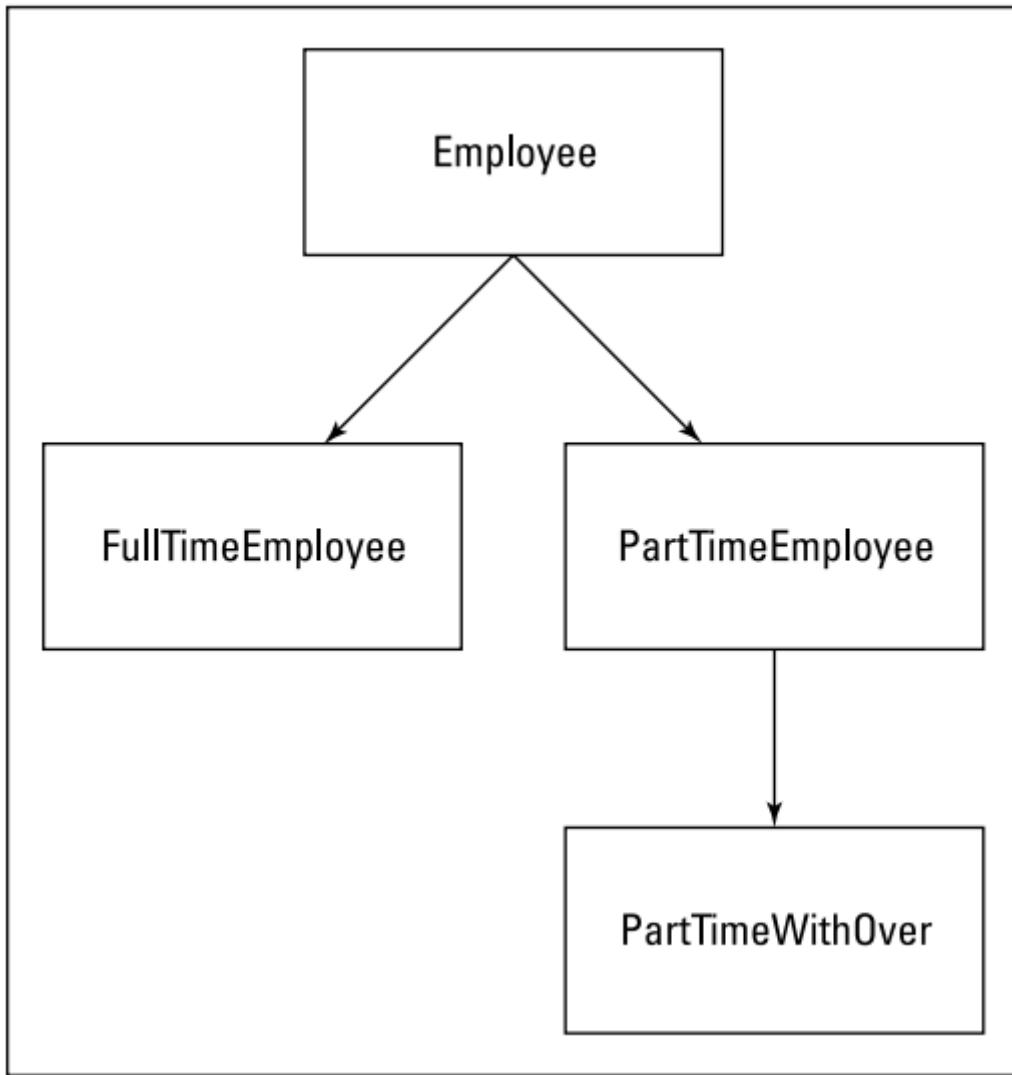
خوب ، شما میتوانید کدهای کلاس PartTimeEmployee را کند و کاو کرده ، و روی آنها کمی تغییرات اعمال کنید.( به نظر من ایده‌ی خوبی نیست).

شما میتوانید از آموزش های بخش قبلی استفاده کرده و یک زیر کلاس ، از کلاس موجود findPaymentAmount کلاس PartTimeEmployee ایجاد نمایید. اما کلاس PartTimeEmployee قبل از متدهای findPaymentAmount نیاز داریم ؟؟

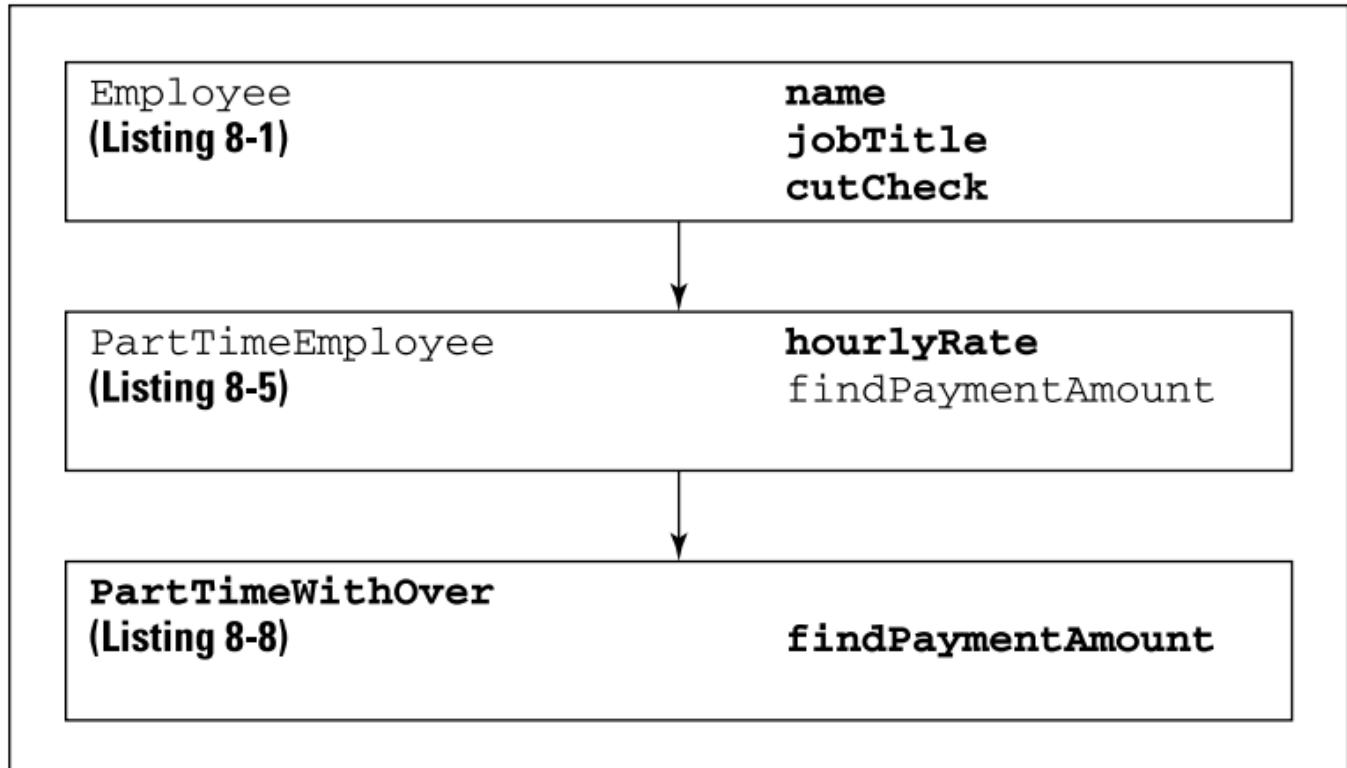
در اینجا شما خوش شانس هستید که از زبان شیءگرای جاوا استفاده میکنید. با برنامه نویسی شیءگرای شما میتوانید زیر کلاس هایی خلق نمایید که در کار کرد اساسی کلاس والد خود را دارا نیستند. مثل ۷-۸ چنین زیر کلاسی را نشان میدهد:

```
public class PartTimeWithOver extends PartTimeEmployee {
    @Override
    public double findPaymentAmount(int hours) {
        if(hours <= 40) {
            return getHourlyRate() * hours;
        } else {
            return getHourlyRate() * 40 +
                getHourlyRate() * 2 * (hours - 40);
        }
    }
}
```

شکل ۸-۸ رابطه‌ی مابین مثال ۷-۸ و سایر تکه کدهای موجود در این فصل را نشان میدهد. اگر دقیق شویم PartTimeWithOver یک زیرکلاس از یک زیرکلاس است. در برنامه نویسی شیء‌گرا زنجیره‌های اینچنین غیر معمول نیست. بلکه حتی شاهد زنجیره‌های طولانی‌تر از این هم هستیم.



کلاس PartTimeWithOver کلاس PartTimeEmployee را گسترش میدهد، اما هر آنچه که میخواهد از کلاس از کلاس PartTimeEmployee به ارت بردن PartTimeWithOver را انتخاب میکند. زیرا که دارای اعلان خودش برای متدهای findPaymentAmount بوده، و کلاس PartTimeWithOver از والد خود به ارت نمیبرد. به شکل ۸-۹ که در زیر آمده دقت کنید:



اگر شما یک شیء از کلاس PartTimeWithOver ایجاد کنید. شیء ایجاد شده دارای name و jobTitle و cutCheck و hourlyRate میباشد.. اما شیء دارای مند findPaymentAmount که در مثال ۸-۷ تعریف شده نیز میباشد.

## استفاده از متدها از درون زیرکلاس‌ها و کلاس‌ها

```
public class DoPayrollTypeF {  
    public static void main(String args[]) {  
        FullTimeEmployee ftEmployee = new FullTimeEmployee();  
        ftEmployee.setName("HadiKhodapanah");  
        ftEmployee.setJobTitle("CEO");  
        ftEmployee.setWeeklySalary(5000.00);  
        ftEmployee.setBenefitDeduction(500.00);  
        ftEmployee.cutCheck(ftEmployee.findPaymentAmount());  
        PartTimeEmployee ptEmployee = new PartTimeEmployee();  
        ptEmployee.setName("Ali Alizadeh");  
        ptEmployee.setJobTitle("Computer Book Author");  
        ptEmployee.setHourlyRate(7.53);  
        ptEmployee.cutCheck(ptEmployee.findPaymentAmount(50));  
        PartTimeWithOver ptoEmployee = new PartTimeWithOver();  
        ptoEmployee.setName("Kouroshkhashayari");  
        ptoEmployee.setJobTitle("Driver");  
        ptoEmployee.setHourlyRate(7.53);  
        ptoEmployee.cutCheck(ptoEmployee.findPaymentAmount(50));  
    }  
}
```

ابتدا به مثال ۸-۸ دقیق کرده سپس از روی این مثال، مبحث توضیح داده خواهد شد

خروجی:

```
Pay to the order of Hadi Khodapanah (CEO) ****$4,500.00  
Pay to the order of Ali Alizadeh (Computer Book Author) ****$376.50  
Pay to the order of Kourosh khashayari (Driver) ****$451.80
```

با زیرکلاس‌ها، هر سه تای این کارمندان، در مثال ۸-۸ باهم وجود دارند. مطمئناً یک زیرکلاس، از کلاس قدیمی PartTimeEmployee می‌آید. اما این بدین معنی نیست که شما نمیتوانید شیء‌ای از کلاس PartTimeEmployee ایجاد کنید. در واقع، جاوا خیلی هوشمندانه عمل میکند، مثال ۸-۸ سه فراخوانی به متدهای findPaymentAmount دارد و هر فراخوانی به ورژن متفاوتی از متدهای میرسد.

در اولین فراخوانی، `ftEmployee` متغیر `ftEmployee.findPaymentAmount` یک نمونه از کلاس `FullTimeEmployee` میباشد. پس متدهای فراخوانی شده مربوط به مثال ۳-۸ میباشد.

در دومین فراخوانی، `ptEmployee` متغیر `ptEmployee.findPaymentAmount` یک نمونه ای از کلاس `PartTimeEmployee` میباشد. پس متدهای فراخوانی شده مربوط به مثال ۵-۸ میباشد.

در سومین فراخوانی `ptoEmployee` متغیر `ptoEmployee.findPaymentAmount` نمونه ای از کلاس `PartTimeWithOver` میباشد. پس متدهای فراخوانی شده مربوط به مثال ۷-۸ میباشد.

# فصل ٧

از زبان پردازی

## مقیاس دما (temperature scale) چیست؟

### ( انواع enum جاوا )

جاوا راه های گوناگونی را، به منظور گروه بندی چیز های (things)، برای شما فراهم می آورد. در این فصل شما چیز هایی را داخل یک نوع enum گروه بندی خواهید نمود.

ایجاد یک enum پیچیده، آسان نیست ولی برای ایجاد یک enum ساده، کافی است یک گروه از کلمات داخل یک جفت آکولاد بنویسید. مثال ۹-۱ یک نوع enum تعریف میکند که نوع enum در این مثال نام TempScale دارد.

```
public enum TempScale {  
    CELSIUS, FAHRENHEIT, KELVIN, RANKINE,  
    NEWTON, DELISLE, RÉAUMUR, RØMER, LEIDEN  
};
```

وقتی شما نوع enum را تعریف میکنید دو اتفاق مهم می افتد:

\* شما مقادیری را ایجاد میکنید، همانطور که ۱۳ و ۱۵ مقادیر int هستند. CELSIUS و FAHRENHEIT نیز مقادیری TempScale هستند.

\* شما میتوانید متغیر هایی را برای ارجاع به آن مقادیر ایجاد کنید، در مثال ۹-۲ من فیلد های number و scale را اعلان نموده ام. همانگونه که doublenumber متغیر number را از نوع double اعلان میکند.

TempScale نیز متغیر scale از نوع TempScale اعلان میکند

از نوع TempScale بودن به این معناست که شما مقادیر CELSIUS,FAHRENHEIT, KELVIN,..... را دارا میباشید.

یک نوع enum، یک کلاس جاوا در استعاره است. به همین دلیلی است که مثال ۹-۱ شامل تمام فایلی که یک، به یک چیز اختصاص داده شده، میشود، یعنی اعلان نوع TempScale enum . همانند اعلان یک

کلاس ، در اعلان نوع enum به فایل مخصوص خودش متعلق است. در مثال ۹-۱ به فایلی به نام TempScale.java متعلق است.

## چیست؟ Temperature

هر temperature ( درجه حرارت) دو چیز در نظر گرفته میشود: یک عدد و یک مقیاس حرارت. مثال ۲ در زیر این حقیقت را روشن میسازد.

```
public class Temperature {  
    private double number;  
    private TempScale scale;  
    public Temperature() {  
        number = 0.0;  
        scale = TempScale.FAHRENHEIT;  
    }  
    public Temperature(double number) {  
        this.number = number;  
        scale = TempScale.FAHRENHEIT;  
    }  
    public Temperature(TempScale scale) {  
        number = 0.0;  
        this.scale = scale;  
    }  
    public Temperature(double number, TempScale scale) {  
        this.number = number;  
        this.scale = scale;  
    }  
    public void setNumber(double number) {  
        this.number = number;  
    }  
    public double getNumber() {  
        return number;  
    }  
    public void setScale(TempScale scale) {  
        this.scale = scale;  
    }  
    public TempScale getScale() {  
        return scale;  
    }  
}
```

مثال ۹-۲ متد های قرار دهنده (setter) و دریافت کننده (getter) معمول را داراست.(متد های دسترسی دارند به فیلد های number و scale).

نکته ای که باید متوجه آن شده باشید این است که، مثال ۹-۲ ، چهار قسمت ، شبیه به متد وجود دارد..هر یک از این بخش های شبیه به متد، Temperature نام دارند. مشابه با نام کلاس را داشت باشد. هیچ یک از این چیزهای Temperature شبیه به متد، یک نوع return از هیچ نوعی را دارا نمیباشند. نه حتی void که نوع return میباشد. هر یک از این چیزهای شبیه به متد سازنده (constructor) نام دارند.

یک سازنده شبیه یک متد است ، به استثنای اینکه یک سازنده، هدف بسیار مخصوصی دارد-ایجاد اشیاء جدید. هر زمانی که اشیاء جدیدی ایجاد می کند، کامپیوتر دستورات درون سازنده را اجرا میکند.

## با سازنده (statements) چه کاری میتوان انجام داد

در مثال ۹-۳ شما سازنده ای که در مثال ۹-۲ اعلان شده را فراخوانی میکنید. تصویر شماره ۹-۱ نشان میدهد که وقتی شما تمام این کد را اجرا کردید چه اتفاقی خواهد افتاد.

```
import static java.lang.System.out;
public class UseTemperature {
    public static void main(String args[]) {
        final String format = "%5.2f degrees %s\n";
        Temperature temp = new Temperature();
        temp.setNumber(70.0);
        temp.setScale(TempScale.FAHRENHEIT);
        out.printf(format, temp.getNumber(),
        temp.getScale());
        temp = new Temperature(32.0);
        out.printf(format, temp.getNumber(),
        temp.getScale());
        temp = new Temperature(TempScale.CELSIUS);
        out.printf(format, temp.getNumber(),
        temp.getScale());
        temp = new Temperature(2.73, TempScale.KELVIN);
        out.printf(format, temp.getNumber(),
        temp.getScale());
    }
}
```

```
70.00 degrees FAHRENHEIT  
32.00 degrees FAHRENHEIT  
0.00 degrees CELSIUS  
2.73 degrees KELVIN
```

در مثال ۹-۳ هر دستور مانند :

```
temp = new Temperature(blah,blah,blah);
```

یک سازنده را از مثال ۹-۲ فراخوانی میکند. پس همانطور که مثال ۹-۳ را اجرا میشود، آن چهار شیء از کلاس Temperature ایجاد میکند. هر شیء با فراخوانی یک سازنده‌ی متفاوت از مثال ۹-۲ ایجاد میشود.

در مثال ۹-۳ آخرین سازنده دارای دو پارامتر ۲,۷۳ و TempScale.KELVIN بود. این مخصوص فراخوانی سازنده‌ها نیست. فراخوانی یک متده‌ی فراخوانی هم نیز میتواند یک لیست از پارامتر‌ها را داشته باشد. شما پارامتر‌ها را با کاما از یکدیگر جدا میکنید.

تنها قاعده‌ای که بایستی بدان توجه کنید این است که: پارامتر‌های فراخوانی با پارامترهایی که اعلان شده اند تطابق داشته باشند. برای مثال در مثال ۹-۳ هنگام فراخوانی چهارمین و آخرین سازنده:

```
new Temperature(2.73, TempScale.KELVIN)
```

دو پارامتر دارد. اولی از نوع double و دومی از نوع TempScale. جاوا با این فراخوانی موافق خواهد نمود زیرا که آن با هدر(header) تعریف شده در مثال ۹-۲ تطابق دارد:

```
public Temperature(double number, TempScale scale)
```

دو پارامتر دارد اولی از نوع double و دومی از نوع TempScale میباشد. که اگر این تطابق را رعایت نکنید با مشکل مواجه خواهید شد.

## دو راه برای انجام یک کار

مثال ۹-۴ دارای دو سازنده درون خود میباشد. من از دو نام متفاوت استفاده میکنم : number و whatever. در دومین سازنده ، من به دو نام احتیاج ندارم. به جای ایجاد یک نام جدید برای پارامتر سازنده. من تصمیم گرفتم از نام موجود با نوشتن this.number دوباره استفاده کنم.

```
//Use this constructor ...
public Temperature(double whatever) {
    number = whatever;
    scale = TempScale.FAHRENHEIT;
}

//... or use this constructor ...
public Temperature(double number) {
    this.number = number;
    scale = TempScale.FAHRENHEIT;
}

//... but don't put both constructors in your code.
```

در دستور this.number اسماً number و this.number ارجاع میکند.-  
فیلدي که در بالاي مثال ۹-۲ تعلان شده است.

در دستور this.number اسماً number و this.number ارجاع دارد.

```
public class Temperature {
    private double number;
    private ScaleName scale;
```

```
    public Temperature(double number) {
        this.number = number;
        scale = ScaleName.fahrenheit;
    }
}
```

در حالت کلی، `this.someName` به فیلدی که که حاوی کد ارجاع میکند..به عبارت ساده `someName` به نزدیکترین مکان که `someName` اعلام شده رجوع میکند. در دستور `this.number=number` که نزدیکترین مکان لیست پارامتر سازنده `Temperature` میباشد.

## ساخت `temperatures` بہتر

مثال ۹-۵

```
import static java.lang.System.out;
public class TemperatureNice extends Temperature {
    public TemperatureNice() {
        super();
    }
    public TemperatureNice(double number) {
        super(number);
    }
    public TemperatureNice(TempScale scale) {
        super(scale);
    }
    public TemperatureNice(double number, TempScale scale)
    {
        super(number, scale);
    }
    public void display() {
        out.printf("%5.2f degrees %s\n",
            getNumber(), getScale());
    }
}
```

در متدهای `getNumber` و `getScale` متد `display` توجه کنید. چرا چنین کاری را انجام دادیم؟؟ خوب در واقع درون کد کلاس `TemperatureNice` هیچ ارجاع مستقیمی به فیلدهای `number` و `scale` نمود. درست است، هر شیء یک زیر کلاس از کلاس `Temperature` میباشد و کد کلاس `TemperatureNice` فیلد های `Temperature` را دارد. در کنار این همه

و scale را تعریف میکند). اما چون number و scale به صورت خصوصی درون کلاس اعلان گردیده اند. تنها کدی که درست درون کلاس Temperature باشد میتواند از این فیلد ها استفاده کند.

دستور super() در مثال ۹-۵ پارامتر های سازنده‌ی Temperature را در مثال ۹-۲ فراخوانی میکند. که سازنده‌ی بدون پارامتر . . . را به فیلد TempScale و number اختصاص میدهد. FAHRENHEIT به فیلد .field

دستور super(number, scale) در مثال ۹-۵ سازنده‌ی

را که در مثال ۹-۲ آورده شده فراخوانی میکند. به صورتی مشابه، دستورات super(scale) و super(number) در مثال ۹-۵ سازنده‌ها از مثال ۹-۲ فراخوانی میکند.

## Using all this stuff

```
public class UseTemperatureNice {  
    public static void main(String args[]) {  
        TemperatureNice temp = new TemperatureNice();  
        temp.setNumber(70.0);  
        temp.setScale(TempScale.FAHRENHEIT);  
        temp.display();  
        temp = new TemperatureNice(32.0);  
        temp.display();  
        temp = new TemperatureNice(TempScale.CELSIUS);  
        temp.display();  
        temp = new TemperatureNice(2.73,  
                                   TempScale.KELVIN);  
        temp.display();  
    }  
}
```

کد مثال ۹-۶ در مثال بالا خیلی شبیه به کد ۳-۳ میباشد. تنها تفاوت آنها عبارت اند از :

در مثال ۹-۶ نمونه ای از کلاس TemperatureNice ایجاد میشود. به همین دلیل است مثال ۹-۶ سازنده ها را از کلاس TemperatureNice فراخوانی میکند نه از کلاس Temperature.

در مثال ۹-۶ از امتیاز متدهای display در کلاس TemperatureNice بهره میگیرد. بنابراین کد مثال ۹-۶ منظم و آرایسته تر از همتایش در مثال ۳-۳ است.

## یک سازنده‌ی پیش‌فرض

```
publicFullTimeEmployee() {  
super();}
```

سازنده‌ی که در مثال ۹-۷ در بالا استفاده کردیم هیچ پارامتری نمیگیرید. و تنها دستور موجود در آن، سازنده‌ی هر کلاسی که شما توسعه داده اید را فراخوانی میکند. (در صورتی که کلاسی که شما توسعه داده اید سازنده‌ی بدون پارامتر نداشته باشد ، اشتباه رخ خواهد داد).

به مثال ۹-۸ در زیر توجه نمایید:

```
public class FullTimeEmployee extends Employee {  
private double weeklySalary;  
private double benefitDeduction;  
publicFullTimeEmployee(double weeklySalary) {  
this.weeklySalary = weeklySalary;  
}  
public void setWeeklySalary(double weeklySalaryIn) {  
weeklySalary = weeklySalaryIn;  
}  
public double getWeeklySalary() {  
return weeklySalary;  
}  
public void setBenefitDeduction(double benefitDedIn) {  
benefitDeduction = benefitDedIn;  
}  
public double getBenefitDeduction() {  
return benefitDeduction;  
}  
public double findPaymentAmount() {  
return weeklySalary - benefitDeduction;  
}  
}
```

اگر شما مانند زیر از کد FullTimeEmployee در مثال ۹-۸ استفاده کنید، برنامه‌ی شما کار نخواهد کرد:

```
FullTimeEmployee ftEmployee = new FullTimeEmployee();
```

این برنامه کار نخواهد کرد زیرا که سازنده‌ی FullTimeEmployee اعلان شده که یک پارامتر double بگیرد، (یک سازنده‌ی بدون پارامتر پیش فرض را قبول نمیکند).

### سازنده‌ای که کارهای بیشتری انجام میدهد

در این بخش میخواهیم مثالی از یک سازنده بزنم که کاری بیشتر از اختصاص مقادیری به چند فیلد انجام میدهد. مثال‌های ۹-۹ و ۹-۱۰ و نتیجه‌ی خروجی نیز در مثال ۹-۳ نشان داده شده است.

### Frame تعریف

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JButton;
@SuppressWarnings("serial")
public class SimpleFrame extends JFrame {
    public SimpleFrame() {
        setTitle("Don't click the button!");
        setLayout(new FlowLayout());
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        add(new JButton("Panic"));
        setSize(300, 100);
        setVisible(true);
    }
}
```

```
public class ShowAFrame {
    public static void main(String args[]) {
        newSimpleFrame();
    }
}
```

نشان دادن Frame

شکل ۹-۳



کد مثال ۹-۹ عمدتاً از متدهای API های جاوا شکل گرفته است. (قصد ما از این مثال توضیح دادن تمام ایم متدها نیست). به هر حال متدهای اصلی مثال ۹-۱۰ فقط یک دستور دارد: یک فراخوانی به سازنده در کلاس SimpleFrame. توجه کنید که چگونه شیء ای که این فراخوانی ایجاد میکند، حتی به یک متغیر هم تخصیص داده نشده است. همه چیز درست است زیرا که ما در این کد نیازی به ارجاع به شیء مورد نظر در هیچ جای دیگری نداریم.

در بالا، کلاس SimpleFrame فقط تنها یک اعلان سازنده (constructor) موجود است. این سازنده متدهای از متدهای دیگر را از API جاوا فراخوانی میکند.

تمام متدهای فراخوانی شده در کلاس سازنده‌ی SimpleFrame، از کلاس والد JFrame می‌آیند. کلاس JFrame در داخل پکیج javax.swing موجود است. این پکیج دیگر پکیج‌های موجود در java.awt به شما کمک میکند که تصاویر، پنجره، نقاشی، ... و دیگر اسباب و ابزار را روی صفحه‌ی نمایش کامپیوتر قرار دهید.

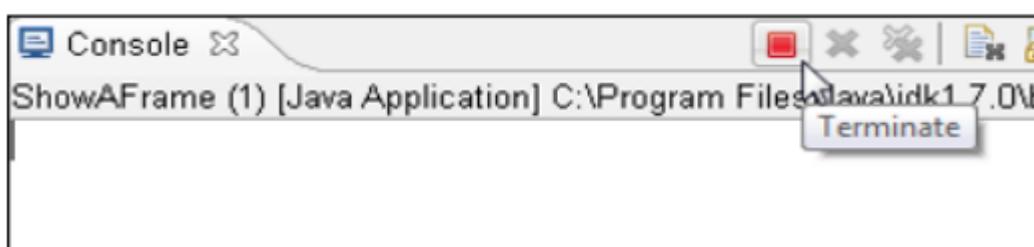
متدهای setTitle فراخوانی کلمات را در نوار عنوان فریم قرار میدهد. (شیء جدید SimpleFrame خودش را فراخوانی کیکند).

متدهای FlowLayout فراخوانی : یک نمونه از کلاس FlowLayout، اشیاء روی فریم رابه سبک ماشین تحریر در مرکزیت قرار میدهد. زیرا که فریم تصویر ۹-۳ تنها یک دکمه در خود دارد. آن دکمه در مرکزیت و نزدیک بالای فریم قرار داده شده است. اگر فریم هشت دکمه می‌داشت. پنج تای آن‌ها ممکن بود در خط بالایی فریم قرار گرفته و سه تای باقی مانده در مرکزیت خط پایین فریم قرار بگیرند.

متدهای setLayout فراخوانی : FlowLayout شیء جدید را متصدی مرتب ساختن اجزاء میکند. مانند دکمه‌ها یا فریم. (شیء جدید SimpleFrame متدهای setLayout خودش را فراخوانی میکند).

روی `setDefaultCloseOperation` فراخوانی `setDefaultCloseOperation` به جاوا میگوید که وقتی شما در گوشه بالا-راست کلیک میکنید، چه انجام دهد. بدون فراخوانی این متد، فریم خودش ناپدید میشود. اما ماشین مجازی جاوا (Java Virtual Machine) به فعالیت ادامه میدهد. اگر شما از Eclipse استفاده میکنید شما بایستی JVM را با کلیک روی مربع چهار گوش بالای کنسول ، متوقف کنید.(شکل ۹-۴ را ببینید).

فراخوانی `setDefaultCloseOperation(EXIT_ON_CLOSE)` به جاوا میگوید که هنگامی که شما روی `X` در گوشه بالا-سمت راست کلیک میکنید، خودش را بیندد.



شکل ۹-۴

`JButton` کلاس در پکیج `javax.swing` موجود است. یکی از سازنده های کلاس، برای پارامتر هایش نمونه ای از `String` را میگیرد.(مانند "Panic"). فراخوانی این سازنده باعث میشود که نمونه(`String`)`instance` روی دکمه `i` جدید برچسب زده شود.

: شیء `SimpleFrame` جدید متد `add` خودش را فراخوانی میکند. فراخوانی متد `add`، دکمه را روی سطح ظاهری شیء قرار میدهد.(در این مثال سطح ظاهری فریم).

: فریم `setSize` ۳۰۰ پیکسل عرض ۱۰۰ پیکسل ارتفاع دارد.(در پکیج `javax.swing` هنگامی که شما ابعاد دو بعد را مشخص کنید، عدد مشخص کننده `ی` عرض(پهنا) قبل از عدد مشخص کننده `ی` ارتفاع میآید.

: زمانی نخست که آن ایجاد میشود. یک فریم جدید ناپدید میشود. اما زمانی که فریم جدید `setVisible` را فراخوانی میکند. روی صفحه `i` نمایش کامپیوتر شما ظاهر میشود.

## مَنَابِعُ وَمَاحْدَذُ :

**Java How to Program (9th Edition) ,By Paul Deitel , Harvey Deitel, Publisher : Prentice Hall**

**Java 8: The Fundamentals, By Dane Cameron, Publisher: Cisdal Publishing**

**Head First Java, By Kathy Sierra , Bert Bates, Publisher: O'Reilly Media**

**Java The Complete Reference, 8th Edition,By Herbert Schildt, Publisher: McGraw-Hill Osborne**

**Java 8 Pocket Guide, By Robert Liguori , Patricia Liguori, Publisher: O'Reilly Media**

**Learn Java for Android Development: Java 8 and Android 5 Edition,By Jeff Friesen, Publisher: Apress**

**Pro Java 8 Programming, By Terrill Brett Spell, Publisher: Apress**

**Java 8: The Fundamentals, By Dane Cameron, Publisher: Cisdal Publishing**

**Java SE8 for the Really Impatient: A Short Course on the Basics 1st Edition,By Cay S. Horstmann, Publisher: Addison-Wesley Professional**

**Beginning Java, Author : Ivor Horton,Publication : Wrox**