برنا مەنویسی جاوا

الهام منتظری

# مهارت‌های برنامه‌نویسی
# Programming Skills

# سرفصل مطالب

- موضوع این جلسه: مروری بر چند نکته و مهارت برنامه‌نویسی
- مستندات جاوا (javadoc)
- مفهوم Classpath
- فایل‌های JAR
- امکانات محیط‌های توسعه (IDE)
  - تکمیل کد
  - امکانات اشکال‌یابی (Debugging)
  - تولید کد (تولید سازنده، تولید equals، toString، getter و setter)

# فایل‌های آرشیو جاوا
# JAR Files

# فایل‌های JAR

## JAR => Java Archive

- یک فایل که مجموعه‌ای از فایل‌ها و کلاس‌های جاوا را نگهداری می‌کند
- به طور معمول، یک JAR شامل کلاس‌های ترجمه شده (.class) است
  - و احتمالاً فایل‌های تکمیلی
  - مثل فایل‌های تنظیمات که در برنامه‌ها استفاده می‌شوند


- معمولاً متن برنامه‌ها (فایل‌های .java) در JAR گنجانده **نمی‌شود**
- در نگاه اول، یک فایل JAR مانند یک فایل فشرده (zip) است
  - شامل کلاس‌های مختلف یک پروژه که در شاخه‌های (بسته‌ها) مختلف قرار دارند
  - البته پسوند این فایل هم .jar است

# فایل‌های JAR (ادامه)

- مزایای استفاده از فایل‌های JAR
  - مجموعه‌ای از فایل‌ها، برنامه‌ها و کلاس‌ها به صورت یکجا مدیریت می‌شوند
  - دانلود ساده‌تر، امکان فشرده‌سازی، نسخه‌بندی نرم‌افزار و غیره
- نحوه ایجاد فایل JAR از یک برنامه
  - استفاده از دستور jar : ***jar*** *cf jar-file input-files*
  - و یا استفاده از ابزارها (Eclipse، Ant یا Maven)
- نحوه مشاهده محتویات فایل JAR
  - ساده‌ترین راه: از ابزارهای zip استفاده کنید (مثلاً WinRAR)
  - سایر راهها: استفاده از دستور jar یا امکانات محیط توسعه (IDE)

# مفهوم ClassPath

# مفهوم Classpath (CP)

- یک پارامتر برای کامپایلر جاوا یا JVM است
- مشخص می‌کند در چه محل‌هایی به‌دنبال کلاس‌ها و بسته‌ها بگردند
- این پارامتر به java یا javac پاس می‌شود
  - یا به‌صورت یک متغیر محیطی (Environment Variable) تعریف می‌شود
- البته محل کلاس‌های موجود در زبان جاوا نیازی به معرفی در CP ندارند
  - مثلاً String
- با مفهوم path اشتباه نگیرید
  - path مربوط به سیستم‌عامل است و محل فایل‌های اجرایی را مشخص می‌کند
- با مفهوم Working Directory اشتباه نگیرید

# کاربرد Classpath

```
D:\myprogram\
       |
       ---> org\
              |
              ---> mypackage\
                     |
                     ---> HelloWorld.class
                     ---> SupportClass.class
                     ---> UtilClass.class
```

/home/user/myprogram/

- در ویندوز:
- java -cp D:\myprogram org.mypackage.HelloWorld
- در لینوکس
- java -cp /home/user/myprogram org.mypackage.HelloWorld

# مثال‌هایی از کاربرد Classpath

- استفاده از متغیر محیطی:

```
set CLASSPATH=D:\myprogram
java org.mypackage.HelloWorld
```

- تعیین چند فولدر یا JAR در classpath :

```
java -cp D:\prog;D:\lib\support.jar org.HelloWorld
```

- برای جداکردن بخش‌های مختلف cp در لینوکس : و در ویندوز ;
- **-cp** معادل –classpath است
- مثال: استفاده از چندین فایل JAR و شاخه جاری به عنوان cp :

```
java -classpath '.:/mylib/*' MyApp
```

# مستندات جاوا
# Javadoc

# تولید مستندات با کمک javadoc

- یکی از امکانات محیط جاوا: دستور **javadoc**
- امکانی برای ایجاد خودکار مستندات (documents) از برنامه‌های جاوا
- مستندات: متن‌هایی که برنامه‌ها را به خوبی توصیف می‌کند
- امکانی بسیار مفید برای شناختن برنامه‌های دیگران
  - کلاس‌ها، متدها و ...
- مثال:


```
C:\> javadoc Rectangle.java
```

# نحوه تعریف javadoc

```
/** This class represents a human.*/
public class Person {
 /** national ID (SSN) */
 private String ID;
 ...
}
```

- **کامنتی** که با **/**** شروع می‌شود (به جای */)
  به عنوان **جاواداک** در نظر گرفته می‌شود
- این جاواداک قبل از تعریف هر چیزی که باشد، همان را توصیف می‌کند
  - مثلاً قبل از یک کلاس، متد، سازنده، یا ویژگی
- دستور javadoc این توصیفات را به یک مستند HTML تبدیل می‌کند
- امکاناتی برای توصیف بهتر برنامه نیز وجود دارد (فراتر از متن)
  - تگ‌هایی (tags) که توضیح خاصی اضافه می‌کنند (مثل @author)
  - امکاناتی برای برقراری ارتباط بین مستندات مختلف (مثل @see)

# مثال

```java
/**
 * This class represents a human. Objects of this class are immutable...
 * @author Sadegh Aliakbary
 * @see    java.lang.String
*/
public class Person {
 /** national ID (SSN) */
 private String ID;
 private String name; //No javadoc
 /**
  * The only constructor of the class
  * @param id The social security number (national ID)
  * @param name The full-name, including first-name and last-name
  */
 public Person(String id, String name) {
  ID = id;
  this.name = name;
 }
 /**
  * This method should be called to ask the person run
  * @param speed The speed of running
  * @return returns true if he/she can run with that speed
  */
 public boolean run(double speed){...}
}
```

# بخش‌هایی از مستند تولیدشده توسط javadoc

**ir.javacup.practicenotes.Person**

This class represents a human being. Objects of this class are immutable, no setter or getter method is available. Two objects are considered equal if they have identical IDs.

**Author:**

Sade

**See Also**

java

java

**ir.javacup.practicenotes.Person.Person(String id, String name)**

The only constructor of the class

**Para**

**boolean ir.javacup.practicenotes.Person.run(double speed)**

This method should be called to ask the person to run

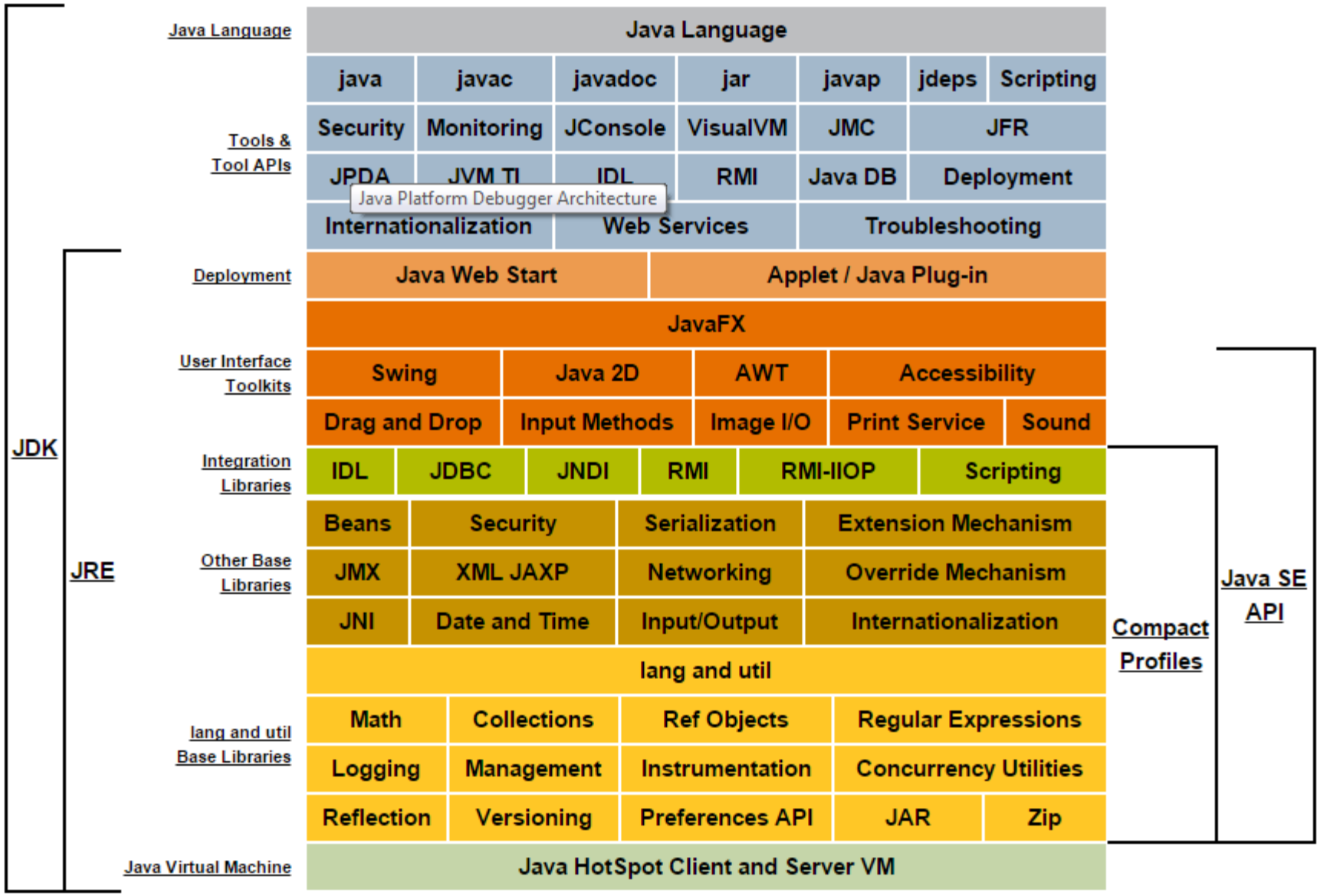
**Parameters:**

**speed** The speed of running

**Returns:**

returns true if he/she can run at the specified speed

# مثال: https://docs.oracle.com/javase/8/docs/

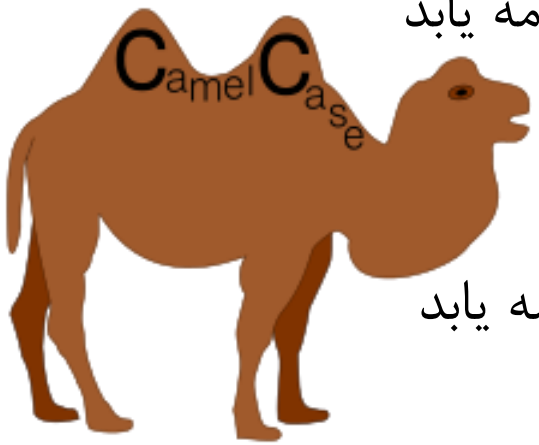


# قواعد مرسوم در برنامه‌نویسی جاوا
# Java Code Conventions

# مفهوم Code Convention

- در هر زبان برنامه‌نویسی، برنامه‌نویسان عرف و قواعد مرسومی دارند
  - که توسط زبان برنامه‌نویسی **تحمیل نمی‌شود**
  - ولی رعایت آن‌ها **رایج و مرسوم و مفید** است
- به این موارد، Code convention گفته می‌شود
- شامل توصیه‌هایی در زمینه:
  - **نامگذاری** (نام بسته، کلاس، متد، متغیر، ثابت و ...)
  - قالب خطوط برنامه در دستورات پیچیده‌تر (if-then-else ، switch و ...)
  - نحوه دندانه‌گذاری
- این قواعد کمک می‌کند **خوش‌خط** برنامه‌نویسی کنید
  - **دست‌خط** برنامه‌نویسی شما چگونه است؟ (تمیز کد بزنید)

# آداب و رسوم نامگذاری (Java Naming Conventions)




- **بسته**: همه حروف کوچک. مثل com.sun.eng
- **کلاس**: با حرف بزرگ شروع شود و با الگوی «کوهان شتر» ادامه یابد
  - مثال: Raster یا ImageSprite
  - از «اسم» برای نام‌گذاری استفاده کنید
- **متد**: با حرف کوچک شروع شود و با الگوی «کوهان شتر» ادامه یابد
  - مثال: runFast یا getBackground
  - از «فعل» برای نام‌گذاری استفاده کنید
- **متغیرها**: شروع با حرف کوچک و ادامه با الگوی «کوهان شتر»
  - مثال: myWidth یا maxNumber
- **ثابت‌ها**: همه حروف بزرگ، کلمات مختلف در نام با underscore (_) جدا شوند
  - مثال: MIN_WIDTH

# مثال‌هایی از رسوم جاوا

```
switch (condition) {
case ABC:
     statements;
     /* falls through */
case DEF:
     statements;
     break;

case XYZ:
     statements;
     break;

default:
     statements;
     break;
}
```

```
if (condition) {
     statements;
} else if (condition) {
     statements;
} else {
     statements;
}
```

```
for (initialization; condition; update) {
     statements;
}
```

# توصیه: نامگذاری مناسب

- برای کلاس‌ها، متغیرها و متدها از اسامی بامعنی و گویا استفاده کنید
- نام‌هایی مانند این‌ها مناسب نیستند:

- a,b,c, x, y,z
- a1, var1, var2
- method1,Class, MyClass, myMethod

- برای متغیرهای شمارنده (مثل شمارنده for) نام‌هایی مثل i و j اشکالی ندارد
- گاهی نام‌های مناسب، نیاز به کامنت و جاواداک را کمتر می‌کند

# امکانات محیط‌های توسعه
# IDE Features

# امکانات محیط‌های توسعه (IDE)

- محیط‌های توسعه دارای امکانات مفیدی هستند
  - برنامه‌نویسی را تسهیل می‌کنند
  - باعث تسریع برنامه‌نویسی می‌شوند
  - اشتباه‌های برنامه‌نویس را کاهش می‌دهند
  - بسیاری از کارها قابل خودکارسازی است
  - اشکال‌یابی و اشکال‌زدایی (Debugging) را آسان می‌کنند

# برخی امکانات کمکی IDE

- راه‌های میانبُر برای کارها
  - کامپایل و اجرای برنامه
  - تولید javadoc
  - میانبرهایی (shortcut) برای تولید کد
- تکمیل کد (معمولاً با Ctrl+Space)
- پیشنهاد راه‌حل در مواقع بروز خطا یا هشدار
- جستجوی هوشمند
  - مثال: این متد/کلاس/متغیر در چه جاهایی استفاده شده است؟

# اشکال‌یابی و اشکال‌زدایی (Debugging)

- روش اشکال‌یابی:
  - اجرای خط‌به‌خط برنامه
  - مشاهده مقادیر متغیرها و نحوه اجرای برنامه
  - پیدا کردن اشکال برنامه
- نکته: برنامه‌های بزرگ، معمولاً با کمک **لاگ** (**log**) اشکال‌یابی می‌شوند
  - **لاگ**: ثبت اتفاقاتی که در برنامه می‌افتد و مقدار برخی متغیرها در خروجی
    - در خروجی استاندارد، یا یک فایل، یا ...
  - در هنگام بروز اشکال، «**لاگ**»ها مطالعه و بررسی می‌شوند

# تمرین عملی

# تمرین عملی

- امکانات IDE:
  - تکمیل کد، تولید کد، پیشنهاد اصلاح
- dot, Ctrl+Space, templates, formatters, …
  - دیباگ (Breakpoint، اجرای خط‌به‌خط، Watch، Inspect، Expression)
  - جستجوی حرفه‌ای (مثلاً Find usages)
  - تنظیم پارامترها (classpath ، working dir ، VM args و app args)
- ساخت JAR
- مفهوم و اهمیت هشدارها (Warning)
- تولید و مشاهده javadoc
  - نگاهی به جاواداک‌های کلاس‌های جاوا

# جمع‌بندی

# جمع‌بندی

- مستندات جاوا (javadoc)
- مفهوم Classpath
- فایل‌های JAR
- آداب و رسوم جاوا (Java Coding Conventions)
- امکانات محیط‌های توسعه (IDE)
  - تکمیل کد، امکانات Debugging، تولید خودکار کد

# مطالعه کنید و پیش بروید

- مستندات IDE مورد علاقه خود را مرور کنید
  - مثلاً یکی از این موارد برای Eclipse :

http://www.tutorialspoint.com/eclipse/

http://eclipsetutorial.sourceforge.net/totalbeginner.html

http://wiki.eclipse.org/Eclipse_Articles,_Tutorials,_Demos,_Books,_and_More

- از یک دوست «حرفه‌ای‌تر» بخواهید:
  - برخی فوت و فن‌های IDE موردنظرتان را به شما نشان دهد
  - نگاهی به «دست‌خط» برنامه‌نویسی شما بیاندازد و اشکالات خط را بگوید

# تمرین

- مستندات کلاسهای System و String را بخوانید (javadoc)
  - متن این کلاس‌ها و نحوه اضافه کردن مستندات در متن برنامه را هم ببینید
- یک کلاس (مثلاً Person) پیاده‌سازی کنید و :
  - بخش‌های ممکن را به صورت خودکار تولید کنید (سازنده، getter، setter و ..)
  - مستندات کاملی برای آن ایجاد کنید
- یکی از پروژه‌هایی که تابه‌حال نوشته‌اید را به فایل JAR تبدیل کنید
  - از این فایل استفاده کنید (با کمک IDE و بدون کمک IDE)
- برنامه‌های قبلی که نوشته‌اید را از منظر رسوم و قواعد جاوا مرور کنید
  - اصلاحات لازم را انجام دهید تا این قواعد رعایت شوند
- اشکال‌یابی و اجرای خطبه‌خط و امکانات مرتبط را تمرین کنید

# جستجو کنید و بخوانید

- فایل‌های JAR
  - Signed JAR Files
  - Manifest
  - اجرای برنامه‌ای که به صورت JAR درآمده است
  - سایر قالب‌های مشابه: WAR، EAR ، SAR و APK (اندروید)
- استفاده از چه امکانات و تگ‌هایی در javadoc ممکن است؟
- امکانات و ابزارهای مهم برنامه‌نویسی
  - Apache Ant
  - Apache Maven
- روش‌های logging (مثل Java Logging API)

# پایان