

انجمن جاوا کا پتہ دیم می کند

دوره برنامه نویسی جاوا

امکانات شیء گرا در جاوا

OOP in Java, a Deeper Look

صادق علی اکبری

- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- باز نشر یا تدریس آن چه توسط جاواکاپ و به صورت عمومی منتشر شده است، با ذکر مرجع (جاواکاپ) بلامانع است
- اگر این اثر توسط جاواکاپ به صورت عمومی منتشر نشده است و به صورت اختصاصی در اختیار شما یا شرکت شما قرار گرفته، باز نشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست



سرفصل مطالب

- روش طبقه‌بندی کلاس‌ها در جاوا

- بسته (package)

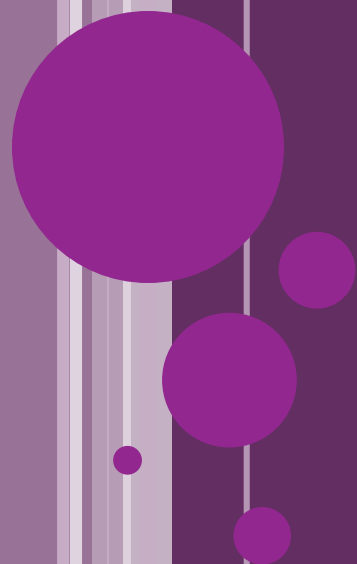
- سطوح دسترسی

- Access Levels

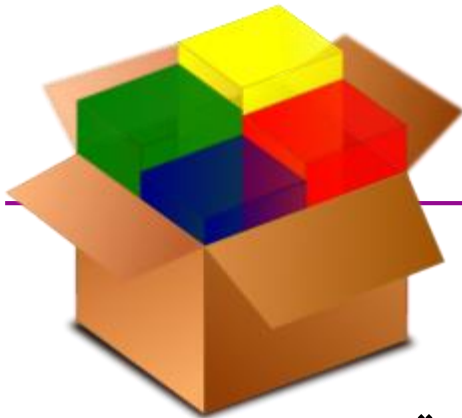
- مفهوم استاتیک (static)

- کلیدواژه this





پسته
Package



بسته (package)

- یک بسته، شامل تعدادی کلاس است
- بسته، امکانی برای طبقه‌بندی و گروه‌بندی کلاس‌های جاوا است
- مانند مفهوم شاخه (Folder) در مدیریت فایل‌ها
- هر بسته شامل کلاس‌هایی است که به یک دسته متعلق هستند
- و یا کاربرد مشابهی دارند
- بسته، یک فضای نامگذاری (namespace) یکتا برای کلاس‌ها فراهم می‌کند
- نام دقیق کلاس = نام بسته + نام کلاس
- هر بسته می‌تواند شامل بسته‌های دیگر نیز باشد
- سلسله‌مراتبی از بسته‌ها



تعریف بسته

- کلاسهای یک بسته در یک شاخه (Folder) نگهداری می‌شوند
- برای تعریف یک بسته، کافایت یک شاخه جدید بسازیم
- برای قرار دادن یک کلاس در یک بسته:

۱- فایل کلاس را درون شاخه بسته قرار می‌دهیم

۲- در متن برنامه کلاس، حضور این کلاس در آن بسته را تصریح می‌کنیم

◦ در اولین خط از تعریف کلاس، با استفاده از دستور `package`

◦ مثال `package ir.javacup;`

- برای قرار دادن یک بسته در یک بسته دیگر
- شاخه (Folder) بسته اول را درون شاخه بسته دوم قرار می‌دهیم



```

Person.java
package human;

public class Person {

}

Rectangle.java
package shape.twodimensional;

public class Rectangle {

}

Instructor.java
package human.edu;

public class Instructor {

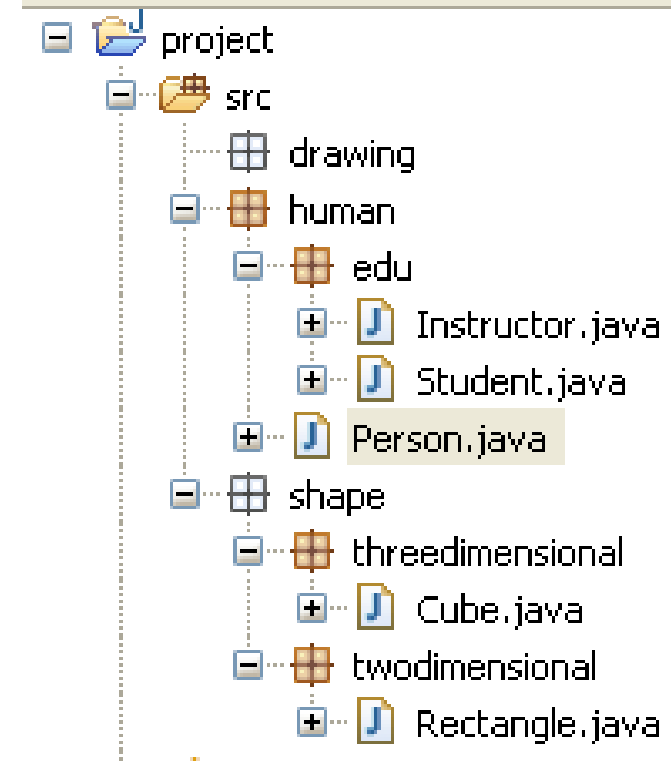
}

Cube.java
package shape.threedimensional;

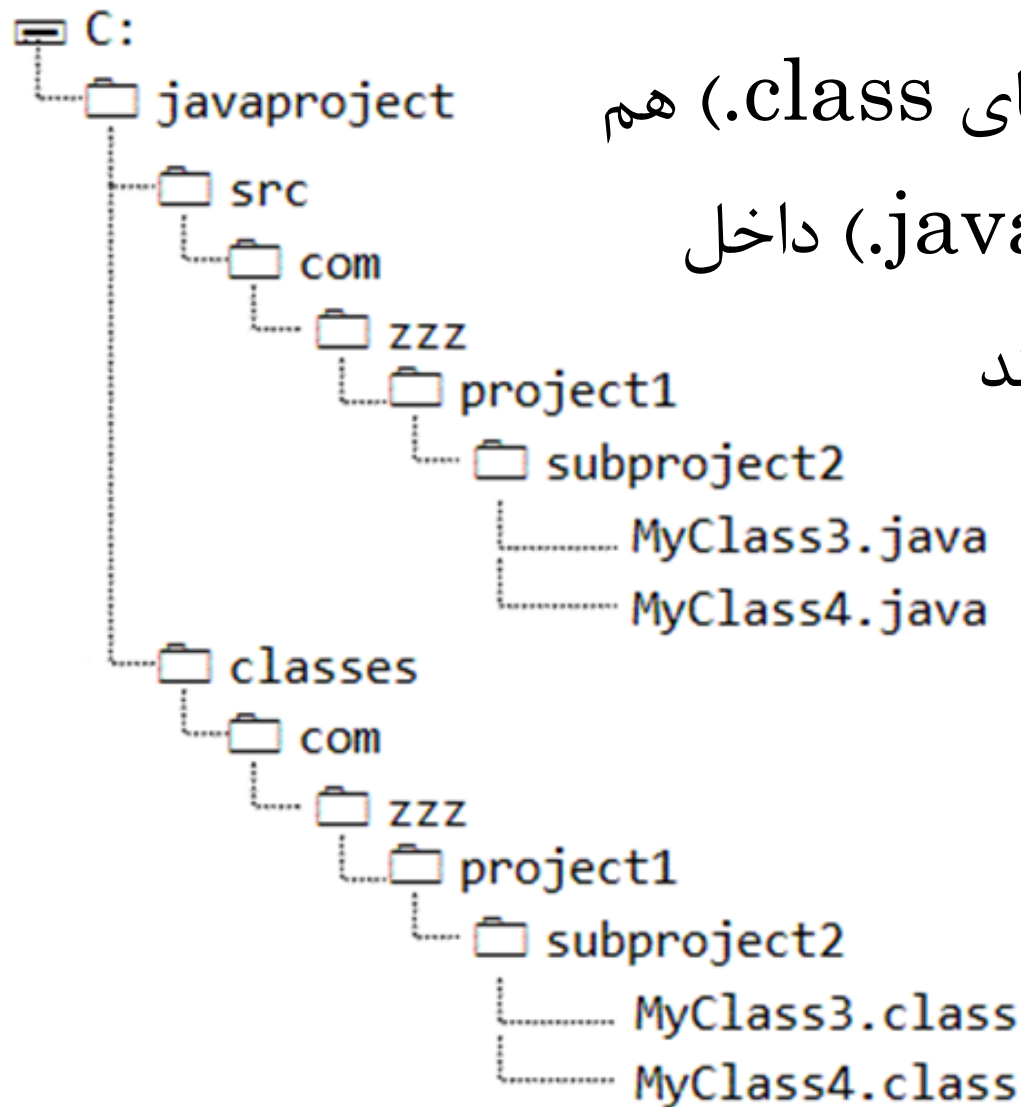
public class Cube {

}

```



بسته‌ها و شاخه‌ها (Packages and Folders)



کلاس‌های کامپایل شده (فایل‌های .class) هم
مانند متن برنامه‌ها (فایل‌های .java) داخل
شاخه‌بندی بسته‌ها قرار می‌گیرند

نحوه استفاده از بسته‌ها

- روش اول: استفاده از نام کامل کلاس‌ها
- نام کامل کلاس = نام بسته + نقطه + نام کلاس

```
public class Swapping {  
    public static void main(String[] args) {  
        java.util.Scanner scanner ;  
        scanner = new java.util.Scanner(System.in) ;  
        int nextInt = scanner.nextInt() ;  
        System.out.println(nextInt) ;  
    }  
}
```



دستور import

- روش دوم: استفاده از دستور **import**

- نام کامل کلاس را در دستور **import** مشخص می‌کنیم

- همه دستورهایی **import** باید در ابتدای فایل (بعد از **package**) باشند

```
import java.util.Scanner;

public class Swapping {
    public static void main(String[] args) {
        Scanner scanner ;
        scanner = new Scanner(System.in) ;
        int nextInt = scanner.nextInt() ;
        System.out.println(nextInt) ;
    }
}
```



دستور import (ادامه)

- ممکن است چند بار از import استفاده کنیم

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

public class Swapping {
    public static void main(String[] args) {
        Scanner scanner ;
        scanner = new Scanner(System.in);
        int nextInt = scanner.nextInt();
        System.out.println(nextInt);

        int[] original = new int[5];
        int[] copy = Arrays.copyOf(original, 10);
    }
}
```



دستور import (ادامه)

- با کمک ستاره (*) همه کلاس‌های یک بسته قابل استفاده می‌شوند
- نکته: فقط همه کلاس‌های همان بسته، و نه بسته‌های زیرمجموعه آن

```
import java.util.*;
```

```
public class Swapping {  
    public static void main(String[] args) {  
        Scanner scanner ;  
        scanner = new Scanner(System.in);  
        int nextInt = scanner.nextInt();  
        System.out.println(nextInt);  
  
        int[] original = new int[5];  
        int[] copy = Arrays.copyOf(original, 10);  
    }  
}
```



چند بسته معروف در جاوا

- java.lang
 - java.lang.String
 - java.lang.Math
- java.util
 - java.util.Scanner
 - java.util.Arrays
- java.io
- java.sql
- ...

● نکته:

● بسته‌ی java.lang به طور

ضمنی import شده است

● کلاس‌های این بسته لازم نیست

import شوند یا با نام کامل

ذکر شوند

● مثلاً String



- دستور `import` فقط مربوط به کامپایلر است
- کامپایلر جاوا با کمک این دستور، نام دقیق کلاس‌ها را می‌فهمد
- و نام کلاس‌ها را با نام کامل آن‌ها جایگزین می‌کند
- دستور `import` در کلاس کامپایل شده دیده نمی‌شود
- فایل `.class` یا `bytecode`
- دستورهای `import` بلااستفاده هیچ تأثیری در زمان اجرا ندارد
- فقط تأثیر بسیار ناچیزی بر روی کامپایلر دارند



نام‌گذاری بسته‌ها

- هر شرکت یا گروه تولیدکننده نرم‌افزار، یک آدرس سایت دارد
 - یا برای خودش متصور است
- مثال: `google.com` ، `javacup.ir` ، `apache.org`
 - یا `taghitaghavi.ir`
- نحوه معمول نام‌گذاری بسته‌ها : از کل به جزء
 - زیرمجموعه . پروژه . دپارتمان . شرکت . دامنه
- مثال:

`ir.javacup`

`org.apache.commons.io`





سطوح دسترسی Access Levels

سطوح دسترسی

- سطح دسترسی به هر متد یا ویژگی (property) توسط یک کلیدواژه قابل تعیین است
- این کلید واژه: تعیین کننده دسترسی (Access Specifier یا Access Modifier)
- تعیین کننده های دسترسی: public ، private و هیچ.
- عمومی (public)
- از همه جا قابل دسترسی است. دسترسی به آن آزاد است.
- خصوصی (private)
- دسترسی به آن فقط داخل همین کلاس ممکن است.
- حالت پیش فرض: عدم تعیین سطح دسترسی (package access)
- دسترسی به آن فقط در داخل کلاس های همین بسته ممکن است.

دسترسی protected
را نیز بعداً خواهیم دید



```
package ir.javacup.oop;
public class Person {
    public String name;
    private int birthyear;
    boolean married;
    void run(){}
    public int getAge() {
        return computeAge();
    }
    private int computeAge(){
        return 2015-birthyear;
    }
}
```

```
package ir.javacup.oop;
public class Util {
    public void f(){
        Person p = new Person();
        p.married = true;
        p.name = "Ali";
        p.birthyear = 1992;
        int age = p.getAge();
        p.run();
        age = p.computeAge();
    }
}
```

Syntax Error

اگر کلاس Util در بسته ir.javacup بود چطور؟



سطح دسترسی به کلاس

- سطوح دسترسی به متدها و ویژگی‌ها را دیدیم
- public, private, package access
 - سطوح دسترسی به کلاس‌ها: public و package access
 - سطح private برای یک کلاس معنی ندارد
 - بنابراین کلمه public می‌تواند برای تعریف کلاس ذکر نشود
 - چنین کلاسی فقط در داخل همه کلاس‌های همان بسته قابل استفاده است
 - در داخل یک فایل می‌توانیم صفر یا چند کلاس غیرعمومی تعریف کنیم
 - هر فایل جاوا حداکثر یک کلاس عمومی دارد
 - کلاس عمومی، در صورت وجود، باید دقیقاً همانام فایل باشد
 - ممکن است یک فایل جاوا، کلاس عمومی نداشته باشد



درباره setter و getter

- هنگام تعریف کلاس‌ها در فرایند محصورسازی (Encapsulation)
- معمولاً ویژگی‌ها (Property) به صورت `private` تعریف می‌شوند
- برای تغییر ویژگی‌ها، متدهای `setter` تعریف می‌شوند
- برای دریافت مقدار ویژگی‌ها متدهای `getter` تعریف می‌شوند
- متدهای `getter` و `setter` به صورت `public` تعریف می‌شوند
- به متدهای `getter` ، `accessor` هم گفته می‌شود
- به متدهای `setter` ، `mutator` هم گفته می‌شود



```
public class Book {  
    private String title;  
    private int pages;  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    public int getPages() {  
        return pages;  
    }  
  
    public void setPages(int pages) {  
        this.pages = pages;  
    }  
}
```



چرا getter و setter تعریف کنیم؟

```
public void setAge(int a) {  
    if (a > 0 && a < 150)  
        age = a;  
}
```

- چرا ویژگی‌ها را public نکنیم؟
- امکان اعتبارسنجی در setter ها
- اجازه هر مقداری را ندهیم.
- امکان شبیه‌سازی ویژگی‌هایی که در واقع وجود ندارند
- setAge و getAge براساس ویژگی پنهان «تاریخ تولد»
- محدود کردن نحوه دسترسی
- مثلاً برای یک ویژگی خاص getter را public تعریف کنیم و setter را private تعریف کنیم (یا اصلاً تعریف نکنیم)
- بسیاری مزایای دیگر



کوییز

کدام گزینه درباره یک فایل برنامه جاوا صحیح است؟

- ۱- حداقل باید شامل یک کلاس `public` باشد
- ۲- باید نامی دقیقاً برابر با کلاسی که در آن قرار گرفته داشته باشد
- ۳- نام فایل باید دارای پسوند `.java` باشد
- ۴- محتوای این فایل حتماً باید با دستور `package` آغاز شود

پاسخ صحیح:

گزینه ۳




```
package ir.javacup.oop;
public class Person {
    public String name;
    private int birthyear;
    boolean married;
    void run(){}
    public int getAge() {
        return computeAge();
    }
    private int computeAge(){
        return 2015-birthyear;
    }
}
```

- در هر یک از شرایط زیر برای کلاس Util، کدام ویژگی‌ها و متدهای کلاس Person در کلاس Util قابل استفاده خواهند بود؟
- اگر در همین فایل تعریف شود
- اگر در همین بسته تعریف شود
- اگر در بسته ir تعریف شود
- اگر در بسته ir.javacup تعریف شود
- اگر در بسته org.apache تعریف شود



سؤال: مخفی برای کلاس یا مخفی برای شیء!؟

```
public class Access {  
    private String name;  
    public Access(String name) {  
        this.name = name;  
    }  
    public void check(Access access) {  
        access.name = name;  
    }  
    public static void main(String[] args) {  
        Access a = new Access("Ali");  
        Access b = new Access("Taghi");  
        a.check(b);  
        System.out.println(a.name);  
        System.out.println(b.name);  
    }  
}
```

با توجه به این که `name` یک ویژگی `private` است، آیا دستور `a.check(b)` دچار خطا می‌شود؟

(با توجه به این که دسترسی به ویژگی خصوصی `name` برای شیء دیگری فراهم شده است)



پاسخ: مخفی برای کلاس

```
public class Access {  
    private String name;  
    public Access(String name) {  
        this.name = name;  
    }  
    public void check(Access access) {  
        access.name = name;  
    }  
    public static void main(String[] args) {  
        Access a = new Access("Ali");  
        Access b = new Access("Taghi");  
        a.check(b);  
        System.out.println(a.name);  
        System.out.println(b.name);  
    }  
}
```

تعیین دسترسی برای کلاسها تعریف می شود
نه برای اشیاء

وقتی یک عضو خصوصی تعریف می شود،
در خارج از کلاس قابل مشاهده نیست
نه این که برای سایر اشیاء قابل مشاهده نباشد



تمرین عملی

- تعریف بسته (package)
- تناظر شاخه‌ها و بسته‌ها
- استفاده از `import` یا نام کامل کلاس‌ها
- Name conflict
- Organize imports
- تعیین سطح دسترسی برای متد، ویژگی و کلاس
- تعریف چند کلاس در یک فایل





مفہوم استاتیک

Static Members

مثال: برنامه نمایشگاه (فروش) خودرو

```
class Peykan{...}
```

```
class Zhian{...}
```

```
public class Pride {
```

```
    int color;
```

```
    int price;
```

```
    boolean isHatchBack;
```

ویژگی‌های نمونه‌ها (اشیاء)

```
static int designYear;
```

```
static int length, width;
```

```
}
```

ویژگی‌های کلاس
(مشترک بین همه اشیاء)



متغیرهای استاتیک (Static Variables)

- یک متغیر استاتیک، در واقع یک ویژگی برای کلاس است
- نه اشیاء

• مثل `Pride.length`

- یک متغیر استاتیک، در بین تمام اشیاء آن کلاس مشترک است
- یک متغیر استاتیک، فقط یک خانه در حافظه دارد
- هر شیء، احتیاج به حافظه مستقلی برای این ویژگی ندارد
- بدون ساختن هیچ شیء می‌توانیم از متغیرهای استاتیک استفاده کنیم
 - با کمک اسم کلاس

• مثلاً: `Pride.length = 393;`

- یک ویژگی غیراستاتیک، به ازای هر شیء یک خانه در حافظه ایجاد می‌کند



متدهای استاتیک

- به طور معمول، هر متد بر روی یک شیء فراخوانی می شود

```
person.getAge()  
book.setName("...")
```

- برخی متدها، به هیچ شیء خاصی از کلاس مرتبط نیستند
- چنین متدهایی بدون ساخت هیچ شیءی باید قابل استفاده (فراخوانی) باشند

• مثال:

```
Pride.setLength(393);  
int max = Person.getMaxAge();  
double eight = Math.pow(2, 3);  
String s = String.valueOf(12);  
public static void main(String[] args)
```

- چنین متدهایی به صورت استاتیک (**static**) تعریف می شوند



مثال: کلاس Pride

```
public class Pride {  
    private int color;  
    public int getColor() {  
        return color;  
    }  
    public Pride(int color) {  
        this.color = color;  
    }  
    public void start(){...}  
  
    private static int Length;  
    public static void setLength(int length) {  
        Pride.Length = length;  
    }  
}
```



مثال: کلاس Person

```
package human;

public class Person {
    private String name;
    private int age;
    public static int MAX_AGE = 150;
    public void setAge(int a) {
        if(a < MAX_AGE)
            age = a;
    }
    public static int getMAX_AGE() {
        return MAX_AGE;
        //no access to age and name
    }
}
```



مثال: نگاهی به کلاس `java.lang.Math`

- (کلاس `Math` با تغییرات جزئی نمایش داده شده است)

```
public class Math {  
    public static double PI = 3.1415926;  
  
    public static double pow(double a, double b) {...}  
    public static int round(float a) {...}  
    public static int abs(int a) {...}  
    public static double max(double a, double b) {...}  
    public static double sqrt(double a) {...}  
}
```

```
double r = 3;  
double area = Math.PI * Math.pow(r, 2);
```



متدهای استاتیک (ادامه)

- متدهای استاتیک در واقع عملیات (رفتارهای) کلاس هستند
 - نه رفتارهای شیء
 - بر روی یک شیء فراخوانی نمی شوند
 - مثال: `Pride.setLength(393);`
- متدهای استاتیک به شیء خاصی دسترسی ندارند
 - پس به ویژگی های اشیاء آن کلاس دسترسی ندارند
- متدهای استاتیک فقط می توانند از ویژگی های استاتیک استفاده کنند
- متدهای معمولی (غیر استاتیک) از همه ویژگی ها می توانند استفاده کنند
 - چه استاتیک چه غیر استاتیک



امکان import static

- می‌دانیم: با کمک import می‌توانیم یک یا چند کلاس را مشخص کنیم
- اگر از این کلاس‌ها بدون ذکر آدرس دقیق (بسته) آن‌ها استفاده کنیم:
- کامپایلر جاوا می‌فهمد منظور ما دقیقاً کدام کلاس است

```
import ir.javacup.*;  
import java.util.Scanner;
```

- با کمک **import static** یک یا چند **عضو استاتیک** را مشخص می‌کنیم
- عضو استاتیک (static member): متد یا متغیر استاتیک
- اگر از این اعضا بدون ذکر آدرس دقیق (بسته و کلاس) آن‌ها استفاده کنیم:
- کامپایلر جاوا می‌فهمد منظور ما دقیقاً کدام عضو است

```
import static java.lang.Math.*;  
import static ir.javacup.oop.Pride.Length;  
import static ir.javacup.oop.Pride.setLength;
```



مثال import static

```
package ir.javacup.oopadeeperlook;

import static java.lang.Math.*;
import static ir.javacup.oop.Pride.Length;
import static ir.javacup.oop.Pride.setLength;

public class StaticImport {
    public static void main(String[] args) {
        double r = 2;
        double area = PI * pow(r, 2);
        setLength(Length+2);
    }
}

class AnotherClass{
    double squareRoot(double d){
        return sqrt(d);
    }
}
```



کوییز

- می‌خواهیم کلاس کتاب (Book) را برای برنامه کتابخانه طراحی کنیم
- فرض کنید ویژگی‌ها و متدهای زیر را برای این کلاس در نظر گرفته‌ایم
- کدام ویژگی‌ها باید استاتیک باشند؟
- کدام متدها باید استاتیک باشند؟
- نام کتاب
- امانت داده شدن
- تعداد صفحه کتاب
- دریافت فهرست همه نویسندگان
- تعداد کتابهای کتابخانه
- بیشترین تعداد صفحات ممکن





آماده‌سازی متغیرهای استاتیک

Static Initialization

آماده‌سازی متغیرهای استاتیک (Static Initialization)

- ویژگی‌های معمولی (غیراستاتیک) آماده‌سازی (Initialize) می‌شوند
- با کمک سازنده (Constructor) و یا از طریق بلوک آماده‌سازی اولیه یا مقداردهی در خط
- متغیرهای استاتیک هم مثل ویژگی‌های معمولی باید مقداردهی اولیه شوند
- تا مقادیر و حالت اولیه معتبری داشته باشند
- روش‌های فوق برای آماده‌سازی ویژگی‌های یک شیء جدید است
- وقتی یک شیء جدید ساخته می‌شود:
- با این روش‌ها ویژگی‌های شیء مقداردهی اولیه می‌شود
- اما ویژگی‌های استاتیک مربوط به یک شیء نیستند (مربوط به کلاس هستند)
- با کمک سازنده یا امکانات مشابه نمی‌توان متغیرهای استاتیک را مقداردهی اولیه کرد



آماده‌سازی متغیرهای استاتیک (ادامه)

- آماده‌سازی اشیاء، به ازای هر شیء جدید باید انجام شود
- آماده‌سازی متغیرهای استاتیک، یک بار برای همیشه انجام می‌شود
- وقتی که کلاس موردنظر اولین بار در برنامه مورد استفاده قرار می‌گیرد
- بخشی از JVM با نام `ClassLoader` این کلاس را بارگذاری می‌کند
- قسمتی از حافظه را به این کلاس اختصاص می‌دهد
- بخشی از این حافظه مربوط به متغیرهای استاتیک این کلاس است
- در همین زمان، ویژگی‌های استاتیک مقداره‌ی اولیه می‌شوند
- چگونه؟!



روش‌های مقداردهی اولیه استاتیک

- دو روش برای مقداردهی اولیه متغیرهای استاتیک:

۱- مقداردهی در خط

```
public static int MAX_AGE = 150;
```

```
private static double PI = 3.14;
```

```
static String defaultName = theDefaultName();
```

```
private static String theDefaultName() {
```

```
    return "Ali Alavi";
```

```
}
```

۲- بلوک استاتیک (Static Block)



بلوک استاتیک (Static Block)

```
public class Person {  
    public static int MAX_AGE ;  
    private static double PI ;  
    static String defaultName ;  
    private static String theDefaultName() {  
        return "Ali Alavi";  
    }  
  
    static{  
        MAX_AGE = 150;  
        PI = 3.14;  
        String s = theDefaultName();  
        if(s != null)  
            defaultName = theDefaultName();  
    }  
}
```



ترتیب مقداردهی اولیه

- یک بار برای هر کلاس:
- مقداردهی در خط به متغیرهای استاتیک
- بلوک استاتیک (static block)
- یک بار به ازای ایجاد هر شیء
- مقداردهی در خط به ویژگی‌ها
- بلوک مقداردهی اولیه (instance initialization block)
- سازنده (Constructor)



کوییز

در برنامه زیر
به ترتیب کدام
مقداردهی‌ها
انجام می‌شود؟

```
public class Person {  
    public static int MAX_AGE ;  
    private static double PI = 3.14;  
    static{  
        MAX_AGE = 150;  
    }  
    private String nation = "Iran";  
    private int age;  
    private String name;  
    {  
        name = "Ali";  
    }  
    public Person() {  
        age = 10;  
    }  
    public Person(int a, String n){  
        age = a;  
        name = n;  
    }  
}
```

```
public static void main(String[] args) {  
    Person p1 =new Person();  
    Person p2 =new Person(20, "Taghi");  
}
```



مرور چند مسأله جالب

مسأله اول

- چگونه کلاسی بنویسیم که:

تعداد اشیاء زنده که از این کلاس ساخته شده را نگهداری کند

- شیء زنده: شیءی که ایجاد شده و هنوز توسط زباله‌روب حذف نشده است

- هدف: می‌خواهیم متدی بنویسیم که تعداد اشیاء زنده این کلاس را برگرداند

- به جزئیات دقت کنید

- چه روشی برای مقداردهی اولیه مناسب است؟

- چگونه به ازای ایجاد هر شیء تعداد را افزایش دهیم

- کدام بخش‌ها `public` باشند و کدام بخش‌ها نباشند؟

- کدام متغیرها و متدها استاتیک باشند و کدام‌ها نباشند؟



```
public class LiveObjects {  
    private static int liveInstances = 0;  
    {  
        liveInstances++;  
    }  
    public static int getLiveInstances() {  
        return liveInstances;  
    }  
    protected void finalize() {  
        liveInstances--;  
    }  
}
```



```
LiveObjects lives = new LiveObjects();  
new LiveObjects();  
new LiveObjects();  
new LiveObjects();  
new LiveObjects();  
new LiveObjects();
```

```
System.out.println(LiveObjects.getLiveInstances());  
// prints 6
```

```
System.gc();  
Thread.sleep(1000);  
System.out.println(LiveObjects.getLiveInstances());  
// prints 1
```



مسأله دوم

- می‌خواهیم کلاسی بنویسیم که ساختن اشیاء جدید از این کلاس غیرممکن باشد!
- فقط یک شیء از این کلاس ایجاد شود
- هر کس به شیء از آن کلاس نیاز دارد، از همان شیء استفاده کند
- و شیء جدیدی نسازد (اصلاً نتواند شیء جدیدی بسازد)
- در این کلاس، متدی تعریف شود که همان شیء را برگرداند
- به جزئیات دقت کنید
- چه روشی برای مقداردهی اولیه مناسب است؟
- چگونه ایجاد شیء جدید را غیرممکن کنیم؟
- کدام بخش‌ها `public` باشند و کدام بخش‌ها نباشند؟
- کدام متغیرها و متدها استاتیک باشند و کدام‌ها نباشند؟



```
public class Singleton {  
    private static Singleton instance = new Singleton();  
    private Singleton(){ }  
    public static Singleton getInstance( ) {  
        return instance;  
    }  
}
```

Singleton s = new Singleton(); ❌

Singleton s = Singleton.getInstance(); ✅

چگونه برنامه فوق را تغییر دهیم که اولین بار که متد `getInstance` فراخوانی شد، همان زمان شیء ساخته شود؟

• الگوی طراحی Singleton

• الگوهای طراحی (Design Patterns)



تمرین عملی

تمرین عملی

- مفهوم استاتیک
- ویژگی‌ها و متدهای استاتیک
- عدم دسترسی متدهای استاتیک به ویژگی‌های غیراستاتیک
- فرایند مقداردهی اولیه
- استاتیک
- شیء
- فراخوانی متدهای استاتیک بر روی یک شیء
- ایجاد یک هشدار (Warning) و نه خطا
- اتفاقی که در واقع می‌افتد: فراخوانی روی کلاس
- حتی اگر شیء موردنظر null باشد، خطایی رخ نمی‌دهد





کلیدواژه this
What is this?!

کلیدواژه this

- هر متد معمولی (غیراستاتیک) روی یک شیء فراخوانی می شود
- و رفتاری از آن شیء را اجرا می کند
- مثال: `circle.getArea()` ;
- یک ارجاع به شیءی که متد روی آن فراخوانی شده، هنگام فراخوانی این متد، به آن پاس می شود (مثل یک پارامتر پنهانی)
- این ارجاع (پارامتر پنهانی) ، **this** نام دارد
- ارجاع **this** در متدهای غیراستاتیک قابل استفاده است
- ارجاع **this** به شیء اشاره می کند که متد روی آن فراخوانی شده است



کاربرد this

- برای تمایز متغیرهای محلی که هم نام یک ویژگی (property) هستند

```
public class Book {  
    private String name;  
    private Person author;  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public void setAuthor(Person author) {  
        this.author = author;  
    }  
}
```



نمونه دیگری از کاربرد this

- این رویکرد (فراخوانی آبشاری) بین برنامه‌نویسان C++ بسیار محبوب است
- بین برنامه‌نویسان جاوا کمتر استفاده می‌شود

```
public class Leaf {  
    int i = 0;  
    Leaf increment() {  
        i++;  
        return this;  
    }  
    void print() {  
        System.out.println("i = " + i);  
    }  
    public static void main(String[] args) {  
        Leaf x = new Leaf();  
        x.increment().increment().increment().print();  
    }  
}
```



یادآوری: کاربرد this برای فراخوانی سازنده دیگر

- البته در این کاربرد، this مانند یک پارامتر ضمنی نیست

```
public class Country {  
    private String name;  
    private int population;  
  
    public Country(int number) {  
        population = number;  
    }  
    public Country(String n, int number) {  
        this(number);  
        name = n;  
    }  
}
```



متدهای استاتیک و `this`

- ارجاع `this` به شیء اشاره می‌کند که متد روی آن فراخوانی شده است
- متدهای استاتیک روی یک شیء فراخوانی نمی‌شوند
- روی کلاس فراخوانی می‌شوند
- بنابراین متدهای استاتیک به `this` دسترسی ندارند
- استفاده از `this` در یک متد استاتیک:
- به خطای کامپایل منجر می‌شود



کوییز


```
package ir.javacup.oop;
```

```
class That{
```

```
String name = "A";
```

```
public static void test(This name) {
```

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

```
}
```

```
}
```

```
public class This {
```

```
String name = "B";
```

```
public void test(String name){
```

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

```
    That.test(this);
```

```
}
```

```
public static void main(String[] args) {
```

```
    new This().test("C");
```

```
}
```

```
}
```

سؤال:

– اسم این فایل جاوا؟
– خروجی این برنامه؟



تمرین عملی

- استفاده از `this`
- برای اشاره به یک ویژگی (property)
- برای تمایز یک ویژگی از یک متغیر محلی
- برای فراخوانی یک متد
- عدم دسترسی متدهای استاتیک به `this`



جمع بندی

- بسته (package)

- سطوح دسترسی

- Access Levels: public, private, package access
- Access Specifiers

- مفهوم استاتیک (static)

- متغیرهای استاتیک

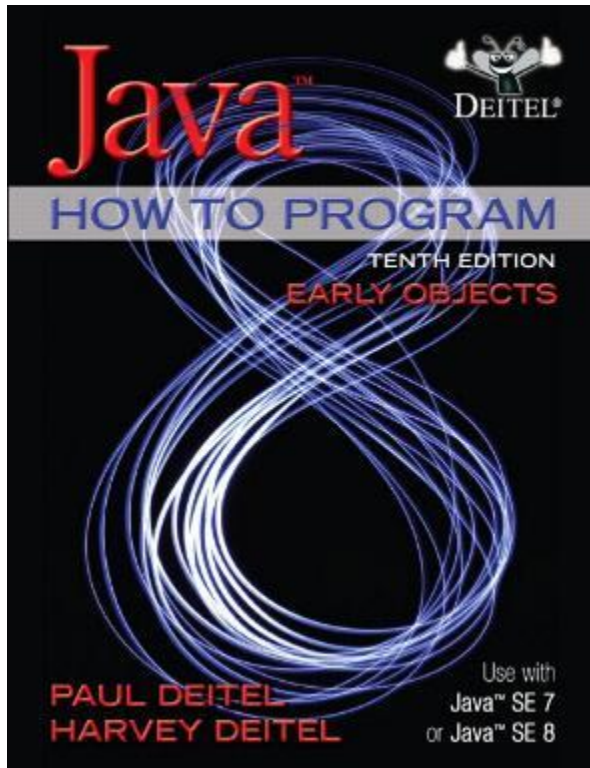
- متدهای استاتیک

- کلیدواژه this



- فصل‌های ۶ و ۸ کتاب دایتل

Java How to Program (Deitel & Deitel)



- 6- Methods: A Deeper Look
- 8- Classes and Objects: A Deeper Look

- تمرین‌های همین فصل‌ها از کتاب دایتل



تمرین: پیاده‌سازی دو کلاس Date و Person

- با توجه به نام شرکت خودتان (یا یک نام فرضی) بسته‌هایی را برای این تمرین طراحی کنید
- برای هر کلاس سازنده (یا سازنده‌ها) و getter و setter های مناسب ایجاد کنید
- و آن را در بسته مناسب مجزایی قرار دهید
- خواص هر متد و متغیر را با دقت تعیین کنید (استاتیک؟ public؟ ...)
- کلاس **Date**: هر «تاریخ» (Date) شامل ویژگی‌های روز و ماه و سال است
- حداقل زمان سال ۱۹۷۰ و حداکثر سال ۲۱۰۰ باشد (این اعداد قابل تنظیم باشند)
- کلاس **Person**
- امکان تعیین تاریخ تولد با کمک کلاس Date
- سن کسی بیشتر از ۱۵۰ سال نشود (حداکثر ۱۵۰ در متغیری قابل تنظیم باشد)





جستجو کنید و بخوانید

- موضوعات پیشنهادی برای جستجو:
- چرا Constructor در کلاس Math به صورت private است؟!
- چه مزایایی برای تعریف getter و setter وجود دارد؟
- کلاس‌هایی مانند Integer و String اصطلاحاً immutable هستند.
- یعنی چه؟ چرا؟
- چگونه برنامه‌ای که شامل بسته‌های مختلف است را کامپایل و اجرا کنیم؟
- مفهوم Namespace در زبان‌های برنامه‌نویسی دیگر
- مثل C++ و C#
- الگوهای طراحی (Design Patterns)
- Singleton
- به چه شکل‌های دیگری می‌توان الگوی Singleton را پیاده کرد؟



پایان

تاریخچه تغییرات

نسخه	تاریخ	توضیح
۱.۰.۰	۱۳۹۴/۳/۱۰	نسخه اولیه ارائه آماده شد

