

انجمن جاوا کا پتھیم می کند

دوره برنامه نویسی جاوا

تولد و مرگ اشیاء

Objects Initialization and Cleanup

صادق علی اکبری

- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- بازنشر 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، بازنشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست



# سرفصل مطالب

- چگونه یک شیء به دنیا می آید؟
- وضعیت اولیه یک شیء چگونه ایجاد می شود؟
  - فرایند ایجاد اولیه اشیاء (Initialization)
  - سازنده (Constructor)
  - فرایند مرگ اشیاء (Cleanup)





# آماده‌سازی اولیه اشیاء Object Initialization

# آماده‌سازی اولیه اشیاء (Initialization)

- یک شیء، بعد از ساخته شدن، لزوماً یک شیء آماده استفاده نیست
- ممکن است هنوز یک شیء نامعتبر باشد

● مثال: `Person p = new Person();`

- شیء `p` که به آن اشاره می‌کند، احتمالاً معتبر نیست
- این شیء هیچ یک از ویژگی‌های لازم برای یک «فرد» را ندارد
- مثلاً هنوز نام این فرد مشخص نشده است
- این شیء باید آماده‌سازی اولیه شود
- مقداردهی اولیه (Initialization)



# راهکار بدوی برای آماده‌سازی اولیه اشیاء

```
public class Student {
    //Mandatory
    private String name;
    private long id;
    //Optional
    private String homepage;
    public void setName(String s) {
        name = s;
    }
    public void setId(long idValue) {
        id = idValue;
    }
    public void setHomepage(String addr) {
        homepage = addr;
    }
    public void init(String name, long id) {
        setName(name);
        setId(id);
    }
}
```

● ایجاد یک متد (مثلاً `init`)



# استفاده از متد `init`

```
public static void main(String[] args) {  
    Student st = new Student();  
    // ارجاع st هنوز به یک شیء نامعتبر اشاره می کند  
    st.init("Hossein Alizadeh", 45205068);  
    // حالا st به شیء معتبری اشاره می کند و آماده استفاده است  
    System.out.println(st.getName());  
    System.out.println(st.getId());  
}
```



# مثال‌های دیگر

```
Circle c = new Circle ();
```

```
c.init(12);
```

```
Book b1 = new Book ();
```

```
b1.init("من او", "رضا امیرخانی");
```

```
Book b2 = new Book ();
```

```
b2.init("شاهنامه", "ابوالقاسم فردوسی");
```





# اشکال راهکار استفاده از متد `init` چیست؟

- این متد باید به صورت دستی (`manual`) فراخوانی شود
- برنامه‌نویس ممکن است فراموش کند و آن را فراخوانی نکند
  - تضمینی برای اجرای این متد وجود ندارد
- قبل از فراخوانی این متد، شیء در حالت نامعتبر است
- راهکار مطلوب:
- فراخوانی این متد به صورت خودکار هنگام ایجاد شیء انجام شود





# سازنده (Constructor)

# سازنده (Constructor)

```
public class Circle {  
    private double radius;  
  
    public Circle(double r) {  
        radius = r ;  
    }  
}
```

- سازنده، یک متد خاص است
- نام سازنده دقیقاً همان نام کلاس است
- و نوع برگشتی ندارد (هیچ نوعی، حتی void)
- به جای متدی مثل `init` استفاده می شود
- زمانی که شیء ایجاد می شود، سازنده به صورت خودکار فراخوانی می شود
- مثلاً وقتی که با کمک `new` شیء جدیدی می سازیم
- به این ترتیب شیء همواره، از بدو تولد، در حالت معتبر خواهد بود



# نحوه کار سازنده (Constructor)

```
public class Circle {  
    private double radius;  
    public double getArea() {  
        return radius*radius * 3.14;  
    }  
  
    public Circle(double r) {  
        radius = r ;  
    }  
  
    public static void main(String[] args) {  
        Circle c = new Circle(2);  
        System.out.println(c.getArea());  
    }  
}
```



# سازنده‌هایی با پارامتر

- یک سازنده ممکن است پارامترهایی داشته باشد
- یک کلاس می‌تواند سازنده‌های مختلفی داشته باشد
  - به شرطی که پارامترهای متفاوتی داشته باشند
- اگر برنامه‌نویس، هیچ سازنده‌ای برای یک کلاس تعریف نکند، کامپایلر جاوا به صورت خودکار یک سازنده پیش‌فرض برای آن کلاس در نظر می‌گیرد
  - سازنده پیش‌فرض (Default constructor) هیچ پارامتری ندارد
  - بدنه این سازنده خالی است (در واقع کارهایی می‌کند که بعدها خواهیم دید)
- وقتی برنامه‌نویس اولین سازنده را تعریف می‌کند: جاوا سازنده پیش‌فرضی برای این کلاس اضافه نمی‌کند



```
public class Circle {  
    private double radius;  
    public double getArea(){  
        return radius*radius*3.14;  
    }  
    public Circle(double r) {  
        radius = r;  
    }  
    public Circle() {  
    }  
}
```

```
Circle c;  
c = new Circle();  
c = new Circle(12);
```



# کلاسی بدون سازنده پیش فرض

```
public class Circle {  
    private double radius;  
    public double getArea() {  
        return radius * radius * 3.14;  
    }  
    public Circle(double r) {  
        radius = r;  
    }  
    public static void main(String[] args) {  
        Circle c;  
        c = new Circle(12);  
        c = new Circle();  
    }  
}
```



*Compile Error:*

*The Constructor Circle() is undefined*



# وظیفه سازنده

```
public class Car {
    private Engine engine;
    private Tyre[] tyres;
    public Car() {
        engine = new Engine();
        tyres = new Tyre[4];
        for (int i = 0; i < tyres.length; i++) {
            tyres[i] = new Tyre();
        }
    }
}
```

- هرگاه یک شیء جدید ساخته می شود، سازنده اش فراخوانی می شود

- سازنده، ویژگی های شیء را مقداردهی اولیه می کند

- در صورت لزوم، ویژگی هایی که خودشان شیء هستند را هم ایجاد می کند





# مقادیر پیش فرض

- اگر سازنده مقدار یک ویژگی (Property) را مشخص نکند، چه می شود؟
- در این صورت، هر ویژگی مقدار پیش فرض نوع داده خودش را می گیرد
- مثلاً یک ویژگی از نوع `int`، مقدار صفر می گیرد
- مقادیر پیش فرض انواع مختلف داده:
  - مقدار پیش فرض `boolean` : `false`
  - مقدار پیش فرض بقیه انواع داده اولیه (مثل `int`، `char` و `long`) : صفر
  - مقدار پیش فرض متغیرهای ارجاعی (اشیاء) : `null`



کوییز

# خروجی این برنامه چیست؟

```
public class ConstructorQuiz {
    private int number;
    private double real;
    private boolean condition;
    private String name;
    private Circle circle;
    public ConstructorQuiz(int num, String title) {
        number = num;
        name = title;
    }
    public static void main(String[] args) {
        ConstructorQuiz q = new ConstructorQuiz(5, "Ali");
        System.out.println(q.number);
        System.out.println(q.real);
        System.out.println(q.condition);
        System.out.println(q.name);
        System.out.println(q.circle);
    }
}
```

پاسخ صحیح

5  
0.0  
false  
Ali  
null



# خروجی این برنامه چیست؟

```
public class ConstructorQuiz {
    private int number;
    private double real;
    private boolean condition;
    private String name;
    private Circle circle;
    public ConstructorQuiz(int num, String title) {
        number = num;
        name = title;
    }
    public static void main(String[] args) {
        ConstructorQuiz q = new ConstructorQuiz();
        System.out.println(q.number);
        System.out.println(q.real);
        System.out.println(q.condition);
        System.out.println(q.name);
        System.out.println(q.circle);
    }
}
```

پاسخ صحیح: خطای کامپایل



چند نکته دیگر درباره تولد اشیاء

# سایر روش‌های مقداردهی اولیه

- دو روش مهم دیگر برای مقداردهی اولیه ویژگی‌های یک شیء
- (به غیر از سازنده)

۱- مقداردهی اولیه درخط (Inline Initialization)

```
public class Car {  
    private Engine engine = new Engine();  
    private int numberOfTyres = 4;  
    private Tyre[] tyres = new Tyre[numberOfTyres];  
  
    public Car() {  
        for (int i = 0; i < tyres.length; i++) {  
            tyres[i] = new Tyre();  
        }  
    }  
}
```

۲- بلوک مقداردهی اولیه (Initialization Block)



# بلوک مقداردهی اولیه (Initialization Block)

- یک (یا چند) بلوک بدون نام که در میان تعریف کلاس قرار می‌گیرد

```
public class Car {  
    private int numberOfTyres = 4;  
    private Tyre[] tyres;
```

Initialization Block

```
{  
    tyres = new Tyre[numberOfTyres];  
    for (int i = 0; i < tyres.length; i++) {  
        tyres[i] = new Tyre();  
    }  
}
```

- هر گاه یک شیء جدید ایجاد شود، بلوک مقداردهی اولیه اجرا می‌شود



# ترتیب مقداردهی اولیه

- ۱- همه مقداردهی‌های در خط اجرا می‌شوند
- ۲- همه بلوک‌های مقداردهی اولیه اجرا می‌شوند
  - معمول نیست که یک کلاس چند بلوک مقداردهی اولیه داشته باشد
  - (البته ممکن است)
- ۳- یکی از سازنده‌ها اجرا می‌شود
  - کدام یک؟
  - همان که فراخوانی شده
  - تعدد سازنده‌ها کاملاً عادی و در موارد لزوم، رایج است





# کاربرد this برای سازنده‌ها

- گاهی لازم است که یک سازنده، سازنده دیگری را فراخوانی کند
- به خصوص از منظر استفاده مجدد از کد (Code reuse)
- تا کدی که در یک سازنده نوشته شده، در سازنده دیگر تکرار (کپی) نشود
- یک سازنده، با کمک کلیدواژه **this** می‌تواند سازنده دیگری را فراخوانی کند
- در صورت وجود، این فراخوانی باید حتماً اولین دستور سازنده باشد
- مشخص می‌کنیم که دقیقاً کدام سازنده‌ی دیگر باید فراخوانی شود
- با کمک پارامترهای **this**
- کلیدواژه **this** کاربردهای دیگری هم دارد که بعداً خواهیم دید



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



# خلاصه مقداردهی اولیه

- برای هر ویژگی، به مقداردهی اولیه لازم دقت کنید
- و روش (یا روش‌های) مقداردهی اولیه مناسب را انتخاب کنید
- اگر مقداردهی، ساده و در حد یک مقدار مشخص است
- از مقداردهی در خط (inline initialization) استفاده کنید
- اگر یک مجموعه کد برای آماده‌سازی اولیه،  
در همه سازنده‌ها تکرار می‌شود  
و نیاز به پارامتر خاصی ندارد
- از بلوک مقداردهی اولیه (initialization block) استفاده کنید
- اگر مقداردهی اولیه، به پارامتر نیاز دارد: از سازنده استفاده کنید



کوییز

## خروجی قطعه برنامه زیر چیست؟

```
public class Quiz {
    public int number = f();
    private int f() {
        System.out.println("Inline Initialization"); return 1;
    }
    {System.out.println("Initialization Block"); number = 2;}
    public Quiz() {
        System.out.println("NO-arg constructor"); number = 3;
    }
    public Quiz(int num) {
        System.out.println("ONE-arg constructor"); number = num;
    }
}
```

```
Quiz q = new Quiz();
System.out.println(q.number);
q = new Quiz();
System.out.println(q.number);
q = new Quiz(7);
System.out.println(q.number);
```

```
Inline Initialization
Initialization Block
NO-arg constructor
3
Inline Initialization
Initialization Block
NO-arg constructor
3
Inline Initialization
Initialization Block
ONE-arg constructor
7
```

## خروجی قطعه برنامه زیر چیست؟

```
public class Quiz {
    public int number = f();
    private int f() {
        System.out.println("Inline Initialization"); return 1;
    }
    {System.out.println("Initialization Block"); number = 2;}
    public Quiz() {
        System.out.println("NO-arg constructor"); number = 3;
    }
    public Quiz(int num) {
        this();
        System.out.println("ONE-arg constructor"); number = num;
    }
}
```

```
Quiz q = new Quiz();
System.out.println(q.number);
q = new Quiz(7);
System.out.println(q.number);
```

```
Inline Initialization
Initialization Block
NO-arg constructor
3
Inline Initialization
Initialization Block
NO-arg constructor
ONE-arg constructor
7
```

# فرایند مرگ اشیاء

# متد finalize()

- هر چند جاوا Destructور ندارد،  
برای هر کلاس متد ویژه‌ای با نام **finalize** قابل تعریف است
- هرگاه زباله‌روب یک شیء را آزاد کند،  
متد **finalize** از این شیء را فراخوانی می‌کند
- اگر زباله‌روب یک شیء را حذف نکند:  
هرگز متد **finalize** برای این شیء فراخوانی نمی‌شود





# مثال برای متد finalize()

```
public class Circle {  
    private double radius;  
    public Circle(double r) { radius = r; }  
    public void finalize() {  
        System.out.println("Finalize...");  
    }  
    public static void main(String[] args) {  
        f();  
        System.gc();  
    }  
    private static void f() {  
        Circle c = new Circle(2);  
        System.out.println(c.radius);  
    }  
}
```

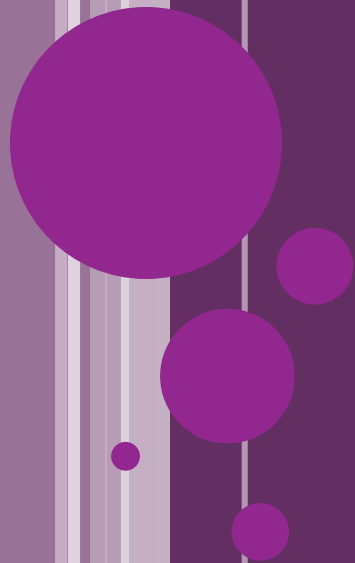


# درباره finalize

- چه نیازی به متد finalize است؟
- مگر زباله‌روب مرگ شیء را مدیریت نمی‌کند؟
- زباله‌روبی (Garbage Collection) فقط درباره حافظه است
- گاهی منابعی غیر از حافظه باید آزاد شود
- زباله‌روب این کار را نمی‌کند
- پیاده‌سازی finalize کار رایجی نیست و کاربردهای خاصی دارد



# تمرین عملی



# تمرین عملی

---

- ترتیب اجرا
- `this` باید اولین دستور سازنده باشد



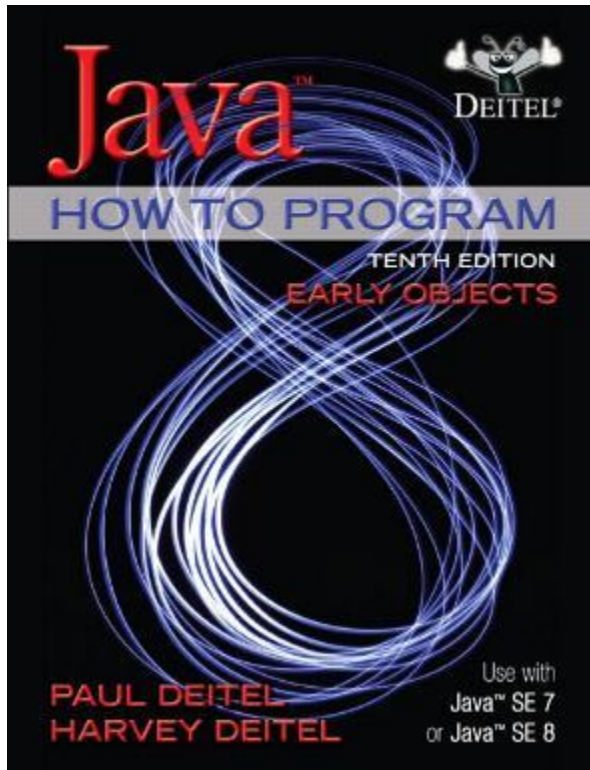
جمع بندی

- فرایند مقداردهی اولیه اشیاء
- حالت هر شیء، هنگام ایجاد شدن، باید تنظیم شود
- مقدار اولیه مناسب برای ویژگی‌های شیء تعیین شود
- مفهوم سازنده (Constructor)
- ترتیب اجرای بخش‌های مختلف مقداردهی اولیه
- مرگ اشیاء
- زباله‌روب، Destructor (نداریم) و متد `finalize`



- بخش‌هایی از فصل‌های ۳ و ۸ از کتاب دایتل

## Java How to Program (Deitel & Deitel)



- 3- Introduction to Classes, Objects, Methods and Strings
- 8- Classes and Objects: A Deeper Look

- تمرین‌های همین فصل‌ها از کتاب دایتل



- کلاس فوتبالیست را تعریف کنید
- طوری تعریف کنید که فرایند مقداردهی اولیه مناسبی داشته باشد
- از امکانات زیر در آن استفاده کنید:
- Inline Initialization
- Initialization Block
- سازنده‌های مناسب
- از این کلاس اشیاء مختلفی ایجاد کنید
- و صحت مقداردهی ویژگی‌های این اشیاء را بررسی کنید
- کلاس‌های دیگری به همین ترتیب تعریف کنید و از آن‌ها اشیاء بسازید
- مثلاً کلاس‌های کتاب، ماشین و ...





# جستجو کنید و بخوانید

- کلمات و عبارات پیشنهادی برای جستجو:
- Constructor and Destructor
- **finalize()** method applications
- Constructor Overloading
- java constructor best practices



پایان

# تاریخچه تغییرات

نسخه	تاریخ	توضیح
۱.۰.۰	۱۳۹۴/۳/۵	نسخه اولیه ارائه آماده شد

