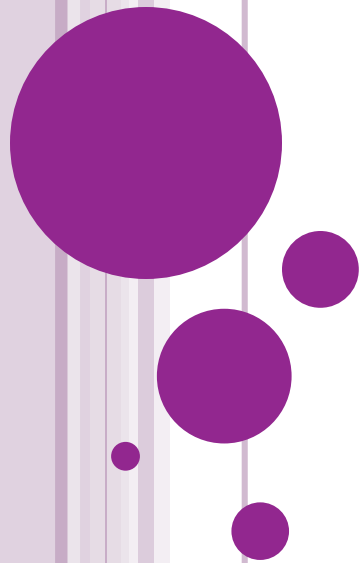


انجمن جاوا کا پتہ قدم می کند

دوره برنامه نویسی جاوا

چند داستان کوتاه درباره امکانات جاوا
Java Short Stories

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

- متدهایی با تعداد متغیر پارامتر (Variable Argument List)
- کلاس‌های لفاف انواع اولیه (Primitive Wrapper Classes)
- سربار کردن متدها (overloading)
- متد toString
- متد equals
- متغیرهای ثابت (final variables)
- اشیاء تغییرناپذیر (Immutable)
- انواع داده شمارشی (enum)





متدهایی با تعداد متغیر پارامتر Variable Argument List

تعداد متغیر پارامتر

- امکانی در زبان جاوا تحت عنوان `varargs` وجود دارد:

متدهای تعریف کنیم که از یک آرگومان، صفر یا چند پارامتر بپذیرند

- مثال: `void print(String... args){...}`

- هنگام فراخوانی متد `print` می‌توانیم صفر یا چند رشته به آن پاس کنیم

- یعنی همه فراخوانی‌های زیر صحیح هستند:

```
print();
```

```
print("Ali");
```

```
print("A", "B", "C", "D");
```



نحوه تعریف فهرست متغیر پارامترها (varargs)

- هنگام تعریف متدی که شامل پارامتر varargs می‌شود:
- این پارامتر به شکل یک آرایه قابل استفاده است
- با توجه به نحوه فراخوانی متد (تعداد پارامترها)، این آرایه مشخص می‌شود

```
static void print(String... params) {  
    String[] array = params;  
    System.out.println(array.length);  
    for (String p : params) {  
        System.out.println(p);  
    }  
}
```

print(); → *array.Length==0*
print("Ali"); → *array.Length==1*
print("Ali", "Taghi"); → *array.Length==2*



تفاوت پارامتر آرایه و پارامتر varargs

این دو متد چه تفاوتی دارند؟
`String[] array = {"A", "B"};`

`void print1(String[] args) {...}`

`void print2(String... args) {...}`

متد اول فقط به یک شکل قابل فراخوانی است:

`print1(array);`

فراخوانی متد دوم به همه اشکال زیر صحیح است (دست کاربر باز است)

`print2();` `print2("Ali", "Taghi");`

`print2("Ali");` `print2(array);`



● فرض کنید: `String s;`

● کدام یک از متدهای زیر پارامتر `varargs` دارند؟

```
s = String.format("[%s=%5.2f]", "PI", 3.14);
```

```
s = String.valueOf(3.14);
```

```
s = "pi=3.14".replace("pi", "PI");
```





کلاس‌های لفاف انواع اولیه

Primitive Wrapper Classes

کلاس‌های لفاف انواع اولیه

- انواع داده اولیه (primitive data types) را می‌شناسیم
- byte, short, int, long, float, double, char, boolean

- می‌دانیم متغیرهایی که از این انواع هستند، شیء نیستند
- به یک شیء اشاره نمی‌کنند،
- بلکه مستقیماً یک مقدار را نگهداری می‌کنند

Primitive type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

- متناظر هر نوع اولیه یک کلاس تعریف شده

Primitive Wrapper Classes

- هرگاه که یک شیء مورد نیاز باشد
- از آن‌ها به جای انواع اولیه استفاده می‌کنیم
- امکان مهم: اشیاء این کلاس‌ها، برخلاف انواع داده اولیه، می‌توانند null باشند



```
Double n = new Double(12.2);  
double d = n.doubleValue();  
int i = n.intValue();  
double max = Double.MAX_VALUE;  
  
Integer a = new Integer(12);  
int maxint = Integer.MIN_VALUE;
```



unboxing و autoboxing

● از نسخه ۱.۵ به بعد (Java 5+) این دو امکان به وجود آمده است

● autoboxing :

- اگر یک مقدار primitive به عنوان یک شیء استفاده شود:
- به صورت خودکار به شیء از نوع متناظر wrapper تبدیل می شود

● مثال: `Integer i = 2;`

● unboxing : فرایند برعکس autoboxing

- اگر یک شیء از نوع wrapper به عنوان یک primitive استفاده شود:
- به صورت خودکار به یک مقدار از نوع متناظر primitive تبدیل می شود

● مثال: `int askajd = new Integer(12);`



```
Integer i = new Integer(2);  
Integer j = new Integer(2);  
i = j; //Reference Assignment  
i = 2; //OK. Autoboxing.  
Long l = 2; //Syntax Error. Why?  
Long l = 2L; //OK  
l = i; //Syntax Error. Why?  
System.out.println(i==j);  
//Prints false. Why?
```





سربار کردن متد
Method Overloading

سربار کردن متد (Method Overloading)

- در یک کلاس، می‌توانیم متدهای مختلفی با نام یکسان تعریف کنیم
- به شرطی که مجموعه پارامترهای متفاوتی داشته باشند
- به این کار سربار کردن متد می‌گویند
- همه متدهایی که سربار شده‌اند، قابل استفاده هستند



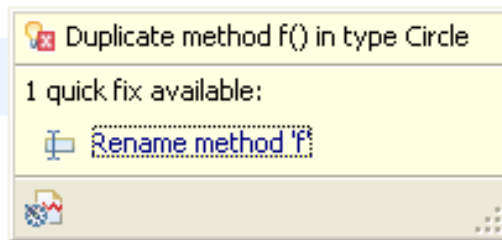
```
void f(){
    System.out.println("f is called");
}
void f(int number){
    System.out.println("f is called with number = " + number);
}
void f(String s){
    System.out.println("f is called with s = " + s);
}
void f(String s, int number){
    System.out.println(
        "f is called with s = " + s + ", number =" + number);
}
public static void main(String[] args) {
    Circle circle = new Circle();
    circle.f();
    circle.f(5);
    circle.f("salam");
    circle.f("salam", 7);
}
```



سربار: فقط براساس تفاوت در پارامترها ممکن است

- چرا براساس مقدار برگشتی نمی‌توانیم متدها را سربار کنیم؟

```
void f() {  
    System.out.println("f is called");  
}  
int f() {  
    System.out.println("another f");  
}
```



```
int f(){return 0;}  
void f(int a){}
```

- اما این حالت اشکالی ندارد:





متد toString

تبدیل به رشته

- در بسیاری از مواقع نیازمند تبدیل یک شیء به رشته هستیم
- مثلاً برای چاپ یا نمایش اطلاعات یک شیء
- یا برای ذخیره آن در فایل
- تبدیل محتوای یک شیء به یک رشته، سناریویی پرکاربرد است
- آیا جاوا اجازه‌ی این تبدیل را می‌دهد؟

مثلاً: `Person person = new Person("Ali", 25);`
`String s = person;`

و یا `Integer number = new Integer(12);`
`String s = number;` البته که نه!

• جاوا در تبدیل نوع بسیار سخت‌گیر است



راه حل: متد toString

- اگر شیءی قرار است به رشته تبدیل شود،

کلاس آن باید متد toString را پیاده‌سازی کند

```
Person person = new Person("Ali", 25);  
String s = person.toString();
```

```
Integer number = new Integer(12);  
String s = number.toString();
```

- toString متد ویژه‌ای است

- همه کلاس‌ها این متد را دارند، حتی اگر برای آن‌ها تعریف نشده باشد

- اما پیاده‌سازی صحیح آن را برای کلاس‌های جدید باید تعریف کنیم



خروجی این برنامه؟ حالا چطور؟

```
package ir.javacup.oop;

public class Circle {
    private double radius;
    public Circle(double radius) {
        this.radius = radius;
    }

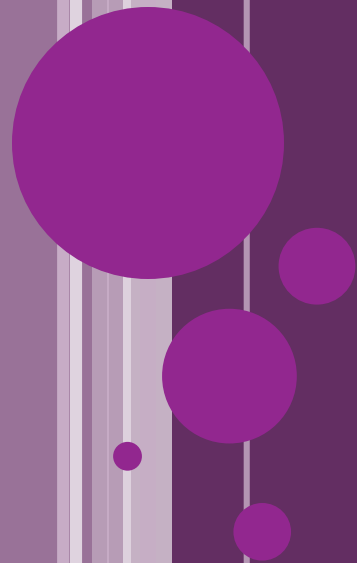
    public String toString() {
        return "Circle [radius=" + radius + "]";
    }

    public static void main(String[] args) {
        Circle c = new Circle(2.0);
        System.out.println(c.toString());
    }
}
```

```
ir.javacup.oop.deeperlook.Circle@15db9742
```

```
Circle [radius=2.0]
```





متد equals

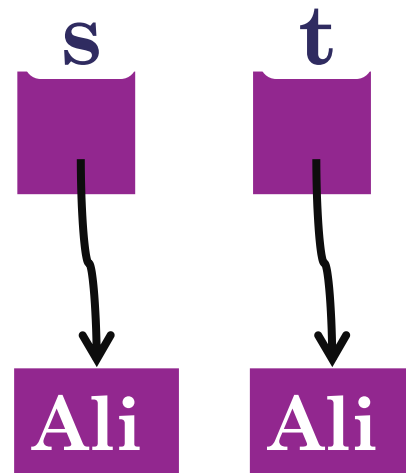
بررسی تساوی دو مقدار

- فرض کنید می‌خواهیم دو مقدار یا دو متغیر a و b را مقایسه کنیم
- و تساوی این دو را بررسی کنیم
- اگر این متغیرها از انواع اولیه (primitive) باشند
 - عملگر `==` مناسب است
 - تبصره: در مقایسه اعداد اعشاری با عملگر `==` ملاحظات لازم است
- اگر این متغیرها ارجاع (Reference) باشند
 - عملگر `==` تساوی ارجاع‌ها (اشاره‌گرها) را بررسی می‌کند
 - مهم: عملگر `==` تساوی محتوای دو شیء را بررسی نمی‌کند
 - عملگر `==` هویت (identity) را بررسی می‌کند، نه وضعیت (حالت یا state)



مقایسه تساوی اشیاء

```
String s = new String("Ali");  
String t = new String("Ali");  
if(s == t)  
    System.out.println("s==t");
```



• پس چگونه محتوای دو شیء را مقایسه کنیم؟

• مقایسه‌ی حالت یا وضعیت یا state

• راه حل: استفاده از متد equals

```
if (s.equals(t))
```

```
    System.out.println("s equals t");
```



متد equals

- بسیاری از کلاس‌های مهم جاوا، متد **equals** مناسبی دارند
 - که تساوی محتوای دو شیء را بررسی می‌کند
 - مثل `String` و کلاس‌های لفاف انواع اولیه (`Character`، `Integer` و ...)
 - وقتی کلاس جدیدی تعریف می‌کنیم:
 - می‌توانیم متد **equals** مناسبی برای آن پیاده‌سازی کنیم
- متد **equals** یک شیء به‌عنوان پارامتر می‌گیرد و `boolean` برمی‌گرداند
 - ویژگی‌های خودش را با ویژگی‌های شیء پارامتر مقایسه می‌کند
 - حالت (وضعیت یا `state`) پارامتر با `this` مقایسه می‌شود



تعریف متد equals

```
public class Person {  
    private String nationalID;  
    private String name;  
    private String email;  
    private int age;
```

راستش را بخواهید این تعریف غلط است!
تعریف equals کمی پیچیده تر است
پارامترش باید از جنس Object باشد

```
public boolean equals(Person other) {  
    return nationalID.equals(other.nationalID);  
}
```

در این باره بعداً بیشتر
صحبت می کنیم

```
}  
Person p1 = new Person("1290786547", "Ali Alavi");  
Person p2 = new Person("1290786547", "Taghi Taghavi");  
Person p3 = new Person("0578905672", "Taghi Taghavi");  
System.out.println(p1.equals(p2));  
System.out.println(p2.equals(p3));
```



مثال و نکته

```
String str1 = new String("Ali");  
String str2 = new String("Ali");  
String str3 = "Ali";  
String str4 = "Ali";
```

همه این اشیاء با هم equal هستند

```
str1 == str2  
str2 == str3  
str3 == str4
```

```
Integer int1 = new Integer(2);  
Integer int2 = new Integer(2);  
Integer int3 = 2;  
Integer int4 = 2;
```

همه این اشیاء با هم equal هستند

autoboxing

```
int1 == int2  
int2 == int3  
int3 == int4
```





متغیرهای ثابت
Final Variables

متغیرهای ثابت (final)

- برخی از متغیرها یک بار مقدار می‌گیرند و هرگز تغییر نمی‌کنند
- به این متغیرها ثابت (constant) گفته می‌شود
- مثال: `Integer.MAX_VALUE` و `Math.PI`
- در جاوا متغیرهای ثابت با کلیدواژه `final` مشخص می‌شوند
- مقدار یک متغیر ثابت (final) قابل تغییر نیست
- اگر متغیر ثابت از انواع داده اولیه باشد: مقدارش قابل تغییر نیست
- اگر متغیر ثابت، یک شیء باشد: دیگر به شیء دیگری نمی‌توان ارجاع دهد



```
final int i = 2;
```

```
i = 3; 
```


مقدار متغیرهایی از انواع اولیه (primitive) غیرقابل تغییر است

```
final Person p1 = new Person();
```

```
Person p2 = new Person();
```

```
p1 = p2; 
```

```
p1 = new Person(); 
```

```
p1.setName("Ali"); 
```

هویت یک شیء ثابت قابل تغییر نیست

وضعیت (ویژگی‌ها، محتوا) یک شیء ثابت قابل تغییر است



اشکال متغیرهای ثابت

● متغیرهای ثابت به شکل‌های مختلفی دیده می‌شوند:

```
class SomeClass{
    private final String name;
    public final int val = 12;
    void f(final int a){
        final int b = a+1;;
    }
    void g(){
        final String s = "123";
    }
    public SomeClass(String name) {
        this.name = name;
    }
}
```

● پارامتر ثابت

● متغیر محلی ثابت

● ویژگی ثابت

● متغیر استاتیک ثابت

● هر متغیر ثابت، باید بلافاصله مقداردهی شود

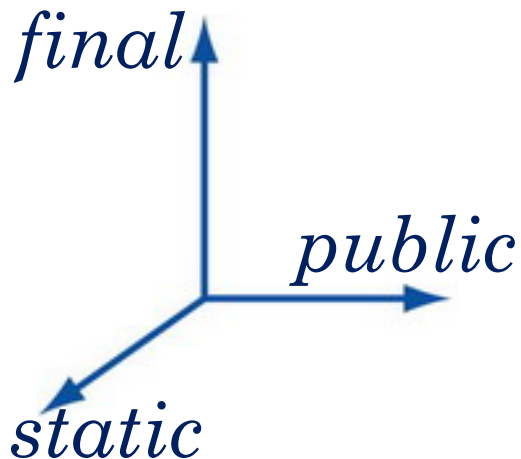
● مثلاً یک ویژگی ثابت، باید در فرایند مقداردهی اولیه شیء، مقداردهی شود

○ مثلاً در سازنده



- این مفاهیم مستقل از هم هستند:
- سطح دسترسی (public, private, package access)
- استاتیک بودن یا نبودن
- ثابت (final) بودن یا نبودن
- مثلاً هر متغیر استاتیک:

- ممکن است final باشد یا نباشد
- ممکن است public باشد یا نباشد



A decorative vertical stripe on the left side of the slide, composed of several thin, parallel lines in shades of purple and white. To the right of the stripe, there are several overlapping circles of varying sizes, all in a dark purple color.

اشياء تغيير ناپذير

Immutable Objects

تغییرپذیری اشیاء

- اشیاء به دو دسته تقسیم می‌شوند: **تغییرپذیر** و **تغییرناپذیر**
- **Mutable & Immutable**
- ویژگی‌های اشیاء **تغییرناپذیر** بعد از ساخت این اشیاء قابل تغییر نیست
- اشیاء **تغییرپذیر** متدهایی دارند که امکان تغییر وضعیت آن‌ها را می‌دهد
 - مثلاً `setter` دارند
- موضوع «**تغییرناپذیری**» با «**ثابت بودن**» متفاوت است
 - **ثابت بودن** درباره ثبات هویت است و با کلیدواژه **final** مشخص می‌شود
 - **تغییرناپذیری** درباره ثبات وضعیت (`state`) است
- **تغییرناپذیری** یک مفهوم است و کلیدواژه خاصی ندارد



اشیاء تغییرناپذیر (Immutable Objects)

- اشیاء تغییرناپذیر مزایایی دارند

- ساده‌تر هستند

- فهمشان آسان‌تر است

- مزایایی در کارایی برنامه دارند

- مزایایی در برنامه‌های هم‌رند و موازی دارند (Thread-safe)

- اشیاء برخی از کلاس‌هایی که می‌شناسیم، تغییرناپذیر هستند. مثال:

- String (مثلاً متد setValue ندارد)

- همه کلاس‌های لفاف انواع اولیه (Integer ، Boolean ، Double و غیره)





نوع داده شمارشی (enum)

- فرض کنید یک کلاس، تعداد محدود و مشخصی شیء خواهد داشت
- نمونه‌های این کلاس محدود هستند
- نمونه جدیدی در آینده اضافه نخواهد شد.
- مثلاً:

- Student Type : <BS, MS, PhD>
- SMS Status : <Sent, Delivered, Pending, Error>
- Color : <Blue, Green, Black, Red>
- چنین نیازی را چگونه پیاده‌سازی می‌کنید؟



```
class Color{
```

```
    public static final Color BLACK = new Color();
```

```
    public static final Color BLUE = new Color();
```

```
    public static final Color GREEN = new Color();
```

```
    public static final Color RED = new Color();
```

```
    private Color() {  
    }
```

مثال از کاربرد این کلاس:

```
Color c = Color.RED;
```

راه ساده تری که جاوا پیشنهاد می کند:

```
enum Color {  
    BLACK, BLUE, GREEN, RED  
}
```



انواع داده شماری (enum)

- Enumerated type یا enumeration یا enum
 - اگر یک کلاس، تعداد محدود و مشخصی شیء دارد
 - بهتر است به جای کلاس، با کلیدواژه enum تعریف شود
 - و همان جا همه اشیاء (نمونه‌ها) آن مشخص شود

```
enum Color {  
    BLACK, BLUE, GREEN, RED  
}
```

```
enum StudentType {  
    BS, MS, PHD  
}
```

```
enum Shape {  
    RECTANGLE, CIRCLE, SQUARE  
}
```

- همه این نمونه‌ها، به صورت ضمنی public، static و final هستند



```
Color color = Color.BLACK;  
Shape shape = Shape.CIRCLE;  
show(shape, color);
```

```
void show (Shape s, Color c) {  
    switch (s) {  
        case CIRCLE : ...  
        case RECTANGLE : ...  
    }  
}
```



چند نکته درباره انواع داده enum

- هیچ نمونه (شیء) جدیدی نمی‌تواند ایجاد شود
 - نمونه‌سازی با عملگر **new** منجر به خطای کامپایل می‌شود
 - ارث‌بری از انواع enum ممکن نیست
 - مفهوم وراثت را بعداً خواهیم دید
 - معمولاً یک enum تعریفی بسیار ساده شامل اسم نمونه‌ها دارد
- مثال: `enum Color{ BLACK, BLUE, GREEN, RED }`
- اما یک enum می‌تواند کلاس پیچیده‌تری باشد
 - با سازنده‌های مختلف و ویژگی‌ها و متدهای متنوع



تعريف انواع پیچیده تر enum

```
enum Shape {  
    Rectangle(1),  
    Circle(2),  
    Square(3);  
  
    private int number;  
  
    Shape(int i) {  
        number = i;  
    }  
  
    public int getNumber() {  
        return number;  
    }  
}
```

```
Shape sh = Shape.CIRCLE;  
print (sh.getNumber());
```

```
sh = Shape.valueOf("CIRCLE");  
print (sh.getNumber());
```

```
Shape[] array = Shape.values();  
for (Shape s : array) {  
    print (s.name());  
}
```

```
// Runtime Error:  
sh = Shape.valueOf("PYRAMID");
```



کوییز

خروجی این برنامه چیست؟

```
public class Quiz {
    static int sum(Integer... numbers){
        int s = 0;
        for (Integer i : numbers) {
            s+=i;
        }
        return s;
    }
    static int sum(String s1, String s2){
        Integer[] values =
            {Integer.valueOf(s1), Integer.valueOf(s2)};
        return sum(values);
    }
    public static void main(String[] args) {
        System.out.println(sum());           0
        System.out.println(sum(1,2));        3
        System.out.println(sum(1,2, new Integer(3))); 6
        System.out.println(sum("1", "2"));   3
        System.out.println(sum("One", "Two"));
    }
}
```

Auto-boxing

Runtime Error



```

enum Status{ SENT, DELIVERED, PENDING }
public class SMS {
    private Status status;
    private final String msg;
    private final String from, to;
    public SMS(String msg, String from, String to) {
        this.msg = msg;
        this.from = from;
        this.to = to;
    }
    public void setStatus(Status status) {
        this.status = status;
    }
    public String toString() {
        return String.format("%s=>%s:%s(%s)", from, to, msg, status);
    }
}

```

۱- آیا شیء sms تغییرناپذیر است؟

خیر

۲- خروجی این قطعه برنامه چیست؟

0912=>0935:Salam! (DELIVERED)

```

SMS sms = new SMS("Salam!", "0912", "0935");
sms.setStatus(Status.DELIVERED);
System.out.println(sms);

```



تمرین عملی

● کلاس Person

- سطح تحصیلات: enum
- متد toString
- سربار کردن سازنده (چند سازنده)
- سن: عدد صحیح و قد: اعشاری، هر دو اختیاری
- پس بهتر است به جای Primitive ، Wrapper باشند
- اسم یک فرد را ثابت کنید (final)
- درباره تغییرپذیری اشیاء این کلاس بحث کنید



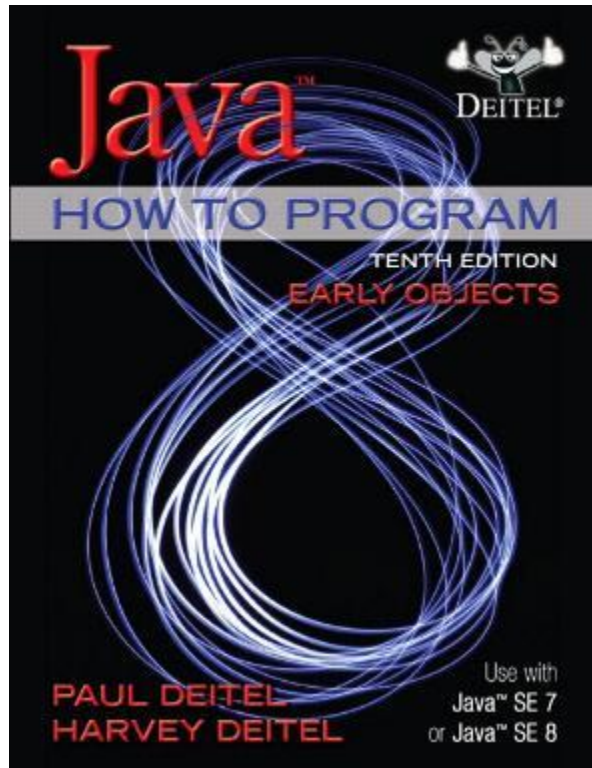
جمع بندی

- متدهای var-args
- کلاس‌های لفاف انواع اولیه (Primitive Wrapper Classes)
- سربار کردن متد
- متد toString
- متد equals
- متغیرهای ثابت (final)
- اشیاء تغییرناپذیر (Immutable)
- انواع داده شمارشی (enum)



- فصل‌های هشتم کتاب دایتل

Java How to Program (Deitel & Deitel)



8- Classes and Objects: A Deeper Look

- تمرین‌های همین فصل‌ها از کتاب دایتل



- کلاس Book را تعریف (تکمیل) کنید
- وضعیت کتاب: امانت، آماده، محتاج صحافی (نوع enum)
- متد toString مناسب
- ویژگی قیمت برای هر کتاب اجباری و ویژگی تعداد صفحات اختیاری است
 - یکی را از نوع int و دیگری را از نوع Integer تعریف کنید. (چرا؟!)
- setter ها و getter ها و سازنده‌های مناسب برایش تعریف کنید
- عنوان و نویسنده کتاب ثابت هستند، ولی امکان تغییر قیمت وجود دارد
 - کدام ویژگی‌ها final هستند؟



جستجو کنید و بخوانید



- موضوعات پیشنهادی برای جستجو:

- مزایای اشیاء تغییرناپذیر (Immutable Objects)

- نحوه صحیح پیاده‌سازی متد equals

- کلاس Number

- کلاس‌هایی مانند BigInteger و BigDecimal

- کلاس‌هایی مانند AtomicInteger و AtomicLong

- java.util.Enumeration

- Interned Strings



پایان

مطالب تکمیلی

```

public boolean equals (Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Person other = (Person) obj;
    if (nationalID == null) {
        if (other.nationalID != null)
            return false;
    } else if (!nationalID.equals(other.nationalID))
        return false;
    return true;
}

```



تاریخچه تغییرات

نسخه	تاریخ	توضیح
۱.۰.۰	۱۳۹۴/۳/۱۴	نسخه اولیه ارائه آماده شد

