HBO Graduaat Informatica Optie Programmeren

Java Basics

Classes Initialization

cvo leerstad

volwassenenonderwijs

Object Initialization Mechanism

- Three mechanisms
  - Constructors
  - Instance variable initializers
  - Instance initializers

Instance variables

Type	Default Value
boolean	false
double	0.0d
object reference	null

```
public class CoffeeCup {

    private int innerCoffee = 100;

    // The rest of the class...
}
```

```
// This class has no constructors or initializers
public class CoffeeCup {

    private int innerCoffee;

    // The rest of the class...
}
```

Constructors

- Class with constructor

```
public class CoffeeCup {
        public CoffeeCup(){
        }
}
```

- Overloading constructor

```
public class CoffeeCup {
private int amount;
        public CoffeeCup(int amount){
        this.amount = amount;
        }
}
```

Constructors

- This() invocation

```
public class CoffeeCup {
private int innerCoffee;
        public CoffeeCup(){
        this(237)
        }
public CoffeeCup(int innerCoffee){
  this.innerCoffee = innerCoffee;
```

When you specify a variable-length argument list, the Java compiler essentially reads that as "create an array of type <argument type>".

In fact , this is zero or more <argument type>"arguments...

- Creating a variable length argument constructor

```
public class CoffeeCup {
        public CoffeeCup(int amount, String name, String … features){
        ….
```

Constructors

- Constructor chaining

```
Horse horse = new Horse();
```

- What happens when you say new Horse()?

  (Horse extends Animal)

Constructors

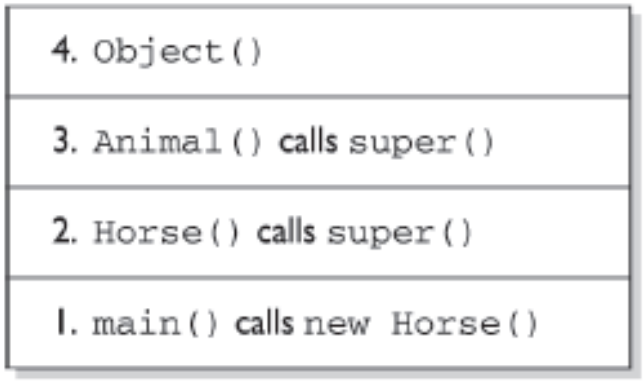
1. Horse constructor is invoked. Every constructor invokes the constructor of its superclass with an (implicit) call to super(), unless the constructor invokes an overloaded constructor of the same class .
2. Animal constructor is invoked (Animal is the superclass of Horse).
3. Object constructor is invoked. At this point we're on the top of the stack.

Constructors

4. Object instance variables are given their explicit values. By *explicit values,* we mean values that are assigned at the time the variables are declared, like "int x = 27", where "27" is the explicit value (as opposed to the default value) of the instance variable.
5. Object constructor completes.
6. Animal instance variables are given their explicit values (if any).

Constructors

7. Animal constructor completes.
8. Horse instance variables are given their explicit values (if any).
9. Horse constructor completes.



Constructors

Class Code (What You Type)	Compiler Generated Constructor Code (in Bold)

`class Foo { }`	`class Foo {`
	`  **Foo() {**`
	`  **super();**`
	`  **}**`
	`}`

`class Foo {`
`  Foo() { }`
`}`
	`class Foo {`
	`  Foo() {`
	`  **super();**`
	`  }`
	`}`

`public class Foo { }`	`class Foo {`
	`  **public Foo() {**`
	`  **super();**`
	`  **}**`
	`}`

`class Foo {`
`  Foo(String s) { }`
`}`
	`class Foo {`
	`  Foo(String s) {`
	`  **super();**`
	`  }`
	`}`

`class Foo {`
`  Foo(String s) {`
`  super();`
`  }`
`}`
	Nothing, compiler doesn't need to insert anything.

`class Foo {`
`  void Foo() { }`
`}`
	`class Foo {`
	`  void Foo() { }`
	`  **Foo() {**`
	`  **super();**`
	`  }`
	`}`
	`(void Foo()` *is a method, not a constructor.*`)`

Initializers

- Instance variable initializers

```
public class CoffeeCup {
private int innerCoffee;
            public CoffeeCup(){
            innerCoffee=355;
            }
}
```

```
public class CoffeeCup {
private int innerCoffee=355;
            }
```

- Instance initialization block

```
public class CoffeeCup {
private int innerCoffee;
    {
    innerCoffee=355;
    }
}
```

Initializers

- S
- V
- C
b

```
public static void main(String args[]) {
    }
    static {
        printGreeting();
    }
            static String s = "Wake up and smell the coffee!";
    static void printGreeting() {
        System.out.println(s);
    }
                                                }
```

Class initialization method

- All the class variable initializers and static initialization blocks of a class are collected by the Java compiler
- `<clinit>` invoked by JVM

Initialization variables

- The Java Virtual Machine initializes classes and interfaces on their first active use.
- An active use is:
  - The invocation of a method declared by the class (not inherited from a superclass)
  - The invocation of a constructor of the class
  - The use or assignment of a field declared by a class (not inherited from a superclass), **except for fields that are both static and final**, and are initialized by a compile-time constant expression.

Initialization arrays

- `int[] a1 = new int[5];`
- `int[] a1 = {1,2,3,4,5};`
- `int[][] a1 = {`

```
                    { 1, 2, 3, },
                    { 4, 5, 6, },
                };
```

- `int [][]a1 = new int[4][5];`

Enumerated types

- Creating an enum
  - The enum keyword
  - A name for the new type
  - A list of allowed values for the type
- Optional components
  - An interface or set of interfaces that the enum implements
  - Variable definitions
  - Method definitions
  - Value-specific class bodies

Enumerated types

- Enums are classes
- Extend `java.lang.Enum`
- Aren't integers
- No public constructor
- Are `public, static, final`
- Can be compared with == or equals
- Provide a `valueOf()` method
- `ordinal()` method returns the integer position
- `values()` method for iteration

Questions ??