



ORACLE

Academy



Java Foundations

7-3 Constructors

ORACLE
Academy



Objectives

- This lesson covers the following objectives:
 - Understand default values
 - Crash the program with a null reference
 - Understand the default constructor
 - Write a constructor that accepts arguments
 - Initialize fields with a constructor
 - Use this as an object reference



Remember the Prisoner Class

- It may have looked like this code
- It contains fields and methods

```
public class Prisoner {  
    //Fields  
    public String name;  
    public double height;  
    public int sentence;  
  
    //Methods  
    public void think(){  
        System.out.println("I'll have my revenge.");  
    } //end method think  
} //end class Prisoner
```

Fields Are Variables

- Variables hold values
- The values can be accessed.
- Code may need to access variables to ...
 - Make calculations
 - Check current values
 - Change a value
- What might happen if a field is accessed before it's assigned a value?

Exercise 1

- Continue editing with the `PrisonTest` project
 - A version of this program is provided for you in the files `PrisonTest_Student_7_3.java` and `Prisoner_Student_7_3.java`
- Investigate what happens when fields are accessed before they're assigned values
 - Instantiate a `Prisoner`
 - Try printing the value of each field



Variable: p01
Name: ???
Height: ???
Sentence: ???

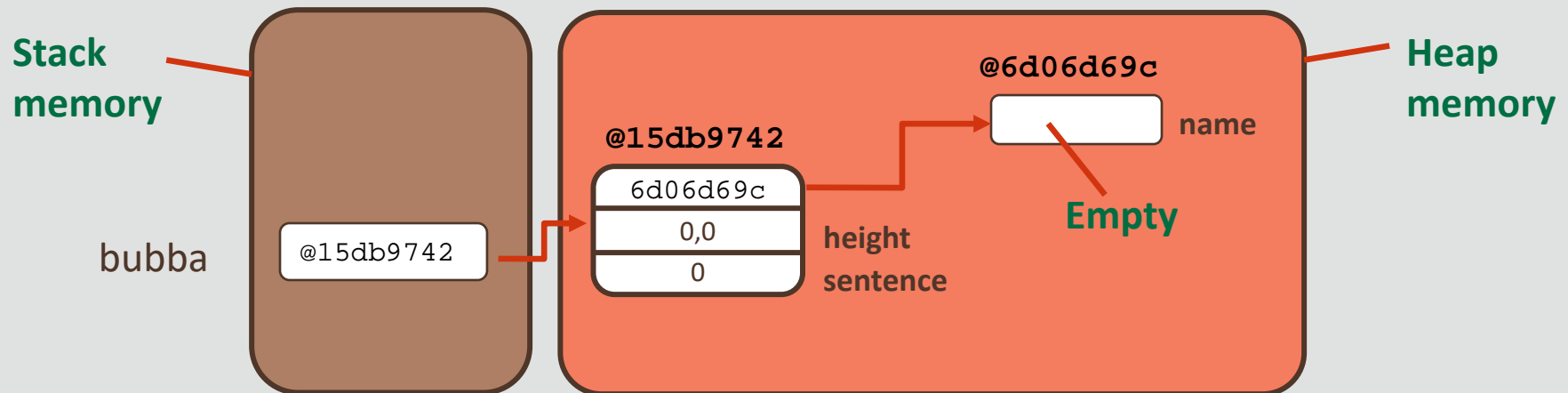
Accessing Uninitialized Fields

- If fields aren't initialized, they take on a default value
- Java provides the following default values:

Data Type	Default Value
<code>boolean</code>	<code>false</code>
<code>int</code>	<code>0</code>
<code>double</code>	<code>0.0</code>
<code>String</code>	<code>null</code>
Any Object type	<code>null</code>

Null Object References

- Objects can have a null value
- A null object points to an empty location in memory
- If an Object has another Object as a field (such as a String), its default value is null



Accessing Null Objects Is Dangerous

- What if a null object contains a field or method that needs to be accessed?
 - This causes the program to crash!
 - The specific error is a `NullPointerException`

```
public static void main(String[] args){  
    String test = null;  
    System.out.println(test.length());  
} //end method main
```

The Importance of Initializing Fields

- It's always good to minimize the chances that your program will crash
- And sometimes, Java's default values aren't desirable
- The remaining topics in this lesson examine helpful alternatives for initializing fields

Setting Prisoner Fields

- Currently, we need a line of code to set each field
- Four lines are required for each Prisoner object

```
public class PrisonTest {  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.name = "Bubba";  
        p01.height = 2.08;  
        p01.sentence = 4;  
        p02.name = "Twitch";  
        p02.height = 1.73;  
        p02.sentence = 3;  
    } //end method main  
} //end class PrisonTest
```

Methods Make Code More Efficient

- If you find yourself repeating similar lines of code ...
 - Programming can become tedious
 - It may be possible to do the same work in fewer lines
 - Try to write that code as part of a method instead

```
p01.name = "Bubba";  
p01.height = 2.08;  
p01.sentence = 4;
```

} First occurrence

```
p02.name = "Twitch";  
p02.height = 1.73;  
p02.sentence = 3;
```

} Repeated

Exercise 2

- Continue editing with the `PrisonTest` project
- Can fields be set more efficiently?
 - Add a `setFields()` method to the `Prisoner` class
 - This method should take three arguments, which are used to set the values for every field
 - Replace code in the `main` method with calls to this method



Variable: p01
Name: Bubba
Height: 6'10"
(2,08m)
Sentence: 4 years



Variable: p02
Name: Twitch
Height: 5'8"
(1,73m)
Sentence: 3 years

Writing a Method to Set Fields

- Your solution may have looked something like this:

```
public class Prisoner {  
    public String name;  
    public double height;  
    public int sentence;  
  
    public void setFields(String n, double h, int s){  
        name = n;  
        height = h;  
        sentence = s;  
    } //end method setFields  
} //end class Prisoner
```

Setting Prisoner Fields

- Two lines are required for each Prisoner object
- But it's possible to do the same work in even fewer lines!

```
public class PrisonTest {  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.setFields("Bubba", 2.08, 4);  
        p02.setFields("Twitch", 1.73, 3);  
  
    }//end method main  
}//end class PrisonTest
```


Calling a Constructor

- A constructor is a special method
- Its goal is to “construct” an object by setting the initial field values
- An object’s constructor is called once
 - This occurs during instantiation
 - And is never called again
- We’ve been calling constructors this whole time

Constructor method call

```
Prisoner p01 = new Prisoner();
```

The Default Constructor

- Java automatically provides a constructor for every class
- It's never explicitly written in a class
- This is called the default constructor
- It's considered a zero-argument constructor

Accepts zero arguments



```
Prisoner p01 = new Prisoner();
```

Writing a Constructor Method

- You can replace the default constructor with a constructor that you wrote yourself
- Constructors are written like any other method, except:
 - They have no return type (not even `void`)
 - They're named the **same** as the class

```
//Constructor  
public Prisoner(){  
    System.out.println("This is a constructor");  
}//end constructor
```

Exercise 3, Part 1

- Continue editing with the `PrisonTest` project
- Copy the constructor into the `Prisoner` class
 - Run the program
 - Observe how the code in this method is executed when `Prisoner` objects are instantiated

```
//Constructor  
public Prisoner(){  
    System.out.println("This is a constructor");  
}//end constructor
```

Exercise 3, Part 2

- How could you modify this constructor so that it sets every field in the class?
 - Use your understanding of methods to find a solution
 - Remember, constructors are methods
 - Remove the setFields() method
 - Your solution should make this method redundant
- Your IDE will complain in the main method:
 - How could these issues be fixed?
 - Run the program after you have a solution

You May Have Noticed ...

- Constructors can be written so that they accept arguments that set initial field values
- When you write your own constructor, the default constructor is no longer available
- Code becomes more useful and requires fewer lines
 - The next few slides illustrate this increased efficiency

```
//Constructor
public Prisoner(String n, double h, int s){
    name = n;
    height = h;
    sentence = s;
} //end constructor
```

Setting Fields Without a Constructor

- 4 lines are required for each Prisoner object

```
public class PrisonTest {  
  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.name = "Bubba";  
        p01.height = 2.08;  
        p01.sentence = 4;  
  
        p02.name = "Twitch";  
        p02.height = 1.73;  
        p02.sentence = 3;  
    } //end method main  
} //end class PrisonTest
```


Setting Fields with a Method

- 2 lines are required for each Prisoner object

```
public class PrisonTest {  
  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.setFields("Bubba", 2.08, 4);  
        p02.setFields("Twitch", 1.73, 3);  
  
    }//end method main  
}//end class PrisonTest
```

Setting Fields with a Constructor

- 1 line is required for each Prisoner object

```
public class PrisonTest {  
  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner("Bubba", 2.08, 4);  
        Prisoner p02 = new Prisoner("Twitch", 1.73, 3);  
  
    }//end method main  
}//end class PrisonTest
```

Naming Parameters

- Single-character variable names are commonly used ...
 - If the variable has a very limited scope
 - If there aren't a lot of variables to keep track of
 - For testing purposes
- But earlier in this course, we encouraged giving variables a descriptive names
 - This helps avoid confusion
 - Definitely follow this convention for fields
 - Some developers like to apply this convention to method parameters

Naming Parameters the Same as Fields

- This is also a common practice, especially with constructors
 - It's clearer what your parameters refer to
 - But this creates scope complications
- In the following code, is the name field or parameter printed?

```
public class Prisoner {  
    public String name;  
  
    public setName(String name){  
        System.out.println(name);  
    } //end method setName  
} //end class Prisoner
```

Which Version of name Is Printed?

- The parameter is printed
 - Variables within the most local scope take priority
 - In other words, the variables within the most recent scope
- Can the field still be accessed?
 - Yes! Fields exist within the scope of their class methods
 - But more syntax is required to access to them

```
public class Prisoner {  
    public String name;  
  
    public setName(String name){  
        System.out.println(name);  
    } //end method setName  
} //end class Prisoner
```

The this Keyword

- this is a reference to the current object
 - You can treat it like any other object reference
 - Which means you can use the dot operator (.)
- this.name accesses the Prisoner's field
- this.setName() accesses the Prisoner's method

```
public class Prisoner {  
    public String name;  
  
    public setName(String name){  
        System.out.println(name);  
    } //end method setName  
} //end class Prisoner
```

Exercise 4

- Modify the `Prisoner` constructor
 - Change the parameters of this method so that each parameter's name matches the name of a field
 - Set each field's value by using the `this` keyword

Summary of Constructors

- Are special methods in a class
- Named the same as the class
- Have no return type (not even void)
- Called only once during object instantiation
- May accept arguments
- Used to set initial values of fields
- If you don't write your own constructor, Java provides a default zero-argument constructor

Summary

- In this lesson, you should have learned how to:
 - Understand default values
 - Crash the program with a null reference
 - Understand the default constructor
 - Write a constructor that accepts arguments
 - Initialize fields with a constructor
 - Use this as an object reference





ORACLE

Academy

