

Using Encapsulation

9



ORACLE®

Interactive Quizzes



Objectives

After completing this lesson, you should be able to:

- Use public and private access modifiers
- Restrict access to fields and methods using encapsulation
- Implement encapsulation in a class
- Overload a constructor by adding method parameters to a constructor



Topics

- Access control
- Encapsulation
- Overloading constructors



What Is Access Control?

Access control allows you to:

- Hide fields and methods from other classes
- Determine how internal data gets changed
- Keep the implementation separate from the public interface

- Public interface:

```
setPrice( Customer cust)
```

- Implementation:

```
public void setPrice(Customer cust){  
    // set price discount relative to customer  
}
```

Access Modifiers

- `public`: Accessible by anyone
- `private`: Accessible only within the class

```
1 public class Item {  
2     // Base price  
3     private double price = 15.50;  
4  
5     public void setPrice(Customer cust){  
6         if (cust.hasLoyaltyDiscount()){  
7             price = price*.85; }  
8     }  
9 }
```



Access from Another Class

```
1 public class Item {
2     private double price = 15.50;
3
4     public void setPrice(Customer cust){
5         if (cust.hasLoyaltyDiscount()) {
6             price = price*.85; }
7     }
8 }
```

```
1 public class Order{
2     public static void main(String args[]){
3         Customer cust = new Customer(1002); //Example ID#1002
4         Item item = new Item();
5         item.price = 10.00;
6         item.setPrice(cust);
7     }
8 }
```

Won't compile

You don't need to know how setPrice works in order to use it.

Another Example

The data type of the field does not match the data type of the data used to set the field.

```
1 private int phone;  
2 public void setPhoneNumber(String s_num) {  
3     // parse out the dashes and parentheses from the  
4     // String first  
5     this.phone = Integer.parseInt(s_num);  
6 }
```


Using Access Control on Methods

```
1 public class Item {  
2     private int id;  
3     private String desc;  
4     private double price;  
5     private static int nextId = 1;  
6  
7     public Item() {  
8         setId();  
9         desc = "--description required--";  
10        price = 0.00;  
11    }  
12  
13    private void setId() {  
14        id = Item.nextId++;  
15    }
```

Called from within a public method

Private method

Topics

- Access control
- Encapsulation
- Overloading constructors



Encapsulation

- Encapsulation means hiding object fields. It uses access control to hide the fields.
 - Safe access is provided by getter and setter methods.
 - In setter methods, use code to ensure that values are valid.
- Encapsulation mandates programming to the interface:
 - A method can change the data type to match the field.
 - A class can be changed as long as interface remains same.
- Encapsulation encourages good object-oriented (OO) design.

Get and Set Methods

```
1  public class Shirt {
2      private int shirtID = 0;           // Default ID for the shirt
3      private String description = "-description required-"; // default
4      private char colorCode = 'U';      //R=Red, B=Blue, G=Green, U=Unset
5      private double price = 0.0;        // Default price for all items
6
7      public char getColorCode() {
8          return colorCode;
9      }
10     public void setColorCode(char newCode) {
11         colorCode = newCode;
12     }
13
14     // Additional get and set methods for shirtID, description,
15     // and price would follow
16 } // end of class
```

Why Use Setter and Getter Methods?

```
1 public class ShirtTest {  
2     public static void main (String[] args) {  
3         Shirt theShirt = new Shirt();  
4         char colorCode;  
5         // Set a valid colorCode  
6         theShirt.setColorCode('R');  
7         colorCode = theShirt.getColorCode();  
8         System.out.println("Color Code: " + colorCode);  
9         // Set an invalid color code  
10        theShirt.setColorCode('Z'); Not a valid color code  
11        colorCode = theShirt.getColorCode();  
12        System.out.println("Color Code: " + colorCode);  
13    }  
14 ...
```

Output:

```
Color Code: R  
Color Code: Z
```

Setter Method with Checking

```
15  public void setColorCode(char newCode) {
16      if (newCode == 'R') {
17          colorCode = newCode;
18          return;
19      }
20      if (newCode == 'G') {
21          colorCode = newCode;
22          return;
23      }
24      if (newCode == 'B') {
25          colorCode = newCode;
26          return;
27      }
28      System.out.println("Invalid colorCode. Use R, G, or B");
29  }
30 }
```

Using Setter and Getter Methods

```
1 public class ShirtTest {  
2     public static void main (String[] args) {  
3         Shirt theShirt = new Shirt();  
4         System.out.println("Color Code: " + theShirt.getColorCode());  
5  
6         // Try to set an invalid color code  
7         theShirt.setColorCode('Z'); ———— Not a valid color code  
8         System.out.println("Color Code: " + theShirt.getColorCode());  
9     }
```

Output:

Color Code: U ———— Before call to setColorCode() – shows default value
Invalid colorCode. Use R, G, or B ———— call to setColorCode prints error message
Color Code: U ———— colorCode not modified by invalid argument passed to setColorCode()

Exercise 9-1: Encapsulate a Class

In this exercise, you encapsulate the `Customer` class.

1. Open **Exercise_09-1** project in NetBeans.
2. Change access modifiers so that fields must be read or modified through public methods.
3. Allow the `name` field to be read and modified.
4. Allow the `ssn` field to be read but not modified (read only).



Topics

- Access control
- Encapsulation
- Overloading constructors



Initializing a Shirt Object

Explicitly:

```
1 public class ShirtTest {  
2     public static void main (String[] args) {  
3         Shirt theShirt = new Shirt();  
4  
5         // Set values for the Shirt  
6         theShirt.setColorCode('R');  
7         theShirt.setDescription("Outdoors shirt");  
8         theShirt.price(39.99);  
9     }  
10 }
```

Using a constructor:

```
Shirt theShirt = new Shirt('R', "Outdoors shirt", 39.99);
```

Constructors

- Constructors are usually used to initialize fields in an object.
 - They can receive arguments.
 - When you create a constructor with arguments, it removes the default no-argument constructor.

Shirt Constructor with Arguments

```
1 public class Shirt {
2     public int shirtID = 0;                // Default ID for the shirt
3     public String description = "-description required-"; // default
4     private char colorCode = 'U';          // R=Red, B=Blue, G=Green, U=Unset
5     public double price = 0.0;             // Default price all items
6
7     // This constructor takes three argument
8     public Shirt(char colorCode, String desc, double price ) {
9         setColorCode(colorCode);
10        setDescription(desc);
11        setPrice(price);
12    }
```

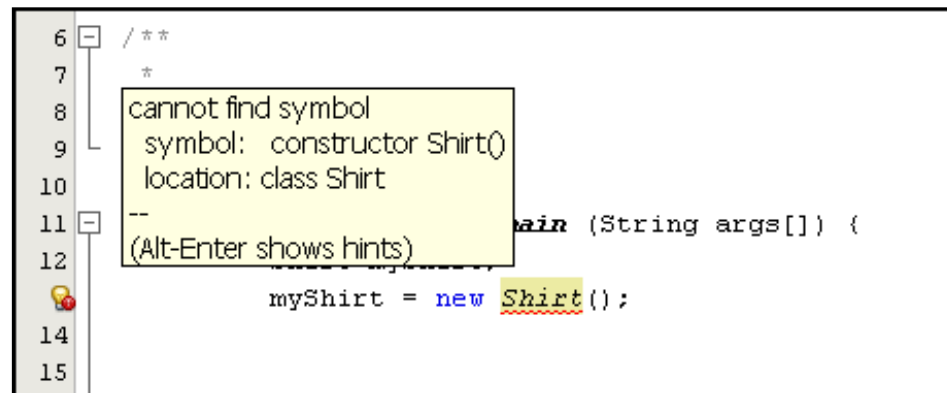
Default Constructor and Constructor with Args

When you create a constructor with arguments, the default constructor is no longer created by the compiler.

```
// default constructor  
public Shirt()
```

This constructor is not in the source code. It only exists if no constructor is explicitly defined.

```
// Constructor with args  
public Shirt (char color, String desc, double price)
```



Overloading Constructors



```
1 public class Shirt {
2     ... //fields
3
4     // No-argument constructor
5     public Shirt() {
6         setColorCode('U');
7     }
8     // 1 argument constructor
9     public Shirt(char colorCode ) {
10         setColorCode(colorCode);
11     }
12     // 2 argument constructor
12     public Shirt(char colorCode, double price) {
14         this(colorCode);
15         setPrice(price);
16     }
```

If required, must be added explicitly

Calling the 1 argument constructor

Quiz



What is the default constructor for the following class?

```
public class Penny {  
    String name = "lane";  
}
```

- a. `public Penny(String name)`
- b. `public Penny()`
- c. `class()`
- d. `String()`
- e. `private Penny()`



Exercise 9-2: Create an Overloaded Constructor

1. Continue editing **Exercise_09-1** or open **Exercise_09-2** in NetBeans.

In the `Customer` class:

2. Add a custom constructor that initializes the fields.

In the `ShoppingCart` class:

3. Declare, instantiate, and initialize a new `Customer` object by calling the custom constructor.
4. Test it by printing the `Customer` object name (call the `getName` method).



Summary

In this lesson, you should have learned how to:

- Use public and private access modifiers
- Restrict access to fields and methods using encapsulation
- Implement encapsulation in a class
- Overload a constructor by adding method parameters to a constructor



Practices Overview

- 9-1: Encapsulating Fields
- 9-2: Creating Overloaded Constructors

