CHAPTER **8**

Object Oriented Programming – A Case Study

(based on material from slides accompanying
Horstmann: Java for Everyone: Late Objects,
John Wiley and Sons Inc, with updates by Simon Jones)

Slides by Donald W. Smith
TechNeTrain.com

Final Draft
10/30/2011

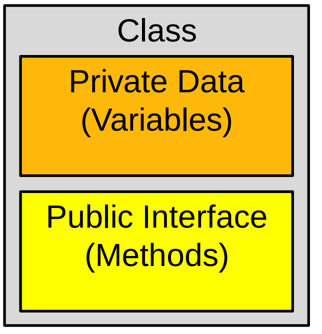
Contents

- A Simple Case Study in Object-Oriented Programming
  - A Cash Register class
  - A simulated cash register that tracks the item count and the total amount due
  - A supermarket system might have *several instances* of this class:

    checkouts + customer services kiosk

Reminder: Diagram of a Class

- *Private* Data
  - Each object has its own *private* data that other objects cannot directly access
  - Methods of the *public* interface provide access to *private* data:
  - This is called Encapsulation
- *Public* Interface
  - Each object has a set of *public* methods available for other objects to use
  - This *public* interface is the key to design
- *There may be private methods too*



8.3 Public Interface of a Class

- When you design a class, start by specifying the *public interface* of the new class
  - Example: A Cash Register Class
    - What tasks will this class perform?
    - What methods will you need?
    - What parameters will the methods need to receive?
    - What will the methods return?

Task	Method	Returns
Add the price of an item	`addItem(double)`	void
Get the total amount owed	`getTotal()`	double
Get the count of items purchased	`getCount()`	int
Clear the cash register for a new sale	`clear()`	void

Writing the Public Interface

```
/**
  A simulated cash register that tracks the item count
  and the total amount due.
*/
public class CashRegister
{
  /**
    Adds an item to this cash register.
    @param price: the price of this item
  */
  public void addItem(double price)
  {
    // Method body
  }
  /**
    Gets the price of all items in the current sale.
    @return the total price
  */
  public double getTotal()  ...
```

The method declarations make up the *public interface* of the class

The data and method bodies make up the *private implementation* of the class

Note: Non-static Methods

- We are now writing methods *without* using the `static` modifier:

```
public void addItem(double val)
```

- This is the correct technique when we have a class that we *instantiate:*

```
// Construct a CashRegister object
CashRegister register1 = new CashRegister();
```

- And then need to call methods *within a specific instance (object):*

```
// Invoke a method of the object
register1.addItem(1.95);
```

8.4 Designing the Data Representation

- An object stores data in instance variables
  - Variables declared inside the class
  - All methods inside the class have access to them
    - Can change or access them
  - What data will our `CashRegister` methods need?

Task	Method	Data Needed
Add the price of an item	`addItem()`	`total, count`
Get the total amount owed	`getTotal()`	`total`
Get the count of items purchased	`getCount()`	`count`
Clear the cash register for a new sale	`clear()`	`total, count`

```
private int itemCount;
private double totalPrice;
```

Once again, ***not*** `static`

8.5 Implementing Instance Methods

- Implement instance methods that use the private instance variables, for example:

```
public void addItem(double price)
{
  itemCount++;
  totalPrice = totalPrice + price;
}
```

- Similarly:

Task	Method	Returns
Add the price of an item	`addItem(double)`	void
Get the total amount owed	`getTotal()`	double
Get the count of items purchased	`getCount()`	int
Clear the cash register for a new sale	`clear()`	void

8.6 Constructors

- A *constructor* is a method that initializes instance variables of an object
  - It is automatically called when an object is created
  - It has exactly the same name as the class

```
public class CashRegister
{
  . . .
  /**
    Constructs a cash register with cleared item count and total.
  */
  public CashRegister() // A constructor
  {
    itemCount = 0;
    totalPrice = 0;
  }
}
```

Constructors never return values, but do not use void in their declaration

CashRegister.java

```
/**
   A simulated cash register that tracks the ite
   the total amount due.
*/
public class CashRegister
{
   private int itemCount;
   private double totalPrice;

   /**
      Constructs a cash register with cleared i
   */
   public CashRegister()
   {
      itemCount = 0;
      totalPrice = 0;
   }

   /**
      Adds an item to this cash register.
      @param price the price of this item
   */
   public void addItem(double price)
   {
      itemCount++;
      totalPrice = totalPrice + price;
   }

   /**
      Gets the price of all items in the current sale.
      @return the total amount
   */
   public double getTotal()
   {
      return totalPrice;
   }

   /**
      Gets the number of items in the current sale.
      @return the item count
   */
   public int getCount()
   {
      return itemCount;
   }

   /**
      Clears the item count and the total.
   */
   public void clear()
   {
      itemCount = 0;
      totalPrice = 0;
   }
}
```

CashRegisterTester.java

```
/**
   This program tests the CashRegister clas
*/
public class CashRegisterTester
{
   public static void main(String[] args)
   {
      CashRegister register1 = new CashRegister();
      register1.addItem(1.95);
      register1.addItem(0.95);
      register1.addItem(2.50);
      System.out.println(register1.getCount());
      System.out.println("Expected: 3");
      System.out.printf("%.2f\n", register1.getTotal());
      System.out.println("Expected: 5.40");
   }
}
```

Program Run

```
3
Expected: 3
5.40
Expected: 5.40
```

- Test all methods
  - Print expected results
  - Output actual results
  - Compare results

Summary: Classes and Objects

- A class describes a set of objects with the same behavior.
  - Every class has a public interface: a collection of methods through which the objects of the class can be manipulated.
  - Encapsulation is the act of providing a public interface and hiding the implementation details.
  - Encapsulation enables changes in the implementation without affecting users of a class

Summary: Variables and Methods

- An object's instance variables store the data required for executing its methods.
- Each object of a class has its own set of instance variables.
- An instance method can access the instance variables of the object on which it acts.
- A private instance variable can only be accessed by the methods of its own class.