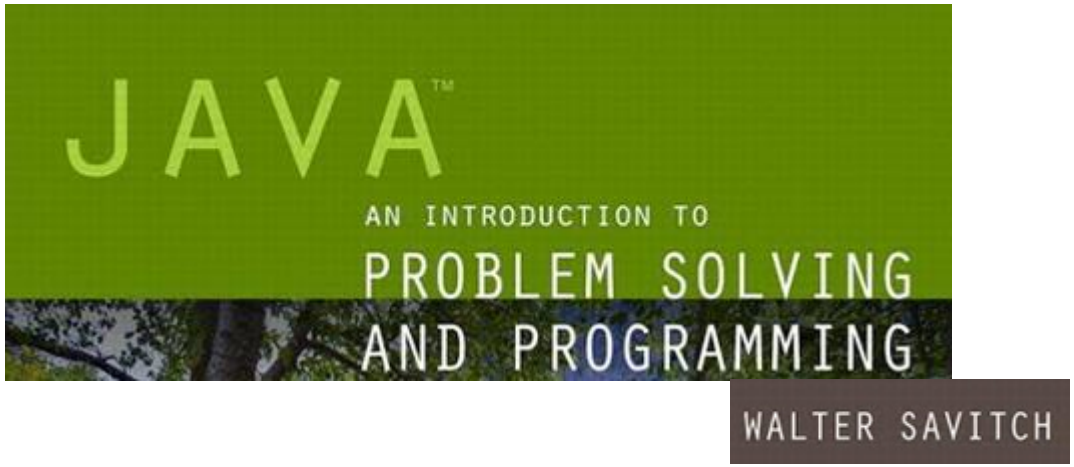


INFORMATION HIDING AND ENCAPSULATION

Ch 5.2

Information Hiding, Encapsulation: Outline

- Information Hiding
- The public and private Modifiers
- Methods Calling Methods
- Encapsulation
- UML Class Diagrams

Information Hiding

- Programmer using a class method need not know details of the implementation of the method
  - Only needs to know what the method does
- Information hiding:
  - Designing a method so it can be used without knowing details
- Also referred to as abstraction
- Method design should separate what from how

Careful documentation of what a method does can achieve that.
For more info read: pre/post conditions

The `public` and `private` Modifiers

- When an identifier is specified as **`public`**:
  - Any other class can directly access that identifier by name
- When an identifier is specified as **`private`**:
  - Only the class itself can directly access that identifier by name
  - It can not be accessed directly from the outside

- Classes are generally specified as **`public`**
- Instance variables are usually **`private`**

Programming Example

```
public class Rectangle
{
  public  int width;
  public  int height;
  private int area;

  public void setDimensions
         (int newWidth,
          int newHeight)
  { width = newWidth;
    height = newHeight;
    area = width * height;
  }

  public int getArea ()
  { return area;
  }

}
```

- Could be used like this in main:

```
Rectangle box = new Rectangle();
box.setDimensions(10,5);
System.out.println(box.getArea());
```

- A statement such as

```
box.width = 6;
```

is legal since width is `public`

Is it a good idea? Why?

No, it causes an inconsistency. Area would still be 50.

Programming Example

```
public class Rectangle
{
  private int width;
  private int height;
  private int area;

  public void setDimensions
         (int newWidth,
          int newHeight)
  { width = newWidth;
    height = newHeight;
    area = width * height;
  }

  public int getArea ()
  { return area;
  }

}
```

- Could be used like this in main:

```
Rectangle box = new Rectangle();
box.setDimensions(10,5);
System.out.println(box.getArea());
```

- A statement such as

```
box.width = 6;
```

is illegal since width is `private`

Is it a good idea? Why?

Yes, keeps remaining elements of the class consistent

Can we improve it more?

Programming Example

```
public class Rectangle2
{
  private int width;
  private int height;
  private int area;

  public void setDimensions
         (int newWidth,
          int newHeight)
  { width = newWidth;
    height = newHeight;
    area = width * height;
  }

  public int getArea ()
  { return width * height;
  }

}
```

- Could be used like this in main:

```
Rectangle box = new Rectangle();
box.setDimensions(10,5);
System.out.println(box.getArea());
```

- A statement such as

  `box.width = 6;`

  is illegal since width is `private`

- And now we are NOT storing **`area`** but computing it when needed.

Accessor and Mutator Methods

- When instance variables are private must provide methods to access values stored there
  - Typically named **`getSomeValue`**
  - Referred to as an accessor method
- Must also provide methods to change the values of the private instance variable
  - Typically named **`setSomeValue`**
  - Referred to as a mutator method

Accessor and Mutator Methods

```
public class Rectangle
private int width;
private int height;
public void setWidth(int w)
{ width = w; }
public void setHeight(int h)
{ height = h; }
public int getWidth()
{ return width; }
public int getHeight()
{ return height; }
public int getArea()
{ return width * height;}
```

```
public class RectangleTest {
public static void main (String[] args) {
Rectangle box1 = new Rectangle();
box1.setWidth(5);
box1.setHeight(10);
System.out.println("The dimensions of"+
"box1 are ("+ box1.getWidth() + ","+
box1.getHeight() + ")");
System.out.println("The area of box1 is "
+ box1.getArea());
}
}
```

Methods Calling Methods

- A method body may call any other method
- If the invoked method is within the same class

  Need not use prefix of receiving object

```
public void method1 (int x, int y)
{ int sum = method2(x,y);
x++; y++;
method3(sum); }
public int method2(int i, int j)
{ i++; return i + j;
}
public void method3(int s )
{System.out.println(s);
System.out.println("Done!" );}
```

What will be printed when method1(2,3) is called?

6
Done!

Activity

- Supermarkets often give prices for a group of items such as 5 for $1.25 or 3 for $1.00, instead of the price for one item. They hope that if they price apples at 5 for 1.25$ you will buy 5 apples instead of 2. But 5 for $1.25 is really 0.25 each, and if you buy 2 apples, they charge you only $0.5.
- Let's define a class named Purchase to manage the purchase of multiple identical items. What instance variables would be needed?

The instance variables are as follows:

```
private String name;
private int groupCount; //Part of a price, like the 2
                        // in 2 for $1.99.
private double groupPrice;//Part of a price, like the $1.99
                          //in 2 for $1.99.
private int numberBought;//Number of items bought.
```

Activity(cont)

- what about the methods ?
  1. Accessor and mutator methods
  2. Compute the total cost
  3. Compute a single unit cost
  4. Read information
  5. Print information

```
import java.util.Scanner;
/**
Class for the purchase of one kind of item, such as 3 oranges.
Prices are set supermarket style, such as 5 for $1.25.
*/
public class Purchase
{
    private String name;
    private int groupCount;     //Part of a price, like the 2 in
                                //2 for $1.99.
    private double groupPrice;  //Part of a price, like the $1.99
                                // in 2 for $1.99.
    private int numberBought;   //Number of items bought.
    public void setName(String newName)
    {
        name = newName;
    }
    /**
    Sets price to count pieces for $costForCount.
    For example, 2 for $1.99.
    */
    public void setPrice(int count, double costForCount)
    {
        if ((count <= 0) || (costForCount <= 0))
        {
            System.out.println("Error: Bad parameter in " +
                               "setPrice.");
            System.exit(0);
        }
        else
        {
            groupCount = count;
            groupPrice = costForCount;
        }
    }
```

```
public void setNumberBought(int number)
{
    if (number <= 0)
    {
        System.out.println("Error: Bad parameter in " +
                           "setNumberBought.");
        System.exit(0);
    }
    else
        numberBought = number;
}
```

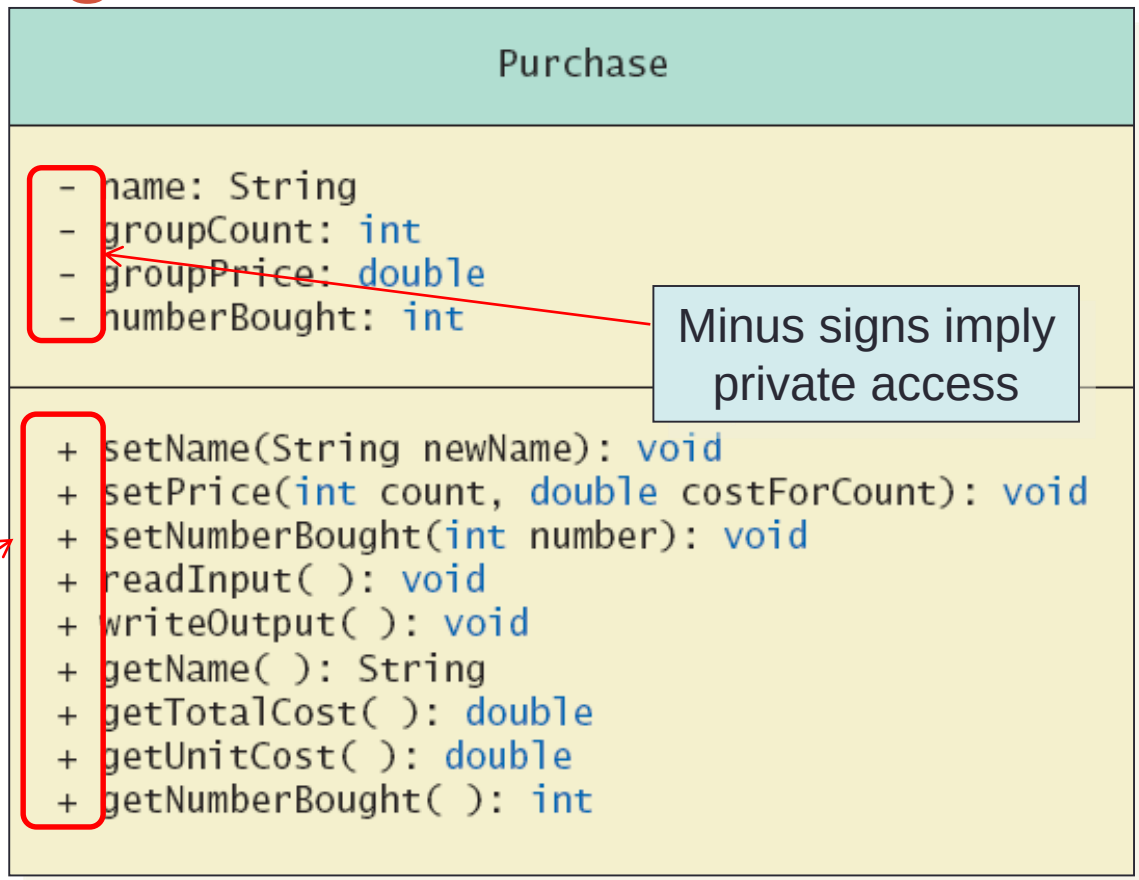
```
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter name of item you are purchasing:");
    name = keyboard.nextLine();
    System.out.println("Enter price of item as two numbers.");
    System.out.println("For example, 3 for $2.99 is entered as");
    System.out.println("3 2.99");
    System.out.println("Enter price of item as two numbers, " +
                       "now:");
    groupCount = keyboard.nextInt();
    groupPrice = keyboard.nextDouble();

    while ((groupCount <= 0) || (groupPrice <= 0))
    { //Try again:
        System.out.println("Both numbers must " +
                           "be positive. Try again.");
        System.out.println("Enter price of " +
                           "item as two numbers.");
        System.out.println("For example, 3 for " +
                           "$2.99 is entered as");
        System.out.println("3 2.99");
        System.out.println(
                           "Enter price of item as two numbers, now:");
        groupCount = keyboard.nextInt();
        groupPrice = keyboard.nextDouble();
    }
    System.out.println("Enter number of items purchased:");
    numberBought = keyboard.nextInt();

    while (numberBought <= 0)
    { //Try again:
        System.out.println("Number must be positive. " +
                           "Try again.");
        System.out.println("Enter number of items purchased:");
        numberBought = keyboard.nextInt();
    }
}
```

```
umber being purchased.

out()

ln(numberBought + " " + name);
ln("at " + groupCount +
   " for $" + groupPrice);

e()

alCost()

ce / groupCount) * numberBought;

tCost()

e / groupCount;

Bought()

ght;
```

```
public class PurchaseDemo
{
    public static void main(String[] args)
    {
        Purchase oneSale = new Purchase();
        oneSale.readInput();


        oneSale.writeOutput();
        System.out.println("Cost each $" + oneSale.getUnitCost());
        System.out.println("Total cost $" +
                oneSale.getTotalCost());
    }
}
```

UML Class Diagrams

- Note Figure 5.4 for the **`Purchase`** class



More code samples

- listing 5.11
- listing 5.12
- listing 5.13
- listing 5.14
- listing 5.15