Designing Classes – Part 2

CSC 1051 – Data Structures and Algorithms I

Dr. Mary-Angela Papalaskari
Department of Computing Sciences
Villanova University

Course website:
www.csc.villanova.edu/~map/1051/

Encapsulation

• An encapsulated object can be thought of as a black box -- its inner workings are hidden from the client
• The client invokes the interface methods which in turn manage the instance data

Violating Encapsulation

It is possible for a class to access the instance data of another class directly

Violating Encapsulation - WRONG

It is possible for a class to access the instance data of another class directly – but it's not a good idea!

Getting to know classes so far

• Using predefined classes from the Java API.
• Defining classes for our own datatypes.

datatypes:
– Account
– Die
– Shoe
– Person

Clients (Driver classes):
– Transactions, OnePercent
– RollingDice
– YouVeGotShoes (Project)
– PeopleBeingPeople (Lab)

Next: visibility modifiers, graphical objects,
Use Accessors & Mutators - RIGHT

• Indirect access through methods
• accessors and mutators (“getters” and “setters”)
• Usually named getX() and setX()

Transactions.java

int x1 = acct1.getBalance();

Account.java

\begin{verbatim}
public final double RATE = 0.015;
\end{verbatim}

Visibility Modifiers

• In Java, we enforce encapsulation through the appropriate use of visibility modifiers:
  - public – can be referenced from other classes
  - private – can be referenced only within that class:
  - protected – involves inheritance (discussed later)
• Data declared without a visibility modifier have default visibility
  and can be referenced by any class in the same package
• An overview of all Java modifiers is presented in Appendix E

Violating Encapsulation experiment

• Revisit the Account & Transactions example from last week
  (be sure to use our simplified versions from the course website)
• Add some code to the Transactions.java class to modify the value of an instance variable, eg:
  \begin{verbatim}
  acct1.name = "Joe";
  \end{verbatim}
  (This should work as expected)
• Now modify Account.java – insert the modifier private in front of that variable declaration:
  \begin{verbatim}
  private String name;
  \end{verbatim}
• Re-compile the Account class and run your program again. Note the error you get:

Public Constants... OK

Example: The Account class can have a constant for the interest rate:

\begin{verbatim}
public final double RATE = 0.015;
\end{verbatim}

A client (eg, OnePercent.java) can access this constant directly:

\begin{verbatim}
System.out.print("Interest rate = "+ acct1.RATE);
\end{verbatim}
Public Constants... BETTER

It is better to declare constants as static.

```java
public final static double RATE = 0.015;
```

This way, a client can access the constants without creating an object, using the class name:

```java
System.out.println("Interest rate = " + Account.RATE);
```

Graphical Objects

- Some objects contain information that determines how the object should be represented visually
- Graphical objects
  - data about position, size, and other attributes
  - methods to draw the object
- Let’s look at some other examples of graphical objects:

  Example 1: SmilingFacePanel
  SmilingFacePanel.java

  Example 2: SplatPanel
  SplatPanel.java

Visibility Modifiers – the RULES

<table>
<thead>
<tr>
<th></th>
<th>public</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>NO</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>(but OK for public constants)</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>Yes</td>
<td>Yes, for support methods only</td>
</tr>
</tbody>
</table>

See also imInUrClassMessingUrInstanceData.java
The `SmilingFace` program draws a face by defining the `paintComponent` method of a panel.

A UML class diagram:

- **SmilingFace**
  - `main (args : String[]): void`

- **SmilingFacePanel**
  - `BASEX: int`
  - `BASEY: int`
  - `paintComponent (p: Graphics): void`

---

```java
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.*;

public class SmilingFacePanel extends JPanel {
    private final int BASEX = 120, BASEY = 60; // base point for head

    public SmilingFacePanel() {
        setBackground(Color.blue);
        setPreferredSize(new Dimension(320, 200));
        setFont(new Font("Arial", Font.BOLD, 16));
    }
}
```

---

The `SmilingFace` program draws a face by defining the `paintComponent` method of a panel.

A UML class diagram:

- **SmilingFace**
  - `main (args : String[]): void`

- **SmilingFacePanel**
  - `BASEX: int`
  - `BASEY: int`
  - `paintComponent (p: Graphics): void`

```java
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.*;

public class SmilingFacePanel extends JPanel {
    private final int BASEX = 120, BASEY = 60; // base point for head

    public SmilingFacePanel() {
        setBackground(Color.blue);
        setPreferredSize(new Dimension(320, 200));
        setFont(new Font("Arial", Font.BOLD, 16));
    }
}
```
// Draws a face.
public void paintComponent(Graphics page) {
    super.paintComponent(page);
    page.setColor(Color.yellow);
    page.fillOval(BASEX, BASEY, 80, 80); // head
    page.fillOval(BASEX-5, BASEY+20, 90, 40); // ears
    page.setColor(Color.black);
    page.drawOval(BASEX+20, BASEY+30, 15, 7); // eyes
    page.fillOval(BASEX+45, BASEY+30, 15, 7); // pupils
    page.drawArc(BASEX+20, BASEY+25, 15, 7, 0, 180); // eyebrows
    page.fillOval(BASEX-105, BASEY-15); // mouth
    page.drawString("Always remember that you are unique!", BASEX-105, BASEY-15);
    page.drawString("Just like everyone else.", BASEX-45, BASEY+105);
}

Smiling Face Example
- Every Swing component has a paintComponent method
- The paintComponent method accepts a Graphics object that represents the graphics context for the panel
- We define the paintComponent method to draw the face with appropriate calls to the Graphics methods
- Note the difference between drawing on a panel and adding other GUI components to a panel

Objects with a draw() method
- The next example - Splat - is structured differently
- It draws a set of colored circles on a panel, but each circle is represented as a separate object that maintains its own graphical information
- The paintComponent method of the panel "asks" each circle to draw itself
- See Splat.java
- See SplatPanel.java
- See Circle.java
import javax.swing.*;
import java.awt.*;

public class Splat {
    public static void main (String[] args) {
        JFrame frame = new JFrame("Splat");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new SplatPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
Designing Classes – Part 2

CSC 1051 Villanova University

Dr. Papalaskari
```java
import javax.swing.*;
import java.awt.*;

public class SplatPanel extends JPanel {
  private Circle circle1, circle2, circle3, circle4, circle5;
  public SplatPanel() {
    circle1 = new Circle(30, Color.red, 70, 35);
    circle2 = new Circle(50, Color.green, 30, 20);
    circle3 = new Circle(100, Color.cyan, 60, 85);
    circle4 = new Circle(45, Color.yellow, 170, 30);
    circle5 = new Circle(60, Color.blue, 200, 60);
    setPreferredSize (new Dimension(300, 200));
    setBackground (Color.black);
  }
  public void paintComponent (Graphics page) {
    super.paintComponent(page);
    circle1.draw(page);
    circle2.draw(page);
    circle3.draw(page);
    circle4.draw(page);
    circle5.draw(page);
  }
}
```
continue

//ellido-------------------------------------------------------------
//  Draws this circle in the specified graphics context.
//ellido-------------------------------------------------------------
public void draw (Graphics page)
{
  page.setColor (color);
  page.fillOval (x, y, diameter, diameter);
}

//ellido-------------------------------------------------------------
// Diameter mutator.
//ellido-------------------------------------------------------------
public void setDiameter (int size)
{
  diameter = size;
}

//ellido-------------------------------------------------------------
// Color mutator.
//ellido-------------------------------------------------------------
public void setColor (Color shade)
{
  color = shade;
}

continue

//-----------------------------------------------------------------
// X mutator.
//-----------------------------------------------------------------
public void setX (int upperX)
{
  x = upperX;
}

//-----------------------------------------------------------------
// Y mutator.
//-----------------------------------------------------------------
public void setY (int upperY)
{
  y = upperY;
}

//-----------------------------------------------------------------
// Diameter accessor.
//-----------------------------------------------------------------
public int getDiameter ()
{
  return diameter;
}

continue

//-----------------------------------------------------------------
// Color accessor.
//-----------------------------------------------------------------
public Color getColor ()
{
  return color;
}

//-----------------------------------------------------------------
// X accessor.
//-----------------------------------------------------------------
public int getX ()
{
  return x;
}

//-----------------------------------------------------------------
// Y accessor.
//-----------------------------------------------------------------
public int getY ()
{
  return y;
}