Centering Shapes

Assuming the diameter of the circle is 1/2 the diameter of the rectangle, where should we draw the circle?

Design Emphasis

- Abstraction
- Allows understanding something without knowing all the details
- A method defines a procedural abstraction
- A class defines a data abstraction
- All abstractions perform information hiding

Data Types

- Used to indicate the meaning of a chunk of memory
- Limits how we can use that memory
  ```
  int index;
  index = 0;
  a[index] = String name;
  name = "Gwen";
  index++;
  name.length()
  ```
Abstract Data Types

- A class defines an ADT (Abstract Data Type)
- Data members indicate how to interpret the bits
- Methods indicate how values of the data type can be manipulated.

Class Definition

class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

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

Rectangle as an ADT

- "Data" is a rectangle
- Instance variables are hidden
- The only things we can do with Rectangles are create them and get their area.
Parts of a Contract

- Syntax - Method signature
  - Method name
  - Parameter list
  - Return type
- Semantics - Comments
  - Preconditions: requirements placed on the caller
  - Postconditions: what the method modifies and/or returns

Contract Example

Signature

/*
 * Create a rectangle of a particular size.
 * @param width the width of the rectangle.
 *   This must be > 0.
 * @param height the height of the rectangle.
 *   This must be > 0.
 */

public Rectangle(double width, double height) {

Preconditions

Postcondition

Contracts

- Precondition - what the caller must satisfy to call the method legally
- Postcondition - what will be true when the method returns if the precondition is true when the method is called

ANYTHING CAN HAPPEN IF THE PRECONDITION IS NOT SATISFIED WHEN THE METHOD IS CALLED!!!
Checking Preconditions

- Normally, a method does not check preconditions! It is the caller's responsibility!
- During development, it is a good idea to check preconditions using "assert":

```java
public Rectangle(double width, double height) {
    assert width > 0;
    assert height > 0;
    this.width = width;
    this.height = height;
}
```

Assertion Failure

Executing:
```java
new Rectangle(-5, 1)
```
Leads to:
```java
Exception in thread "main" java.lang.AssertionError
at Rectangle.<init>(Rectangle.java:19)
at MainClass.main(MainClass.java:8)
```

javadoc

```java
/**
 * Create a rectangle of a particular size.
 * @param width the width of the rectangle.
 *     This must be > 0.
 * @param height the height of the rectangle.
 *     This must be > 0.
 */
public Rectangle(double width, double height) {

/**
 * Return the area encompassed by the rectangle.
 * @return the area encompassed by the rectangle.
 */
public double getArea() {
```
/** Create a rectangle of a particular size.  
 * @param width  the width of the rectangle.  
 *   This must be > 0.  
 * @param height the height of the rectangle.  
 *   This must be > 0.  
 */

public Rectangle(double width, double height) {
  this.area = width * height;
}

public class Rectangle {
  private double area;

  public Rectangle(double width, double height) {
    this.area = width * height;
  }

  public double getArea() {
    return area;
  }
}

public class Circle {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  public double getArea() {
    return Math.PI * radius * radius;
  }
}
Circle as an ADT

- "Data" is a circle
- Instance variables are hidden
- Methods define in what ways the circle can be manipulated or examined

Similarities?

Interface as a Contract

```java
/** A 2-dimensional shape */
interface Shape {
    /** Returns the area of the shape. */
    @return the area of the shape */
    public double getArea();
}
```
public class Rectangle implements Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

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

The Supplier Side of the Contract

public class Rectangle implements Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

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

Must define this to satisfy the contract.

The Client Side of the Contract

Can use interface names as types

Can assign instance of implementing classes to variables declared with interface types

Shape someShape;

someShape = new Rectangle (3, 5);

double area = someShape.getArea();

Can call methods declared in the interface.

Shape someShape;

someShape = new Rectangle (3, 5);

double area = someShape.getArea();

Can call methods declared in the interface.
Polymorphism

public class ShapeArray {
    private Shape[] shapes = new Shape[10];
    private int nextIndex = 0;

    public void addShape(Shape newShape) {
        if (nextIndex < shapes.length) {
            shapes[nextIndex] = newShape;
            nextIndex++;
        }
    }

    public double getArea(int index) {
        if (index >= 0 && index < nextIndex) {
            return shapes[index].getArea();
        }
        return -1;
    }
}

Which entries in the array are circles???

Swing

- Package used to create Graphical User Interfaces (GUIs) in Java

<table>
<thead>
<tr>
<th>JButton</th>
<th>JComboBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCheckBox</td>
<td>JSlider</td>
</tr>
<tr>
<td>JTextField</td>
<td>JLabel</td>
</tr>
</tbody>
</table>

Displaying Buttons in Swing

- Construct a JButton, passing in its label
  JButton click = new JButton("Click me!");
- Add the button to a panel
  JPanel buttonPanel = new JPanel();
  buttonPanel.add(click);
- Add the panel to the display, no need to specify size or exact location
  getContentPane().add(buttonPanel, BorderLayout.CENTER);

Tuesday, January 29, 13
ActionListener Interface

```java
public interface ActionListener {
    public void actionPerformed (ActionEvent e);
}
```

- `actionPerformed` is the method that Java calls when the user clicks on a JButton.
- To handle button clicks, you must do 3 things:
  - Have your class "implement ActionListener"
  - Define the `actionPerformed` method
  - Attach your ActionListener object to the JButton: `myButton.addActionListener (this);`

Handling Button Clicks

- Tell Java where to find the event handler
  ```java
  click.addActionListener (this);
  ```
- Define the event handler
  ```java
  public class ClickMe implements ActionListener
  
  public void actionPerformed (ActionEvent event) {
      counter++;
      counterLabel.setText("" + counter);
  }
  ```

Swing Events

- Every event handling method has a parameter describing the event that occurred.
- The most useful method for us is:
  ```java
  public Object getSource()
  ```
- This returns the Swing component that the user interacted with.
- Useful if an event handling method handles events for more than 1 object.
BorderLayout Rules

- One component per compass point
- Component expands to fill the entire area

BorderLayout

- NORTH
- WEST
- EAST
- SOUTH

BorderLayout lays out components using compass directions + CENTER
Default for a JFrame’s content pane.

What Gives???

- There are 2 buttons in the EAST!
- The buttons do not fill the entire east area!
- Conclusion: My professor lied to me!
More on “adding” components

Adding a button to a panel:
```java
JPanel buttonPanel = new JPanel();
buttonPanel.add(clickButton);
```
- JPanel lays out its components like words on a page

Adding a panel to the frame:
```java
getContentPane().add(buttonPanel, BorderLayout.CENTER);
```
- contentPane lays out its components using compass directions: NORTH, SOUTH, EAST, WEST and CENTER

Grouping Components with JPanel

```java
JPanel buttonPanel = new JPanel();
buttonPanel.add(new JButton("Place order"));
buttonPanel.add(new JButton("Cancel order"));
contentPane.add(buttonPanel, BorderLayout.EAST);
```

FlowLayout

- A JPanel uses FlowLayout as its layout manager rather than BorderLayout.
- With BorderLayout, we say:
  ```java
  contentPane.add (buttonPanel, BorderLayout.SOUTH);
  ```
- With FlowLayout, we omit the compass direction:
  ```java
  buttonPanel.add(orderButton);
  ```
- Components are added from left to right.
FlowLayout

There are 2 buttons in the button panel.
There is 1 button panel in the EAST.
FlowLayout sizes components in a more natural way.
Conclusion: My professor didn't lie to me!

JPanel gives "natural" size

cContentPane.add(comboBox, BorderLayout.SOUTH);
toppingsPanel.add (comboBox);
cContentPane.add(toppingsPanel, BorderLayout.SOUTH);