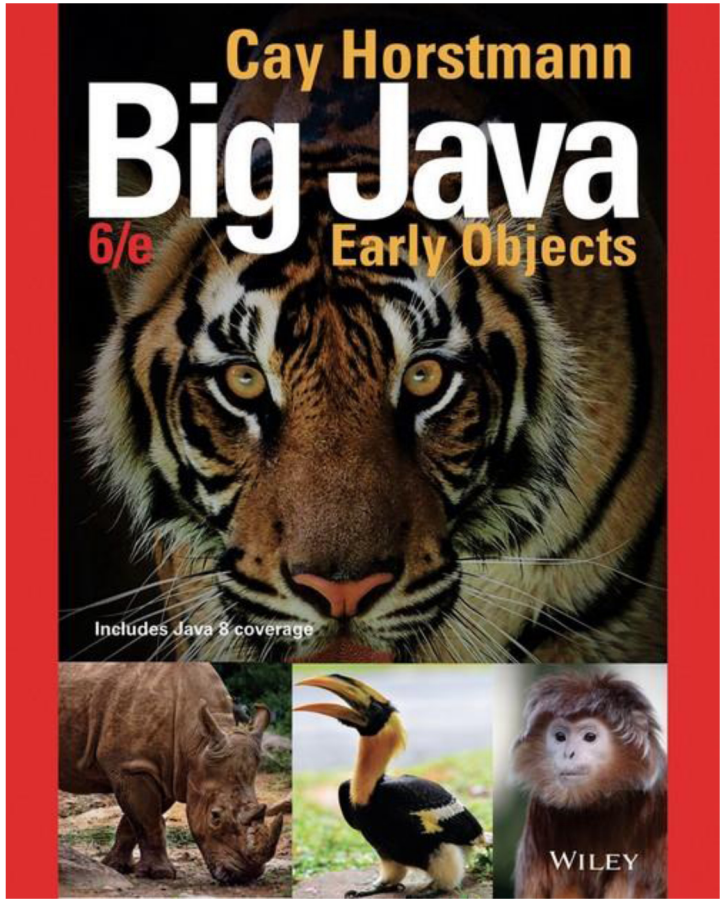

Chapter 10 – Interfaces and Polymorphism Part II (Optional)

Using Interfaces for Callbacks

- Limitations of `Measurable` interface:
  - Can add `Measurable` interface only to classes under your control
  - Can measure an object in only one way
    - E.g., cannot analyze a set of cars by both speed and price
- **Callback:** a mechanism for specifying code that is executed at a later time.
- Problem: the responsibility of measuring lies with the added objects themselves.
- Alternative: give the average method both the data to be averaged and a method of measuring.
- Create an interface:

```
public interface Measurer
{
   double measure(Object anObject);
}
```

- All objects can be converted to `Object`.

Using Interfaces for Callbacks

- The code that makes the call to the callback receives an object of class that implements this interface:

```
public static double average(Object[] objects, Measurer meas)
{
  double sum = 0;
  for (Object obj : objects)
  {
    sum = sum + meas.measure(obj);
  }
  if (objects.length > 0) {
  return sum / objects.length;
  }  else { return 0; }
}
```

- The average method simply makes a callback to the measure method whenever it needs to measure any object.

Using Interfaces for Callbacks

- A specific callback is obtained by implementing the `Measurer` interface:

```
public class AreaMeasurer implements Measurer
{
  public double measure(Object anObject)
  {
    Rectangle aRectangle = (Rectangle) anObject;
    double area =
    aRectangle.getWidth() *
    aRectangle.getHeight();  return
    area;
  }
}
```

- Must cast from `Object` to `Rectangle`:

```
Rectangle aRectangle = (Rectangle) anObject;
```

Using Interfaces for Callbacks

- To compute the average area of rectangles:

  construct an object of the `AreaMeasurer` class and pass it to the average method:

```
Measurer areaMeas = new AreaMeasurer();
Rectangle[] rects = { new Rectangle(5, 10, 20,
     30), new Rectangle(10, 20, 30, 40) };
double averageArea = average(rects, areaMeas);
```

- The `average` method will ask the `AreaMeasurer` object to measure the rectangles.

Using Interfaces for Callbacks

- The `Data` class (which holds the `average` method) is decoupled from the class whose objects it processes (`Rectangle`).
- You provide a small "helper" class `AreaMeasurer`, to process rectangles.

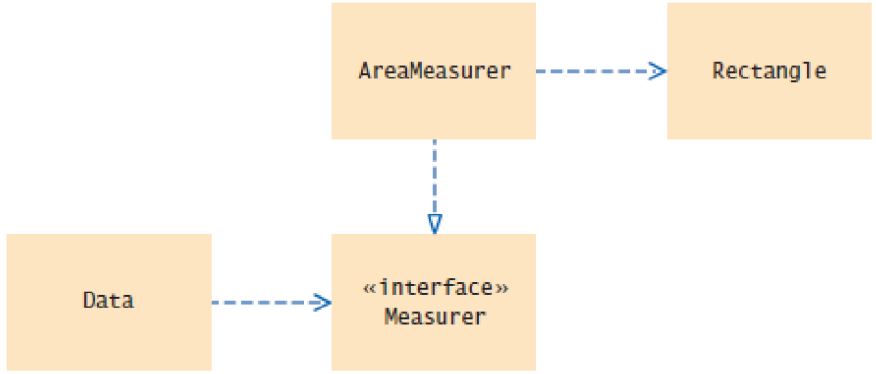

Figure 6 UML Diagram of the `Data` Class and the `Measurer` Interface

section_4/Measurer.java

```
1  /**
2     Describes any class whose objects can measure other objects.
3  */
4  public interface Measurer
5  {
6     /**
7        Computes the measure of an object.
8        @param anObject the object to be measured
9        @return the measure
10    */
11    double measure(Object anObject);
12 }
```

section_4/AreaMeasurer.java

```
1  import java.awt.Rectangle;
2
3  /**
4     Objects of this class measure rectangles by area.
5  */
6  public class AreaMeasurer implements Measurer
7  {
8     public double measure(Object anObject)
9     {
10       Rectangle aRectangle = (Rectangle) anObject;
11       double area = aRectangle.getWidth() * aRectangle.getHeight();
12       return area;
13    }
14 }
```

section_4/Data.java

```
1  public class Data
2  {
3     /**
4        Computes the average of the measures of the given objects.
5        @param objects an array of objects
6        @param meas the measurer for the objects
7        @return the average of the measures
8     */
9     public static double average(Object[] objects, Measurer meas)
10    {
11       double sum = 0;
12       for (Object obj : objects)
13       {
14          sum = sum + meas.measure(obj);
15       }
16       if (objects.length > 0) { return sum / objects.length; }
17       else { return 0; }
18    }
19 }
```

section_4/MeasurerTester.java

```
1  import java.awt.Rectangle;
2
3  /**
4     This program demonstrates the use of a Measurer.
5  */
6  public class MeasurerTester
7  {
8     public static void main(String[] args)
9     {
10       Measurer areaMeas = new AreaMeasurer();
11
12
13       Rectangle[] rects = new Rectangle[]
14          {
15             new Rectangle(5, 10, 20, 30),
16             new Rectangle(10, 20, 30, 40),
17             new Rectangle(20, 30, 5, 15)
18          };
19
20       double averageArea = Data.average(rects, areaMeas);
21       System.out.println("Average area: " + averageArea);
22       System.out.println("Expected: 625");
23 }   }
```

Program Run:

```
Average area: 625
Expected: 625
```

Self Check 10.16

Suppose you want to use the `average` method of Section 10.1 to find the average length of `String` objects. Why can't this work?

Answer: The `String` class doesn't implement the `Measurable` interface.

Self Check 10.17

How can you use the `average` method of this section to find the average length of `String` objects?

Answer: Implement a class `StringMeasurer` that implements the `Measurer` interface.

Self Check 10.18

Why does the `measure` method of the `Measurer` interface have one more argument than the `getMeasure` method of the `Measurable` interface?

Answer: A measurer measures an object, whereas `getMeasure` measures "itself", that is, the implicit parameter.

Self Check 10.19

Write a method `max` with three arguments that finds the larger of any two objects, using a `Measurer` to compare them.

Answer:

```
public static Object max(Object a, Object b, Measurer m)
{
   if (m.getMeasure(a) > m.getMeasure(b))
   {
      return a;
   }
   else { return b; }
}
```

Self Check 10.20

Write a call to the method of Self Check 19 that computes the larger of two rectangles, then prints its width and height.

Answer:

```
Rectangle larger = (Rectangle) max(first, second, areaMeas);
System.out.println(larger.getWidth() + " by " + larger.getHeight());
```

Lambda Expressions

- Using a method such as `average` is too much work
- That's why you don't find many such methods in the standard library
- Java 8 makes this much easier: Use a *lambda expression*
- Works with interfaces that have a *single abstract method*

  Such interfaces are called *functional interfaces...*

  ...because instances are similar to mathematical functions

- Lambda expression specifies:

  Parameters

  Code for computing the returned value

- Example of a lambda expression: A function that gets the balance of a bank account

```
(Object obj) -> ((BankAccount) obj).getBalance()
```

  Name “lambda expression” comes from a mathematical notation that uses the greek letter lambda (λ) instead of the -> symbol

Lambda Expressions 2

- In Java, a lambda expression cannot stand alone.
- It *must* be assigned to a variable whose type is a functional interface:

```
Measurer accountMeas =
   (Object obj) -> ((BankAccount) obj).getBalance();
```

- Now the following actions occur:
  1. A class is defined that implements the functional interface. The single abstract method is defined by the lambda expression.
  2. An object of that class is constructed.
  3. The variable is assigned a reference to that object.
- Can also pass lambda expression to a method:

```
double averageBalance = average(accounts,
   (Object obj) -> ((BankAccount) obj).getBalance());
```

Then the parameter variable is initialized with the object

Event Handling

- In an event-driven user interface, the program receives an event whenever the user manipulates an input component.

© Serly Tryapitsyn/iStockphoto.

- User interface **events** include key presses, mouse moves, button clicks, and so on.
- Most programs don't want to be flooded by irrelevant events.
- A program must indicate which events it needs to receive.

Event Handling

- **Event listeners:**
  - A program indicates which events it needs to receive by installing event listener objects
  - Belongs to a class provided by the application programmer
  - Its methods describe the actions to be taken when an event occurs
  - Notified when event happens
- **Event source:**
  - User interface component that generates a particular event
  - Add an event listener object to the appropriate event source
  - When an event occurs, the event source notifies all event listeners

Events Handling

- **Example:** A program that prints a message whenever a button is clicked.
- Button listeners must belong to a class that implements the `ActionListener` interface:

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

- Your job is to supply a class whose `actionPerformed` method contains the instructions that you want executed whenever the button is clicked.

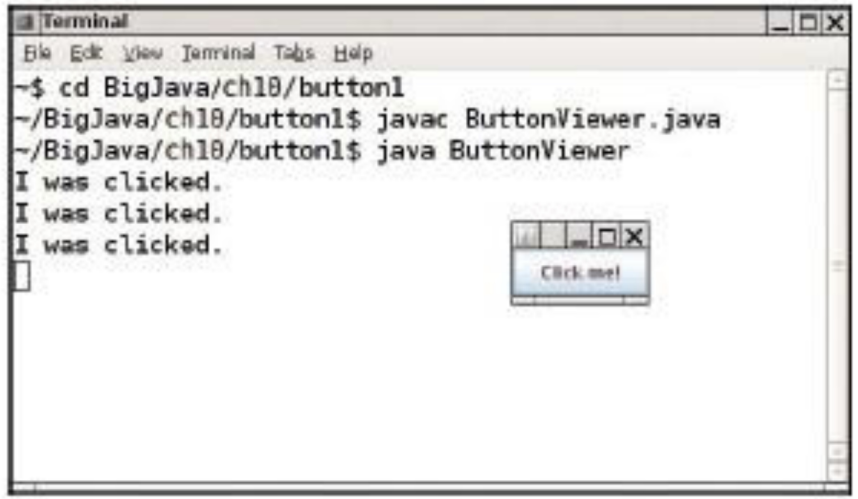

Figure 7 Implementing an Action Listener

section_7_1/ClickListener.java

```
1  import java.awt.event.ActionEvent;
2  import java.awt.event.ActionListener;
3
4  /**
5     An action listener that prints a message.
6  */
7  public class ClickListener implements ActionListener
8  {
9     public void actionPerformed(ActionEvent event)
10    {
11       System.out.println("I was clicked.");
12    }
13 }
```

Event Handling - Listening to Events

- `event` parameter of `actionPerformed` contains details about the event, such as the time at which it occurred.
- Construct an object of the listener and add it to the button:

```
ActionListener listener = new ClickListener();
button.addActionListener(listener);
```

- Whenever the button is clicked, it calls:

```
listener.actionPerformed(event);
```

  And the message is printed.

- Similar to a callback
- Use a `JButton` component for the button; attach an `ActionListener` to the button.

section_7_1/ButtonViewer.java

```
1  import java.awt.event.ActionListener;
2  import javax.swing.JButton;
3  import javax.swing.JFrame;
4
5  /**
6     This program demonstrates how to install an action listener.
7  */
8  public class ButtonViewer
9  {
```

Using Inner Classes for Listeners

- Implement simple listener classes as inner classes like this:

```
JButton button = new JButton(". . .");

// This inner class is declared in the same method as the button variable
class MyListener implements ActionListener
{
  . . .
}
ActionListener listener = new MyListener();
button.addActionListener(listener);
```

- Advantages

  Places the trivial listener class exactly where it is needed, without cluttering up the remainder of the project

  Methods of an inner class can access instance variables and methods of the surrounding class:

Using Inner Classes for Listeners

- Local variables that are accessed by an inner class method must be declared as `final` (or in Java 8, `effectively final` [not modified after initialized]).
- **Example:** add interest to a bank account whenever a button is clicked:

```
JButton button = new JButton("Add Interest");
frame.add(button);
final BankAccount account = new BankAccount(INITIAL_BALANCE);

class AddInterestListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      // The listener method accesses the account variable
      // from the surrounding block
      double interest = account.getBalance() * INTEREST_RATE / 100;
      account.deposit(interest);
   }
}
ActionListener listener = new AddInterestListener();
button.addActionListener(listener);
```

section_7_2/InvestmentViewer1.java

```
1  import java.awt.event.ActionEvent;
2  import java.awt.event.ActionListener;
3  import javax.swing.JButton;
4  import javax.swing.JFrame;
5
6  /**
7     This program demonstrates how an action listener can access
8     a variable from a surrounding block.
9  */
10 public class InvestmentViewer1
11 {
12    private static final int FRAME_WIDTH = 120;
13    private static final int FRAME_HEIGHT = 60;
14
15
16    private static final double INTEREST_RATE = 10;
17    private static final double INITIAL_BALANCE = 1000;
18
19    public static void main(String[] args)
20    {
21       JFrame frame = new JFrame();
22
23       // The button to trigger the calculation
24       JButton button = new JButton("Add Interest");
25       frame.add(button);
26
27       // The application adds interest to this bank account
28       final BankAccount account = new BankAccount(INITIAL_BALANCE);
29
30       class AddInterestListener implements ActionListener
31       {
32          public void actionPerformed(ActionEvent event)
33          {
34             // The listener method accesses the account variable
35             // from the surrounding block
```

Program Run:

```
balance: 1100.0
balance: 1210.0
balance: 1331.0
balance: 1464.1
```

```
double interest = account.getBalance() * INTEREST_RATE / 100;
```

Self Check 10.26

Which objects are the event source and the event listener in the `ButtonViewer` program?

Answer: The `button` object is the event source. The `listener` object is the event listener.

Self Check 10.27

Why is it legal to assign a `ClickListener` object to a variable of type `ActionListener`?

Answer: The `ClickListener` class implements the `ActionListener` interface.

Self Check 10.28

When do you call the `actionPerformed` method?

Answer: You don't. It is called whenever the button is clicked.

Self Check 10.29

Why would an inner class method want to access a variable from a surrounding scope?

Answer: Direct access is simpler than the alternative — passing the variable as an argument to a constructor or method.

Self Check 10.30

If an inner class accesses a local variable from a surrounding scope, what special rule applies?

Answer: The local variable must not change. Prior to Java 8, it must be declared as `final`.

Building Applications with Buttons

- **Example:** investment viewer program; whenever button is clicked, interest is added, and new balance is displayed:

- Construct an object of the `JButton` class:

```
JButton button = new JButton("Add Interest");
```

- We need a user interface component that displays a message:

```
JLabel label = new JLabel("balance: " + account.getBalance());
```

- Use a `JPanel` container to group multiple user interface components together:

```
JPanel panel = new JPanel();
panel.add(button);
panel.add(label);
frame.add(panel);
```

Event Handling

Whenever a button is pressed, the `actionPerformed` method is called on all listeners.

Specify button click actions through classes that implement the `ActionListener` interface.

Building Applications with Buttons

- AddInterestListener class adds interest and displays the new balance:

```
class AddInterestListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      double interest = account.getBalance() * INTEREST_RATE / 100;
      account.deposit(interest);
      label.setText("balance=" + account.getBalance());
   }
}
```

- Add `AddInterestListener` as inner class so it can have access to surrounding variables (prior to Java 8, `account` and `label` must be declared `final`).

section_8/InvestmentViewer2.java

```
1  import java.awt.event.ActionEvent;
2  import java.awt.event.ActionListener;
3  import javax.swing.JButton;
4  import javax.swing.JFrame;
5  import javax.swing.JLabel;
6  import javax.swing.JPanel;
7
8  /**
9     This program displays the growth of an investment.
```

Self Check 10.31

How do you place the "`balance: . . .`" message to the left of the "`Add Interest`" button?

Answer: First add `label` to the panel, then add `button`.

Self Check 10.32

Why was it not necessary to declare the `button` variable as `final`?

Answer: The `actionPerformed` method does not access that variable.

Processing Timer Events

- `javax.swing.Timer` generates equally spaced timer events, sending events to installed action listeners.
- Useful whenever you want to have an object updated in regular intervals.
- Declare a class that implements the `ActionListener` interface:

```
class MyListener implements ActionListener
{
   void actionPerformed(ActionEvent event)
   {
       Listener action (executed at each timer event)
   }
}
```

- To create a timer, specify the frequency of the events and an object of a class that implements the `ActionListener` interface:

```
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```

section_9/RectangleComponent.java

Displays a rectangle that moves

The `repaint` method causes a component to repaint itself. Call this method whenever you modify the shapes that the `paintComponent` method draws.

```
1  import java.awt.Graphics;
2  import java.awt.Graphics2D;
3  import java.awt.Rectangle;
4  import javax.swing.JComponent;
5
6  /**
7     This component displays a rectangle that can be moved.
8  */
9  public class RectangleComponent extends JComponent
```

section_9/RectangleFrame.java

```
1  import java.awt.event.ActionEvent;
2  import java.awt.event.ActionListener;
3  import javax.swing.JFrame;
4  import javax.swing.Timer;
5
6  /**
7     This frame contains a moving rectangle.
8  */
9  public class RectangleFrame extends JFrame
```

section_9/RectangleViewer.java

```
1  import javax.swing.JFrame;
2
3  /**
4     This program moves the rectangle.
5  */
6  public class RectangleViewer
7  {
8     public static void main(String[] args)
9     {
```

Self Check 10.33

Why does a timer require a listener object?

Answer: The timer needs to call some method whenever the time interval expires. It calls the `actionPerformed` method of the listener object.

Self Check 10.34

What would happen if you omitted the call to `repaint` in the `moveBy` method?

Answer: The moved rectangles won't be painted, and the rectangle will appear to be stationary until the frame is repainted for an external reason.

Mouse Events

- Use a mouse listener to capture mouse events.
- Implement the `MouseListener` interface which has five methods:

```
public interface MouseListener
{
  void mousePressed(MouseEvent event);
     // Called when a mouse button has been pressed on a component
     void mouseReleased(MouseEvent event);
     // Called when a mouse button has been released on a component
     void mouseClicked(MouseEvent event);
     // Called when the mouse has been clicked on a component
     void mouseEntered(MouseEvent event);
     // Called when the mouse enters a component
     void mouseExited(MouseEvent event);
     // Called when the mouse exits a component
}
```

Mouse Events

- Add a mouse listener to a component by calling the `addMouseListener` method:

```
public class MyMouseListener implements MouseListener
{
   // Implements five methods
}
MouseListener listener = new MyMouseListener();
component.addMouseListener(listener);
```

- Sample program: enhance `RectangleComponent` – when user clicks on rectangle component, move the rectangle to the mouse location.

section_10/RectangleComponent2.java

First add a `moveRectangle` method to `RectangleComponent`:

```
1  import java.awt.Graphics;
2  import java.awt.Graphics2D;
3  import java.awt.Rectangle;
4  import javax.swing.JComponent;
5
6  /**
7     This component displays a rectangle that can be moved.
8  */
9  public class RectangleComponent2 extends JComponent
```

Mouse Events

- Call `repaint` to tell the component to repaint itself and show the rectangle in its new position.
- When the mouse is pressed,the mouse listener moves the rectangle to the mouse location:

```
class MousePressListener implements MouseListener
{
   public void mousePressed(MouseEvent event)
   {
      int x = event.getX();
      int y = event.getY();
      component.moveTo(x, y);
   }
   // Do-nothing methods
   public void mouseReleased(MouseEvent event) {}
   public void mouseClicked(MouseEvent event) {}
   public void mouseEntered(MouseEvent event) {}
   public void mouseExited(MouseEvent event) {}
}
```

- All five methods of the interface must be implemented; unused methods can be empty.

RectangleViewer2 Program Run

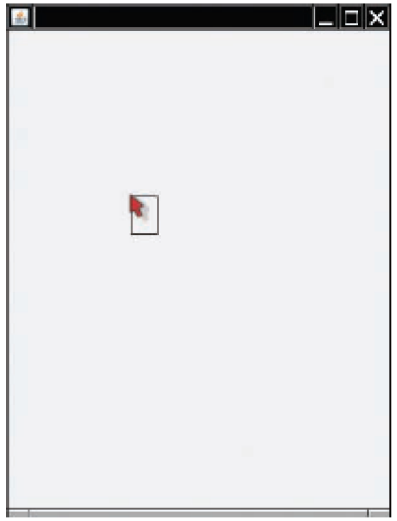

Figure 9 Clicking the Mouse Moves the Rectangle

section_10/RectangleFrame2.java

```
1  import java.awt.event.MouseListener;
2  import java.awt.event.MouseEvent;
3  import javax.swing.JFrame;
4
5  /**
6     This frame contains a moving rectangle.
7  */
8  public class RectangleFrame2 extends JFrame
9  {
```

section_10/RectangleViewer2.java

```
1  import javax.swing.JFrame;
2
3  /**
4     This program displays a rectangle that can be moved with the
mouse.
5  */
6  public class RectangleViewer2
7  {
8     public static void main(String[] args)
9     {
```

Self Check 10.35

Why was the `moveRectangleBy` method in the `RectangleComponent` replaced with a `moveRectangleTo` method?

Answer: Because you know the current mouse position, not the amount by which the mouse has moved.

Self Check 10.36

Why must the `MousePressListener` class supply five methods?

Answer: It implements the `MouseListener` interface, which has five methods.