Lab 01: Interfaces and Polymorphism

Objectives

- Defining `interfaces` to enable reusability
- Understand the concept of `polymorphism`
- Implement helper classes as `inner classes`
- Implement `event listeners`

Implementing an interface

- Define an interface with functions that are generic to a set of classes.
- All classes that implements the interface needs to implement **all** the functions defined in the interface.
  - All the functions must have the `same signature` as defined in the interface.
  - The function must have `public` as access specifier.

In-lab exercises

* Go through self check questions (1-24)

1. Implement self check question 1: Page 355
   - Use the `DataSet` class to find the `Country` object with the largest population.
2. Using interfaces for callbacks: Page 361
   - Understand the use of callbacks to enable passing objects that have not implmented the interface as parameters.
   - Implement `DataSetTester2.java`. Update the program to find the longest `String` from a set of inputs.
3. Implementing inner classes for listeners: Page 373
   - Implement `BankAccount` class and run `InvestmentViewer1.java`
4. Implement `Timer` and `Mouse` events: Page 379
   - Integrate Timer listener and Mouse listener for `RectangleComponentViewer.java`

In-lab exercises

1. Implement self check question 1: Page 355

   - Use the `DataSet` class to find the `Country` object with the largest population.

     (a) Create `Measurable` interface

     ```
     public interface Measurable
     {
         double getMeasure();
     }
     ```

     (b) TODO: Create `Country` class that implements the Measurable interface. *Hint:*getMeasure() should return the popoulation of the Country object.

     (c) Create `DataSet` class that implements the add() and getMaximum() functions

     ```
     public class DataSet
     {
         // TODO: Define the private instance variables.
         ...

         public void add(Measurable x)
         {
             sum = sum + x.getMeasure();
             if (count == 0 || maximum.getMeasure() < x.getMeasure())
                 maximum = x;
             count++;
         }
         public Measurable getMaximum()
         {
             return maximum;
         }
     }
     ```

     (d) Create main class `DataSetTester` with the following input data.

     ```
     // TODO: Define the class and the entry point main function.
     ...

     DataSet countryData = new DataSet();
     countryData.add(new Country(19920));
     countryData.add(new Country(20000));
     countryData.add(new Country(45000));
     countryData.add(new Country(1100));
     Measurable max = countryData.getMaximum();
     System.out.println("Highest population: " + max.getMeasure());// dynamic method lookup
     // __Note__: You may type-cast the max object reference as follows:
     // (Uncomment the two lines below to verify).
     //Country country = (Country)max;
     //System.out.println("Highest population: " + country.getMeasure());
     System.out.println("Expected: 45000");
     ```

     (e) Run the program and check the results. Show the output result to the lab instructor.

2. Using interfaces for callbacks: Page 361

   - Understand the use of callbacks to enable passing objects that have not implmented the interface as parameters.
   - Implement `DataSetTester2.java` to identify `Rectangle` object with the maximum area.

     (a) Create `Measurer` interface

```
/**
Describes any class whose objects can measure other objects.
*/
public interface Measurer
{
    /**
    Computes the measure of an object.
    @param anObject the object to be measured
    @return the measure
    */
    double measure(Object anObject);
}
```

In your homework, please try to document your code the best way you can. You do not need to be very verbose but try to keep it short and simple, easy for the reader to comprehend.

(b) TODO: Create `RectangleMeasurer` class that implements the `Measurer` interface. *Hint: The RectangleMeasurer class should contain the functions with the same signature as specified in the interface. The measure() should return the perimeter of the rectangle.*

(c) Create `DataSet` class. Notice that in this class, the add() function has `Object` as paramter and the constructor has `Measurer` as parameter.

```
public class DataSet
{
    // TODO: Declare the private instance variables.
    ...

    public DataSet(Measurer aMeasurer)
    {
        sum = 0;
        count = 0;
        maximum = null;
        measurer = aMeasurer;
    }

    public void add(Object x)
    {
        sum = sum + measurer.measure(x);
        if (count == 0 || measurer.measure(maximum) < measurer.measure(x))
            maximum = x;
        count++;
    }

    public double getAverage()
    {
        // TODO: Check divide by zero. Compute the average value.
        ...
    }

    public Object getMaximum()
    {
        return maximum;
    }
}
```

(d) Create DataSetTester2.java with the following test case.

```
Measurer m = new RectangleMeasurer();
```

```
DataSet data = new DataSet(m);
// Can use object as parameter now because of enabling using interfaces for callbacks
// The add method now measures objects
data.add(new Rectangle(15, 10, 12, 30));
data.add(new Rectangle(40, 20, 30, 4));
data.add(new Rectangle(20, 10, 50, 15));
System.out.println("Average area: " + data.getAverage());
System.out.println("Expected: 410.0");
Rectangle max = (Rectangle) data.getMaximum();  // Type casting
System.out.println("Maximum area rectangle: " + max);
System.out.println("Expected: java.awt.Rectangle[x=10,y=20,width=30,height=40]");
```

(e) Run the DataSetTester2 program and show the output results to the lab instructor.

- Update the program to find the longest `String` from a set of inputs.

(a) TODO: Create `StringMeasurer` class that implements the `Measurer` interface. *Hint: The measure() should return the length of the string.*

(b) Create `DataSetTester3` class with the following test case.

```
Measurer m = new StringMeasurer();
DataSet data = new DataSet(m);
data.add( new String( "test" ) );
data.add( new String( "testing" ) );
data.add( new String( "tester" ) );
data.add( new String( "retest" ) );
data.add( new String( "contest" ) );
System.out.println("Average length: " + data.getAverage());
System.out.println("Expected: 6");
String maxlength = (String) data.getMaximum();  // Type casting
System.out.println("Maximum length string: " + maxlength);
System.out.println("Expected:testing" );
```

(c) Run the `DataSetTester2` program and show the output results to the lab instructor.

3. Implementing inner classes for listeners: Page 373

- Implement `BankAccount` class and run `InvestmentViewer1.java`

(a) Create `BankAccount` class.

```
public class BankAccount
{
    private double balance;
    public BankAccount()
    {
        balance = 0;
    }

    public BankAccount(double initialBalance)
    {
        balance = initialBalance;
    }

    public void deposit(double amount)
    {
        balance = balance + amount;
    }

    public void withdraw(double amount)
```

```
        {
            balance = balance - amount;
        }

        public double getBalance()
        {
            return balance;
        }
    }
```

(b) Create `InvestmentViewer1` class. This class creates an interface with a button which when clicked adds interest to the balance of the account.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

/**
This program demonstrates how an action listener can access
a variable from a surrounding block.
*/
public class InvestmentViewer1
{
    private static final int FRAME_WIDTH = 120;
    private static final int FRAME_HEIGHT = 60;
    private static final double INTEREST_RATE = 10;
    private static final double INITIAL_BALANCE = 1000;
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        // The button to trigger the calculation
        JButton button = new JButton("Add Interest");
        frame.add(button);

        // The application adds interest to this bank account
        final BankAccount account = new BankAccount(INITIAL_BALANCE);

        class AddInterestListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                //NOTE:
                // 1. The listener method accesses the account variable
                //    from the surrounding block
                // 2. The account variable needs to be define "final" reserved
                //    word.
                // 3. If this inside class is defined within static method
                //    it can only access static variables.
                double interest = account.getBalance() * INTEREST_RATE / 100;
                account.deposit(interest);
                // Output result to console.
                System.out.println("balance: " + account.getBalance());
            }
        }
```

```
            // TODO: Add AddInterestListener object to the button object.
            ...

            frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
        }
    }
```

(c) Run the `InvestmentViewer1` program and show the output results to the lab instructor.

(d) Create `InvestmentViewer2` class very similar to `InvestmentViewer1` class and make the following changes to display the balance in a label.

```
// The label for displaying the results
final JLabel label = new JLabel("balance: " + account.getBalance());
// The panel that holds the user-interface components
JPanel panel = new JPanel();
panel.add(button);
panel.add(label);
frame.add(panel);
class AddInterestListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        double interest = account.getBalance() * INTEREST_RATE / 100;
        account.deposit(interest);
        label.setText("balance: " + account.getBalance());
    }
}
```

4. Implement `Timer` and `Mouse` events: Page 379

- Integrate `Timer` listener and `Mouse` listener for `RectangleComponentViewer.java`

(a) Create `RectangleComponent` class to define Rectangle object.

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import javax.swing.JComponent;

public class RectangleComponent extends JComponent
{
    private static final int BOX_X = 100;
    private static final int BOX_Y = 100;
    private static final int BOX_WIDTH = 20;
    private static final int BOX_HEIGHT = 30;
    private Rectangle box;

    public RectangleComponent()
    {
        box = new Rectangle(BOX_X, BOX_Y, BOX_WIDTH, BOX_HEIGHT);
    }
    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;
        g2.draw(box);
    }
```

```
        public void moveBy(int dx, int dy)
        {
            box.translate(dx, dy);
            repaint();  // Check: What happens if you do not specify repaint()?
        }

        public void moveTo(int x, int y)
        {
            box.setLocation(x, y);
            repaint();
        }
    }
```

(b) Create an interactive interface in `RectangleComponentViewer` class.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.Timer;
/**
This program moves the rectangle.
*/
public class RectangleMover
{
    private static final int FRAME_WIDTH = 300;
    private static final int FRAME_HEIGHT = 400;

    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setTitle("An animated rectangle");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final RectangleComponent component = new RectangleComponent();
        frame.add(component);
        frame.setVisible(true);

        // Add mouse press listener
        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) {}
        }
```

```
            // TODO: Add MousePressListener to component.
            ...

            class TimerListener implements ActionListener
            {
                public void actionPerformed(ActionEvent event)
                {
                    component.moveBy(1, 1);
                }
            }

            // TODO: Add TimerListener() object to Timer object.
            ...
        }
    }
```

(c) Run the animation program and show the results to the lab instructor.

Tip: If you see that no action is performed when you generate events, you should check if you have added the listener to the component. Also check if you have specified the `repaint` *function.*

Homework Exercises

1. Exercise P8.3 on page 390
2. Exercise P8.6 on page 391
3. Exercise P8.12 on page 391
4. Exercise P8.19 on page 393

Submission Guidelines

You should email the homework excercises to `icpgrader@gmail.com`. Please follow the following submission guidelines:

1. Create a zip file containing a top level folder consisting on your ID and lab number e.g., if your `student ID` is `106003` and you are submitting lab 7 then the folder name should be `106003-Lab07`.
2. Then inside the top level folder you must have a separate folder for every exercise, lets say if you have to submit five exercises then you must have `HW1`, `HW2`, `HW3`, `HW4`, and `HW5` folders inside the top level folder.
3. Every homework exercise folder must contain all Java files (souce code) involved to solve the exercise and a file name execute.sh with the following contents:

   ```
   javac *.java
   java <NAME_OF_CLASS_CONTAINS_MAIN_METHOD>
   ```

4. Make sure to email the zip file before starting the next lab session.

For programs, provide the source code and output in your report. Take care to use appropriate fonts for code vs. text.

Submit a hard copy of your report to the lab instructor **before the beginning of the next lab session**. Please use both sides of the paper and set font size = `8`.