**CS102A: Introduction to Computer Programming**

**Lab 13**

**Objectives:**

* Practice interface
* Practice basic GUI programming

**Part1. Interface**

**[Demo]**

First, **create** two **interfaces**, one of which represents **animals** that can speak and the other models **movable** things. Please put these two interfaces into two separate .java files.

Animal.java:

public interface Animal {  
 public abstract void speak();  
 }

Movable.java

public interface Movable {  
 public abstract void move();  
 }

Then we **create** two classes: **Human** and **Monkey**. We **make both classes implement the two interfaces**. Please also put the two classes into the corresponding .java files.

Human.java

public class Human implements Animal, Movable {  
 @Override  
 public void speak() {  
 System.*out*.println("Hello, I am a human :)");  
 }  
  
 @Override  
 public void move() {  
 System.*out*.println("I jumped three feet.");  
 }  
 }

Monkey.java

public class Monkey implements Animal, Movable {  
 @Override  
 public void speak() {  
 System.*out*.println("I am a monkey, aaaa!");  
 }  
  
 @Override  
 public void move() {  
 System.*out*.println("I climbed up a banana tree.");  
 }  
 }

After defining the interfaces and classes, we can write a main method to test them. **Interfaces can be used to declare references to point to objects that are of the interface types**.

Demo1.java

public class Demo1 {  
 public static void main(String[] args) {  
 Human human = new Human();  
 Monkey monkey = new Monkey();  
 Movable[ ] movables = {human, monkey};  
 Animal[ ] animals = {human, monkey};  
 System.*out*.println("\*\*\*\*\*Animal self-introduction\*\*\*\*\*");  
 for(Animal a : animals) a.speak();  
 System.*out*.println("\*\*\*\*\*Animal movement\*\*\*\*\*");  
 for(Movable m : movables) m.move();  
 }  
}

The main method would print:

Text

Description automatically generated

**[Exercise]**

**Step1:** Please define an interface Geometry to represent geometric objects (rectangles, circles, and etc.). The interface has three abstract methods:

1. A method area that takes no argument and returns the area (double type) of the geometric object.
2. A method perimeter that takes no argument and returns the perimeter (double type) of the geometric object.
3. A method draw that takes no argument and returns a printable Unicode character that represents the geometric object.

**Step 2:** Please define a class Rectangle that implements the interface Geometry. The class has the following fields and methods:

1. A field of double type that represents the height of the rectangle
2. A field of double type that represents the width of the rectangle
3. A constructor that takes two arguments of double type to initialize the two fields, respectively.
4. The class should implement the area method.
5. The class should implement the perimeter method.
6. The class should implement the draw method. The code point of a black rectangle character ▬ in Unicode table is 25AC (i.e., the literal value of the character is ‘\u25AC’).

**Step 3:** Please define a class Circle that implements the interface Geometry. The class has the following field and methods:

1. A field of double type that represents the radius of the circle
2. A constructor that takes one argument of double type to initialize the radius field
3. The class should implement the area method.
4. The class should implement the perimeter method.
5. The class should implement the draw method. The code point of a black circle character ● in Unicode table is 25CF (i.e., the literal value of the character is ‘\u25CF’).

Step 4: Create a class Exercise1 with the following main method to test the defined interface and classes:

public static void main(String[] args) {  
 Rectangle rect = new Rectangle(3.0, 2.0);  
 Circle circle = new Circle(3.0);  
 Geometry[] geometries = {rect, circle};  
 for(Geometry g: geometries){  
 System.*out*.printf("%c : perimeter = %.2f, area = %.2f\n",

g.draw(), g.perimeter(), g.area());  
 }  
}

If your implementation is correct, the main method would print:

Text, application

Description automatically generated

**Part2. GUI**

**[Demo]**

1. The following is a simple example of displaying a .jpg file with swing API.

**Copy the following code to** DisplayJpg**.java**

import javax.swing.\*;  
import java.awt.\*;  
  
public class DisplayJpg {  
 public static void main(String[] args)  
 {  
 JFrame window=new JFrame(); //create a Frame  
 //load a picture from computer  
 ImageIcon picture=new ImageIcon("C:\\swing.jpg");  
 JLabel label=new JLabel(picture); //add the picture to a label  
  
 window.add(label); //add the label to the frame  
 window.setVisible(true); //set the window to visible  
 window.setSize(400,400); //set the size of the window  
 //let the window can be close by click "x"  
 window.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 }  
 }

Save “a .jpg file” to your PC and modify the path to your .jpg file in the above code.

Compile and run the program.

Now if your .jpg file is too large, the window cannot display it in full size.

It was because we hardcode the size when we set the window:

window.setSize(400,400);

Can you modify the code so that it can set the window size to the size of your image?

Hints: Look for the functions from class **ImageIcon** to get back the size of the image.

Next, can you rescale your image to 50% of its size and display it?

Hints: obtain an object of class **Image** from your existing **ImageIcon** object and use getScaledInstance() from class **Image**. Check display2().

1. Fill in the code below to implement the following functions:
   * 1. Draw a circle in the center of the canvas (画布)
     2. Increase the radius of the circle by 10% with a click of the Enlarge button
     3. Decrease the radius of the circle by 10% with a click of the Shrink button.

import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
  
public class ControlCircle extends JFrame {  
 private JButton jbtEnlarge = new JButton("Enlarge");  
 private JButton jbtShrink = new JButton("Shrink");  
 private CirclePanel canvas = new CirclePanel();  
  
 public ControlCircle() {  
 JPanel panel = new JPanel(); // Use the panel to group buttons  
 panel.add(jbtEnlarge);  
 panel.add(jbtShrink);  
  
 this.add(canvas, BorderLayout.*CENTER*); // Add canvas to center  
 this.add(panel, BorderLayout.*SOUTH*); // Add buttons to the frame  
  
 /\* Fill in the code to listen to the action event \*/  
  
 }  
  
 */\*\* Main method \*/* public static void main(String[] args) {  
 JFrame frame = new ControlCircle();  
 frame.setTitle("ControlCircle2");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setSize(400, 400);  
  
 frame.setVisible(true);  
 }  
  
 class Listener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {

// Fill in the code to response the enlarge or shrink event

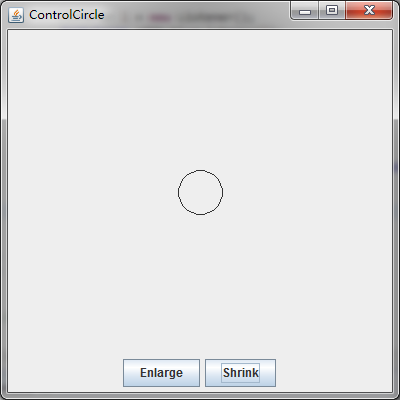
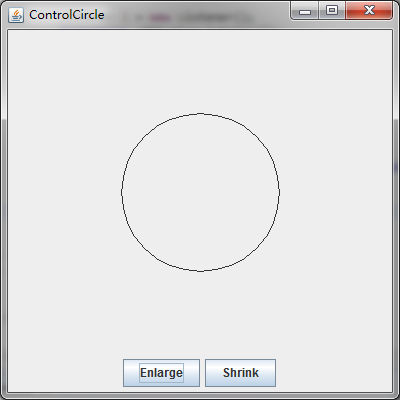
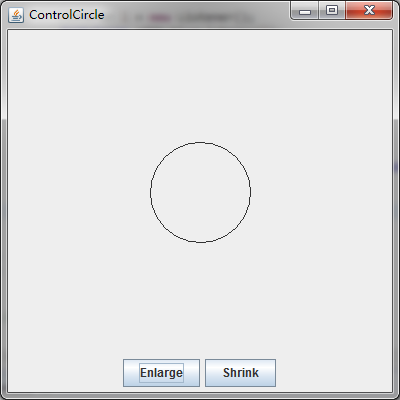
}  
 }  
}

class CirclePanel extends JPanel {  
private int radius = 50; // Default circle radius  
  
*/\*\* Enlarge the circle \*/*public void enlarge() {  
 radius = (int)(radius \* 1.1);  
 this.repaint();  
 }  
  
*/\*\* Enlarge the circle \*/*public void shrink() {  
 radius = (int)(radius \* 0.9);  
 this.repaint();  
 }  
  
*/\*\* Repaint the circle \*/*protected void paintComponent(Graphics g) {  
 super.paintComponent(g);

/\* Fill in the code to draw a circle in the center of the canvas with the radius of this class \*/

}  
}

Here is a sample run.



**[Exercise]**

1. Understand the following code and fill in the actionPerformed() method to implement the plus and minus operation.

import java.awt.BorderLayout;  
import java.awt.GridLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.JTextField;  
  
public class Calculation extends JFrame {  
 private JButton plainJButton1;  
 private JButton plainJButton2;  
 private JButton plainJButton3;  
 private JButton plainJButton4;  
 private JButton plainJButton5;  
 private JButton plainJButton6;  
 private JButton plainJButton7;  
 private JButton plainJButton8;  
 private JButton plainJButton9;  
 private JButton plainJButton0;  
 private JButton plainJButtonAdd;  
 private JButton plainJButtonSub;  
 private JButton plainJButtonEq;  
  
 private JTextField answer;  
  
 private String operation1 = "";  
 private String operation2 = "";  
 private String operator = "";  
  
 // ButtonFrame adds JButtons to JFrame  
 public Calculation() {  
 super( "Calculator" );  
 JPanel jp = new JPanel();  
 jp.setLayout( new GridLayout(4,4) );  
  
 plainJButton1 = new JButton( "1" );  
 jp.add( plainJButton1 );  
  
 plainJButton2 = new JButton( "2" );  
 jp.add( plainJButton2 );  
  
 plainJButton3 = new JButton( "3" );  
 jp.add( plainJButton3 );  
  
 plainJButton4 = new JButton( "4" );  
 jp.add( plainJButton4 );  
  
 plainJButton5 = new JButton( "5" );  
 jp.add( plainJButton5 );  
  
 plainJButton6 = new JButton( "6" );  
 jp.add( plainJButton6 );  
  
 plainJButton7 = new JButton( "7" );  
 jp.add( plainJButton7 );  
  
 plainJButton8 = new JButton( "8" );  
 jp.add( plainJButton8 );  
  
 plainJButton9 = new JButton( "9" );  
 jp.add( plainJButton9 );  
  
 plainJButton0 = new JButton( "0" );  
 jp.add( plainJButton0 );  
  
 plainJButtonAdd = new JButton( "+" );  
 jp.add( plainJButtonAdd );  
  
 plainJButtonSub = new JButton( "-" );  
 jp.add( plainJButtonSub );  
  
 plainJButtonEq = new JButton( "=" );  
 jp.add( plainJButtonEq );  
  
 add(jp, BorderLayout.*SOUTH*);  
  
 answer = new JTextField("");  
 answer.setEditable(false);  
 answer.setHorizontalAlignment(JTextField.*RIGHT*);  
 add(answer, BorderLayout.*CENTER*);  
  
 // create new ButtonHandler for button event handling  
 ButtonHandler handler = new ButtonHandler();  
 plainJButton1.addActionListener( handler );  
 plainJButton2.addActionListener( handler );  
 plainJButton3.addActionListener( handler );  
 plainJButton4.addActionListener( handler );  
 plainJButton5.addActionListener( handler );  
 plainJButton6.addActionListener( handler );  
 plainJButton7.addActionListener( handler );  
 plainJButton8.addActionListener( handler );  
 plainJButton9.addActionListener( handler );  
 plainJButton0.addActionListener( handler );  
 plainJButtonAdd.addActionListener( handler );  
 plainJButtonSub.addActionListener( handler );  
 plainJButtonEq.addActionListener( handler );  
 } // end ButtonFrame constructor  
  
 public int compute(String operation1, String operation2, String operator) {  
 int a = Integer.*parseInt*(operation1);  
 int b = Integer.*parseInt*(operation2);  
 if (operator.charAt(0) == '+') {  
 return a + b;  
 } else {  
 return a - b;  
 }  
 }  
  
  
 public static void main( String[] args ) {  
 Calculation calculationFrame = new Calculation(); // create ButtonFrame  
 calculationFrame.setDefaultCloseOperation( JFrame.*EXIT\_ON\_CLOSE* );  
 calculationFrame.setLocationRelativeTo(null);  
 calculationFrame.pack(); // set frame size  
 calculationFrame.setVisible( true ); // display frame  
 } // end main  
  
 // inner class for button event handling  
 private class ButtonHandler implements ActionListener {  
 // handle button event  
 public void actionPerformed( ActionEvent event ) {

// Fill in the code

} // end method actionPerformed  
 } // end private inner class ButtonHandler  
 } // end class ButtonFrame

Here is a sample run.

