Practical no. 4

FS19CO042

Aim:

4.1 Create a package com.gpm.complex. Create an interface Complex in it with following member methods: realPart(), imgPart(), magnitude() and argument() along with default methods plus(), minus(), into() and divideBy() having appropriate parameters and return types.

4.2 In the same package create class CartesianComplex with real and img and class PolarComplex with r and theta as their member fields. Make the classes implement the Complex interface. Override all non-default methods in the interface. Also override toString().

4.3 Now in main(), create one objects of both the classes defined in 4.2 and print their addition and multiplication.

4.4 Create a Java swing frame by creating a subclass of javax.swing.JFrame class. Add a java.awt.event.MouseListener by passing an object of an anonymous subclass of java.awt.event.MouseAdapter on the JFrame. Display the coordinates of point at which mouse is clicked

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
  }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

1. javac -d directory javafilename

For **example**

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output:Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
```

```
    }
  }
```

Output:Hello

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

Output:Hello

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Java JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

JFrame Example

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.Jpanel;
public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Code:

4.1 Create a package com.gpm.complex. Create an interface Complex in it with following member methods: realPart(), imgPart(), magnitude() and argument() along with default methods plus(), minus(), into() and divideBy() having appropriate parameters and return types.

Code :

```
package com.gpm.complex;
public interface Complex {
   void realPart();
   void imgPart();
   void magnitude();
   void argument();
   default float plus(float a, float b) {
      return a + b;
   }
   default float minus(float a, float b) {
      return a - b;
   }
   default float into(float a, float b) {
      return a * b;
   }
   default float divideBy(float a, float b) {
      return a / b;
   }
}
```

4.2 In the same package create class CartesianComplex with real and img and class PolarComplex with r and theta as their member fields. Make the classes implement the Complex interface. Override all non-default methods in the interface. Also override toString().

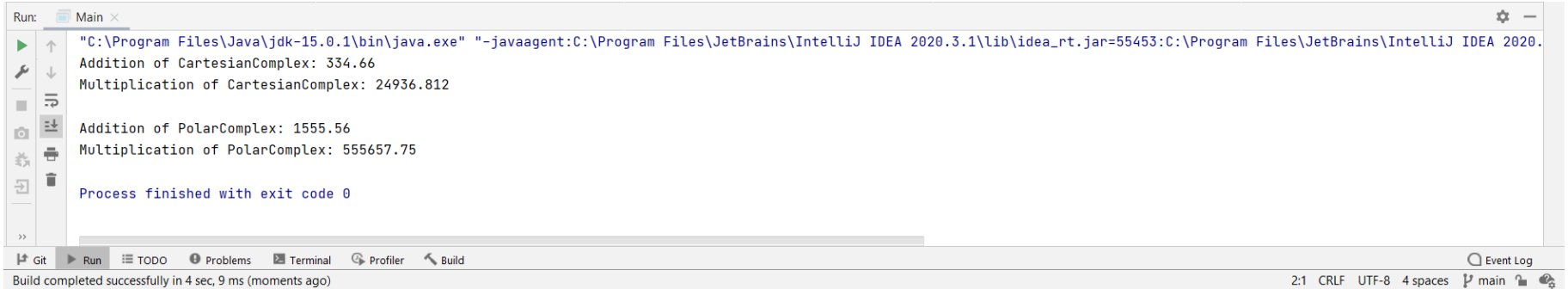
```
package com.gpm.complex;

public class CartesianComplex implements Complex {
   CartesianComplex real;
   CartesianComplex img;

   @Override
   public String toString() {
      return "CartesianComplex";
   }
   @Override
   public void realPart() {
   }
   @Override
   public void imgPart() {
   }
   @Override
   public void magnitude() {
   }
   @Override
   public void argument() {
   }
}
package com.gpm.complex;
public class PolarComplex implements Complex {
   PolarComplex r;
   PolarComplex theta;

   @Override
   public String toString() {
      return "PolarComplex";
   }
   @Override
   public void realPart() {
   }
   @Override
   public void imgPart() {
   }
   @Override
   public void magnitude() {
   }
   @Override
   public void argument() {
   }
}
```

4.3 Now in main(), create one objects of both the classes defined in 4.2 and print their addition and multiplication.

Code :

```
package com.gpm.complex;
public class Main {

   public static void main(String[] args) {

      CartesianComplex cartesianComplex = new CartesianComplex();
      PolarComplex polarComplex = new PolarComplex();

      System.out.println("Addition of CartesianComplex: "+cartesianComplex.plus(111.99f, 222.67f));
      System.out.println("Multiplication of CartesianComplex: "+cartesianComplex.into(111.99f, 222.67f)+"\n");

      System.out.println("Addition of PolarComplex: "+polarComplex.plus(555.78f, 999.78f));
      System.out.println("Multiplication of PolarComplex: "+polarComplex.into(555.78f, 999.78f));

   }
}
```

Output :



4.4 Create a Java swing frame by creating a subclass of javax.swing.JFrame class. Add a java.awt.event.MouseListener by passing an object of an anonymous subclass of java.awt.event.MouseAdapter on the JFrame. Display the coordinates of point at which mouse is clicked.

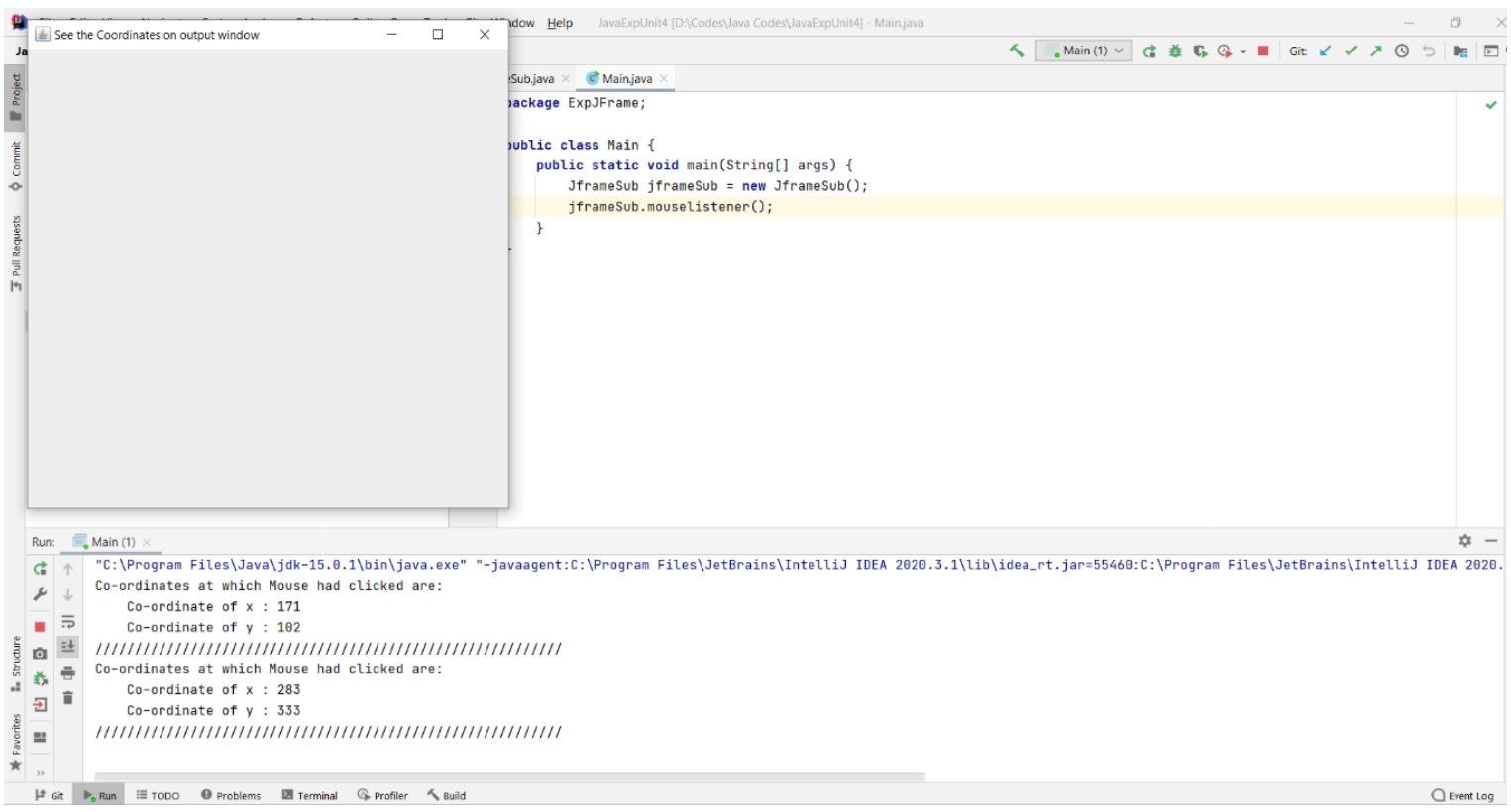
Code :

```
package ExpJFrame;

import javax.swing.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class JframeSub extends JFrame {

   public JframeSub() {

   }

   public void mouselistener() {

      addMouseListener(new MouseAdapter() {
         public void mousePressed(MouseEvent e) {
            int x = e.getX();
            int y = e.getY();
            System.out.println("Co-ordinates at which Mouse had clicked are: \n" +
                  "\tCo-ordinate of x : " + x +
                  "\n\tCo-ordinate of y : " + y);
            System.out.println("//////////////////////////////////////////////////////////////");
         }
      });

      setTitle("See the Coordinates on output window");
      setLayout(null);
      setVisible(true);
      setSize(500, 500);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   }
}
```

Main method :

```
package ExpJFrame;

public class Main {
   public static void main(String[] args) {
      JframeSub jframeSub = new JframeSub();
      jframeSub.mouselistener();
   }
}
```

Output :



Conclusion: Thus, we understood and executed programs regarding interfaces and swing framework.