LAB10 INTERFACES

Objective

In the lecture you were described interface features of Object Oriented Programming. This lab will help you understand the rules of defining programmer defined interfaces and how they are implemented. We will explain how interfaces allow subclasses to exhibit multiple inheritance which is not allowed normally in Java.

Understanding Interfaces

As you have already learned, objects define their interaction with the outside world through the methods that they expose. Methods form the object's interface with the outside world; the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off.

In its most common form, an interface is a group of related methods with empty bodies. A Vehicle's behavior, if specified as an interface, might appear as follows:

```
interface Vehicle {
                    void    move (int distance);
                    double  getDistanceTravelled( );
                    void    setSpeed ( double speed);
                    double  getSpeed( )  ;
                    void    setTopSpeed( double tspeed );
                    double  getTopSpeed( );
                    boolean isBroken( );
                    int     getSeats ( );
                    void    setSeats (int seats );
                    void    setPropulsionTechnology( String technology);
                    String  getPropulsionTechnology( ); // fueling mechanism
                    void    setPick( double pick);
                    double  getPick( );
}
```

To implement this interface, the name of your class would change (to Automobile, for example), and you'd use the implements keyword in the class declaration:

```
class Automobile implements Vehicle {

// All methods of the interface and remainder of this class implemented as before
}
```

Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile. A class can implement any number of interfaces it desires unlike inheritance in which only one class can be inherited. So in this way multiple inheritance is achieved through Interfaces.

EXERCISE

1. Modify the shape inheritance program developed in last lab to include following interfaces with shape class implementing all of them.

Colorable interface

```
interface Colorable
{
  // sets the color of the implementing object
        public void setColor( java.awt.Color c );
  // gets the color of the implementing object
        public java.awt.Color getColor();
 }
```

Rotateable interface

```
interface Rotatable
{
  // rotate the implementing object at angle degrees about point pt.
        void rotate(JPoint pt,double angle);
}
```

Translateable interface

```
interface Translateable
{
    // translate the object according to tx and ty.
        void translate(int tx, int ty);
}
```

Scaleable interface

```
interface Scaleable
{
    // Scale the object according to sx and sy.
        void scale(int sx,int sy);
}
```

Develop a runner program with the following specification

Create any shape object of any coordinates and set its colour to java.awt.Color.PINK. Rotate it in a circle around point (250 , 250) in steps of 10 degrees and redraw it after each step. During complete rotation translate shape according to (5 ,5) before each step. After that change the colour to java.awt.Color.RED and scale it by (2,1) and redraw it again.

Hint:

For Colorable interface use setBodyColor(java.awt.Color) & java.awt.Color getBodyColor( ) functions of Turtle Class.

Hint:

For Rotateable interface consider the point at the center of a circle with radius equal to distance between the point and the corresponding point defining the shape, rotate it according to rotation formulae about a fixed point and change the value of point defining the shape. Use same technique for all the points defining the shape.

.

Hint:

For Translateable interface use the following formulae to translate according to a point

$$X` = X + tx$$

$$Y` = Y + ty$$

Hint:

For Scaleable interface use the following formulae to scale according to values

$$X` = X * sx$$

$$Y` = Y * sy$$

2. You are asked to develop the password validation system for UET network. The specifications given are as follows:

   a) Develop an interface named Validator. It has only one function setPassword(String s).
   b) Develop a class named User which implements Validator. Its specifications are as follow:

It has following members

- **String UserID:** It stores user ID.
- **String Password:** It stores a valid user password.

It has following constructors

- **User(String userid)** : It initializes UserID with argument string and set Password to UET1234.
- **User(String userid,String password)** : It initializes UserID and Password. If password is not valid them Password is set to UET1234.

It has following member functions

- **Public void setPassword(String PWord)** : Same function implemented due to interface sets a valid Password.
- **Public boolean isValid (String PWord)** : Returns a value which indicates whether Password is valid or not

Validation Criteria

Following validation criteria should be fulfilled.

1. A password must have a minimum length of 8 characters
2. A password must have a minimum of three, of these types of characters.
   - An upper case character
   - A lower case character
   - A digit
   - A special character ! , @, #, $ or %.
3. A password must not match 5 consecutive characters compared without case with previous password
4. A password must not match 5 consecutive characters compared without case with UserID.

Develop a program which instantiate an object of User class and set different valid and invalid passwords and show class response.