Manavendra Sen | 199302058 | IT 3A

OOP LAB 6

EXERCISE 1:

Interface Movable and its implementations MovablePoint and MovableCircle

```
import java.util.*;

interface Movable {
 // All methods in an interface are by default public abstract
 public void moveUp();

 public void moveDown();

 public void moveLeft();

 public void moveRight();
}

class MovablePoint implements Movable, Comparable<MovablePoint> {
 int x, y, xSpeed, ySpeed; // Package Private by default

 // Constructor
 public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
   this.x = x;
   this.y = y;
   this.xSpeed = xSpeed;
   this.ySpeed = ySpeed;
 }

 @Override
 public String toString() {
   return ("Point at (" + x + "," + y + ")" + "xSpeed: " + xSpeed + " |
ySpeed: " + ySpeed);
 }
```

```
 // Implement abstract methods defined in the interface Movable
 public void moveUp() {
   y -= ySpeed;
 }

 public void moveDown() {
   y += ySpeed;
 }

 public void moveLeft() {
   x -= xSpeed;
 }

 public void moveRight() {
   x += xSpeed;
 }

 public int compareTo(MovablePoint point) {
   if (this.x == point.x) {
     return (this.y - point.y);
   } else {
     return (this.x - point.x);
   }
 }
}

class MovableCircle implements Movable {
 // instance variables
 private MovablePoint center;
 private int radius;

 // accessor for radius
 public int getRadius() {
   return radius;
 }
```

```java
public MovablePoint getCenter() {
  return center;
}

// Constructor
public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {
  // allocate the center
  this.center = new MovablePoint(x, y, xSpeed, ySpeed);
  this.radius = radius;
}

// Override abstract methods in Movable
@Override
public void moveUp() {
  center.y -= center.ySpeed;
}

@Override
public void moveDown() {
  center.y += center.ySpeed;
}

@Override
public void moveLeft() {
  center.x -= center.xSpeed;
}

@Override
public void moveRight() {
  center.x += center.xSpeed;
}
```

```
 public String toString() {
   return ("Center at (" + center.x + "," + center.y + ") xSpeed: " +
center.xSpeed + " | ySpeed: " + center.ySpeed
       + " | Radius: " + this.radius);
 }
}

// Comparators for sorting
class CompareByRadius implements Comparator<MovableCircle> {
 public int compare(MovableCircle m1, MovableCircle m2) {
   return (m1.getRadius() - m2.getRadius());
 }
}

class CompareByCenter implements Comparator<MovableCircle> {
 public int compare(MovableCircle m1, MovableCircle m2) {
   return (m1.getCenter().compareTo(m2.getCenter()));
 }
}

class CompareByRadiusCenter implements Comparator<MovableCircle> {
 public int compare(MovableCircle m1, MovableCircle m2) {
   if (m1.getRadius() == m2.getRadius())
     return (m1.getCenter().compareTo(m2.getCenter()));
   else
     return (m1.getRadius() - m2.getRadius());
 }
}
```

```java
class TestMovableSecond {
 public static void main(String[] args) {

   // Making a sample point at origin
   System.out.println("Movable Point ->");
   Movable m1 = new MovablePoint(5, 6, 10, 20); // upcast
   System.out.println(m1);
   m1.moveLeft();
   System.out.println(m1);

   System.out.println("\nMovable Circle ->");
   Movable m2 = new MovableCircle(2, 1, 2, 20, 25); // upcast
   System.out.println(m2);
   m2.moveRight();
   System.out.println(m2);

 }
}
```

```
mavn:Java/ $ javac LAB6_1.java                    [2:36:17]
mavn:Java/ $ java TestMovableSecond               [2:36:20]
Movable Point ->
Point at (5,6)xSpeed: 10 | ySpeed: 20
Point at (-5,6)xSpeed: 10 | ySpeed: 20

Movable Circle ->
Center at (2,1) xSpeed: 2 | ySpeed: 20 | Radius: 25
Center at (4,1) xSpeed: 2 | ySpeed: 20 | Radius: 25
mavn:Java/ $                                      [2:36:22]
```

EXERCISE 2

Using Interface and ArrayList

a. Define an interface named ‘BinaryInterface’
b. Define the class name ‘OneComplement’ which encapsulates One’s complement binary representation of decimal number. This class also implements interface ‘BinaryInterface’
c. Define the class name ‘SignedMagnitude’ which encapsulates One’s complement binary representation of decimal number. This class also implements interface ‘BinaryInterface’
d. Define the class ‘TwoComplement’ which encapsulates One’s complement binary representation of decimal numbers. This class also implements interface ‘BinaryInterface’.
e. Write the suitable Driver class named BinaryTest to show Runtime Polymorphism Approach.

Sample Output:

One's Complement of n = -16 is [1, 1, 1, 0, 1, 1, 1, 1]
Two'sComplement of n = -16 is [1, 1, 1, 1, 0, 0, 0, 0]
Sign Magnitude of n = -16 is [1, 0, 0, 1, 0, 0, 0, 0]

```
import java.util.*;

interface BinaryInterface {
 public void toBinary(int n);
}

class SignedMagnitude implements BinaryInterface {
 ArrayList<Integer> a = new ArrayList<Integer>();
 int temp;

 public void toBinary(int x) {
   int n = Math.abs(x);
   if (x < 0) {
     a.add(1);
   } else {
     a.add(0);
   }
   while (n != 0) {
```

```
    temp = n % 2;
    a.add(1, temp);
    n = n / 2;
  }
  int k = a.size();
  for (int i = 0; i < (8 - k); i++) {
    a.add(1, 0);
  }
  System.out.println(a);
 }
}

class OneComplement implements BinaryInterface {
 ArrayList<Integer> a = new ArrayList<Integer>();
 int temp;

 public void toBinary(int x) {
  int n = Math.abs(x);
  while (n != 0) {
    temp = n % 2;
    a.add(0, temp);
    n = n / 2;
  }
  int k = a.size();
  for (int i = 0; i < (8 - k); i++) {
    a.add(0, 0);
  }
  for (int z = 0; z < a.size(); z++) {
    if (a.get(z) == 0) {
      a.set(z, 1);
    } else {
      a.set(z, 0);
    }
  }
  System.out.println(a);
 }
```

```
}

class TwoComplement implements BinaryInterface {
 ArrayList<Integer> a = new ArrayList<Integer>();
 int temp;

 public void toBinary(int x) {
   int n = Math.abs(x);
   while (n != 0) {
     temp = n % 2;
     a.add(0, temp);
     n = n / 2;
   }
   int k = a.size();
   for (int i = 0; i < (8 - k); i++) {
     a.add(0, 0);
   }
   int flag = 0;
   for (int i = a.size() - 1; i >= 0; i--) {
     if (flag == 1) {
       if (a.get(i) == 0) {
         a.set(i, 1);
       } else {
         a.set(i, 0);
       }
     }
     if (a.get(i) == 1) {
       flag = 1;
     }
   }
   System.out.println(a);
 }
}
```

```java
class BinaryTest {
 public static void main(String args[]) {

   OneComplement num2 = new OneComplement();
   System.out.print("One's Complement of -16 is ");
   num2.toBinary(-16);

   TwoComplement num3 = new TwoComplement();
   System.out.print("Two's Complement of -16 is ");
   num3.toBinary(-16);

   SignedMagnitude num1 = new SignedMagnitude();
   System.out.print("Sign Magnitude of -16 is ");
   num1.toBinary(-16);

 }
}
```

```
mavn:Java/ $ javac LAB6_2.java                    [2:46:55]
mavn:Java/ $ java BinaryTest                      [2:46:57]
One's Complement of -16 is [1, 1, 1, 0, 1, 1, 1, 1]
Two's Complement of -16 is [1, 1, 1, 1, 0, 0, 0, 0]
Sign Magnitude of -16 is [1, 0, 0, 1, 0, 0, 0, 0]
mavn:Java/ $                                      [2:48:03]
```

```
mavn:Java/ $ javac LAB6_2.java                    [3:22:26]
mavn:Java/ $ java BinaryTest                      [3:22:29]
One's Complement of -20 is [1, 1, 1, 0, 1, 0, 1, 1]
Two's Complement of -20 is [1, 1, 1, 0, 1, 1, 0, 0]
Sign Magnitude of -20 is [1, 0, 0, 1, 0, 1, 0, 0]
mavn:Java/ $                                      [3:22:30]
```