Day 9 - Core Java / Functional Programming

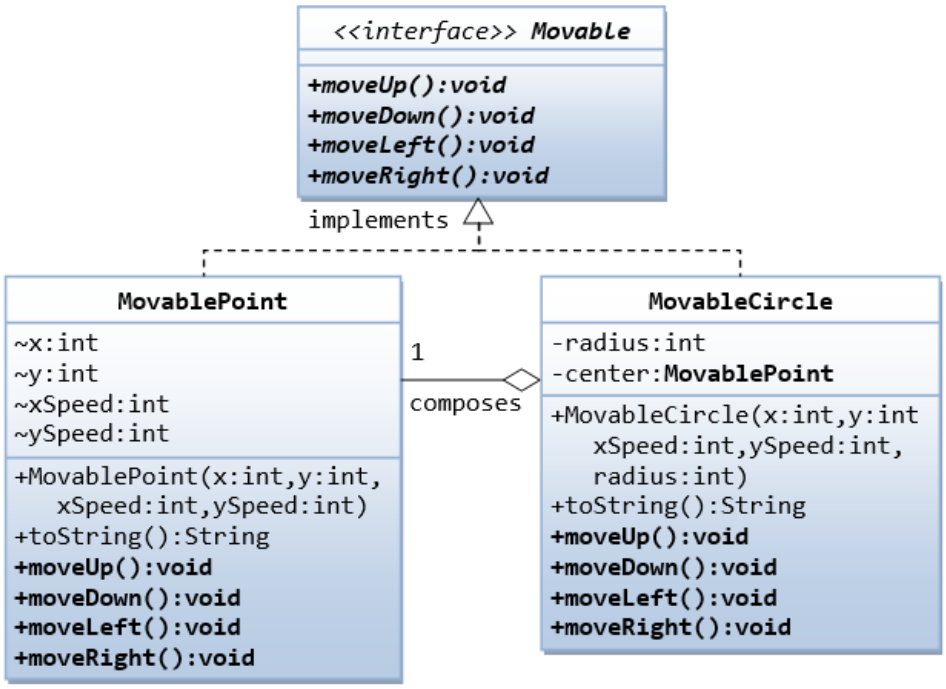
Topics:

- Introduction to functional programming
- Functional Interfaces with Lambdas
- Runnable interfaces with lambdas
- Built in Comparators
- Consumer Interface
- forEach
- Predicate Interfaces
- Method References
- Lambda Expressions

Assignments:

Interface *Movable* and its implementations *MovablePoint* and *MovableCircle*

Write two concrete classes - MovablePoint and MovableCircle - that implement the Movable interface.



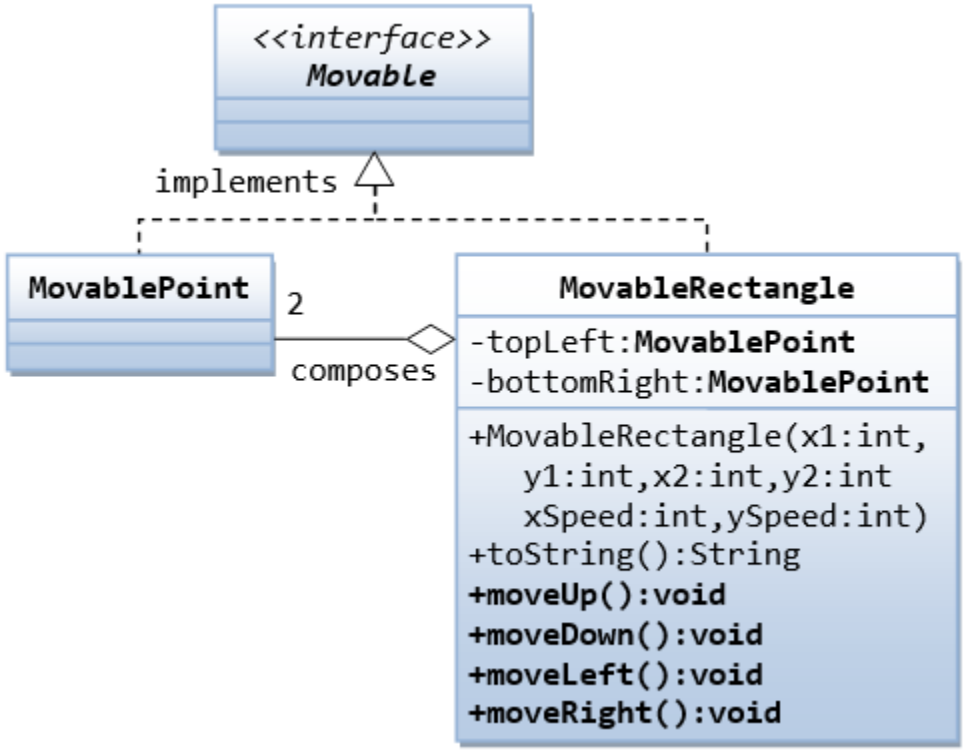
For the MovablePoint class, declare the instance variables x, y, xSpeed and ySpeed with package access as shown with '~' in the class diagram (i.e., classes in the same package can access these variables directly). For the MovableCircle class, use a MovablePoint to represent its center (which contains four variables x, y, xSpeed and ySpeed). In other words, the MovableCircle composes a MovablePoint, and its radius.

```
public class MovablePoint implements Movable {
   // instance variables
   int x, y, xSpeed, ySpeed;     // package access

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

   // Implement abstract methods declared in the interface Movable
   @Override
   public void moveUp() {
      y -= ySpeed;   // y-axis pointing down for 2D graphics
   }
   ......
}

public class MovableCircle implements Movable { // saved as
"MovableCircle.java"
   // instance variables
   private MovablePoint center;   // can use center.x, center.y directly
                                  //  because they are package accessible
   private int radius;

   // Constructor
   public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {
      // Call the MovablePoint's constructor to allocate the center instance.
      center = new MovablePoint(x, y, xSpeed, ySpeed);
      ......
   }
   ......

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

Write a test program and try out these statements:

```
Movable m1 = new MovablePoint(5, 6, 10, 15);     // upcast
System.out.println(m1);
m1.moveLeft();
System.out.println(m1);

Movable m2 = new MovableCircle(1, 2, 3, 4, 20);  // upcast
System.out.println(m2);
m2.moveRight();
System.out.println(m2);
```

Write a new class called *MovableRectangle*, which composes two *MovablePoint* objects (representing the top-left and bottom-right corners) and implements the *Movable* Interface. Make sure that the two points have the same speed.



In the test program, add the following code to test the new implementation:

```
Movable m3 = new MovableRectangle(1, 2, 3, 4, 25, 35);  // upcast
System.out.println(m3);
m3.moveUp();
System.out.println(m3);
```

Use lambda expressions wherever possible/applicable.