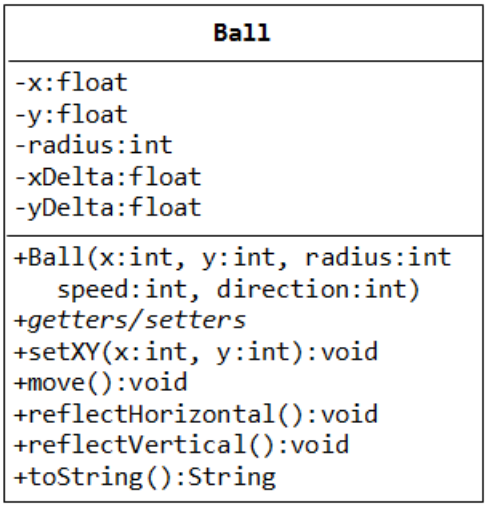
Lenguajes de Programación

Taller: Conceptos de Programación Orientada a Objetos

Abstracción:

Exercise #1: Bouncing Balls - Ball and Container Classes

A class called **`Ball`** is designed as shown in the class diagram.

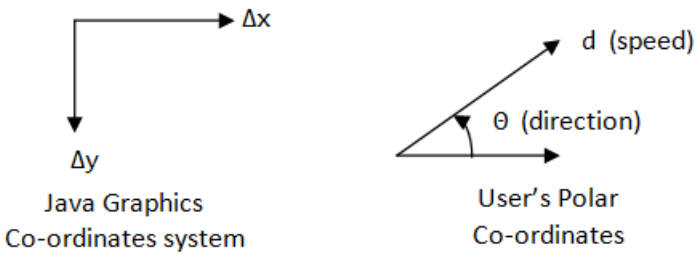


The `Ball` class containsthe following `private` instance variables:

- `x`, `y` and `radius`, which represent the ball's center `(x, y)` co-ordinates and the radius, respectively.
- `xDelta` (Δx) and `yDelta` (Δy), which represent the displacement (movement) per step, in the `x` and `y` direction respectively.

The `Ball` class containsthe following `public` methods:

- A constructor which accepts `x`, `y`, `radius`, `speed`, and `direction` as arguments. For user friendliness, user specifies `speed` (in pixels per step) and `direction` (in degrees in the range of `(-180°, 180°]`). For the internal operations, the `speed` and `direction` are to be converted to `(Δx, Δy)` in the internal representation. Note that the y-axis of the Java graphics coordinate system is inverted, i.e., the origin `(0, 0)` is located at the top-left corner.




```
Δx = d × cos(θ)
Δy = -d × sin(θ)
```

- **Getter** and **setter** for all the instance variables.
- A method `move()` which move the ball by one step.

  ```
  x += Δx
  y += Δy
  ```

- `reflectHorizontal()` which reflects the ball horizontally (i.e., hitting a vertical wall)

  ```
  Δx = -Δx
  Δy no changes
  ```

- `reflectVertical()` (the ball hits a horizontal wall).

  ```
  Δx no changes
  Δy = -Δy
  ```

- `toString()` which prints the message "`Ball at (x, y) of velocity (Δx, Δy)`".

Write the **`Ball`** class. Also write a test program to test all the methods defined in the class.

A class called **`Container`**, which represents the enclosing box for the ball, is designed as shown in the class diagram. It contains:

Container
-x1:int -y1:int -x2:int -y2:int
+Container(x:int,y:int,width:int,height:int) *+getters/setters* +collidesWith(ball:Ball):boolean +toString():String

- Instance variables `(x1, y1)` and `(x2, y2)` which denote the top-left and bottom-right corners of the rectangular box.
- A constructor which accepts `(x, y)` of the top-left corner, `width` and `height` as argument, and converts them into the internal representation (i.e., `x2=x1+width-1`). `Width` and `height` is used in the argument for safer operation (there is no need to check the validity of `x2>x1` etc.).
- A `toString()` method that returns "`Container at (x1,y1) to (x2, y2)`".
- A `boolean` method called `collidesWith(Ball)`, which check if the given `Ball` is outside the bounds of the container box. If so, it invokes the `Ball`'s `reflectHorizontal()` and/or `reflectVertical()` to change the movement direction of the ball, and returns `true`.

```
public boolean collidesWith(Ball ball) {
   if (ball.getX() - ball.getRadius() <= this.x1 ||
       ball.getX() - ball.getRadius() >= this.x2) {
      ball.reflectHorizontal();
      return true;
   }
   ......
}
```

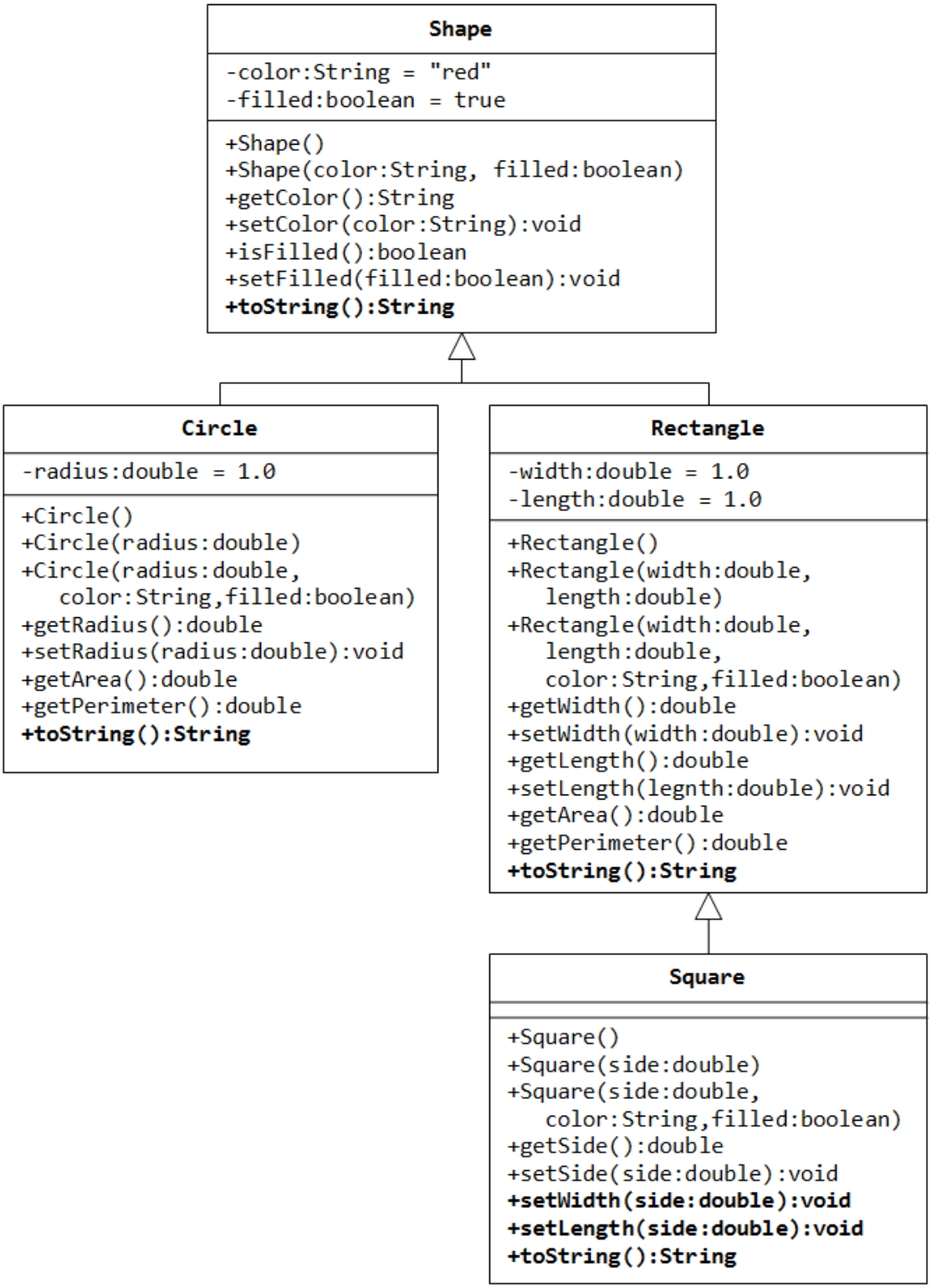
Use the following statements to test your program:

```
Ball ball = new Ball(50, 50, 5, 10, 30);
Container box = new Container(0, 0, 100, 100);

for (int step = 0; step < 100; ++step) {
   ball.move();
   box.collidesWith(ball);
   System.out.println(ball); // manual check the position of the ball
}
```

Herencia:

Exercise #2:



Write a superclass called Shape (as shown in the class diagram), which contains:

- Two instance variables color (String) and filled (boolean).
- Two constructors: a no-arg (no-argument) constructor that initializes the color to "green" and filled to true, and a constructor that initializes the color and filled to the given values.
- Getter and setter for all the instance variables. By convention, the getter for a boolean variable xxx is called isXXX() (instead of getXxx() for all the other types).
- A toString() method that returns "A Shape with color of xxx and filled/Not filled".

Write a test program to test all the methods defined in Shape.

Write two subclasses of Shape called Circle and Rectangle, as shown in the class diagram.

The Circle class contains:

- An instance variable radius (double).
- Three constructors as shown. The no-arg constructor initializes the radius to 1.0.
- Getter and setter for the instance variable radius.
- Methods getArea() and getPerimeter().
- Override the toString() method inherited, to return "A Circle with radius=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

The Rectangle class contains:

- Two instance variables width (double) and length (double).
- Three constructors as shown. The no-arg constructor initializes the width and length to 1.0.
- Getter and setter for all the instance variables.
- Methods getArea() and getPerimeter().
- Override the toString() method inherited, to return "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

Write a class called Square, as a subclass of Rectangle. Convince yourself that Square can be modeled as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in the class diagram). Hint:

  ```
  public Square(double side) {
  super(side, side);  // Call superclass Rectangle(double, double)
  }
  ```

- Override the toString() method to return "A Square with side=xxx, which is a subclass of

yyy", where yyy is the output of the toString() method from the superclass.

- Do you need to override the getArea() and getPerimeter()? Try them out.
- Override the setLength() and setWidth() to change both the width and length, so as to maintain the square geometry.

Polimorfismo:

Exercise #3:

Examine the following codes and draw the class diagram.

```
abstract public class Animal {
   abstract public void greeting();
}

public class Cat extends Animal {
   @Override
   public void greeting() {
      System.out.println("Meow!");
   }
}

public class Dog extends Animal {
   @Override
   public void greeting() {
      System.out.println("Woof!");
   }

   public void greeting(Dog another) {
      System.out.println("Wooooooooooof!");
   }
}

public class BigDog extends Dog {
   @Override
   public void greeting() {
      System.out.println("Woow!");
   }

   @Override
   public void greeting(Dog another) {
      System.out.println("Wooooooowwwww!");
   }
}
```

Explain the outputs (or error) for the following test program.

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

```
        // Using the subclasses
        Cat cat1 = new Cat();
        cat1.greeting();
        Dog dog1 = new Dog();
        dog1.greeting();
        BigDog bigDog1 = new BigDog();
        bigDog1.greeting();

        // Using Polymorphism
        Animal animal1 = new Cat();
        animal1.greeting();
        Animal animal2 = new Dog();
        animal2.greeting();
        Animal animal3 = new BigDog();
        animal3.greeting();
        Animal animal4 = new Animal();

        // Downcast
        Dog dog2 = (Dog)animal2;
        BigDog bigDog2 = (BigDog)animal3;
        Dog dog3 = (Dog)animal3;
        Cat cat2 = (Cat)animal2;
        dog2.greeting(dog3);
        dog3.greeting(dog2);
        dog2.greeting(bigDog2);
        bigDog2.greeting(dog2);
        bigDog2.greeting(bigDog1);
    }
}
```

Referencias:

http://www.ntu.edu.sg/home/ehchua/programming/
http://www.ntu.edu.sg/home/ehchua/programming/java/J3a_OOPBasics.html
http://www.ntu.edu.sg/home/ehchua/programming/cpp/cp6_Inheritance.html