Level	M1
Course	Java introduction
Subject	Lab Work 3
Duration	

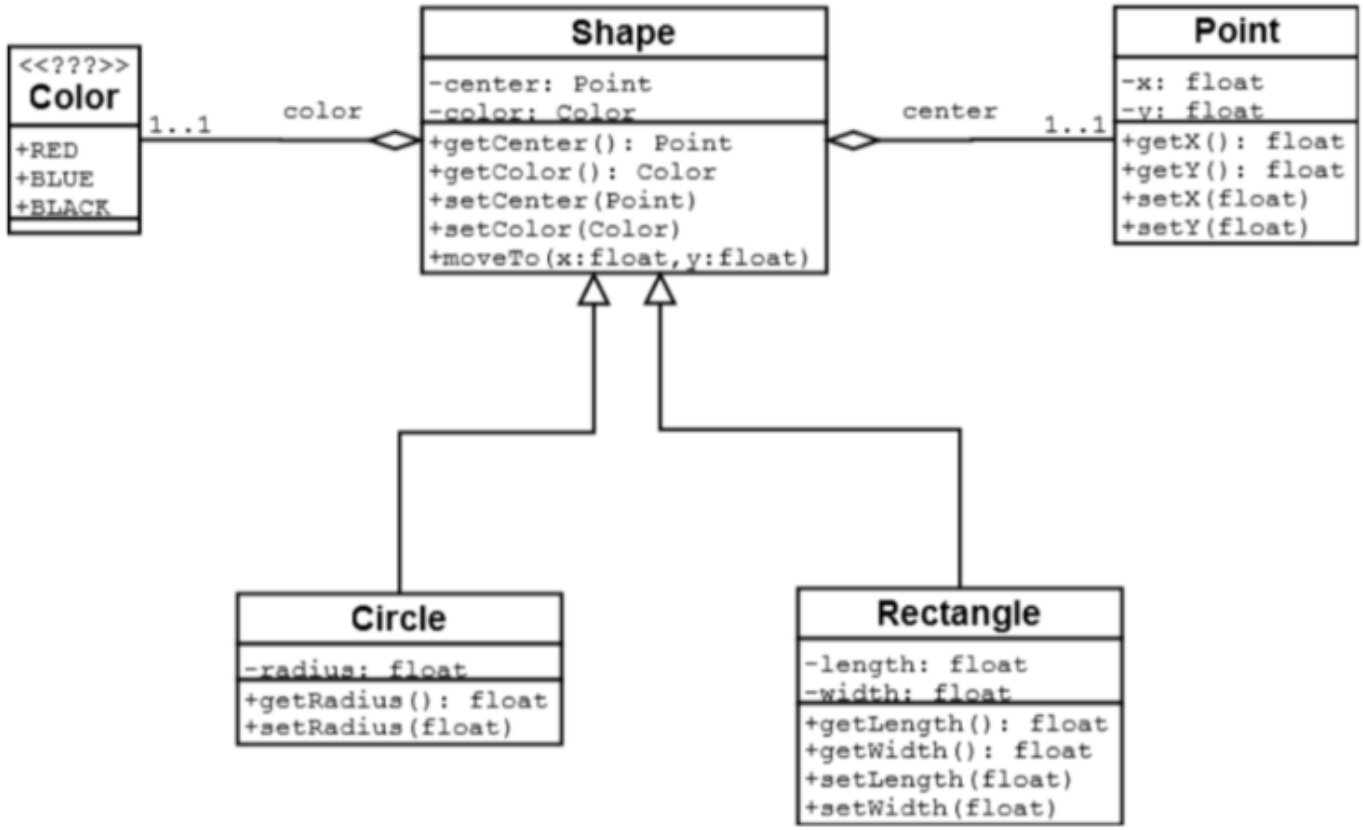
JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

***Objective**: Practice Inheritance in Java programming*

- *Create daughter class, Enumerations, Arrays*
- *Use some keywords knowing the reason*
- *Use of abstraction*

I.1: Beginning

Implement the class diagram below:



You need to identify:

- The required constructors to simplify your code
- Find out what can be the <<???>> Stereotype and implement it.
- The color definition is not mandatory
- The default color of a shape ifs BLACK

I.2: Overriding

In every class, redefine the behavior of the following methods inherited from the Object class:

- toString
- clone
- equals

A clone object has to be completely independent from the original object.

The different elements have to be represented on screen in the form:
For a point with coordinates (x, y):

```
"[x, y]"
```

For a circle of center (x, y), with a color c and a radius r

```
"Center: [x, y] - Color: c - Radius: r"
```

For a rectangle of center (x, y), with a color c, of length l and width w

```
"Center: [x, y] - Color: c - Length: l - Width: w"
```

- equals is required to compare the geometrical shapes characteristics, not their references.

I.3: Test

Create in a separate TestClass the following:

- Create a circle and a rectangle at two different places.
- Applied the following coding:

```
System.out.println(rectangle);
System.out.println(circle);

rectangle.setCenter( circle.getCenter() );
circle.moveTo(10,10);

System.out.println(rectangle);
System.out.println(circle);
```

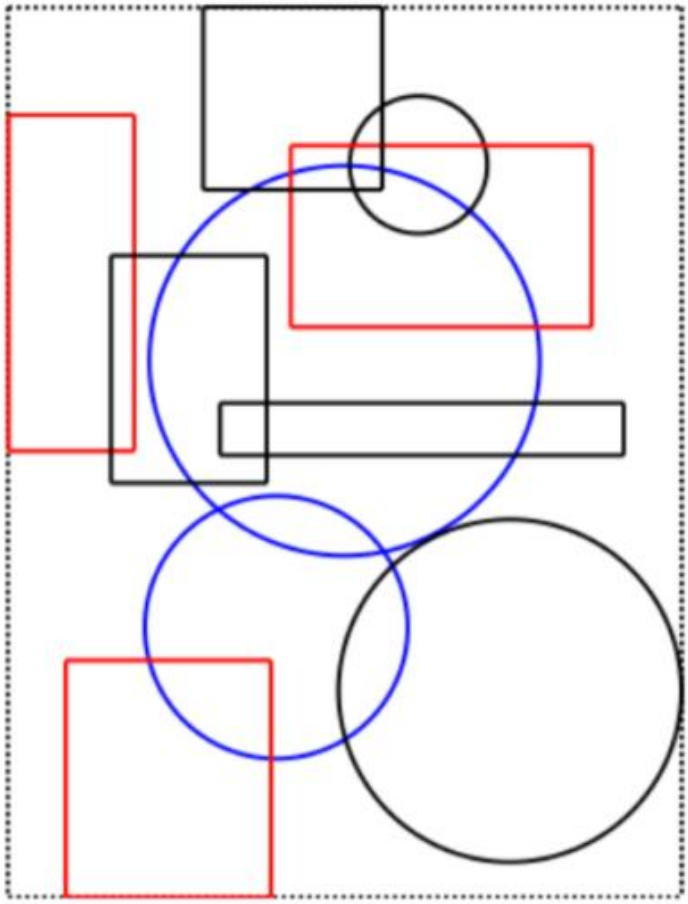
If you find something is going wrong during the execution, modify your code accordingly.

I.4: BoundingBox

Inside the Shape class, add the following method:

Public static Rectangle getBoundingBox(Shape [ ] shapes)

That calculates and returns the characteristics of the bounding box



The method getBoundingBox does not need to be changed if we add subclasses to Shape.

I.5: Dynamic printing

Find out a way to include dynamically (so not in a hard coded way) the class name in its string representation:

For a point with coordinates (x, y):

```
"Point(x, y)"
```

For a circle of center (x, y), with a color c and a radius r

```
"Circle(Center: Point(x, y) - Color: c - Radius: r)"
```

For a rectangle of center (x, y), with a color c, of length l and width w

```
"Rectangle(Center: Point(x, y) - Color: c - Length: l - Width: w)"
```

If we change the name of the classes, then everything is adapted without modifying the toString codes.