Part 1

The goal is to write functional class describing angle. The angle is defined so that it can have values from -pi to +pi. Adding/ subtracting two angles in a way that would result in a value from outside of this range should result in additional "wrapping" to the range [-pi, pi).

The “wrapping” should work as follows:
if value > pi then 2*pi should be subtracted from it
if value < -pi then 2*pi should be added to it
This procedure may need to be repeated i.e. if we create Angle(233.14) we would need to repeat the subtraction many times.
The implementation has to be tile-recursive.

The class needs to have **companion object with factory methods and useful contents**: Angle.halfPi, Angle.Pi, Angle.Zero.

Additional helper method to calculate angular distance in 2D Angle.DR which would take 4 angles like this:
Angle.DR(angle1, angle2)(angle3, alngle4)
and calculate angular distance (also an angle) dr = sqrt((angle1 - angle2) ^2 + (angle3 - angle4) ^2)

The **Angle class needs to have operations +/- with another instance of Angle**

e.g. val a = `Angle(0.7); val b = Angle(1.5); val c = a - b;`

and multiplication/division by a double. e.g.: val a = `Angle(0.7); val b = a*1.5;`

Part 2

The goal of the second exercise is to write a trait Color with a following abstract methods:

`setColor(r,g,b)`

`getColor()` returning tuple of three integers r,g,b

Based on that interface additional rich interface methods should be added:

```
increaseRedBy(redPercentage: float): Unit
increaseGreenBy(...)
increaseBlueBy(...)
decreaseRedBy(redPercentage: float)
decraaseGreenBy(...)
decreaseBlueBy(...)
```

The next step then would be to make Point class implementing the setColor and getColor.

It should be then possible to make objects of class Point _with and without_ the rich Color interface like this:

```
val p = new Point
p.setColor(56, 0, 120)
print(p)

// and
val pc = new Point with Color
pc.setColor(56, 0, 120)
p.increaseRedBy( 20 )
p.decreaseBlueBy( 10 )
println(pc)
```