Classes

Defining Classes

- Here we define a class called Point.
- It has a constructor and defines two properties.
- The `val` in the constructor is a shorthand to define the constructor and properties together.

```
class Point(val x: Int, val y: Int)
```

Methods

Inside this class:

```
fun distanceTo(p: Point): Double {
        val dx = x - p.x
        val dy = y - p.y
        return sqrt((dx * dx + dy * dy).toDouble())
    }
```

This is a method on Point that calculates the distance to another Point. Methods are functions operating on the data belonging to an object.

The `toString` Method

We can override the default `toString` behaviour to provide a customised way of displaying Points as Strings.

```
override fun toString(): String = "($x, $y)"
```

Enabling Destructuring

This implementation uses a string template to make concatenation neater.

- We can define `component1()` and `component2()`
- This enables the destructuring operator for our Point type, so we can say `val (x,y) = p`
- To be able to destructure your own types, you can define these up to `component6()`

```
operator fun component1(): Int = x
operator fun component2(): Int = y
```

Creating Instances

We can create an instance of our class by calling the constructor function.

```
val p1 = Point(3, 4)
val p2 = Point(5, 6)
```

Here we print the objects directly, which invokes `.toString()`

```
println(p1)
println(p2)
```