Lecture 3: Introduction to Object Oriented Programming with Kotlin 20-11-2023

Defining our own data types

Type aliases

- Consider the code:

```kotlin
fun distanceBetween(Pair<Int, Int>, Pair<Int, Int>) {
    val (x1, y1) = p1
    val (x2, y2) = p2

    val dx = x2 - x1
    val dy = y2 - y1

    return sqrt((dx * dx + dy * dy).toDouble())
}
```

- Consider if we want to introduce a different type to make what the two pairs are very clear (to make it clear that these pairs are points)
- One simple way of doing this is using a typealias
- We can do this using a type alias:

```kotlin
typealias Point = Pair<Int, Int>
```

- We can then change hte function as follows to now take Points instead of Pairs:

```
fun distanceBetween(p1: Point, p2: Point) {
    val (x1, y1) = p1
    val (x2, y2) = p2

    val dx = x2 - x1
    val dy = y2 - y1

    return sqrt((dx * dx + dy * dy).toDouble())
}
```

Creating classes

- Say we want to create a circle with an x and y coordinate for the center and a radius

```
class Circle(x: Int, y: Int, radius: Int) {
    val centreX = x
    val centreY = y
    val radius = radius
    // The value on the left is the property name and the value on the
right is the constructor
}

fun main() {
    // The following is an object that has been created in the memory of
the program
    val c1 = Circle(3, 4, 2)
    println("Circle 1 has a radius of " + c1.radius)
    val c2 = Circle(6, 6, 4)
}
```

- WE can now define functions within circle:

```
import kotlin.math.PI

class Circle(x: Int, y: Int, radius: Int) {
    val centreX = x
    val centreY = y
    val radius = radius
    fun area() = PI * radius * radius
    fun circumference() = 2 * PI * radius
}
```

- If you want to be able to destructure the items of a class like `val (x1, y1) = p1`, you will need to define a component 1 function:

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