Lecture 4: Introduction to Object Oriented Programming with Kotlin 21-11-2023

Lists of objects

- We can create a list of objects below

```
import lecture3.Circle
import lecture3.Point

class Radar {

    val points: List<Point> = listOF(Point(4,2), Point(2,7), Point(5,5),
Point(3,5), Point(-1,3))

    val fieldOfView = Circle(Point(3,4), 3)
}

fun main() {
    val radar = Radar()
    println(radar)
}
```

- Here when we print content, it prints the memory address of the object
- Hence we should implement a toString

```
import lecture3.Circle
import lecture3.Point

class Radar {

    val points: List<Point> = listOF(Point(4,2), Point(2,7), Point(5,5),
Point(3,5), Point(-1,3))

    val fieldOfView = Circle(Point(3,4), 3)
    override fun toString(): String = points.filter {p ->
p.fieldOfView.includes(p)}.map {p -> p.toString()}.joinToString(" :: ")
}

fun main() {
    val radar = Radar()
    println(radar)
}
```

- This will produce the output

```
(4, 2) :: (5, 5) :: (3, 5)
```

- We can clean this up using `it`

```
override fun toString(): String = points.filter
{fieldOfView.includes(it)}.joinToString(" :: ") {it.toString()}
```

- We can also achieve this using imperative programming techniques

```
override fun toString(): String {
val sb = mutableListOf<String>()
for (p in points) {
    if (fieldOfView.includes(p)) {
        sb.add(p.toString())
    }
}
return sb.joinToString("  ::  ")
}
```

Sets

```
val set = setOf("quick", "quick", "slow")

set.forEach(::println)
```

```
// Double colon allows for you to treat a function like a variable and
"send" it places
```