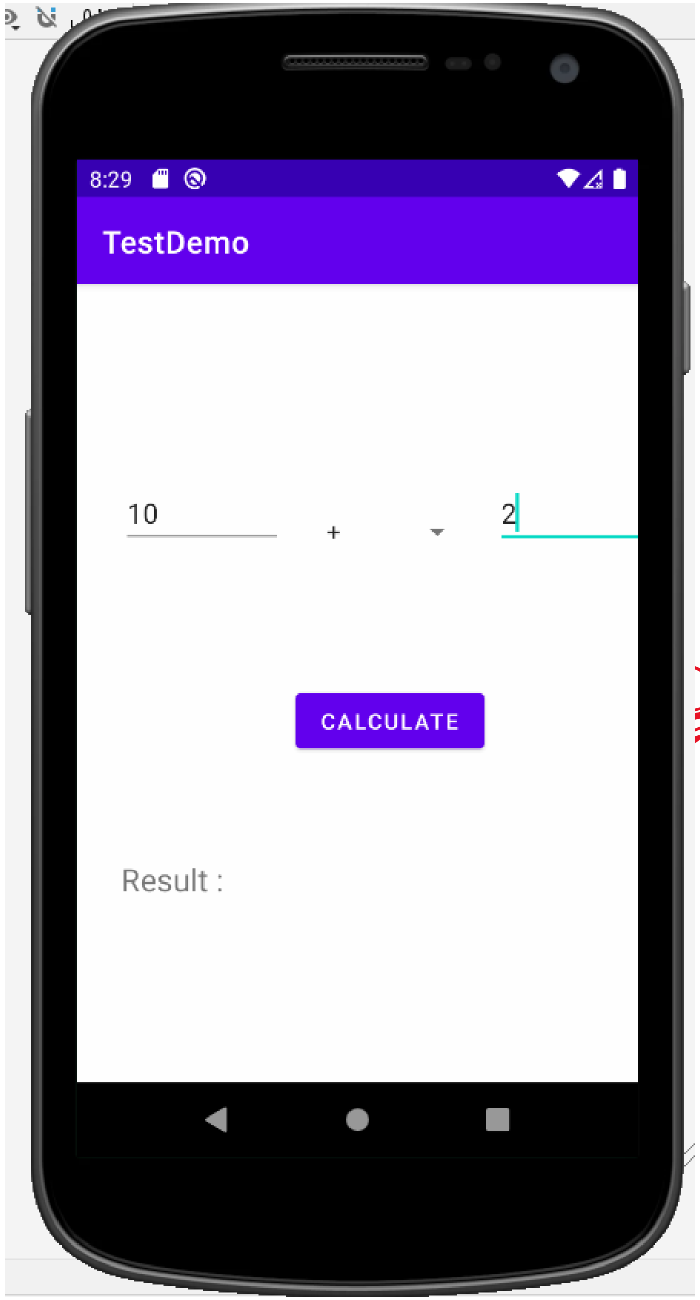
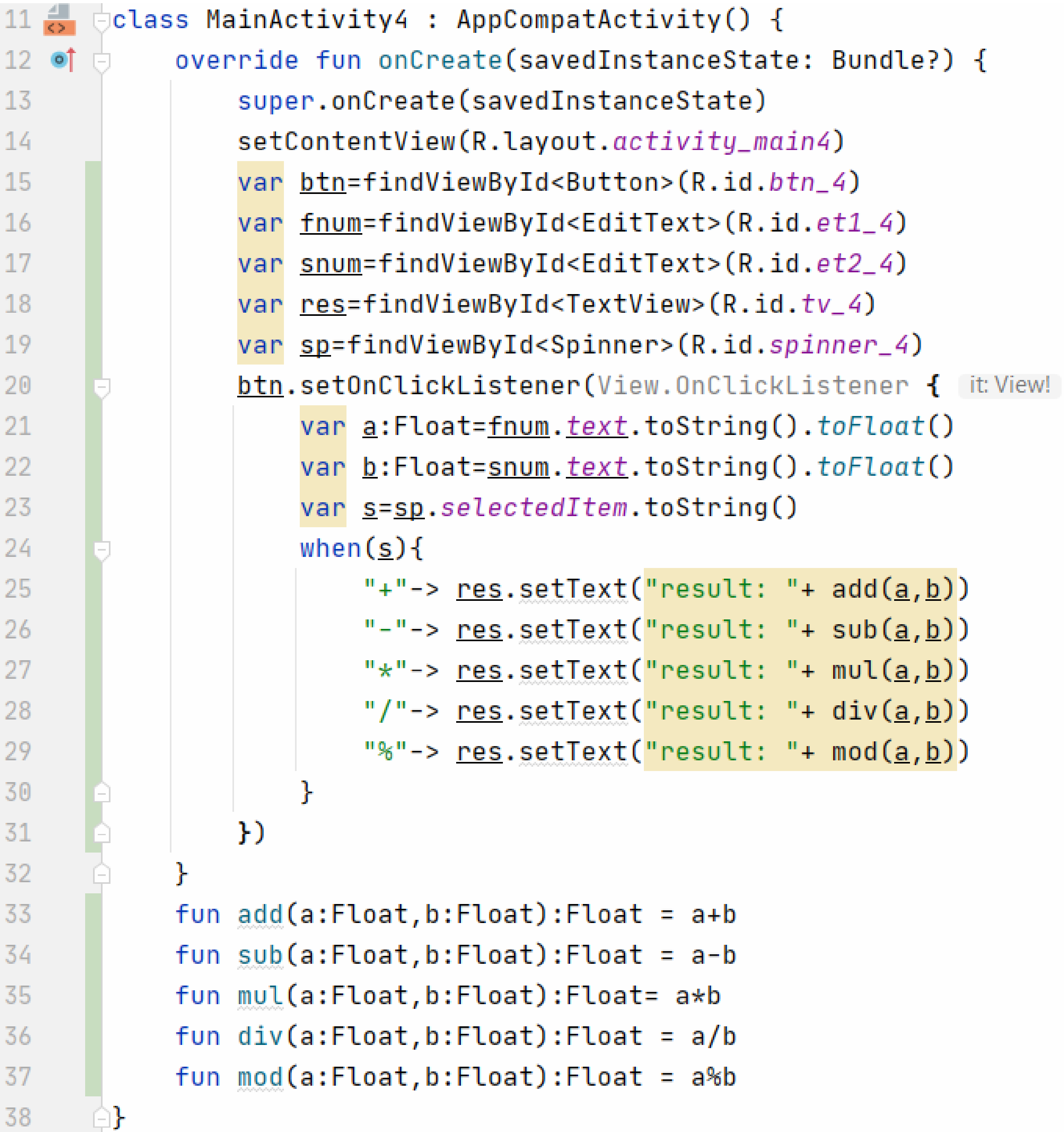
Week 2 : Session 1

More about Function, Classes and Object

Exercise





```kotlin
class MainActivity4 : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main4)
        var btn=findViewById<Button>(R.id.btn_4)
        var fnum=findViewById<EditText>(R.id.et1_4)
        var snum=findViewById<EditText>(R.id.et2_4)
        var res=findViewById<TextView>(R.id.tv_4)
        var sp=findViewById<Spinner>(R.id.spinner_4)
        btn.setOnClickListener(View.OnClickListener {
            var a:Float=fnum.text.toString().toFloat()
            var b:Float=snum.text.toString().toFloat()
            var s=sp.selectedItem.toString()
            when(s){
                "+"-> res.setText("result: "+ add(a,b))
                "-"-> res.setText("result: "+ sub(a,b))
                "*"-> res.setText("result: "+ mul(a,b))
                "/"-> res.setText("result: "+ div(a,b))
                "%"-> res.setText("result: "+ mod(a,b))
            }
        })
    }
    fun add(a:Float,b:Float):Float = a+b
    fun sub(a:Float,b:Float):Float = a-b
    fun mul(a:Float,b:Float):Float= a*b
    fun div(a:Float,b:Float):Float = a/b
    fun mod(a:Float,b:Float):Float = a%b
}
```

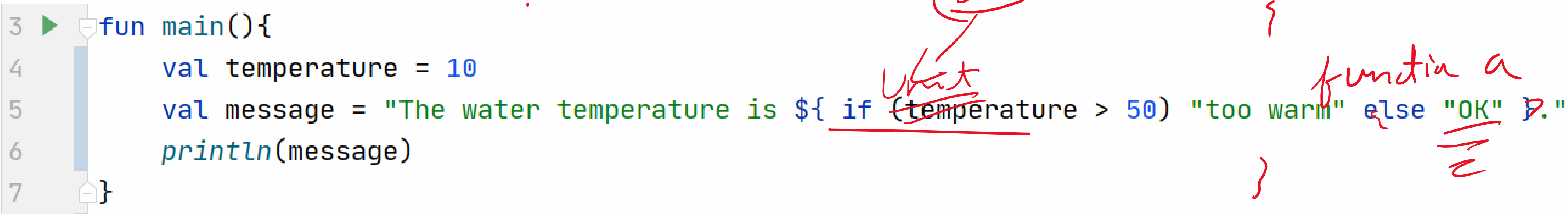
Agenda

- More about Functions
  - Learn why (almost) everything has a value
  - Learn more about functions
  - Explore default values and compact functions
- Object-oriented Programming (OOP)
- Class and Objects
- Exercise 5

Q

Exer

More about Functions



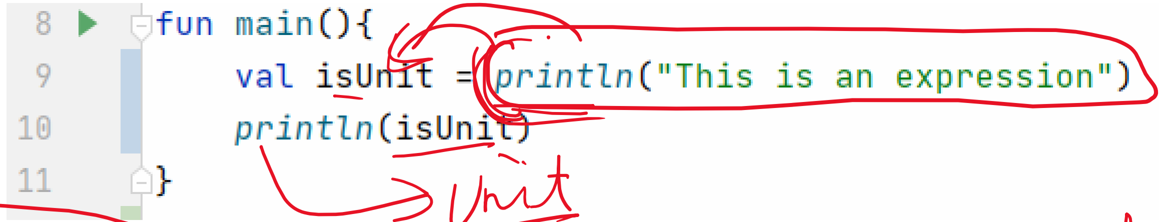
```
fun main(){
    val temperature = 10
    val message = "The water temperature is ${ if (temperature > 50) "too warm" else "OK" }."
    println(message)
}
```

a

Unit

class
{
function a
}

- In Kotlin, almost everything is an *expression* and has a value.



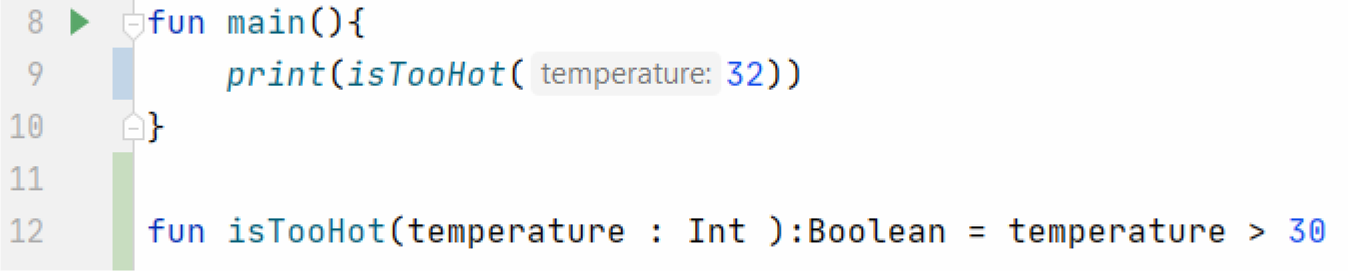
```
fun main(){
    val isUnit = println("This is an expression")
    println(isUnit)
}
```

Unit

class
{
function println(
{
// logic
}
}

Kotlin?
Unit

- Compact functions :
  - also called *single-expression functions*
  - When a function returns the results of a single expression, you can specify the body of the function after an = symbol, omit the curly braces {}, and omit the return



```
fun main(){
    print(isTooHot( temperature: 32))
}

fun isTooHot(temperature : Int ):Boolean = temperature > 30
```

More about Functions

Example : Create a program in kotlin to help Bob, what food should he feed to fishes in the aquarium on particular day and does he need to change the water

To change the water optimal

temperature > 30

dirt sensor reading > 30

if day is Sunday

Display random single day's description.

Monday -> flakes

Tuesday -> pellets

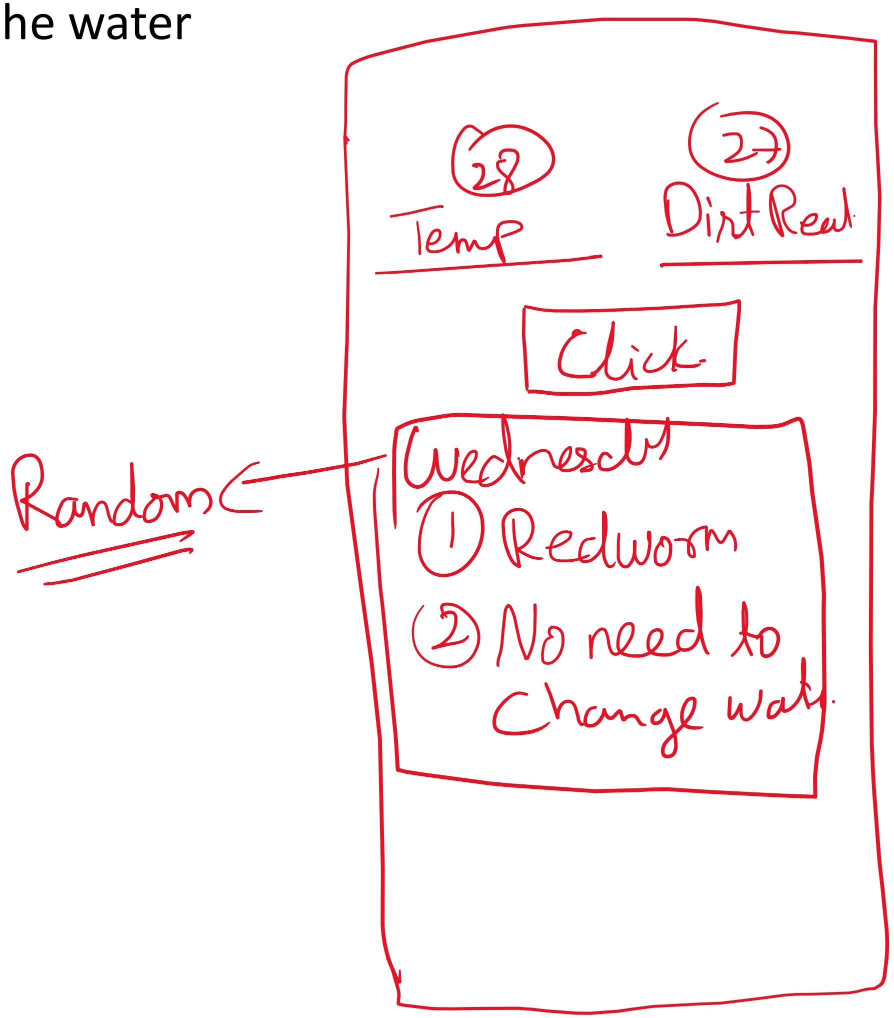
Wednesday -> redworms

Thursday -> granules

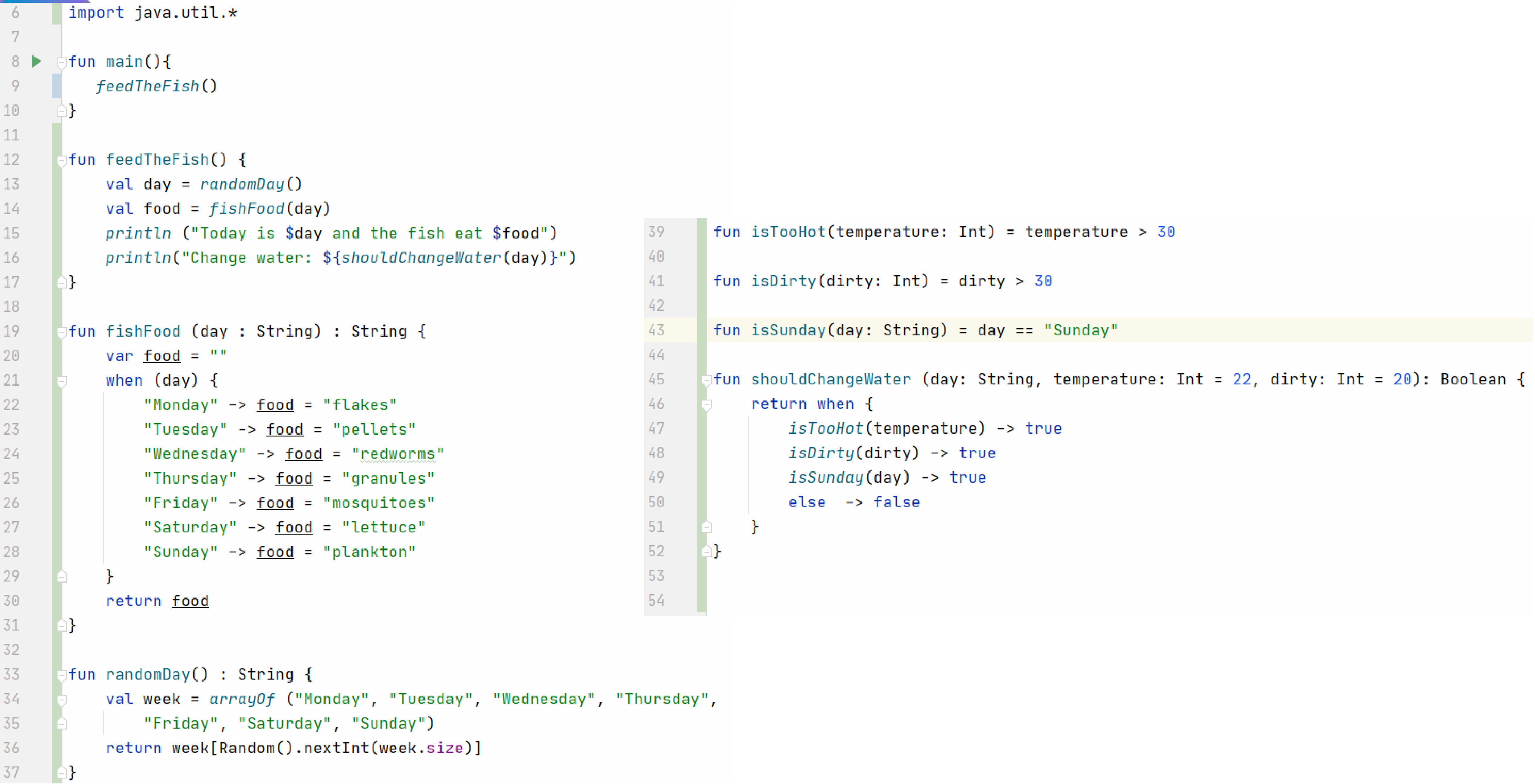
Friday -> mosquitoes

Saturday -> lettuce

Sunday -> plankton



Solution

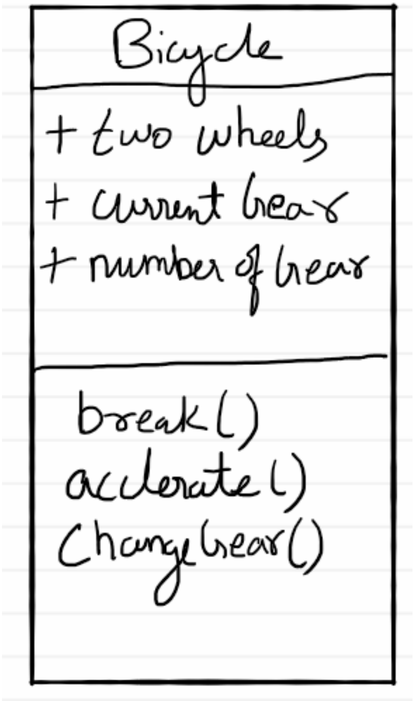


```kotlin
import java.util.*

fun main(){
    feedTheFish()
}

fun feedTheFish() {
    val day = randomDay()
    val food = fishFood(day)
    println ("Today is $day and the fish eat $food")
    println("Change water: ${shouldChangeWater(day)}")
}

fun fishFood (day : String) : String {
    var food = ""
    when (day) {
        "Monday" -> food = "flakes"
        "Tuesday" -> food = "pellets"
        "Wednesday" -> food = "redworms"
        "Thursday" -> food = "granules"
        "Friday" -> food = "mosquitoes"
        "Saturday" -> food = "lettuce"
        "Sunday" -> food = "plankton"
    }
    return food
}

fun randomDay() : String {
    val week = arrayOf ("Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday", "Sunday")
    return week[Random().nextInt(week.size)]
}
```

```kotlin
fun isTooHot(temperature: Int) = temperature > 30

fun isDirty(dirty: Int) = dirty > 30

fun isSunday(day: String) = day == "Sunday"

fun shouldChangeWater (day: String, temperature: Int = 22, dirty: Int = 20): Boolean {
    return when {
        isTooHot(temperature) -> true
        isDirty(dirty) -> true
        isSunday(day) -> true
        else  -> false
    }
}
```

Object-oriented Programming (OOP)

- Divide a complex problem into smaller sets by creating objects.
- These objects share two characteristics:
  - state
  - behavior
  - E.g
    - Bicycle is an object
    - It has current gear, two wheels, number of gear etc. states.
    - It has braking, accelerating, changing gears etc. behavior.
- Features of an object-oriented programming
  - *Data encapsulation*
  - *Inheritance*
  - *Polymorphism*



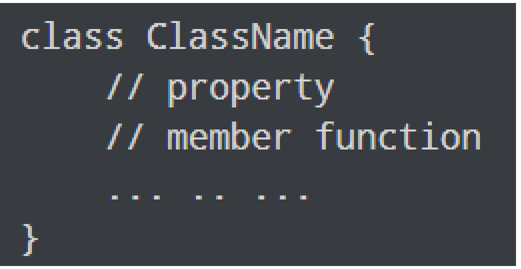
Bicycle

+ two wheels
+ current Gear
+ number of Gear

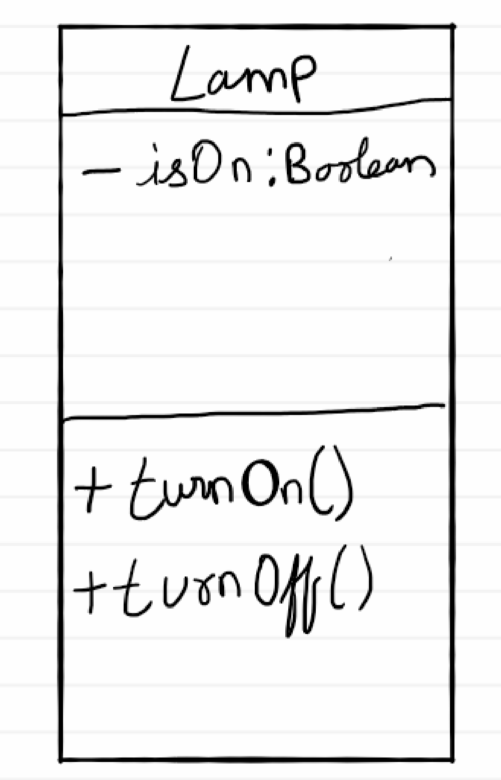
break()
acclerate()
ChangeGear()

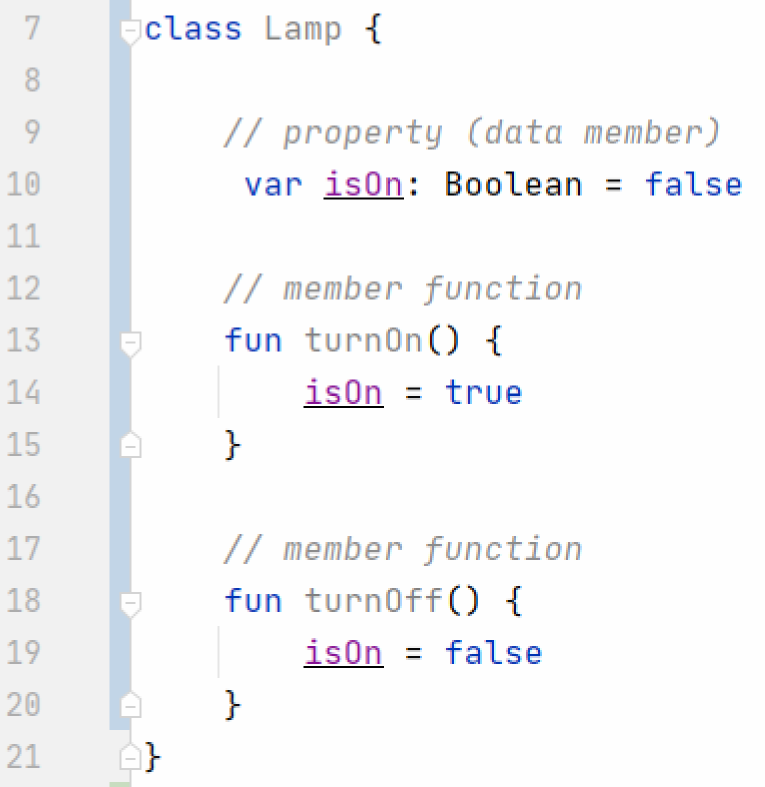
Kotlin Class

- A class is a blueprint for the object(sketch (prototype)).

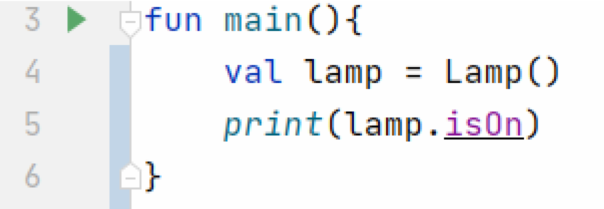


```
class ClassName {
    // property
    // member function
    ... .. ...
}
```





```
class Lamp {

    // property (data member)
     var isOn: Boolean = false

    // member function
    fun turnOn() {
        isOn = true
    }

    // member function
    fun turnOff() {
        isOn = false
    }
}
```



```
fun main(){
    val lamp = Lamp()
    print(lamp.isOn)
}
```