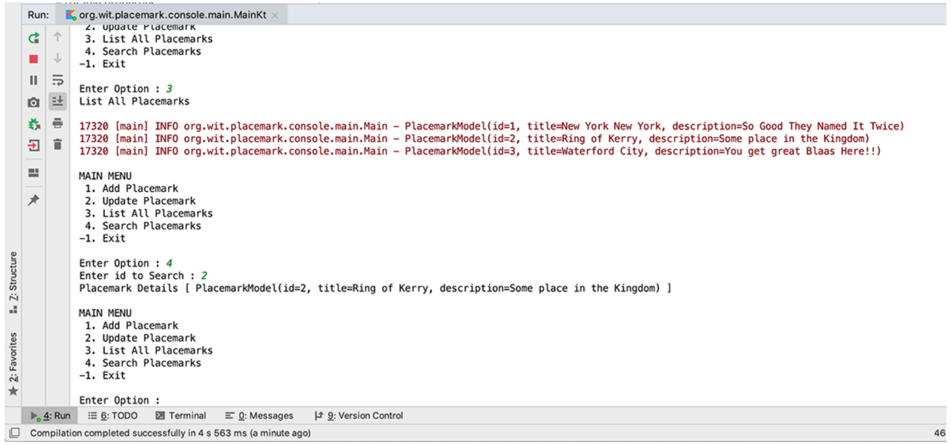
Mobile Application Development

Produced by David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Placemark-Console Version 3.0



```
 2. Update Placemark
 3. List All Placemarks
 4. Search Placemarks
-1. Exit

Enter Option : 3
List All Placemarks

17320 [main] INFO org.wit.placemark.console.main.Main - PlacemarkModel(id=1, title=New York New York, description=So Good They Named It Twice)
17320 [main] INFO org.wit.placemark.console.main.Main - PlacemarkModel(id=2, title=Ring of Kerry, description=Some place in the Kingdom)
17320 [main] INFO org.wit.placemark.console.main.Main - PlacemarkModel(id=3, title=Waterford City, description=You get great Blaas Here!!)

MAIN MENU
 1. Add Placemark
 2. Update Placemark
 3. List All Placemarks
 4. Search Placemarks
-1. Exit

Enter Option : 4
Enter id to Search : 2
Placemark Details [ PlacemarkModel(id=2, title=Ring of Kerry, description=Some place in the Kingdom) ]

MAIN MENU
 1. Add Placemark
 2. Update Placemark
 3. List All Placemarks
 4. Search Placemarks
-1. Exit

Enter Option :
```

```
Compilation completed successfully in 4 s 563 ms (a minute ago)                46
```

Features Covered (from Part 1)

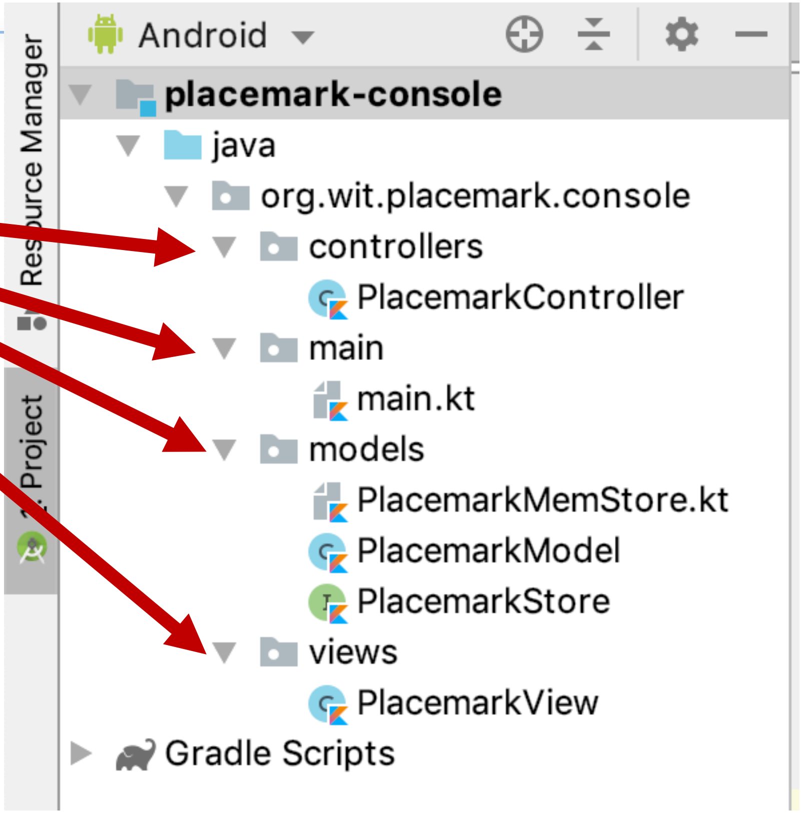
- ❑ Basic Types
- ❑ Local Variables (val & var)
- ❑ Functions
- ❑ Control Flow (if, when, for, while)
- ❑ Strings & String Templates
- ❑ Ranges (and the *in* operator)
- ❑ Type Checks & Casts
- ❑ Null Safety
- ❑ Comments

Features Covered (from Part 2)

- ❑Writing Classes (properties and fields)
- ❑Data Classes (just for data)
- ❑Collections: Arrays and Collections
- ❑Collections: *in* operator and lambdas
- ❑Arguments (default and named)

Project Structure

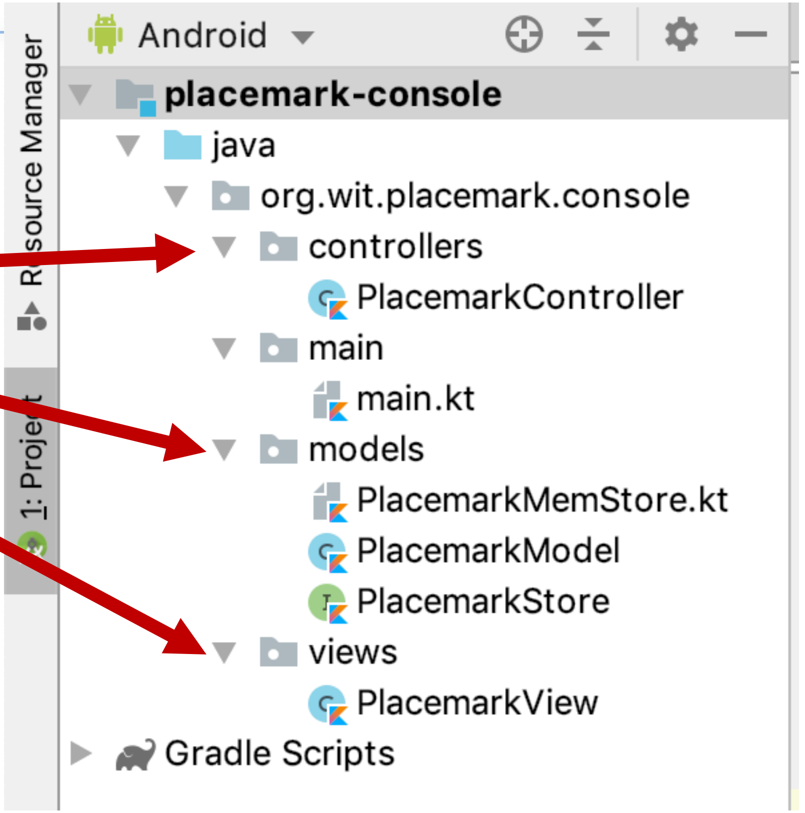
- Now more complex
  - Multiple packages



```
placemark-console
  java
    org.wit.placemark.console
      controllers
        PlacemarkController
      main
        main.kt
      models
        PlacemarkMemStore.kt
        PlacemarkModel
        PlacemarkStore
      views
        PlacemarkView
Gradle Scripts
```

Project Structure

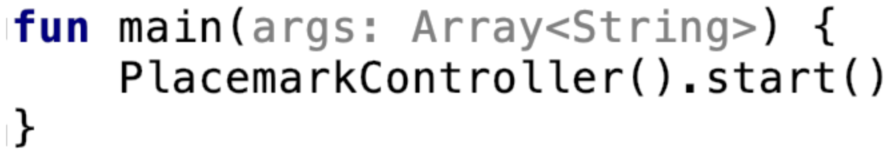
- Now more complex
  - Multiple packages
  - MVC Pattern



```
placemark-console
  java
    org.wit.placemark.console
      controllers
        PlacemarkController
      main
        main.kt
      models
        PlacemarkMemStore.kt
        PlacemarkModel
        PlacemarkStore
      views
        PlacemarkView
Gradle Scripts
```

main.kt

- Codebase in main file substantially Reduced
  - **fun main**
  - Single line of code


```
fun main(args: Array<String>) {
    PlacemarkController().start()
}
```

Classes & Interfaces

Placemark-Console Version 3.0

Classes & Interfaces in Placemark

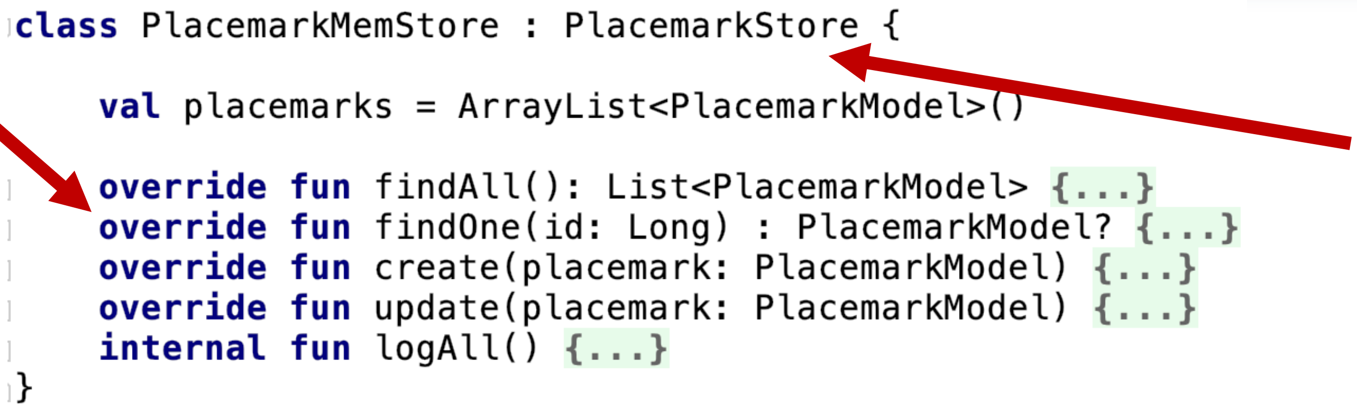
- ❑ Version 3.0 refactors the app to make use of the MVC (Model-View-Controller) Design Pattern
- ❑ Allows for SoC (Separation of Concerns)
- ❑ A number of new Classes & Interfaces introduced to improve overall design

Interface **`PlacemarkStore`**

```
interface PlacemarkStore {
    fun findAll(): List<PlacemarkModel>
    fun findOne(id: Long): PlacemarkModel?
    fun create(placemark: PlacemarkModel)
    fun update(placemark: PlacemarkModel)
}
```

- 4 functions, responsible for defining how our implementation classes will behave (for the moment)
- Classes that implement this interface <u>must</u> implement these functions

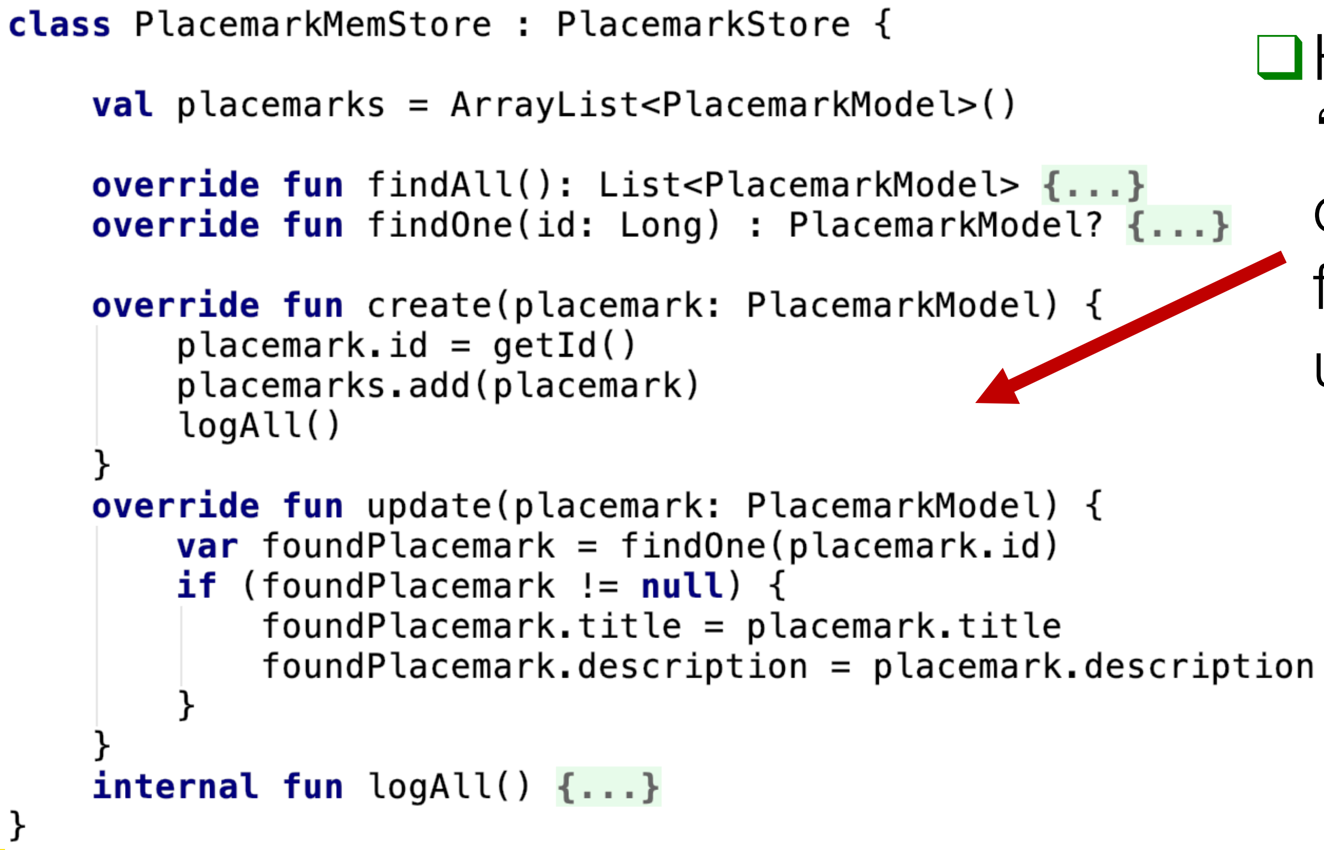
Class **PlacemarkMemStore**



```
class PlacemarkMemStore : PlacemarkStore {

    val placemarks = ArrayList<PlacemarkModel>()

    override fun findAll(): List<PlacemarkModel> {...}
    override fun findOne(id: Long) : PlacemarkModel? {...}
    override fun create(placemark: PlacemarkModel) {...}
    override fun update(placemark: PlacemarkModel) {...}
    internal fun logAll() {...}
}
```

- This class implements **`PlacemarkStore`** – note modifier **`override`** to signify the function is an implementation of the function defined in interface
- We also have an **`internal`** function (just 'local' to this class)

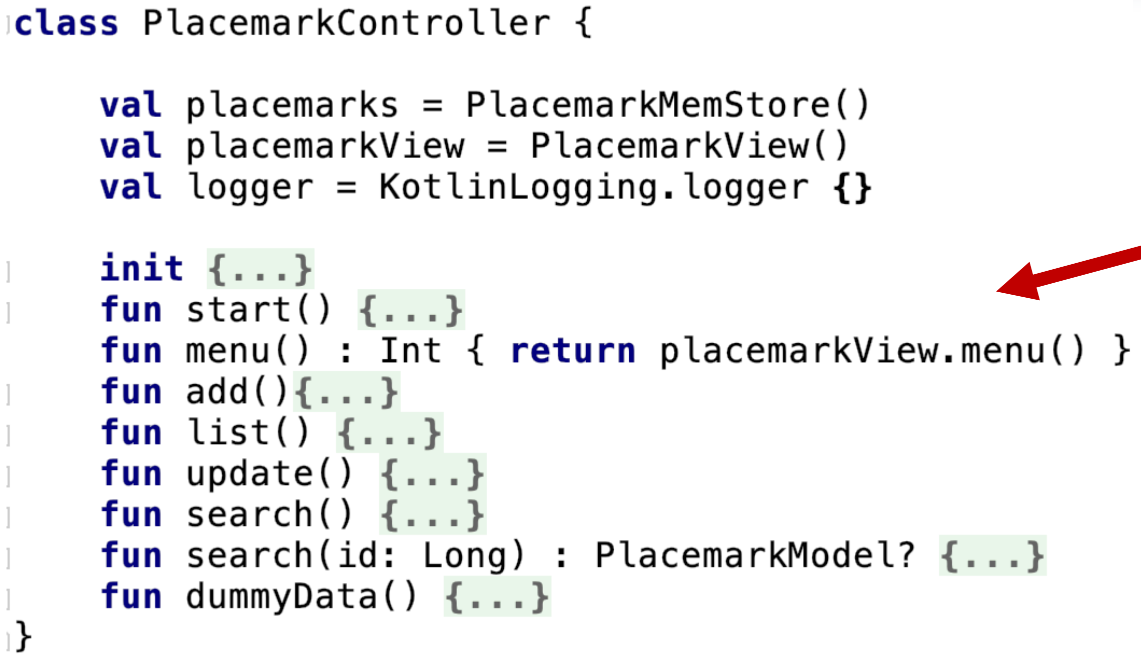
Class **`PlacemarkMemStore`**



```
class PlacemarkMemStore : PlacemarkStore {

    val placemarks = ArrayList<PlacemarkModel>()

    override fun findAll(): List<PlacemarkModel> {...}
    override fun findOne(id: Long) : PlacemarkModel? {...}

    override fun create(placemark: PlacemarkModel) {
        placemark.id = getId()
        placemarks.add(placemark)
        logAll()
    }
    override fun update(placemark: PlacemarkModel) {
        var foundPlacemark = findOne(placemark.id)
        if (foundPlacemark != null) {
            foundPlacemark.title = placemark.title
            foundPlacemark.description = placemark.description
        }
    }
    internal fun logAll() {...}
}
```

- ❑ Here we see the ‘inner workings’ of our implemented functions (create & update)

Class **PlacemarkController**



```
class PlacemarkController {

    val placemarks = PlacemarkMemStore()
    val placemarkView = PlacemarkView()
    val logger = KotlinLogging.logger {}

    init {...}
    fun start() {...}
    fun menu() : Int { return placemarkView.menu() }
    fun add(){...}
    fun list() {...}
    fun update() {...}
    fun search() {...}
    fun search(id: Long) : PlacemarkModel? {...}
    fun dummyData() {...}
}
```

- Main purpose to act as 'link' between **Model** and **View**
- No interaction with user - 'controls' the flow of data between the user and storage

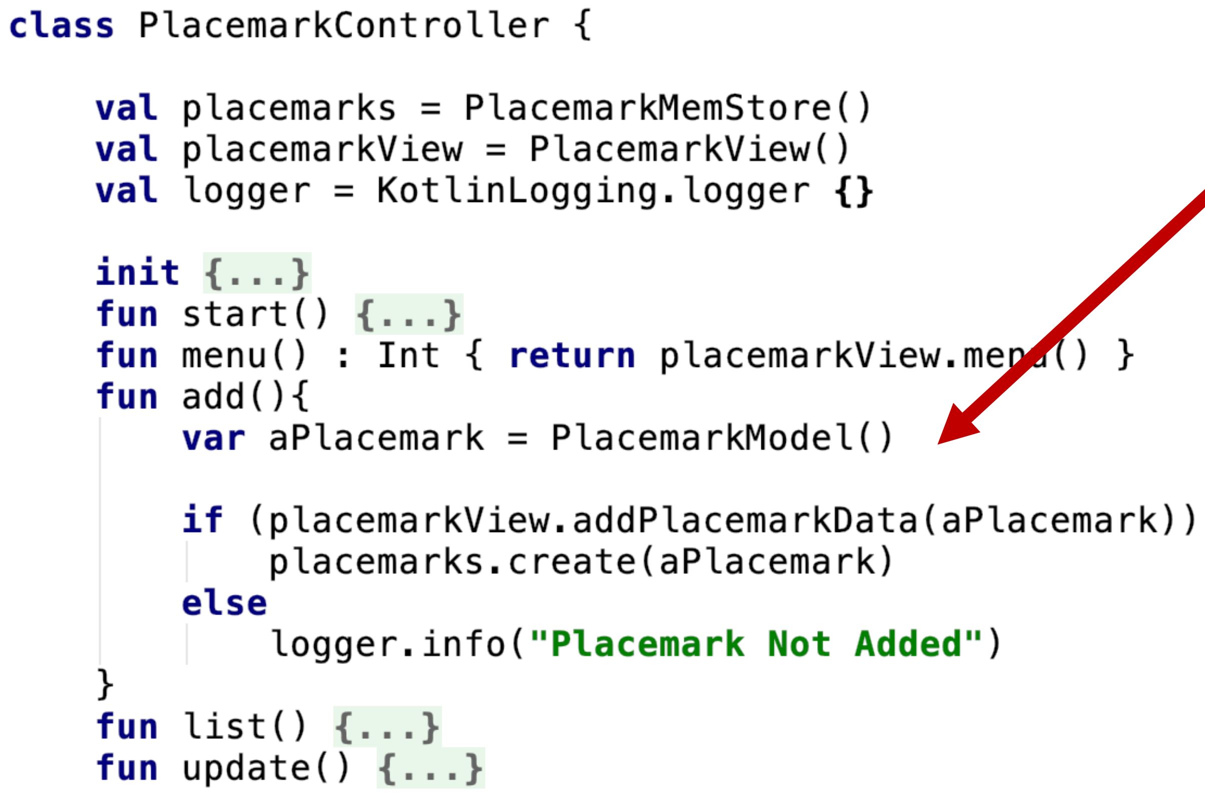
Class `PlacemarkController - start()`

```
class PlacemarkController {

    val placemarks = PlacemarkMemStore()
    val placemarkView = PlacemarkView()
    val logger = KotlinLogging.logger {}

    init {...}
    fun start() {
        var input: Int

        do {
            input = menu()
            when (input) {
                1 -> add()
                2 -> update()
                3 -> list()
                4 -> search()
                -99 -> dummyData()
                -1 -> println("Exiting App")
                else -> println("Invalid Option")
            }
            println()
```

```
fun main(args: Array<String>) {
    PlacemarkController().start()
}
```

- Initial point of entry in application
- Triggers CRUD features based on user options from ‘View’

Class **`PlacemarkController - add()`**

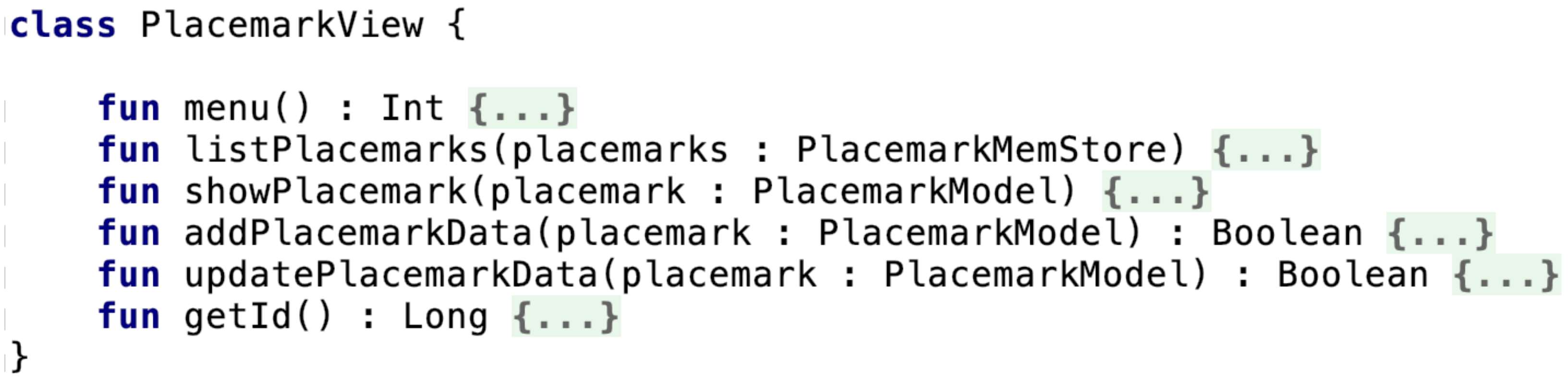


```
class PlacemarkController {

    val placemarks = PlacemarkMemStore()
    val placemarkView = PlacemarkView()
    val logger = KotlinLogging.logger {}

    init {...}
    fun start() {...}
    fun menu() : Int { return placemarkView.menu() }
    fun add(){
        var aPlacemark = PlacemarkModel()

        if (placemarkView.addPlacemarkData(aPlacemark))
            placemarks.create(aPlacemark)
        else
            logger.info("Placemark Not Added")
    }
    fun list() {...}
    fun update() {...}
```

- ❑ Interacting with both **Model** and **View** to add a Placemark

- ❑ The remaining functions operate in a similar fashion

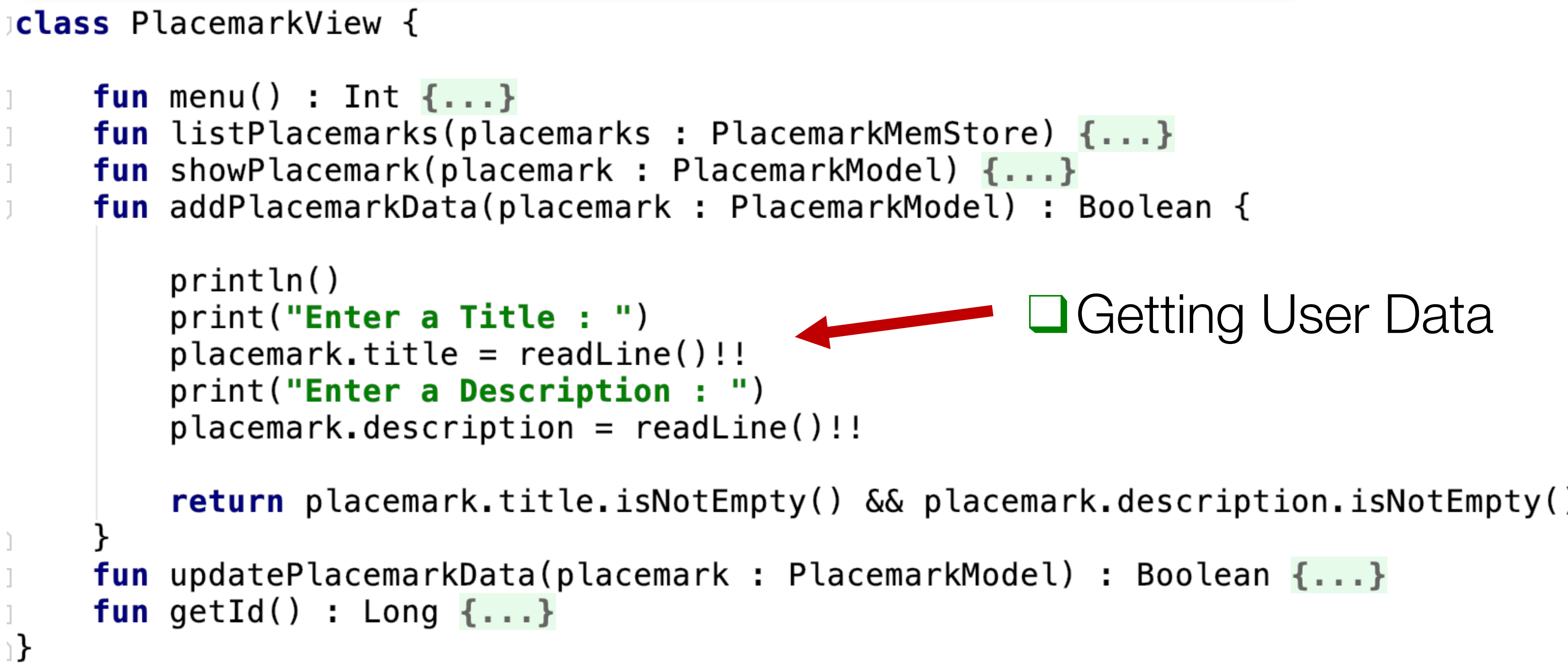
Class **`PlacemarkView`**



```kotlin
class PlacemarkView {

    fun menu() : Int {...}
    fun listPlacemarks(placemarks : PlacemarkMemStore) {...}
    fun showPlacemark(placemark : PlacemarkModel) {...}
    fun addPlacemarkData(placemark : PlacemarkModel) : Boolean {...}
    fun updatePlacemarkData(placemark : PlacemarkModel) : Boolean {...}
    fun getId() : Long {...}
}
```

- ‘User Facing’ class to present console UI for the application
- Responsible for displaying user menu, Placemarks, getting Placemark data etc.

Class **`PlacemarkView - addPlacemarkData()`**



```
class PlacemarkView {

    fun menu() : Int {...}
    fun listPlacemarks(placemarks : PlacemarkMemStore) {...}
    fun showPlacemark(placemark : PlacemarkModel) {...}
    fun addPlacemarkData(placemark : PlacemarkModel) : Boolean {

        println()
        print("Enter a Title : ")
        placemark.title = readLine()!!
        print("Enter a Description : ")
        placemark.description = readLine()!!

        return placemark.title.isNotEmpty() && placemark.description.isNotEmpty(
    }
    fun updatePlacemarkData(placemark : PlacemarkModel) : Boolean {...}
    fun getId() : Long {...}
}
```

- ❑ Getting User Data

Arrays & Collections

Placemark-Console Version 3.0

Arrays & Collections in Placemark

- **val** variable **placemarks** is declared as a **ArrayList** of **PlacemarkModel** objects inside **PlacemarkMemStore**

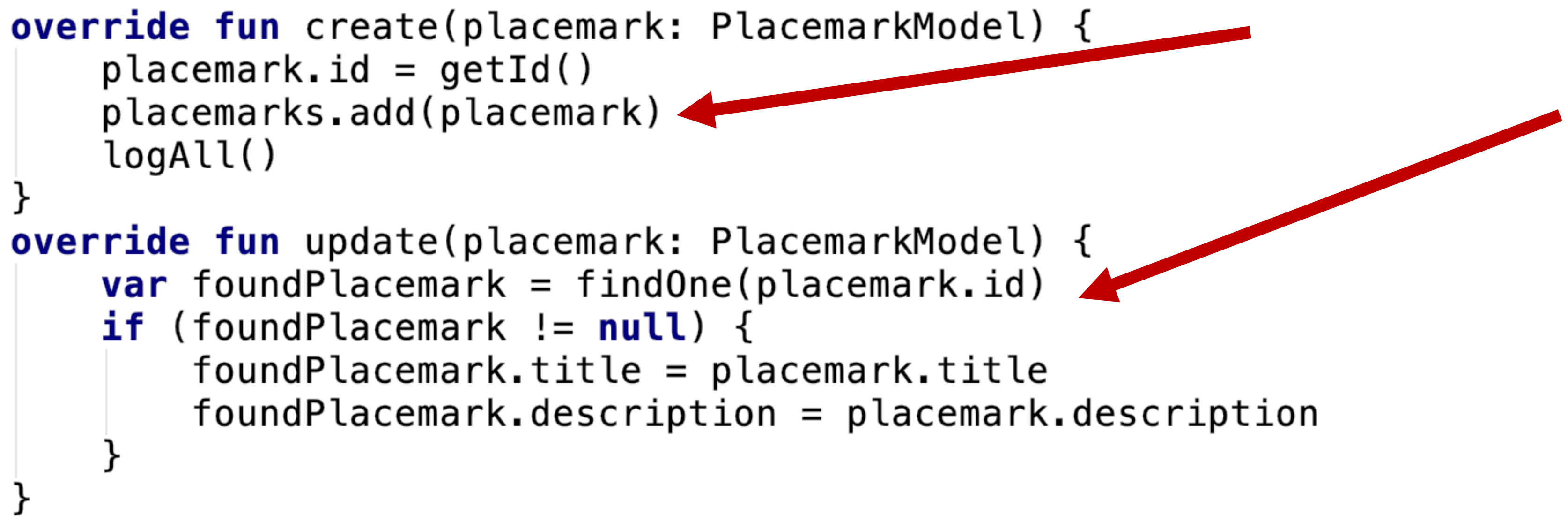
```
val placemarks = ArrayList<PlacemarkModel>()
```

- we can add to this collection manually like so

```
placemarks.add(PlacemarkModel(1, "New York New York", "So Good They Named It Twice"))
```

Arrays & Collections in Placemark

- We can also carry out CRUD operations using **`PlacemarkMemStore`** functions



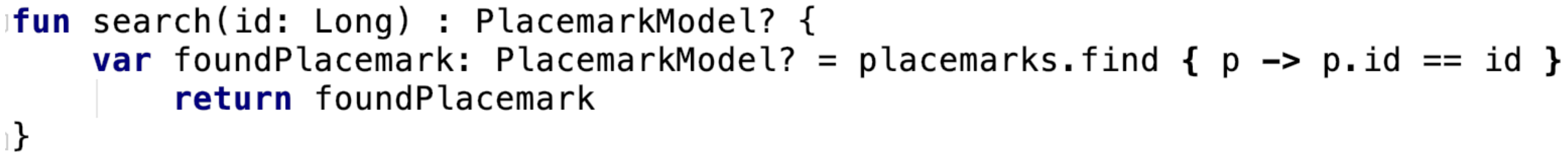
```
override fun create(placemark: PlacemarkModel) {
    placemark.id = getId()
    placemarks.add(placemark)
    logAll()
}
override fun update(placemark: PlacemarkModel) {
    var foundPlacemark = findOne(placemark.id)
    if (foundPlacemark != null) {
        foundPlacemark.title = placemark.title
        foundPlacemark.description = placemark.description
    }
}
```

Lambdas

Placemark-Console Version 3.0

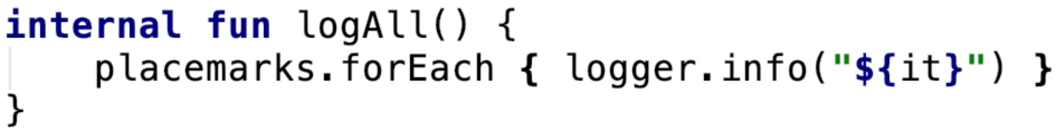
Lambdas in Placemark

- Here the Collections **`find`** function is a ‘Higher Order’ function which means we can assign it as a parameter and pass anonymous functions to it, and use lambda expressions



```
fun search(id: Long) : PlacemarkModel? {
    var foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.id == id }
        return foundPlacemark
}
```

- Another Higher Order Function – **`forEach`** (note use of **`it`**)



```
internal fun logAll() {
    placemarks.forEach { logger.info("${it}") }
}
```

References

Sources:

http://kotlinlang.org/docs/reference/basic-syntax.html
http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/
https://www.programiz.com/kotlin-programming
https://medium.com/@napperley/kotlin-tutorial-5-basic-collections-3f114996692b

Thanks.