Mobile App Development

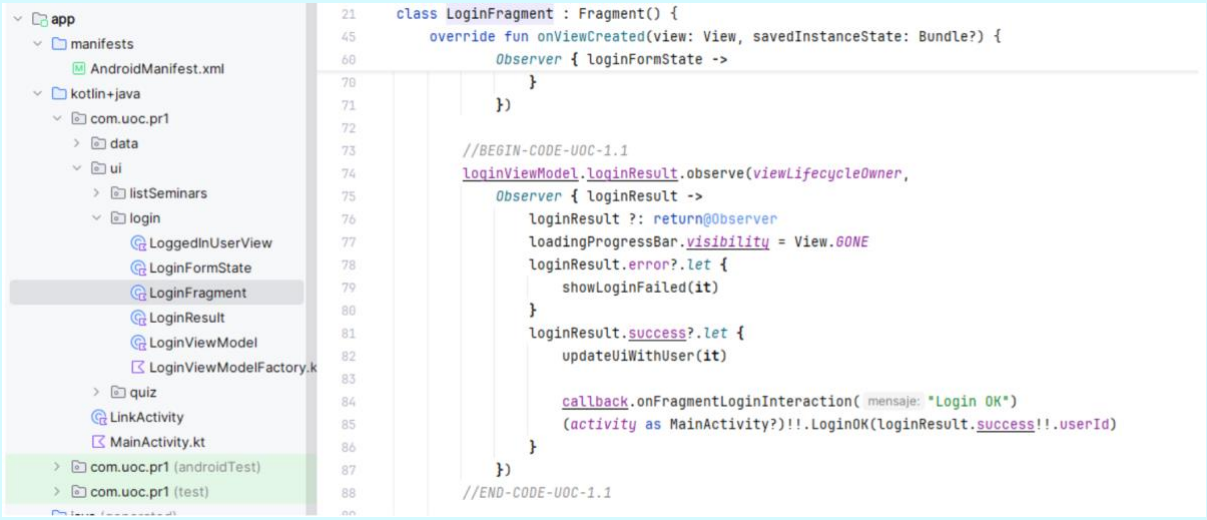
Continuous assessment Test 2 – CAT2

Nicolas D'Alessandro Calderon

Problem statement

1. Login screen (Weight: 10%)

The login screen is the first element of our app that will require accessing data. Remember that, in this activity, all the relevant data is hard-coded.



a) In the class `LoginFragment`, locate the callback `onViewCreated`. In a PDF explain in 5 lines or less what the block of code between `//BEGIN-CODE-UOC-1.1` and `//END-CODE-UOC-1.1` is doing.

Clue: Notice that the code is observing the property `loginResult` of our model. What is it doing?

(b) In the same callback, the code block between the comments `//BEGIN-CODE-UOC-1.2` and `//END-CODE-UOC-1.2` implements the response to clicking the login button. Also, in the PDF file explain in 5 lines or less what actions are performed when the button is pressed. Which method and class is being used? Why?

Clue: When you move your mouse over a method call while pressing the Ctrl, pressing the left button of the mouse takes you to the method declaration.

(c) Implement the following behavior: whenever we execute the login method from class `DataSourceHardcode`, if the username is "user1@uoc.com" you should return the user `User(1, "Jane Doe")`. Otherwise, you should return the user `User(2, "John Doe")`.

Your code should be placed between the placeholder comments `//BEGIN-CODE-UOC-1.3` and `//END-CODE-UOC-1.3`.

Login Screen answers:

(a)

- `LoginFragment.kt`
- `onViewCreated()` method

```
//BEGIN-CODE-UOC-1.1
loginViewModel.loginResult.observe(viewLifecycleOwner,
   Observer { loginResult ->
      loginResult ?: return@Observer
      loadingProgressBar.visibility = View.GONE
      loginResult.error?.let {
         showLoginFailed(it)
      }
      loginResult.success?.let {
         updateUiWithUser(it)

         callback.onFragmentLoginInteraction("Login OK")
         (activity as MainActivity?)!!.LoginOK(loginResult.success!!.userId)
      }
   })
//END-CODE-UOC-1.1
```

This code is observing (listening for changes) in `loginResult` from the `loginViewModel`, and when it changes:

1. Hide the progress bar.
2. If there´s an error, it calls `showLoginFailed(it)` to display an error message.
3. If the login is correct, the UI is updated with the user data (`updateUiWithUser(it)` call) and notify to the callback with the success of the login, that it then calls to the "`Login OK`" method in the `MainActivity` passing the ID of the user.

(b)

- `LoginFragment.kt`
- `onViewCreated()` method

```
//BEGIN-CODE-UOC-1.2
loginButton.setOnClickListener {
   loadingProgressBar.visibility = View.VISIBLE
   loginViewModel.login(
      usernameEditText.text.toString(),
      passwordEditText.text.toString()
   )
}
//END CODE-UOC-1.2
```

This fragment of code is implementing the functionality **when pressing the login button**, that when it is clicked:

1. Show the progress bar.
2. Calls the `login` method in the `loginViewModel` passing the username and the password input by the user.
3. The login method will be then in charge of handling the authentication using the repository and update the UI accordingly.

(c)

- `DataSourceHarcode.kt`
- `login()` method

```
//BEGIN-CODE-UOC-1.3
user_harcoded = if (username == "user1@uoc.com") {
   User(1, "Jane Doe")
} else {
   User(2, "John Doe")
}
//END-CODE-UOC-1.3
```

This code verifies if the username is equal to “user1@uoc.com” and returns a user with ID 1 and the name “Jane Doe”. For all the rest of usernames will return a user with the ID 2 and the name “John Doe”.

2. List of Seminars (Weight: 10%)

- `item_seminary.xml`: Added LinearLayout with TextView and an ImageView
- `SeminarsAdapter.kt`:
  - `init()` method: Code uncommented //BEGIN-CODE-UOC-2.1 and //END-CODE-UOC-2.1
  - `bind()` method: Code uncommented //BEGIN-CODE-UOC-2.2 and //END-CODE-UOC-2.2
- `fragment_list_seminars.xml`: Added background.

3. New Seminar button (Weight: 5%)

- `fragment_list_seminars.xml`: Added the button “New Seminar” in the bottom.
- `SeminarsFragment.kt`:
  - `onCreateView()` method: Listener added to show the Toast message when pressing the button.

4. Quiz design (Weight: 15%)

- `fragment_quiz.xml`: Added all the layout elements requested (four answers TextViews, TextView for HELP, buttons PREV, NEXT and FINISH) and the background.

5. Quiz logic (Weight: 30%)

- `QuizFragment.kt`:
  - `onCreateView()` method: Views and listener initialization.
  - `ButtonsViewLogic()` method: Visibility control of the buttons.
  - `LoadValuesCurrentQuestion()` method: Load questions and answers.
  - `BClickAnswerLogic()` method: Handle answer selection.

6. End quiz logic (Weight: 15%)

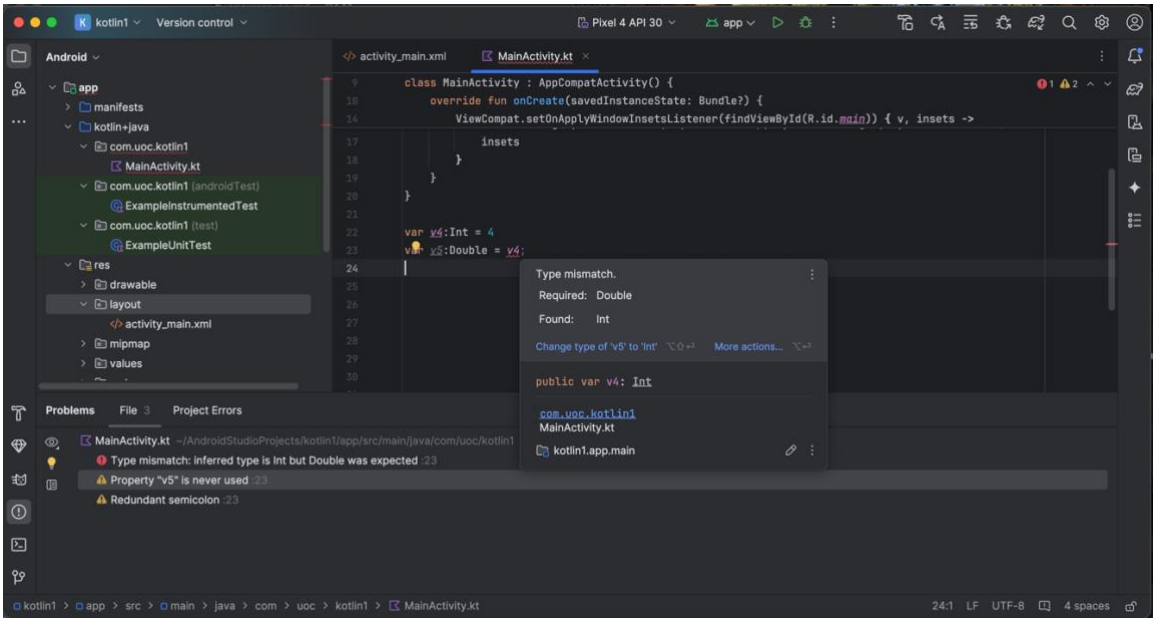
- `QuizFragment.kt`:
  - `Finish()` method: Calculation of the correct answers and call to returnFromQuiz
- `MainActivity.kt`:
  - `returnFromQuiz()` method: Show resutls

7. Link Activity (Weight: 15%)

- `activity_link.xml`: New file created and WebView added
- `LinkActivity.kt`: New file created
  - `onCreate()` method: Binding initiation and URL load.
  - Comment added of the permission in the Manifest file.
- `MainActivity.kt`:
  - `OpenLink()` method: LinkActivity implementation

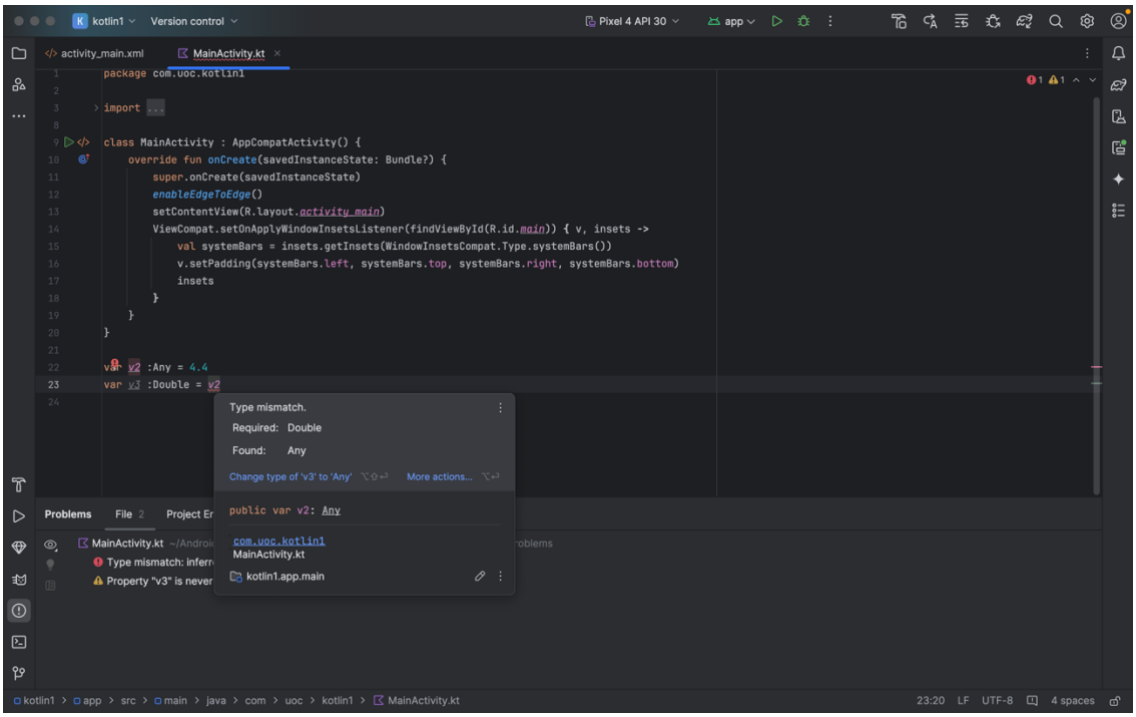
Kotlin fundamentals answers:

(a)
```
var v4 : Int = 4
var v5 :Double = v4;

// This is not working because we can´t assign an Int to a variable Double.
```




```
// To solve this, we can convert the Int to Double using the method toDouble()
// The .toDouble() conversion adds a .0 but does not lose any data.

var v4 : Int = 4
var v5 :Double = v4.toDouble;
```

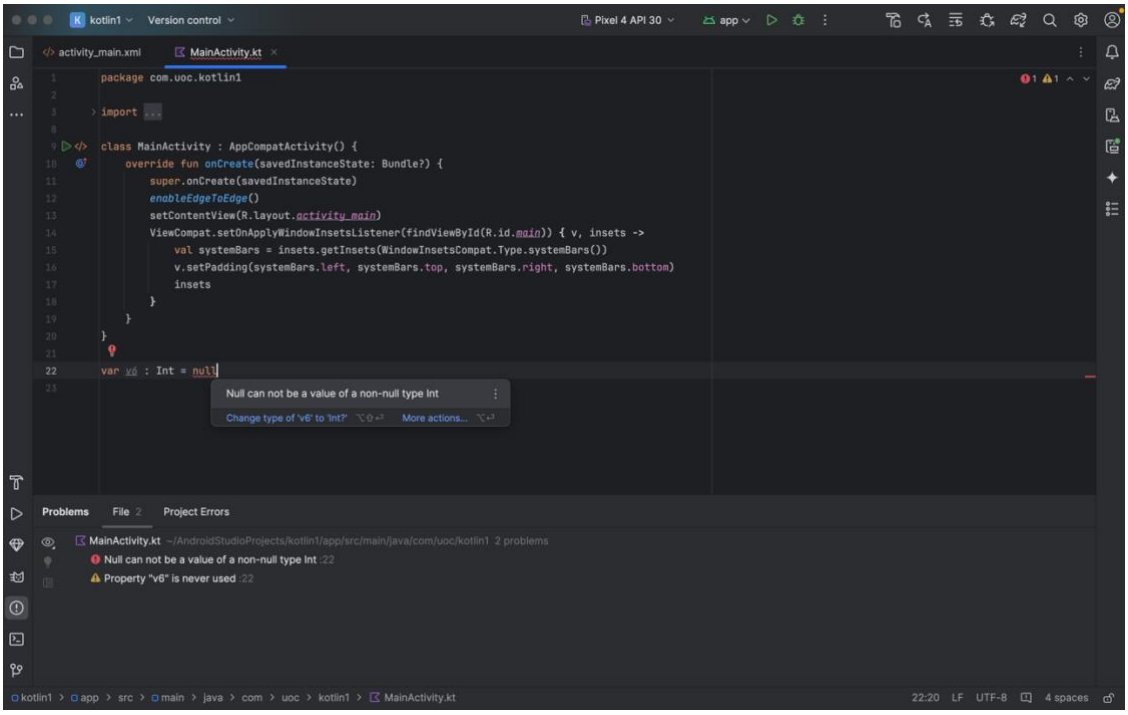
(b)
```
var v2 :Any = 4.4
var v3 :Double = v2

// This is not working because the variable v2 (even if it contains a Double)
// has been declared as Any, so when declaring a variable V3 as Double and
// assigning a variable Any to it , we have a “Type mismatch”.
```




```
// To solve this, we can cast V2 to Double using the as operator:

var v2 :Any = 4.4
var v3 :Double = v2 as Double
```

(d) var v6 : Int = null

// This is not working because Int is a non-nullable type in Kotlin:



// To solve this, we can make the var nullable by adding a question mark:

var v6 : Int? = null

8. Collections in Kotlin (Weight: 20%)

In this exercise, we will solve simple tasks using Kotlin collections (again in the context of the `MainActivity`).

(a) We want to access a car's price as quickly as possible using its model name. What data structure should we use? Then, add a car with its price to the structure and query its price using the model name.

(b) Create a list of String called `car_name_list`. Add 5 different items to the list and then remove the string in the second position. Finally, iterate through the list and print the value for each position in the Logcat window using `Log.d("debug",v).`

Collections in Kotlin answers:

(a) To quiclkly access to a car´s price using it s model name, we should use the HashMap data structure. With this structure we can do search by keys with O(1) complexity

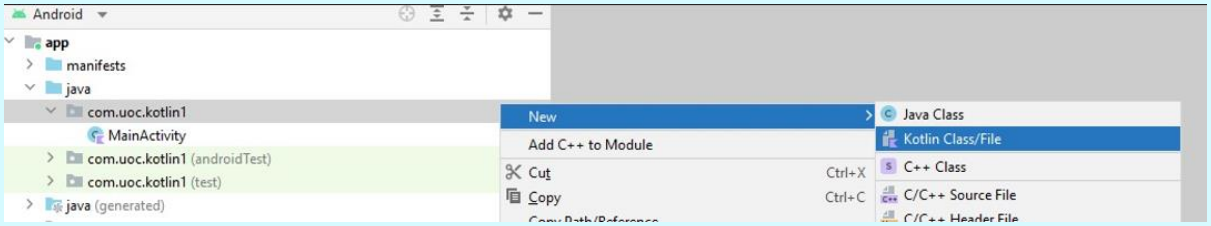
```
// Create a HashMap that store car models and their prices
val carPrices = HashMap<String, Int>()

// Add a car with its price
carPrices["Volkswagen TRoc"] = 34230000

// Query its price using the model name
val trocPrice = carPrices["Volkswagen TRoc"]
Log.d("CarPrice", "The Volkswagen TRoc Costs $34230000")
```

(b) List operations

```
// Creating a list of car names
val car_name_list = ArrayList<String>()

// Add 5 different items to the list
car_name_list.add("Polo")
car_name_list.add("T-Roc")
car_name_list.add("T-Cross")
car_name_list.add("Tiguan")
car_name_list.add("Golf")

// Removing the string in the second position (index 1)
car_name_list.removeAt(1)

// Iterate through the list and print the value for each position
for (carName in car_name_list) {
    Log.d("debug", carName)
}
```

9. Classes in Kotlin (Weight: 30%)

In this exercise, we will create and extend Kotlin classes (in the context of the app we created).



(a) Create a class called `Car`. It will include an attribute `name` of type `string` and an attribute `price` of type `Int`. Its constructor will have two parameters called `pname` and `pprice` that will initialize the previously mentioned attributes.

(b) Create a class called User. It will include an integer attribute called id and a string attribute called username. Its constructor will have a parameter called `pid` that will initialize the previously mentioned id attribute and username as null. Moreover, this class will have an attribute called cars of type `HashMap<String,Car>`. The HashMap key is the Car name.

(c) In class User, add a method with signature fun `addCar(d:Car)` that will allow adding its parameter d to the attribute of type `HashMap<String,Car>`.

(d) In class User, add a method with signature fun `removeCar(d:String)` that will allow removing the Car with name d from the attribute of type `HasMap` cars.

(e) Why has the price been declared as has the price been declared as type Int in the class Car instead of type double? Hint: What units are we storing?

Classes in Kotlin answers:

(a)

```
// Create a class called Car
class Car(pname: String, pprice: Int) {
    var name: String = pname
    var price: Int = pprice
}
```

***Note: We may use the keyword `val` and the constructor parameters automatically become property, but for more verbose purpouses of this CAT I have used `var`.

(b)

```
// Create a class called User
class User(pid: Int) {
    var id: Int = pid
    var username: String? = null
    var cars: HashMap <String, Car> = HashMap <String, Car>()
}
```

(c)

```
// In class User, add a method
class User(pid: Int) {
    var id: Int = pid
    var username: String? = null
    var cars: HashMap <String, Car> = HashMap <String, Car>()

    fun addCar(d: Car){
        cars[d.name] = d
    }
}
```

(d)

```
// In class User, add a method
class User(pid: Int) {
    var id: Int = pid
    var username: String? = null
    var cars: HashMap <String, Car> = HashMap <String, Car>()

    fun addCar(d: Car){
        cars[d.name] = d
    }

    fun removeCar(pname: String){
        cars.remove(pname)
    }
}
```

(e) Because we are storing the price the price in the minimal monetary unit (probably cents), with this we avoid precision problems with the float point when using Double. Example 34,23 euros can be stored as 3423 cents.

10. Use classes (Weight: 10%)

Going back to the `MainActivity` of our app, perform the following tasks:

(a) Create a User with `id 18`.

(b) Create a Car with name `"Ferrari Purosangue"` and price `39835000`.

(c) Add the Car to the information about the user.

(d) Remove the previous Car from the information about the user.

Use classes answers:

Option 1: Using the `var` keyword for variable declarations:

```
var user : User = User(18)
var car : Car = Car ("Ferrari Purosangue", 39835000)
user.addCar (car)
user.removeCar("Ferrari Purosangue")
```

Option 2: Using `val` keyword to create immutable variables

```
val user = User(18)
val car = Car("Ferrari Purosangue", 39835000)
user.addCar(car)
user.removeCar("Ferrari Purosangue")
```