

Mobile App Development

Continuous assessment Test 1 – CAT1

Nicolas D'Alessandro Calderon

Problem statement

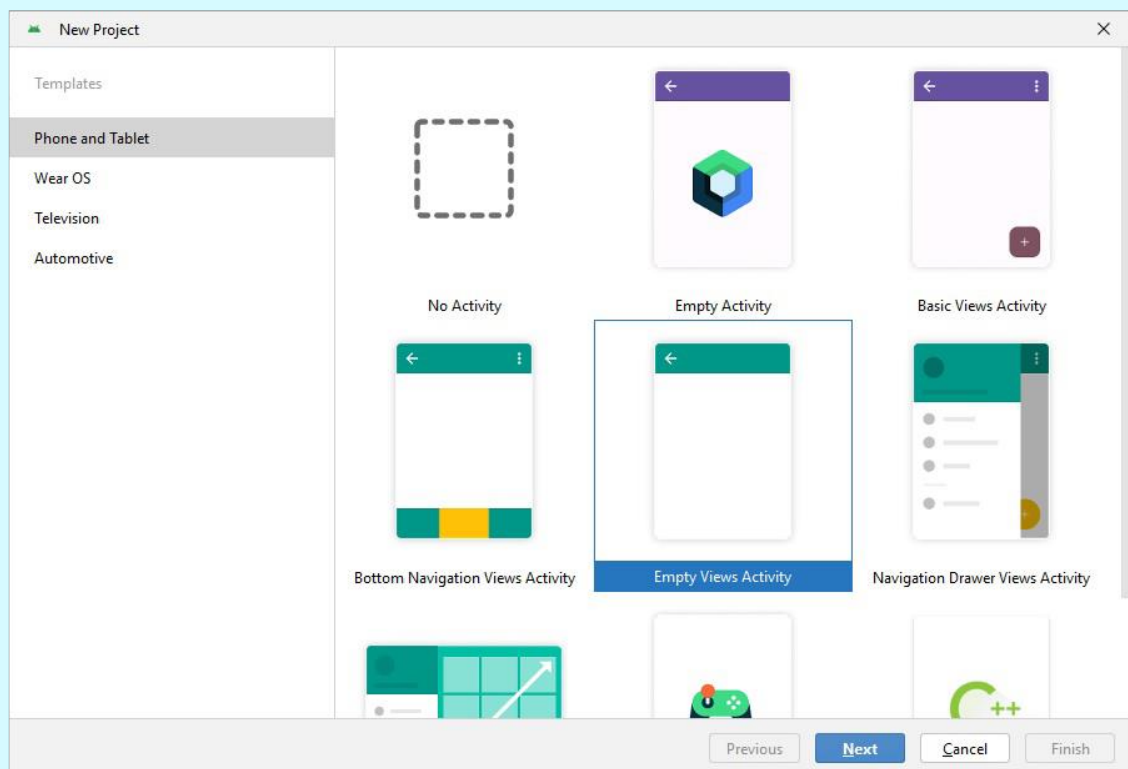
1. API levels (Weight: 20%)

Each Android API level offer different sets of features, with more recent API levels offering more capabilities. Explain from which API level it is possible to perform the following:

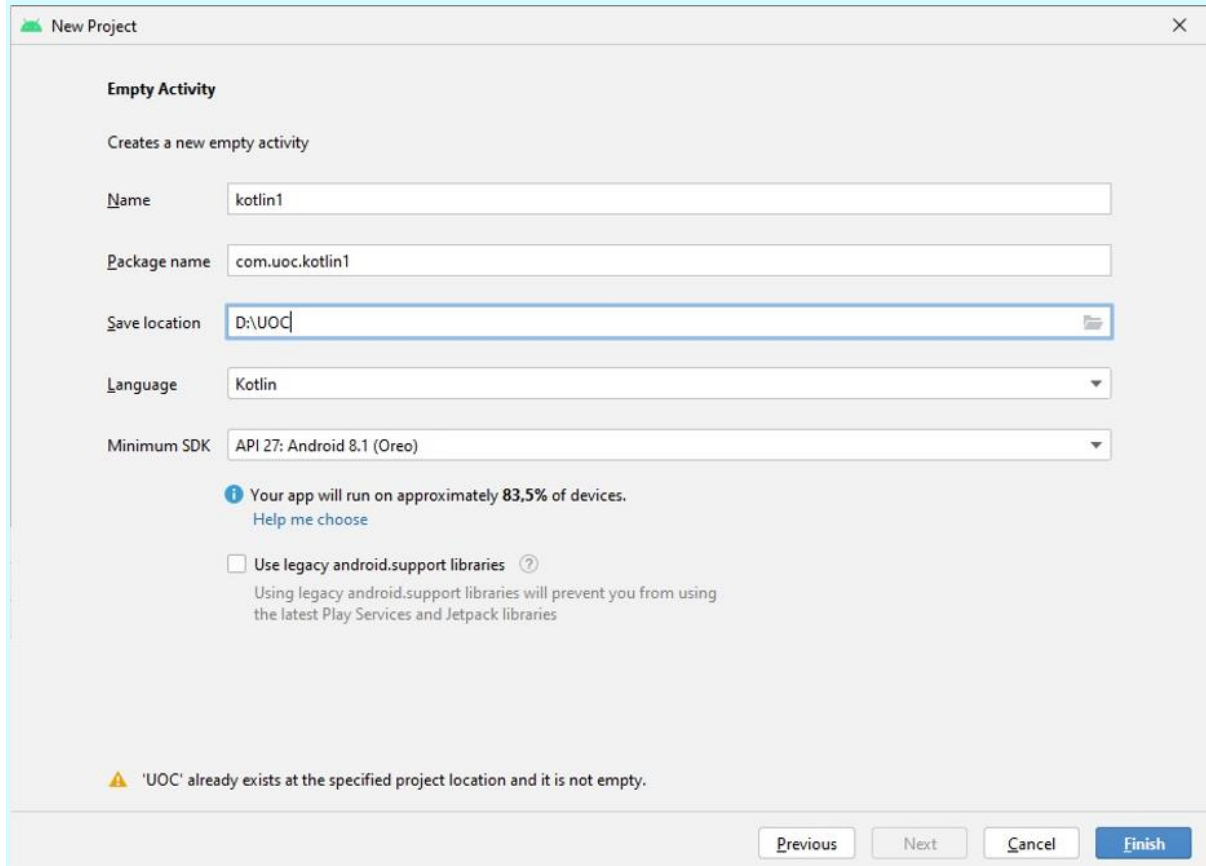
- (a) Animated vector graphics.
- (b) Full-screen applications.

In order to solve the remaining exercises of the CAT, we will need to create an Android app.

- First of all, we will need to install Android Studio and then create an app (using the menu option File/New/New Project or New Project from the starting dialog) of type Empty Views Activity.



We can use the following settings for this project:



New Project

Empty Activity

Creates a new empty activity

Name

Package name

Save location

Language

Minimum SDK

Information: Your app will run on approximately 83,5% of devices.
[Help me choose](#)

☐ **Use legacy android.support libraries** [?](#)
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

Warning: 'UOC' already exists at the specified project location and it is not empty.

Buttons: Previous, Next, Cancel, Finish

- Using the Device Manager, create a device of type Pixel 7, API level 33 and with Google API. Test the app you have created on this device, as you will be using it to solve the remaining exercises.

API levels answers:

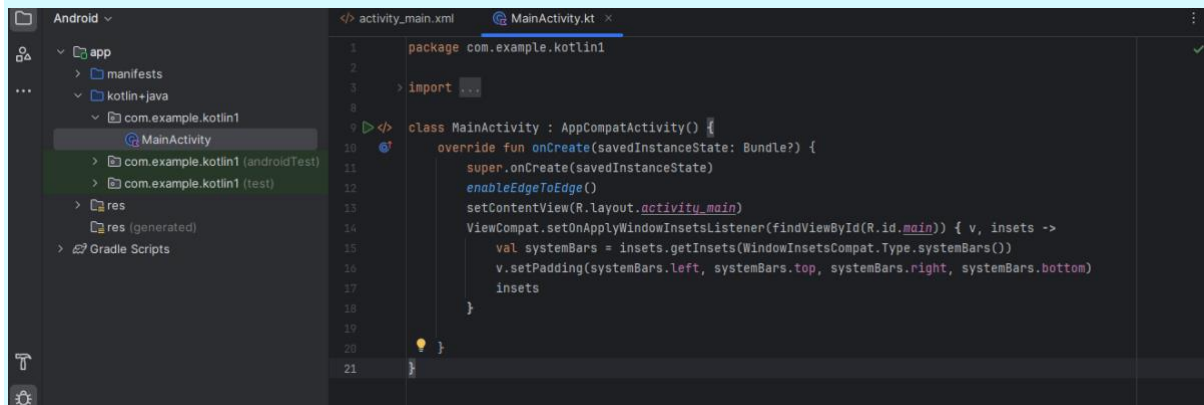
- The **Animated vector graphics** are available since the **API Level 21 – Android 5.0 Lollipop**. In this release Android added the class `AnimatedVectorDrawable` which allow to the developers to create animated graphics that scale without losing the quality.
- Some basic full-screen was possible from the earliest Android versions, but the most complete **Full-screen applications** mode with the possibility to hide system bars are available since the **API Level 19 – Android 4.4 KitKat**.
This functionality called “**immersive mode**” allow to the applications to hide the navigation bar as well as the status bar, generating a more complete full-screen experience. We can use the `SYSTEM_UI_FLAG_IMMERSIVE`, `SYSTEM_UI_FLAG_HIDE_NAVIGATION`, and `SYSTEM_UI_FLAG_FULLSCREEN` flags.

2. Kotlin fundamentals (Weight: 20%)

In this exercise, we will look at simple errors in Kotlin code and how they can be fixed.

Find the class `MainActivity` in the Empty Views Activity application we just created.

There, look for method override `fun onCreate(savedInstanceState: Bundle?)`. Position yourself at the end of this method.



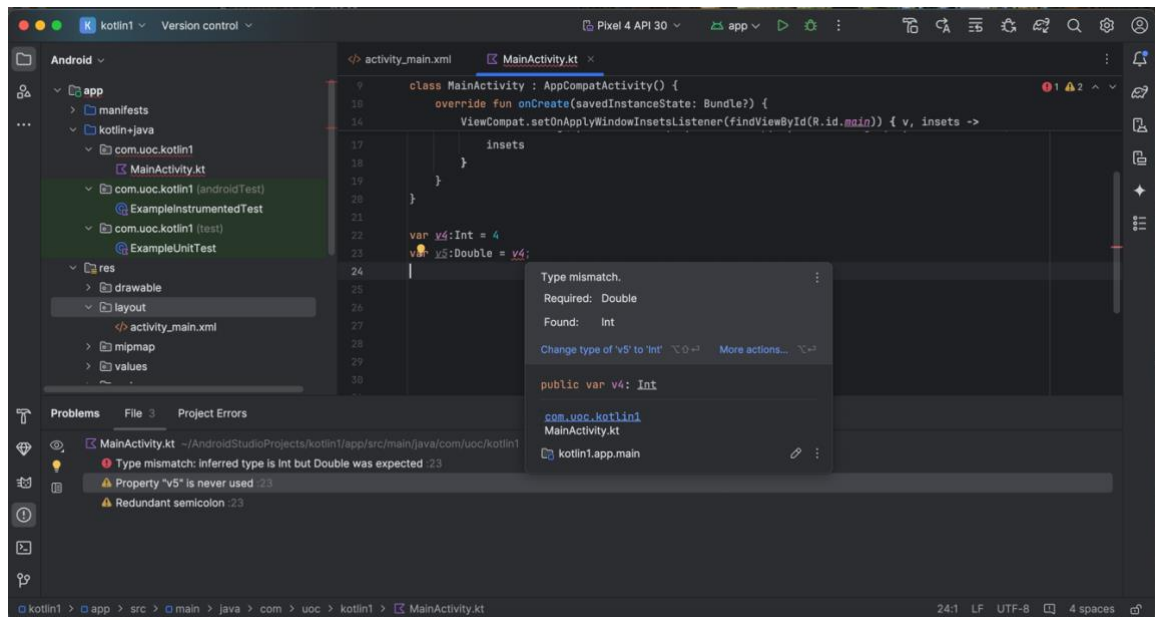
Then, copy each of the following (incorrect) code snippets one by one. For each one, explain why they do not work and propose a fix that solves the problem. You can leave the erroneous code commented, adding your explanation within the comment. You can only add the symbol `'?'` to the left side of the assignment statement if you need it. You must do the other modifications on the right side. In some cases, there may be more than one correct answer. You should choose the one that has the lowest CPU cost. The `" as "` cast is the lowest cost operator because it does nothing at runtime. The `" as "` operator only acts at compile time.

- (a) `var v4 : Int = 4`
`var v5 :Double = v4;`
- (b) `var v2 :Any = 4.4`
`var v3 :Double = v2`
- (c) `var v6 : Int = null`

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.

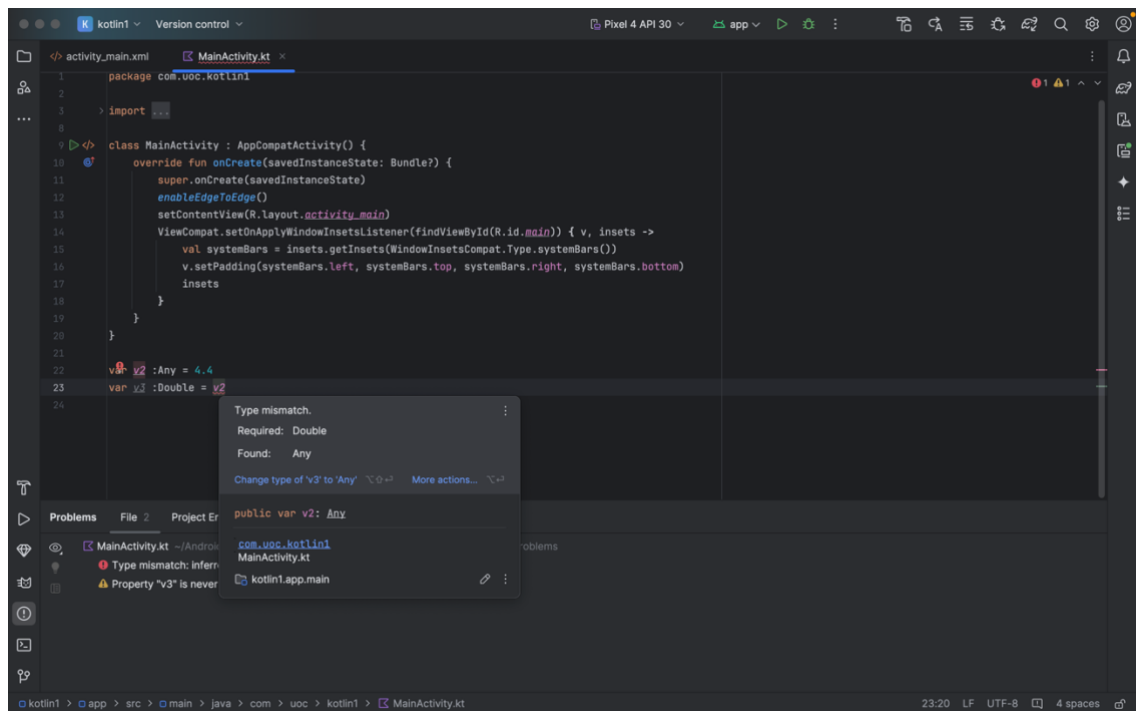


// To solve this, we can convert the Int to Double using the method `toDouble()`
 // The `.toDouble()` conversion adds a `.0` but does not lose any data.

`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”.



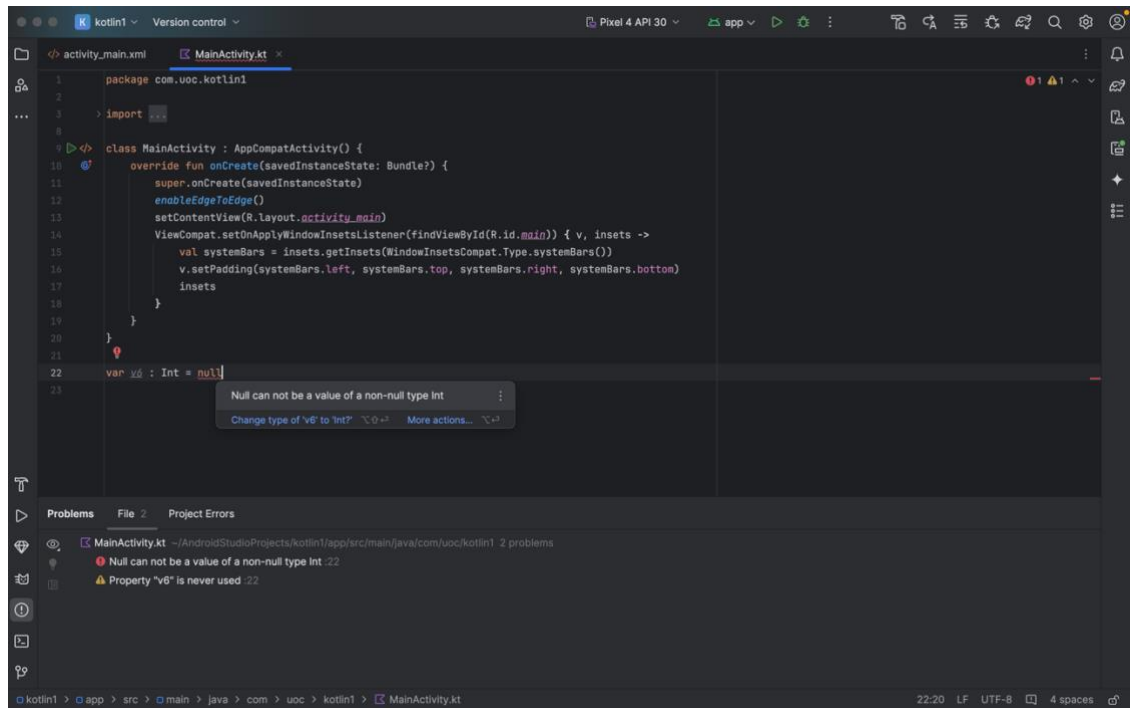
// To solve this, we can cast V2 to Double using the as operator:

```

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

(c) `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`

3. 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 quickly access to a car's price using its 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 osts $34230000")
```

(b) List operations

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

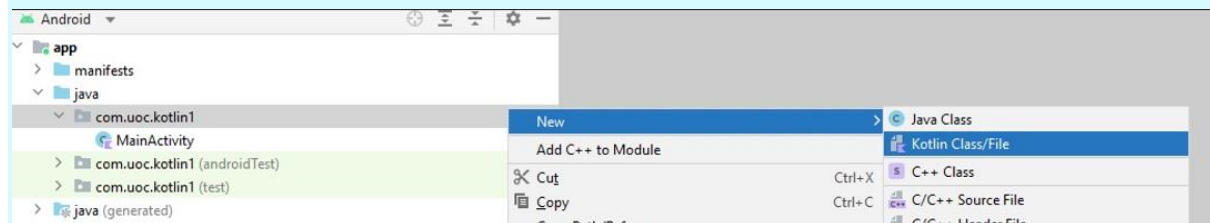
// Add 5 different items to the list
car_name_list.add("Polo")
car_name_list.add("T-Roc")
car_name_list.add("T-Cross")
car_name_list.add("Tiguan")
car_name_list.add("Golf")

// 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)
}
```


4. Classes in Kotlin (Weight: 30%)

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



- 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.
- 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.
- 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>`.
- 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 `HashMap` `cars`.
- Why has the price been declared as `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 purposes 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.

5. 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")
```