IT UNIVERSITY OF COPENHAGEN

Exercise #01

IT University of Copenhagen (ITU)
Mobile App Development, BSc. (MOAPD)
(Spring 2024)

January 30, 2024

Introduction Throughout the semester, you will create an Android app named "CopenhagenBuzz." This app is about sharing events in the Copenhagen area, like festivals and concerts. You will build it step by step, enhancing the app's look each week and adding new features based on what you learn in class. You will turn in your finished app for a mandatory assignment at the end of the course. For Assignment #01, you must do the following tasks:

- ☐ Choose `CopenhagenBuzz` as the project name.
- ☐ Set the package name to `dk.itu.moapd.copenhagenbuzz.<YOUR ITU'S ACCOUNT>`.
- ☐ Create a responsive layout that adapts vertically and horizontally to fit various phone and tablet sizes and resolutions.

CopenhagenBuzz App Version 1 This week's task is to implement version 1.0.0 of the CopenhagenBuzz App. The user interface should allow users to input data for a single event. This includes the `event name`, `location`, `start date` (using `java.util.Date` for simplicity), `end date`, `event type` (selected from predefined options), a `short description`, and the URL/URI of the `event photo`. Once users fill in these details, they can set the attributes of the `Event` object by tapping the `Done` button. The class properties describe the sentence that appears in the log system, generated by sending a message via the `Log.d()` method. Feel free to adjust the class to meet your specific requirements or add extra functionality as needed. The goals you will achieve through this exercise include:

- Mastering the art of building mobile apps using Android Studio.
- Crafting your inaugural Android user interface (UI).
- Establishing connections between your UI components and source code.
- Implementing anonymous inner classes and UI components listeners.
- Navigating the Log System API effectively.
- Getting acquainted with the Kotlin programming language.

IT UNIVERSITY OF COPENHAGEN

The user interface for CopenhagenBuzz version 1.0.0 should correspond to the layout shown in Figure 1.

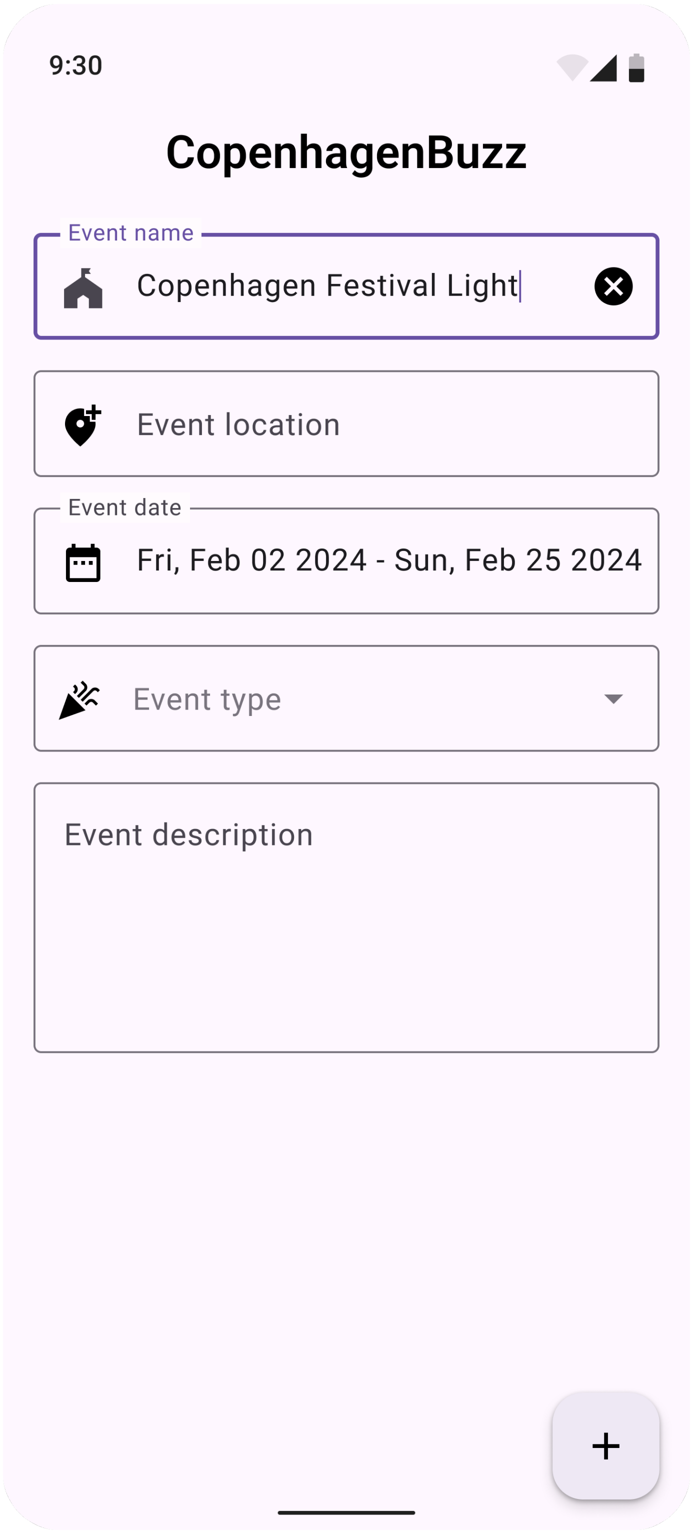

Figure 1: The initial version of the CopenhagenBuzz app.

Exercise 01.01. *CopenhagenBuzz app version 1.0.0* – For this exercise, follow the instructions outlined on pages 4-6 of Textbook #01[1] (Chapter 1, Your First Android Application) to build the CopenhagenBuzz app version 1.0.0 using Android Studio:

(1) **On page 5:** Select `Basic Views Activity` as shown in Figure 2;

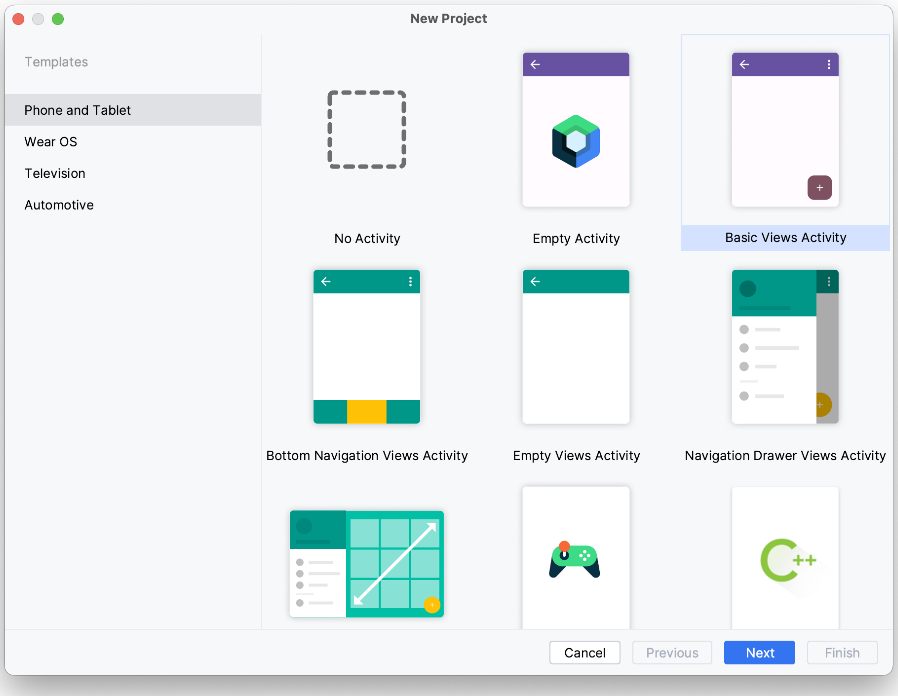

Figure 2: You must choose option "Basic Views Activity" in the "New Project" dialog.

(2) **On page 6:** You must set the following configuration settings:

- Name your application `CopenhagenBuzz`.
- Set the package name as `dk.itu.moapd.copenhagenbuzz.<YOUR ITU'S ACCOUNT>`.
- Choose a save location without spaces or special characters.
- Select `Kotlin` as the programming language.
- Set the minimum API level to `API 26 ("Oreo"; Android 8.0)`.
- Set the build configuration to `Kotlin DSL (build.grade.kts)[Recommended]`.

(3) After creating the project, there are some unnecessary files that you must delete. You can delete the following files:

- `FirstFragment.kt`.

[1] Android Programming – The Big Nerd Ranch Guide

IT UNIVERSITY OF COPENHAGEN

- `SecondFragment.kt`.
- `res/layout/fragment_first.xml`.
- `res/layout/fragment_second.xml`.
- `res/menu/menu_main.xml`.
- `res/navigation/nav_graph.xml`.

(4) You must also delete the following lines (source-code) from these files:

- `res/layout/activity_main.xml`: Line 3, Lines 10-20, and Lines 24-31.
- `res/layout/content_main.xml`: Lines 8-18.
- `res/values/strings.xml`: Lines 3-45.
- `MainActivity.kt`:
  - Lines 25-34.
  - `appBarConfiguration` attribute.
  - `onCreateOptionsMenu()` method.
  - `onOptionsItemSelected()` method.
  - `onSupportNavigateUp()` method.
  - Unused import statements.

(5) By following these steps, you should now have an Android project with all the necessary files. You must implement your `Kotlin` code in the `MainActivity.kt` file and design the primary user interface in the `content_main.xml` file.

Exercise 01.02. *UI Layout* – In this exercise, you will modify the default files of the CopenhagenBuzz app version 1.0.0. For creating Android app user interfaces, we strongly suggested to use `ConstraintLayout`[2]. These layouts provide flexibility and a potent constraint-based system for arranging UI elements. `ConstraintLayout` offer a versatile approach for crafting intricate UIs, enabling precise positioning and alignment. This is important to ensure your UI remains responsive across different screen sizes and orientations. Check out Listing 1 for the XML code, which includes a code skeleton example of the UI demonstrated in Figure 1.

Listing 1: Source code skeleton of res/layout/content_main.xml.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

[2]Build a responsive UI with `ConstraintLayout` https://developer.android.com/develop/ui/views/layout/constraint-layout

IT UNIVERSITY OF COPENHAGEN

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        style="?textAppearanceHeadlineSmall"
        android:id="@+id/text_view_app_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/margin_medium"
        android:text="@string/app_name"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/text_field_event_name"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="@dimen/margin_medium"
        android:layout_marginStart="@dimen/margin_medium"
        android:layout_marginTop="@dimen/margin_large"
        android:hint="@string/event_name"
        app:startIconDrawable="@drawable/baseline_festival_24"
        app:endIconMode="clear_text"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/text_view_app_name">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/edit_text_event_name"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="text" />

    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/text_field_event_location"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="@dimen/margin_medium"
        android:layout_marginStart="@dimen/margin_medium"
        android:layout_marginTop="@dimen/margin_standard"
        android:hint="@string/event_location"
        app:startIconDrawable="@drawable/baseline_add_location_alt_24"
        app:endIconMode="clear_text"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/text_field_event_name">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/edit_text_event_location"
            android:layout_width="match_parent"
```

```
57            android:layout_height="wrap_content"
58            android:inputType="text" />
59
60    </com.google.android.material.textfield.TextInputLayout>
61
62    <!-- Implement the other UI components here. -->
63
64 </androidx.constraintlayout.widget.ConstraintLayout>
```

Please complete all the missing UI components as guided by the comment in Line 60. It is acceptable if your final layout differs from the one illustrated in Figure 1 as long as it includes, at a minimum, the elements shown in Figure 1.

Exercise 01.03. *The Kotlin Code* – For this exercise, your task is to implement two Kotlin files: (i) `MainActivity.kt` and (ii) `Event.kt`. The `Event` class represents the event scheduled in Copenhagen. Although you will create various versions of this class in the upcoming weeks, Listing 2 shows the the initial version of the `Event` class:

Listing 2: Source code of the `Event` class.

```kotlin
package dk.itu.moapd.copenhagenbuzz.<YOUR ITU's ACCOUNT>

class Event {

    private var eventName: String
    private var eventLocation: String
    // Add the missing attributes.

    constructor(eventName: String, eventLocation: String) {
        this.eventName = eventName
        this.eventLocation = eventLocation
        // Initialize the missing attributes.
    }

    fun getEventName(): String {
        return eventName
    }

    fun setEventName(eventName: String) {
        this.eventName = eventName
    }

    fun getEventLocation(): String {
        return eventLocation
    }

    fun setEventLocation(eventLocation: String) {
        this.eventLocation = eventLocation
    }

    // Implement the missing accessors and mutators methods.

    override fun toString(): String {
        return "Event(eventName='$eventName',
            eventLocation='$eventLocation')"
    }

}
```

Important Note: The current implementation in Kotlin might seem unconventional. I have shared this code for its similarity to a Java-based structure. Do not worry; we will study deeper into Kotlin next week, where you will gain a better understanding and be able to enhance and optimize the `Event.kt` file. Meanwhile, the Kotlin source code below provides an essential structure for `MainActivity.kt`:

Listing 3: Source code of the `MainActivity` class.

```kotlin
package dk.itu.moapd.copenhagenbuzz.<YOUR ITU's ACCOUNT>

import android.os.Bundle
import android.util.Log
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.WindowCompat
import
    com.google.android.material.floatingactionbutton.FloatingActionButton
import dk.itu.moapd.copenhagenbuzz.<YOUR ITU's
    ACCOUNT>.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    // A set of private constants used in this class.
    companion object {
        private val TAG = MainActivity::class.qualifiedName
    }

    // GUI variables.
    private lateinit var eventName: EditText
    private lateinit var eventLocation: EditText
    private lateinit var addEventButton: FloatingActionButton

    // TODO: Implement the missing GUI variables

    // An instance of the `Event` class.
    private val event: Event = Event("", "")

    override fun onCreate(savedInstanceState: Bundle?) {
        WindowCompat.setDecorFitsSystemWindows(window, false)
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // IMPORTANT: This is an awful implementation. I only implemented
        //            it like that because the students need to learn more
        //            about Kotlin. This implementation is quite similar
        //            to a Java code.  We will refactor this code in the
        //            next exercise session.

        // Link the UI components with the Kotlin source-code.
        eventName = findViewById(R.id.edit_text_event_name)
        eventLocation = findViewById(R.id.edit_text_event_location)
        addEventButton = findViewById(R.id.fab_add_event)

        // Listener for user interaction in the ‘Add Event‘ button.
        addEventButton.setOnClickListener {

            // Only execute the following code when the user fills all
            // ‘EditText‘.
            if (eventName.text.toString().isNotEmpty() &&
                eventLocation.text.toString().isNotEmpty()) {

                // Update the object attributes.
                event.setEventName(
                    eventName.text.toString().trim()
                )
                event.setEventLocation(
                    eventLocation.text.toString().trim()
                )

                // TODO: Implement the missing code here.

                // Write in the ‘Logcat‘ system.
                showMessage()
            }
        }
    }

    private fun showMessage() {
        Log.d(TAG, event.toString())
    }

}
```

Once more, you must complete all the currently incomplete sections, as denoted by the `TODO` comments.