5COSC023W - Tutorial 7 Exercises

As part of the tutorial for this week, you should complete **ALL** the tasks described in the following specifications: (**make sure that you ask questions to your tutor for anything that you do not understand or if you are stuck at any point**).

Tutorial sessions are practical sessions that you need to work towards the exercises set. They will give you the chance to practice the material learned in the lectures and learn new things as well.

You should not use these sessions to work towards the assessed assignments! If you decide to work towards your assessed work instead, then you are not considered as part of the tutorial session. You will not get any help on the code of the assessed work by your tutor but you can ask your tutor ONLY about any clarifications you might need regarding the specification of the coursework.

Like all other modules, you are expected to study towards you module outside the lecture and dedicated tutorial slots for a number of hours. If you do not finish all of the exercises in the tutorial session, make sure that you finish them on your own time and by the end of the week. This is a normal process and part of your university learning.

For all the tasks you should use Jetpack Compose and NOT Views!

1 Extending the Lecture Example

Implement the example developed in the lecture which restores the state of the application after the mobile device is rotated.

1. Extend the application so that the background colour alternates between red, green and yellow. I.e. the first time the button is clicked the background becomes red, after the second click it becomes green, after the third click it becomes blue, after the fourth click it becomes red, etc.
2. Add an additional label displayed next to the score which displays the name of the user. The name should be typed by adding a new `TextField` that the user type their name. The label with the name next to the score should be retained even after the device is rotated.
3. Extend the application so that it displays an additional label with the total number of times that the mobile device has been rotated.

   For this part you should use `onSaveInstanceState()` **and NOT** `rememberSaveable`.

2 Implementing a Count Down Timer

The abstract class `CountDownTimer` in the `android.os.CountDownTimer` package implements a count down timer. As it is an abstract class you need to implement its abstract methods `override fun onFinish()` and `override fun onTick(millisUntilFinished: Long)` before using it.

The constructor of the class `CountDownTimer(millisInFuture, countDownInterval)` accepts 2 arguments: how the duration of the timer (in milliseconds) and the how often the timer will "tick" (in milliseconds).

The `onTick` method will be called every `countDownInterval` milliseconds and the `onFinish` method will be called on completion.

Once an object is created (from a concrete subclass of the abstract class), the timer can be started by calling the `start()` method.

An abstract class needs to be subclassed before it can be used or we could create an anonymous subclass and an object out of the anonymous class using the following:

```
val timer = object : CountDownTimer(10000, 1000) {
            override fun onFinish() {
                // ... what happens when the count down is over
            }

            override fun onTick(millisUntilFinished: Long) {
                // ... what happens on every tick
            }
        }.start()
```

The following example illustrates all of this.

Make sure that you type all of the code and ask your tutor if you do not understand something in it.

The application displays a countdown from 10 to 0 secs and it displays a message when it is done.

```
package uk.ac.westminster.countdowntimerexample

import android.os.Bundle
import android.os.CountDownTimer
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
```

```kotlin
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.sp


class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            displayGUI()
        }
    }
}

@Composable
fun displayGUI() {
    var time_left by remember{ mutableStateOf("10") }
    var run_timer_before by remember{ mutableStateOf(false) }

    Column(Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center) {
        Text("$time_left", fontSize = 50.sp)
    }

    if (!run_timer_before) {
        run_timer_before = true
        val timer = object : CountDownTimer(10000, 1000) {
            override fun onFinish() {
                time_left = "Time is up!"
            }

            override fun onTick(millisUntilFinished: Long) {
                time_left = "" + millisUntilFinished/1000
            }
        }.start()
    }
}
```

- Extend the application so that it contains a `TextField`. The user types an arbitrary value in the textfield and the count down will start from that value down to 0.
- Extend the application so that it contains a second `TextField`. The user types an integer in that field and the countdown timer will count down from the value of the first textfield down to the value of the second textfield.

3 A Tic-Tac-Toe Game

In this exercise we will develop an Android version of the classic tic-tac-toe game.

The game is played with two players, **X** (user) and **O** (computer), who take turns marking the spaces in a 3 × 3 grid. The **X** player goes first. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game (see Figure 1 below).

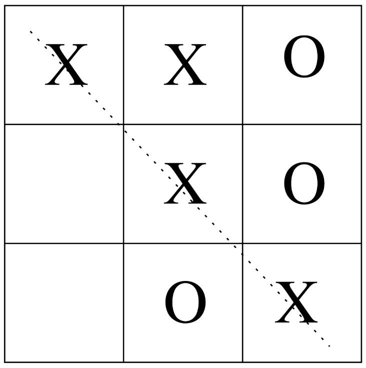

Figure 1: An instance of the tic-tac-toe board where the X player has won since he managed to place three 'X' in a diagonal.

1. Implement an Android version of the game that a human player can use to play against the computer. Initially, the computer player chooses a random cell as its next move.

   Hint: Try to develop the game from scratch. Each cell in the grid could be represented by a clickable button.

   If after 15–20 minutes you do not know how to start and after asking your tutor, download the skeleton of a sample solution from the following URL and start filling the blank code identified as `*** TODO ***`:

   `https://ddracopo.github.io/DOCUM/courses/5cosc023w/tic_tac_toe_composable_skeleton.zip`

2. Create a new function `checkWinner(): Char` inside a new Kotlin file `Board.kt` which checks if there is a winner. The function returns 'X' if player `X` has won and 'O' if player `O` has won. If there is no winner, the function returns '-'.

3. Call the `checkWinner()` function at the appropriate places within the main activity to check for winners.

4. Modify your code so that once a winner is determined, a `Toast` pop up message appears with the winner information and make sure that the game is not playable any more (**Hint:** Make the buttons non-clickable, using the `enabled` attribute).