[Your Name]
16/03/2025

Task Assignment 2: – Spike: Rent with Intent

Goals:

This task addresses the following Unit Learning Outcomes:

- ULO1 Explain the key differences between development of systems to run on mobile devices and on typical personal computing or internet-based environments, and apply this knowledge in the design of mobile device software.
- ULO2 Design effective applications for a mobile device by taking into consideration the underlying hardware-imposed restrictions such as screen size, memory size and processor capability.
- ULO3 Build, test and debug graphical applications for mobile devices by using the standard libraries that are bundled as part of the developers' toolkit for the mobile device.

The following list outlines the goal broken down into more specific knowledge gaps involved in the goal.

- Designing the UI without using RecyclerView
- Passing data around with Parcelable objects
- Keeping data and states upon changing to a new orientation screen (from assignment 1)

Tools and Resources Used

This section lists related software, tools, libraries, API's, and other resources used for this knowledge gap.

- Android Studio
- Android SDK
- Kotlin language
- Espresso (for testing purposes)

Knowledge Gaps and Solutions

This section presents the listed knowledge gaps and their solutions with supporting images, screenshots and captions where appropriate/required.

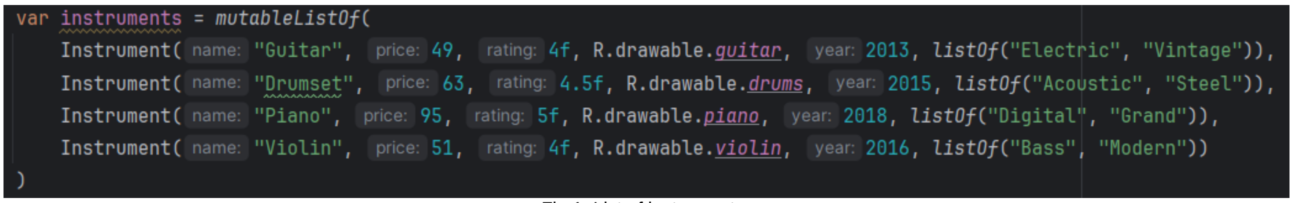
Gap 1: Designing the UI without using RecyclerView

Problem: Display 3-4 instruments available for rent without using RecyclerView (as stated in the instruction).

Solution:

1. Create a single view layout (in my case I went with ConstraintLayout) that has ImageView, TextView, RatingBar and a few Buttons to navigate to the other screen.

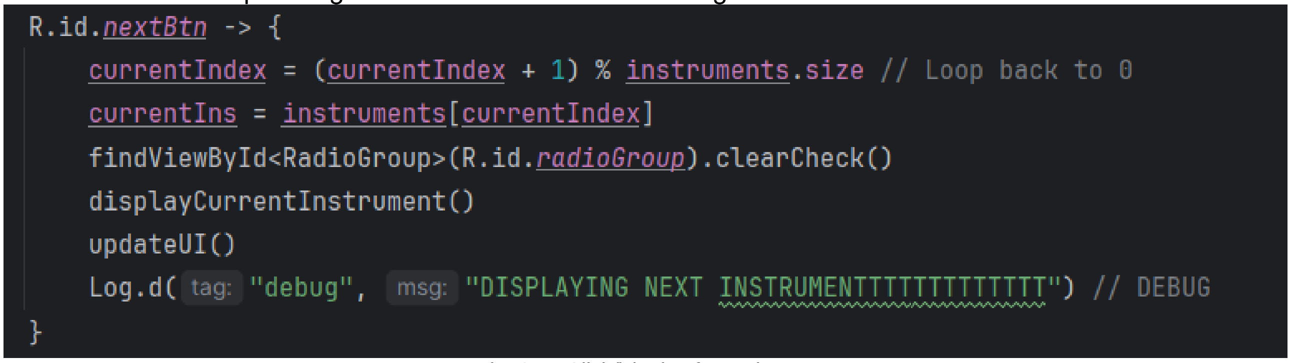
2. In MainActivity.kt, create a *mutableListOf()* List of instruments in-memory and uses array indexes to keep track of them.



```
var instruments = mutableListOf(
    Instrument( name: "Guitar",  price: 49,  rating: 4f, R.drawable.guitar,  year: 2013, listOf("Electric", "Vintage")),
    Instrument( name: "Drumset",  price: 63,  rating: 4.5f, R.drawable.drums,  year: 2015, listOf("Acoustic", "Steel")),
    Instrument( name: "Piano",  price: 95,  rating: 5f, R.drawable.piano,  year: 2018, listOf("Digital", "Grand")),
    Instrument( name: "Violin",  price: 51,  rating: 4f, R.drawable.violin,  year: 2016, listOf("Bass", "Modern"))
)
```

Fig.1. List of instruments.

3. “Next” button is used to display the next item in the list, there is no “Back” button, as pressing “Next” on the last item will navigate back to the first one in the list.



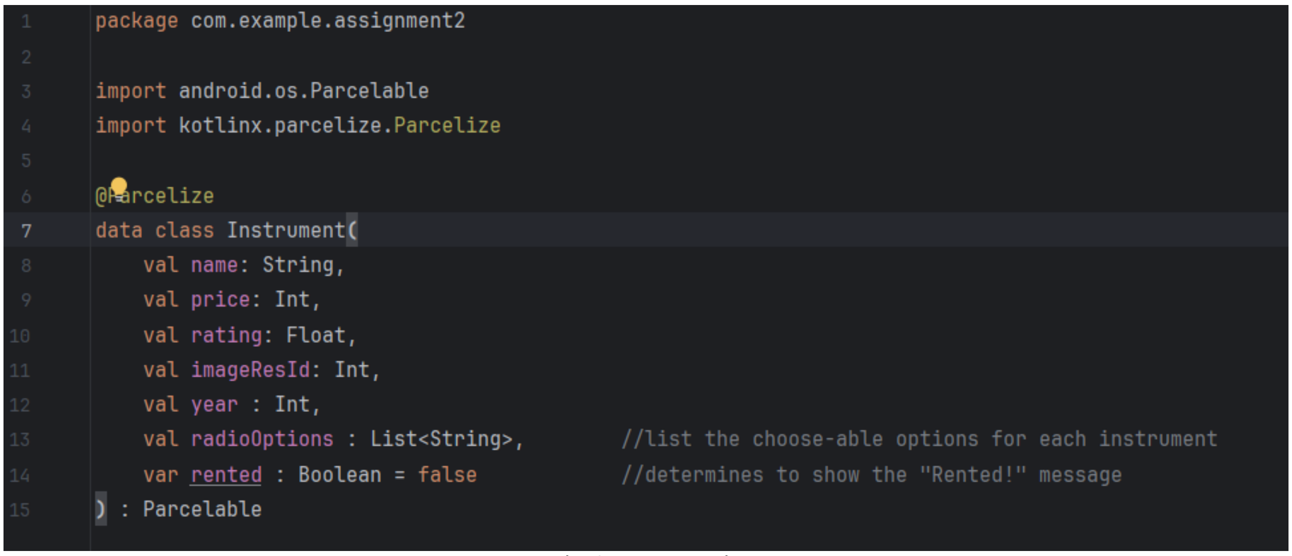
```
R.id.nextBtn -> {
    currentIndex = (currentIndex + 1) % instruments.size // Loop back to 0
    currentIns = instruments[currentIndex]
    findViewById<RadioGroup>(R.id.radioGroup).clearCheck()
    displayCurrentInstrument()
    updateUI()
    Log.d( tag: "debug",  msg: "DISPLAYING NEXT INSTRUMENTTTTTTTTTTTTT") // DEBUG
}
```

Fig. 2. onClick() logic of Next button

Gap 2: Passing data around as Parcelable object

Problem: When “Borrow” button is clicked, the current instrument on screen must somehow be “shipped” to another screen (DetailsActivity.kt), and upon “Save”, must somehow display back to the main screen with updates.

Solution: Create a Parcelable data class:



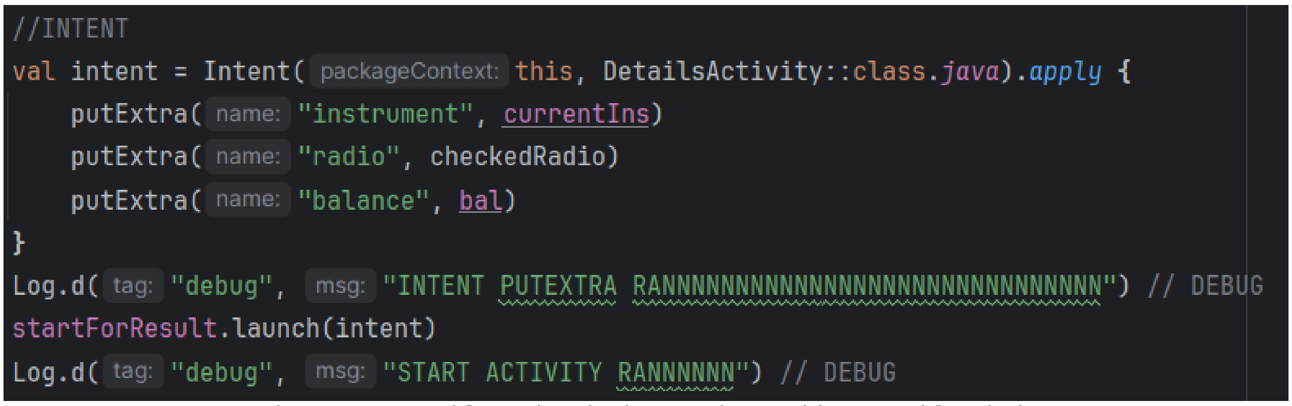
```
package com.example.assignment2

import android.os.Parcelable
import kotlinx.parcelize.Parcelize

@Parcelize
data class Instrument(
    val name: String,
    val price: Int,
    val rating: Float,
    val imageResId: Int,
    val year : Int,
    val radioOptions : List<String>,        //list the choose-able options for each instrument
    var rented : Boolean = false            //determines to show the "Rented!" message
) : Parcelable
```

Fig. 3. Instrument.kt

Later, I created a `private lateinit var currentIns: Instrument` to pass the current instrument to DetailsActivity.kt

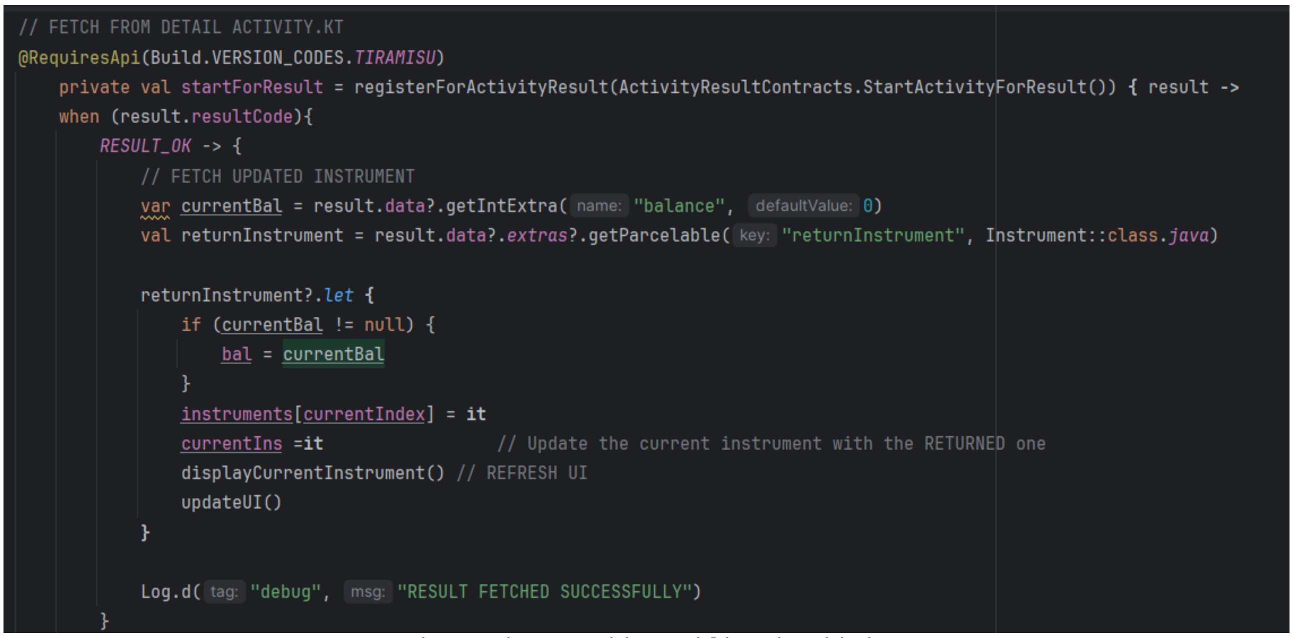


```
//INTENT
val intent = Intent( packageContext: this, DetailsActivity::class.java).apply {
    putExtra( name: "instrument", currentIns)
    putExtra( name: "radio", checkedRadio)
    putExtra( name: "balance", bal)
}
Log.d( tag: "debug",  msg: "INTENT PUTEXTRA RANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN") // DEBUG
startForResult.launch(intent)
Log.d( tag: "debug",  msg: "START ACTIVITY RANNNNNNN") // DEBUG
```

Fig. 4. startForResult() to replace the deprecated startActivityForResult() method.

The Intent tells Android Studio which activity to launch, in this case, DetailsActivity, and I passed a total of 3 different kind of data to it using *putExtra()*: An Instrument object which carries the current instrument's information (to further display), a string value of the user's chosen radio option and the current balance of the user (for error checking). I also intentionally lengthen the console log so that I could notice it faster while debugging when testing with *startForResult()* method.

While on this matter, I will also explain the fetching results process back from DetailsActivity.kt to display a message back to the user notifying of their successfully rented instrument.



```
// FETCH FROM DETAIL ACTIVITY.KT
@RequiresApi(Build.VERSION_CODES.TIRAMISU)
    private val startForResult = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
    when (result.resultCode){
        RESULT_OK -> {
            // FETCH UPDATED INSTRUMENT
            var currentBal = result.data?.getIntExtra( name: "balance",  defaultValue: 0)
            val returnInstrument = result.data?.extras?.getParcelable( key: "returnInstrument", Instrument::class.java)

            returnInstrument?.let {
                if (currentBal != null) {
                    bal = currentBal
                }
                instruments[currentIndex] = it
                currentIns =it                  // Update the current instrument with the RETURNED one
                displayCurrentInstrument() // REFRESH UI
                updateUI()
            }

            Log.d( tag: "debug",  msg: "RESULT FETCHED SUCCESSFULLY")
        }
```

Fig. 5. *registerForActivityResult()* in MainActivity.kt

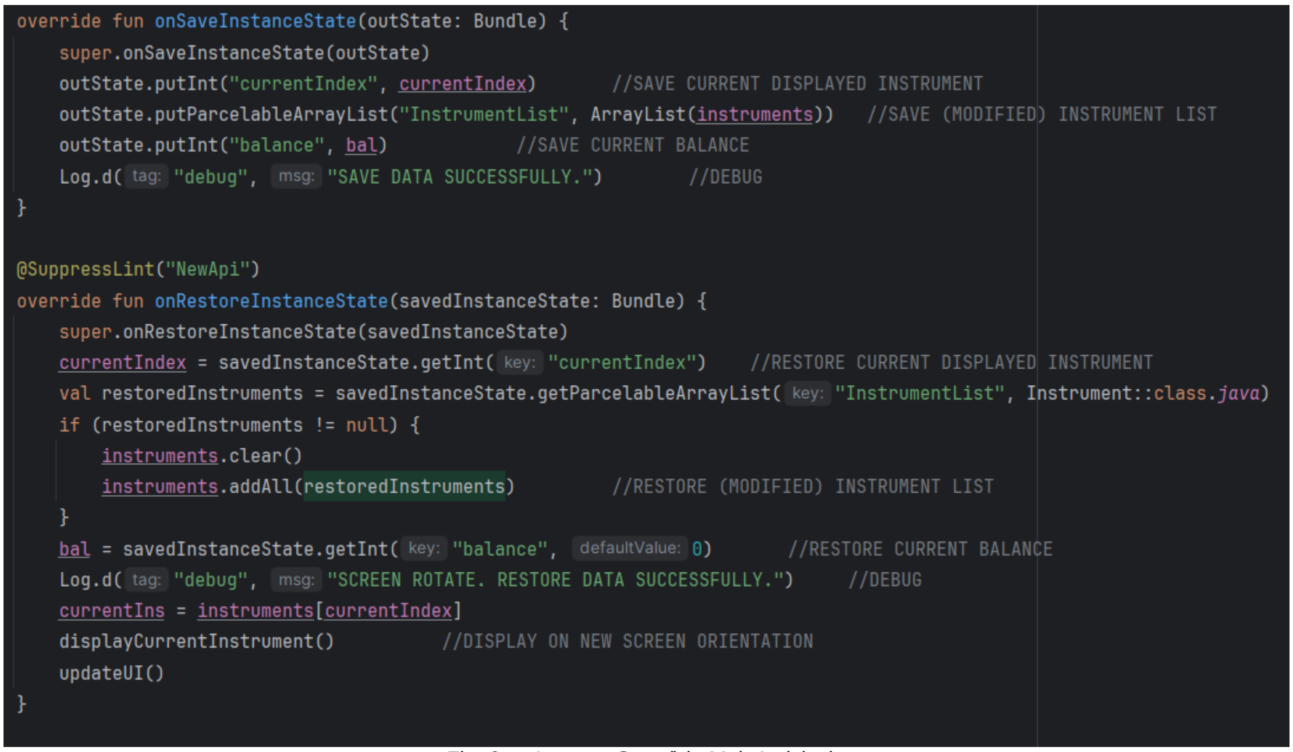
I used *registerForActivityResult()* in order to catch the results coming back from DetailsActivity. By using *when()* to compare the resultCode to whether if it is RESULT_OK: use *getParcelable()* and *getIntExtra()* to retrieve the balance after borrowing the instrument, and update its 'rented' bool attribute to True. This helps me to modify certain behaviors of the app using simple if else statements such as disabling the "Borrow" button so that the user could not borrow it again, or displaying a non-persistent message (Snackbar) to alert user that this equipment has been rented.

Parcelable objects are the most ideal when transferring data through Intent efficiently. They were built directly into the Android SDK which we used to build the project and is written and read in Parcel, which reduce runtime errors. For the assignment's requirement I've only used a few different fields (name, price, balance, year) in transitioning data, using Parcelable helps keeping it even more lightweight and avoid extra overhead, even more crucial on mobile devices.

Gap 3: Maintaining states across different orientations

Problem: All data and states upon flipping one's screen 90 degrees will be lost and refreshed, effectively resetting all progresses and purchases (as occurred in assignment 1)

Solution: Implementing *onSaveInstanceState()* and *onRestoreInstanceSave()* to save and restore crucial data once on the new screen.



```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putInt("currentIndex", currentIndex)       //SAVE CURRENT DISPLAYED INSTRUMENT
    outState.putParcelableArrayList("InstrumentList", ArrayList(instruments))   //SAVE (MODIFIED) INSTRUMENT LIST
    outState.putInt("balance", bal)            //SAVE CURRENT BALANCE
    Log.d( tag: "debug",  msg: "SAVE DATA SUCCESSFULLY.")        //DEBUG
}

@SuppressLint("NewApi")
override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)
    currentIndex = savedInstanceState.getInt( key: "currentIndex")    //RESTORE CURRENT DISPLAYED INSTRUMENT
    val restoredInstruments = savedInstanceState.getParcelableArrayList( key: "InstrumentList", Instrument::class.java)
    if (restoredInstruments != null) {
        instruments.clear()
        instruments.addAll(restoredInstruments)         //RESTORE (MODIFIED) INSTRUMENT LIST
    }
    bal = savedInstanceState.getInt( key: "balance",  defaultValue: 0)       //RESTORE CURRENT BALANCE
    Log.d( tag: "debug",  msg: "SCREEN ROTATE. RESTORE DATA SUCCESSFULLY.")     //DEBUG
    currentIns = instruments[currentIndex]
    displayCurrentInstrument()          //DISPLAY ON NEW SCREEN ORIENTATION
    updateUI()
}
```

Fig. 6. onInstanceState() in MainActivity.kt

Because once rotated, the *onCreate()* method will be called again on the new screen, thus re-initializing the default instrument list, that's why I cleared that list and repopulated it with the saved one before screen rotate, the rest is fetched as normal.

Open Issues and Recommendations

This section outlines any open issues, risks, and/or bugs, and highlights potential approaches for trying to address them in the future.

Open Issues:

1. The app only checks for "sufficient" credits, haven't checked for negative ones (edge cases).
2. Only a "Next" button to cycle through the lists, may create confusion for users due to not knowing how many instruments there are.
3. No "Undo" button, only accept if you misbook and move on =D.

Recommendations:

1. More error handling.
2. A small guiding text of “Item 1 out of 4” would be appreciated.
3. A “Refund” button (maybe, but outside of assignment’s scope).