NELSON MANDELA UNIVERSITY

Faculty of Science

WRPV302: Advanced Programming 3.2

Assignment 3: Due Tuesday, 16 September 2025

Department of Computing Sciences

Please read this document carefully and keep it in a safe place — it contains important information that you will need to refer to during the semester.

Possession of this document **does not** guarantee your registration (or acceptance) for this module, nor does it guarantee access to the departmental computer laboratories.

Summerstrand South Campus
Secretary: 041-5042247

Contents

General assignment information

This document contains the information that is needed to complete Assignment 1 for the WRPV302 module.

Be sure to submit your assignment in a zipped file on the Funda site before the due date and time. Late submissions will not be accepted. You are also required to *peer assess* assignments submitted by your peers for all the practical assignments.

Remember to complete *all* the Activities, as *all* of them count towards the Final Mark for this Module.

Note: Practical Assignments are **not** group Assignments, they are to be completed **individually**. Be sure to read the section on Academic Integrity in the Modules Guide. The **minimum** penalty for any acts of Plagiarism or Cheating is 0% for that specific assignment for **all** parties involved.

Note: Working individually means that *you* do the work. Making use of other people's code is **not** individual work; making use of code generated by ChatGPT (or anything else) is **not** individual work – *you* did not write the code. If detected, the minimum punishment is zero for the assignment for all involved (i.e., the one(s) copied from *and* the one(s) doing the copying).

Note: You **are** allowed to "plagiarise" lecture example code in assignments, i.e., you may extend/modify lecture example code to your heart's delight. A **maximum** of 5 lines[1] of code from *another* source for an *entire* task in an assignment is allowed. These 5 lines must be clearly marked as coming from elsewhere and a reference to the code must be provided.

Note: At 3rd year level, **all** programs are expected to compile and run. If a task does not compile without syntax errors, it will be given a zero!

General assignment procedure

Please read this entire section before doing anything:

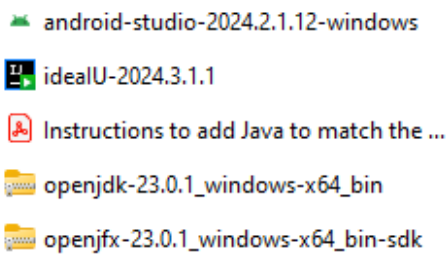
- Assignments are to be completed and submitted before the cut-off date. Submission is via the module's Funda page (https://funda.mandela.ac.za/).
- **Late submissions are not allowed and will result in an automatic mark of 0!** Do *not* attempt submission 5 minutes before the cut off time – *anything* can go wrong. Submit timeously.
- **Submit your solutions in a single zip file**. Please use proper naming conventions, document your code using comments and write neat, easy to read code. **If using Android Studio, IntelliJ and Java at home, *please* use the same versions and install them in the same locations as on the lab machines**. This makes transferring projects between computers easier and it will make the lives of your assessors easier if they do not need to struggle to get your projects compiling and running on their machines. This will probably result in better marks for you.

[1] Five lines of properly formatted code, i.e., 5 lines of code after the auto-format tool is used.

For this academic year, the following software versions are being used (**i.e., use these versions at home, even if newer versions exist**):

- Android Studio 2024.2.1.12
- IntelliJ 2024.3.1
- OpenJDK 23.0.1
- OpenJavaFX SDK 23.0.1

You may find the installation files and some instructions on the courses folder \\postgrad-new.mandela.ac.za\wrpv software:



Be sure to be connected to the VPN to access the above folder.

On a weekly basis you will be required to:

- *self-assess* your submission. This process will involve "marking" the previous assignment's solution that you submitted. Its intention is to have you critically evaluate your own work according to the rubric provided on the Funda site; and
- *peer assess* some of your peers (as well as your own work). This process will involve "marking" the previous assignment's solutions of some of your peers (you will be assigned new classmates to assess every time). Marking will be done using a rubric that will be provided to you on the Funda site. Marking your peers' work will be assessed and *will* contribute towards your final mark, so make sure you do it.

In other words, the assignment solution you submit will be marked and will contribute towards your final mark. In addition to this, *your marking* of a peer's solution will *also* be marked and will *also* contribute towards your final mark.

Your marking of a peer's assignment with be compared with the official marked one. The more accurate your marking[2], the higher **your** *assessment mark* will be.

It is recommended that the structure of your solution has a single project per week, with a single module per task. Use proper naming conventions[3].

Part of the assessment of your assignments *will* be on the presentation of your code. So, use proper/descriptive variable names, provide comments, document your methods, format your code appropriately[4], etc.

[2] i.e., did you mark correctly, just thumb sucked a mark, just gave high marks, etc.
[3] Use these naming conventions (http://www.oracle.com/technetwork/java/codeconventions-135099.html).
[4] This is a soft skill you need to learn for the workplace one day.

Note that you must use Java for all coding done in this module, not Kotlin!

Assignment schedule

The schedule of assignment submissions and peer assessments is given below:

Week & Date		Lecture: Monday 10:25-11:35 05 00 03 Practical: Wed/Thurs 15:45-18:15	
1	21-25 Jul	Course Administration **Lecture 1:** Introduction to Android	Start Assignment 1
2	28 Jul - 1 Aug	**Lecture 2:** Android views & activities	
3	04-08 Aug	**Lecture 3:** Android persistence (files & databases) & exceptions	Submit & Peer Assess Assignment 1
4	11-15 Aug	**Lecture 4:** Android multiple views	Start Assignment 2
5	18-22 Aug	**Lecture 5:** Android fragments	Term 3 assessment: Saturday, 23 August
6	25-29 Aug	**Lecture 6:** Collections	
7	01-05 Sep	**Lecture 7:** Multi-threading	Submit & Peer Assess Assignment 2 Start Assignment 3
RECESS 06-14 September			
8	15-19 Sep	**Lecture 8:** Threads & animations	Submit & Peer Assess Assignment 3 Start Assignment 4
9	22-26 Sep	**Lecture 9:** Networking I	Wed 24 Sep Heritage Day
10	29 Sep - 03 Oct	**Lecture 10:** Networking II	Submit & Peer Assess Assignment 4 Start Assignment 5
11	06-10 Oct	**Lecture 11:** Networking III	Term 4 assessment: Saturday, 11 October
12	13-17 Oct	**Lecture 12:** Multi-media	Submit & Peer Assess Assignment 5
13	20-24 Oct		Sick assessment: Friday, 24 October
14	27-31 Oct		
EXAM (examinable modules only) 04–26 Nov **Recess 27 Nov – 31 Dec** **RE-EXAM (examinable modules only) 15–23 Jan**			

* See changes highlighted with an orange border.

Assignment 3

Topics

- Views
- Multiple Activities
- Collections
- Regular Expressions

Task 1: Upgraded Sequence-Dice Game

In Assignment 1, you created very simple versions of the Sequence-Dice game. You are now required to update the Android version in a number of ways:

- Include a splash screen that displays the game's name and a suitable image for a few seconds, or until tapped, before closing and going to the main screen;
- The main screen should have options for starting a game, reading the rules and viewing a leader board;
- When starting a new game, the number of players in a game must be chosen, each player can enter their name, and the teams can choose their colour. Once the start button is clicked, the app switches over to the game play screen;
- The game play screen should now display the game board in a more visually appealing style than that of Task 1, i.e., it is now necessary to use images as well, and the die should look like die, instead of simply a number.
- Once a game is over, a report screen should be shown indicating which team won, as well as statistics about the game. These statistics should include how many rounds were played, how many tokens placed, how many tokens removed, and longest sequence of each team during play.
- The rules should be displayed on a separate screen that the user can browse through. A link should be provided, that when clicked on, will take the user to a YouTube video showing how to play the game. This must be done using an intent.
- The leader board should display the top 10 wins of players (based on their name), sorted in descending order. This information must persist between games being played.

Task 2: Budget Watcher

Financial institutions (such as ABSA, FNB, Woolworths Credit Card, etc.), as part of their customer services, often send SMS notifications of financial transactions that have occurred in any of their accounts to the customer's mobile device.

These text messages contain a lot of information, such as the date of the transaction, which account was involved, what kind of transaction it was (cash withdrawal/deposit, EFT, debit orders, deposits, payments, etc.), a description (which store was being paid, or a user entered string – in the case of an EFT), the amount of the transaction, as well as other information.

Obtaining data directly from a financial institution using web-services, etc. is complicated, involves data privacy laws, security issues, etc. However, SMS

messages received on a user's phone contain a lot of useful information that can be extracted and used, and can be from multiple financial institutions (for the same client).

The use-case for this task is a budget app, installed on the user's mobile device, that allows them to see how they are spending money, where, times of the month when spending happens, etc. This is done by analysing SMS messages that have been received, determining which are from financial institutions, and extracting information from these messages. This information is then aggregated and analysed in some way, and the results presented to the user. The advantage of this app is that as the user spends or receives money, their expenditure is updated, so that they know how things stand (in more or less real-time – depending on how quickly the notifications come through).

Below are *some* example SMS messages from *actual* financial institutions, specifically ABSA and the Woolworths Credit Card division (underwritten by ABSA). A brief explanation is provided of each:

ABSA

```
Absa: CHEQ1234, Pur, 02/12/23 SETTLEMENT/C - AIRTIME DEBIT, MTN: 0264851994, R-149.00,
Available R1,000.00. Help 0860008600; ACCID 001

Absa: CHEQ1234, Wthdr, 15/04/21 RASC PE - ATM WITHDRAWAL, R-500.00, Available R500.00.
Help 0860008600; ACCID 001

Absa: CHEQ1234, Dep, 29/01/24 SETTLEMENT/C - MY PLACE OF WORK, R20,000.00, Available
R21,000.00. Help 0860008600; ACCID 001

Absa: CHEQ1234, Pmnt, 02/12/23 SETTLEMENT/C - DIGITAL PAYMENT DT, MY RENTAL ACCOMODATION,
R-10,000.00, Available R11,000.00. Help 0860008600; ACCID 001

Absa: CHEQ1234, Sch t, 01/12/23 SETTLEMENT/C - ACB DEBIT:EXTERNAL, ATLAS SEC 12345
NETCASH, R-865.00, Available R8,500.00. Help 0860008600; ACCID 001
```

Account: this is the account on which the transaction was performed (in the examples above `CHEQ1234`). It is possible that a client has multiple accounts at the same financial institution, e.g., a current account into which their salary is paid, and typical payments are made from, and other accounts such as saving or investment accounts.

Transaction Type: this is the type of transaction. In the examples above:

- Purchase (`Pur`): in this case, it refers to purchases made from via ABSA (using the website or mobile app), for things such as airtime, prepaid electricity, etc.
- Withdrawal (`Wthdr`): cash was withdrawn from the account at an ATM (or some other teller).
- Deposit (`Dep`): money was deposited *into* this account.
- Payment (`Pmnt`): money was paid electronically *out of* this account (EFT).
- Debit Order (`Sch t`): money that is scheduled to be paid electronically on a specific day each money, e.g., insurance, security company, gym membership, etc.

Date of Transaction: the date on which the transaction occurred, in DD/MM/YY format.

Description: a string describing the transaction, typically refers to a store or company name, but could be text entered by someone paying the client via an EFT.

Amount: the amount of the transaction. Note that some transactions are positive, others negative.

In these messages, extra information was provided, specifically a number to contact if fraud is suspected, the current balance of the account after the transaction[5] and a text identifier for the account name.

Woolworths Financial Services

```
WFS : CCRD1013, Pur, 14/07/21 AUTHORIZATION, SeattleCoff Walmer Eastern, R72.00, Total
Avail Bal R1,000.00. Help 0861502005; CCID 004

WFS : CCRD1000, Transf. 09/04/21 INTERNAL FUNDS TRANSFER, ***1018, R20,000.00, Total
Avail Bal R21,000.00. Help 0861502005; CCID 004
```

The information provided in the SMS messages from the credit card division is very similar to that of the banking division. The main difference is a purchase indicates that a purchase was made with a credit card (and the amount owed becomes more), while a transfer indicates that money was paid in to the credit card (to make the amount owed less).

Different institutions will send SMS messages with different formats, but typically with similar information.

For this task, you are required to write two separate apps. One will be an **SMS Generator**, the other **Budget Watcher** (SMS message processor).

The **SMS Generator** app will generate *and send* random SMS messages to a mobile device, specifically an emulator. The messages should be in the format of the messages shown above, as well as some general messages (such as "You've won 1,000,000USD!!!"[6]). The user should be able to specify a date range, as well as how many messages should be sent in a batch. Additionally, the user should be able to enter and send a specific SMS as well. Note that the contact number of the emulators is 5554 for the first emulator started, 5556 for the second, 5558 for the third, etc.

The **Budget Watcher** app has the following functionality:

- It will extract all the SMS messages on the mobile device and process them;
- Processing SMS messages involves:
    - *filtering* messages and discarding messages that are not from a financial institution (do not match the message patterns above);
    - *extracting* information about the transaction from the text message (i.e., transaction type, date of transaction, description, amount, etc.);
    - *tagging* a transaction based on the information and type of transaction (e.g., based on the description, the transaction was a purchase for food, accommodation, transport, clothes, tuition, etc. or some other way of classifying transactions). A transaction can have multiple tags, e.g., expense, food, restaurant. So, a purchase at Steers would be tagged as an expense (you paid money), food (you ate it) and restaurant (you didn't make the food yourself). This will allow the user to see all expenses (regardless of type), all food purchases (groceries or restaurant) or amount of money spent eating out;

[5] Note that the balance might be slightly different than what you expect if you only work using the transaction SMSs, because financial institutions do not *necessarily* send SMS notifications for *all* transactions. For example, ABSA does not send SMS notifications when bank charges have been made, the amount is simply deducted from the balance. Similarly, for interest earned on an investment, a notification is not received.

[6] Maybe ask ChatGPT or Claude to generate some random SMS messages?

- Aggregates tagged transactions and displays this information to the user. Aggregating involves using a date range and a collection of tags, then all transactions that occurred within the given date range *and* contain *all* the tags are collected together and the amounts totalled. The aggregated values are then displayed to the user as a recycler list[7].

Appropriate use of the collection classes, the fluid API and the regular expression classes **must** be made for this task. Failing to do so will result in a zero for this task!

The message filtering, information extraction, and tagging **must not** be hard-coded. It **must** be possible to add new filters, fields, and tags at run-time (by the user adding new ones), or loaded from a file (although this functionality is not required for *this* task). To accomplish this, you must make good use of the regular expression classes discussed. There **may not** be a separate class for each type of message, etc. - you need to think of a generic way in which to do this, the proper classes to use, and appropriate algorithms to achieve the desired effect.

An appropriate UI should be created to allow the user to:

- initiate the process;
- see all filtered messages and their extracted information and tags;
- specify date ranges or choose per week, month or year for each of the aggregations and view the aggregations; and
- inspect aggregations to view the transactions *in* the aggregation, e.g., if the food purchases for March 2024 amounted to R10,000, you could dig deeper into the aggregation to see the *exact* transactions tagged as food during that period.

It is up to you whether you wish to use a single or multiple Activities for this task.

[7] Or if you're feeling adventurous, as a graph (bar-graph, scatter plot, etc.) There are libraries, such as this one here and here, that you may investigate.

Rubric

Below is the rubric, available on Funda, that will be used when assessing this Assignment:

Criteria	Marks	Option
Task 1: Upgraded Sequence-Dice Game (Activities) You were to implement multiple Activities to make the game created in Assignment 1 seem more complete. The minimum Activities required are: 1. a splash screen; 2. main menu screen; 3. game rules; 4. game play screen (i.e., what was done in Assignment 1); 5. end of game report screen; and 6. the leader board screen.	0	Not implemented, or does not compile.
	1	1-2 of the screens implemented.
	2	3-4 of the screens implemented.
	3	All of the screens implemented.
Task 1: Upgraded Sequence-Dice Game (Functionality) Each of the Activities was required to have specific functionality: 1. a splash screen – display for a few seconds (or until tapped) before next displayed; 2. main menu screen – directs user to different Activities; 3. game rules – browse rules; 4. game play screen – play the game, now including images; 5. end of game report screen – display game play statistics; and 6. the leader board screen – display number of wins by top ten players in descending order. Information to persist between game plays.	0	Not implemented, or does not compile.
	1	Less than 30% of functionality implemented.
	2	Less than 60% of functionality implemented.
	3	All functionality implemented.
Task 2: Budget Watcher (SMS Generator) The SMS Generator was supposed to generate and send SMS messages of different kinds to an emulated mobile device. The user was to specify a date range and number of financial SMS messages to generate, and then send them. The user could type and send an SMS as well.	0	Not implemented, or does not compile.
	1	Less than 30% of functionality implemented.
	2	Less than 60% of functionality implemented.
	3	All functionality implemented.
Task 2: Budget Watcher (SMS Processor - functionality) The SMS Processor part of the task was required to have the following functionality: 1. extract all SMS messages on the device; 2. filter out uninteresting messages (not from a financial institution); 3. process interesting messages, extracting information from them; 4. tagging messages based on the information extracted; 5. aggregating messages based on date ranges and tags contained; 6. display the information extracted in #2 in a list; and 7. display the aggregations from #5 in a recycler view. Clicking on an aggregation displays the transactions in the aggregation (similar to #6).	0	Not implemented, or does not compile.
	3	Less than 30% of functionality implemented.
	6	Less than 60% of functionality implemented.
	9	All functionality implemented.
Task 2: Budget Watcher (SMS Processor - implementation)	0	Did not adhere to any of the requirements.
	1	Adhered to some of the requirements.

For this task, you were *explicitly* asked to make use of the Java Collection framework, its fluid API, as well as the regular expression classes to make the app flexible.	2	Adhered to all of the requirements.