NELSON MANDELA UNIVERSITY

Faculty of Science

WRPV302: Advanced Programming 3.2

Assignment 5: Due Tuesday, 14 October 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 this Assignment 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:

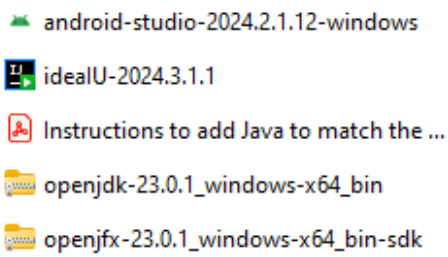
- Assignments are to be completed and submitted before the cut-off date. Submission is via the module's Funda page (https://funda.mandela.ac.za/).
- **Late submissions are not allowed and will result in an automatic mark of 0!** Do *not* attempt submission 5 minutes before the cut off time – *anything* 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:** Threads & animations	Submit & Peer Assess Assignment 2 Start Assignment 3
RECESS 06-14 September			
8	15-19 Sep	**Lecture 8:** Multi-threading	Submit Assignment 3 Start Assignment 4
9	22-26 Sep	**Lecture 9:** Networking I	Peer Assess Assignment 3 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**			

Assignment 5

Topics

- Networking
- Threads

Task 1: Networked Pub/Sub Broker

In Assignment 2, you were asked to implement a Publish/Subscribe Broker pattern where Subscribers subscribe to receive messages about a specific topic via the Broker. Publishers can publish messages about specific topics, which are then forwarded via the Broker to all subscribers subscribed to those topics. Subscribers can unsubscribe from a topic at any time and will then not receive any further notifications.

A very important aspect of this was that Publishers and Subscribers did not know about each other explicitly. Instead, all communication was handled via a single, shared Broker object. In your assignments, the Broker, Publishers and Subscribers were all within the *same* application and on the *same* device.

Topic Name	Subscribers
a	s_1, s_3
b	s_2
c	s_2, s_3

For this task, you are to create a *thread-safe networked* version of the Publish/Subscribe Broker. In this scenario, the Broker is hosted on a Server and the Publishers and Subscribers run in different client applications and possibly different devices. The Publishers and Subscribers communicate with each other indirectly via the Broker. Communication with the Broker is handled using a TCP/IP connection[5].

You have been provided with an implementation of a Publish/Subscribe Broker class and associated interface where everything was running within the same application in the "Assignment 4 Files" folder on Funda. Modify this implementation, or your own from Assignment 2, to allow for a networked version of the pattern.

Note that your implementation *must not* interfere with the normal execution of an application, e.g., it must not block an Android app's UI thread. Also, a client should not "block" while waiting for publications, it should be possible to publish anything at any time (and receive anything at any time).

[5] Networked messages in this task are very similar, in many ways, to the concepts of web services, remote procedure calls, etc. used elsewhere.

For this implementation, when a Client connects to the Broker Server, it is allocated a unique integer ID. This ID is communicated with the Client. The ID is used as the "publisher" in a message.

The Server must be a Java console-based application. The Client-side classes must be able to run in either regular Java applications or Android apps, i.e., don't use any Android only or Java only classes.

Make appropriate use of the Java Collection Framework, Lambdas, threads, sockets, etc. in your solution.

Task 2: Multi-platform Networked Sequence-Dice

In Assignment 1, you were required to implement an Android version of the game Sequence-Dice.

For this task, you will be required to extend the app to create a networked version of Sequence-Dice, such that:

- there is a Java console game server that manages networked playing of the game between *multiple* groups of 2-4 players, *at the same time*; and
- connecting to the server will be two *types* of clients, one written as a Java console-menu app (desktop) and the other in Android (mobile). For a particular game pairing, clients can be any combination of the two types.

When a player wishes to play a game, they will click the start button on their app, which will then connect to the Java server. The Java server will group players and start a game once there are enough players (and will go back to creating more games – i.e., use a thread per game grouping). For each game, the server will co-ordinate whose turn it is, the placing of tokens, and game results, etc. Once a game is over, the connection to the server is terminated. When creating a new game, the server will randomly determine how many players there will be (2-4).

Make use of the networked pub/sub broker written in Task 1 to implement the networked communication.

The client apps will respond to messages from the server, updating the user interface. Commands[6] issued by players via the UI will be sent to the server, who will broadcast the Commands to the playing clients, who will effect the Command on each player's Sequence-Dice Model (which will update the UI, etc.). Sufficient information needs to be sent with Commands so that the internal state of the Model of each player is *identical*, e.g., the placements of tokens, etc.

Both the Java console-menu and Android versions of the game that have already been written need to be updated to now allow for networked communication.

[6] You may use the networked pub/sub broker created in Task 1 if you wish to.

Rubric

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

Criteria	Marks	Option
Task 1: Pub-Sub Server/Client Implementation Was a thread safe pub-sub broker implemented? This implementation should have addressed issues like - concurrent access (simultaneous publishing and subscribing actions by the server), and - handling of multiple client connections.	0	Not implemented, or does not compile.
	2	Non-thread safe Pub-Sub server implemented and could only communicate with either one publisher and/or one subscribe.
	4	Thread-safe pub-sub implemented but could only communicate with one publisher/subscriber at a time.
	6	Thread-safe pub-sub server implemented and could communicate with multiple clients simultaneously.
Task 1: Pub-Sub Server/Client Implementation Dependencies Was the thread-safe pub-sub implementation dependent on any JavaFX / Android-specific libraries?	0	Not implemented, or does not compile.
	1	Some JavaFX / Android specific libraries were used.
	2	No specific to Android/JavaFX libraries or classes.
	4	No specific to Android/JavaFX libraries or classes and JCF / lambdas were used.
Task 1: Pub-Sub Server Implementation Did the pub-sub server allow for unique identification of clients via an "ID" and was this ID used to identify publishers/subscribers?	0	Not implemented, or does not compile.
	1	IDs were implemented but did not work correctly.
	2	IDs were implemented but not used.
	3	Unique IDs were implemented and correctly identified clients.
Task 1: Pub-Sub Client A client implementation was required to interact with the thread-safe pub-sub broker. The client implementation required: • the client should be able to subscribe to or publish to any specific topic. • the client itself could be a publisher and subscriber of the same topic. • the client pub-sub should also run off of the main (UI) thread running as a non-blocking operation.	0	Not implemented, or does not compile.
	2	Client could either subscribe/publish but not both.
	4	Client could subscribe/publish but to the same topic or was implemented with a blocking style implementation.

	6	Client could publish and subscribe (to the same topic) and was implemented in a non-blocking style ensuring continuous user interaction while waiting for publish messages.
Task 2: Sequence-Dice Server You were required to write a server that: • was a **Java console** application; • made use of multiple threads to manage the player grouping, and then game play; and • handled communication with clients on different platforms (in this case desktop & mobile) – by using a common communication protocol. The server should have functionality similar to: • a single main grouping thread, that accepts connections, then waits for 2-4 connections (determined randomly by the server), before starting a game play thread with the players; • a per game play thread, that runs for the duration of a game, passing commands between the players. The server is used as the hub through which communication happens (and for debugging purposes should probably display what is happening at all times). Provision for network connections being lost must be made.	0	Not done, or does not compile.
	2	Groups players, game play not functional.
	4	Groups players, one game at a time.
	6	Groups players, multiple simultaneous games, fully threaded, handles connections being lost.
Task 2: Java Console-Menu Sequence-Dice Client The implementation from Task 1 (or one obtained from a peer), modified to be networked. This would include: • a way to connect to server; • indication of waiting for an opponent; • sending user commands to the server; • updating internal state based on network commands received; and • disconnecting from the server once a game has completed. Handles lost connections gracefully.	0	Not done, or does not compile.
	2	Can play the game, but not networked.
	4	Networked, but not all of the functionality working correctly.
	6	Fully networked, works well, handles lost connections.
Task 2: Android Sequence-Dice Client The Android implementation, modified to be networked. This would include: • a way to connect to server; • indication of waiting for an opponent; • sending user commands to server; • updating internal state based on network commands received; and • disconnecting from the server once a game has completed. Handles lost connections gracefully.	0	Not done, or does not compile.
	2	Can play the game, but not networked.
	4	Networked, but not all of the functionality working correctly.
	6	Fully networked, works well, handles lost connections.