NELSON MANDELA UNIVERSITY

Faculty of Science

WRPV302: Advanced Programming 3.2

Assignment 4: Due Tuesday, 30 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.

Contents

General assignment information 4
General assignment procedure 4
Assignment schedule 6
Assignment 4 7
    Topics 7
    Instructions 7
    Task 1: Sudoku Generator 7
    Task 2: Threaded Generation of Sudoku Boards 9
    Rubric 10

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:

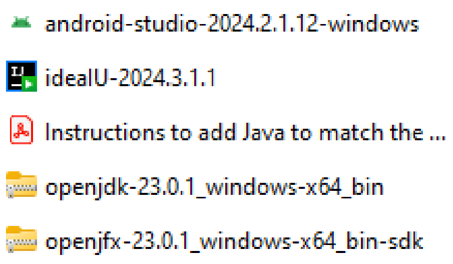
- Assignments are to be completed and submitted before the cut-off date. Submission is via the module's Funda page (https://funda.mandela.ac.za/).
- **Late submissions are not allowed and will result in an automatic mark of 0!** Do *not* attempt submission 5 minutes before the cut off time – *anything* 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:



android-studio-2024.2.1.12-windows
idealU-2024.3.1.1
Instructions to add Java to match the ...
openjdk-23.0.1_windows-x64_bin
openjfx-23.0.1_windows-x64_bin-sdk

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**			

* See changes highlighted with an orange border.

Assignment 4

Topics

- Collections
- Threads

Instructions

You may implement this as a Java console app or an Android app, although a Java app may be easier to debug.

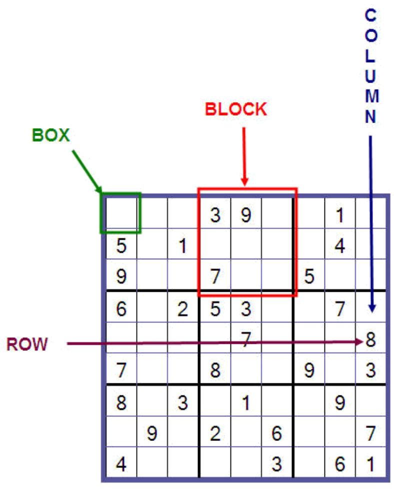
Task 1: Sudoku Generator

This task was an exam question in previous years.

For this task, you are required to write a Sudoku generator using the *Wave Collapse Function* algorithm. You *must* make use of the streaming API wherever possible.

Hint: consider the data structure(s) you use carefully.

The Rules of Sudoku



The classic Sudoku game involves a 9x9 grid of 81 boxes. The grid is divided into nine blocks, each containing nine boxes.

The rules of the game are simple:

- each of the nine blocks must contain all the numbers 1-9 within its boxes; and
- each number can only appear once in a row, column or block.

One way to randomly generate a *valid* Sudoku grid is using the *Wave Function Collapse* algorithm[5] from Quantum Mechanics. This algorithm can be used to generate many things, including a Sudoku grid.

The algorithm works by initially having each box contain a set of *all* possible valid values, e.g., 1, 2, 3, 4, 5, 6, 7, 8, 9. So initially the grid contains all possible Sudoku grids that can *ever* be generated.

Then using a process of elimination, the possible Sudoku grids are reduced until finally a *single* particular Sudoku board is generated.

The process of reducing the possibilities works as follows:

- pick a box, b, with the least number of possibilities remaining[6], then randomly pick one of the possibilities, p, in that box. This becomes **fact** for box b (i.e., there are no other possibilities for it). Note that a box with one possibility is *not*

[5] Here is another example of the Wave Collapse Function being used. Also includes a nice description of the general algorithm used for creating Wedding seating charts: https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/

[6] If there are multiple boxes with the same number of possibilities remaining, randomly pick one of them.

a fact until it is selected as a fact (with all the subsequent checks taking place as well);
- remove p from the possibilities of all boxes in the same row, column and block as b.

Repeat this process until all the boxes contain only facts.

At this stage, a valid Sudoku grid should have been generated, where all the boxes have a single value (fact) and none of the rules have been violated.

The table below shows an example of what a grid could look like after a few reduction steps. Box A4 contains all possibilities (1-9). Boxes B2, C2 and D3 have facts (2, 3 and 6 respectively). Based on these facts, the possibilities of other boxes have been reduced. For example, box D2 has possibilities 1, 4, 5, 7, 8 and 9. This is because the possibilities 2 and 3 were removed due to being in the same row as boxes B2 and C2 and the possibility of 6 being removed because it is in the same block (and column) as D3.

	A	B	C	D	E	F
1	1,4,5,6,7,8,9	1,4,5,6,7,8,9	1,4,5,6,7,8,9	1,2,3,4,5,7,8,9	1,2,3,4,5,7,8,9	1,2,3,4,5,7,8,9
2	1,4,5,6,7,8,9	2	3	1,4,5,7,8,9	1,4,5,7,8,9	1,4,5,7,8,9
3	1,4,5,6,7,8,9	1,4,5,6,7,8,9	1,4,5,6,7,8,9	6	1,2,3,4,5,7,8,9	1,2,3,4,5,7,8,9
4	1,2,3,4,5,6,7,8,9	1,3,4,5,6,7,8,9	1,2,4,5,6,7,8,9	1,2,3,4,5,7,8,9	1,2,3,4,5,6,7,8,9	1,2,3,4,5,6,7,8,9

When picking a new box, one of boxes D2, E2 or F2 will be selected as they all have 6 possibilities left. If box E2 was selected, then one of the possibilities 1, 4, 5, 7, 8 or 9 would be randomly chosen. If, for example, 9 was chosen, then E2 would hold the fact 9 and all the 9s would be removed from the possibilities of the boxes in row 2, column E and the block in which E2 resides namely (D1, E1, F1, D2, F2, E3 and F3).

Should there *ever* exist a box with 0 possibilities, then the grid is invalid[7]. Display an error message and try again.

Write two classes named `SudokuGenerator` and `SudokuGrid`.

The `SudokuGrid` class represents a board and the values (the final facts) placed in it. It must provide functionality to set, retrieve and clear box values, as well as a method to determine whether placing a value in a given box is valid, invalid due to row conflict, invalid due to column conflict or invalid due to block conflict. Note that it is possible to be invalid for any combination of the three reasons (e.g., invalid due to row conflict AND invalid due to column conflict).

The `SudokuGenerator` class is responsible for generating a valid complete `SudokuGrid` using the Wave Function Collapse algorithm described above.

You must make appropriate use of the Java Collection Framework (collections, algorithms, streaming API, etc.) when completing this task.

[7] Not 100% sure if this case *can* ever occur though, but just in case.

Demonstrate that your implementation works correctly by displaying generated boards after the user presses “enter” and display the number of invalid boxes on the board (using the method you wrote previously).

Task 2: Threaded Generation of Sudoku Boards

In Task 1, you wrote an algorithm to generate Sudoku game boards (full solutions). Now you need to write a program that uses threads to do the following:

- generate 10,000[8] *unique* Sudoku game boards, allocate each a unique number and save these to a file – as quickly as possible.

Game boards are numbered 0, 1, 2, 3… in the order that they are generated. A game board is saved in a text file named 0.txt 1.txt, 2.txt, etc., which simply contains all the numbers in each box of the game board.

A game board is considered the same regardless of how it is rotated, flipped along the rows, flipped along the columns, rotated 90°, 180° or 270°, or any combination therefore[9]. So, a game board is considered the *same* as a mirrored or rotated (or mirrored and rotated) version of itself, etc.

You will need to have threads to generate game boards, some to check for uniqueness and another for saving unique game boards to file.

The correct choices in data structures and algorithms used are *very* important in this task.

[8] Or some other large number…

[9] Note that the representation of your data can go a *long* way to how efficiently the comparisons, rotations, mirroring, etc. can be done. Think about this, unless you feel like sitting for months waiting to get all 50,000 game boards.

Rubric

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

Criteria	Marks	Option
Task 1: Sudoku Grid Class For the grid class, there was supposed to be: 1. a suitable data structure used (from the JCF); 2. methods to clear all boxes, set a box value and query a box value; 3. a method to query whether placing a value in a box is valid or not. The method should be able to return whether it is a valid or invalid option, as well as the reason(s).	0	Not done, or does not compile.
	1	At least 50% of expected functionality.
	2	At least 75% of expected functionality.
	3	All functionality implemented and correct.
Task 1: Sudoku Generator Class For the generator class, there was supposed to be: 1. an initial set up, in which all possibilities exist; 2. the WCF that gradually determines values for all the boxes (should not be more than 81 iterations needed); 3. stops when a valid board has been generated, or an impossible board is generated. Appropriate use of the JCF was to be made. The streaming API should be used where possible.	0	Not done, or does not compile.
	1	At least 50% of expected functionality.
	2	At least 75% of expected functionality.
	3	All functionality implemented and correct. The streaming API was used where possible.
Task 1: Demonstration A simple program to generate and display valid generated boards.	0	Not done, or does not compile.
	1	Done and runs correctly.
Task 2: Threaded Sudoku Board Generation A threaded approach to generating multiple boards was required. Each thread should generate a board and once done, make it available for checking in another thread.	0	Not done, or does not compile.
	1	Generated boards, but all done in a single thread.
	2	Multiple threads generating boards. Boards passed to checking thread.
Task 2: Threaded Sudoku Board Uniqueness Checking Once a board has been generated, it needs to be checked for uniqueness. This requires: • all boards that have been generated to be stored; • a method (threaded?) to check a new board against all existing boards; • if no matches are found, store the board. New boards should be passed to a saving thread. The matching method should consider the rotation (0, 90, 180, 270 degrees) and mirror row/column variations of the boards.	0	Not done, or does not compile.
	1	Did all of the functionality on the Main thread.
	2	Did at least 75% of the functionality on the separate thread.

The data structure chosen is very important for this. Using a simple string or integer array are examples of really simple, efficient examples that could be used.	3	Did all of the functionality on the separate thread. Used efficient data structures.
Task 2: Threaded Saving New, unique boards were to be saved.	0	Not done, or does not compile.
	1	Done, on Main thread.
	2	Done, on separate thread.
Task 2: Passing boards between threads An efficient way of passing boards between the different threads was required. One such way is to use two thread-safe queues: • one for newly created boards - to be checked; and • one for boards to be saved. Other mechanisms could be used too, but they need to be thread-safe, i.e., locks might be needed to prevent race conditions.	0	Not done, or does not compile.
	1	Suitable mechanism used.