UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS226 - Concurrent systems

# Assignment 2 Specifications - Concurrent Testing Framework

Release date: 07-10-2024 at 06:00
Due date: 18-10-2024 at 23:59
(Submission will remain open until 20-10-2024 23:59 for technical difficulties)
Total marks: 50

## Contents

## 1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.

## 2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent), and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding.** Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

## 3 Outcomes

On completion of this practical, you will have gained experience with the following:

- Synchronisation: Course-Grained, Fine-Grained and Optimistic

## 4 Introduction

### 4.1 Testing Concurrently

Testing is an essential part of software development, ensuring that code behaves as expected under various conditions. As projects grow, the number of tests increases, resulting in longer testing times. This can slow down development and impede the feedback cycle. One way to mitigate this issue is by running tests concurrently using a multithreaded testing framework. This approach allows multiple tests to be executed in parallel, significantly reducing overall testing time.

Running tests concurrently can be particularly effective in environments where tests are not dependent on shared resources or where resources can be efficiently isolated or mocked. By leveraging multi-core processors and taking advantage of idle CPU cycles, concurrent testing can dramatically speed up the process of running comprehensive test suites. It also enables quicker detection of failures and performance bottlenecks, which can be critical for agile development practices that rely on continuous integration and delivery pipelines. Furthermore, it allows teams to more readily embrace test-driven development (TDD) practices by reducing the time it takes to run tests after making small changes to the code.

In some testing scenarios, certain tests may need to be run in a specific sequence because they are dependent on the outcomes or side effects of earlier tests. For example, a test that verifies the deletion functionality in a CRUD application may need to be preceded by tests that handle creation and update functionalities to ensure that there exists data to delete. This dependency necessitates a testing framework that can manage and enforce the ordering of test execution.

A well-designed testing framework should provide capabilities to explicitly specify the order in which tests are executed. This can be achieved through annotations. The ability to define dependencies between tests helps in constructing a reliable and coherent testing process that accurately reflects the workflow of the application.

## 5 Tasks

You are tasked with implementing a multi-threaded testing framework. This will involve three main steps. First, you will create a thread pool to which you can submit Methods. Next, you will create annotations. These will annotate methods which are "Tests" and will give information as to the order of tests. Finally, you will create a `TestRunner` which will use the thread pool and the annotations to run tests as concurrently and print the results to the console.
After you have implemented the framework, run all your tests sequentially and then run them using the framework. Measure the time taken in each case and compare.

### 5.1 Creating a Thread Pool

To handle the concurrent execution of tests, you will implement a thread pool. This pool manages a fixed number of threads that wait for test methods to be submitted for execution. The Java `ExecutorService` interface is an example of such a pool. This service can be instantiated using

`Executors.newFixedThreadPool(int n)` where `n` is the number of threads in the pool. Methods are submitted to the pool using the `submit()` method, which schedules them for execution as threads become available.

You must implement your own version of this using the knowledge that you have gained in the module. See `https://en.wikipedia.org/wiki/Thread_pool` for more information on thread pools;

## 5.2 Defining Annotations for Test Methods

Annotations in Java provide a way to add metadata to method declarations. For the testing framework, one annotation that must be created is the `@Test` annotation. The `@Test` annotation identifies which methods are test methods. These methods should be run by the `TestRunner`. You should also implement one or more annotations to help determine if any tests need to be executed in a particular order. Java's reflection API should be used to discover methods annotated with these annotations and to read the values to prioritise test execution.

An example of annotation use has been provided.

## 5.3 Implementing the TestRunner

The `TestRunner` class is responsible for orchestrating the test execution process using the thread pool and annotations. It retrieves all methods marked with the `@Test` annotation, orders them based on the other provided annotation(s) and submits them to the thread pool. The results of the test executions are collected and printed to the console. You must print at least the following:

- Total time taken
- Number of tests passed
- Number of tests failed
- Number of tests skipped (tests should be skipped if a test they depend on fails or is skipped)

You may print more.

## 5.4 Performance Measurement and Comparison

To evaluate the performance of the multi-threaded framework you must create a set of tests (make sure you create an appropriate number of tests, too few will not be worth the overhead). First run your tests sequentially in a single-threaded manner and then use the multi-threaded framework. Measure the execution time for each approach using system time stamps at the start and end of the runs. Compare the times to determine the effectiveness of the multi-threaded approach in reducing the total execution time for all tests. This comparison should help to understand the benefits and potential overheads of using concurrency in test execution.

# 6 Mark Breakdown

- **Implementing the ThreadPool** - 15 marks
  - Design and implementation of a custom thread pool to manage threads.
  - Correct handling of thread life cycles and task submission.
- **Creating and Handling Annotations** - 15 marks

  - Correct implementation of the `@Test` annotation.
  - Implementation of additional annotations for ordering or prioritizing tests.
  - Proper use of reflection to detect and manage test methods.

- **TestRunner Implementation** - 15 marks
  - Correct scheduling and execution of test methods using the thread pool.
  - Accurate collection and reporting of test outcomes (passed, failed, skipped).
  - Efficiency in test execution ordering and dependency management (think of a good data structure).

- **Performance Measurement and Comparison** - 5 marks
  - Correct measurement of execution times for both single-threaded and multi-threaded runs.
  - Analysis and comparison of performance metrics.

- **Total - 50 marks**

## 7 Upload checklist

Upload all the files, including your report, that you need for your demo in a single archive.
**NB: Submit to the ClickUp Module, there will be no FF submission.**

## 8 Allowed libraries

These libraries will be allowed but you must still do the given tasks with your own code (i.e you may not use the libraries to complete the task for you)

- `import java.util.*`

## 9 Submission

You need to submit your source files on the ClickUp module website. Place all the necessary files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

Upload your archive to the appropriate practical on the ClickUp website well before the deadline.
**No late submissions will be accepted!**