Software Design 236700 — Assignment 1

Teaching Assistant: Dor Brekhman <brekhman.d@cs.technion.ac.il>

April 28, 2022

1 General remarks

- For any questions that are not personal please use Piazza: `piazza.com/technion.ac.il/spring2022/236700`.
  If you want to e-mail the TA please include "software design hw1" and your name in the subject of the e-mail.
  You are REQUIRED to follow the piazza for announcements.
- Please think of the environment and *do not* print out this document!
- The assignments will be presented in English for the benefit of our international students. If you have any troubles with the language used in the document, or are concerned about ambiguities or misunderstandings, please contact the TA.
- We encourage you to read all relevant documents & files from the course websites before starting work on this assignment.
- Submission is electronic via the course website.
- You *must* work in pairs. The number of team-members is two for each team. no more and no less.
- You are allowed to switch homework partners, simply correct `README.md` .
- The course staff are most familiar with IntelliJ, therefore will only able to provide technical support for that environment. However, we will still attempt to assist you if you choose to forgo an IDE and use only the command-line tools.
- Any updates and changes to the assignment will appear *in this document*, and will be tracked in section A on page 12. You should occasionally refresh this document from the course website, and be sure to go through it before your final submission. Also note the date shown at the top of this page.
- In this assignment we will use dependency injection with Guice.

Contents

2 Introduction: SifriTaub™

You've been hired to make a new version of the old CS Faculty Library book lending system.
We will allow users to borrow books and wait for books to become available.
During the semester, we will focus on designing and implementing a book lending system, which we call SifriTaub™, the new CS Faculty Library book lending system.

3 General Architecture

Note: This section may prove somewhat confusing, so make sure you read it all and understand what's going on. Then, a deep dive into the code should clear most things out.

We will be developing an *application* that receives input and outputs from *users* of the system, and uses *storage* to persist information across runs. There are four basic types of operations, summarized with the initialism *CRUD*: *C*reate, *R*ead, *U*pdate, *D*elete.

Create operations Commands of this type load new data, from an external source, into the application's data store, which will be used by the other types of commands. The data must be saved in a *persistent* fashion using the external library (see section 3.1) provided with the assignment. These operations must be reasonably efficient, but it is acceptable if they are slightly slower (see section 3.3.4 and section 4.3). There will be a generous time-limit for these, which will be calculated for the entire set of create operations (i.e., amortized time).

Read operations These operations query the data from the data-store, returning information, possibly performing a calculation based on the raw data. These operations need to be efficient—there will be a time-limit on each operation of this kind (i.e., constant time).

Update operations These operations read data from the data-store, transform it somehow, then write it to an existing location, replacing what was there before. The "shape", or number of entries, of the data store is unchanged. There will be a time-limit on each operation of this kind (i.e., constant time).

Delete operations An action to remove some data from the data-store. There will be a generous time-limit for these, which will be calculated for the entire set of create & delete operations (i.e., amortized time).

In our testing, the various operations will be interleaved (e.g., you can have create operations after read operations).

3.1 Primitive Storage

An external library is provided (see section 3.3.4 for how) with this assignment, providing a very basic abstraction over a general database or file-system. You must use this interface *exclusively*, and not any other persistence[1] mechanisms. **After create operations, the entire application may be restarted, and read operations following the restart are expected to return correct values as if the restart**

[1] So, you can't open files directly, or use anything that accesses the file-system directly, like a database.

did not occur. The same idea is true for update and delete operations. This can be achieved using the external library.

This primitive layer exposes a few basic operations; for this assignment, the only supported operations are reading and writing key-value records. The API will be extended and possibly changed in the next assignments.

3.2 Your Library

You should wrap the provided library with your own code, in order to provide higher-level facilities and richer functionality. This library will be useful not only for this assignment, but for all following assignments as well (and possibly to another team in the final assignment.)

3.3 Gradle

The provided starter project is managed by Gradle, a build system. We use the Kotlin DSL option for writing Gradle's configuration files. Gradle is a very powerful tool, revisit the tutorial on Gradle and the online docs if needed. To import the starter project in a recent version of IntelliJ, go to "File ▶ Open...", navigate to the `assignment1` directory, and press "OK". The default import options are fine.

When you're done, you should have the base project loaded in your IDE, inside which are several subprojects.

To check if Gradle is properly configured from the command-line (Or from IntelliJ, in the menu that appears after pressing "Ctrl" twice), try running: `gradle run` (`gradlew` in command-line). In the output, you should see "Success!" signifying that the starter project is working correctly.

3.3.1 Multi-project structure

The starter project is comprised out of 4 projects:

base This is the root project. All the others are subprojects of it.

library This is the library described in section 3.3.4, which will also be used in future assignments. You should add storage-layer code here.

techwm-app The business logic part. This is where code specific to meeting concrete requirements of this exercise goes.

techwm-test The purpose of this subproject is to hold tests for your code by the course staff. It is currently populated by some basic tests; the test-suite used for grading will be larger and more extensive. You shouldn't put anything important here as this entire sub-project will be replaced with the staff test-suite.

The recommended architecture for the assignments in this course is shown in figure 1. The tests that will determine your grade will test only the application, which should use features from the library, which in turn must use the provided primitive storage layer.

Note that this is a mere recommendation—The only actual requirements are that the test subproject be able to test the application subproject, and that the primitive storage layers be used. You *are* allowed to structure your solution in any way you'd like.

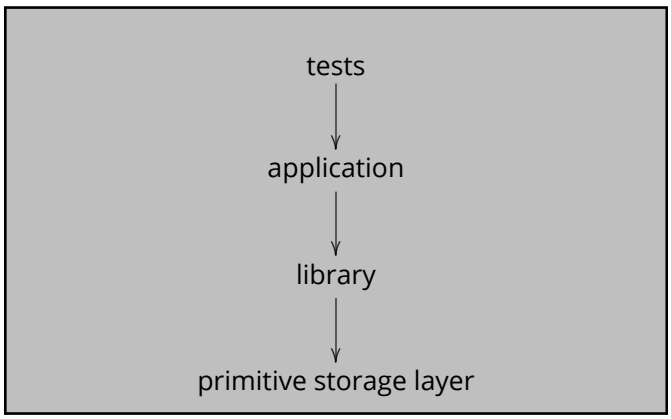

Figure 1: Recommended architecture. Each layer uses code from the lower layer, and the provided storage layer acts as the lowest abstraction

3.3.2 Gradle project directory structure

The provided starter project directory structure is shown in figure 2. The `build.gradle.kts` are the various project's configuration, most important of which are the dependencies (you may want to add some of your own!). Each project has a `src/main/kotlin` directory, inside which you should place your main code, and a `src/test/kotlin` for tests. There are also `src/{main,test}/resources` directories where you should put associated files (e.g., test data) for the main code and tests respectively.

```
.
└── base
    ├── build.gradle.kts
    ├── techwm-app
    │   ├── build.gradle.kts
    │   └── src
    │       ├── main
    │       │   ├── kotlin
    │       │   └── resources
    │       └── test
    │           ├── kotlin
    │           └── resources
    ├── techwm-test
    │   ├── build.gradle.kts
    │   └── src
    │       └── test
    │           ├── kotlin
    │           └── resources
    ├── library
    │   ├── build.gradle.kts
    │   └── src
    │       ├── main
    │       │   ├── kotlin
    │       │   └── resources
    │       └── test
    │           ├── kotlin
    │           └── resources
    └── settings.gradle.kts
```

Figure 2: Starter project directory structure

3.3.3 Inter-project dependencies

The projects have been configured to reference each other, as in figure 1. You can look at how this is done in the `build.gradle.kts` files. You are allowed to make changes here, but this is not recommended.

3.3.4 External Library

There is an external library consisting of two interfaces:

```
interface SecureStorageFactory {
    fun open(name: ByteArray): SecureStorage
}

interface SecureStorage {
    fun write(key: ByteArray, value: ByteArray) { /* ... */ }
    fun read(key: ByteArray): ByteArray? { /* ... */ }
}
```

`SecureStorageFactory` opens a named database - `SecureStorage` that you can use to write and read. The different `SecureStorage` are independent, persistent, and are first initialized with no data. The `write` function adds an entry to the database, replacing if it already exists. `write` supports writing a ByteArray with a **limit to its size - 100 bytes**. Writing an array of more than 100 bytes will have an undefined behaviour.
`read` returns the contents of an entry given its key or `null` if the key doesn't exist.

The starter project has already been configured (specifically, the `library` subproject) to use a non-functioning implementation of this API. When grading your submission, a real implementation will be used. Once again, note that you *must* use the implementation provided, and nothing else, for persistence in the application.

4 The SifriTaub™ application

This semester, as mentioned, we are developing a book lending system, SifriTaub. We target the Technion CS Faculty community, in hopes that both students, researchers, and system administrators will find this system useful. This assignment focuses on an integral part of the system, *user management*.

4.1 Operations

We've defined create, delete, and read operations. Look at `SifriTaub.kt` in the starter project for the operations you must implement and their documentation.

4.2 Managing books

Users would like to add books to the library and view which books are available. In the future, we allow users to borrow books and wait for them.

4.3 Time limits

The purpose of the time limits is to make your implementation fairly efficient. When reading data from the database, make sure you're doing it in linear time in respect to the read data. Anyway if the tests run under the time limits, you are OK.

4.3.1 External library

The external library is slow, and has the following properties:

1. The `open` method is linear in the number of databases created, 100ms for each database. So, if you've created 10 databases, the method will return after 1 second.
2. The `write` method has constant time complexity. It returns immediately.
3. The `read` method is linear in the size of the returned value, 1ms for each byte returned. So, if the returned value is "softwaredesign" (16 bytes in UTF-8 encoding), the method will return after 16ms. Again, the length of the key does not influence time complexity.

4.3.2 Test sizes

Each test can include up to one million (1,000,000) users and up to 1,000,000 books.

4.3.3 Test time limits

The entire test suite has a time limit of about 10 minutes (this value may be adjusted according to the size of the test suite.). This includes loading users and adding books into the system.

Each test can contain up-to 5 read queries, a single create, update or delete call, and any number of constructor calls, and should take up to 10 seconds. The test may have a setup or teardown part which is not counted in the 10 seconds, but is counted for the overall time limit.

Tests that include listing book ids will have a more generous time limit of 20 seconds and will list at most 10 book ids at a time (But you can definitely pass them in under 10 seconds).

4.4 Dummy implementation

For your benefit, a dummy implementation of the persistent storage layer is provided. The implementation is non-functional.

4.5 Dependency injection and GUICE

You must use GUICE in this assignment. Write your classes to use dependency injection, and use `@Inject` in any class where dependency injection is relevant. You must implement `SifriTaubModule` so that it contains all of your bindings (you can use several modules if that is more useful to you, but our tests will use only `SifriTaubModule` .)

You are provided with `SecureStorageModule`, which provides a binding for `SecureStorageFactory` (a default, dummy implementation). That module and the fully-implemented module that we will test your code with have no binding for `SecureStorage`; you should think about how to dependency inject `SecureStorage` using Guice.

Note that our tests *only* use `SifriTaubModule` (see the staff test for an example.) You need to configure that module to include `SecureStorageModule` by yourselves.

4.6 Tips & Tricks & Hints

1. There are many ways to implement this exercise, and some of the design questions you will encounter have no correct answer. It is very important that you discuss the assignment amongst yourselves and make sure you understand and have a clear plan of attack. This is a crucial step of the design process, and part of the reason you are split into pairs.
2. You should think about abstractions and data-structures, then implement them in your library. However, be aware that spending too much time on the library can be a waste, resulting in a interface that is "too generic" instead of solving the problem directly.
3. Use ideas that were taught in class, even if you're not required to.
4. You are allowed and encouraged to use external libraries, and you're also allowed to change Gradle's build files as required.
5. You can have multiple "files" using the primitive layer. Your data-structure may span several keys.
6. Use source control! Git is awesome. But use a private repository, we *will* check for plagiarism.

5 Dry part: *How to Design a Good API and Why It Matters*

Watch Joshua Bloch's lecture on API design from Google's "Tech Talks" series, which is available on YouTube (`https://youtu.be/heh4OeB9A-c`). This talk is part of the course material (we may even ask questions about it in the exam!).
Afterwards, answer the following questions:

1. What is an example *from the lecture* of a misuse of inheritance?
2. What is an example *from the lecture* of a disadvantage of checked exceptions?

5.1 Dry part, addendum: Absolute Minimum You Have To Know About Unicode

Read Joel Splosky's article, The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!), from his blog, "Joel on Software". This blog post is part of the course material, and may even be discussed in the exam. After reading, answer the following questions:

3. According to the article, what do you need to know in order to make use of a string?
4. Describe the symbol you were shown instead of an unknown Unicode code point.

6 Future assignments

This assignment is an introduction. The next two assignments will iterate over this application, adding features and improving the design of the library. Each assignment will add new features, working with what you have already implemented, with possible minor refactorings. The general architecture described in section 3 will stay mostly the same.

7 Administrivia

7.1 Late submission

In special cases, assignments can be submitted up to 5 days after the submission closes. The late submission penalty is $n \cdot 5$, where n is the number of days after the submission date.

The exceptions to this rule are of course reserve duty (Miluim), and very exceptional circumstances to be decided by the TA. In any case, special extensions must be requested in advance. Last minute extensions are unlikely to be granted, and extensions *after* the submission date will *not be granted* at all.

7.2 Your grade

You will be graded based on the following criteria:

1. Number of tests passed (Automatic testing).
2. Light code-review. If your design is deeply problematic, you will be penalized for that. However, if your design could be better, you will not be penalized, and you'll get notes which you should fix for the next assignment (Manual testing).
3. The existence and quality of your tests, and that they pass (Manual testing).

8 Submission instructions

The project must be submitted as a ZIP file. We have provided automation for this purpose in the form of a GRADLE task that zips the entire project directory into a file named `submission.zip`. To use it, run the following command from the root directory of your project (the `assignment1` directory that contains everything else) or from INTELLIJ in the prompt that appears after pressing "Ctrl" twice:

```
gradle submission
```

In addition to writing your code, you should fill out `README.md` with your names, ID numbers, and project documentation, and add your answers to the dry questions in section 5 on the preceding page as a PDF file named `dry.pdf`. Everything should be in the project root directory, and subsequently in the `submission.zip` file.

You *must* make sure the submission file is ***less than 1MB***. Larger submissions will be silently ignored! Use Gradle to decrease the size of your submission. Make sure you do not submit any log files, compiled binary files (e.g., .class or .jar files) or other spurious baggage. Look at the `exclude` rules in the `build.gradle.kts` to change what is not included with the submission.
Make sure to *test* your submission after creating the zip file!

8.1 Submission checklist

- Did you use the provided external library, remembering not to use any other persistence methods?
- Did you add all bindings to the Guice module?
- Did you write tests for everything? How's your coverage[2]?
- Do the provided minimal tests pass (with an implementation of the external library)?
- Did you use the Sequence API, and other Kotlin features we learned in class? You don't have to, but they're useful.
- Did you answer the dry questions and add a PDF?
- Did you fill out `README.md` ?
- Did you extract the submission zip to a different location and make sure it compiles, including running `./gradlew test` *from the command-line*[3]?

If you answered yes to all, you're good to go! ☺

And most importantly, have fun!

[2]Your grade does not depend on a value returned by a coverage tool, but as we learned in class coverage is a useful tool.
[3]Your submission will be tested from the command-line, so make sure that works!

A Changelog

1. **14.4**: Removed irrelevant items added by mistake to section 4.4 (Tips & Tricks)