Futher Java
Supervision 1

Supervisor: Joe Isaacs (josi2).
All work should be submitted in PDF form with page numbers 36 hours before the supervision to the email `josi2@cam.ac.uk`. If you have any questions on the course please include these at the top of the supervision work and we can talk about them in the supervision. Please include all code that you think is relevant (that you want me to look at in the PDF).

A *key-value store* (KVS)[1] is a typed (possible persistent) database used for storing values with a associated keys. In this supervision you will build a KVS in Java. This KVS will be a service that clients can connect to then possibly read (or write) values to (or from) this store. While you are implementing this KVS you will be carrying out simple analysis and then evaluating you implementation. You will implement a server which (one or more) client(s) can connect (simultaneously) to then query the value for one or more keys and set values for one or more keys. The state in this KVS should be serlisable (in the database sense[2]). The KVS should be typed and support `String`s.

A *future* construct used to model a computation that may or may not be finished. The computation will be run asynchronously, calling `get` on a future will either return the result of the computation or cause the calling thread to wait until the computation is finished and then the calling thread will continue with the result.

Tasks:

1. Implement a thread safe hash map (Do not use any *thread safe* standard library classes[3]) Write a test to show that its implementation is indeed thread safe. There are frameworks for writing tests JUnit[4] if you can try using this. It is integrated into Intellij IDEA[5].

2. Discuss the protocol which clients and servers will communicate KVS operations via with you supervision partners. The operations should be [`put`, `get`, `remove`]. Describe this protocol in your answers. How do you handle faliure?

   `get` for example could either return a value or an error if no value is present.

3. Implement a KVS server and client. Ensure that all on the server resources are cleaned up when a client disconnects. For each part of the protocol in the previous question show where you implement this in your code. How do the client and server commutative?

4. (a) Implement a future with the `Future<V>` interface[6] ignore `cancel`,
   - using threads [7]

[1]`https://en.wikipedia.org/wiki/Key-value_database`
[2]`https://en.wikipedia.org/wiki/Serializability`
[3]However you many use other standard library classes
[4]`https://junit.org/junit5/docs/current/user-guide/#writing-tests`
[5]`https://www.jetbrains.com/help/idea/configuring-testing-libraries.html`
[6]`https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/Future.html`
[7]you will need to add another method `void run()`

- using a thread pool[8] where work can be submitted. Implement your own thread pool. Use the `Callable<T>` interface[9] in the thread pool. Can the builder pattern[10] allow for a cleaner API for the creation of many futures for a given thread pool?

(b) Compare the benefits of using futures implemented using thread against thread pools. What sort of workloads would I use for each and why?

[8] Have a list of thread and a list of callable's

[9] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/Callable.html`

[10] `https://en.wikipedia.org/wiki/Builder_pattern`