Ex 8: Welcome to the Future

Basic Information

- **Deadline:** 13 November 2024, Wednesday, 23:59 SGT
- **Difficulty:** ★★★

Prerequisite

- Caught up to Unit 39 of Lecture Notes.
- Familiar with CS2030S Java style guide.

Source

You are given a `.class` file called `Source.class`. This is the source of data. It has a built-in busy-waiting operations that will purposefully delay computation of otherwise simple operations. Such busy-waiting operations simulate extensive computation.

This `Source.class` file is compiled on PE node using Java 21. It may not work with other version of Java. In case you lost the `Source.class` file, we have created a directory called `Pristine` to store a copy of the `Source.class` file.

To ensure fair comparison between parallel and sequential computation, we let the busy-waiting to be deterministic for each run. Do not rely on this as they are not deterministic across different runs.

Wednesday

As the deadline is Wednesday, you may not have received your feedback during your lab session.

Preliminary

The rental data `Rent.csv` shows the state of housing rental in Singapore. The data is composed of 4 rows.

Town	Block	Type	Rent
The town name	The block name	Unit type	Monthlu rental price

While we actually have the entire data from 2021, we restrict this to only January of 2023. Otherwise we will be dealing with millions of rows. In fact, two towns are simplified even further to only contain a small number of blocks.

They are `"BISHAN"` and `"QUEENSTOWN"`. Most town have over 10000 rental units with over 100 blocks. As such, running them may take minutes if not hours. These two are simplified so that we can easily check for correctness.

Restriction

You are not allowed to modify the `import`. As such, you are only allowed to use `CompletableFuture` to solve this. You are also not allowed to use parallel stream.

Additionally, you are not allowed to modify the following. We will use our version during testing.

- `Rent.csv`: Some of our computation may be order specific.
  - If you wish to understand the data, we recommend making a copy and analyze the copied data.
- `RentData.java`: This file is given so you understand the details of the row.
  - The class file `Source.class` is compiled with the original `RentData.java`. Changes to the `RentData.java` may cause `Source`.class to throw unexpected error.
- `Sequential.java`: The class is given as a reference for a sequential computation.
  - The content is copied to `Parallel.java`.
  - Your task is to modify `Parallel.java` to utilize asynchronous computation.
- `Computation.java`: This is simply an abstract class to make `Timing.java` simpler.

Note, you may use loop.

Computation

The computation to be performed is as follows:

1. Get the name of the town.
2. Given the name of the town, find all the blocks available within the town.
    - This is done by `this.source.findBlock(town)`.
    - The available blocks within the town is returned as a `String[]`.
3. For each block, find all the unit type within the block and within the town. This is the `static` method called `processBlock`.
    - This is done by `this.source.findTypeInBlock(town, block)`.
    - The available unit types is returned as a `String[]`.
4. For each type, find the minimum price.
    - This is done by `this.source.findMinPrice(town, block, type)`.
    - The minimum price for the given unit type within the given block and within the given town is returned as an `int`.

Tasks

Speedup

While we do not worry about efficiency, we want to have a reasonable speedup. Testing on PE node, we manage to get at least 350% speedup. Try to get at least 350% speedup on PE node. Otherwise, penalty may be given.

Task 1: Asynchronous Computation

The following methods can be rather expensive to compute.

- `String[] Source::findBlock(String town)`
- `String[] Source::findTypeInBlock(String town, String block)`
- `int Source::findMinPrice(String town, String block, String type)`

The given program runs the computation synchronously. Whenever possible they should be run asynchronously. Your task is to make both `processBlock` and `run` methods asynchronous. Note that `processBlock` is private so you are allowed to modify the return type as you deem fit.

The general idea in making the program asynchronous is to utilize `CompletableFuture`. You are to make both methods above asynchronous using `CompletableFuture`. You may add some private method to simplify your work that returns a `CompletableFuture`. With `CompletableFuture`, you simply have to take care of the task dependency.

However, note that the print order should remain the same regardless of how the concurrency is executed. You must ensure that in the end, the print order is the same as the original sequential computation.

Some complication that you may encounter is that the number of blocks within the town as well as the number of unit types within the blocks cannot be known in advanced. So you cannot know how many `CompletableFuture` is needed. As such, you may need a collection of `CompletableFuture`.

Then, in the case you need to wait for all of the `CompletableFuture` to finish, remember that the method signature is

```
static CompletableFuture<Void> allOf(CompletableFuture<?>... cfs);
```

But if you do not know how many `CompletableFutures` are there, how can you write the code? What else can the method above accepts?

test.sh

```
javac -Xlint:rawtypes -Xlint:unchecked *.java
bash test.sh Parallel
$ java -jar ~cs2030s/bin/checkstyle.jar -c ex8_style.xml Parallel.java
```

Timing.java

```
javac -Xlint:rawtypes -Xlint:unchecked Timing.java
java Timing
$ java -jar ~cs2030s/bin/checkstyle.jar -c ex8_style.xml Parallel.java
```

Following CS2030S Style Guide

You should make sure your code follows the given Java style guide.

Further Deductions

Additional deductions may be given for other issues or errors in your code. These deductions may now be unbounded, up to 5 marks. This include *but not limited to*

- run-time error.
- failure to follow instructions.
- improper designs (*e.g.*, not following good OOP practice).
- not comenting `@SuppressWarnings`.
- misuse of `@SuppressWarnings` (*e.g.*, not necessary, not in smallest scope, *etc*).