Distributed Computational Frameworks

The goal of this recitation is to help you prepare for the final homework assignment, in which you will build a distributed MapReduce framework. In this recitation you will build a client/server system in which worker servers execute code on behalf of a remote client, much as they do with MapReduce.

A simplified computational framework

Implement a simple framework that supports remote computation. Specifically:

1. A client sends a task to the worker servers where the data is stored.
2. The servers each execute the task and send the final result back to the client.
3. When the client receives all results from the workers, it combines the results and outputs the final combined result.

All tasks in this framework implement the `Task<T>` interface, which requires a single method `execute(InputStream)` that returns generic type `T`. A `CountWordTask` that counts the number of occurrences of a given word has already been implemented for you and is included in the recitation starter code.

Remote computation on a single worker

Start by implementing a framework that supports remote computation by a single worker. To do this, complete the following tasks:

1. Finish implementing the `ExecuteTaskCommand` class. The `ExecuteTaskCommand` is a simple `WorkerCommand` that executes a generic `Task<T>` and sends the calculated result back to the client.
2. Implement the `run()` method in the `CountWordClient`. The `CountWordClient` should send the `CountWordTask` to a worker server and then wait for the worker to respond before printing the final answer.

Remote computation on multiple workers

Modify the `run()` method in the `CountWordClient` to send the `CountWordTask` to multiple worker servers. The `CountWordClient` should wait for all worker servers to finish before printing the final answer.

One tricky aspect of this exercise is learning how to wait for *all* workers to finish before printing the final result. Java's higher-level abstractions in the `java.util.concurrent` package simplify this task. Consider using the `ExecutorService#invokeAll()` method, which takes a collection of `Callable`s and returns after all those `Callable`s return.