Executor Services and Java RMI

The goal of this recitation is to better familiarize you with the use of ExecutorServices and coordination between parallel tasks.

Remote procedure call with Java RMI

We have rewritten the code for recitation 12 with Java RMI. Take a look at the differences between this and the recitation 12 solution.

RMI takes care of the reflection for you. However, RMI requires a Security manager to do so, as executing arbitrary code from the Internet is a clear security hazard. We provide a simple security policy for use. Add the following arguments to your IDE's run configuration under VM options for the servers.

```
-Djava.security.manager -Djava.security.policy="simple.policy"
```

Using Executor Service to coordinate work

We build upon recitation 12 by adding a new type of server, the aggregate server, that supports the aggregate operation. That is, given a list of file names and a task, the aggregate server asks all the other servers with those files to perform the tasks in parallel, before aggregating all the results and returning it to the client.

We have already provided the RMI interface and all the necessary infrastructure for this to work. Take a look at AggregateService, the RMI interface that defines this operation, and complete the class AggregateServer.

Putting it all together

To run this locally, do the following in order:

1. run FileServer with the command line arguments corresponding to peer1 in the Client class, with root directory asset1.

2. run FileServer with the command line arguments corresponding to peer2 in the Client class, with root directory asset2.

3. run AggregateServer with command line arguments corresponding to aggregate in the Client class.

4. run Client.