Lab Exercise – Build A Simple File Hosting Service using Java RMI

In this lab, you will implement a simple file hosting service which uses Java RMI. You will create:

1. **A FileService interface:** This is the interface which will tell your client how to interact with the file hosting service. The interface will extend java.rmi.Remote, and will have three methods:
    a. getFile(): takes the requested file name as an argument, and returns the file as a byte array
    b. getFileNames(): returns a list of the names of all files hosted by the service
    c. uploadFile(): takes two arguments, the name of the file to be uploaded, and the file contents as a byte array
2. **FileServiceImpl:** This is the implementation of the FileService interface.
3. **FileServiceSetup:** This contains the main method which will be used to start the FileService. Server-side files should be stored in a folder called "serverFiles". Create one or two sample .txt files in the serverFiles folder, e.g. "serverFile1.txt", "serverFIle2.txt". The main method of FileServiceSetup should:
    a. Create an instance of FileServiceImpl, and pass the list of file names in the serverFiles folder to its constructor.
    b. Start the RMI registry on port 1099
    c. Bind the instance of FileServiceImpl to the RMI registry with the human-readable name "fileService"
    d. Print out "Server ready" to the console once the above is accomplished
4. **Client:** This is the client which will connect to the service. Client-side files will be stored in a folder called "clientFiles". The client should:
    a. Ask the RMI registry running on localhost and listening in port 1099 for the instance of the FileService object that is bound to the name fileService
    b. Print the list of file names on the server to the console
    c. Make a remote invocation to the service to get one of the files listed and write that file out to disk into the clientFiles folder
    d. Upload a file called "uploadTest.txt" to the file service. First read the file to be uploaded as a byte[], then make a remote invocation to the service's uploadFile() method to upload the file
    e. Print out the list of file names on the server to the console again, showing that the file "uploadTest.txt" has been uploaded to the server

Extra Features: Refactor the client so that rather than having hard coded behaviour, it can instead take command line arguments for listing all files on the service, retrieving a particular file from the service, or uploading a file to the service.

A full solution to this lab exercise will be posted on Moodle next week.

HINT: to read a file to a byte array

```
byte[] bytes = Files.readAllBytes(new File("/path/filename.txt").toPath());
```

HINT: to write from a byte array to a file

```
FileOutputStream stream = new FileOutputStream("/path/filename.txt");
stream.write(bytes);
stream.close();
```