Task #1:

1. To accomplish this task, a directory structure was created; three folders were created and the required files were saved inside of them.

FileServer

Server: FileInterface.java, FileImpl.java, FileServer.java, policy.txt
Client: FileInterface.java, FileClient.java
I couple text files were created and put inside the server folder. These are the files to be downloaded by the client from the server; **text1.txt, text2.txt**
The rmiregistry was run, the server was run with the policy.txt file (simply specifies who has permission to run), and finally the client was run last with the IP address specified and the name of the file to be downloaded.

The result was that the file to be downloaded appeared in the "client" folder.

2. A challenge which was faced for this task is that I tried to run the applications in the Eclipse IDE. Some errors occurred because the package needed to be specified. This problem was solved by simply running the files on command prompt.

3. I learned how to utilize Java RMI remotely invoke a method via the server and receive a response (a text file) on the client.

Task #2:

1. In this task, the code from task #1 needed to be modified so that useful messages appeared on the console when: a) the server starts, b) the ip address of the connecting client, and c) when the client downloads a file successfully, the client prints a message.

This was easily solved by inserting the correct message after the specified event occurred.

2. The only challenge that occurred was how to print the IP address of the connecting client. This was solved by utilizing the RemoteServer.getClientHost() method by importing java.rmi.server.RemoteServer.

3. This task mainly showed me how to get the client host IP address. I already knew how to show useful messages for the required events.

Task #3:

1. For this task, all that needed to be done is to run the task 2 code on a server, and then run the client code on a different machine (a peer's machine). To do this, my peer ran the rmiregistry and server on his machine, then I ran the client by specifying their IP address and the file located on their machine. I saw the file show up in the folder on my machine, in the client folder.

2. There were no issues that were encountered in this task; every step ran smoothly.

3. I gained hands-on experience running the client/server application on two separate machines. We simply needed to find the IP address of the server host machine.

Task #4:

1. This task required us to modify the code so that the client uploads a file to the server; essentially the vise-versa operation. To accomplish this, a new method needed to be created in the FileInterface and implemented in the FileImpl file. The client looks for the file using the file name binding feature in Java RMI. The file was sent as a byte array to the server and downloaded.

2. The main challenge which was faced was a naming rebind error. This was solved by changing the path of the file to accurately reflect where the file is located on the client.

3. I learned how to reverse the communication between the client and the server so that their roles are switched.