Client

```
import java.io.*;

public class Client
{
	final String baseFileName = "dlToast";
	final String fileExtension = ".txt";
	final int numThreads = 5;
	final int basePort = 3339;
	ClientThread[] clientThreads = new ClientThread[numThreads];


	private void mergeFiles() throws Exception {
		FileOutputStream wholeThing = new FileOutputStream(baseFileName + fileExtension);
		for (int i = 0; i < numThreads; i++) {
			FileInputStream fileInput = new FileInputStream(baseFileName + (i + 1) +
fileExtension);
			while (fileInput.available() > 0) {
				wholeThing.write(fileInput.read());;
			}
			fileInput.close();
		}
		wholeThing.close();
		System.out.println("Finished merging files.");
	}

	private void downloadFiles() throws Exception{
		for (int i = 0; i < numThreads; i++) {
			clientThreads[i] = new ClientThread(i+1, basePort + i, baseFileName,
fileExtension);
			clientThreads[i].start();
		}
		for (int i = 0; i < numThreads; i++) {
			clientThreads[i].join();
		}
	}

	public static void main(String[] args) throws Exception
	{
		Client c = new Client();
		c.downloadFiles();
		c.mergeFiles();
	}
}
```

ClientThread

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.net.Socket;

public class ClientThread extends Thread {
	private Socket socket = null;
	private DataInputStream inStream = null;
	private DataOutputStream outStream = null;
	private int targetPort, portion;
	private String baseFileName, fileExtension;


	public ClientThread(int portion, int port, String basename, String extension)
	{
		targetPort = port;
		this.portion = portion;
		baseFileName = basename;
		fileExtension = extension;
	}

	public void createSocket()
	{
		try
		{
			socket = new Socket("localHost", targetPort);
			//socket.
			System.out.println("Connected");
			inStream = new DataInputStream(socket.getInputStream());
			outStream = new DataOutputStream(socket.getOutputStream());
		}
		catch (Exception u)
		{
			u.printStackTrace();
		}
	}

	public void receiveFile()
	{
		//FileOutputStream fileOut;
		try
		{
			int fileSize = inStream.readInt();
			byte data[] = new byte[fileSize];
			FileOutputStream fileOut = new FileOutputStream(baseFileName + portion + 
fileExtension,true);

			//		int totalBytes = inStream.read(data, 0, fileSize);
			//		fileOut.write(data);
			//		fileOut.flush();

			int count =0, totalBytes=0;
			while(true)
			{
				count = inStream.read(data,0,fileSize);
				byte[] arrayBytes = new byte[count];
				System.arraycopy(data, 0, arrayBytes, 0, count);
				totalBytes = totalBytes + count;
				if(count>0)
				{
					fileOut.write(arrayBytes);
					fileOut.flush();
				}
				if(totalBytes == fileSize)
					break;
			}
			System.out.println("File Size is: "+fileSize + ", number of bytes read are: " + 
totalBytes);

			socket.close();
			inStream.close();
			fileOut.close();
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
	}

	public void run()
   {
      createSocket();
      receiveFile();
   }
}
```

Server

```
import java.io.File;

public class Server {

	public static void main(String[] args) {
		final String baseFileName = "Toast";
		final String fileExtension = ".txt";
		final int numThreads = 5;
		final int basePort = 3339;
		String fileName = baseFileName + fileExtension;
		ServerThread[] serverThreads = new ServerThread[numThreads];
		CustomFileManager cfm = new CustomFileManager(fileName, numThreads);

		cfm.createPartitions(baseFileName, fileExtension);
		for (int i = 0; i < numThreads; i++) {
			serverThreads[i] = new ServerThread(basePort + i, new File(baseFileName + (i +
1) + fileExtension));
			serverThreads[i].start();
		}

		for (int i = 0; i < numThreads; i++) {
			try {
				serverThreads[i].join();
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}

	}
}
```

ServerThread

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;

public class ServerThread extends Thread {

        String hostName;
        int portNumber;
  private File file = null;

  private ServerSocket serverSocket = null;
  private Socket socket = null;
  private DataInputStream inStream = null;
  private DataOutputStream outStream = null;
  private final int bufferSize = 512*1024; //buffer size 512 KB




        public ServerThread(int i, File file) {
                portNumber = i;
                this.file = file;
        }


  private void createSocket()
  {
    try
    {
      serverSocket = new ServerSocket(portNumber);
      socket = serverSocket.accept();

      inStream = new DataInputStream(socket.getInputStream());
      outStream = new DataOutputStream(socket.getOutputStream());
      System.out.println("Connected");
    }
    catch (IOException io)
```

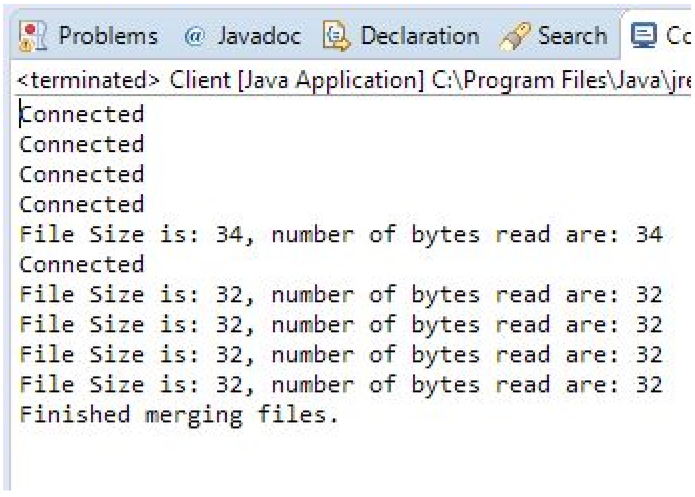
```
        {
          io.printStackTrace();
        }
    }

    private void sendFile()
    {
        try
        {
//write the filename below in the File constructor
            //File file = new File("Big Data.zip");
            FileInputStream fileInput = new FileInputStream(file);
            int fileSize = (int) file.length();
            System.out.println("Server: File size is:" + fileSize);
            byte [] data = new byte[(int) fileSize];

            fileInput.read(data);

            //first send the size of the file to the client
            outStream.writeInt(fileSize);
            outStream.flush();

            outStream.write(data);
            outStream.flush();
            fileInput.close();
            serverSocket.close();
            socket.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public void run()
    {
        createSocket();
        sendFile();
    }


}
```

CustomFileManager

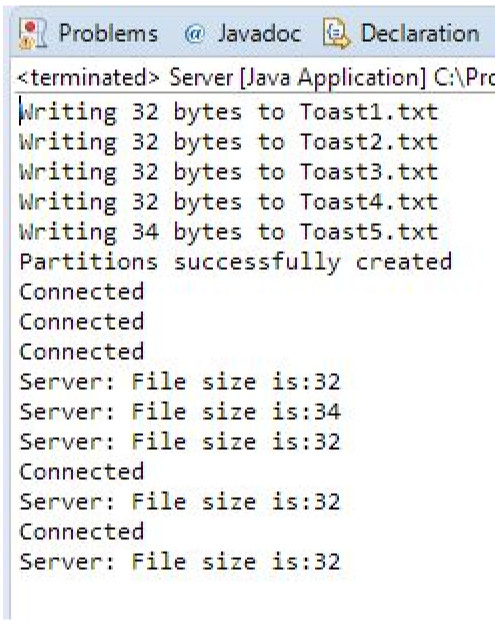
```
//package ProjectFall2016;
import java.io.*;

public class CustomFileManager
{
    private File source;
    private int partitions;
    FileInputStream reader;
    FileOutputStream writer;
    private byte [] data;
    private int fileMinSize;
    private int leftOverBytes;
    private boolean lastPartitionFlag = false;
    //user will pass the source file and number of partitions required
    CustomFileManager(String fileName, int p)
    {
        try
        {
            source = new File(fileName);
            partitions = p;
            //min size of each partition
            fileMinSize  = (int) (source.length()/partitions);
            //extra bytes with the last partition along with min size
            leftOverBytes = (int) (source.length()%partitions);
            reader = new FileInputStream(source);
        }
        catch(Exception e)
        {
            e.printStackTrace();
            System.out.println("Error opening the source file");
        }
    }

    //This method will find the size of each partition and call to helper methods for read source file
    //and write into each partition
    public void createPartitions(String fileBase, String extension)
    {
        int fileCounter = 1;
        for(int i = 0 ; i < partitions ; i++)
        {
            //check if it's the last partition
            if(i == (partitions - 1))
            {
                lastPartitionFlag = true;
            }
            data = readFile();
            writeFile(data, fileBase + fileCounter + extension );
            fileCounter++;
        }
        System.out.println("Partitions successfully created");
    }

    private void writeFile(byte [] d, String fileName)
    {
        try
        {
            System.out.println("Writing " + d.length + " bytes to " + fileName);
            writer = new FileOutputStream(new File(fileName));
            writer.write(d);
            writer.flush();
            writer.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
            System.out.println("Error writing file");
            System.exit(0);
        }
    }

    private byte [] readFile()
    {
        byte [] tempData;
        if(lastPartitionFlag)
            tempData = new byte[fileMinSize+leftOverBytes];
        else
            tempData = new byte[fileMinSize];
        try
        {
            reader.read(tempData, 0, tempData.length);
        }
        catch(Exception e)
```

```
        {
            e.printStackTrace();
            System.out.println("Error reading file");
            System.exit(0);
        }
        return tempData;
    }

}
```

Screenshots of Sample Runs



```
<terminated> Client [Java Application] C:\Program Files\Java\jre
Connected
Connected
Connected
Connected
File Size is: 34, number of bytes read are: 34
Connected
File Size is: 32, number of bytes read are: 32
File Size is: 32, number of bytes read are: 32
File Size is: 32, number of bytes read are: 32
File Size is: 32, number of bytes read are: 32
Finished merging files.
```



```
<terminated> Server [Java Application] C:\Prc
Writing 32 bytes to Toast1.txt
Writing 32 bytes to Toast2.txt
Writing 32 bytes to Toast3.txt
Writing 32 bytes to Toast4.txt
Writing 34 bytes to Toast5.txt
Partitions successfully created
Connected
Connected
Connected
Server: File size is:32
Server: File size is:34
Server: File size is:32
Connected
Server: File size is:32
Connected
Server: File size is:32
```

Client

Server

Analysis and Design

Our assignment was to create a download accelerator, given a file partitioner, a file server and and a file client. The purpose of the program was to produce a set of classes that could transfer a file faster than a single server-client pair could. This could be achieved if the server and client could utilize multiple ports simultaneously to transfer the file in several pieces.

The server class would need to divide the file into a known number of parts, and prepare several threads to utilize that number of ports at once. The client class would need to know that number of ports, prepare corresponding client threads to accept each portion of the file, and then combine them on its own at the end. The server and client thread classes were modified from the provided file server class and file client class, respectively, to allow the port number and file name to be selected by the main server and client, so that each thread could use a different port and so the files wouldn't overwrite each other.

The base server class used the file partitioner's power to split the file to be sent, and then created the right number of server thread classes to allow each piece to be sent at once. The client class created the same number of client thread classes and assigned them to open ports, then downloaded each piece of the file with numbered titles. The client then creates a file without a number to be filled with the contents of the sent file. The client opens this file, then opens each of the numbered files in order and copies their contents, byte for byte, into the newly created file. And thus, the original file is successfully transferred faster than a single client-server pair could under the same conditions.