

```
package vUtil;  
  
public class Math {  
}
```

```

/**My personal String utility class
*
*/

package vUtil;

public class string {
    public static final String alpha = "
    abcdefghijklmnopqrstuvwxyz";
    java.util.Random randy;

    /**Splits a string into 2 halves, each put into a 2
element array
    * halves[0] = first half of the string
    * halves[1] =
    *
    * @param input String to be cut in half
    * @return
    */
    String[] halves(String input){
        int delta = (int) java.lang.Math.floor(input.
length()/2);
        String front = "";
        String back = "";

        for(int i = 0; i < delta; i++)
        {
            front += input.charAt(i);
        }
        for(int i = delta; i < input.length(); i++){
            back +=input.charAt(i);
        }

        String[] array = {front, back};

        return array;
    }

    /**Finds the number of characters different in 2
strings of equal length
    *

```

```

    * @param a First string to be compared
    * @param b Second string to be compared
    * @return The number of characters different
between the 2 strings
    */
    int differencesInString(String a, String b){
        int counter = 0;
        for(int i = 0; i < a.length() && i < b.length
()); i++) {
            if(a!=b) counter++;
        }
        return counter;
    }

    public static String alphaAt(int index){return
String.valueOf(alphaAt(index));}

    /**Gets character of alphabet at index
    *
    * @param index
    * @return
    */
    public static char alphAt(int index){return alpha.
charAt(index);}

    /**Gets random lowercase letter of the alphabet
    *
    * @return Random lowercase letter    */
    public char getRandomLetterLower(){return alphAt(
randy.nextInt(26));}
}

```

```
package vUtil;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class writer {
    FileWriter writer;
    File output;

    writer(String fileName) {
        //if(fileName == null)throw new IOException("
        fileName is null");
        try {
            output = new File(fileName);
            writer = new FileWriter(fileName);
        } catch (IOException e) {
            //throw Exception("Setting up new writer
            failed.");
        }
    }
}
```

```

/**Creates txt files of various sizes
* Good for testing stuff
* @Author Steve Mastrokalos 7276900
*
*/
package vUtil.File;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class createTXT {
    FileWriter writer;
    File output;
    Random randy;
    int letterNum;

    createTXT() {
        String nameList = "";
        randy = new Random();
        for(int i = 0; i < 20; i++) {
            letterNum = i;
            if(i >= 3) {
                int n = randy.nextInt(1023) + 1;
                String temp = n + "KB";
                nameList += "\"" + temp + "\", ";
                setUpWriter(temp);
                nKBsOfTXT(n);
                endWriter();
            }else{
                int n = randy.nextInt(8) + 1;
                String temp = n + "MB";
                nameList += "\"" + temp + "\", ";
                setUpWriter(temp);
                nMBsOfTXT(n);
                endWriter();
            }
        }
        nameList = nameList.trim();
        System.out.println(nameList);
    }
}

```

```

    }

    void nBlankBytesOfText(int n, Boolean MB){
        if(MB) nMBsOfTXT(n);
        else nKBsOfTXT(n);
    }

    void nMBsOfTXT(int n) {
        for (int i = 0; i < n; i++) createMBofTXT();
    }

    void createMBofTXT() {
        nKBsOfTXT(1024);
    }

    void nKBsOfTXT(int n) {
        for (int i = 0; i < n; i++) createKilobyteOfTXT
    };
    }

    void createKilobyteOfTXT() {for(int i = 0; i < 1024
; i++) output(vUtil.string.alphaAt(letterNum));}

    /**
     * Sets up the file writer with the given file name
     *
     * @param fileName Name of the output file
     */
    void setUpWriter(String fileName) {
        try {
            output = new File(fileName);
            writer = new FileWriter(fileName);
        } catch (IOException e) {
        }
    }

    /**Outputs the inputted parameter to output file
     *
     * @param out output
     */
    void output(String out) {

```

```
        try {
            writer.write(out + "\n");
        } catch (IOException e) {

        }
    }

    /**Closes the writer
     *
     */
    void endWriter() {
        try {
            writer.close();
        } catch (IOException e) {
        }
    }

    public static void main(String[] args){
        createTXT ct = new createTXT();
    }
}
```

```

/** A utility class that encrypts or decrypts a file.
* Part of my personal utility library
*
*
* @author Steve Mastrokalos, based on code from www.
codejava.net */
package vUtil.Encryption;

import java.io.*;
import java.security.*;

import javax.crypto.*;

public class Encryption {

    /**Creates an encrypted file from a given
unencrypted file
    *
    * @param key Encryption key
    * @param inputFile File to be encrypted
    * @return Encrypted File
    * @throws EncryptionException */
    public static File encrypt(String key, File
inputFile) throws EncryptionException {
        try{
            File output = File.createTempFile("
tempEncryptionFile", "");
            encrypt(key, inputFile, output);
            return output;
        }catch(IOException e){

        }
        return null;
    }

    /**Creates an encrypted file from a given
unencrypted file
    *
    * @param key Encryption key
    * @param inputFile File to be encrypted
    * @return Encrypted File

```



```

    * @throws EncryptionException */
    public static File decrypt(String key, File
inputFile) throws EncryptionException {
        try{
            File output = File.createTempFile("
tempDecryptionFile","");
            decrypt(key, inputFile, output);
            return output;
        }catch(IOException e){

        }
        return null;
    }

    /**Encrypts a file into into the output file
    *
    * @param key    Encryption key
    * @param inputFile File to encrypt
    * @param outputFile    Location to send encrypted
File
    * @throws EncryptionException */
    public static void encrypt(String key, File
inputFile, File outputFile) throws EncryptionException
    {crypt(Cipher.ENCRYPT_MODE, key, inputFile, outputFile
);}

    /**Decryptes a file to the output file
    *
    * @param key    Encryption key
    * @param inputFile File to decrypt
    * @param outputFile    Location to put decrypted
File
    * @throws EncryptionException */
    public static void decrypt(String key, File
inputFile, File outputFile) throws EncryptionException
    {crypt(Cipher.DECRYPT_MODE, key, inputFile, outputFile
);}

    /**Encrypts or decrypts a file
    *
    * @param cipherMode    Encryption vs Decryption

```

```

modes
    * @param key           Encryption key
    * @param inputFile      File to be en/decrypted
    * @param outputFile     Output file
    * @throws EncryptionException Exception in case
en/decryption fails */
    private static void crypt(int cipherMode, String
key, File inputFile, File outputFile) throws
EncryptionException {
        try {
            //Sets up encoding/decoding algorithm
            String algo = "AES";
            Key cryptKey = new javax.crypto.spec.
SecretKeySpec(key.getBytes(), algo);
            Cipher cipher = Cipher.getInstance(algo);
            cipher.init(cipherMode, cryptKey);

            //en/decodes input
            FileInputStream inputStream = new
FileInputStream(inputFile);
            byte[] inputBytes = new byte[(int)inputFile
.length()];
            inputStream.read(inputBytes);

            //Outputs result
            byte[] outputBytes = cipher.doFinal(
inputBytes);

            FileOutputStream outputStream = new
FileOutputStream(outputFile);
            outputStream.write(outputBytes);

            //Closes streams
            inputStream.close();
            outputStream.close();
        } catch (NoSuchPaddingException |
InvalidKeyException | IllegalBlockSizeException |
NoSuchAlgorithmException | BadPaddingException |
IOException type) { //Catches any of the many possible
exceptions

```

```
        throw new EncryptionException("Error during  
en/decryption of files", type);  
    }  
}
```

```
/**Encryption Exception extention  
*Does what it says on the tin  
*/  
package vUtil.Encryption;  
  
/**Exception class for Encryption class  
* Only really one way to do this */  
public class EncryptionException extends Exception {  
  
    /**Exception for encryption failure */  
    public EncryptionException() {}  
  
    /** Exception for encryption failure  
    *  
    * @param message Message to send on failure  
    * @param throwable Part of extending an exception  
    */  
    public EncryptionException(String message,  
    Throwable throwable) {  
        super(message, throwable);  
    }  
}
```

```

/**A client class that creates client objects that send
    encrypted files to a server
    *
    * @author Steve Mastrokalos
    */
package Client;

import vUtil.Encryption.Encryption;

import java.io.*;
import java.net.Socket;

public class Client {
    private static DataOutputStream dataOutputStream =
null;
    private static DataInputStream dataInputStream =
null;
    private static final String[] FILE_NAMES = {"6MB",
"8MB", "7MB", "445KB", "599KB", "408KB", "229KB", "
233KB", "532KB", "443KB", "1003KB", "238KB", "802KB", "
785KB", "817KB", "143KB", "904KB", "14KB", "209KB", "
698KB"};

    /**Runs a client that will send a file to the
    server
    *
    * @param host
    * @param port
    * @param fileNumber
    */
    Client(String host, int port, int fileNumber){
        String filePath = FILE_NAMES[fileNumber];
        Client c = new Client(host, port, filePath);
    }
    Client(String host, int port, String filePath){
        try (Socket socket = new Socket(host, port)) {

            dataInputStream = new DataInputStream(
socket.getInputStream());
            dataOutputStream = new DataOutputStream(
socket.getOutputStream());

```

```

        System.out.println("Sending " + filePath +
" to the Server");

        sendFile(filePath);

        dataInputStream.close();
        dataOutputStream.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**Sends encrypted files to server in chunks
 * Two of my advanced features
 *
 * @param path Path of file to be sent
 * @throws Exception */
private static void sendFile(String path) throws
Exception {
    int bytes = 0;
    File file = new File(path); //gets file that
will be sent
    FileInputStream fileInputStream = new
FileInputStream(Encryption.encrypt("KingdomHeartsIII",
file)); //Gets input stream from encrypted file (The
key is one of my favorite games, worked out to 16
characters nicely)

    dataOutputStream.writeLong(file.length()); //
Sending file

    //File gets chunked here, one of the advanced
features
    byte[] buffer = new byte[4 * 1024];
    while ((bytes = fileInputStream.read(buffer
)) != -1) { //Sends the chunky goodness to the server
socket

        dataOutputStream.write(buffer, 0, bytes);
        dataOutputStream.flush();
    }
}

```

```
        fileInputStream.close();  
    }  
}
```

```

package Client;

/**Controls and runs multiple clients simultaneously (
So I don't need to start multiple client objects myself
)
 *
 * @Author Steve Mastrokalos 7276900 */
public class ClientControl {
    Client[] clients;
    ClientControl(int numberOfClients){createClients(
numberOfClients);}

    /**Defaults to sending the clients to port 727 (
part of my Student ID)
    *
    * @param numOfClients  Number of clients to create
    */
    void createClients(int numOfClients){createClients(
"localhost", 727, numOfClients);}

    /**Creates clients to send files to the given host
and port
    *
    * @param host  Host IP
    * @param port  Port number
    * @param numOfClients  Number of clients to create
    */
    void createClients(String host, int port, int
numOfClients){
        clients = new Client[numOfClients];
        for(int i = 0; i < numOfClients; i++){
            new Client(host, port, i);
        }
    }

    //Runs 20 clients
    public static void main(String[] args) {
ClientControl c = new ClientControl(20);}
}

```



```

/**Server class
* Recieves and stores files
*
* @author Steve Mastrokalos 7276900
*/
package Server;

import java.net.*;
import java.io.*;
import java.util.concurrent.Semaphore;

public class Server {
    private ServerSocket serverSocket;
    public int counter = 0;
    public static Semaphore semaphore;

    /**Creates server with a maximum number of allowed
active clients at once
    * The rate limitation/alt threading is one of my
advanced features
    *
    * @param maxClients    max active clients allowed
at once
    */
    Server(int maxClients){
        semaphore = new Semaphore(maxClients);    //
Creates a server with a limit of 5 active clients at a
time
    }

    /**Starts the server on given port
    *
    * @param port    Port to recieve data from */
    public void start(int port){
        try {
            serverSocket = new ServerSocket(port);
            while(true){
                new ClientHandler(serverSocket.accept
                (), counter).start();
                System.out.println(semaphore.
                availablePermits());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

        counter++;
    }
} catch (IOException e) {
    e.printStackTrace();
} finally {
    stop();
}
}

/**Ends the connection to a client
 * (Specifically closes the socket the client
connected to)
 *
 */
public void stop() {
    try {
        serverSocket.close ();
    } catch (IOException e) {
        e.printStackTrace ();
    }
}

/**Starts server on port 727, with 5 clients max
 *
 * @param args
 */
public static void main(String[] args) {
    Server server = new Server(5);
    server.start(727);
}
}

```

```

/**client handler class for server
 * Can handle multiple clients at once through
 multithreading
 *
 * @author Steve Mastrokalos 7276900 */
package Server;

import java.io.*;
import java.net.Socket;

/**Creates a multithreaded client handler
 *Allows server to deal with multiple clients at once
 *
 */
public class ClientHandler extends Thread {
    private Socket clientSocket;
    private DataOutputStream out;
    private DataInputStream in;
    int counter;

    /**Runs the thread when semaphore is free
     *
     * @param socket
     * @param fileNumber
     */
    public ClientHandler(Socket socket, int fileNumber
) {
        while(!Server.semaphore.tryAcquire()){
            this.clientSocket = socket;
            counter = fileNumber;
        }

        /**Runs the thread
         * Recieves the files from the clients
         *
         */
        public void run(){
            try {
                out = new DataOutputStream(clientSocket.
getOutputStream());
                in = new DataInputStream(clientSocket.

```

```

getInputStream());
        receiveFile("File" + counter + ".txt", in);

        clientSocket.close();

        in.close ();
        out.close ();
        Server.semaphore.release();

    } catch (IOException e) {

    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**Recives the files in chunks
 * Saves as a whole
 *
 * @param fileName
 * @param input
 * @throws Exception
 */
private static void receiveFile(String fileName,
DataInputStream input) throws Exception {
    int bytes = 0;
    FileOutputStream fileOutputStream = new
FileOutputStream(fileName);

    long size = input.readLong(); // read file size
    byte[] buffer = new byte[4 * 1024]; //prepares
buffer for dechunking
    while (size > 0 && (bytes = input.read(buffer,
0, (int) Math.min(buffer.length, size))) != -1) {
        //Writes in new file dechunked file
        fileOutputStream.write(buffer, 0, bytes);
        size -= bytes;
    }

    //Prints ot console job done
    System.out.println("File Received");
}

```

```
        fileOutputStream.close();  
    }  
}
```