CT255 Assignment 3
Steganography
Maxwell Maia
21236277

## Problem 1

```
/**
 * CT255 - Assignment 3
 * Skeleton code for Steganography assignment.
 *
 * @author Michael Schukat, Maxwell Maia
 * @version 1.0
 */

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Stegano1
{
    /**
     * Constructor for objects of class Stegano1
     */
    public Stegano1()
    {
    }
```

```
// use these arguments to test: "A", "inp.txt", "out.txt", "0010101"
public static void main(String[] args) {
    String arg1, arg2, arg3, arg4;
    Boolean err = false;

    if (args != null && args.length > 1) { // Check for minimum number of arguments
        arg1 = args[0];
        arg2 = args[1];

        if (arg2 == "") {
            err = true;
        }
        else if ((arg1.equals("A")) && (args.length > 3)){
            // Get other arguments
            arg3 = args[2];
            arg4 = args[3];
            if (arg3 == "" || arg4 == "") {
                err = true;
            }
            else {
                // Hide bitstring
                hide(arg2, arg3, arg4);
            }
        }
        else if (arg1.equals("E")){
            // Extract bitstring from text
            retrieve(arg2);
        }
        else {
            err = true;
```

```
            }
        }
        else {
            err = true;
        }

        if (err == true) {
            System.out.println();
            System.out.println("Use: Stegano1 <A:E><Input
File><OutputFile><Bitstring>");
            System.out.println("Example: Stegano1 A inp.txt out.txt 0010101");
            System.out.println("Example: Stegano1 E inp.txt");

        }
    }

    static void hide(String inpFile, String outFile, String binString) {
        System.out.println("------\nHide function running...");

        System.out.println("\nThis is the bitstring we want to hide in the document: " +
binString);

        //
        BufferedReader reader;
        BufferedWriter writer;

        try {
            reader = new BufferedReader(new FileReader(inpFile));
            writer = new BufferedWriter(new FileWriter(outFile));
            String line = reader.readLine();
```

```
// Your code starts here
int index = 0;
String currentBit = "";
int binStringLength = binString.length(); // =7
System.out.println("\nBin string length = " + binStringLength);

//binString = "0010101";
String spaces = "err";

/*
 * This function will encrypt each bit into a line of the input file.
 * The first bit will be encrypted at the first line.
 * The second bit will be encrypted at the second line.
 * etc...
 *
 * If the bit is a 0, append 1 space to the end of the line.
 * If the bit is a 1, append 2 spaces to the end of the line.
 */

//For each line of the document...
while (line != null) {
    //Nothing will be added if there are no bits to encrypt.
    spaces = "";

    //... as long as there are still bits to encrypt...
    if(index < binStringLength)
    {
        //Get bit
        currentBit = Character.toString(binString.charAt(index));
```

```
        //Check if 1 or 2 spaces should be added.
        if(currentBit.equals("0"))
        {
            spaces = " ";
        }
        else if(currentBit.equals("1"))
        {
        spaces = "  ";
        }
        else
        {
            spaces = "error condition";
        }

    }



    // Store amended line in output file
    //... add the encrypted bit into the line of the file.
    writer.write(line + spaces);
    writer.newLine();

    // read next line
    line = reader.readLine();

    //increment index
    index++;
}
reader.close();
writer.close();
```

```
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }

        System.out.println("Output file updated with secret code in spaces.");

    }

    // use these arguments to test: "E", "out.txt"
    //Make sure that the file is encoded using this Stegano1 solution before trying to decode it.
    static void retrieve(String inpFile) {
        System.out.println("------\nRetrieve function running...");




        BufferedReader reader;
        String code = "";

        try {
            reader = new BufferedReader(new FileReader(inpFile));
            String line = reader.readLine();

            /*
             * 00 corresponds to 1 space
             * 01 corresponds to 2 spaces
             * 10 corresponds to 3 spaces
             * 11 corresponds to 4 spaces
             * 0 corresponds to 5 spaces
```

```
 * 1 corresponds to 6 spaces
 */
int lineLength = 0;
int index = 0;
int noSpaces = 0;





while (line != null)
{
    lineLength = line.length();
    index = lineLength;
    noSpaces = 0;



    // Count number of spaces at the end of a line
    // Start at the last index of the line string...
    // ... if that is a space increment spaces counter and move along
(decrement index counter).

    for(int i = 0; i < line.length(); i++)
    {
        if(Character.toString(line.charAt(index-1)).equals(" "))
        {
            noSpaces++;
            index = index - 1;
        }
    }

    //The number of spaces correspond with a different bit.
```

```
            // 1 space on a line = "0" in the code
            // 2 spaces on a line = "1" in the code

            switch(noSpaces)
            {
                case 1: code += "0"; break;
                case 2: code += "1"; break;
                default: break;

            }



            // System.out.println(line);

            // read next line
            line = reader.readLine();
        }
        reader.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    System.out.println("The retrieved code is: " + code);
  }
}
```

# Problem 1 Screenshots

```
BlueJ: Terminal Window - bibliobop
Options
------
Hide function running...

This is the bitstring we want to hide in the document: 0010101

Bin string length = 7
Output file updated with secret code in spaces.
------
Retrieve function running...
The retrieved code is: 0010101
------
```

# Problem 2

```
/**
 * CT255 - Assignment 3
 * Skeleton code for Steganography assignment.
 *
 * @author Michael Schukat, Maxwell Maia
 * @version 1.0
 */

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
```

```
public class Stegano2
{
    /**
     * Constructor for objects of class Stegano1
     */
    public Stegano2()
    {
    }

    //"A", "inp.txt", "out.txt", "0010101"
    public static void main(String[] args) {
        String arg1, arg2, arg3, arg4;
        Boolean err = false;

        if (args != null && args.length > 1) { // Check for minimum number of arguments
            arg1 = args[0];
            arg2 = args[1];

            if (arg2 == "") {
                err = true;
            }
            else if ((arg1.equals("A")) && (args.length > 3)){
                // Get other arguments
                arg3 = args[2];
                arg4 = args[3];
                if (arg3 == "" || arg4 == "") {
                    err = true;
                }
                else {
                    // Hide bitstring
```

```
                hide(arg2, arg3, arg4);
            }
        }
        else if (arg1.equals("E")){
            // Extract bitstring from text
            retrieve(arg2);
        }
        else {
            err = true;
        }
    }
    else {
        err = true;
    }

    if (err == true) {
        System.out.println();
        System.out.println("Use: Stegano1 <A:E><Input
File><OutputFile><Bitstring>");
        System.out.println("Example: Stegano1 A inp.txt out.txt 0010101");
        System.out.println("Example: Stegano1 E inp.txt");


    }
  }

  static void hide(String inpFile, String outFile, String binString) {
    System.out.println("------\nHide function running...");

    System.out.println("\nThis is the bitstring we want to hide in the document: " +
binString);
```

```
//My solution will store 2 bits at a time (per line)
/*
 * If we divdie up the bitstring...
 *
 * ...there are 6 possible combinations. They are storing a:
 * 00
 * 01
 * 10
 * 11
 * 0
 * 1
 *
 * I have chosen that each of these correspond with a certain number of
invisible charcters (spaces).
 * Only I know this correspondance which is what makes this encryption secret.
 *
 * 00 corresponds to 1 space
 * 01 corresponds to 2 spaces
 * 10 corresponds to 3 spaces
 * 11 corresponds to 4 spaces
 * 0 corresponds to 5 spaces
 * 1 corresponds to 6 spaces
 *
 * As it happens, my solution doesn't require a padding bit.
 */

//System.out.println("\nWith a padding digit added if the there are an odd
number of bits: " + binString);

//
BufferedReader reader;
```

```
BufferedWriter writer;

try {
    reader = new BufferedReader(new FileReader(inpFile));
    writer = new BufferedWriter(new FileWriter(outFile));
    String line = reader.readLine();

    // Your code starts here
    int index = 0;
    String currentBit = ""; //No longer needed.

    String firstBit = "";
    String secondBit = "";

    int binStringLength = binString.length(); // =7
    System.out.println("\nBin string length = " + binStringLength);

    //binString = "0010101";
    String spaces = "err";

    /*
     * This function will encrypt 2 bits at a time into a line of the input file.
     * The first 2 bits will be encrypted at the first line.
     * The second 2 bits will be encrypted at the second line.
     * etc...
     *
     * If the bit is a 0, append 1 space to the end of the line.
     * If the bit is a 1, append 2 spaces to the end of the line.
     */
```

```
        //the number of remaining bits.
        int remainingBits = binStringLength;

        //flag to stop runnning code for the last bit once the last bit has been
encrypted.
        boolean lastBitIsEncrypted = false;

        //For each line of the document...
        while (line != null) {
            //Nothing will be added if there are no bits to encrypt.
            spaces = "";

            if(remainingBits - 2 >= 0)
            {
                //We have 2 bits to encrypt
                //they are:
                //(binStringLength - remainingBits)
                //(binStringLength - (remainingBits - 1))

                //Get the first bit
                firstBit = Character.toString(binString.charAt( binStringLength -
remainingBits ));
                //Get the second bit
                secondBit = Character.toString(binString.charAt( binStringLength -
(remainingBits - 1) ));


                if(firstBit.equals("0"))
                {
                    //case where bits are: 00
                    if(secondBit.equals("0"))
```

```
    {
        spaces = " "; //00 corresponds to 1 space
    }

    //case where bits are: 01
    if(secondBit.equals("1"))
    {
        spaces = "  "; //01 corresponds to 2 spaces
    }
}
else if (firstBit.equals("1"))
{
    //case where bits are: 10
    if(secondBit.equals("0"))
    {
        spaces = "   "; //10 corresponds to 3 spaces
    }

    //case where bits are: 11
    if(secondBit.equals("1"))
    {
        spaces = "    "; //11 corresponds to 4 spaces
    }
}
else
{
    spaces = "error in condition";
}

remainingBits = remainingBits - 2;
```

```
        }
        else
        {
            if(!lastBitIsEncrypted)
            {
                //We are at the end of the string and there is only 1 bit to encrypt.
                //The one bit to encrypt is:
                //(binStringLength - (remainingBits))

                //Get the bit
                firstBit = Character.toString(binString.charAt( binStringLength -
remainingBits ));

                if(firstBit.equals("0"))
                {
                    //case where bit is : 0
                    spaces = "     "; //0 corresponds to 5 spaces
                }
                else if (firstBit.equals("1"))
                {
                    //case where bit is : 1
                    spaces = "      "; //1 corresponds to 6 spaces
                }
                else
                {
                    spaces = "error in condition";
                }

                //There will only ever be 1 bit at the end of the string one time.
                //So this boolean will prevent anymore occasions of this code running.
                lastBitIsEncrypted = true;
```

```
                }

            }

            // Store amended line in output file
            //... add the encrypted bit/(s) into the line of the file.
            writer.write(line + spaces);
            writer.newLine();

            // read next line
            line = reader.readLine();

            //increment index
            index++;
        }
        reader.close();
        writer.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }

    System.out.println("Output file updated with secret code in spaces.");

}


// use these arguments to test: "E", "out.txt"
```

```
    //Make sure that the file is encoded using this Stegano2 solution before trying to
decode it.
    static void retrieve(String inpFile) {
        System.out.println("------\nRetrieve function running...");




        BufferedReader reader;
        String code = "";

        try {
            reader = new BufferedReader(new FileReader(inpFile));
            String line = reader.readLine();

            /*
             * 00 corresponds to 1 space
             * 01 corresponds to 2 spaces
             * 10 corresponds to 3 spaces
             * 11 corresponds to 4 spaces
             * 0 corresponds to 5 spaces
             * 1 corresponds to 6 spaces
             */
            int lineLength = 0;
            int index = 0;
            int noSpaces = 0;



            while (line != null)
            {
                lineLength = line.length();
                index = lineLength;
```

```
noSpaces = 0;



// Count number of spaces at the end of a line
// Start at the last index of the line string...
// ... if that is a space increment spaces counter and move along
(decrement index counter).

for(int i = 0; i < line.length(); i++)
{
    if(Character.toString(line.charAt(index-1)).equals(" "))
    {
        noSpaces++;
        index = index - 1;
    }
}

//The number of spaces correspond to the a different 2 bit code piece.
switch(noSpaces)
{
    case 1: code += "00"; break;
    case 2: code += "01"; break;
    case 3: code += "10"; break;
    case 4: code += "11"; break;
    case 5: code += "0"; break;
    case 6: code += "1"; break;
    default: break;

}
```

```
            // System.out.println(line);

            // read next line
            line = reader.readLine();
        }
        reader.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    System.out.println("The retrieved code is: " + code);
  }
}
```

## Problem 2 Screenshots

```
------
Hide function running...

This is the bitstring we want to hide in the document: 0010101

Bin string length = 7
Output file updated with secret code in spaces.
------
Retrieve function running...
The retrieved code is: 0010101
```