2. Multithreading

Aim:

Implementing a multi-threaded problem for the given scenario

Program:

Government Agent Bob wants to transfer secret information to Agent Alice over the wired medium. But Bob is suspicious that someone is eavesdropping the wired medium. But to their favor, Bob and Alice have already agreed upon a secret code. Bob decides that he will replace every character in the sentence with the third character which appears in the alphabetical order (e.g. A will be replaced by D, b will be replaced by e). So when Alice receives the encrypted text, he can decode it.
Perform the following:
1. Write a menu-driven program to achieve encryption and decryption of the text.
2. Write a multithreaded program for the same.
3. Compare the time required for both.

Single thread implementation of Caesar Cipher:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;

public class SingleThreadedCasear {
    public static void main(String[] args) throws Exception {
        String home = System.getProperty("user.home");
        File ipFile = new File(home + "/Documents/plaintext.text");
        BufferedReader fileReader = new BufferedReader(new
FileReader(ipFile));
        String line;
        ArrayList<String> input = new ArrayList<>();
        ArrayList<String> cipher = new ArrayList<>();
        ArrayList<String> deciphered = new ArrayList<>();
        while((line = fileReader.readLine()) != null) {
            input.add(line);
        }
        long startTime = System.currentTimeMillis();
        for(String str: input) {
            cipher.add(encrypt(str));
        }
        for(String str: cipher) {
            deciphered.add(decrypt(str));
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time Required(in ms) = " +
(endTime-startTime));
    }

    public static String encrypt(String line) {
        int len = line.length();
        StringBuffer sb = new StringBuffer();
        for(int i = 0; i < len; i++) {
            char ch = line.charAt(i);
            if((int)ch == 32)
                sb.append(" ");
            else if(ch != 'X' && ch != 'x' && ch != 'Y' && ch != 'y' && ch
!= 'Z' && ch != 'z')
                sb.append((char)((int)ch+3));
            else if(ch == 'X')
                sb.append('A');
            else if(ch == 'x')
                sb.append('a');
            else if(ch == 'Y')
                sb.append('B');
            else if(ch == 'y')
                sb.append('b');
            else if(ch == 'Z')
                sb.append('C');
            else if(ch == 'z')
                sb.append('c');
            else if(ch == ' ')
                sb.append(' ');
        }
        return sb.toString();
    }

    public static String decrypt(String line) {
        int len = line.length();
        StringBuffer sb = new StringBuffer();
        for(int i = 0; i < len; i++) {
            char ch = line.charAt(i);
            if((int)ch == 32)
                sb.append(" ");
            else if(ch == 'A')
                sb.append('X');
            else if(ch == 'a')
                sb.append('x');
            else if(ch == 'B')
                sb.append('Y');
            else if(ch == 'b')
                sb.append('y');
            else if(ch == 'C')
                sb.append('Z');
            else if(ch == 'c')
                sb.append('z');
            else
                sb.append((char)((int)ch-3));
        }
        return sb.toString();
    }

}
```

Output:

```
Time Required(in ms) = 40
```

Multi-threaded implementation of Caesar Cipher:

```java
import com.google.common.collect.Lists;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;

public class MultiThreadedCeaserCipher {

    private BufferedReader fileReader;
    private static ArrayList<String> input = new ArrayList<>();
    private static ArrayList<Thread> threads = new ArrayList<>();
    private static List<List<String>> chunks = new ArrayList<>();
    public static List<List<String>> cipher = new ArrayList<>();
    public static List<List<String>> plaintext = new ArrayList<>();

    public static void main(String[] args) throws Exception {
        MultiThreadedCeaserCipher mulCipher = new
MultiThreadedCeaserCipher();
        mulCipher.loadInput();
        chunks = Lists.partition(input, 75);
        int numberOfThreads = chunks.size();
        System.out.println("Creating " + numberOfThreads + " threads.");
        mulCipher.initThreads(numberOfThreads);
        long startTime = System.currentTimeMillis();
        for(Thread thread: threads) {
            thread.start();
        }
        for(Thread thread: threads) {
            thread.join();
        }
        long endTime = System.currentTimeMillis();
        mulCipher.initDecryptThreads(numberOfThreads);
        long startTime1 = System.currentTimeMillis();
        for(Thread thread: threads) {
            thread.start();
        }
        for(Thread thread: threads) {
            thread.join();
        }
        long endTime1 = System.currentTimeMillis();
        System.out.println("Time Required(in ms) = " + (endTime-startTime +
endTime1-startTime1));
    }

    public void initThreads(int n) {
        System.out.println("IN initThreads(): Initializing thread
pool...");
        for(int i = 0; i < n; i++) {
            EncryptionThread newThread = new EncryptionThread("Encryption
Thread " + i);
            newThread.setData(chunks.get(i));
            threads.add(newThread);
        }
        System.out.println("IN initThreads(): Threads initialized...");
    }

    public void initDecryptThreads(int n) {
        threads.clear();
        System.out.println("IN initThreads(): Initializing decryption
thread pool...");
        for(int i = 0; i < n; i++) {
            DecryptionThread newThread = new DecryptionThread("Decryption
Thread " + i);
            newThread.setData(cipher.get(i));
            threads.add(newThread);
        }
        System.out.println("IN initThreads(): Decryption Threads
initialized...");
    }

    public void loadInput() throws Exception {
        String home = System.getProperty("user.home");
        File ipFile = new File(home + "/Documents/plaintext.text");
        this.fileReader = new BufferedReader(new FileReader(ipFile));
        System.out.println("IN loadInput(): FileReader initialized
successfully...");
        String line;
        while((line = fileReader.readLine()) != null) {
            this.input.add(line);
        }
        System.out.println("IN loadInput(): Input data loaded
successfully...");
    }

}

class EncryptionThread extends Thread {
    private List<String> data = new ArrayList<>();
    private List<String> cipher = new ArrayList<>();

    EncryptionThread(String name) {
        super(name);
    }

    public void setData(List<String> data) {
        this.data = data;
    }

    private void encrypt(String line) {
        int len = line.length();
        StringBuffer sb = new StringBuffer();
        for(int i = 0; i < len; i++) {
            char ch = line.charAt(i);
            if((int)ch == 32)
                sb.append(" ");
            else if(ch != 'X' && ch != 'x' && ch != 'Y' && ch != 'y' && ch
!= 'Z' && ch != 'z')
                sb.append((char)((int)ch+3));
            else if(ch == 'X')
                sb.append('A');
            else if(ch == 'x')
                sb.append('a');
            else if(ch == 'Y')
                sb.append('B');
            else if(ch == 'y')
                sb.append('b');
            else if(ch == 'Z')
                sb.append('C');
            else if(ch == 'z')
                sb.append('c');
            else if(ch == ' ')
                sb.append(' ');
        }
        this.cipher.add(sb.toString());
    }

    @Override
    public void run() {
        if(!this.data.isEmpty()) {
            for(String str: this.data) {
                encrypt(str);
            }
            MultiThreadedCeaserCipher.cipher.add(this.cipher);
        } else {
            System.out.println(Thread.currentThread().getName() + "IN
run(): Input data is empty");
        }
    }

}

class DecryptionThread extends Thread {
    private List<String> data = new ArrayList<>();
    private List<String> deciphered = new ArrayList<>();

    DecryptionThread(String name) {
        super(name);
    }

    public void setData(List<String> data) {
        this.data = data;
    }

    private void decrypt(String line) {
        int len = line.length();
        StringBuffer sb = new StringBuffer();
        for(int i = 0; i < len; i++) {
            char ch = line.charAt(i);
            if((int)ch == 32)
                sb.append(" ");
            else if(ch == 'A')
                sb.append('X');
            else if(ch == 'a')
                sb.append('x');
            else if(ch == 'B')
```

```
                sb.append('Y');
            else if(ch == 'b')
                sb.append('y');
            else if(ch == 'C')
                sb.append('Z');
            else if(ch == 'c')
                sb.append('z');
            else
                sb.append((char)((int)ch-3));
        }
        this.deciphered.add(sb.toString());
    }

    @Override
    public void run() {
        if(!this.data.isEmpty()) {
            for(String str: this.data) {
                decrypt(str);
            }
            MultiThreadedCeaserCipher.plaintext.add(this.deciphered);
        } else {
            System.out.println(Thread.currentThread().getName() + "IN
run(): Input cipher is empty");
        }
    }
}
```

Output:

```
IN loadInput(): FileReader initialized successfully...
IN loadInput(): Input data loaded successfully...
Creating 7 threads.
IN initThreads(): Initializing thread pool...
IN initThreads(): Threads initialized...
IN initThreads(): Initializing decryption thread pool...
IN initThreads(): Decryption Threads initialized...
Time Required(in ms) = 22
```

Conclusion:

Thus, I understood the concept of multithreading. For the given scenario, I implemented a solution using a single thread, as well as, multiple threads.
When the size of the input to the program was smaller, it was observed that the single-threaded solution took less time, as compared to a multithreaded solution. However, when the size of the input was increased fairly, the multithreaded solution proved to be more efficient than the single-threaded solution.
I also observed that both these solutions behaved differently, on different machines, depending upon the machine architecture and operating system details.