```java
package model;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

import thread.FirstThread;
import thread.SecondThread;

public class Decrypt {
    private Machine m;
    private String[] dico;
    private FirstThread first;
    private Thread thFirst;
/*  private SecondThread second;
    private Thread thSecond;*/


    public Decrypt(Machine mach,String file) throws IOException{
        BufferedReader b=null;
        m=mach;
        try {
             b = new BufferedReader(new FileReader(new File(file)));
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        String temp;
        dico = new String[604];
        int i=0;
        while((temp=b.readLine())!= null){
            dico[i]=temp.trim();
            i++;
        }
        initialiseThread();
    }

    /**
     * Constructeur par copie, les threads ne doivent pas partager les memes ressources!
     */
    public Decrypt(Decrypt d) throws IOException{
        this.m= new Machine(d.getMachine());
        this.dico=d.getDico();
        this.first=null;
        this.thFirst=null;
    }

    public String[] getDico(){
        return this.dico;
    }

    public Machine getMachine(){
        return this.m;
    }
```

```java
/**
 * Initialise les threads et les mets en pause dans leur methode run
 */
public void initialiseThread() throws IOException{
    this.first=new FirstThread(this);
    this.thFirst=new Thread(first);
    thFirst.start();
    /*this.second=new SecondThread(this);
    this.thSecond=new Thread(second);
    thSecond.start();*/
}

/**
 * Calcule les occurences de chaque lettres (a-z) dans une chaine
 *  @param s
 *          La chaine a analyser
 * @return int[] contenant toutes les occurences des differentes lettres
 */
public int[] calculOccurences(String s){
    ArrayList<Integer> n=new ArrayList<Integer>();
    ArrayList<Character> charVerifie=new ArrayList<Character>();
    int occurences;
    int j;
    int i=0;
    while(i<s.length()){

            occurences=0;
                j=s.indexOf(s.charAt(i));
                while(!charVerifie.contains(s.charAt(i)) && j!=-1){
                    occurences++;
                    j=s.indexOf(s.charAt(i),j+1);
                }
                if(occurences !=0){
                    n.add(occurences);
                }
                if(!charVerifie.contains(s.charAt(i))){
                    charVerifie.add(s.charAt(i));
                }
        i++;

    };


    int [] apparitions=new int[n.size()];
    for(i=0;i<n.size();i++){
        apparitions[i]=Integer.valueOf(n.get(i));
    }
    return apparitions;
}

/**
 * Calcule l'indice de coincidence d'une chaine: occurences-lettre/longueur-chaine
 *  @param s
 *          La chaine a calculer
 * @return l'indice de coincidence
 */
```

```
    public float calculIndiceCo(String s){
        int [] nbApparitions=calculOccurences(s);

        float indiceCo=0.0f;
        float num;
        for(int i=0;i<nbApparitions.length;i++){
            num=(nbApparitions[i]*(nbApparitions[i]-1));
            indiceCo+=num/(s.length()*(s.length()-1));
        }
        return indiceCo;

    }

    /**
     * Decrypte une chaine selon son indice de coincidence
     *  @param ch
     *          La chaine que l'on veut decryptee
     * @return La chaine decryptee
     */
    public String decrypterIc(String s) throws InterruptedException{
        int posR1=0;
        int posR2=0;
        int posR3=0;
        String ch="";
        rotorInitial();
        first.setString(s);//Je n'utilise qu'un thread pour le moment

        if(thFirst.isAlive()){
            first.unpause();
        }
        else{
            System.out.println("allo");
            this.thFirst=new Thread(first);
            thFirst.start();
            first.unpause();
        }
        thFirst.join(); //On attends l'arret du thread
        posR1=first.getPosRotors()[0];
        posR2=first.getPosRotors()[1];
        posR3=first.getPosRotors()[2];

        System.out.println(first.getIndice());

        //On regle a nouveau les rotors et on decrypte
        this.m.getRotor(0).avancer(this.m.CONVERT.length+posR1);
        this.m.getRotor(1).avancer(this.m.CONVERT.length+posR2);
        this.m.getRotor(2).avancer(this.m.CONVERT.length+posR3);

        ch=this.m.crypter(s);
        return ch;

    }

    /**
     * Decrypte une chaine si un mot francais est trouve
     *  @param ch
     *          La chaine que l'on veut decryptee
```

```
     * @return La chaine decryptee
     */
    public String decrypter(String s){
        String decryptee="impossible de decrypter la phrase";
        int trouve=-1;
        String mot="";
         afficherPos();
         // a tester avec une phrase contenant le mot message pour le moment sinon, ça prend
         trop de temps :/
        int i=0;
        while(i<this.dico.length && trouve==-1){
            System.out.println(dico[i]);
            rotorInitial();// A chaque mot, on remet les rotors a 0 0 0
            System.out.print("numero de mot : "+i+" /610");
            trouve=cryptagePossibleMot(dico[i],s);//on crypte toutes les possibilites pour
            le mot en cours
            i++;
        }
        if(trouve!=-1){
            mot=dico[i-1];
            System.out.println(mot);
            System.out.println("t "+trouve);
            System.out.println("avant reglage");
            regleRotor(s,mot,trouve);
            System.out.println("avant decryptage");
            afficherPos();
            decryptee=this.m.crypter(s);//Les rotors sont regles, on peut decrypter
            System.out.println(decryptee);
        }

        System.out.println("rotor finaux :");
        afficherPos();
        return decryptee;
    }


    /**
     * Met les rotors en position initiale (0 0 0)
     *
     */
    public void rotorInitial(){

        this.m.getRotor(0).avancer(this.m.CONVERT.length-(this.m.getRotor(0).getPosition()));
        this.m.getRotor(1).avancer(this.m.CONVERT.length-(this.m.getRotor(1).getPosition()));
        this.m.getRotor(2).avancer(this.m.CONVERT.length-(this.m.getRotor(2).getPosition
        ()));
    }

    /**
     * Test les 46^3 possibilites de cryptage pour chaque mot et verifie si il est contenu
     dans la chaine
     *  @param mot
     *          Le mot que l'on va crypte et cherche dans la chaine
     *  @param s
     *          La chaine de reference que l'on veut decrypter
     * @return La chaine decryptee
     */
```

```
    public int cryptagePossibleMot(String mot, String s){
        String m="";
            int r;
            for(int tRotor=0; tRotor<46;tRotor++){
                for(int sRotor=0; sRotor<46;sRotor++){
                    this.m.getRotor(2).avancer(this.m.CONVERT.length-(this.m.getRotor(2).
                    getPosition()-tRotor));
                    this.m.getRotor(0).avancer(this.m.CONVERT.length-(this.m.getRotor(0).
                    getPosition()));
                    for(int pRotor=0; pRotor<46;pRotor++){
                        this.m.getRotor(1).avancer(this.m.CONVERT.length-(this.m.getRotor(1).
                        getPosition()-sRotor));
                        if(mot.equals("jeune")){
                        System.out.println(this.m.getRotor(0).getPosition());
                        }
                         m=this.m.crypter(mot);
                         if(this.m.getRotor(0).getPosition()==17){
                             System.out.println(m);
                             System.out.println(s);
                         }
                         r=s.indexOf(m);
                         if(r!=-1){
                             System.out.println("trouve "+"r "+r);
                             return r;
                         }

                         this.m.getRotor(0).avancer(this.m.CONVERT.length-(m.length()-1));
                    }
                     this.m.getRotor(1).avancer(1);
                }
                 this.m.getRotor(2).avancer(1);
            }

             return -1;
    }

    /**
     * Regle les rotors en fonction de la longueur de la chaine
     * Fais reculer les rotors afin d'avoir la bonne position de depart
     */
    public void regleRotor(String s, String m, int pos){
        System.out.println("pos "+this.m.getRotor(0).getPosition());
        this.m.getRotor(0).avancer(this.m.CONVERT.length-(s.substring(0, pos).length()+m.
        length()));
        if(this.m.getRotor(0).getPosition()>=46-s.length()){
            this.m.getRotor(1).avancer(this.m.CONVERT.length-1);
            if(this.m.getRotor(1).getPosition()==45){
                this.m.getRotor(2).avancer(this.m.CONVERT.length-1);
            }
        }
    }

    /**
     * Affiche la position des trois rotors
     */
    public void afficherPos(){
        System.out.println(this.m.getRotor(0).getPosition());
```

```
        System.out.println(this.m.getRotor(1).getPosition());
        System.out.println(this.m.getRotor(2).getPosition());
    }


}
```