CSIC Computing Project

Vigenere Cipher

Final Report *December 2005*

Project co-ordinator: Dr. Naqvi

Team members:
Maryam Salar Kalantari,
Masood Khosroshahy,
Yong Wu

Table of Contents:

1) Introduction to the project
2) What is requested to be carried out
3) Requirements capture
4) System specification using DFDs
5) Detailed UML Class Diagrams
6) Our Development Environment
7) Classes, their methods and variables, as depicted in Eclipse
8) Program Behavior
9) How the Vigenere table is utilized
10) Program Execution: -Ciphering/Deciphering

Appendix: Source Codes

1) Introduction to the project

A cipher is a message written in a secret code. Cryptography, in a very broad sense, is the study of techniques related to aspects of information security. Hence cryptography is concerned with the writing (ciphering or encoding) and deciphering (decoding) of messages in secret code. Such considerations emphasize privacy and confidentiality of information. The *Vigenere cipher* is a cipher based on using successively shifted alphabets, a different shifted alphabet for each of the 26 English letters.

2) What is requested to be carried out:

Writing a program in Java that ciphers and deciphers messages. The system will take a text file, a keyword, and the desired operation (ciphering or deciphering) as inputs from its user.
The output of the system consists of the creation and display of original and ciphered/deciphered files.

3) Requirements Capture:

Regardless of which method of ciphering/deciphering is going to be utilized, here are the Functional Requirements of the system:

A)-An input interface which accepts a source file, a keyword and the intended operation (Ciphering/Deciphering)

B)-Outputting the file (Saving on disk) after performing the intended operation

4) System specification using DFDs:

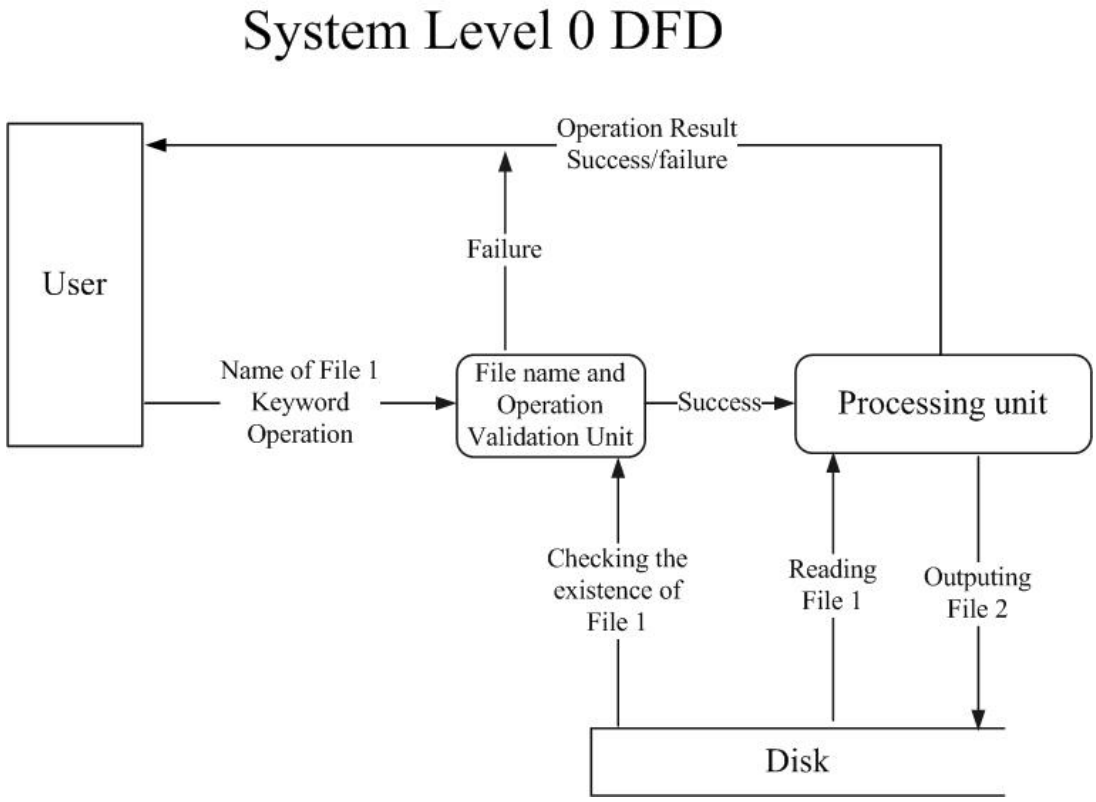

Figure 1. System Level 0 DFD

4-1) Explanation of “System Level 0 DFD”:

The “System Level 0 DFD” is depicted in Figure 1. Execution of the program starts with user typing the name of the program along with 3 arguments which are: “Name of File 1”(Input file), “Keyword” and “Operation”. These arguments are passed to the “Filename and Operation Validation Unit”. This unit checks whether the provided file name is a valid one by checking the disk. Also, it checks to make sure that the requested operation is either Ciphering or Deciphering. If the checks fail, the matter is reported to the user, otherwise processing moves to “Processing Unit”.

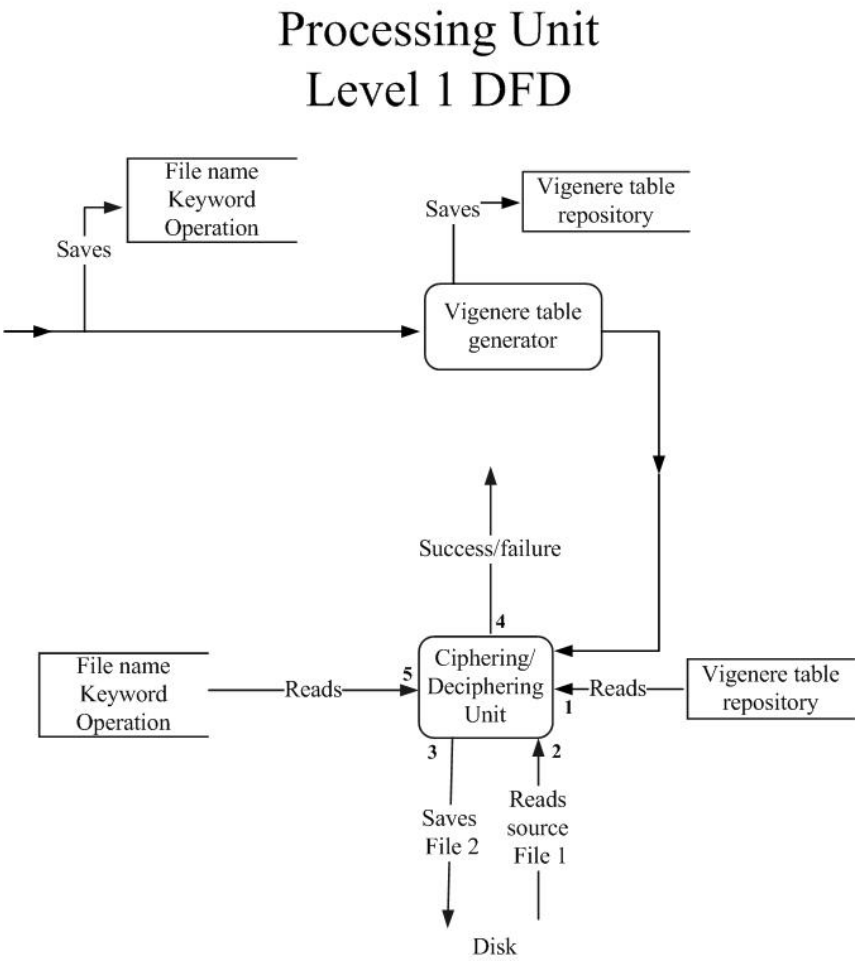

Figure 2. Processing Unit Level 1 DFD

4-2) Explanation of “Processing Unit Level 1 DFD”:

The “Processing Unit Level 1 DFD” is depicted in Figure 2. At the “Processing Unit”, the inputs, which are Filename, Keyword & Operation, are saved. Then, the “Vigenere Table Generator” activates and generates the table and saves it in a 2-dimensional array in memory, characterized as “Vigenere Table Repository” in the figure.
The operation then moves on to “Ciphering/Deciphering Unit”. This unit, in order to perform its operation, reads the saved inputs, on Port 5, reads the source file from disk, on Port 2, consults the Vigenere Table, on port 1, and finally saves the output of the processing on disk, on Port 3. The success or failure of the whole operation will be reported to the user, on Port 4.

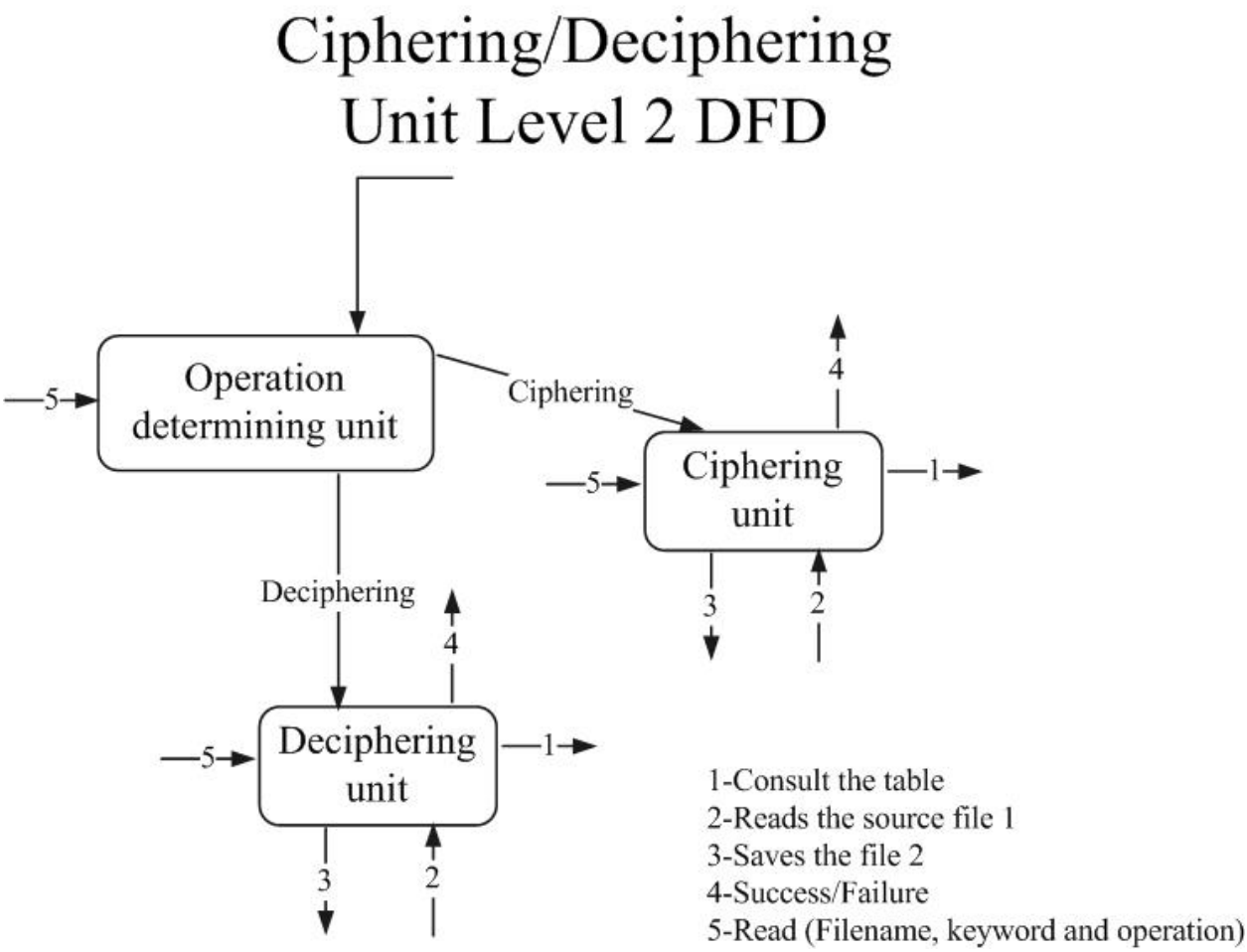

Figure 3. Ciphering/Deciphering Unit Level 2 DFD

4-3) Explanation of "Ciphering/Deciphering Unit Level 2 DFD":

This figure shows the inside of "Ciphering/Deciphering Unit". Here, the type of the operation, Ciphering or Deciphering, is checked first in "Operation determining unit" and then the processing job is dispatched to the appropriate unit.

5) Detailed UML Class Diagrams:

Figure 4 demonstrates the UML class diagrams, designed by Eclipse and Omondo UML Plug-In. As can be seen, the program is implemented using 4 classes: "Vigenere" (Which contains the main() function), "CipheringDeciphering", "Ciphering" and "Deciphering". The relations between the classes are as follows:
"Ciphering" and "Deciphering" classes are derived from "CipheringDeciphering" class. The common functionalities of these classes have been implemented in the "CipheringDeciphering" class. The relation between "Vigenere" class and "CipheringDeciphering" class are of type Association, so is the type of relation between the "Vigenere" class and the "Ciphering" class and the "Deciphering" class.

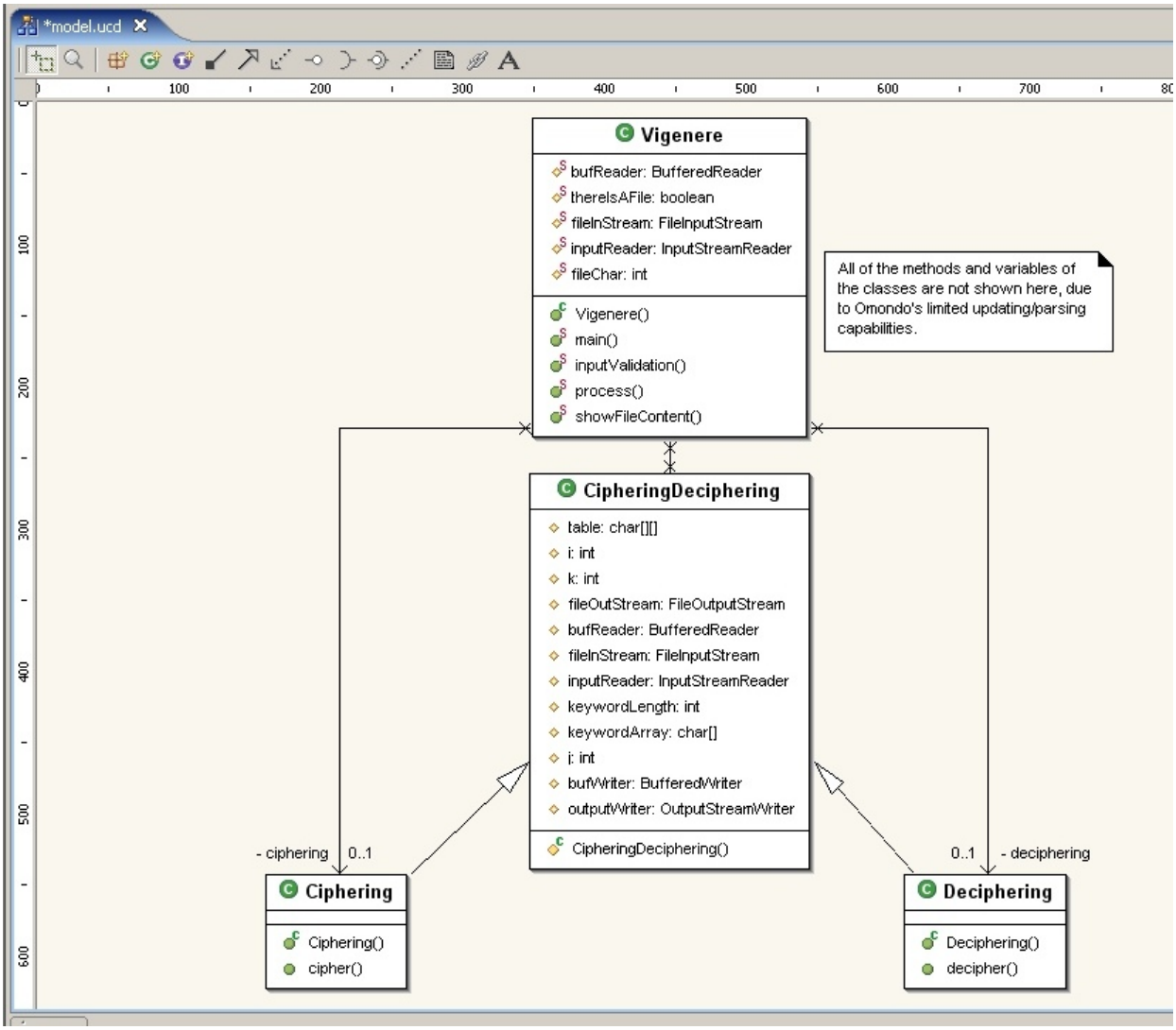

Figure 4. UML Class Diagrams

6) Our Development Environment:

Figure 5 is a snapshot of the Eclipse IDE and the utilized Omondo Plug-In. We first intended to utilize an all open source configuration (Eclipse + Unimod UML Plug-In). But, due to the limited functionalities of Unimod, we switched to Omondo which is of course a commercial product.

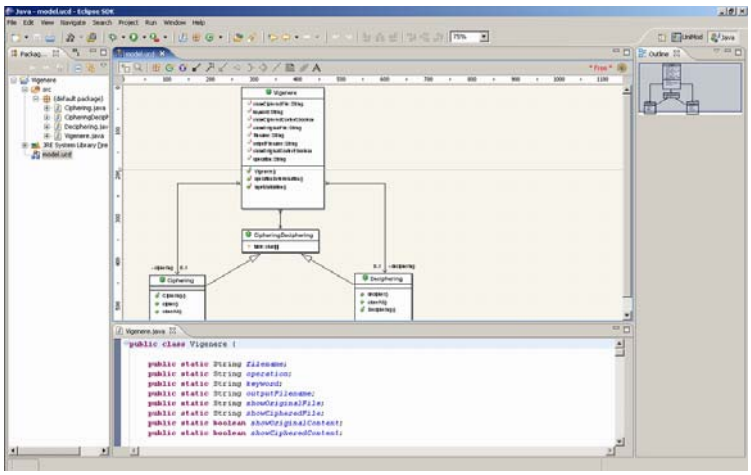

Figure 5. Eclipse/Omondo Development Environment

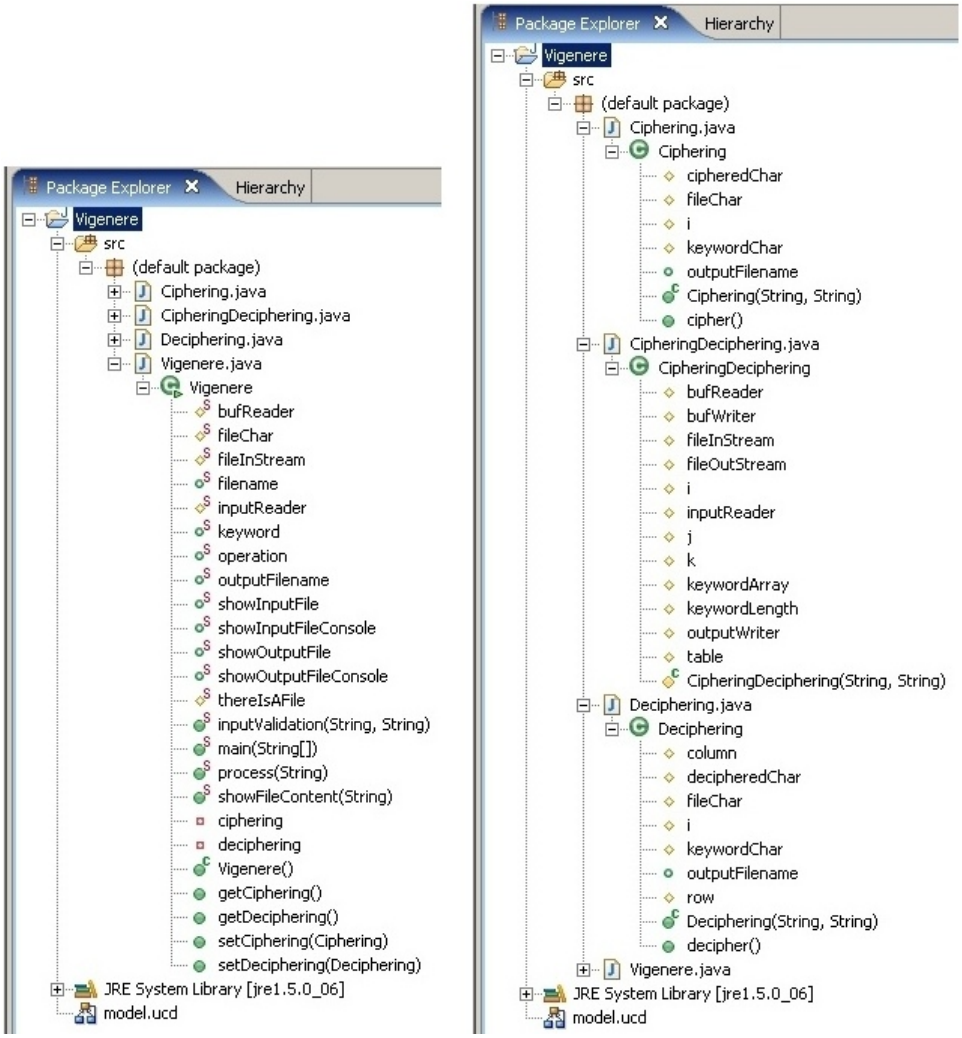

Figure 6. Eclipse Package Explorer –Depicting classes, their methods and variables

7) Classes, their methods and variables, as depicted in Eclipse:

Figure 6 is the Package Explorer view of the Eclipse IDE. In this view, all the classes, their methods and variables are depicted. As can been seen, there are 4 classes. The view of the class Vigenere is expanded on the left-hand side and those of the other classes are expanded in the right-hand side of the figure.

8) Program Behavior:

The program starts with the following line entered:
java Vigenere filename.txt operation keyword
The program takes the file “filename.txt” and, according to the mentioned operation and keyword, processes it. Several checks are done before execution:

- The number of arguments is checked to be correct
- The existence of the input file is checked
- It is made sure that the operation is a valid one
- Asking whether or not the user likes to see the input file on the screen before processing and the output file, after processing (In both modes of Ciphering and Deciphering)

Since each module does the opposite of the other module, each can be considered as the *test-module* of the other one.

9) How the Vigenere table is utilized:(A trimmed-down version)

In Figure 7, a trimmed-down version of Vigenere table is shown. In the implementation of the program, a 2-dimensional array comprising all the ASCII code has been utilized. But here, for demonstration purposes, we use the following table. For better explanation of the operation, we take an example which is shown in Figure 8. In the first column of this example, we demonstrate how we can derive the character "Z" by knowing the "J" and "Q" characters (Ciphering) and how we can derive the character "Q" by knowing the "J" and "Z" characters.
Ciphering: We look up the character "J" in the first row to locate the column. Then, we look up the character "Q" in the first column to locate the row. The intersection of the found column and row indicates the corresponding ciphered character which is the character "Z".
Deciphering: For deciphering, again we look up the first row using the keyword character, "J", to locate the column. In this column, we search for the ciphered character, "Z", when found, we retrieve the first character of the row which the found "Z" is located.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 7. A simplified Vigenere table

Keyword	J	U	N	G	L		E	J	U	N	G		L	E	J
Original text	Q	U	I	C	K		B	R	O	W	N		F	O	X
Ciphered text	Z	O	V	I	V		F	A	I	J	T		Q	S	G

Figure 8. An example for ciphering/deciphering using the table

10) Program Execution: -*Ciphering/Deciphering*

In this section, we provide some snapshots of our program in action. After entering the first line giving the program the necessary information regarding filename, operation and keyword, the program walks the user through some steps asking whether or not they like to see the file contents on the screen. The Figure 9 and Figure 12 are self-explanatory. Figure 10 and Figure 11 show the output of the program in ciphering mode and deciphering mode, respectively.

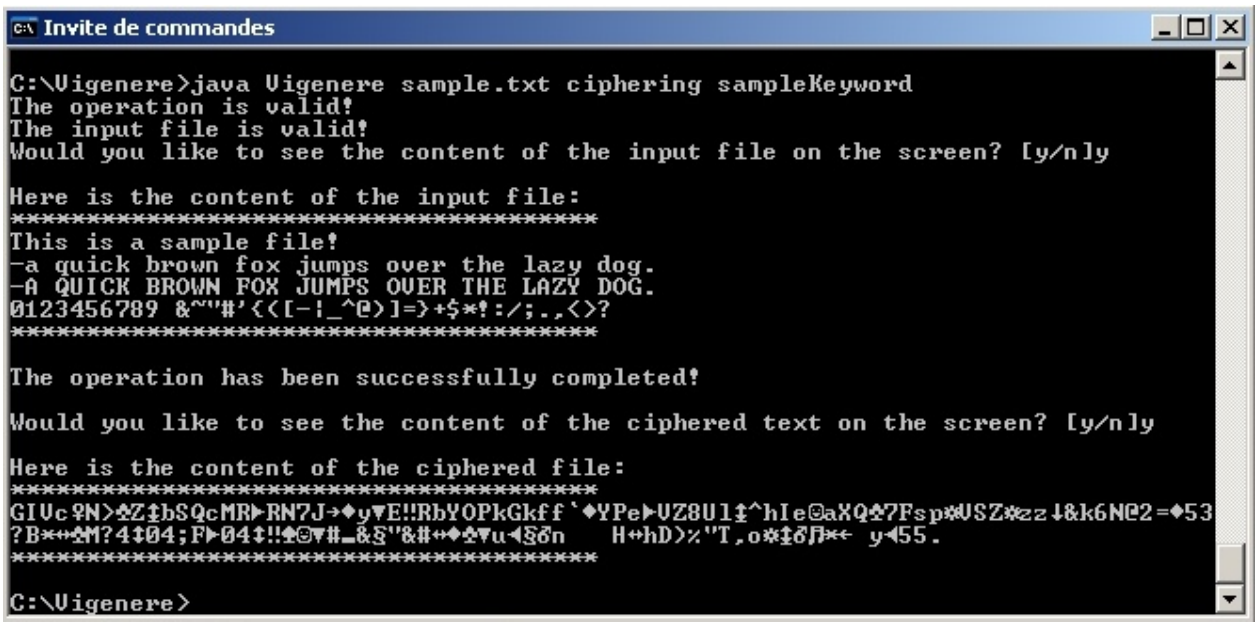

Figure 9. Program output –Ciphering

Figure 10. Produced file –Ciphering

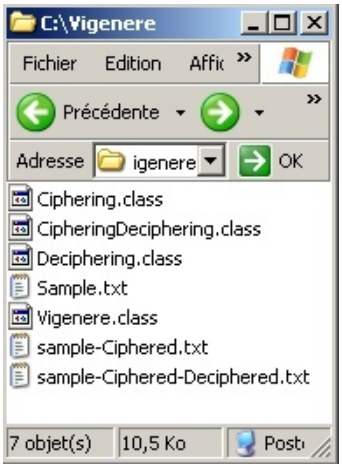

Figure 11. Produced file –Deciphering

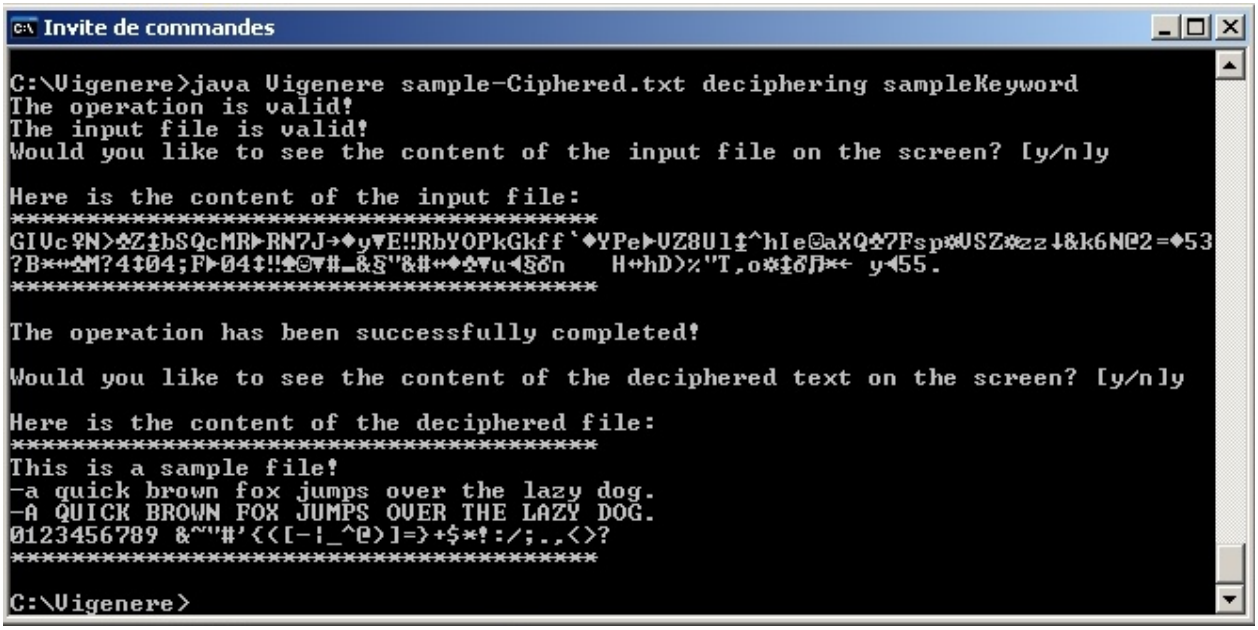

Figure 12. Program output -Deciphering

Appendix :

Class Vigenere

```
import java.io.*;

public class Vigenere {

	public static String filename;
	public static String operation;
	public static String keyword;
	public static String outputFilename;
	protected static int fileChar;  // used for storing one character at a time from the file read from the disk
	protected static boolean thereIsAFile = true;  // used for checking the existence of the file on the disk

	// The following 4 variables are used for storing the answer of the user to the question
	//of whether or not they wish to see the file content on screen
	public static String showInputFileConsole;
	public static String showOutputFileConsole;
	public static boolean showInputFile;
	public static boolean showOutputFile;

	//The following 3 objects are used for setting up the mechanism of reading the file from disk
	protected static FileInputStream fileInStream;
	protected static InputStreamReader inputReader;
	protected static BufferedReader bufReader;

	//These objects have been automatically created by Omondo. Their existence is rather unnecessary.
	private Ciphering ciphering;
	private Deciphering deciphering;

	//The class constructor
	public Vigenere() {
		super();
	}
	public static void main(String[] args) {
		try{
			//The program takes 3 arguments and saves them in the following 3 variables.
			filename = args[0];
			operation = args[1];
			keyword = args[2];

			//This function checks the validity of the operation and exitence of the file on the disk.
			inputValidation(filename, operation);

		//The main functionality of the program and interaction with the user are implemented in this function. See below for details.
			process(operation);
			}
		//The following exceptions have been handled as seen below. Basically, they inform the user of the error encountered.
		catch (FileNotFoundException e1) {System.out.println("There has been a disk related error.");}
		catch (IOException e2) {System.out.println("There has been an input/output error.");}
		catch (ArrayIndexOutOfBoundsException e3){System.out.println("There has been a problem in the " +
				"way the arguments have been entered. Please use the following format:");
		System.out.println("java Vigenere filename.txt operation keyword");
		System.out.println("In which the operation can be either \"ciphering\" or \"deciphering\".");}
	}

	public static void inputValidation(String filename, String operation)throws FileNotFoundException,IOException{
		try{
			//Here, we check the operation validity.
			if (operation.equals("ciphering") || operation.equals("deciphering"))
				System.out.println("The operation is valid!");
			else{
				System.out.println("The operation is invalid!");
				System.out.println("The operation can be either \"ciphering\" or \"deciphering\".");
				System.out.println("Please use the following format: java Vigenere filename.txt operation keyword");
			}
		//Here, we try to open the file, filename of which is provided by user. The file is immediately closed. File existence checking.
			FileInputStream temp = new FileInputStream(filename);
			temp.close();
			System.out.println("The input file is valid!");
			}
```

```
		catch (FileNotFoundException e1) {
			System.out.println("There is no such a file on the disk!");
			// If the file doesn't exist, this value is set accordingly.
			//procees() function reads this value before execution.
			thereIsAFile = false;
			}
}
public static void process(String operation)throws FileNotFoundException,IOException{

		if (thereIsAFile == false) {return;}

//Here, we set up the mechanism by which we read the user input to the question of whether or not they want to see the file content.
		BufferedReader console = new BufferedReader(new InputStreamReader(System.in));

		//If the operation is ciphering, we move to this block of code which first checks to see if the user wants to see the content of
		//file before proceeding. Then, the file is ciphered. At the end, the user is asked to see whether or not they like to see the
		//content of the ciphered file.
		if (operation.equals("ciphering")){
			try {
				System.out.print("Would you like to see the content of the input file on the screen? [y/n]");
				 showInputFileConsole= console.readLine();
				 if (showInputFileConsole.equals("y") || showInputFileConsole.equals("Y")){
					showInputFile = true;
					System.out.println("\nHere is the content of the input file:");
					//The following function shows the content of the file. It is defined below.
					showFileContent(filename);
				 }
				 else if (showInputFileConsole.equals("n") || showInputFileConsole.equals("N"))
					 showInputFile = false;
				 else
					 System.out.println("Please enter either \"y\" or \"n\"");

			}catch (IOException e) {System.out.println("Please enter either \"y\" or \"n\""); }

		//Here, an instance of the class Ciphering is created and the filename and the operation are passed to its constructor.
			Ciphering ciphering = new Ciphering(filename , keyword);

			//The cipher() function of the ciphering object is called to begin the ciphering operation.
			ciphering.cipher();

			System.out.print("Would you like to see the content of the ciphered text " +
					"on the screen? [y/n]");
			showOutputFileConsole= console.readLine();
			 if (showOutputFileConsole.equals("y") || showOutputFileConsole.equals("Y")){
				showOutputFile = true;
				System.out.println("\nHere is the content of the ciphered file:");
				showFileContent(ciphering.outputFilename);
			 }
			 else if (showOutputFileConsole.equals("n") || showOutputFileConsole.equals("N"))
				 showOutputFile = false;
			 else
				 System.out.println("Please enter either \"y\" or \"n\"");
		}

		//If the intended operation is Deciphering, we move to this block of code. The sequence of the actions is exactly
		//like those of the ciphering operation described above.
		else if (operation.equals("deciphering")){
			try {
				System.out.print("Would you like to see the content of the input file on the screen? [y/n]");
				 showInputFileConsole= console.readLine();
				 if (showInputFileConsole.equals("y") || showInputFileConsole.equals("Y")){
					showInputFile = true;
					System.out.println("\nHere is the content of the input file:");
					showFileContent(filename);
				 }
				 else if (showInputFileConsole.equals("n") || showInputFileConsole.equals("N"))
					 showInputFile = false;
				 else
					 System.out.println("Please enter either \"y\" or \"n\"");

			}catch (IOException e) {System.out.println("Please enter either \"y\" or \"n\""); }

			Deciphering deciphering = new Deciphering(filename , keyword);

			deciphering.decipher();
```

```
                        System.out.print("Would you like to see the content of the deciphered text " +
                                "on the screen? [y/n]");
                        showOutputFileConsole= console.readLine();
                        if (showOutputFileConsole.equals("y") || showOutputFileConsole.equals("Y")){
                                showOutputFile = true;
                                System.out.println("\nHere is the content of the deciphered file:");
                                showFileContent(deciphering.outputFilename);
                        }
                        else if (showOutputFileConsole.equals("n") || showOutputFileConsole.equals("N"))
                                showOutputFile = false;
                        else
                                System.out.println("Please enter either \"y\" or \"n\"");
                }
        }

        public static void showFileContent(String filename)throws FileNotFoundException,IOException{

                //The following 3 objects are created for setting up the mechanism by which we read the file.
                fileInStream =new FileInputStream(filename);
                inputReader = new InputStreamReader(fileInStream);
                bufReader = new BufferedReader(inputReader);

                System.out.println("****************************************");

                while( (fileChar = bufReader.read())!= -1){
                        System.out.print((char)fileChar);
                        };

                System.out.println("\n****************************************");

                //After finishing reading the file, we delete the set up mechanism.
                bufReader.close();
                inputReader.close();
                fileInStream.close();
        }

        /* Automatically created by the Omondo plug-in. These are not used in the program.
        These has to be kept in order for the UML model to show the correct associations. */

        public Ciphering getCiphering() {
                return ciphering;
        }
        public void setCiphering(Ciphering ciphering) {
                this.ciphering = ciphering;
        }

        public Deciphering getDeciphering() {
                return deciphering;
        }
        public void setDeciphering(Deciphering deciphering) {
                this.deciphering = deciphering;
        }
}
```

Class CipheringDeciphering

```
import java.io.*;

public class CipheringDeciphering {
	protected char[][] table;   //The table is created below and stored in this 2-dimensional character array.
	protected char[] keywordArray;  //The String keyword is converted to a character array for better utilization.
	protected int i , j, k=0;  //Internal temporary variables used in the creation of the table.
	protected int keywordLength;  //Used for the coversion of the keyword type from String to Character Array

	// The following 6 objects are defined for use in setting up the mechanism by which
	//we read from file (the first 3) and write to a file on disk (the last 3).
	protected FileInputStream fileInStream;
	protected InputStreamReader inputReader;
	protected BufferedReader bufReader;
	protected FileOutputStream fileOutStream;
	protected OutputStreamWriter outputWriter;
	protected BufferedWriter bufWriter;

	//This is the constructor of this class. The procedures within are automatically executed
	//upon creation of the objects of type Ciphering and Deciphering, which are derived from this class.
	protected CipheringDeciphering(String filename, String keyword)throws FileNotFoundException,IOException{
		super();

		keywordLength = keyword.length();
		keywordArray = keyword.toCharArray();

		fileInStream =new FileInputStream(filename);
		inputReader = new InputStreamReader(fileInStream);
		bufReader = new BufferedReader(inputReader);

		// Table creation procedure and saving it in the variable "table" defined above

		table = new char[128][128];
				for (i=0; i<table.length; i++)
					{
						for(j=0; j<table[i].length; j++)
							{
								if( (k+j) >= 128)
									table[i][j] =(char)(j+k-128);
								else
									table[i][j]=(char)(j+k);
							}
						k++;
					}
		}

}
```

Class Ciphering

```
import java.io.*;

public class Ciphering extends CipheringDeciphering {

	protected int cipheredChar;   //Used for storing the derived ciphered character.
	protected int fileChar; //Used for storing the read character from the input file.
	protected int i = 0 ;  //A counting variable
	protected char keywordChar; //Used for storing 1 character from the keyword

	public String outputFilename;  //Output filename is constructed and stored in this variable

	//This is the constructor of this class
	public Ciphering(String filename, String keyword) throws FileNotFoundException,IOException{
		//The filename and keyword are first passed to the parent class's constructor
		super(filename, keyword);

		outputFilename = filename.substring(0, filename.length()-4) + "-Ciphered" +".txt";

		//Here, we setup the mechanism for storing the ciphered characters on disk
		fileOutStream =new FileOutputStream(outputFilename);
		outputWriter = new OutputStreamWriter(fileOutStream);
		bufWriter = new BufferedWriter(outputWriter);
	}

	public void cipher()throws FileNotFoundException,IOException{

		while( (fileChar = bufReader.read())!= -1){
			//1 character is read from the keyword. If that character was the last character of the keyword,
			//we loop back to the first character of the keyword.
			keywordChar = keywordArray[i];
			i++;
			if (i == keywordArray.length)
				i = 0;
			try{
				//In the following 2 lines, the ciphered character is found and written to disk.
				cipheredChar = table[fileChar][(int)(keywordChar)];
				bufWriter.write(cipheredChar);
			}catch(ArrayIndexOutOfBoundsException e){bufWriter.write(fileChar);};
		};
		System.out.println("\nThe operation has been successfully completed!\n");

		//After finishing the processing, the resources are released.
		bufReader.close();
		inputReader.close();
		fileInStream.close();

		bufWriter.close();
		outputWriter.close();
		fileOutStream.close();
	}
}
```

Class Deciphering

```
import java.io.*;

public class Deciphering extends CipheringDeciphering {

	protected int decipheredChar;  //A variable used for storing the deciphered character.
	protected int fileChar;  //A variable used for storing the read character from file.
	protected int row; // Indicating the row number of the Vigenere table
	protected int column; //Indicating the column number of the Vigenere table
	protected int i = 0 ; //A counting variable
	protected char keywordChar; //A variable used for storing 1 character from keyword

	public String outputFilename;   // Output filename is constructed and stored in this variable.

	//This is the constructor of this class
	public Deciphering(String filename, String keyword) throws FileNotFoundException,IOException{
		//The filename and keyword are passed to the constructor of parent class of this class.
		super(filename, keyword);

		outputFilename = filename.substring(0, filename.length()-4) + "-Deciphered" +".txt";

		//Setting up the mechanism by which we write to file.
		fileOutStream =new FileOutputStream(outputFilename);
		outputWriter = new OutputStreamWriter(fileOutStream);
		bufWriter = new BufferedWriter(outputWriter);
	}

	public void decipher()throws FileNotFoundException,IOException{

		while( (fileChar = bufReader.read())!= -1){
			//1 character is read from the keyword. If that character was the last character of the keyword,
			//we loop back to the first character of the keyword.
			keywordChar = keywordArray[i];
			i++;
			if (i == keywordArray.length)
				i = 0;

			//In deciphering, the keyword character indicates the column number.
			column = (int)keywordChar;
			//And then we search the found column for the file character. After finding it,
			//we read the row number and then the deciphering is done.
			for (int x= 0 ; x <128 ; x++){
				if (table[x][column] == fileChar)
					row = x;
			}
			decipheredChar = row;
			//We write the deciphered character to file.
			bufWriter.write(decipheredChar);
		};
		System.out.println("\nThe operation has been successfully completed!\n");

		//We release the resources after finishing the operation.
		bufReader.close();
		inputReader.close();
		fileInStream.close();

		bufWriter.close();
		outputWriter.close();
		fileOutStream.close();
	}
}
```