10 | Vigenère

Manuel Alcántara Juárez
Verónica Esther Arriola Ríos

En esta práctica vamos a implementar el método clásico de cifrado Vigenère para llevar a cabo una comunicación confidencial, utilizando para ello lectura y escritura de archivos.

Meta

Que el alumno aprenda a escribir y leer archivos en el disco duro.

Objetivos

Al finalizar la práctica el alumno será capaz de:

1. Manejar el concepto de *flujo*.
2. Crear y escribir archivos de texto.
3. Leer archivos de texto.

Antecedentes

Entrada y salida en Java

Definición 10.1: Flujo

Un **flujo** es una secuencia de datos. Un programa utiliza un **flujo de entrada** para leer datos desde una fuente, uno a la vez, y un **flujo de salida** para enviar datos a un destino, también uno a la vez. *I/O Streams* 2016

Java distingue entre dos formas de manejar esa información: binaria y como caracteres. Un grupo de clases lee y escribe bytes, mientras que otro está preparado para trabajar con diferentes codificaciones de texto. Las clases más comunmente utilizadas de ambos paquetes están ilustradas en Figura 15.1.

Para esta práctica necesitarás estar consultando constantemente la documentación de la API de Java, así que ten la dirección a la mano: http://docs.oracle.com/javase/8/docs/api/.

Recibiendo entrada desde la terminal

Si quieres hacer un programa con interfaz de usuario basada en texto, como el bash que se ejecuta en la terminal, necesitas acceder al flujo de entrada de la terminal. Java hace esto a través del objeto estático System.in. Éste es un InputStream y lee bytes. No queremos tener que decodificar cada caracter que teclee nuestro usuario byte por byte. El tipo de objeto que puede ayudarnos con ese trabajo es un Scanner. Si al Scanner le damos un flujo de entrada cuando lo construímos, él se encargará de hacer la decodificación por nosotros y devolvernos cadenas de caracteres (String). A continuación se incluye un pequeño ejemplo de cómo utilizarlo para este fin.

Actividad 10.1

Revisa la documentación oficial de java.util.Scanner. ¿Qué es un objeto tipo File?

```java
import java.util.Scanner;

/**
 * Demo básico sobre el uso de la clase Scanner.
 * @author blackzafiro
 */
public class DemoScanner {

  /**
    * Recibe una línea de texto desde el teclado y
    * reimprime el mensaje que recibió, hasta que el usuario
    * se despida.
    * @param args
    */
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    while(s.hasNext()) {
      String linea = s.nextLine();
      System.out.println("Eco: " + linea);
      if(linea.equals("Adios")) break;
    }
```

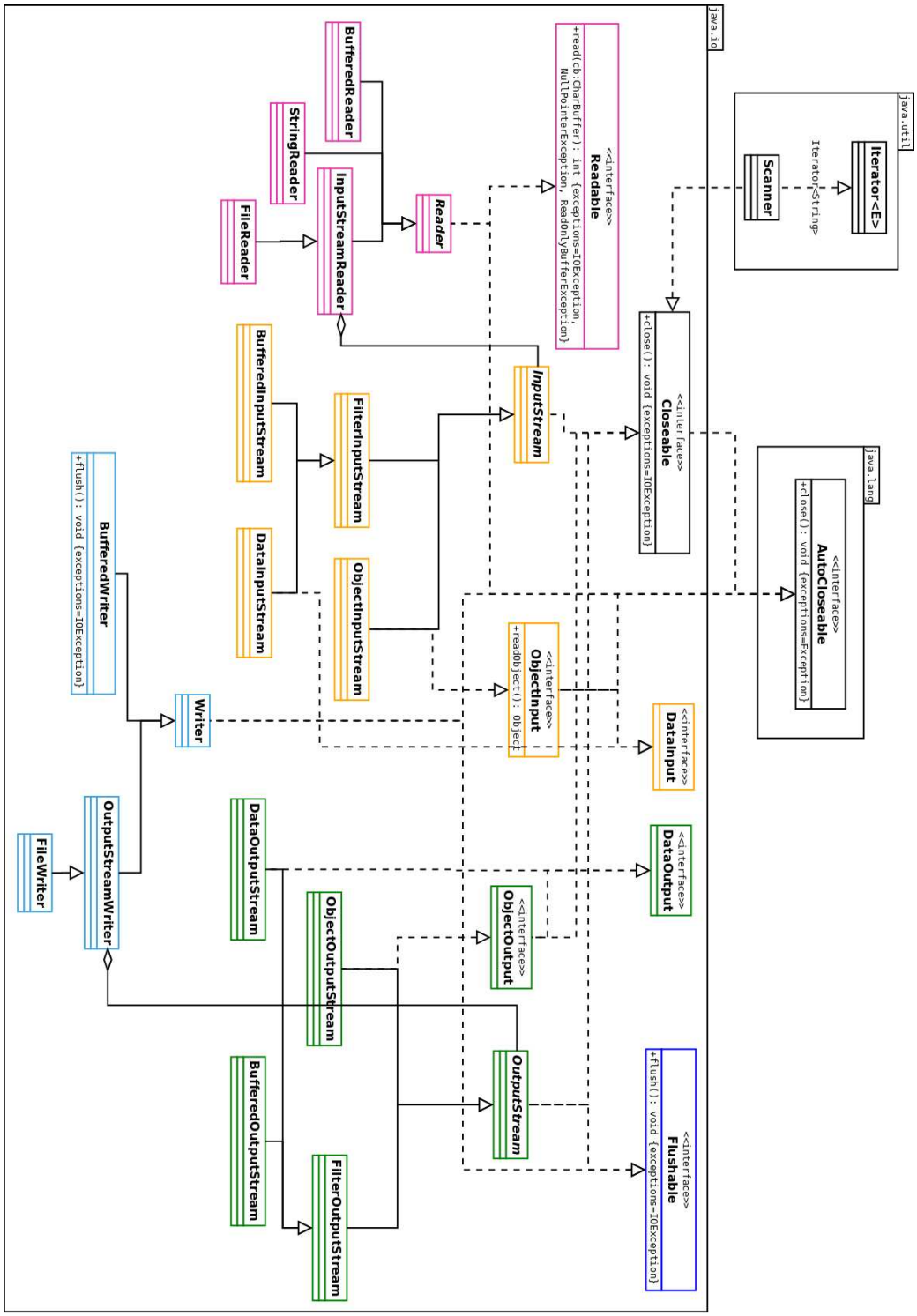

Figura 10.1 Diagrama de clases UML de algunas clases para entrada/salida en `Java`.

```
22      s.close();
23    }
24
25  }
```

Actividad 10.2

Copia y ejecuta el código anterior. Para hacerlo más rápido que con `ant`, cópialo en un archivo `DemoScanner.java` e invoca al compilador escribiendo `javac DemoScanner.java`. El archivo `.class` se generará ahí mismo. Ejecútalo con `java DemoScanner`.

Cambia la línea `18` para usar el método `next()` en lugar de `nextLine()`. ¿Qué hace ahora el programa?

Leyendo archivos de texto

Un `BufferedReader` se parece a un `Scanner`, pero se especializa en archivos. Hay cosas que puedes hacer en un archivo, que no podrías hacer en la terminal, como regresar a la línea anterior o leer todo el archivo en una sola pasada. El demo siguiente lee un archivo de texto llamado `texto.txt` y lo imprime línea por línea. Asegúrate de crear uno al lado de este código, para que puedas probarlo. Observa que si olvidas hacerlo, este método lanza una excepción, para avisar que el archivo no está y el método `main` la cacha, para convertirla en un mensaje más amigable con el usuario.

```
1  import java.io.BufferedReader;
2  import java.io.FileNotFoundException;
3  import java.io.FileReader;
4  import java.io.IOException;
5
6  /**
7   * Demo básico sobre el uso de FileReader y BufferedReader.
8   * @author blackzafiro
9   */
10 public class DemoBufferedReader {
11   /**
12     * Lee un archivo de texto e imprime su contenido línea por línea
         ↪ .
13     * @param args
14     */
15   public static void main(String[] args) {
16     try {
17       BufferedReader in = new BufferedReader(new FileReader("texto.
           ↪ txt"));
18       String line;
19       while((line = in.readLine()) != null) {
20         System.out.println(line);
```

```
21          }
22          in.close();
23        } catch (FileNotFoundException e) {
24          System.err.println("No se encontró el archivo texto.txt ¿
              ↪ Olvidaste crearlo?");
25        } catch (IOException ioe) {
26          System.err.println("Error al leer el contenido de texto.txt");
27        }
28      }
29
30  }
```

Escribiendo archivos de texto

Para escribir archivos de texto utilizaremos un `PrintStream`. Esta clase desciende de `OutputStream` y de hecho la has usado frecuentemente ¿Recuerdas a `System.out`? Pues bien, éste es un objeto de tipo `PrintStream`. Así que siéntete como en casa. El ejemplo de abajo indica cómo crear uno que dirija la salida a un archivo, en lugar de a la terminal. Observa que la sintaxis del `try` es diferente: la creación del objeto que puede lanzar las excepciones se encuentra entre paréntesis. Esto ayudará también a que el objeto se cierre solo al terminar de ejecutar el bloque de código dentro del `try`. También puedes utilizar esta notación con los otros flujos.


```
1   import java.io.FileNotFoundException;
2   import java.io.PrintStream;
3
4   /**
5    * Demo básico sobre el uso de PrintStream.
6    * @author blackzafiro
7    */
8   public class DemoPrintStream {
9     /**
10      * Escribe el texto indicado en un archivo.
11      * @param args
12      */
13    public static void main(String[] args) {
14      String nombreArchivo = "Salida.txt";
15      try (PrintStream fout = new PrintStream(nombreArchivo)) {
16        fout.println("Inicio");
17        fout.format("Línea %d\n", 1);
18        fout.println("Fin");
19      } catch (FileNotFoundException fnfe) {
20        System.err.println("No se encotró el archivo " + nombreArchivo
              ↪  + " y no pudo ser creado");
21      } catch (SecurityException se) {
22        System.err.println("No se tiene permiso de escribir en el
              ↪ archivo");
23      }
```

```
24      }
25
26  }
```

Actividad 10.3

Copia y ejecuta el código anterior.

Revisa la documentación de `PrintStream`. Observa que si ejecutas el programa otra vez, se borra el contenido del archivo y se sobreescribe. Si quieres abrir un archivo para agregarle cosas, en lugar de borrarlo y escribir de nuevo, necesitarás usar un `FileWriter` o un `FileOutputStream` e indicar en el constructor que quieres usar la opción `append`.

El cifrador Vigenère

Se le denominó así en honor al francés Blaise de Vigenère, aunque el primero que lo propuso en realidad fue Giovan Battista Belasso. Este cifrado consiste en lo siguiente: se crea una matriz de 26 x 26 y se rellena escribiendo en cada fila un alfabeto que se irá desplazando a la izquierda de uno en uno, e identificamos a cada fila y columna con una letra como se muestra en la Figura 10.2.

Después se decide una palabra clave y se va repitiendo ininterrumpidamente, de forma que el texto a cifrar y la palabra clave tengan la misma longitud. Por ejemplo, para el texto plano

```
ESTO ES UN SECRETO
```

y la clave

```
PUERTA
```

ambos se emparejan del modo siguiente:

```
ESTOESUNSECRETO
PUERTAPUERTAPUE
```

Para cifrar el texto utilizando la Tabla de Vigenere se procede de la siguiente manera:

- Nos posicionamos en la columna que pertenezca al texto y en la fila que pertenezca a la clave y el elemento que contenga dicha intersección será la letra que cifre dicho par.

  Por ejemplo la letra P cifrada con la clave E nos daría el criptograma T.

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

Figura 10.2 Tabla auxiliar para el cifrado de Vigenère.

- Se repite el procedimiento para cada letra.

  Siguiendo esto como ejemplo nuestro texto plano cifrado quedaría representado de este modo:

  TMXFXSJHWVVRTNS

Para descifrar un mensaje, es necesario volver a pedir la llave y completarla al tamaño del texto. Por ejemplo:

TMXFXSJHWVVRTNS
PUERTAPUERTAPUE

Posteriormente ir al renglón de la palabra clave y buscar la letra del texto cifrado; la columna será el resultado. En nuestro caso tenemos que ir al renglón P y buscar la letra T, a la cual le corresponde la columna E.

Desarrollo

Se te entregan dos clases con código auxiliar para esta práctica: `Main.java`, con una interfaz de usuario la cual desplegará un menú que muestra las opciones cifrar o descifrar un archivo en consola, y `Vigeniere.java`, con el esqueleto para implementar la aplicación de cifrado/descifrado.

Ejercicios

1. Completa la clase Vigeniere que será la responsable de cifrar o descrifrar un mensaje.
2. Posteriormente ejecuta tu programa. Este pide la ruta de un archivo en donde se encuentre el texto plano (archivo `.decoded`) o el texto cifrado (archivo `.encoded`) y finalmente solicita la clave. El programa debe de escribir un archivo con el mismo nombre pero la extensión contraria, ya sea con el texto cifrado o descifrado según la opción elegida en el menú.

NOTA 1: Para cifrar utiliza como entrada archivos con extension .decoded, que en realidad será un txt para proporcionar el texto plano. El programa debe producir un archivo en la misma ruta pero con extensión .encoded. El caso contrario aplica cuando se descifra un archivo.

NOTA 2: Los caracteres que no estén entre a-z deben quedar sin modificación.

Un caso de prueba: `Mesaje.encoded`

UWA JKGV NQ TQIZCUBG! AI ECAK VMTOQPCA GN KWTAQ FM KEK.

SECRETO: `icc`