

Guía para la implementación en Java

El **objetivo** de este documento es proporcionar unas pautas que faciliten la implementación de los programas a partir de los algoritmos diseñados. Se proporcionarán las pautas para la implementación utilizando primero la librería JTF¹ de ACM y posteriormente la funcionalidad estándar.

Este documento no es una guía completa de la implementación. Para obtener información más detallada se recomienda leer los apuntes de la asignatura o la documentación de *Java*.

Se quiere desarrollar un programa que solicite al usuario un número entero positivo y calcule y muestre el factorial de dicho número. Como sólo se puede calcular el factorial de *números naturales*, el programa antes de realizar el cálculo debe comprobar que el dato introducido es válido. Además, hay que tener en cuenta que el factorial se calcula mediante productos de números enteros por lo que se puede producir un error por desbordamiento (*overflow*)². Para evitar este error, el programa solicitará que el valor introducido por el usuario esté en el rango de valores $[0, 15]$. Si el dato introducido por el usuario no es válido, el programa le indica que es un dato incorrecto y vuelve a pedírselo al usuario. Esta comprobación se repite hasta que el usuario introduce un dato válido.

Las operaciones que debe realizar el programa se dividen en los siguientes dos bloques:

¹ <http://www-cs-faculty.stanford.edu/~eroberts/jtf/>

² Se obtiene un valor no representable con los bits disponibles, por lo que el valor obtenido no es válido. Podríamos obtener un valor negativo en lugar del positivo esperado.



1. Obtener un dato de entrada válido.
2. Calcular el factorial. El factorial se puede calcular mediante la fórmula

$$fact(n) = \prod_{i=1}^n i$$

Para facilitar la modificación del programa, se utilizarán constantes tanto para la definición de los mensajes a mostrar como de las cotas del rango de valores admitidos. A continuación se muestra el diseño del programa a implementar, tanto mediante *pseudocódigo* como utilizando un *diagrama de flujo*.

Pseudocódigo

AlgoritmoFactorial

Datos

Const VALOR_MIN \leftarrow 0, VALOR_MAX \leftarrow 15 : Entero
Const MENS_DATOS \leftarrow "Introduce un número entero [" &
VALOR_MIN & "," & VALOR_MAX & "]: Cadena
Const MENS_ERROR \leftarrow "Número incorrecto": Cadena
factorial, numUsuario, i: Entero

Empieza

numUsuario \leftarrow **leeEntero** (MENS_DATOS)
Repetir mientras numUsuario < VALOR_MIN
 OR numUsuario > VALOR_MAX
 Escribe (MENS_ERROR)
 numUsuario \leftarrow **leeEntero**(MENS_DATOS)

Fin Repetir

factorial \leftarrow 1

Para cada i **entre** 1 **y** numUsuario

 factorial \leftarrow factorial * i

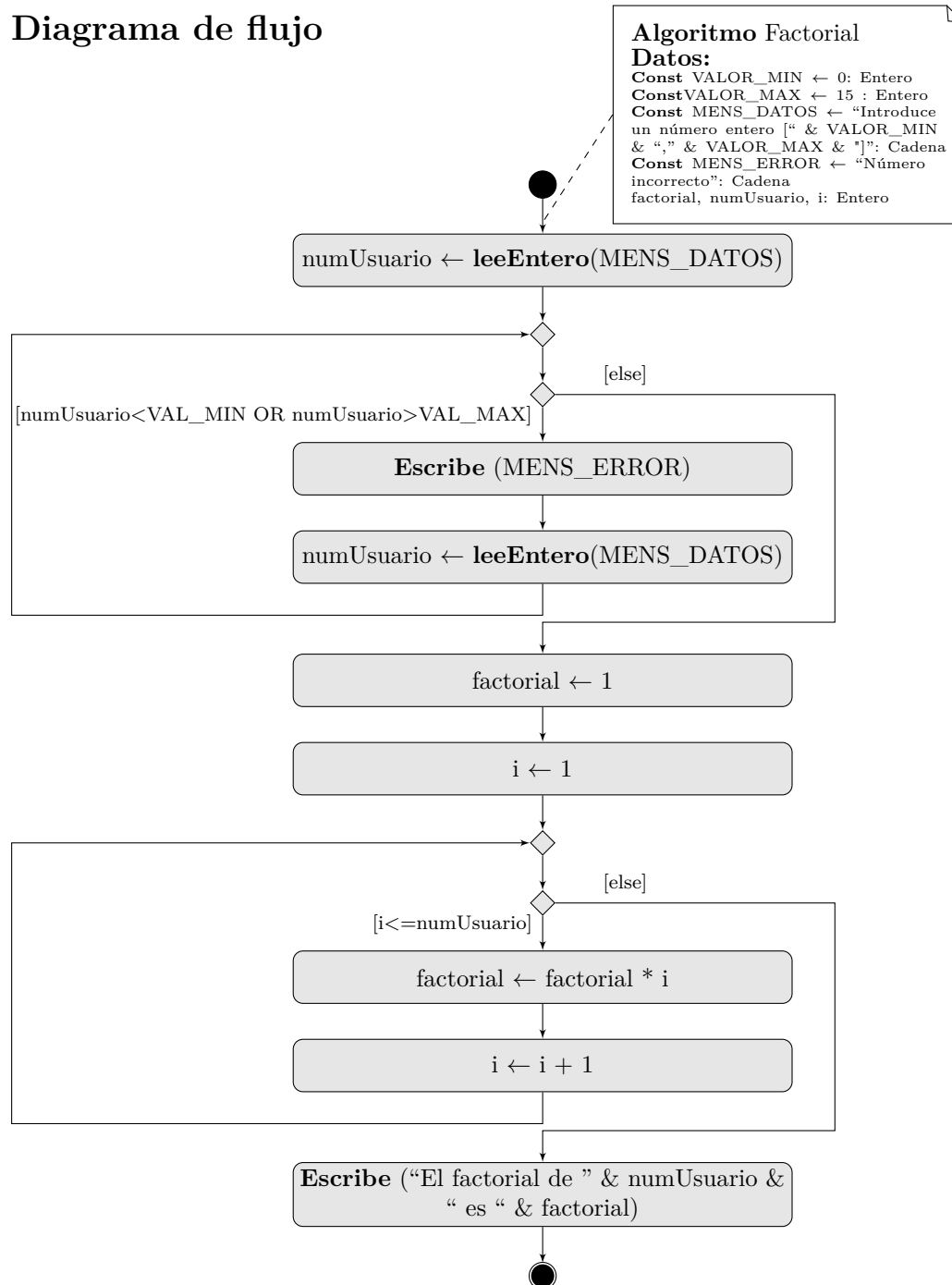
Fin Para Cada

Escribe ("El factorial de " & numUsuario & " es " & factorial)

Fin



Diagrama de flujo





1. Implementación utilizando la librería JTF

Para la implementación, se utilizará la librería *JTF* de *ACM*, que ha sido desarrollada con el ánimo de facilitar el aprendizaje de los principios básicos de programación simplificando algunos aspectos de *Java* que pueden resultar complejos para los neófitos.

A continuación se indica cómo se implementan cada uno de los elementos del algoritmo.

1.1. Implementación del preámbulo

Todo algoritmo comienza por un preámbulo (ver Tabla 1) en el que se indica cómo se llama el algoritmo.

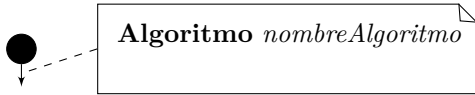
Pseudocódigo	Diagrama de flujo
 Algoritmo <i>nombreAlgoritmo</i>	Algoritmo <i>nombreAlgoritmo</i>

Tabla 1: Preámbulo de un algoritmo

La implementación del preámbulo en *Java* utilizando la librería JTF es:

```
import acm.program.DialogProgram;  
public class NombreClase extends DialogProgram {  
    public void run() {  
  
    }  
}
```

La implementación del algoritmo se realizará en un fichero de texto que tiene la estructura indicada en el recuadro anterior. La primera instrucción (`import acm.program.DialogProgram;`) indica que se va a utilizar la funcionalidad adicional proporcionada en dicha clase. Se puede incorporar más funcionalidad si es necesario con cláusulas `import` adicionales.

En la segunda línea hay que sustituir el *NombreClase* por el nombre que hemos dado a nuestro algoritmo. Por convención, los nombres deben



comenzar por mayúscula (ver la normativa para la nomenclatura). El nombre debe coincidir con el nombre del fichero.

La tercera línea define el programa que vamos a implementar. Siempre va a tener el mismo nombre (**run**). Las llaves permiten definir bloques en los que se van a anidar líneas de código. Conviene tabular las líneas para poder determinar a que instrucción se refiere cada una de las llaves y cuál es su par.

Aunque los programas que vamos a desarrollar por el momento son muy sencillos y podemos incluirlo todo en un único fichero, las aplicaciones pueden ser muy complejas y contener un gran número de ficheros con el código fuente. *Java* permite organizar estos ficheros en *paquetes* o *carpetas*. Si optamos por utilizar paquetes, debemos incluir una instrucción del tipo:

```
package nombrePaquete;
```

El preámbulo del algoritmo que estamos implementando sería el siguiente:

```
package packFactorial;
import acm.program.DialogProgram;
public class MostrarFactorial extends DialogProgram {
    public void run() {

    }
}
```

Sólo falta completar el programa (espacio entre el par de llaves correspondientes al subprograma **run**) con la definición de datos y las instrucciones identificadas en el algoritmo.

1.2. Definición de datos

La definición de los datos se realiza, normalmente, al inicio del subprograma. La Tabla 2 muestra las equivalencias entre los tipos de datos utilizados en los algoritmos con los disponibles en *Java*.

Como se puede observar, algunos tipos de datos tienen más de un equivalente. Los números enteros, por ejemplo, se pueden representar utilizando **byte** (1 byte), **short** (2 bytes), **int** (4 bytes) o **long** (8 bytes). Igualmente, los



Guía de implementación en Java

números reales se pueden representar mediante `float` (4 bytes) o `double` (8 bytes). Hay que utilizar el tipo adecuado en función del rango de valores con los que trabajará la aplicación. En la asignatura, normalmente utilizaremos `int` para los números enteros y `float` para los reales.

Tipo de datos	Tipo de datos en java
entero	byte short int long
real	float double
boolean	boolean
caracter	char
cadena	String

Tabla 2: Tipos de datos

1.2.1. Definición de variables

Las definiciones de las variables se realizan de la siguiente manera:

```
tipo nombreVariable;
```

Por convención, los nombres de las variables deben comenzar por minúscula (ver la normativa para la nomenclatura). La instrucción para definir una variable se puede completar con la asignación del valor inicial³ para la variable, tal como muestra a continuación:

```
tipo nombreVariable = valorInicial;
```

³ Recuerda que toda variable debe tener asignado un valor previo para poder realizar un cálculo a partir de ella.



Observa que el símbolo para la asignación en *Java* es `=`.

Si hay varias variables del mismo tipo, su definición se puede agrupar separando los nombres de las variables con comas. La definición de las variables del algoritmo sería:

```
int numUsuario, factorial;
```

1.2.2. Definición de constantes

La definición de las constantes es similar a la de las variables, con la excepción de que hay que indicar que no se puede modificar su valor y obligatoriamente hay que asignarle un valor inicial.

```
final tipo NOMBRE_CONSTANTE = valorConstante;
```

Las definiciones de las constantes del algoritmo quedarían:

```
final int VAL_MIN = 0;
final int VAL_MAX = 15;
final String MENS_DATOS = "Introduce un número entero [" +
    VAL_MIN + "," + VAL_MAX + "]:";
final String MENS_ERROR = "Número incorrecto";
```

Como se puede ver, el valor de la constante se puede obtener a partir de una expresión. En este algoritmo, el valor de la constante `MENS_DATOS` se obtiene mediante la concatenación de cadenas de caracteres y los valores de las constantes utilizando el operador `+`.

1.3. Implementación de las operaciones de E/S

La librería `JTF` simplifica las operaciones de entrada/salida. Veamos cómo se implementa cada una de las operaciones.



1.3.1. Implementación de las operaciones de salida

La librería JTF proporciona la operación `println` para escribir mensajes en la salida estándar. Esta operación muestra el mensaje especificado en una ventana de diálogo.

En general, utilizaremos la operación `println`, salvo cuando se vaya construyendo el mensaje paso a paso. La operación identificada en el algoritmo se implementaría:

```
println("El factorial de " + numUsuario + " es " +  
        factorial);
```

La librería JTF también proporciona una operación para visualizar los mensajes de error (`showErrorMessage`). En el algoritmo hay una operación para indicar que el número introducido por el usuario no es adecuado. Dicha operación se implementaría así:

```
showErrorMessage(MENS_ERROR);
```

1.3.2. Implementación de las operaciones de entrada

Para implementar las operaciones de lectura, hay que tener en cuenta las equivalencias entre las operaciones que proporciona la librería JTF y las operaciones que utilizamos en nuestros algoritmos (ver Tabla 3).

Operaciones de lectura	Operaciones de lectura en JTF
leeEntero	<code>readInt(<i>mensaje</i>)</code>
leeReal	<code>readDouble(<i>mensaje</i>)</code>
leeBooleano	<code>readBoolean(<i>mensaje</i>)</code>
leeCadena	<code>readLine(<i>mensaje</i>)</code>

Tabla 3: Tipos de datos

Las operaciones para la lectura de datos proporcionadas por la librería JTF permiten indicar el mensaje que queremos que se muestre al usuario. La lectura identificada en el algoritmo se implementaría:



```
numUsuario = readInt(MENSAJE);
```

Recuerda que los valores leídos hay que almacenarlos para poder procesarlos posteriormente, ya que si no esa información se pierde.

1.4. Implementación de estructuras iterativas

En el algoritmos se han utilizado dos tipos de estructuras iterativas distintas. Veamos cómo se implementa cada una de ellas.

1.4.1. Implementación de estructuras Para Cada

La estructura **Para Cada** (ver Tabla 4) permite repetir un conjunto de instrucciones un número determinado de veces. Utilizan una variable que realiza las labores de contador tomando todos los valores en el rango $[valIni, valFinal]$. Este tipo de estructuras permiten especificar el incremento (p.ej. para obtener la secuencia de valores pares en el rango especificado). Aunque en la tabla se muestra la estructura que permite obtener la secuencia de valores de manera creciente, se pueden obtener de mayor a menor.

Pseudocódigo	Diagrama de flujo
<pre>nombreVal ← valIni ... [else] [nombreVal ≤ valFinal] ... nombreVal ← nombreVal + valIncr</pre>	<pre>Para cada nombreVal entre valIni y valFinal [inc valIncr] ... Fin para cada</pre>

Tabla 4: Estructura iterativa Para Cada



Para la implementación de esta estructura, *Java* proporciona la instrucción `for`, que tiene la forma:

```
for (inicialización; condición; incremento){  
    ...  
}
```

Se deben tener en cuenta varios detalles. La instrucción `for` no va seguida por el `;` como la mayoría de las instrucciones. Es una estructura que agrupa el conjunto de instrucciones que se van a repetir entre el par de llaves. En la instrucción `for` debemos especificar la *inicialización* ($\text{nombreVal} \leftarrow \text{valIni}$), la *condición* que se debe cumplir para ejecutar repetidamente las operaciones ($\text{nombreVal} \leq \text{valFinal}$) y, por último el *incremento* que determina cómo se modifica la variable que controla el proceso de una iteración a otra ($\text{nombreVal} \leftarrow \text{nombreVal} + \text{valIncr}$).

Teniendo esto en cuenta, el bucle que permite calcular el factorial se implementaría:

```
factorial = 1;  
for (int i= 1; i<= numUsuario; i++){  
    factorial *= i;  
}
```

En la implementación hemos utilizado dos operaciones especiales: `i++` es una abreviatura de `i = i + 1` mientras que `factorial*=i` equivale a `factorial = factorial * i`.

1.4.2. Implementación de estructuras Repetir

La estructura iterativa **Repetir** (ver Tabla 5) permite que el conjunto de instrucciones se repita en distintas circunstancias (incluidas las soportadas por la expresión **Para Cada**). Recuerda que alguna de las operaciones agrupadas por la estructura **Repetir** debe modificar alguna de las variables que se utilizan en la condición. De lo contrario, podríamos toparnos con un *bucle infinito*.



Pseudocódigo	Diagrama de flujo
	<p>Repetir mientras <i>condición</i></p> <p>...</p> <p>Fin repetir</p>

Tabla 5: Estructura iterativa Repetir

En *Java*, la estructura iterativa **Repetir** se implementa mediante la instrucción **while** que se muestra a continuación:

```
while (condición){  
    ...  
}
```

Utilizando esta estructura, las operaciones que permiten controlar que el valor introducido por el usuario se implementarían:

```
numUsuario = readInt(MENSAJE);  
while (numUsuario<= VAL_MIN  
    || numUsuario >VAL_MAX){  
    showErrorMessage(MENS_ERROR);  
    numUsuario = readint(MENSAJE);  
}
```

El operador **||** es el equivalente al operador booleano **OR** en *Java*, mientras que **&&** es el equivalente al **AND**.



1.5. Implementación completa

Uniendo todo lo que hemos visto en los apartados anteriores, obtendríamos la siguiente implementación para nuestro algoritmo.

```
package packFactorial;
import acm.program.DialogProgram;

public class MostrarFactorial extends DialogProgram {
    public void run() {
        final int VAL_MIN = 0;
        final int VAL_MAX = 15;
        final String MENSAJE = "Introduzca un número
                                entero [" + VAL_MIN + "," + VAL_MAX + "]:";
        final String MENS_ERROR = "Número incorrecto";
        int numUsuario, factorial;

        /* Obtener el valor del que se quiere
           calcular el factorial */
        numUsuario = readInt(MENSAJE);
        while (numUsuario < VAL_MIN || numUsuario > VAL_MAX) {
            showErrorMessage(MENS_ERROR);
            numUsuario = readInt(MENSAJE);
        }
        // Calcular Factorial
        factorial = 1;
        for (int i = 1; i <= numUsuario; i++) {
            factorial *= i;
        }
        println("El factorial de " + numUsuario + " es " +
                factorial);
    }
}
```



2. Implementación utilizando la funcionalidad estándar de Java

La librería JTF, como hemos indicado antes, se ha desarrollado para simplificar el aprendizaje de *Java*. El uso de dicha librería implica modificaciones en el preámbulo, pero permite que nos olvidemos de algunos detalles como la conversión de datos y el tratamiento de errores que pueden resultar tediosos para los neófitos en la materia. Sin embargo, esta librería no se utiliza normalmente en el desarrollo de aplicaciones reales, por lo que en esta sección veremos cómo se implementan los algoritmos con la funcionalidad *Java* estándar.

Para facilitar el aprendizaje, sólo se indicarán los aspectos que cambian respecto a la implementación utilizando la librería JTF (preámbulo y operaciones de entrada/salida). La implementación de las operaciones conlleva el tratamiento de los posibles errores que se pudieran dar al realizar ese tipo de operaciones. Como el tratamiento de errores se verá con más detalle la asignatura de *Programación Modular y Orientación a Objetos*, realizaremos tratamientos muy básicos⁴.

2.1. Implementación del preámbulo

El preámbulo cambia ligeramente respecto al que debemos especificar al utilizar la librería JTF. El siguiente fragmento recoge estos cambios.

```
import acm.program.DialogProgram;  
public class NombreClase extends DialogProgram {  
    public void run() {  
  
    }  
}
```

Si no se utiliza la librería JTF, no hace falta que se indique su uso en el preámbulo, por lo que las partes resaltadas sobran y no se especificarán. Observa que el subprograma `run` también está tachado. Ese subprograma es el que se ejecuta cuando lanzamos una aplicación implementada con la

⁴ El programa se limitará a detectar los errores si se producen e indicar de esta circunstancia



librería JTF, pero no existe en las librerías estándar de *Java*. En su lugar, debemos utilizar el subprograma `main` que se ilustra a continuación⁵:

```
public static void main (String args[]) {  
  
}
```

Otro aspecto que se debe tener en cuenta es si el programa realiza operaciones de lectura/escritura o no. Si lee (tanto de la entrada estándar) o realiza operaciones con ficheros, debes importar la funcionalidad necesaria de las librerías correspondientes (`java.io`).

El preámbulo para el algoritmo que estamos implementando sería:

```
package packFactorial;  
import java.io.*;  
public class MostrarFactorial {  
    public static void main(String args[]) {  
  
    }  
}
```

2.2. Implementación de las operaciones de E/S

La librería JTF proporciona funcionalidad que simplifica las operaciones de entrada/salida encapsulando el tratamiento que describiremos en esta sección. Dicha funcionalidad se limita a obtener datos de la entrada estándar e imprimir mensajes en la salida estándar.

2.2.1. Implementación de las operaciones de salida

Java proporciona dos instrucciones distintas para escribir mensajes (`print` y `println`). La diferencia entre ambas es que la segunda añade un *salto de*

⁵ En el tema 4 y en la asignatura PMyOO se darán más detalles sobre el significado de las palabras reservadas que aparecen (`static`, etc.). De momento, quédate con que hay que insertar este bloque en el preámbulo del programa



línea al final del mensaje. A diferencia de cuando usamos la librería JTF, debemos indicar en qué dispositivo queremos mostrar el mensaje⁶

Para escribir en la salida estándar utilizaremos la instrucción

```
System.out.println(mensaje);
```

mientras que para escribir los mensajes en la salida estándar **para los errores** utilizaremos la instrucción:

```
System.err.println(mensaje);
```

2.2.2. Implementación de las operaciones de entrada

Java proporciona una instrucción básica para la lectura de datos (**read**). Esta operación sólo permite leer un carácter, por lo que no es una operación muy útil. Sin embargo, en la librería `java.io` se definen utilidades para leer cadenas de caracteres. Para ello, además de importar la librería en el preámbulo, debemos añadir la instrucción

```
BufferedReader nombreGestorEntrada = new  
    BufferedReader(new InputStreamReader(System.in));
```

donde *nombreGestorEntrada* será el nombre que le demos al *gestor de lectura* que vamos a utilizar en el programa (p.e., *entrada*). Con esta instrucción, estamos definiendo un *gestor* para la lectura de datos de la entrada estándar⁷. Para leer una cadena de caracteres, utilizaremos la instrucción:

```
nombreGestorLectura.readLine();
```

Con esta operación, siempre se obtiene una cadena de caracteres. Para obtener información de otro tipo, se debe convertir la cadena de caracteres al tipo correspondiente. La Tabla 6 resume cómo se implementan las distintas operaciones de lectura⁸.

⁶ Para ver cómo escribir en ficheros, consulta los apuntes de la asignatura

⁷ Se puede definir de manera similar un gestor de lectura para ficheros. Para más información, consulta los apuntes de la asignatura

⁸ Las operaciones realizan tanto la lectura de datos como la conversión al tipo correspondiente



Hay que tener en cuenta que estas operaciones sólo realizan la lectura de datos. Si el dato lo debe aportar el usuario, se debe añadir operaciones que informen al usuario de que debe introducir los datos necesarios.

Operaciones de lectura	Operaciones de lectura en Java
leeEntero	<code>Integer.parseInt(<i>nombreGestorEntrada</i>.readLine())</code>
leeReal	<code>Float.parseFloat(<i>nombreGestorEntrada</i>.readLine())</code>
leeBooelano	<code>Boolean.parseBoolean(<i>nombreGestorEntrada</i>.readLine())</code>

Tabla 6: Tipos de datos

2.2.3. Tratamiento de errores

Al realizar operaciones de lectura, se pueden producir errores (bien por problemas de comunicación con el dispositivo de entrada o bien porque los datos no son adecuados). Para hacer frente a estas situaciones, insertaremos las acciones que hemos identificado en nuestro algoritmo dentro de un bloque `try ...catch` como el que se muestra a continuación.

```
try {  
  
} catch ( IOException e)  
  
} catch ( NumberFormatException e)  
  
}
```

Las acciones identificadas se incluyen en el bloque definido en el par de llaves tras la palabra reservada `try`. El bloque `catch (IOException e)` sirve para tratar los errores que se pudieran producir al realizar las operaciones de entrada/salida, en este caso la lectura. En nuestro caso, nos limitaremos a notificar que se ha producido un error. El segundo bloque, `catch`



(`NumberFormatException e`), permite tratar los errores que se pudieran producir al realizar las operaciones de la conversión de la cadena de caracteres a un valor numérico (p.e., si queremos transformar “3hxf” en un entero). De nuevo, nos limitaremos a notificar que se ha producido un error.

Si sólo queremos leer cadenas de caracteres, omitiremos el segundo bloque `catch`.

2.3. Implementación completa

Uniendo todo lo que hemos visto en los apartados anteriores, obtendríamos la siguiente implementación para nuestro algoritmo.

```
package packFactorial;
import java.io.*;

public class MostrarFactorial {
    public static void main(String args[]) {
        final int VAL_MIN = 0;
        final int VAL_MAX = 15;
        final String MENS_DATOS = "Introduzca un número
            entero [" + VAL_MIN + "," + VAL_MAX + "]:";
        final String MENS_ERROR = "Número incorrecto";
        final String MENS_ERROR_LECTURA = "Error de
            lectura";
        int numUsuario, factorial;

        // Para realizar las operaciones de lectura
        BufferedReader entrada = new BufferedReader(new
            InputStreamReader(System.in));

        /* Como se pueden producir errores, incluimos las
            instrucciones en un bloque try */
        try {
            /* Obtener el valor del que se quiere
                calcular el factorial */
            System.out.println(MENS_DATOS);
            numUsuario=Integer.parseInt(entrada.
                readLine());
```



```
while (numUsuario<VAL_MIN|| numUsuario>
    VAL_MAX) {
    System.err.println(MENS_ERROR);
    System.out.println(MENS_DATOS);
    numUsuario=Integer.parseInt(entrada.
        readLine());
}
// Calcular Factorial
factorial=1;
for (int i=1;i<=numUsuario;i++){
    factorial*=i;
}
System.out.println("El factorial de " +
    numUsuario " es " + factorial);
}
catch (IOException ex) {
    System.err.println(MENS_ERROR_LECTURA);
}
catch (NumberFormatException ex) {
    System.err.println(MENS_ERROR_LECTURA);
}
}
}
```