lo que significa que cualquier texto codificado en ASCII se representa exactamente igual en UTF-8. Este ha sido un motivo fundamental para la adopción generalizada de UTF-8.

Para las nuevas aplicaciones, siempre hay que utilizar UTF-8 para almacenar texto, a no ser que haya alguna razón de peso para utilizar otra, que normalmente no la hay. A veces hay que importar u obtener textos de otras fuentes, desde donde podrían venir con otras codificaciones. Hay que identificar estas situaciones y hacer la conversión necesaria para recodificar los textos.

Recurso digital

En el anexo web 2.1, disponible en www.sintesis.com y accesible con el código indicado en la primera página del libro, encontrarás información sobre la codificación del contenido de ficheros.

RECUERDA

Cualquier editor de texto debería permitir visualizar correctamente un fichero de texto independientemente de su codificación. Por ejemplo, Notepad en Windows y gedit en Linux. Normalmente, con la opción "Guardar como…" se puede seleccionar la codificación que se va a utilizar, y UTF-8 suele aparecer como una de las opciones.

La manera más fiable y segura de confirmar el tipo de un fichero en Linux es con el comando `file`. Este analiza los contenidos del fichero e indica su tipo. Si es de texto, indica la codificación utilizada para el texto, típicamente: ASCII, UTF-8 o iso-8859-1.

El comando `iconv` de Linux permite recodificar ficheros de texto. El siguiente comando, por ejemplo, se podría utilizar para recodificar a UTF-8 un fichero codificado en ISO 8859-1.

```
iconv -f iso8859-1 -t utf-8 fichero_8859-1.txt > fichero_utf8.txt
```

2.4. La clase File de Java

La versión de Java utilizada para este libro es Java SE 8, una versión LTS (*long term support*, es decir, con soporte a largo plazo). En los servidores web de Oracle se puede consultar la documentación de la biblioteca estándar de clases de Java SE 8.

Las clases que permiten trabajar con ficheros están en el paquete `java.io`.

Recurso web

Se recomienda consultar en http://docs.oracle.com/javase/8/docs/api la documentación de Java SE8 para más información acerca de cualquier clase utilizada en este capítulo. Arriba, a la izquierda, aparece la lista de paquetes por orden alfabético. En ella se puede localizar el paquete `java.io`. Si se pulsa sobre él, aparece debajo la lista de interfaces y clases que contiene el paquete.

La clase `File` permite obtener información relativa a directorios y ficheros dentro de un sistema de ficheros y realizar diversas operaciones con ellos tales como borrar, renombrar, etc. En el siguiente cuadro se proporciona un resumen de la funcionalidad de esta clase, mostrando solo los principales métodos agrupados por categorías.

CUADRO 2.1
Métodos de la clase `File`

Categoría	Modificador/tipo	Método(s)	Funcionalidad
Constructor		`File(String ruta)`	Crea objeto `File` para la ruta indicada, que puede corresponder a un directorio o a un fichero.

Consulta de propiedades	`boolean`	`canRead()`
		`canWrite()`
		`canExecute()`
			Comprueban si el programa tiene diversos tipos de permisos sobre el fichero o directorio, tales como de lectura, escritura y ejecución (si se trata de un fichero). Para un directorio, `canExecute()` significa que se puede establecer como directorio actual.

	`boolean`	`exists()`	Comprueba si el fichero o directorio existe.
	`boolean`	`isDirectory()` `isFile()`	Comprueban si se trata de un directorio o de un fichero.
	`long`	`length()`	Devuelve longitud del fichero.
	`File`	`getParent()` `getParentFile()`	Devuelven el directorio padre.
	`String`	`getName()`	Devuelve nombre del fichero.
Enumeración	`String[]`	`list()`	Devuelve un *array* con los nombres de los directorios y ficheros dentro del directorio.
	`File[]`	`listFiles()`	Devuelve un *array* con los directorios y ficheros dentro del directorio.
Creación, borrado y renombrado	`boolean`	`createNewFile()`	Crea nuevo fichero.
	`static File`	`createTempFile()`	Crea nuevo fichero temporal y devuelve objeto de tipo `File` asociado, para poder trabajar con él.
	`boolean`	`delete()`	Borra fichero o directorio.
	`boolean`	`renameTo()`	Renombra fichero o directorio.
	`boolean`	`mkdir()`	Crea un directorio.
Otras	`java.nio.file.Path`	`toPath()`	Devuelve un objeto que permite acceder a información y funcionalidad adicional proporcionada por el paquete `java.nio`.

El siguiente programa de ejemplo muestra por defecto un listado de los ficheros y directorios que contiene el directorio desde el que se ejecuta el programa. Pero si se le pasa la ruta de un directorio o fichero, muestra información acerca de él y, si se trata de un directorio, muestra los ficheros y directorios que contiene.

```
// Uso de la clase File para mostrar información de ficheros y directorios
package listadodirectorio;
import java.io.File;
public class ListadoDirectorio {
    public static void main(String[] args) {
        String ruta=".";
        if(args.length>=1) ruta=args[0];
        File fich=new File(ruta);
        if(!fich.exists()) {
            System.out.println("No existe el fichero o directorio ("+ruta+").");
        }
        else {
            if(fich.isFile()) {
                System.out.println(ruta+" es un fichero.");
            }
            else {
                System.out.println(ruta+" es un directorio. Contenidos: ");
                File[] ficheros=fich.listFiles(); // Ojo, ficheros o directorios
                for(File f: ficheros) {
                    String textoDescr=f.isDirectory() ? "/":
                    f.isFile() ? "_": "?";
                    System.out.println("("+textoDescr+") "+f.getName());
                }
            }
        }
    }
}
```

PARA SABER MÁS

Se pueden pasar argumentos desde la línea de comandos a cualquier programa. Cualquier programa Java debe tener un método `main(String args [])` que se invoque en el momento de ejecutar el programa. El parámetro `String args[]` proporciona los argumentos de línea de comandos. Para pasar argumentos de línea de comando a un programa que se está desarrollando utilizando un IDE (entorno integrado de desarrollo), lo más cómodo suele ser utilizar las opciones que este IDE proporcione para ello dentro de su interfaz de usuario.

Actividad propuesta 2.1

Modifica el programa anterior para que muestre más información acerca de cada fichero y directorio, al menos el tamaño (si es un fichero), los permisos de los que se dispone sobre el fichero o

directorio, y la fecha de última modificación (en cualquier formato). Los permisos hay que mostrarlos en el formato habitual de Linux, a saber, con tres caracteres seguidos: una *r* si hay permiso para lectura o un guion si no lo hay; una *w* si hay permiso para escritura o un guion si no lo hay; y una *x* si hay permiso para ejecución, en el caso de que se trate de un fichero, o para entrar en el directorio, en el caso de que se trate de un directorio, o un guion si no lo hay.

2.5. Gestión de excepciones en Java

Antes de seguir avanzando con el contenido del capítulo, se incluye un breve repaso a la gestión de excepciones en el lenguaje Java. Cualquier programa escrito en Java debe realizar una adecuada gestión de excepciones. En este apartado se explicará lo más importante de la gestión de excepciones en Java, o al menos todo lo que se vaya a necesitar para este libro.

Una excepción es un evento que ocurre durante la ejecución de un programa y que interrumpe su curso normal de ejecución. Una excepción podría producirse, por ejemplo, al dividir por cero, como se muestra en el siguiente ejemplo.

```
// Excepción no gestionada durante la ejecución de un programa.
package excepcionesdivporcero;
public class ExcepcionesDivPorCero {
    public int divide(int a, int b) {
        return a/b;
    }
    public static void main(String[] args) {
        int a,b;
        a=5; b=2; System.out.println(a+"/"+b+"="+a/b);
        b=0; System.out.println(a+"/"+b+"="+a/b);
        b=3; System.out.println(a+"/"+b+"="+a/b);
    }
}
```

Al ejecutar este programa se obtendrá algo similar a lo siguiente:

```
5/2=2
Exception in thread "main" java.lang.ArithmeticException: / by zero at excepcionesdivporcero.
ExcepcionesDivPorCero.main(ExcepcionesDivPorCero.java:25)
```

2.5.1. Captura y gestión de excepciones

Cuando tiene lugar una excepción no gestionada, como con el programa anterior, aparte de mostrarse un mensaje de error, se aborta la ejecución del programa. Por ese motivo, el programa anterior no muestra el resultado de la última división entera. Cualquier fragmento de programa que pueda generar una excepción debería capturarlas y gestionarlas.

La salida del programa anterior indica el tipo de excepción que se ha producido, a saber, `ArithmeticException`. Se puede capturar esta excepción con un sencillo bloque `try {} catch {}` y gestionarla, con lo que el programa anterior podría quedar así: