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:

```
// Excepción gestionada durante la ejecución de un programa.
package excepcionesdivporcerogest;
public class ExcepcionesDivPorCeroGest {
    public int divide(int a, int b) {
        return a / b;
    }
    public static void main(String[] args) {
        int a, b;
        a = 5; b = 2;
        try {
            System.out.println(a + "/" + b + "=" + a / b);
        } catch (ArithmeticException e) {
            System.err.println("Error al dividir: " + a + "/" + b);
        }
        try {
            b = 0; System.out.println(a + "/" + b + "=" + a / b);
        } catch (ArithmeticException e) {
            System.err.println("Error al dividir: " + a + "/" + b);
        }
        try {
            b = 3; System.out.println(a + "/" + b + "=" + a / b);
        } catch (ArithmeticException e) {
            System.err.println("Error al dividir: " + a + "/" + b);
        }
    }
}
```

Al ejecutar este programa, se captura y gestiona cualquier excepción que se pueda producir, de manera que se muestra un mensaje apropiado y se continúa con la ejecución. La salida del anterior programa sería la siguiente:

```
5/2=2
Error al dividir: 5/0
5/3=1
```

RECUERDA

✓ Es recomendable escribir los mensajes de error en la salida de error `System.err` en lugar de en la salida estándar `System.out`.

Actividad propuesta 2.2

¿Cómo cambiaría la funcionalidad del programa anterior si, en lugar de haber un bloque `try` … `catch` … para cada división, hubiera uno solo para las tres divisiones?

Las excepciones son objetos de Java, pertenecientes a la clase `Exception` o a una subclase de ella. Esta clase tiene varios métodos interesantes para obtener información acerca de la excepción que se ha producido.

CUADRO 2.2
Métodos de la clase `Exception`

Modificador/tipo	Método(s)	Funcionalidad
`void`	`printStackTrace()`	Muestra información técnica muy detallada acerca de la excepción y el contexto en que se produjo. Lo hace en la salida de error, `System.err`. Al principio del desarrollo de un programa, y para programas de prueba, puede ser una buena opción utilizar esta función para mostrar información de todas las excepciones, y perfilar más adelante cómo se gestionan excepciones de tipos particulares.
`String`	`getMessage()`	Proporciona un mensaje detallado acerca de la excepción.
`String`	`getLocalizedMessage()`	Proporciona una descripción localizada (es decir, traducida a la lengua local) de la excepción.

2.5.2. Gestión diferenciada de distintos tipos de excepciones

En un bloque `try {} catch {}` se pueden gestionar por separado distintos tipos de excepciones. Es conveniente incluir un manejador para `Exception` al final para que ninguna excepción se quede sin gestionar.

PARA SABER MÁS

En aplicaciones profesionales, la práctica habitual es utilizar herramientas de *logging* (de registro), y no solo para mensajes de error. Los mensajes se suelen registrar en ficheros. Estas herramientas permiten generar de forma diferenciada distintos tipos de mensaje, típicamente, y por orden de importancia, de mayor a menor: *error, warning* (de aviso), *info* (informativo) y *debug* (para depuración). Permiten configurar el nivel de *logging,* de manera que solo se registren los mensajes a partir del nivel de importancia establecido. Si este se establece como *warning,* se mostrarían los de tipo *warning* y *error*. Entre las herramientas más ampliamente utilizadas para Java están:

- Java Logging API (http://docs.oracle.com/javase/8/docs/technotes/guides/logging)
- Log4j (http://logging.apache.org/log4j)

El siguiente programa rellena un *array* con números y después realiza un cálculo aritmético para cada uno con ellos y muestra la segunda cifra del resultado. Esto no es realmente muy útil, como no sea para mostrar cómo se puede hacer saltar excepciones de diversos tipos, capturarlas y gestionarlas por separado. Solo para un elemento del *array* se realiza el cálculo sin que salte ninguna excepción.

```
// Gestión diferenciada de distintos tipos de excepciones

package excepcionesdiversas;

public class ExcepcionesDiversas {

    public static void main(String[] args) {
    // Rellenar array con números variados
    int nums[][] = new int[2][3];
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 3; j++) {
            nums[i][j] = i + j;
            }
        }
        // Realizar cálculo para cada posición del array.
        // Se producen excepciones de diversos tipos.
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                try {
                    System.out.print("Segunda cifra de 5*nums["+i+"]
                        ["+j+"]/"+j+": ");
                    System.out.println(String.valueOf(5 * nums[i][j] /
                        j).charAt(1));
                } catch(ArithmeticException e) {
                    System.out.println("ERROR: aritmético 5*"+nums[i][j]+"/"+j);
                } catch (ArrayIndexOutOfBoundsException e) {
                    System.out.println("ERROR: No existe nums["+i+"]["+j+"]");
                } catch (Exception e) {
                    System.out.println("ERROR: de otro tipo al calcular segunda
                        cifra de: 5*"+nums[i][j]+"/"+j);
                    System.out.println();
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Para poder interpretar más fácilmente la salida de este programa, todos los mensajes de error se han dirigido hacia `System.out` y no `System.err`. Su salida sería:

```
Segunda cifra de 5*nums[0][0]/0: ERROR: aritmético 5*0/0
Segunda cifra de 5*nums[0][1]/1: ERROR: de otro tipo al calcular segunda cifra de: 5*1/1

java.lang.StringIndexOutOfBoundsException: String index out of range: 1
at java.lang.String.charAt(String.java:658)
at excepcionesdiversas.ExcepcionesDiversas.main(ExcepcionesDiversas.java:31)
Segunda cifra de 5*nums[0][2]/2: ERROR: de otro tipo al calcular segunda cifra de: 5*2/2

java.lang.StringIndexOutOfBoundsException: String index out of range: 1
at java.lang.String.charAt(String.java:658)
at excepcionesdiversas.ExcepcionesDiversas.main(ExcepcionesDiversas.java:31)
Segunda cifra de 5*nums[1][0]/0: ERROR: aritmético 5*1/0
Segunda cifra de 5*nums[1][1]/1: 0
Segunda cifra de 5*nums[1][2]/2: ERROR: de otro tipo al calcular segunda cifra de: 5*3/2
```

```
java.lang.StringIndexOutOfBoundsException: String index out of range: 1
at java.lang.String.charAt(String.java:658)
at excepcionesdiversas.ExcepcionesDiversas.main(ExcepcionesDiversas.java:31)
Segunda cifra de 5*nums[2][0]/0: ERROR: No existe nums[2][0]
Segunda cifra de 5*nums[2][1]/1: ERROR: No existe nums[2][1]
Segunda cifra de 5*nums[2][2]/2: ERROR: No existe nums[2][2]
```

2.5.3. Declaración de excepciones lanzadas por un método de clase

Si el compilador es capaz de determinar que un método de una clase puede originar un tipo de excepción, pero no lo gestiona mediante un bloque `catch(){}`, la compilación terminará con un error. Una posibilidad entonces es gestionar la excepción en el método mediante un bloque `catch(){}`. La otra es añadir el modificador `throws` seguido de la clase de excepción.

La siguiente clase tiene un método que crea un fichero temporal con un nombre que empieza por un prefijo dado, y escribe en él un carácter dado, un número dado de veces. Durante este proceso puede saltar la excepción `IOException` en varias ocasiones, pero no se quiere gestionarla en el método. Por ello se incluye `throws IOException` en su declaración. Por lo demás, el método es sencillo y, en cualquier caso, lo importante es comprender la gestión de excepciones. Las clases y procedimientos utilizados se verán más adelante en este capítulo.

```
// Declaración de excepciones lanzadas por método de clase con throws
package excepcionesconthrows;

import java.io.File;
import java.io.IOException;
import java.io.FileWriter;

public class ExcepcionesConThrows {

    public File creaFicheroTempConCar(String prefNomFich, char car, int
        numRep) throws IOException {
        File f = File.createTempFile(prefNomFich, "");
        FileWriter fw = new FileWriter(f);
        for (int i = 0; i < numRep; i++) fw.write(car);
        fw.close();
        return f;
    }

    public static void main(String[] args) {
        try {
            File ft = new ExcepcionesConThrows().
                creaFicheroTempConCar("AAAA_", 'A', 20);
            System.out.println("Creado fichero: " + ft.getAbsolutePath());
            ft.delete();
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }
}
```