Control de excepciones

Errores en tiempo de compilación y en tiempo de ejecución

Programando en Java se pueden producir diferentes tipos de error. El más común es el que se produce en tiempo de compilación y generalmente ocurre cuando hay alguna sentencia que no está bien escrita, por ejemplo si falta un punto y coma al final de una línea, cuando hay comillas o paréntesis que se abren pero no se cierran, al intentar asignar una cadena de caracteres a una variable que es de tipo entero, etc.

Si un programa da errores en tiempo de compilación, no se crea el archivo de *bytecode* con la extensión `.class` y, por tanto, hasta que no se arreglen los fallos, no compilará y, en consecuencia, tampoco se podrá ejecutar.

La siguiente línea de código provocará un error en tiempo de compilación ¿sabrías decir por qué?

```
System.out.println("¡Hola mundo!);
```

Si ya llevas algún tiempo programando en Java, te habrás topado con cientos de errores como éste. Suelen ser los más fáciles de detectar y corregir.

Errores en tiempo de compilación

- Se suelen producir cuando el código no está bien escrito, por ej. cuando falta el punto y coma al final de la sentencia.
- Hasta que no se arreglen, no se genera el archivo con la extensión `.class`

Existen otros errores que se producen en tiempo de ejecución a los que llamaremos **excepciones**. El programa compila y se puede ejecutar pero, por algún motivo, se produce un fallo y el programa se "rompe". Un buen programador debe preveer esta situación y debe saber encauzar el programa para que quede todo bajo control. En este capítulo estudiaremos estos errores y veremos cómo se pueden mantener a raya.

A continuación se muestra un programa que calcula la media de dos números.

```java
import java.util.Scanner;

class EjemploExcepciones01 {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.println("Este programa calcula la media de dos números");

    System.out.print("Introduzca el primer número: ");
    double numero1 = Double.parseDouble(s.nextLine());

    System.out.print("Introduzca el segundo número: ");
    double numero2 = Double.parseDouble(s.nextLine());

    System.out.println("La media es " + (numero1 + numero2) / 2);
  }
}
```

El programa compila y se ejecuta correctamente. Entonces ¿dónde está el problema? Prueba a introducir la palabra `hola` en lugar de un número.

```
Este programa calcula la media de dos números
Introduzca el primer número: hola
Exception in thread "main" java.lang.NumberFormatException: For input string: "hola"
        at   sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)
        at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
        at  java.lang.Double.parseDouble(Double.java:538)
        at   EjemploExcepciones01.main(EjemploExcepciones01.java:11)
```

El programa ha terminado de forma repentina. ¿Y si, en lugar de ser un simple ejemplo que calcula una media, se tratase de la aplicación que controla un cajero automático o, peor aún, el programa de pilotaje de un avión? Vamos a ver cómo solucionar este problema en los próximos apartados.

Errores en tiempo ejecución

- Se llaman también **excepciones**.
- El programa compila y, en principio, se ejecuta bien... hasta que se dan determinadas circunstancias que provocan un fallo.
- Es necesario preveerlos para que el programa no termine de forma inesperada.

El bloque `try - catch - finally`

El bloque `try - catch - finally` sirve para encauzar el flujo del programa de tal forma que, si se produce una excepción, no se termine de forma drástica y se pueda reconducir el ejecución de una manera controlada.

El formato de este bloque es el siguiente:

```
try {

  Instrucciones que se pretenden ejecutar
  (si se produce una excepción puede que
  no se ejecuten todas ellas).

} catch {

  Instrucciones que se van a ejecutar
  cuando se produce una excepción.

} finally {

  Instrucciones que se van ejecutar tanto
  si se producen excepciones como si no
  se producen.

}
```

Se pueden especificar varios `catch` para controlar diferentes excepciones como veremos más adelante. La parte `finally` es opcional.

Siguiendo con el programa que calcula la media de dos números, vamos a introducir un bloque `try - catch - finally` para que el programa termine de forma controlada si se produce una excepción.

```
public class EjemploExcepciones02 {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.println("Este programa calcula la media de dos números");


    try {

      System.out.print("Introduzca el primer número: ");
      double numero1 = Double.parseDouble(s.nextLine());

      System.out.print("Introduzca el segundo número: ");
      double numero2 = Double.parseDouble(s.nextLine());

      System.out.println("La media es " + (numero1 + numero2) / 2);

    } catch (Exception e) {

      System.out.print("No se puede calcular la media. ");
      System.out.println("Los datos introducidos no son correctos.");

    } finally {

      System.out.println("Gracias por utilizar este programa ¡hasta la próxima!");

    }

  }
}
```

Ahora, si el usuario introduce un dato incorrecto, el programa termina de forma ordenada.

```
Este programa calcula la media de dos números
Introduzca el primer número: hola
No se puede calcular la media. Los datos introducidos no son correctos.
Gracias por utilizar este programa ¡hasta la próxima!
```

Se puede mostrar tanto el tipo de excepción como el error exacto que se produce. Para ello, se aplican los métodos `getClass()` y `getMessage()` respectivamente al objeto `e`. El tipo de excepción viene dado por el nombre de una clase que es subclase de `Exception`. Bastará con añadir las siguientes líneas al ejemplo anterior.

```
System.out.println("Excepción: " + e.getClass());
System.out.println("Error: " + e.getMessage());
```

El ejemplo completo quedaría como se indica a continuación.

```
import java.util.Scanner;

public class EjemploExcepciones02v2 {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.println("Este programa calcula la media de dos números");

    try {

      System.out.print("Introduzca el primer número: ");
      double numero1 = Double.parseDouble(s.nextLine());

      System.out.print("Introduzca el segundo número: ");
      double numero2 = Double.parseDouble(s.nextLine());

      System.out.println("La media es " + (numero1 + numero2) / 2);

    } catch (Exception e) {

      System.out.println("Excepción: " + e.getClass());
      System.out.println("Error: " + e.getMessage());
      System.out.println("No se puede calcular la media. ");
      System.out.println("Los datos introducidos no son correctos.");

    }

  }
}
```

Si el usuario introduce un dato incorrecto, ahora el programa muestra la información adicional sobre la excepción que se produce.

```
Este programa calcula la media de dos números
Introduzca el primer número: hola
Excepción: class java.lang.NumberFormatException
Error: For input string: "hola"
No se puede calcular la media.
Los datos introducidos no son correctos.
```

Control de varios tipos de excepciones

La clase `Exception` hace referencia a una excepción genérica. Existen muchas subclases de ella como `DataFormatException`, `IOException`, `IndexOutOfBoundsException`, etc.

Mediante la utilización de varios `catch` con diferentes subclases de `Exception` se pueden discriminar distintas excepciones.

Utilizaremos como ejemplo un programa que pide el número total de asteriscos y el número de líneas que se quieren pintar. Por ejemplo, si el usuario dice que quiere pintar 10 asteriscos y 3 líneas, el programa pintará dos líneas de 4 asteriscos más una línea de 2 asteriscos.

```
import java.util.Scanner;

public class EjemploExcepciones04 {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.println("Este programa pinta varias líneas de asteriscos");

    System.out.print("Introduzca el número total de asteriscos: ");
    int asteriscos = Integer.parseInt(s.nextLine());

    System.out.print("Introduzca el número de líneas que quiere pintar: ");
    int lineas = Integer.parseInt(s.nextLine());

    int longitud = (asteriscos % lineas) == 0 ? asteriscos / lineas : (int)Math.ceil((double)a\
steriscos / lineas);

    int cuentaAsteriscos = 0;

    for (int i = 1; i <= lineas; i++) {
      System.out.print("Línea " + i + ": ");
      for (int j = 0; (j < longitud) && (cuentaAsteriscos++ < asteriscos); j++) {
        System.out.print("*");
      }
      System.out.println();
    }

  }
}
```

Lo interesante de este ejemplo está en el hecho de que se pueden producir diferentes errores en tiempo de ejecución según los datos que introduzca el usuario.

Cuando el usuario introduce los datos correctamente, es decir, dos números enteros, no hay ningún problema.

```
Este programa pinta varias líneas de asteriscos
Introduzca el número total de asteriscos: 10
Introduzca el número de líneas que quiere pintar: 3
Línea 1: ****
Línea 2: ****
Línea 3: **
```

Ahora bien, si el usuario introduce un número con decimales en lugar de un número entero, se produce la excepción `NumberFormatException` como se muestra a continuación.

```
Este programa pinta varias líneas de asteriscos
Introduzca el número total de asteriscos: 20.75
Exception in thread "main" java.lang.NumberFormatException: For input string: "20.75"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at   java.lang.Integer.parseInt(Integer.java:615)
        at   EjemploExcepciones04.main(EjemploExcepciones04.java:11)
```

La línea que ha producido el fallo se muestra a continuación.

```
int asteriscos = Integer.parseInt(s.nextLine());
```

Lo que ha sucedido es que el método `parseInt()` ha intentado convertir la cadena `"20.75"` en un número entero. Lógicamente, esto no se puede hacer porque un número entero no contiene el punto decimal. Como consecuencia se ha producido la excepción `NumberFormatException`.

Veamos qué otro error se puede producir. Si lo que hace el usuario es introducir el 0 cuando el programa le pregunta cuántas líneas quiere pintar, salta la excepción `ArithmeticException` ya que el programa intenta realizar una división con el divisor igual a 0.

```
Este programa pinta varias líneas de asteriscos
Introduzca el número total de asteriscos: 20
Introduzca el número de líneas que quiere pintar: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at EjemploExcepciones04.main(EjemploExcepciones04.java:16)
```

Ahora, la línea que ha provocado el error es otra distinta.

```
int longitud = (asteriscos % lineas) == 0 ? asteriscos / lineas : (int)Math.ceil((double)aster\
iscos / lineas);
```

Al intentar realizar la división de cualquier número entre la variable `lineas` se produce un error ya que dicha variable vale 0.

Se puede utilizar la estructura `try - catch` con dos excepciones diferentes o más. El siguiente ejemplo es una mejora del anterior en el que se incluye el control de las excepciones `NumberFormatException` y `ArithmeticException`.

```
import java.util.Scanner;

public class EjemploExcepciones05 {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);

    System.out.println("Este programa pinta varias líneas de asteriscos");

    try {

      System.out.print("Introduzca el número total de asteriscos: ");
      int asteriscos = Integer.parseInt(s.nextLine());

      System.out.print("Introduzca el número de líneas que quiere pintar: ");
      int lineas = Integer.parseInt(s.nextLine());

int longitud = (asteriscos % lineas) == 0 ? asteriscos / lineas : (int) Math.ceil((double)\asteriscos / lineas);


      int cuentaAsteriscos = 0;

      for (int i = 1; i <= lineas; i++) {
        System.out.print("Línea " + i + ": ");
        for (int j = 0; (j < longitud) && (cuentaAsteriscos++ < asteriscos); j++) {
          System.out.print("*");
        }
        System.out.println();
      }

    } catch (NumberFormatException nfe) {

      System.out.println("Los datos introducidos no son correctos.");
      System.out.println("Se deben introducir números enteros.");

    } catch (ArithmeticException ae) {
      System.out.println("El número de líneas no puede ser 0.");
    }
  }


    }

  }
}
```

Ahora, si el usuario introduce un número con decimales o una palabra en lugar de un número entero, no hay ningún problema, el programa no se rompe y termina de forma ordenada.

```
Este programa pinta varias líneas de asteriscos
Introduzca el número total de asteriscos: 10
Introduzca el número de líneas que quiere pintar: 4.6
Los datos introducidos no son correctos.
Se deben introducir números enteros.
```

De igual modo, el programa también controla la excepción que se produce si el usuario introduce un 0 para indicar el número de líneas que quiere pintar.

```
Este programa pinta varias líneas de asteriscos
Introduzca el número total de asteriscos: 10
Introduzca el número de líneas que quiere pintar: 0
El número de líneas no puede ser 0.
```

Lanzamiento de excepciones con `throw`

La orden `throw` permite lanzar de forma explícita una excepción. Por ejemplo, la sentencia `throw new ArithmeticException()` crea de forma artificial una excepción igual que si existiera una línea como `System.out.println(1 / 0);`.

```
public class EjemploExcepciones06throw {
  public static void main(String[] args) {
    System.out.println("Inicio");
    throw new ArithmeticException();
  }
}
```

Este programa provoca la siguiente salida.

```
Inicio
Exception in thread "main" java.lang.ArithmeticException
        at   EjemploExcepciones06throw.main(EjemploExcepciones06throw.java:5)
```

De la misma forma, el programa que se muestra a continuación genera también una excepción del tipo `ArithmeticException`.

```
public class EjemploExcepciones07sinthrow {
  public static void main(String[] args) {
    System.out.println("Inicio");
    System.out.println(1 / 0);
  }
}
```

```
Inicio
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at    EjemploExcepciones07sinthrow.main(EjemploExcepciones07sinthrow.java:5)
```

Hay que tener en cuenta que `throw` solo puede lanzar excepciones que pertenezcan a la clase `Throwable`.

Como `throw` permite lanzar de forma explícita una excepción, nos servirá para lanzar excepciones propias como veremos más adelante. También es útil cuando se recoge la excepción en un método y luego, esa misma excepción se vuelve a lanzar para que la recoja, a su vez, otro método y luego otro y así sucesivamente hasta llegar al `main`.

Vamos a ver cómo se recoge y se trata una excepción dentro de una función y, además es la propia función la que *"pasa la bola"* al main. Partimos de un ejemplo sin control de excepciones.

```
import java.util.Scanner;

public class EjemploExcepciones08manzanas {
 public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print("Número de manzanas: ");
    int m = Integer.parseInt(s.nextLine());
    System.out.print("Número de personas: ");
    int p = Integer.parseInt(s.nextLine());
    System.out.print("A cada persona le corresponden " + reparteManzanas(m, p) + " manzanas.");
  }

  public static int reparteManzanas(int manzanas, int personas) {
    return manzanas / personas;
  }
}
```

Cuando el usuario introduce los datos correctamente, el programa realiza su cometido.

```
Número de manzanas: 10
Número de personas: 5
A cada persona le corresponden 2 manzanas.
```

Pero cuando el usuario introduce un 0 como número de personas, el programa provoca una excepción.

```
Número de manzanas: 10
Número de personas: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at EjemploExcepciones08manzanas.reparteManzanas(EjemploExcepciones08manzanas.java:15)
        at EjemploExcepciones08manzanas.main(EjemploExcepciones08manzanas.java:11)
```

Vamos a mejorar el ejemplo para llevar un control de las excepciones tanto en la función como en el `main`.

```
import java.util.Scanner;

public class EjemploExcepciones09ManzanasConThrow {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print("Número de manzanas: ");
    int m = Integer.parseInt(s.nextLine());
    System.out.print("Número de personas: ");
    int p = Integer.parseInt(s.nextLine());
    try {
      System.out.print("A cada persona le corresponden " + reparteManzanas(m, p) + " manzanas.\
");
    } catch (ArithmeticException ae) {
      System.out.println("Los datos introducidos no son correctos.");
    }
  }

  public static int reparteManzanas(int manzanas, int personas) {
    try {
      return manzanas / personas;
    } catch (ArithmeticException ae) {
      System.out.println("El número de personas vale 0.");
      throw ae;
    }
  }
}
```

```
Número de manzanas: 10
Número de personas: 0
El número de personas vale 0.
Los datos introducidos no son correctos.
```

Declaración de un método con `throws`

Hemos visto en el apartado anterior cómo lanzar una excepción desde una función/método con `throw`. Es muy recomendable indicar de forma explícita en la cabecera que existe esa posibilidad, es decir, que el método en cuestión puede provocar una excepción. Se trata de una declaración de intenciones, algo parecido al `@Override`. Si el IDE que estemos utilizando detecta que un método tiene `throws` en su cabecera y, sin embargo, no hemos gestionado bien la posibilidad de provocar una excepción, nos avisará con el correspondiente mensaje de error.

El código completo quedaría como sigue.

```
import java.util.Scanner;

public class EjemploExcepciones09ManzanasConThrowYThrows {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print("Número de manzanas: ");
    int m = Integer.parseInt(s.nextLine());
    System.out.print("Número de personas: ");
    int p = Integer.parseInt(s.nextLine());
    try {
      System.out.print("A cada persona le corresponden " + reparteManzanas(m, p) + " manzanas.\
");
    } catch (ArithmeticException ae) {
      System.out.println("Los datos introducidos no son correctos.");
    }
  }

  public static int reparteManzanas(int manzanas, int personas) throws ArithmeticException {
    try {
      return manzanas / personas;
    } catch (ArithmeticException ae) {

       System.out.println("El número de personas vale 0.");
      throw ae;
     }
   }
}
```

Recomendaciones sobre el tratamiento de excepciones

En este capítulo hemos visto cómo tratar diferentes tipos de excepciones para prevenir que un programa se rompa y termine de forma abrupta. No obstante, en términos generales, no se debe dejar en manos del control de excepciones lo que se puede hacer mediante sencillas comprobaciones previas.

Por ejemplo, antes de realizar una división, el programador puede utilizar un `if` para comprobar si el divisor es 0 y, en tal caso, derivar el flujo de ejecución convenientemente sin tener que recurrir a las excepciones.

Sucede algo muy parecido con la excepción `IndexOutOfBounds`. Si un programador maneja bien los índices de un array, este error nunca debería llegar a producirse.