Grado en Ingeniería Informática
Introducción a la Programación
2021-2022

Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Clases abstractas

Cuando se hace uso de herencia los métodos pueden:

- Permanecer invariantes a lo largo de la jerarquía de clases. (modificador **`final`**)
- Necesitar ser modificados (redefinirlos).
- Una tercera posibilidad es que el método tenga sentido para las clases derivadas y que deba por tanto implementarse en ellas mientras que su implementación no tiene sentido en la clase base. Este sería el caso de los *métodos abstractos*.

Tendremos una **clase abstracta** en cuanto al menos uno de sus métodos sea abstracto (`abstract`), es decir no tenga implementación. En Java, el método abstracto se declara en la clase base con la palabra reservada **`abstract`** y se debe implementar en las clases derivadas.

Una **clase abstracta** es una clase que no permite instanciar objetos, se usa únicamente para definir subclases y al menos uno de sus métodos es *abstracto*.

Tema 3. Clases y Objetos

Clases abstractas e Interfaces

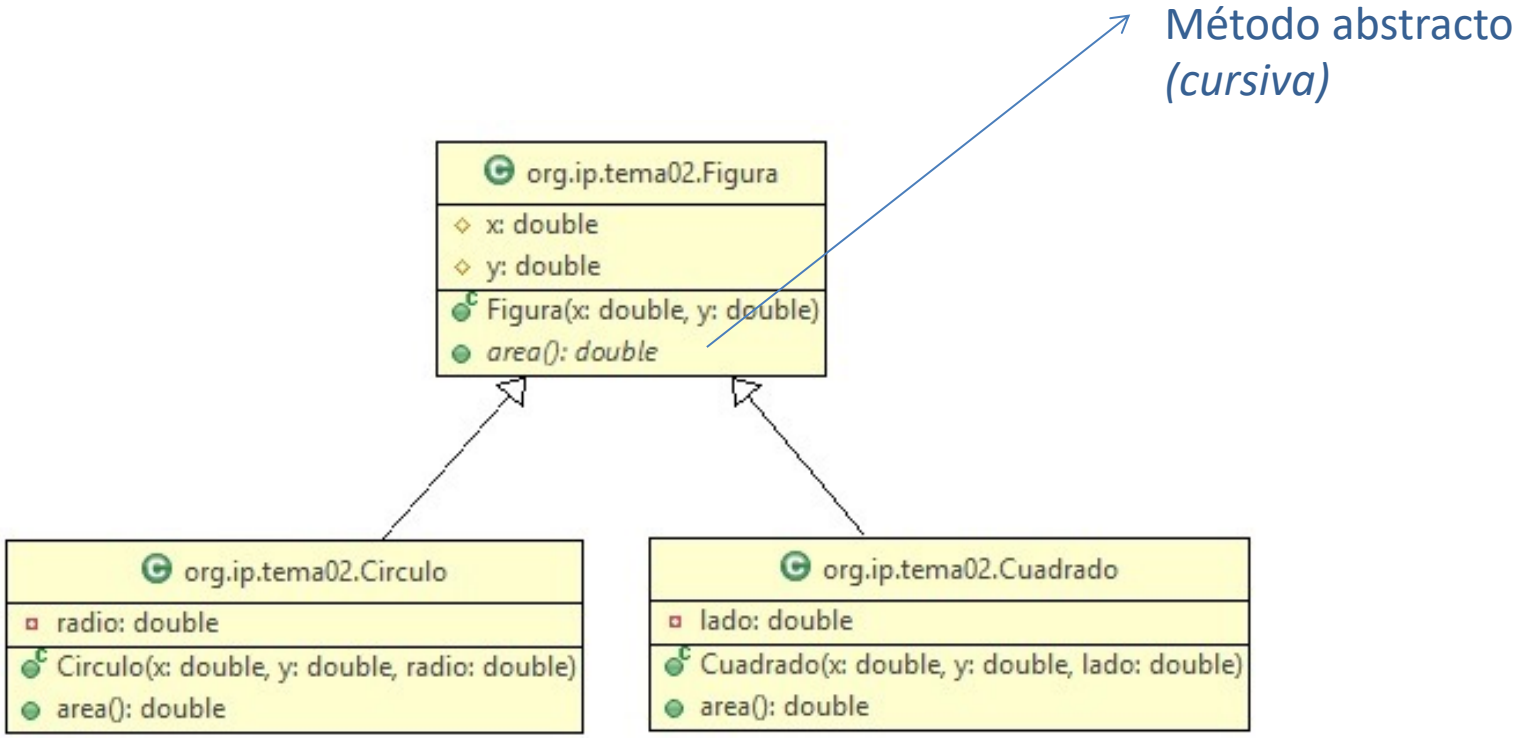
Clases abstractas

Aclaración método abstracto: En la jerarquía de herencia de clases, cuanto más abajo, más específica y más particular es la implementación de los métodos. Y cuanto más arriba, más general y más abstracta. Si subimos lo suficiente, podemos encontrarnos con un método que sabemos que deben tener todas las clases y subclases que heredan de la que estamos considerando, pero que sólo podemos implementar conociendo a la subclase a la que pertenece. Un método definido en una clase, pero cuya implementación se delega en las subclases, se conoce como **método abstracto**. Para declarar un método abstracto se le antepone la palabra reservada **`abstract`** y se declara el prototipo sin escribir el cuerpo de la función.

Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Ejemplo clase abstracta



Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Ejemplo clase abstracta

Clase base

```
package org.ip.tema02;

public abstract class Figura {
    protected double x;
    protected double y;

    public Figura(double x, double y) {
        super();
        this.x = x;
        this.y = y;
    }

    public abstract double area();
}
```

Método abstracto

Implementaciones métodos abstractos

Clases derivadas

```
package org.ip.tema02;

public class Circulo extends Figura {
    private double radio;

    public Circulo(double x, double y, double radio) {
        super(x, y);
        this.radio = radio;
    }
    @Override
    public double area() {
        return Math.PI * radio * radio;
    }
}
```

```
package org.ip.tema02;

public class Cuadrado extends Figura {
    private double lado;

    public Cuadrado(double x, double y, double lado) {
        super(x, y);
        this.lado = lado;
    }
    @Override
    public double area() {
        return lado * lado;
    }
}
```

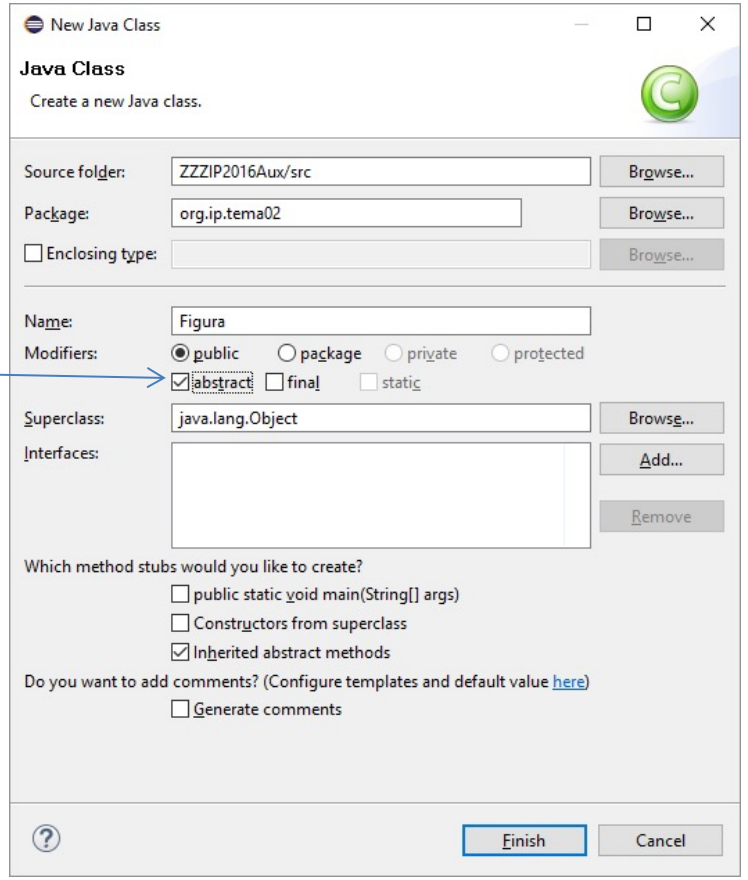
Tema 3. Clases y Objetos

Clases abstractas e Interfaces

En Eclipse

Para crear una clase abstracta: **New → Class**

Además, seleccionaremos el modificador **`abstract`** para indicar que esa clase es abstracta



Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Interfaces

Es una clase que no contiene ningún detalle de implementación.

La diferencia entre una clase abstracta y una interface es que, aunque ambas especifican como deben comportarse las clases derivadas, la interface no puede dar ningún detalle.

Las interfaces se declaran con la palabra reservada **`interface`** de manera similar a como se declaran las clases abstractas.

En la declaración solo puede aparecer declaraciones de métodos (cabeceras, sin implementación) y definiciones de constantes.

Su utilidad reside en definir unos requisitos mínimos que debe cumplir cualquier objeto que creemos a partir de una clase que la implemente.

Una interface **declara** pero **no implementa** las acciones mínimas que poseerá un objeto.

Para indicar que una clase implementa una interface se utiliza la palabra reservada **`implements`.**

Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Interfaces

Una interface puede consistir en más de un método, e incluso puede contener atributos. Diremos que una clase implementa una interface si implementa todos sus métodos (sus métodos abstractos).

Una interface es una declaración de funcionalidades, una especie de compromiso asumido por una clase.

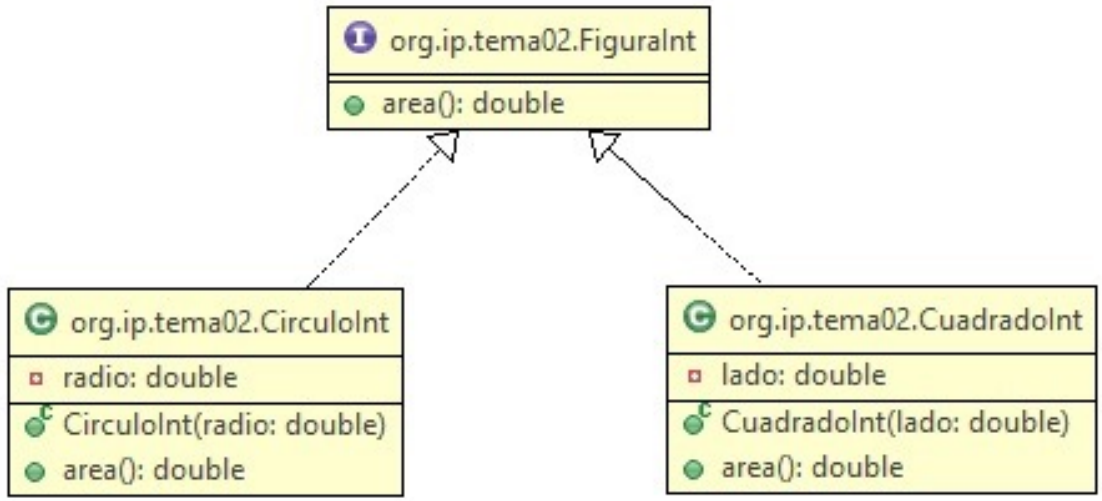
En la definición de una interface hay métodos (métodos abstractos) que son declarados, pero no implementados, métodos por defecto y métodos estáticos que sí son implementados en la clase. Y también puede haber atributos que son **`static`** y **`final`** por defecto.

Las interfaces no son instanciables (no se pueden crear objetos), pueden heredar unas de otras y **pueden crear variables cuyo tipo es una interface**, con las que podemos referenciar cualquier objeto de cualquier clase que la implemente.

Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Ejemplo interface



Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Ejemplo interface

Interface

```
package org.ip.tema02;

public interface FiguraInt {
    public double area();
}
```

Palabras reservadas:

`interface` (Interface)

`implements` (Clases que implementan la Interface)

Clases que implementan la interface

```
package org.ip.tema02;

public class CirculoInt implements FiguraInt {
    private double radio;

    public CirculoInt(double radio) {
        super();
        this.radio = radio;
    }
    @Override
    public double area() {
        return Math.PI * radio * radio;
    }
}
package org.ip.tema02;

public class CuadradoInt implements FiguraInt {
    private double lado;

    public CuadradoInt(double lado) {
        super();
        this.lado = lado;
    }
    @Override
    public double area() {
        return lado * lado;
    }
}
```

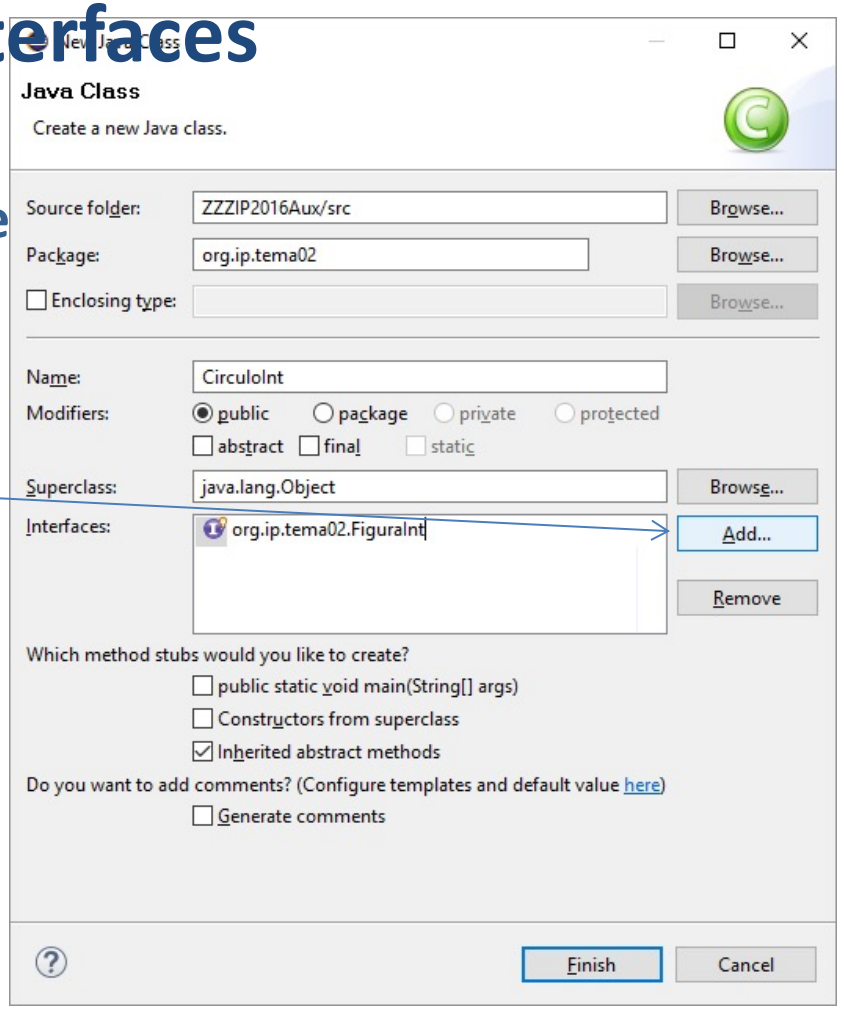
Tema 3. Clases y Objetos

Clases abstractas e Interfaces

En **Eclipse**

Para crear una **interface**: **New → Interface**

Para crear una **clase que implemente** una interface: **New → Class** y después **Add** y seleccionamos la interface que implementa



Tema 3. Clases y Objetos

Clases abstractas e Interfaces

Una clase puede implementar varias interfaces simultáneamente pero solo puede heredar de una clase (herencia simple, múltiples interfaces)

Ejemplo:

`public class` Cuadrado **`extends`** Figura **`implements`** Dibujable, Rotable

Esto quiere decir que la clase `Cuadrado`, heredaría de la clase `Figura` e implementa las interfaces `Dibujable` y `Rotable`.

Tema 3. Clases y Objetos

Clases abstractas e Interfaces

La interface **Comparable. Ejemplo importante**.

Es una interface para comparar objetos, definida en el paquete **java.lang**. Tiene un único método **`compareTo`** cuya implementación, en cualquier clase que implemente la interface, determinará el orden para comparar dos objetos (== 0, < 0, > 0 cuando los objetos sean iguales o el primero menor que el segundo o el primero mayor que el segundo respectivamente).

```
package  java.lang
public  interface Comparable{
        public int compareTo(Object o);
}
```

Tema 3. Clases y Objetos

Excepciones

Las excepciones son la manera que ofrece Java de **manejar los errores en tiempo de ejecución**.

Los lenguajes imperativos simplemente detienen la ejecución del programa cuando surge un error.

Las excepciones nos permiten escribir código para manejar el error y continuar (si lo estimamos conveniente) con la ejecución del programa.

Veamos un ejemplo de un programa que obtiene el cociente de dos números enteros.

Tema 3. Clases y Objetos

Excepciones

```
package org.ip.tema02;

import java.util.Scanner;

public class Cociente {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca dos enteros: ");
        int num1 = entrada.nextInt();
        int num2 = entrada.nextInt();
        System.out.println(num1 + " / " + num2 + " es " + (num1 / num2));
    }
}
```

Salidas

```
Introduzca dos enteros:
5 2
5 / 2 es 2
```

```
Introduzca dos enteros:
5 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at org.ip.tema02.Cociente.main(Cociente.java:12)
```

Tema 3. Clases y Objetos

Excepciones

Una posible solución sería incluir una sentencia **`if`**

```
package org.ip.tema02;

import java.util.Scanner;

public class CocienteConIf {

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

        System.out.println("Introduzca dos enteros: ");
        int num1 = entrada.nextInt();
        int num2 = entrada.nextInt();

        if (num2 != 0)
            System.out.println(num1 + " / " + num2 + " es " + (num1 / num2));
        else
            System.out.println("No se puede hacer una división por cero");
    }
}
```

Salida

```
Introduzca dos enteros:
5 0
No se puede hacer una división por cero
```

Esta forma de gestionar los errores es propia de los lenguajes imperativos que no disponen de otros mecanismos.

Tema 3. Clases y Objetos

Excepciones

En lenguajes de programación orientados a objetos como **Java**, el propio lenguaje incorpora la gestión de errores proporcionando construcciones especiales para ello.

En el ejemplo cociente se produce un error de ejecución (**lanza una excepción**) al intentar dividir por cero.

Cuando se detecta el error, por defecto se interrumpe la ejecución. Pero podemos evitarlo.

La estructura **`try-catch-finally`** nos permitirá capturar excepciones, es decir, reaccionar a un error de ejecución.

De este modo podremos imprimir mensajes de error *a la medida* y continuar con la ejecución del programa si consideramos que el error no es demasiado grave.

Para ver cómo funcionan vamos a modificar el ejemplo anterior, pero asegurándonos ahora de que **capturamos las excepciones**.

Tema 3. Clases y Objetos

Excepciones

```
try {
        // Código que puede hacer que se eleve la excepción
}
catch (TipoExcepcion e) {
        // Gestor de la excepción
}
```

El comportamiento de **Java** es el siguiente:

Si en la ejecución del código dentro del bloque **`try`** se lanza una excepción de tipo TipoExcepcion (o descendiente de éste), Java omite la ejecución del resto del código en el bloque **`try`** y ejecuta el código situado en el bloque **`catch`** (gestor).

Tema 3. Clases y Objetos

Excepciones

`try`

El bloque de código donde se prevé que se lance una excepción. Al encerrar el código en un bloque **`try`** es como si dijéramos: *Prueba a usar estas instrucciones y mira a ver si se produce alguna excepción*. El bloque **`try`** tiene que ir seguido, al menos, por una cláusula **`catch`** o una **`finally`**.

`catch`

El código que se ejecuta cuando se lanza la excepción. Controla cualquier excepción que cuadre con su argumento. Se pueden colocar varios **`catch`** sucesivos, cada uno controlando un tipo de excepción diferente.

`finally`

Bloque que se ejecuta siempre, haya o no excepción.

Tema 3. Clases y Objetos

Excepciones

El ejemplo quedaría:

```
package org.ip.tema02;

import java.util.Scanner;

public class CocienteConExcepcion {

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

        System.out.println("Introduzca dos enteros: ");
        int num1 = entrada.nextInt();
        int num2 = entrada.nextInt();
        try {
            System.out.println(num1 + " / " + num2 + " es " + (num1 / num2));
        } catch (Exception ex) {
            System.out.println("Excepción: un entero "
                    + "no se puede dividir por cero");

        }
        System.out.println("La ejecución continua ...");
    }
}
```

Salida

```
Introduzca dos enteros:
5 0
Excepción: un entero no se puede dividir por cero
La ejecución continua ...
```

Tema 3. Clases y Objetos

Excepciones

Podríamos utilizar un método

```
package org.ip.tema02;

import java.util.Scanner;

public class CocienteConMetodo {
    public static int cociente(int numero1, int numero2) {
        return numero1 / numero2;
    }

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

        System.out.println("Introduzca dos enteros: ");
        int num1 = entrada.nextInt();
        int num2 = entrada.nextInt();
        try {
            int resultado = cociente(num1, num2);
            System.out.println(num1 + " / " + num2 + " es "
            + resultado);
        } catch (Exception ex) {
            System.out.println("Excepción: un entero "
                    + "no se puede dividir por cero");
        }
        System.out.println("La ejecución continua ...");
    }
}
```

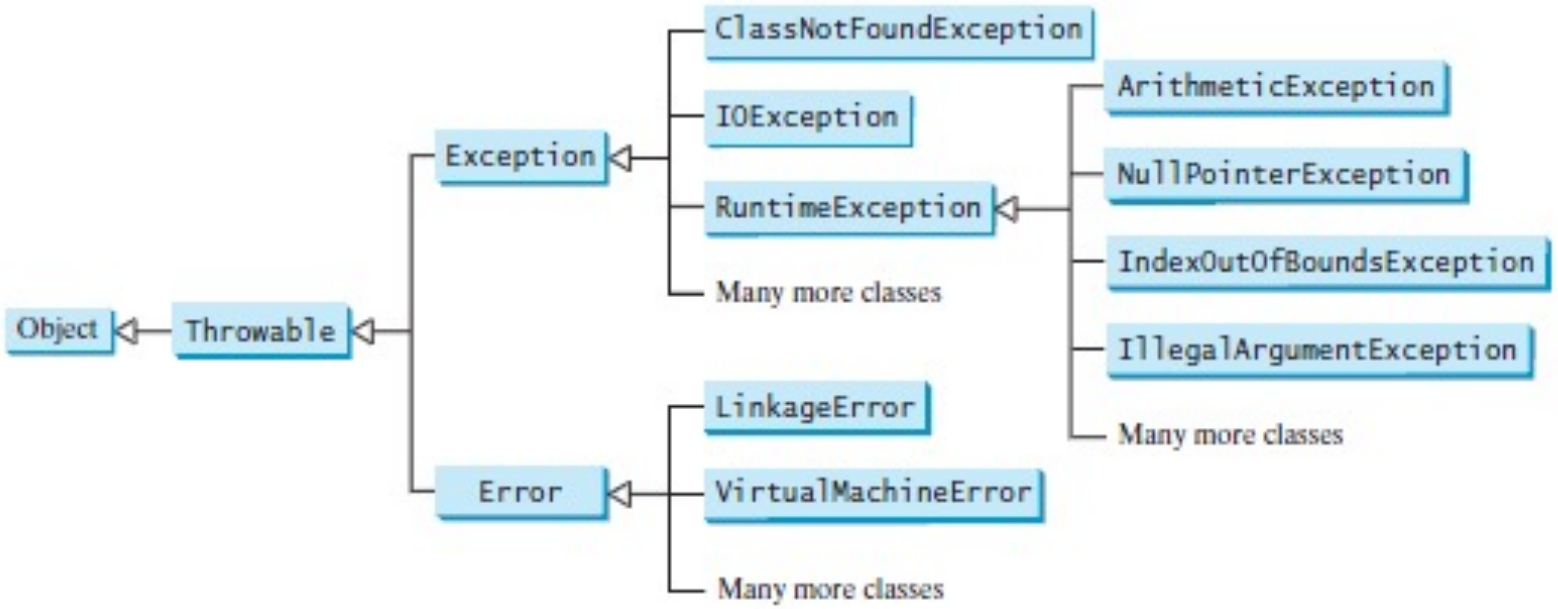
Salida

```
Introduzca dos enteros:
5 0
Excepción: un entero no se puede dividir por cero
La ejecución continua ...
```

Tema 3. Clases y Objetos

Excepciones

Jerarquía de clases para el manejo de excepciones. Excepciones predefinidas en Java



Tema 3. Clases y Objetos

Excepciones

`Throwable`

Superclase que engloba a todas las excepciones

`Error`

Representa los errores graves causados por el sistema (JVM, etc.); no son tratados por los programas.

`Exception`

Define las excepciones que los programas deberían tratar (IOException, ArithmeticException, etc.).

Tema 3. Clases y Objetos

Excepciones

La clase **`Exception`**

Cuando se lanza una excepción, lo que se hace es activar un ejemplar de **`Exception`** o de alguna de sus subclases.

Normalmente las clases derivadas nos permiten distinguir entre los distintos tipos de excepciones.

Esta clase tiene dos **constructores** y dos **métodos destacables**:

`Exception()`: crea una excepción si ningún mensaje específico.

`Exception(String)`: crea una excepción con un mensaje que detalla el tipo de excepción.

`String getMessage()`: el mensaje detallado, si existe (o `null`).

`void printStackTrace()`: muestra una traza que permite ver donde se generó el error (muy útil para depurar).

Tema 3. Clases y Objetos

Excepciones

Captura de excepciones

A **catch** le sigue, entre paréntesis, la declaración de una excepción. Es decir, el nombre de una clase derivada de **`Exception`** (o la propia **`Exception`**) seguido del nombre de una variable. Si se lanza una excepción que es la que deseamos capturar (o una derivada de la misma) se ejecutará el código que contiene el bloque. Así, por ejemplo:

```
catch (Exception e) { ... }
```

se ejecutará siempre que se produzca una excepción del tipo que sea, ya que todas las excepciones se derivan de **`Exception`**.

Tema 3. Clases y Objetos

Excepciones

Captura de excepciones

Se pueden colocar varios bloques **`catch`.** Si es así, se comprobará, en el mismo orden en que se encuentren esos bloques **`catch`**, si la excepción lanzada es la que se trata en el bloque **`catch`**; si no, se pasa a comprobar el siguiente.

Eso sí, **sólo se ejecuta un bloque `catch`**. En cuanto se captura la excepción se deja de comprobar el resto de los bloques.

Veamos algunos ejemplos:

```
// Bloque 1
try {
   // Bloque 2
} catch (Exception error){
   //  Bloque 3
}
   // Bloque 4
```

Sin excepciones: 1 → 2 → 4

Con una excepción en el bloque 2: 1 → 2* → 3 → 4

Con una excepción en el bloque 1: 1*

Tema 3. Clases y Objetos

Excepciones

Captura de excepciones

```
// Bloque 1
try {
   //  Bloque 2
} catch (ArithmeticException ae){
   //  Bloque 3
} catch (NullPointerException ne)
   //  Bloque 4
}
// Bloque 5
```

Sin excepciones: 1 → 2 → 5

Excepción de tipo aritmético: 1 → 2* → 3 → 5

Acceso a un objeto nulo (null): 1 → 2* → 4 → 5

Excepción de otro tipo diferente: 1 → 2*

Tema 3. Clases y Objetos

Excepciones

Captura de excepciones

```
// Bloque 1
try {
   //  Bloque 2
} catch (ArithmeticException ae){
   //  Bloque 3
} catch (Exception  e)
   //  Bloque 4
}
// Bloque 5
```

Sin excepciones: 1 → 2 → 5

Excepción de tipo aritmético: 1 → 2* → 3 → 5

Excepción de otro tipo diferente: 1 → 2* → 4 → 5

Tema 3. Clases y Objetos

Excepciones

Captura de excepciones

```
// Bloque 1
try {
   //  Bloque 2
} catch (Exception  e)
   //  Bloque 3
} catch (ArithmeticException ae)
   //  Bloque 4
}
// Bloque 5
```

¡¡OJO!!

Sin excepciones: 1 → 2 → 5

Excepción de tipo aritmético: 1 → 2* → 3 →5

Excepción de otro tipo diferente: 1 → 2* → 3 → 5

¡El bloque 4 nunca se ejecuta!

Tema 3. Clases y Objetos

Excepciones

Excepciones a medida o propias

El programador puede crear sus propias excepciones, si resulta que ninguna de las predefinidas es adecuada.

Se tratan igual que las excepciones predefinidas.

Para ello, se define una clase derivada de **`Exception`** (o de la clase deseada).

Se suele agregar un constructor con el mensaje de la excepción,

que se inicializa en el constructor llamando al de la clase padre.

Además, toda excepción tiene un método **`getMessage()`** que devuelve un **`String`** con el mensaje.

Tema 3. Clases y Objetos

Ejemplo

```
package org.ip.tema02;

public class ExcepcionIntervalo extends Exception {
    public ExcepcionIntervalo (String msg) {
        super(msg);
    }
}
```

```
package org.ip.tema02;

public class ExcepcionIntervaloPrincipal {

    public static void rango(int num, int den) throws ExcepcionIntervalo {
        if ((num > 100) || (den < -5))
            throw new ExcepcionIntervalo("Numeros fuera de rango");
    }
    public static void main(String[] args) {
        int numerador = 120;
        int denominador = 5;
        int cociente;
        try {
            rango(numerador, denominador);
            cociente = numerador / denominador;
            System.out.println("El cociente es: " + cociente);
        }
        catch (ExcepcionIntervalo ex) {
            System.out.println("Al calcular el rango se ha producido una");
            System.out.println("excepcion con el mensaje: " + ex.getMessage());
        }
        System.out.println("FIN");
    }
}
```

Declaración de las excepciones que se pueden lanzar en el método

Lanzar la excepción

Capturar la excepción

Tratar la excepción

Salida

```
Al calcular el rango se ha producido una
excepcion con el mensaje: Numeros fuera de rango
FIN
```

No es necesario declarar las **RuntimeException** y derivadas

```
package org.ip.tema02;

import java.util.Scanner;

public class CocienteConMetodoYExcepcion {

    public static int cociente(int numero1, int numero2) throws ArithmeticException {
        if (numero2 == 0)
            // Se lanza la excepcion
            throw new ArithmeticException("No se puede dividir por cero");
        return numero1 / numero2;
    }

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca dos enteros: ");
        int num1 = entrada.nextInt();
        int num2 = entrada.nextInt();
        try {
            int resultado = cociente(num1, num2);
            System.out.println(num1 + " / " + num2 + " es "
            + resultado);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
            System.out.println(ex.toString());
            ex.printStackTrace();
        }
        System.out.println("La ejecución continua ...");
        System.out.println("El producto de los valores es: " + (num1 * num2));
    }
}
```

System.out.println(ex.getMessage())

System.out.println(ex.toString())

Salida

```
Introduzca dos enteros:
4 0
No se puede dividir por cero
java.lang.ArithmeticException: No se puede dividir por cero
java.lang.ArithmeticException: No se puede dividir por cero
        at org.ip.tema02.CocienteConMetodoYExcepcion.cociente(CocienteConMetodoYExcepcion.java:10)
        at org.ip.tema02.CocienteConMetodoYExcepcion.main(CocienteConMetodoYExcepcion.java:20)
La ejecución continua ...
El producto de los valores es: 0
```

ex.printStackTrace()

¡MUCHAS GRACIAS!

UNIVERSIDAD DE ALMERÍA