UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica etsinf

Tema 3. Elementos de la POO: Herencia y Tratamiento de Excepciones

Programación (PRG)

Departamento de Sistemas Informáticos y Computación

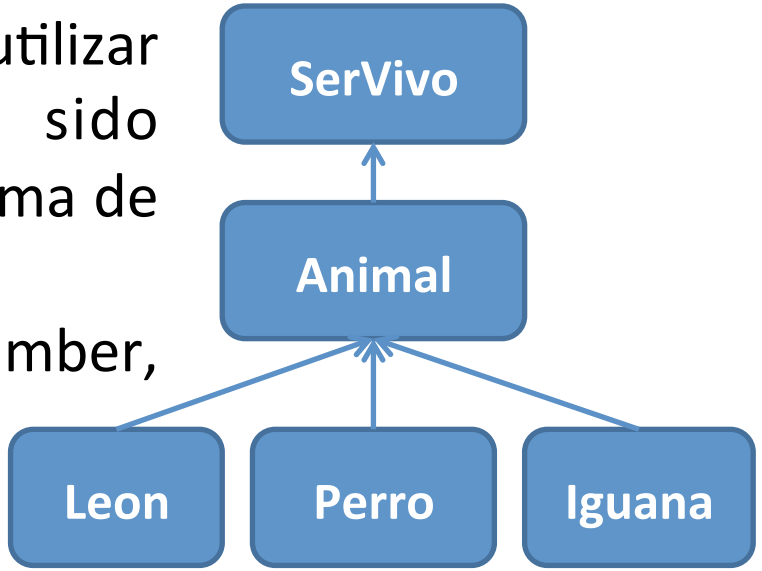
Contenidos

- Conceptos de la POO
  - Herencia
  - La jerarquía de clases en Java
  - La herencia en la documentación del API de Java
- Tratamiento de excepciones en Java
  - La jerarquía `Throwable`
  - Excepciones de usuario
  - Instrucción `try-catch-finally`
  - Instrucción `throws`
  - Instrucción `throw`

Introducción a la Herencia

- La herencia es el mecanismo que proporcionan los lenguajes de programación orientados a objetos para reutilizar el diseño de clases ya existentes para definir nuevas clases.
- Permite modelar de forma intuitiva una relación de tipo ES UN, definiendo una jerarquía de clases.

- Las clases pueden, de esta manera, utilizar atributos y métodos que hayan sido definidos en clases que estén por encima de ellas en la jerarquía.
- Ejemplo de la jerarquía de la clase Number, disponible en el API de Java.



Sobre la necesidad de la herencia

- Se pide construir la clase Estudiante con los atributos *nombre*, *edad*, y *créditos* matriculados. Ya se dispone de la clase Persona.

```
class Persona {
  private String nombre;
  private int edad;
  public Persona(String nombre, int edad) {
        this.nombre = nombre; this.edad = edad; }
  public String getNombre() { return this.nombre; }
  public int getEdad() { return this.edad; }
  public String toString() {
    return " Nombre: " + this.nombre + " Edad: " + this.edad; }
}
```

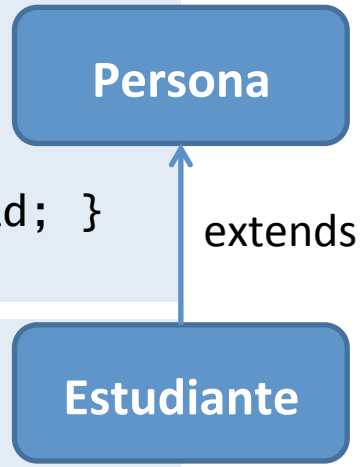
- Opciones posibles:
  - Inapropiada: Ignorar la clase Persona y construir la clase Estudiante con tres atributos (edad, nombre y créditos). Se repite la declaración de atributos y métodos ya realizado en la clase Persona.
  - Apropiada: Usar la herencia para definir la clase Estudiante en base a la clase Persona.

Construyendo una subclase

- Construcción de la clase derivada Estudiante a partir de la clase base Persona.
  - La clase Estudiante hereda (puede usar) todos los atributos y métodos que no son privados en Persona.

```
class Persona {
  protected String nombre;
  protected int edad;
  public Persona(String nombre, int edad) {
        this.nombre = nombre; this.edad = edad; }
  public String getNombre() { return this.nombre; }
  public int getEdad() { return this.edad; }
  public String toString() {
    return "Nombre: "+ this.nombre + " Edad: " + this.edad; }
}
```

```
class Estudiante extends Persona {
  private int creditos;
  public Estudiante(String nombre, int edad) {
        super(nombre,edad); this.creditos = 60; }
  public int getCreditos() { return this.creditos; }
}
```



Utilizando una subclase

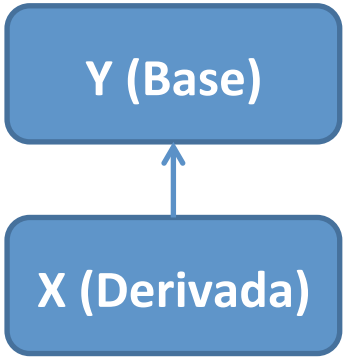
```
class TestEstudiantePersona {
  public static void main(String[] args) {
     Estudiante e = new Estudiante("Luisa Garcia",20);
     Persona p = new Persona("Luisa Garcia",20);
     System.out.println("Persona: " + p.toString());
     System.out.println(e.getNombre() + " : " +
                        e.getCreditos() + " créditos");
  }
}
```

- Se puede invocar a los métodos declarados en Persona desde un objeto Estudiante ya que Estudiante hereda de Persona.
  - La clase Estudiante puede acceder a los atributos y métodos no privados de la clase Persona.
- Muestra por pantalla:

```
Persona: Nombre: Luisa Garcia Edad: 20
Luisa Garcia: 60 créditos
```

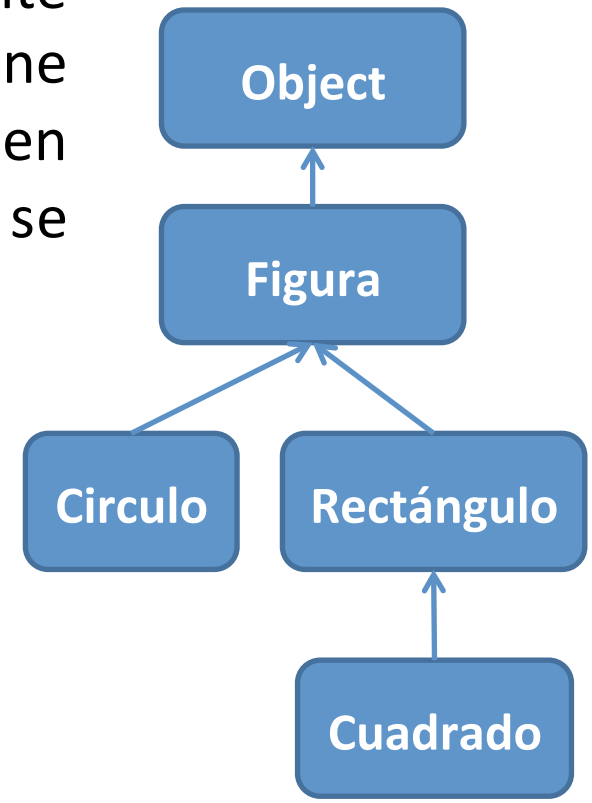
Sobre las jerarquías de clases

- Si X ES UN(A) Y,
  - Se dice que la clase derivada X es una variación de la clase base Y
  - Se dice que X e Y forman una jerarquía, donde la clase X es una subclase (o clase derivada) de Y e Y es una superclase (o clase padre) de X
  - La relación es transitiva: si X ES UN(A) Y e Y ES UN(A) Z, entonces X ES UN(A) Z
  - X puede referenciar todos los atributos y métodos que no sean privados en Y
  - X es una clase completamente nueva e independiente (los cambios en X no afectan a Y)
- Java no soporta herencia múltiple.
  - Una clase solo puede heredar como máximo de otra.



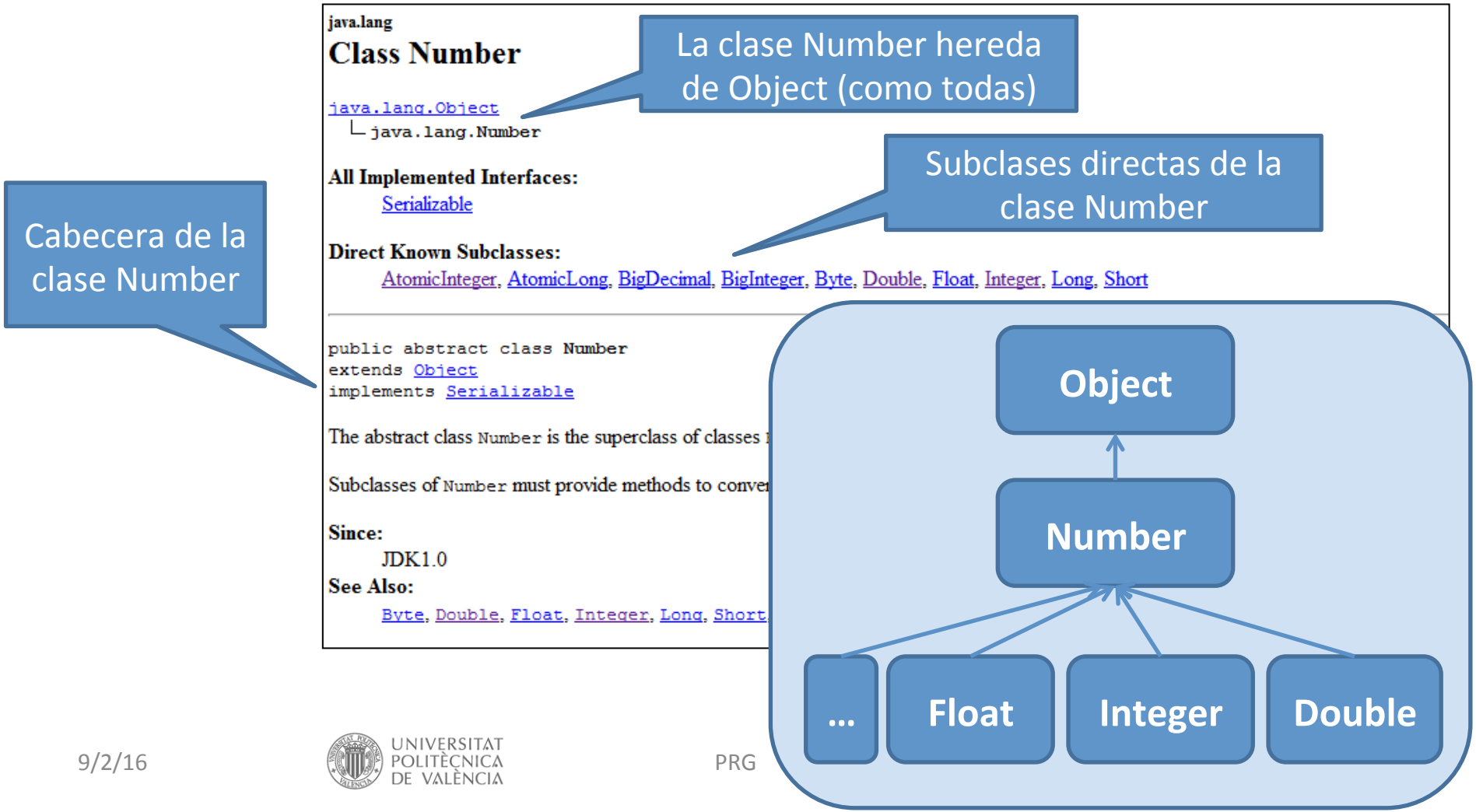
Jerarquías de Clases

- Cualquier clase Java hereda implícitamente de la clase predefinida Object. Ésta define los métodos que pueden ser invocados en cualquier objeto Java. Entre otros, se encuentran:
  - public String toString()
  - public boolean equals(Object x)
- Ejemplo de jerarquía de clases:
  - Una Figura ES UN Object
  - Un Círculo ES UNA Figura
  - Un Rectángulo ES UNA Figura
  - Un Cuadrado ES UN Rectángulo (cuya base y altura coinciden)



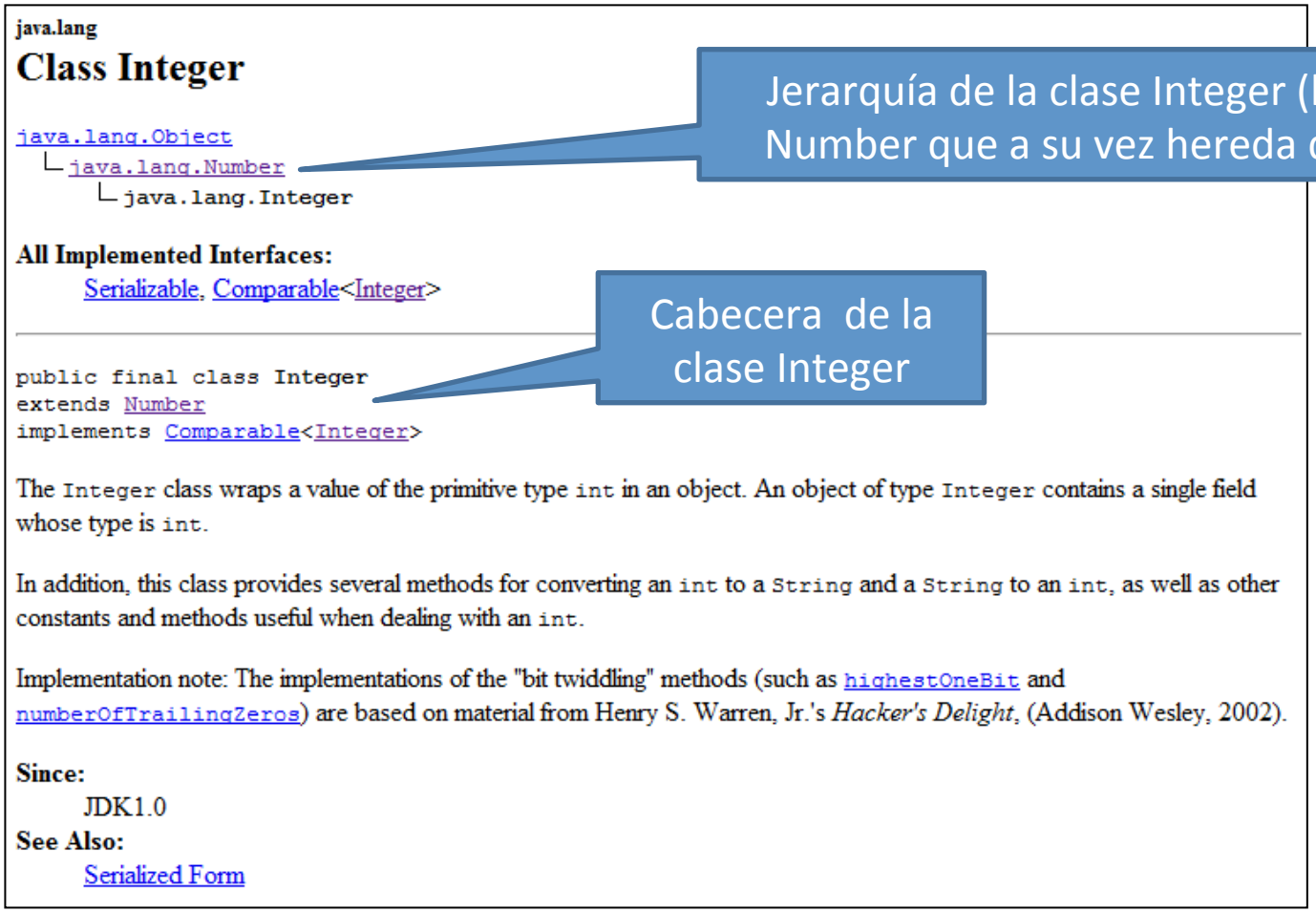
La herencia en la documentación de Java (I)

- Extracto de la documentación de la clase Number en el API Java.



La herencia en la documentación de Java (II)

- Extracto de la documentación de la clase Integer en el API Java.



java.lang

Class Integer

java.lang.Object
  └ java.lang.Number
    └ java.lang.Integer

All Implemented Interfaces:

Serializable, Comparable<Integer>

```
public final class Integer
extends Number
implements Comparable<Integer>
```

The `Integer` class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Implementation note: The implementations of the "bit twiddling" methods (such as `highestOneBit` and `numberOfTrailingZeros`) are based on material from Henry S. Warren, Jr.'s *Hacker's Delight*, (Addison Wesley, 2002).

Since:

JDK1.0

See Also:

Serialized Form