Herencia

Java y Servicios Web I
Master en Ingeniería Matemática

Manuel Montenegro
Dpto. Sistemas Informáticos y Computación

Desp. 467 (Mat)

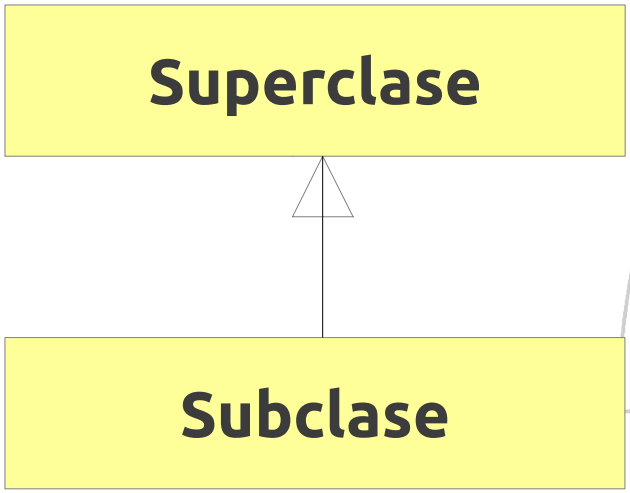
montenegro@fdi.ucm.es

Contenidos

- Extensión mediante herencia.
- Reescritura de métodos.
- Modificador de acceso `protected`.
- Jerarquía de clases.
- La clase `Object`.

Herencia

- Mecanismo de la programación orientada a objetos diseñado para la reutilización y la extensibilidad.
- Permite añadir funcionalidad a una clase ya existente.

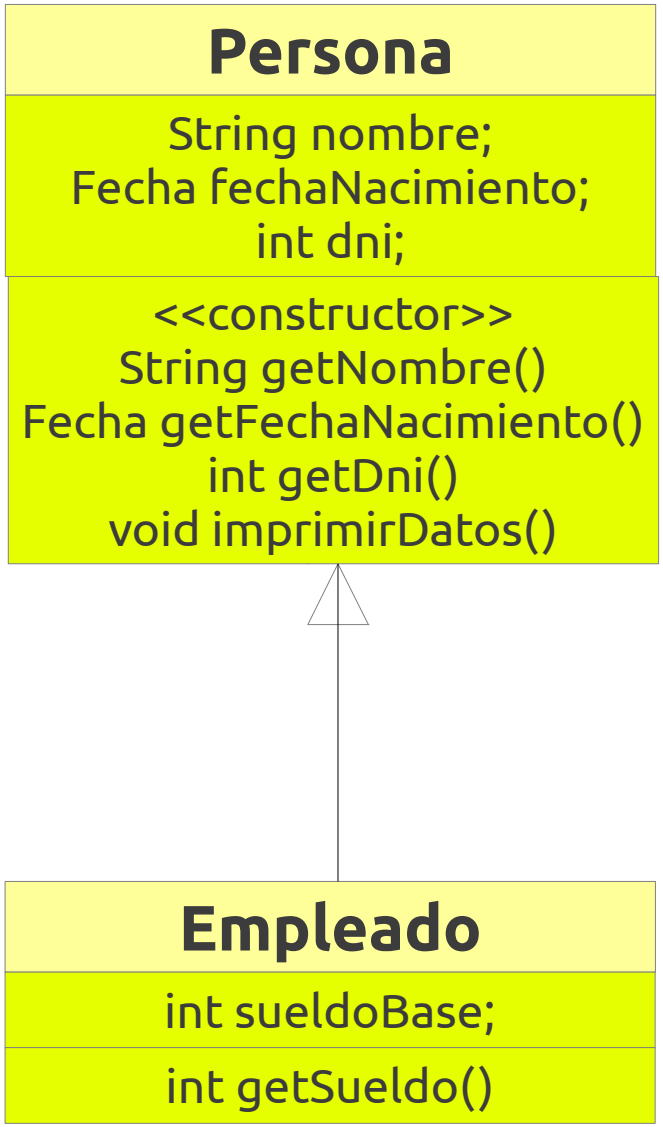


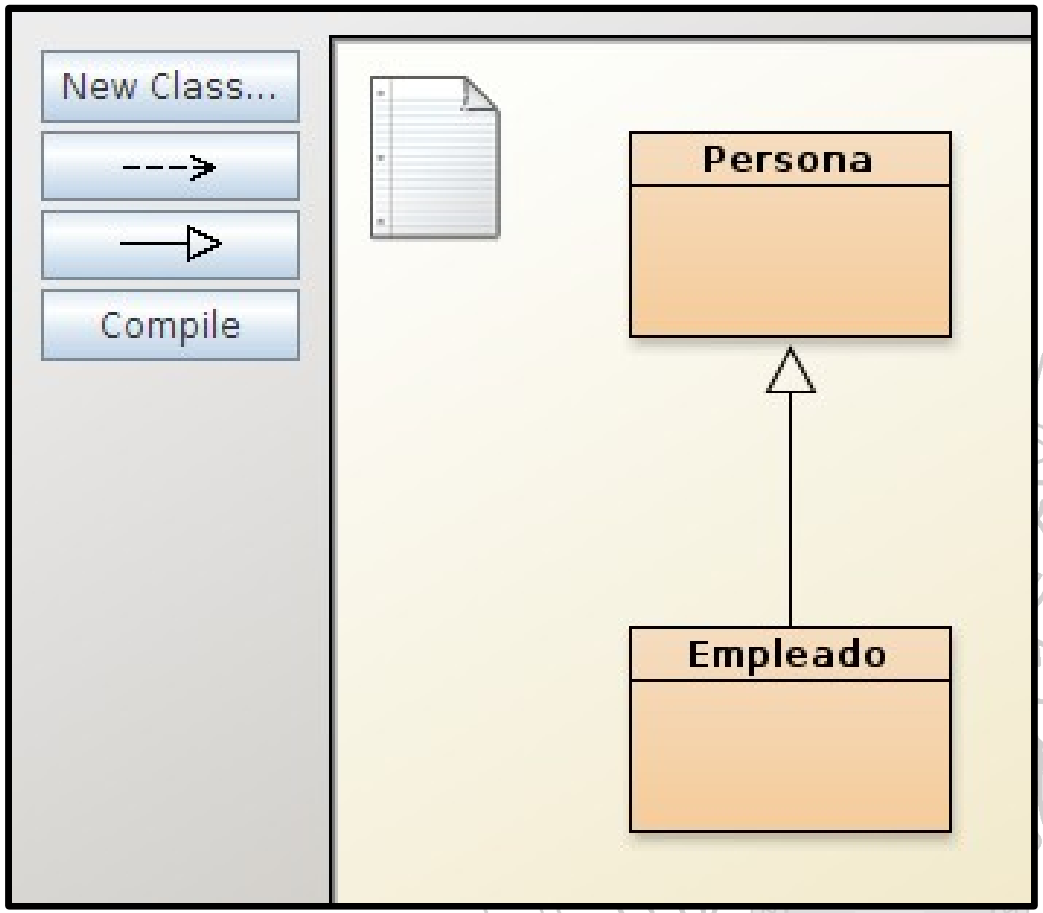
Ejemplo: Persona

Persona
String nombre; Fecha fechaNacimiento; int dni;
<<constructor>> String getNombre() Fecha getFechaNacimiento() int getDni() void imprimirDatos()

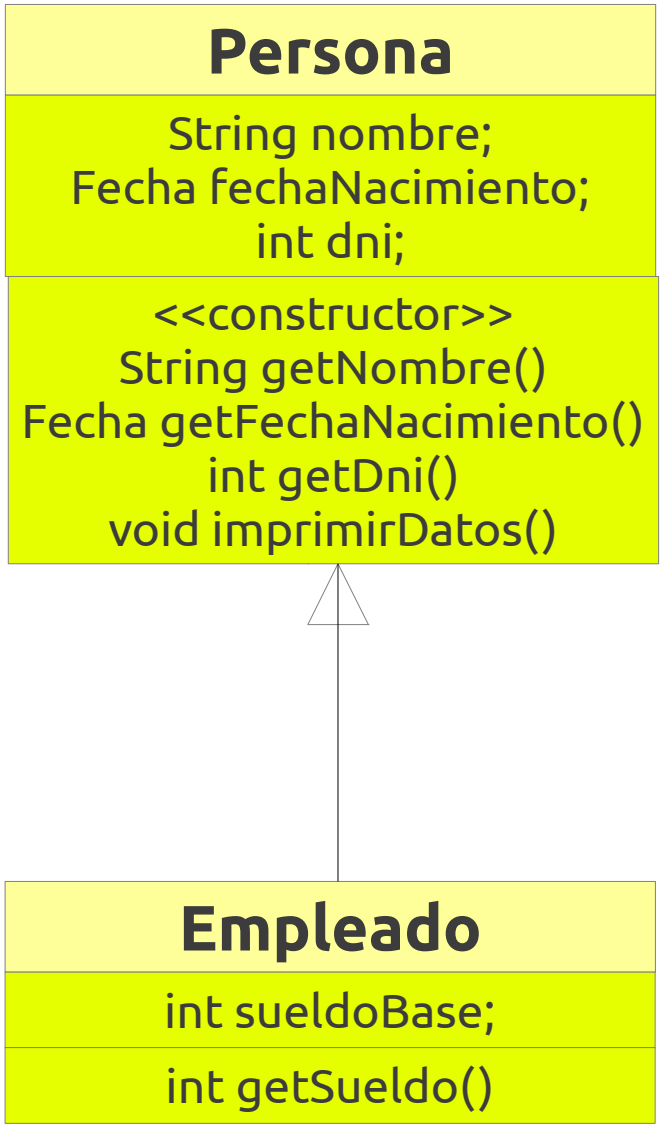
```
public class Persona {

  private String nombre;
  private Fecha fechaNacimiento;
  private int dni;

  public Persona(String nombre,
                 Fecha fechaNacimiento, int dni) {

     this.nombre = nombre;
     this.fechaNacimiento = fechaNacimiento;
     this.dni = dni;

  }

  public void imprimirDatos() {
     System.out.print("DNI: ");
     System.out.println(dni);
     System.out.println("NOMBRE: " + nombre);
     System.out.print("FECHA DE NACIMIENTO: ");
     fechaNacimiento.imprimir();
     System.out.println();

  }
}
```

Ejemplo: Persona





Ejemplo: Persona

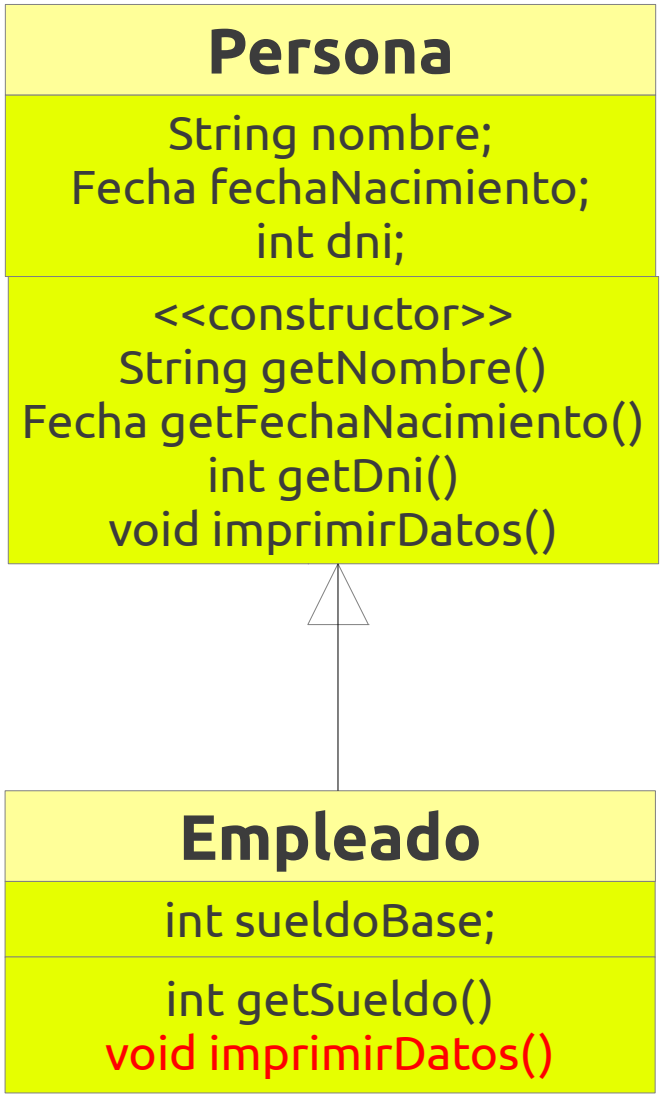



```
public class Empleado extends Persona {
  private int sueldoBase;
  …
  public int getSueldo() {
    return sueldoBase;
  }
}
```

Contenidos

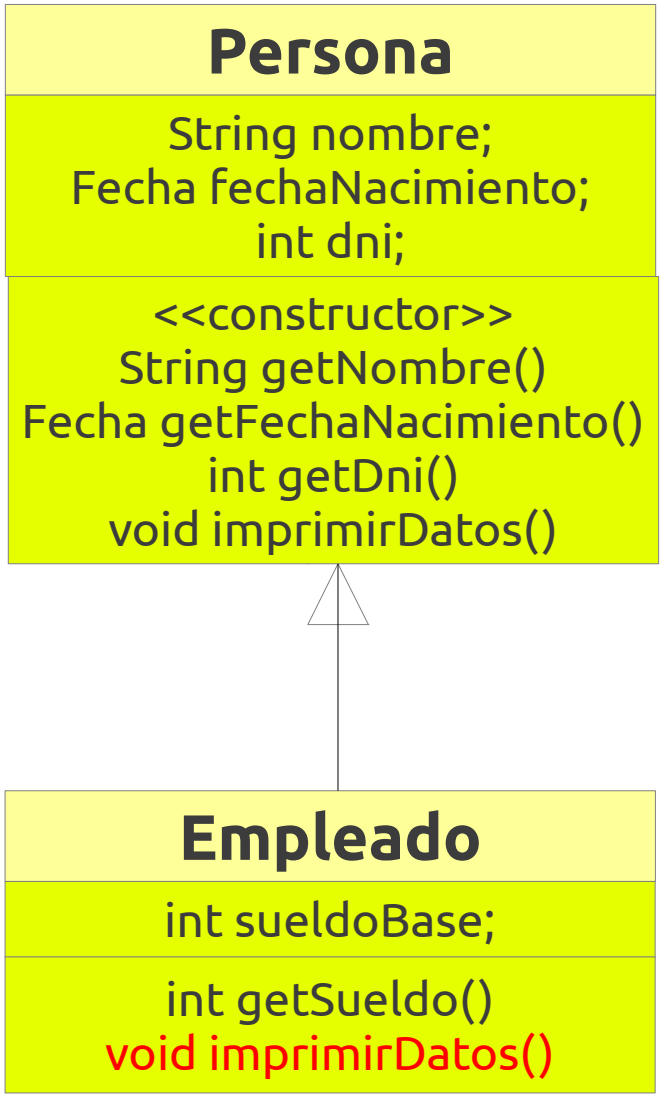
- Extensión mediante herencia.
- Reescritura de métodos.
- Modificador de acceso `protected`.
- Jerarquía de clases.
- La clase `Object`.

Reescritura de métodos



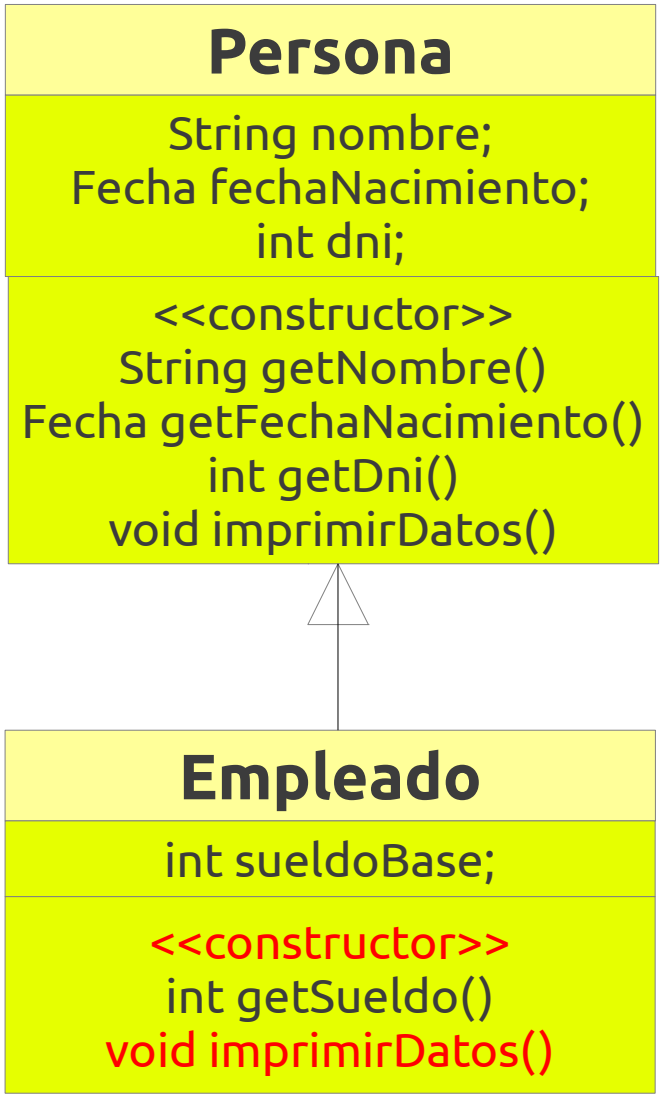

```
public class Empleado extends Persona {

 …
 public void imprimirDatos() {
      System.out.print("DNI: ");
      System.out.println(dni);
      System.out.println("NOMBRE: " + nombre);
      System.out.print("FECHA DE NACIMIENTO: ");
      fechaNacimiento.imprimir();
      System.out.println();
      System.out.print(“SUELDO: “);
      System.out.println(sueldoBase);
 }
}
```

Reescritura de métodos



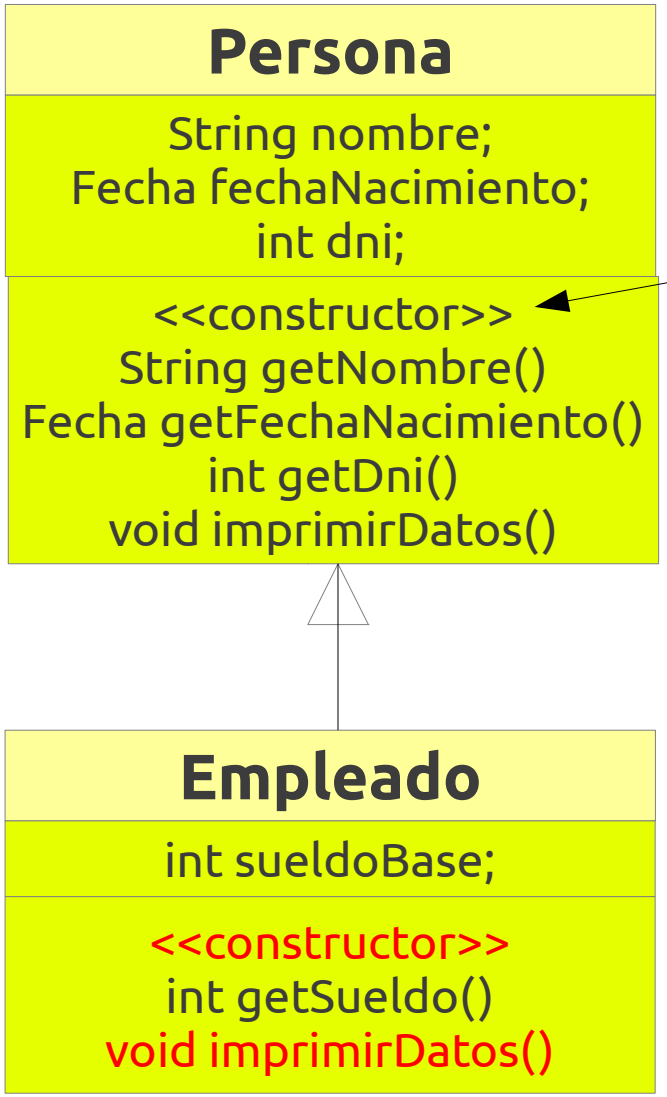

```
public class Empleado extends Persona {

 …
 public void imprimirDatos() {
      super.imprimirDatos();
      System.out.print("SUELDO: ");
      System.out.println(sueldoBase);
 }
}
```

Reescritura de métodos




```
public class Empleado extends Persona {


  public Empleado(String nombre, Fecha fechaNacimiento,
                  int dni, int sueldoBase) {
     this.nombre = nombre;
     this.fechaNacimiento = fechaNacimiento;
     this.dni = dni;
     this.sueldoBase = sueldoBase;
  }

}
```

Reescritura de métodos




```
public class Empleado extends Persona {

   public Empleado(String nombre, Fecha fechaNacimiento,
                   int dni, int sueldoBase) {
      super(nombre, fechaNacimiento, dni);
      this.sueldoBase = sueldoBase;
   }
   …
}
```

- La primera línea del constructor de la subclase ha de llamar al constructor de la superclase.
- Si no lo hace, se intentará llamar automáticamente al constructor por defecto.

Contenidos

- Extensión mediante herencia.
- Reescritura de métodos.
- Modificador de acceso `protected`.
- Jerarquía de clases.
- La clase `Object`.

Modificadores de acceso

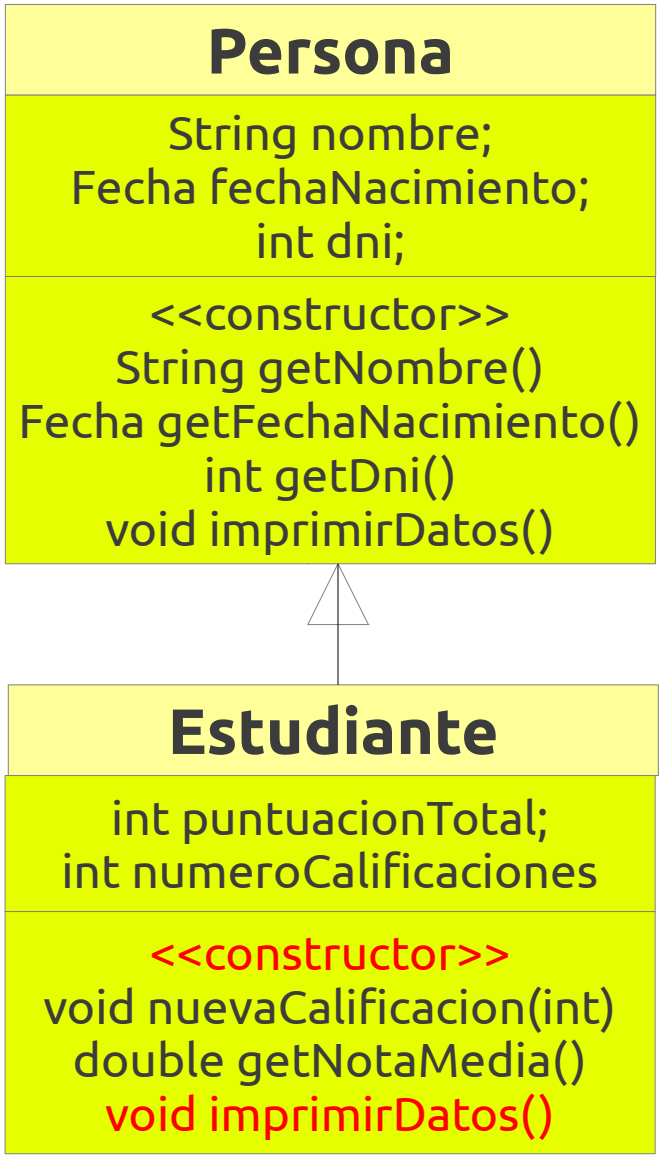
- ¿Qué elementos de la superclase son accesibles desde la subclase?
  - Los métodos y atributos `public` son accesibles.
  - Los métodos y atributos `private` **no** son accesibles, aunque sí se heredan.
- Si queremos que un atributo o método privado sea accesible desde las subclases, tendremos que definirlo en la superclase con el modificador `protected`.

```
public class Empleado extends Persona {
  protected int sueldoBase;
}
```

Modificadores de acceso

	public	protected	private
Dentro de la clase			
Subclases			
Otro sitio			

Ejemplo: Estudiante

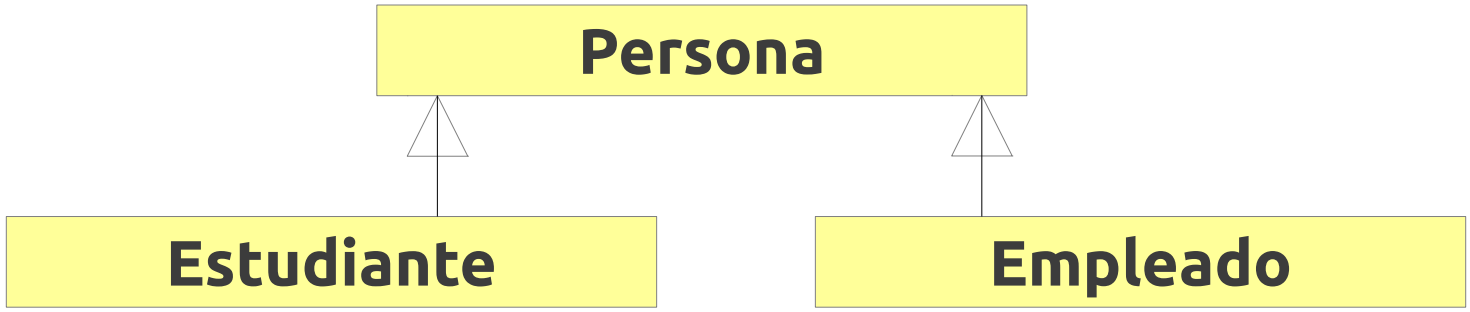



```
public class Estudiante extends Persona {
  private int puntuacionTotal;
  private int numeroCalificaciones;

  public Empleado(String nombre, Fecha fechaNacimiento,
                  int dni) {
     super(nombre, fechaNacimiento, dni);
     this.sueldoBase = sueldoBase;
     this.puntos = 0;
     this.numeroCalificaciones = 0;
  }
  public void nuevaCalificacion(int puntos) {
    puntuacionTotal += puntos;
    numeroCalificaciones++;
  }
  public double getNotaMedia() {
    return ((double) puntos)/numeroCalificaciones;
  }

  public void imprimirDatos() {
    super.imprimirDatos();
    System.out.println("NOTA MEDIA: ");
    System.out.println(getNotaMedia());
  }
}
```

Contenidos

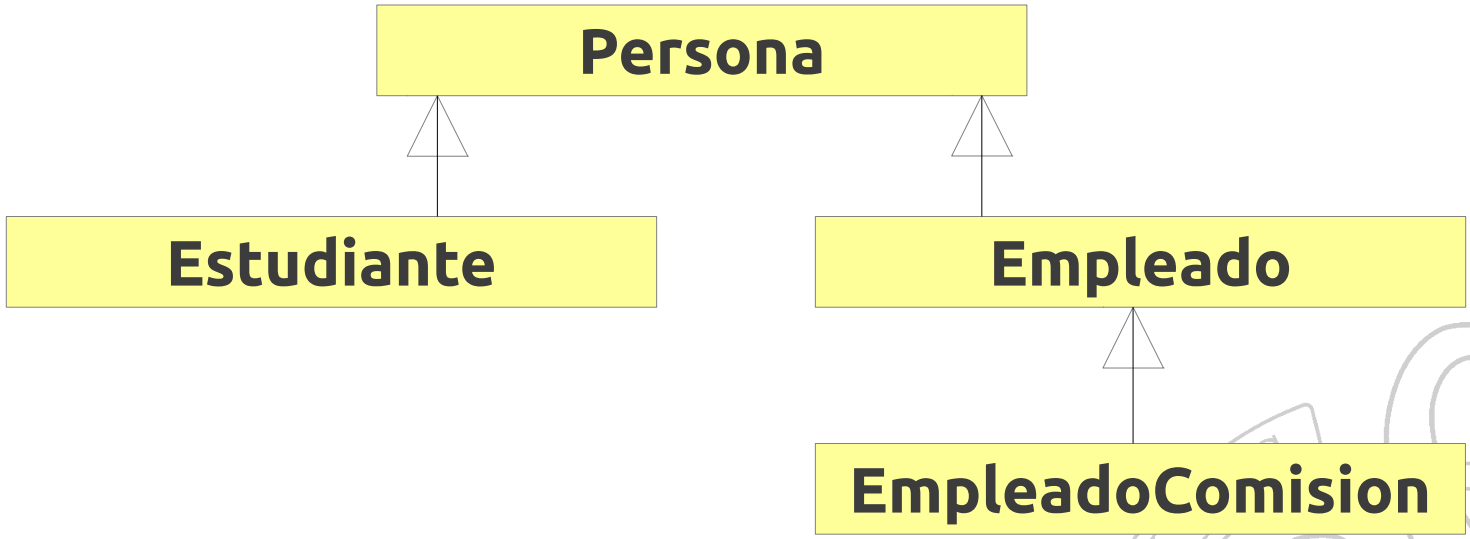
- Extensión mediante herencia.
- Reescritura de métodos.
- Modificador de acceso `protected`.
- Jerarquía de clases.
- La clase `Object`.

Jerarquía de clases



- Conceptualmente, la relación entre las superclases y las subclases es de tipo **es-un**.
- Representan concreciones sobre el concepto de Persona.

Jerarquía de clases



- La jerarquía puede extenderse a varios niveles.
- No se permite herencia múltiple: una clase sólo tiene una superclase.

Ejemplo: EmpleadoComision

```
public class EmpleadoComision extends Empleado
{
    private double porcentajeComision;
    private int dineroVentas;


    public EmpleadoComision(String nombre, Fecha fechaNacimiento, int dni,
                            int sueldoBase, double porcentajeComision)  {
        super(nombre, fechaNacimiento, dni, sueldoBase);
        this.porcentajeComision = porcentajeComision;
        this.dineroVentas = 0;
    }


    public void vender(int dineroVenta) { this.dineroVentas += dineroVenta; }


    public int getSueldo() {
        return sueldoBase + (int) (dineroVentas * porcentajeComision / 100.0);
    }
}
```

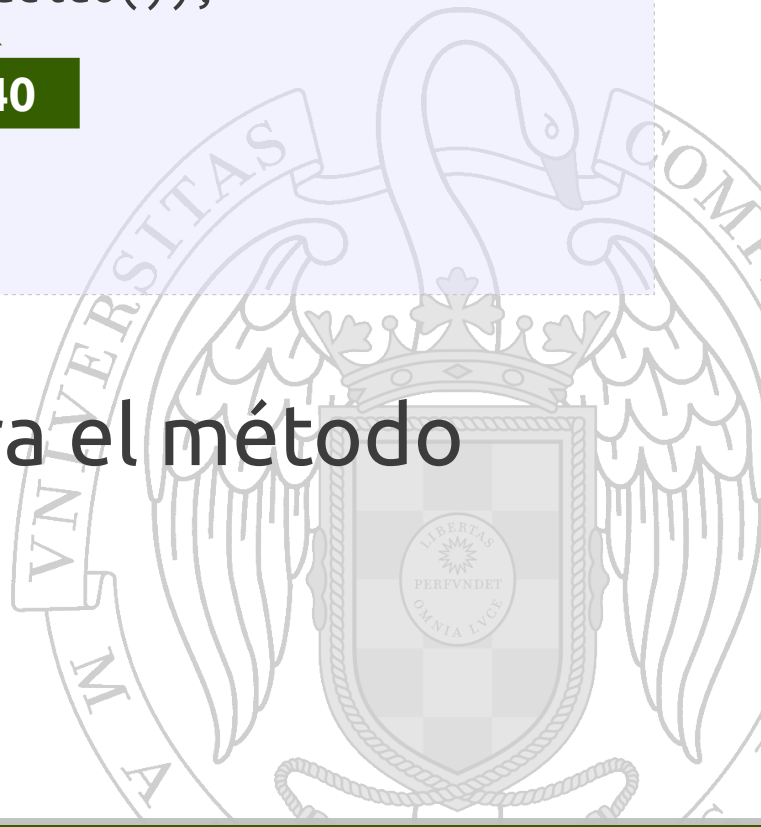
Acceso a atributo protected

Ejemplo: EmpleadoComision

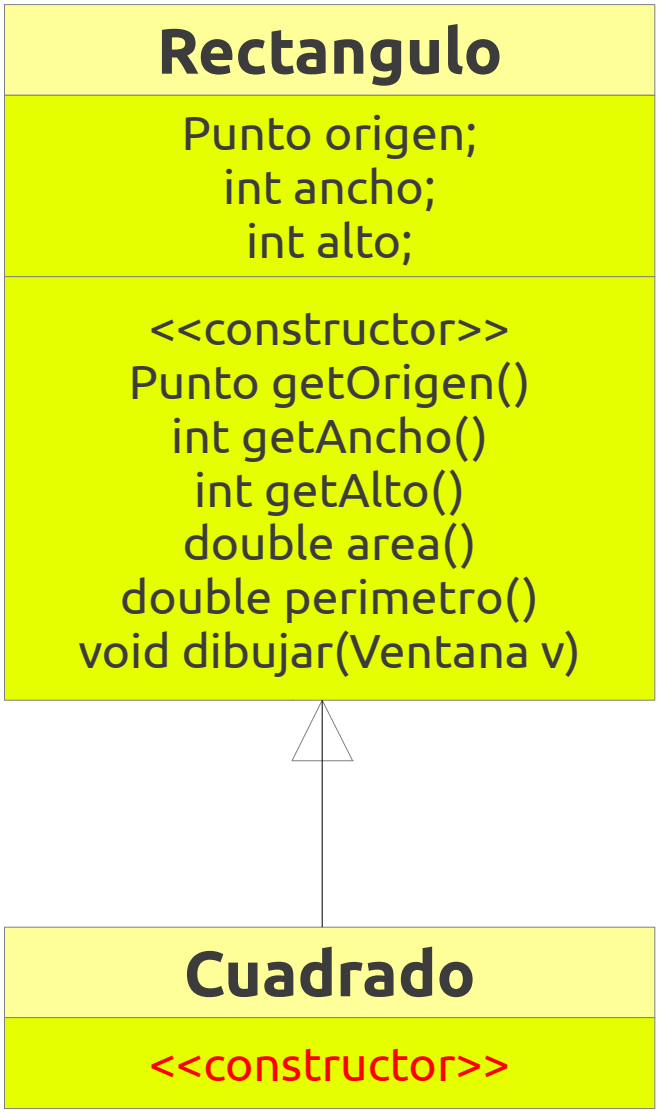
```
public class TestEmpleadoComision {
    public static void main(String[] args) {
        EmpleadoComision ec = new EmpleadoComision("Fuckencio Martinez",
                                                   new Fecha(15, 3, 1979),
                                                   123456, 1000, 20);
        ec.vender(200);
        System.out.print("Sueldo: %d\n", ec.getSueldo());

        ec.imprimirDatos();
    }
}
```

1040

- Discusión: ¿qué sueldo muestra el método `imprimirDatos`?

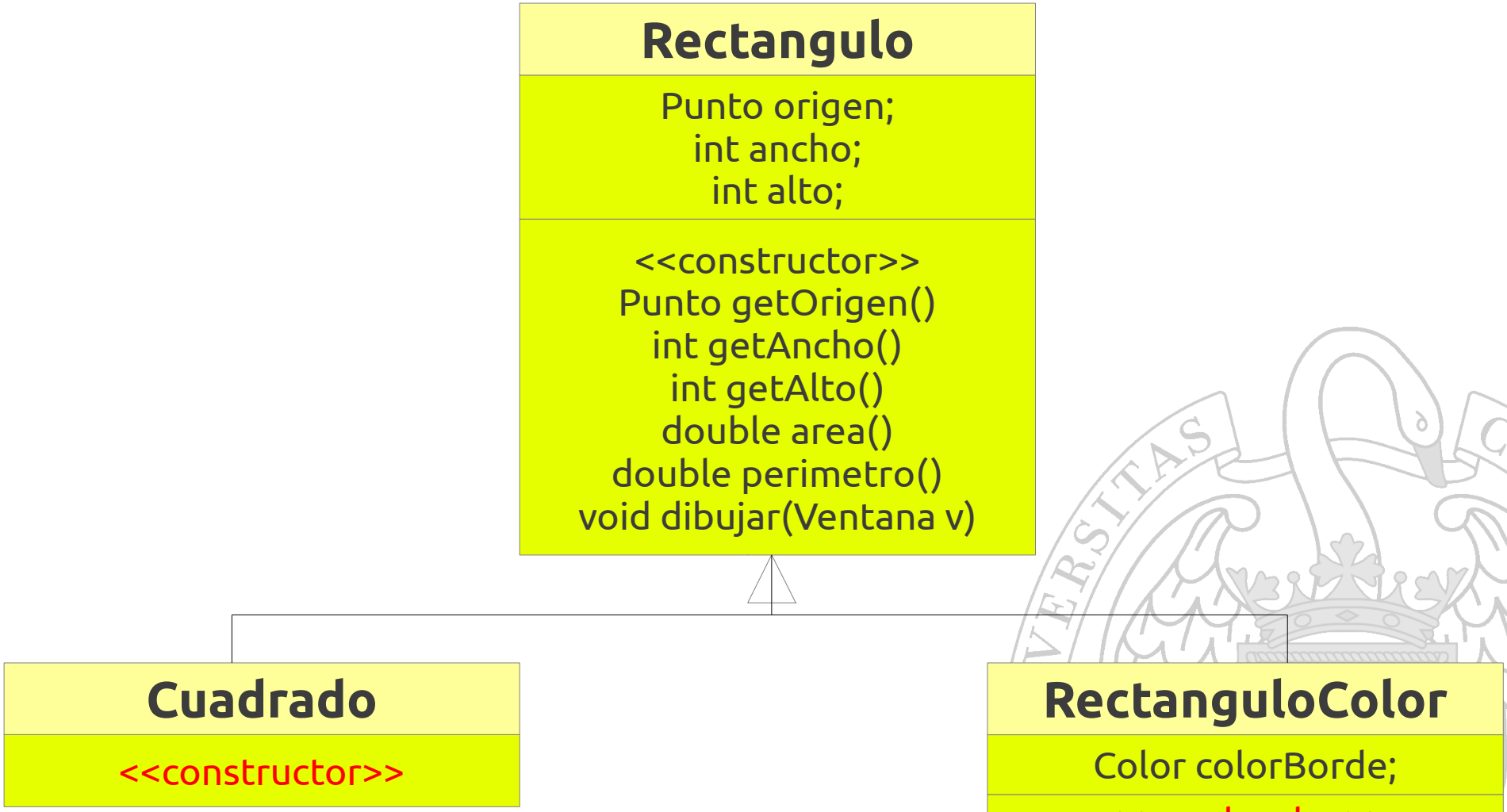


Figuras geométricas

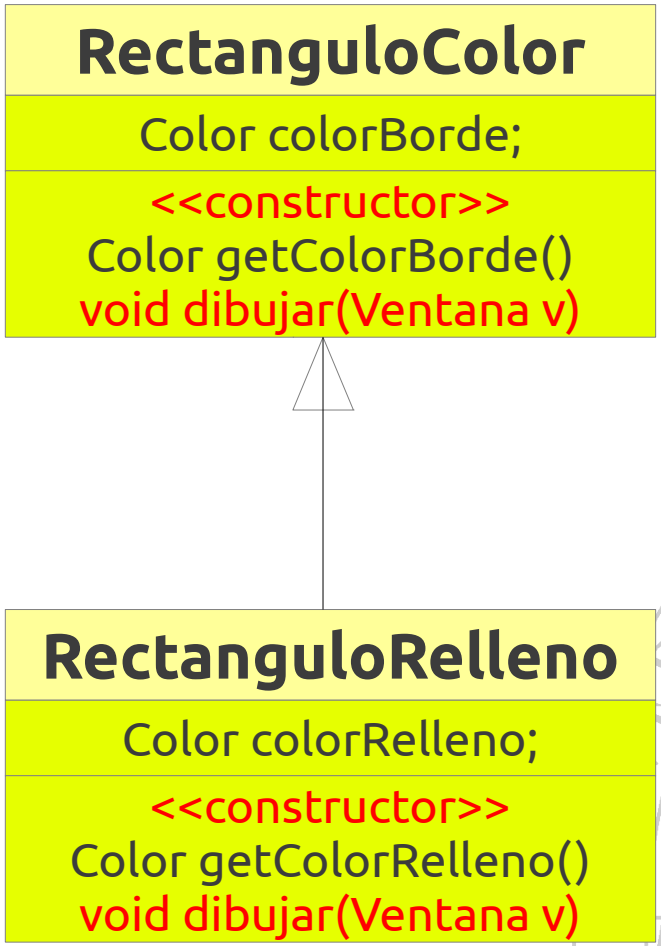



```
public class Cuadrado {
    public Cuadrado(Punto posicion, int ancho) {
        super(posicion, ancho, ancho);
    }
}
```

Figuras geométricas



Figuras geométricas

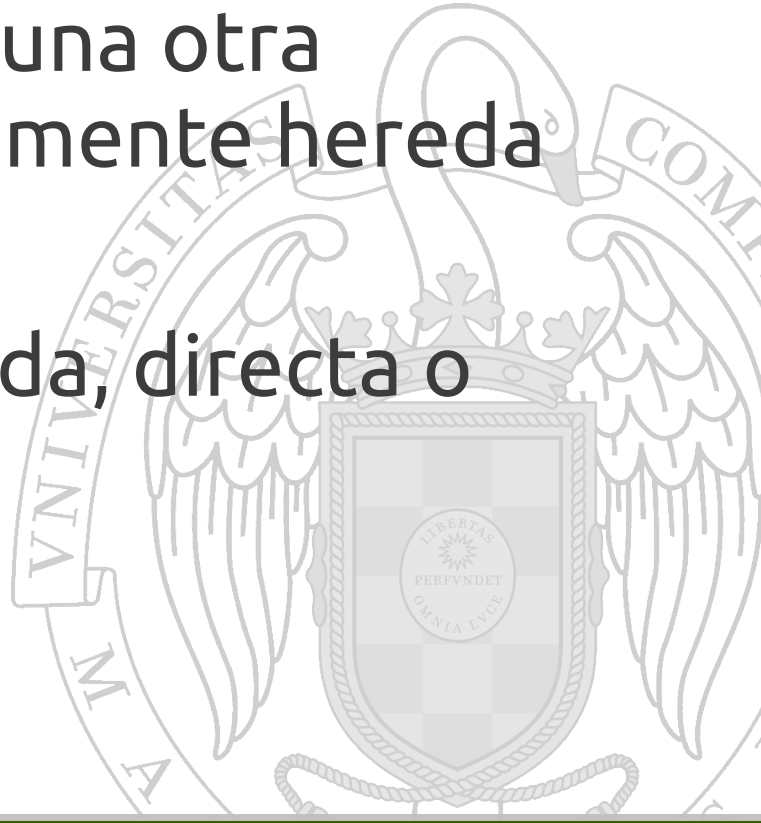


Contenidos

- Extensión mediante herencia.
- Reescritura de métodos.
- Modificador de acceso `protected`.
- Jerarquía de clases.
- La clase `Object`.

La clase `Object`

- Definida en el paquete `java.lang`
  - Siempre se importa por defecto.
- Es la raíz de toda jerarquía de clases.
- Si una clase no hereda de ninguna otra mediante `extends`, automáticamente hereda de `Object`.
- Consecuencia: toda clase hereda, directa o indirectamente, de `Object`.



La clase `Object`

- No tiene ningún atributo público ni protegido.
- Define 11 métodos, que pueden ser sobreescritos en las subclases.

`http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html`

- Nosotros veremos:
  - `toString()`
  - `finalize()`
  - `equals()`

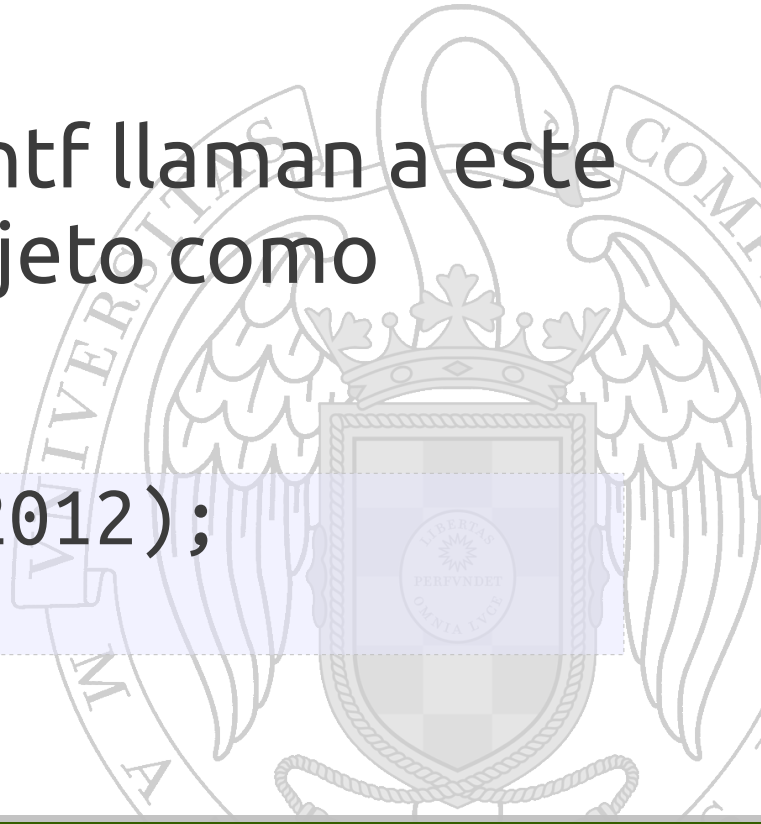
Método toString

- Obtiene una representación del objeto en forma de cadena de caracteres (`String`)

```
public String toString()
```

- Los métodos print/println/printf llaman a este método cuando reciben un objeto como parámetro.

```
Fecha f = new Fecha(14, 5, 2012);
System.out.println(f)
```

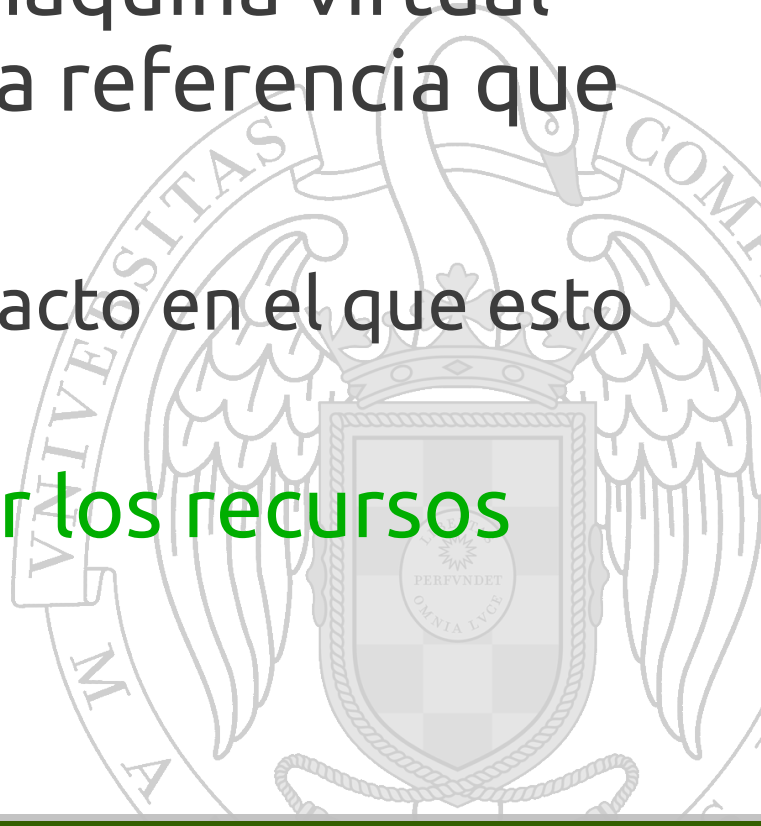


Método toString

```
// Fecha.java
    public String toString() {
        String cadenaMes;
        switch(mes) {
            case 1: cadenaMes = "Enero"; break;
            case 2: cadenaMes = "Febrero"; break;
            case 3: cadenaMes = "Marzo"; break;
            case 4: cadenaMes = "Abril"; break;
            case 5: cadenaMes = "Mayo"; break;
            case 6: cadenaMes = "Junio"; break;
            case 7: cadenaMes = "Julio"; break;
            case 8: cadenaMes = "Agosto"; break;
            case 9: cadenaMes = "Septiembre"; break;
            case 10: cadenaMes = "Octubre"; break;
            case 11: cadenaMes = "Noviembre"; break;
            case 12: cadenaMes = "Diciembre"; break;
            default: cadenaMes = "<<mes desconocido>>"; break;

        }
        return String.valueOf(dia) + " de " + cadenaMes
                                   + " de " + String.valueOf(año);
    }
```

Método `finalize`

- Se ejecuta cuando el objeto va a ser eliminado de la memoria.
- Un objeto es eliminado automáticamente por el recolector de basura de la máquina virtual de Java cuando no hay ninguna referencia que apunte hacia él.
  - No se especifica el momento exacto en el que esto sucede.
- `finalize` se utiliza para liberar los recursos asociados al objeto.

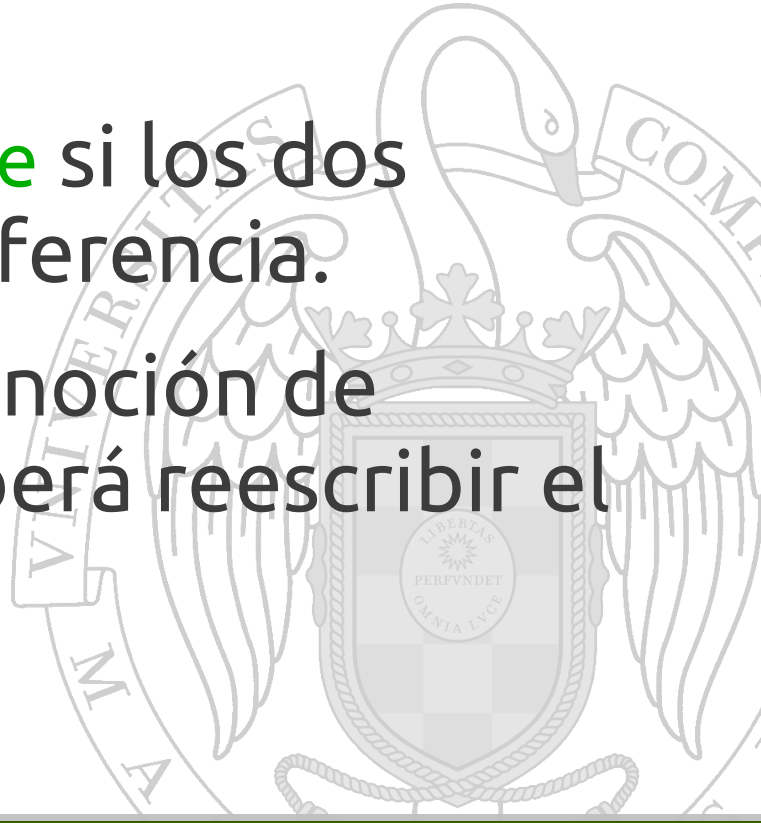


Método equals

- Se utiliza para comprobar si dos objetos son iguales.

```
public boolean equals(Object o)
```

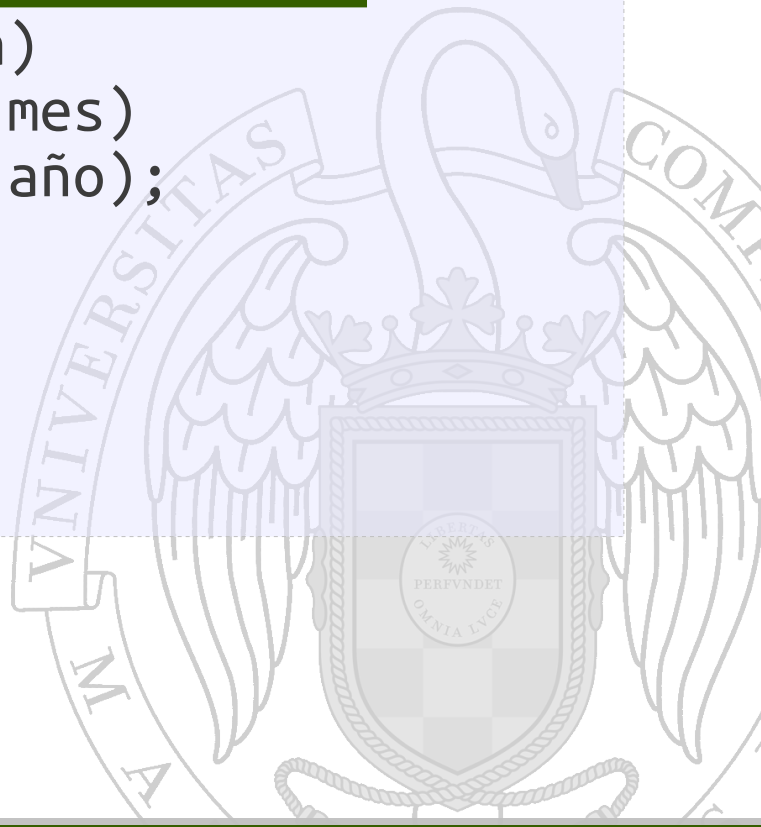
- Por defecto sólo devuelve `true` si los dos objetos apuntan a la misma referencia.
- Si se quiere implementar otra noción de igualdad entre objetos, se deberá reescribir el método.



Método equals

```
// Fecha.java
    public boolean equals(Object o) {
        if (o instanceof Fecha) {
            Fecha f = (Fecha) o;
            return (this.dia == f.dia)
                    && (this.mes == f.mes)
                    && (this.año == f.año);
        } else {
            return false;
        }
    }
```

Conversión



Referencias

- P. Deitel, H. Deitel
  Java. How to Program (9th Edition)
  Cap. 9.
- B. Eckel
  Thinking in Java (3rd Edition)
  Caps. 6.