Polimorfismo

Java y Servicios Web I
Master en Ingeniería Matemática

Manuel Montenegro
Dpto. Sistemas Informáticos y Computación

Desp. 467 (Mat)

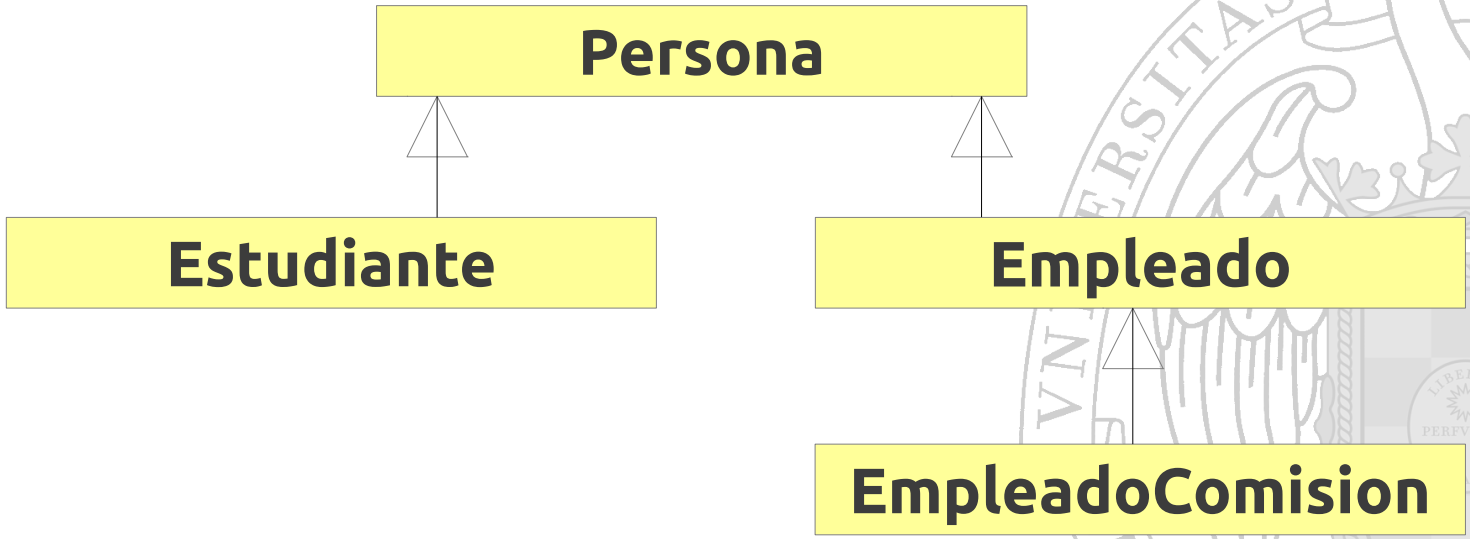
montenegro@fdi.ucm.es

Contenidos

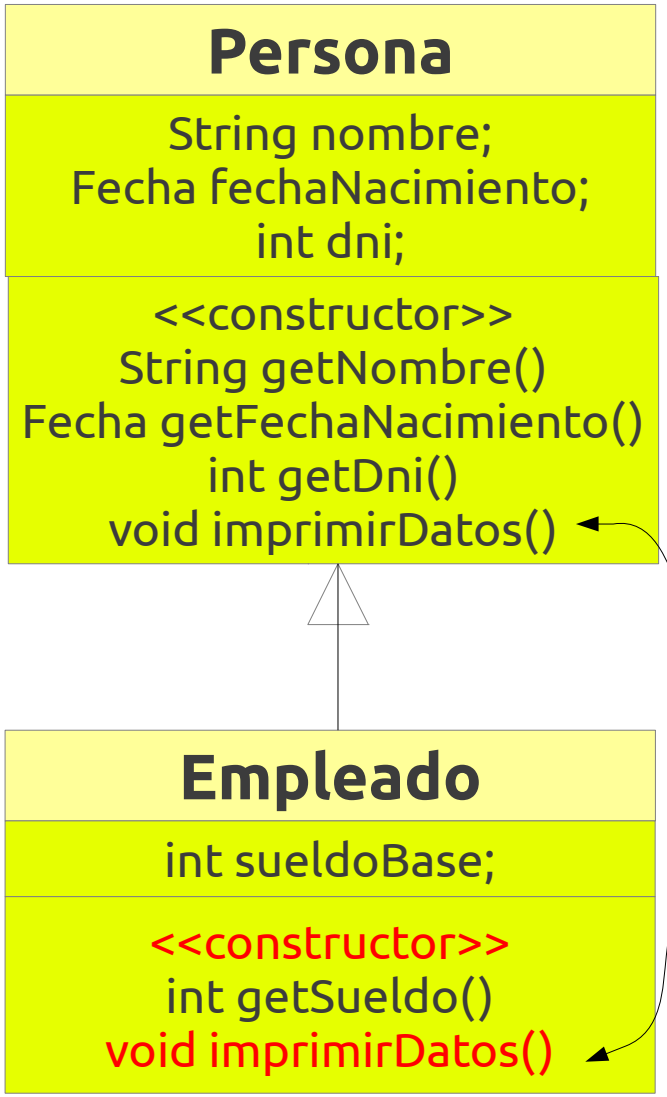
- Introducción.
- Conversión entre tipos.
- Vinculación dinámica.
- Polimorfismo mediante ejemplos.
- Clases abstractas.
- Interfaces.

Polimorfismo

- Capacidad de enviar un mismo mensaje a distintos objetos de naturaleza heterogénea.
- Íntimamente relacionado con el concepto de herencia.



Polimorfismo

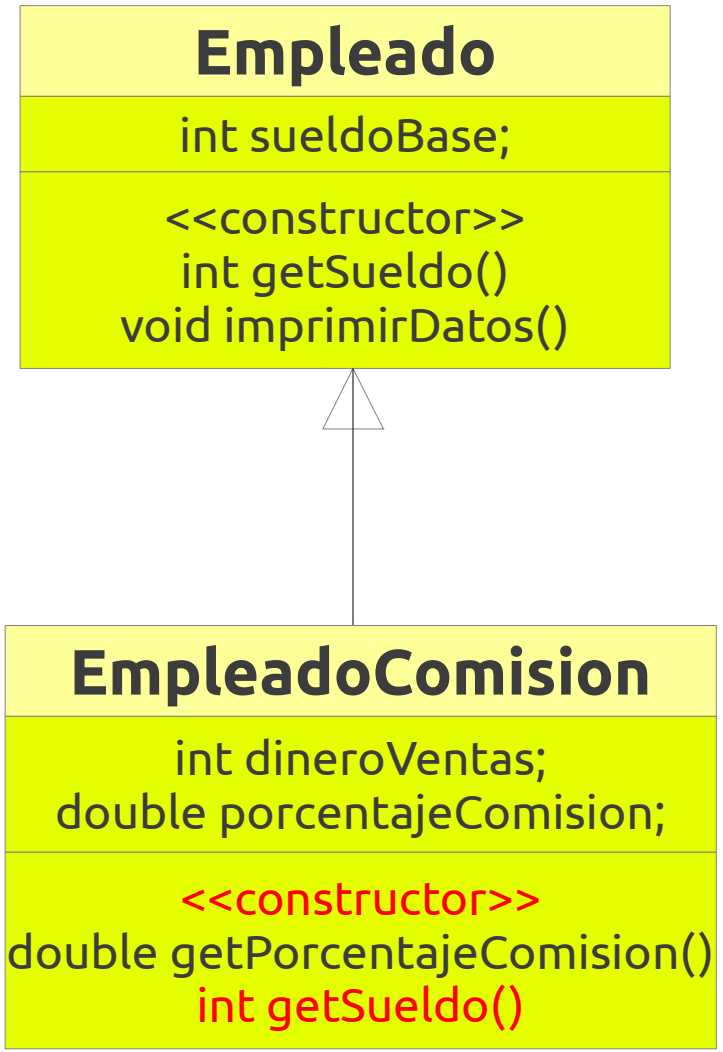


- El método `imprimirDatos()` tiene una funcionalidad distinta según el objeto que reciba la mensaje.

```
Persona p = new Persona(...);
Empleado e = new Empleado(...);


p.imprimirDatos();
e.imprimirDatos();
```

Polimorfismo

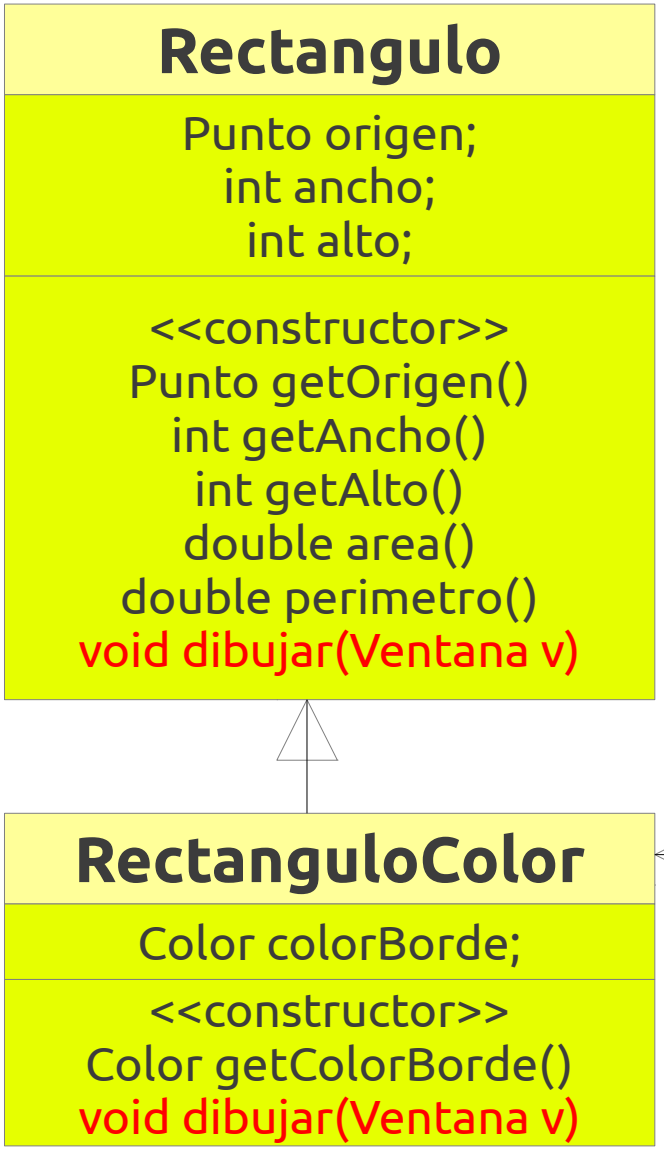


- El método `getSueldo()` calcula de forma distinta el sueldo del empleado, según si trabaja o no a comisión.

```
Empleado e = new Empleado(...);
EmpleadoComision ec = new EmpleadoComision(...);

int sueldo1 = e.getSueldo();
int sueldo2 = ec.getSueldo();
```

Polimorfismo



- El método `dibujar(Ventana v)` llama a funciones distintas de `Ventana` según la clase de figura geométrica a la que se aplique.

Contenidos

- Introducción.
- Conversión entre tipos.
- Vinculación dinámica.
- Polimorfismo mediante ejemplos.
- Clases abstractas.
- Interfaces.

Conversión de tipos

- Hemos visto los elementos de un tipo pueden convertirse a un tipo más *amplio* o *general*.
  - La conversión es segura.

```
int entero = 4;
double real;
real = entero;
```

- La conversión en sentido contrario también es posible, pero ha de indicarse explícitamente.
  - Puede haber pérdida de información.

```
int entero;
double real = 5.34;
entero = (int) real;
```

Conversión hacia arriba (*upcasting*)

- Similarmente, podemos asignar un objeto de una clase B a una variable de tipo A, si A es superclase (directa o indirecta) de B.

```
Persona p;
Empleado e = new Empleado(...);
p = e;
```

Conversión hacia arriba (*upcasting*)

- Similarmente, podemos asignar un objeto de una clase B a una variable de tipo A, si A es superclase (directa o indirecta) de B.

```
Persona p = new Empleado(...);
```

p es de tipo `Persona`, y sólo podrá acceder a los métodos de la clase `Persona`

- Esta conversión es segura, ya que la herencia modela la relación **es-un**.
  - Un `Empleado` es una `Persona`.

Conversión hacia abajo (*downcasting*)

- Es posible asignar una variable de un tipo A a otra de un tipo B, si B es subclase de A.
- Pero esta conversión no siempre es correcta, y ha de indicarse explícitamente.

```
Persona p = new Empleado(...);
Empleado e = (Empleado) p;
```

Correcto

```
Persona p = new Estudiante(...);
Empleado e = (Empleado) p;
```

Error

```
Persona p = new Persona(...);
Empleado e = (Empleado) p;
```

Error

Conversión hacia abajo (*downcasting*)

- Es posible asignar una variable de un tipo A a otra de un tipo B, si B es subclase de A.
- Pero esta conversión no siempre es correcta, y ha de indicarse explícitamente.

```
Persona p = new EmpleadoComision(...);
Empleado e = (Empleado) p;
```

Correcto

Contenidos

- Introducción.
- Conversión entre tipos.
- Vinculación dinámica.
- Polimorfismo mediante ejemplos.
- Clases abstractas.
- Interfaces.

Vinculación dinámica

- El compilador determina los mensajes que puede recibir un objeto a partir de su tipo.

```
Persona p = new Empleado(...);
```

p.getNombre();
p.getFechaNacimiento()
p.getDni();
p.setNombre(...);
p.imprimirDatos();

~~p.getSueldo()~~
~~p.setSueldo(...)~~

```
p.imprimirDatos();
```

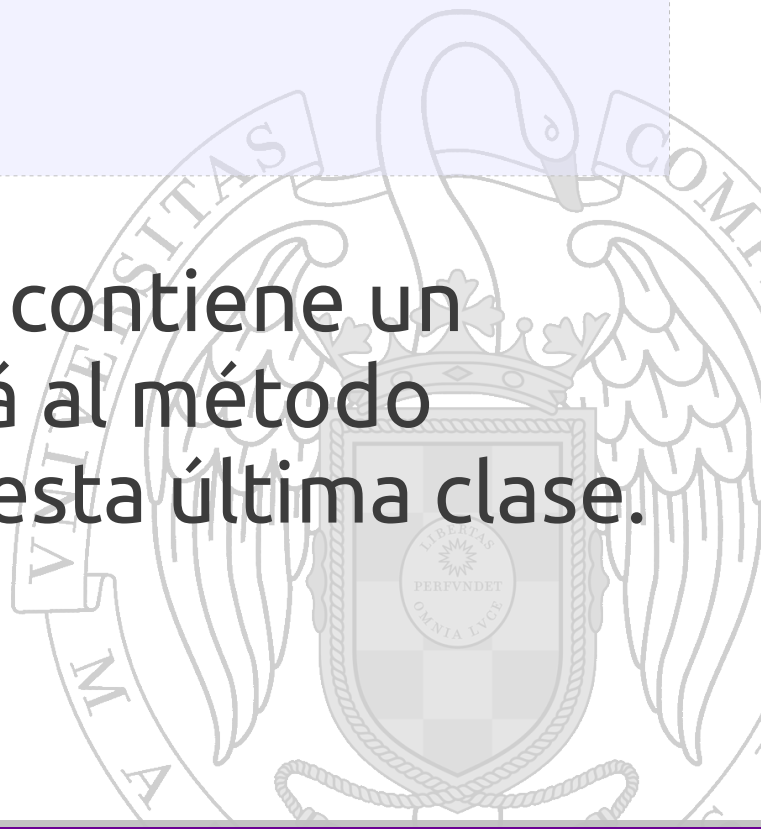
¿El definido en Persona?
¿O el definido en Empleado?

Vinculación dinámica

- La decisión del método específico a ejecutar se realiza en tiempo de ejecución, y en base a la clase *real* del objeto, no a su tipo.

```
p.imprimirDatos();
```

- Aunque `p` sea de tipo `Persona`, contiene un `Empleado`. Por tanto, se llamará al método `imprimirDatos()` reescrito en esta última clase.



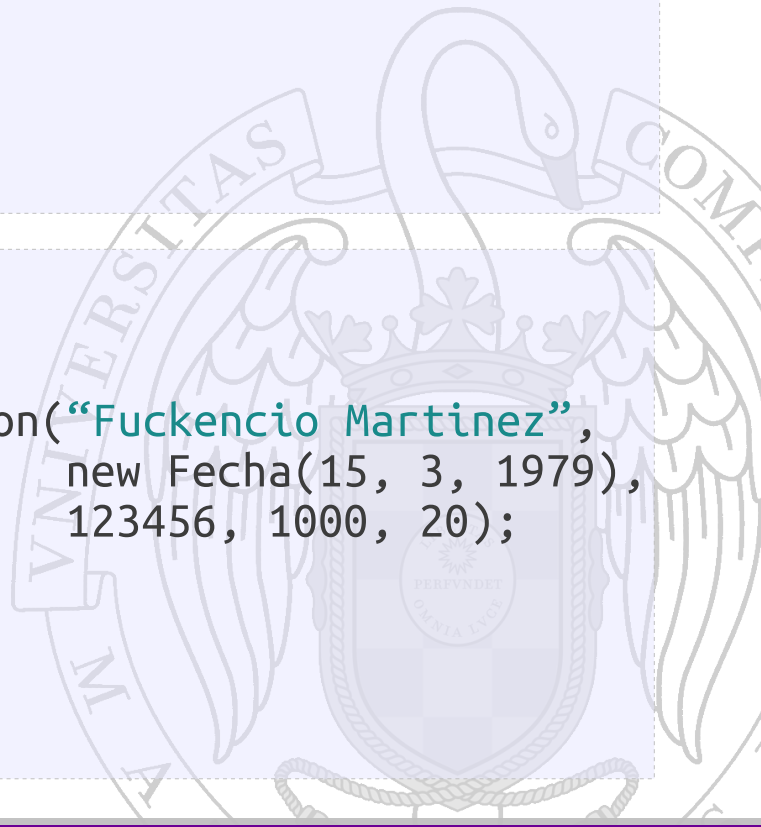
Ejemplo: imprimirDatos

```
public class Empleado extends Persona {

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

```
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);
       ec.imprimirDatos();
    }
}
```



Ejemplo: Rectangulo

```
Ventana v = new Ventana();
v.abrir();
Rectangulo r = new RectanguloRelleno(new Punto(10,10), 100, 100,
                                     Color.AZUL, Color.NARANJA);
r.dibujar(v);
Color c = r.getColorRelleno();
Color c = ((RectanguloRelleno) r).getColorRelleno();
```

Contenidos

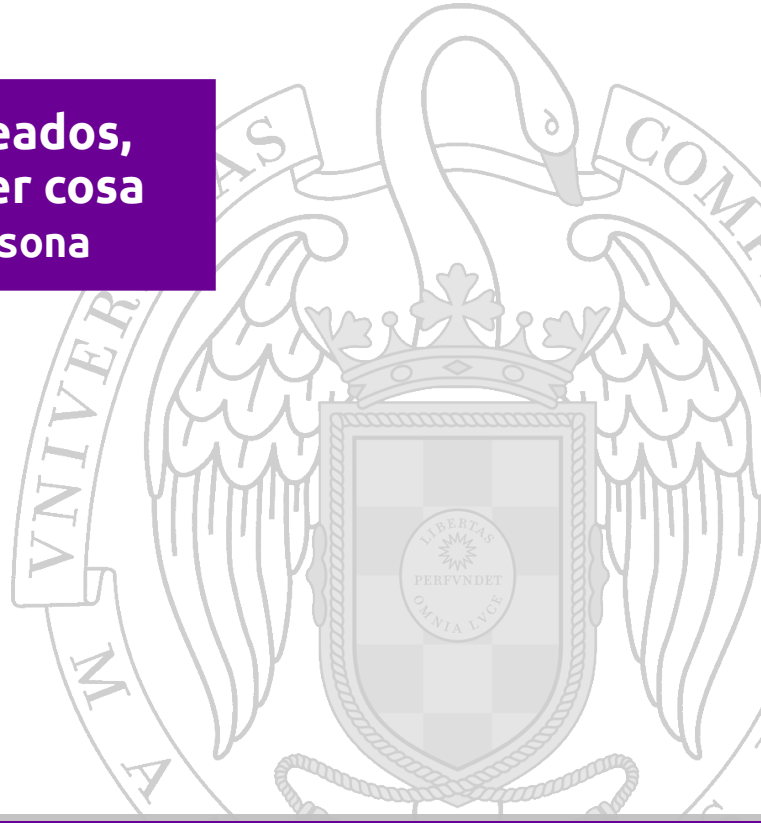
- Introducción.
- Conversión entre tipos.
- Vinculación dinámica.
- Polimorfismo mediante ejemplos.
- Clases abstractas.
- Interfaces.

Motivación de polimorfismo

- El polimorfismo permite tratar con conjuntos de elementos de manera genérica.

```
Persona[] personas = new Persona[10];
```

Puede contener empleados, estudiantes, o cualquier cosa que se derive de `Persona`

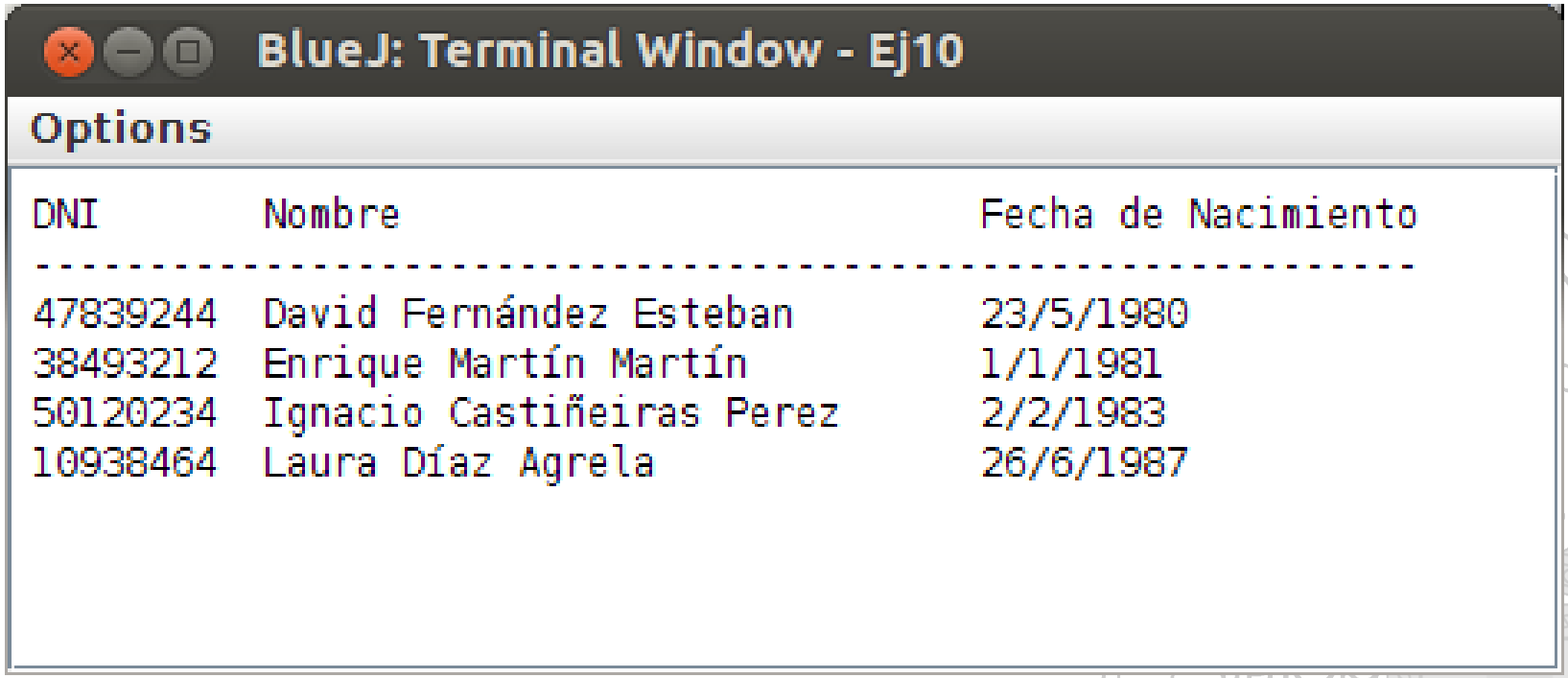


Ejemplo: Listado de personas

```
public class TestEmpleado
{
    public static void imprimirListado(Persona[] personas) {
        System.out.println("DNI       Nombre                          Fecha de Nacimiento");
        System.out.println("-----------------------------------------------------------------");
        for (Persona p : personas) {
            System.out.printf("%-9d %-30s %s\n", p.getDni(),
                                                p.getNombre(), p.getFechaNacimiento());
        }
    }

    public static void main(String[] args) {
        Persona[] personas = new Persona[4];
        personas[0] = new Persona("David Fernández Esteban",
                                  new Fecha(23, 5, 1980), 47839244);
        personas[1] = new Empleado("Enrique Martín Martín",
                                  new Fecha(1, 1, 1981), 38493212, 1600);
        personas[2] = new Estudiante("Ignacio Castiñeiras Perez",
                                  new Fecha(2, 2, 1983), 50120234);
        personas[3] = new EmpleadoComision("Laura Díaz Agrela",
                                  new Fecha(26, 6, 1987), 10938464, 1000, 20);
        imprimirListado(personas);
    }
}
```

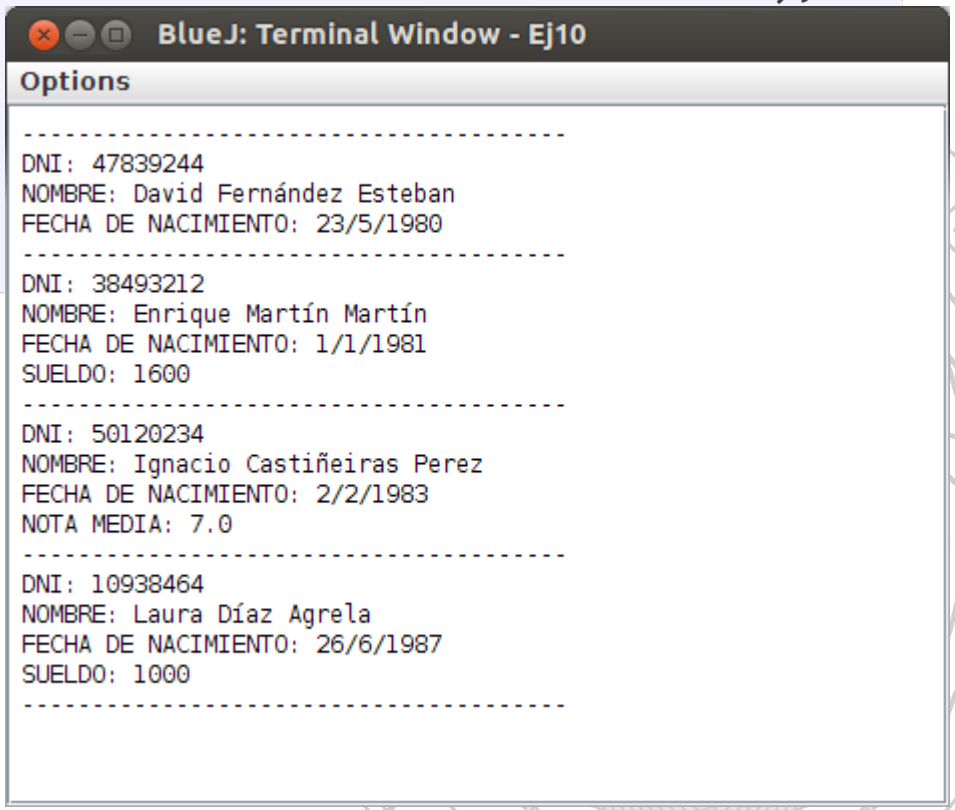
Ejemplo: Listado de personas



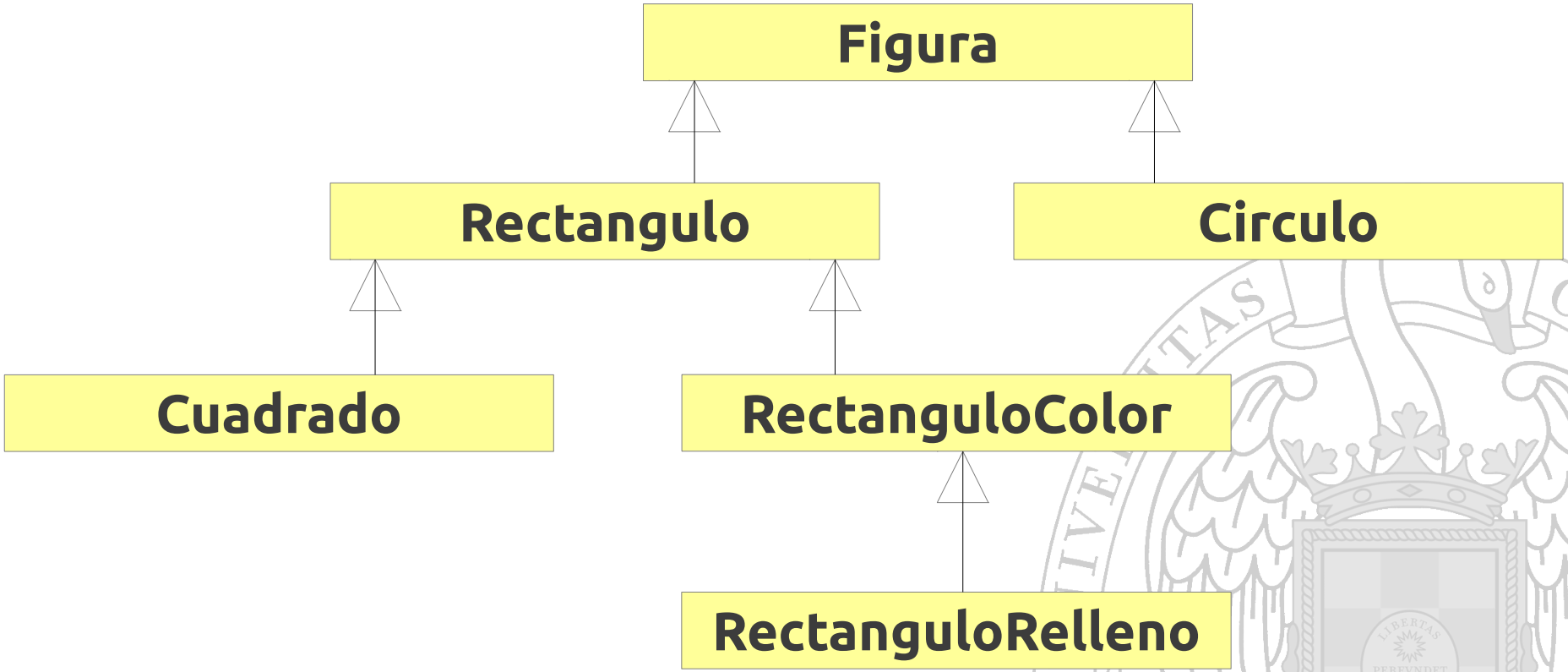
Ejemplo: Listado de personas

```
public class TestEmpleado
{
    public static void imprimirListadoExtendido(Persona[] personas) {
        System.out.println("----------------------------------------");
        for (Persona p : personas) {
            p.imprimirDatos();
            System.out.println("----------------------------------------");
        }
    }

    …
}
```

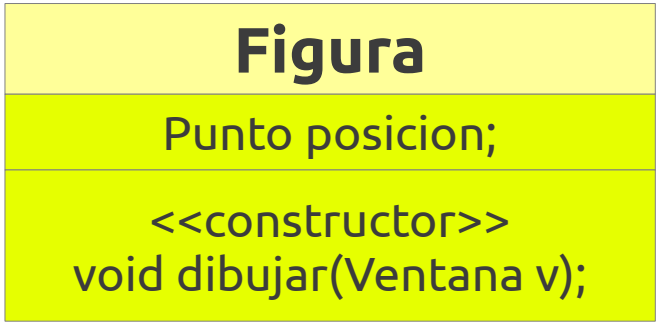


Ejemplo: Figuras geométricas



Ejemplo: Figuras geométricas

- Podemos abstraer las propiedades y métodos comunes en `Rectangulo` y `Circulo`.



- El método dibujar() será sobreescrito por las subclases `Rectangulo` y `Circulo`, que llamarán a las funciones correspondientes de la clase `Ventana`.

Ejemplo: Figuras geométricas

```
public class Escenas
{
    public static void dibujarEscena(Figura[] figuras, Ventana v) {
        for (Figura f : figuras) {
            f.dibujar(v);
        }
    }


    public static void main(String[] args) {
        Ventana v = new Ventana();
        Figura[] circulos = new Figura[10];
        for (int i = 1; i <= 10; i++) {
            circulos[i-1] = new Circulo(new Punto(150, 150), i*10);
        }
        v.abrir();
        dibujarEscena(circulos, v);
    }
}
```

Ejemplo: Figuras geométricas

Contenidos

- Introducción.
- Conversión entre tipos.
- Vinculación dinámica.
- Polimorfismo mediante ejemplos.
- Clases abstractas.
- Interfaces.

Clases abstractas

- Comenzamos a implementar `Figura`

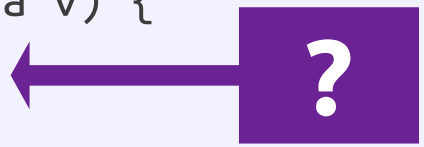
```
public class Figura {
    protected Punto posicion;

    public Figura(Punto posicion) {
        this.posicion = posicion;
    }

    public Punto getFigura() {
        return posicion;
    }

    public void dibujar(Ventana v) {

    }
}
```



Clases abstractas

- ¿Qué se debería pintar en la clase `Figura`?
  - Estamos a un nivel de abstracción demasiado elevado como para dibujar algo concreto en la ventana.
  - ...pero la capacidad de dibujar es algo común a todas las figuras.
- ¿Tiene sentido crear instancias de `Figura`?

```
Figura f = new Figura(new Punto(100,100));
```

Clases abstractas

- `Figura` es una clase abstracta:
  - Sólo tiene sentido como abstracción de lo que es común a varios tipos de figuras.

```
abstract public class Figura {
    protected Punto posicion;

    public Figura(Punto posicion) {
        this.posicion = posicion;
    }

    public Punto getPosicion() { return posicion; }
    abstract public void dibujar(Ventana v);
}
```

Clase abstracta

Método abstracto

Clases abstractas

- Un método abstracto no se implementa.
  - Especifica un método que las subclases de Figura han de reescribir.
- Si una clase tiene un método abstracto, la clase es abstracta.
  - Tiene funcionalidad especificada, pero sin implementar.
- Las subclases de una clase abstracta han de reescribir sus métodos abstractos.
  - De lo contrario, las subclases también deberán declararse como abstractas.
- Las clases abstractas no se pueden instanciar.
  - Les falta funcionalidad!

```
Figura f = new Figura(new Punto(100,100));
f.dibujar(v);
```

Contenidos

- Introducción.
- Conversión entre tipos.
- Vinculación dinámica.
- Polimorfismo mediante ejemplos.
- Clases abstractas.
- Interfaces.

Interfaces

- Son clases abstractas sin atributos, cuyos métodos son todos abstractos.

```
public interface Dibujable {
  public void dibujar(Ventana v);
}
```

```
public interface Cuerpo {
  public void sumarCon(Cuerpo otro);
  public void multiplicarCon(Cuerpo otro);
}
```

```
public interface Comparable {
  public int compareTo(Object other);
}
```

Interfaces

- Una clase puede heredar una interfaz mediante la cláusula `implements`.

```
public class Complejo implements Cuerpo {
    private double real;
    private double imag;


    public void sumarCon(Cuerpo otro) {
        if (otro instanceof Complejo) {
            Complejo c = (Complejo) otro;
            this.real += c.real;
            this.imag += c.imag;
        }
    }
    public void multiplicarCon(Cuerpo otro) { ... }
    …
}
```

Interfaces

- Si una clase implementa una interfaz, ha de reescribir todos los métodos de la misma.

```
public class Fecha implements Comparable {
    …
    private int convertirANumero() {
       return año * 10000 + mes * 100 + dia;
    }
    public int compareTo(Object o) {
      if (o instanceof Fecha) {
        Fecha f = (Object) o;
        return this.convertirANumero() - f.convertirANumero();
      } else {
        // ERROR
      }
    }
}
```

Interfaces

- Si una clase implementa una interfaz, ha de reescribir todos los métodos de la misma.

```
public static Comparable minimo(Comparable obj1, Comparable obj2)
{
  if (obj1.compareTo(obj2) <= 0) {
    return obj1;
  } else {
    return obj2;
  }
}
```

Cualquier objeto que implemente la interfaz `Comparable`.

Interfaces

- Una clase puede implementar varias interfaces.

```
public class A implements Comparable, Dibujable {

    public void dibujar(Ventana v) {…}
    public int compareTo(Object o) {…}

}
```