Programación Orientada a Objetos

Departamento de Lenguajes y
Ciencias de la Computación

E.T.S.I. Telecomunicación
Universidad de Málaga

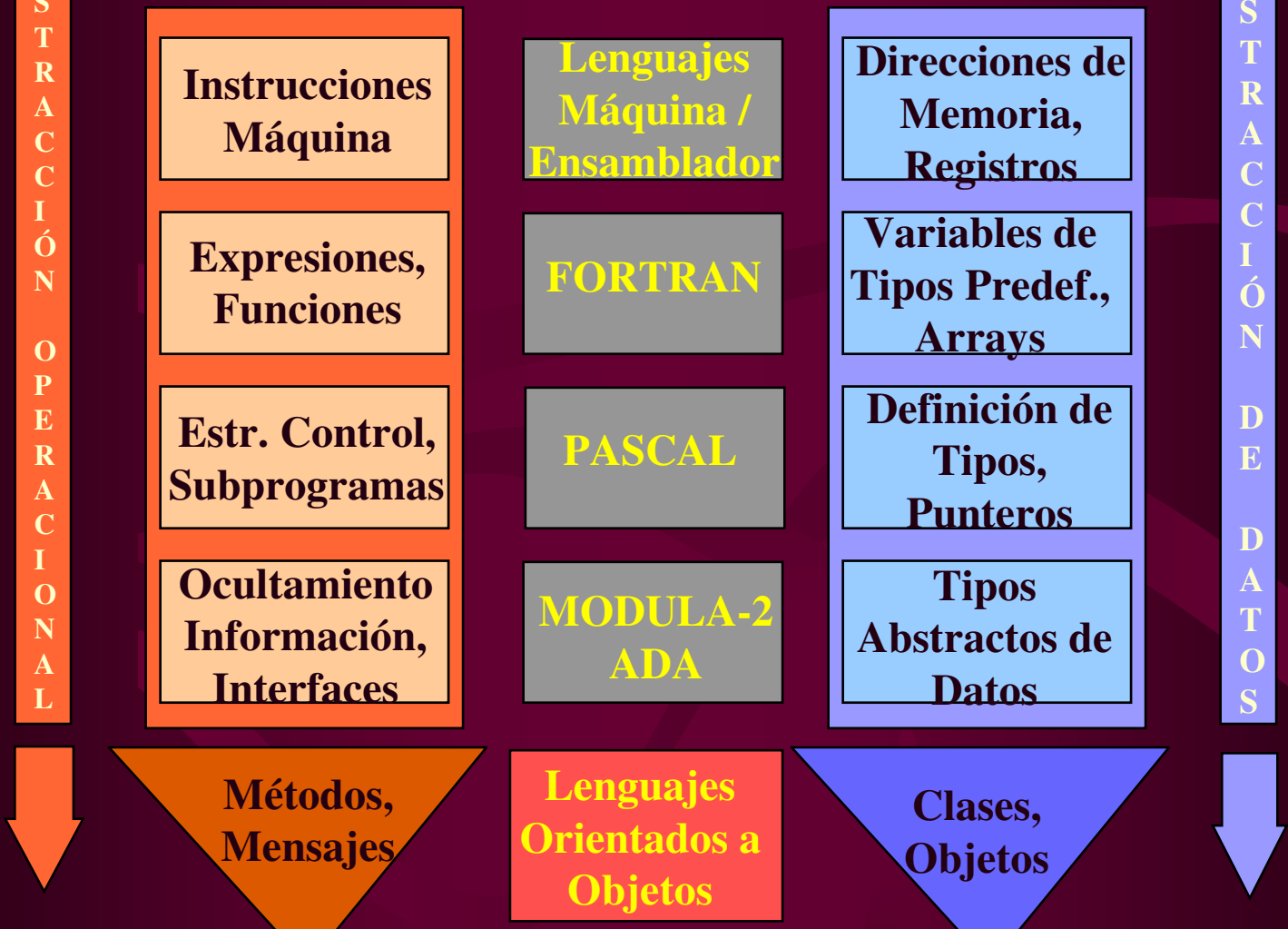
http://www.lcc.uma.es/

Contenido

- Introducción histórica
- Conceptos básicos de la Programación O. O.
- Conceptos avanzados de la Programación O.O.

Introducción Histórica

Evolución de los Lenguajes de Programación



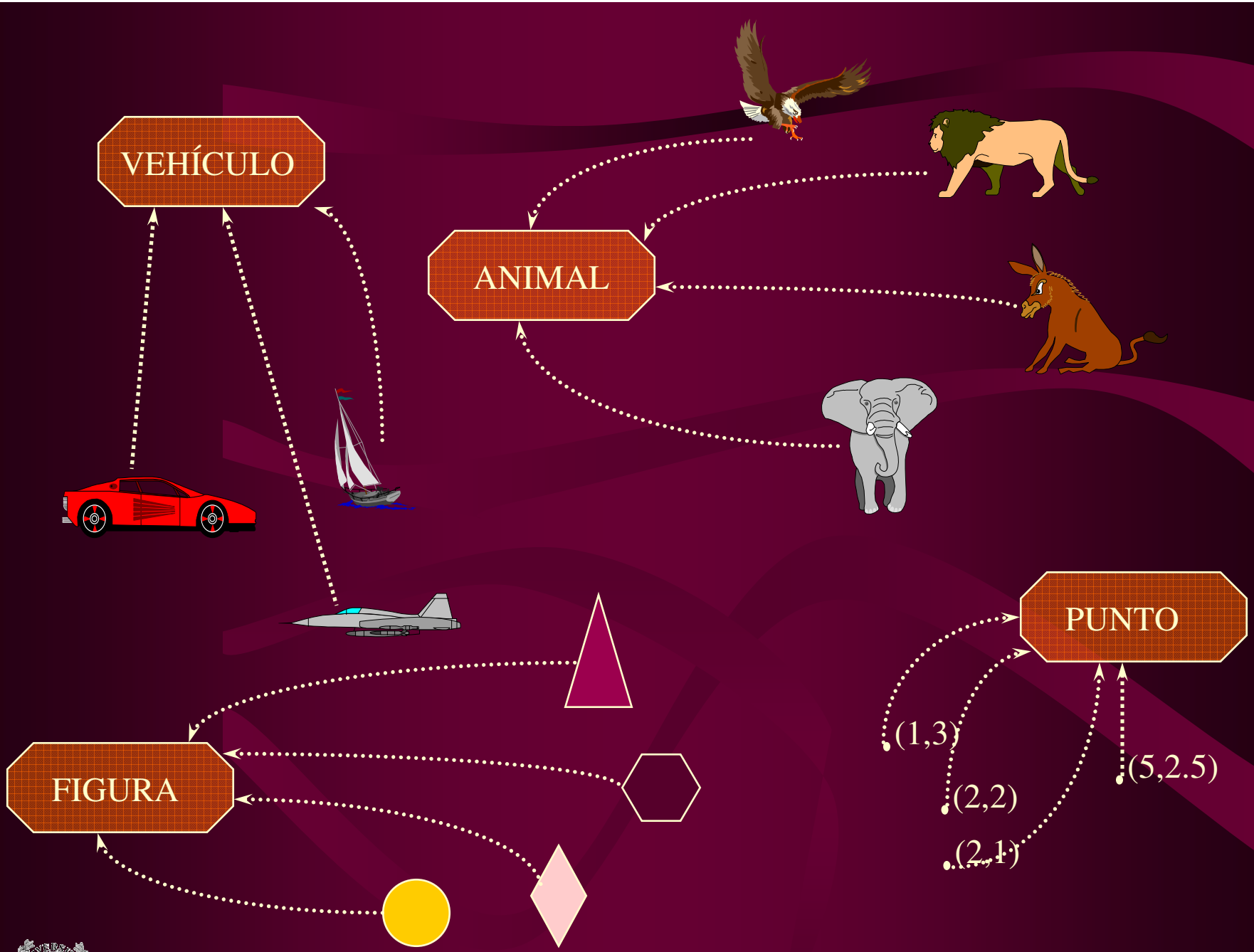
Evolución de los Lenguajes Orientados a Objetos

- Simula (Nygaard, 60s)
- Smalltalk (Xerox PARC, 70s)
- Eiffel (Meyer, 80s)
- C++ (Stroustrup, 80s)
- Java (Sun Microsystems, 90s)
- C# (Microsoft, 00s)

Conceptos Básicos de la Programación O.O.

Clases y Objetos

- CLASE = SUBPROGRAMAS + VARIABLES
  - ✓ Criterio de Modularización
  - ✓ Estado + Comportamiento
  - ✓ Entidad estática
  - ✓ Clase ≈ Tipo
- OBJETO = Instancia de una CLASE
  - ✓ Entidad dinámica
  - ✓ Cada objeto tiene su propio estado
  - ✓ Objetos de una misma clase comparten un comportamiento
  - ✓ Objeto ≈ Variable


VEHÍCULO
ANIMAL
PUNTO
FIGURA
(1,3)
(2,2)
(2,1)
(5,2.5)

¿Qué es una Clase?

Caja negra que oculta en su implementación:

- **Atributos**: variables que codifican el **estado** de una instancia de la clase (objeto)
- **Métodos**: subprogramas que describen el **comportamiento** de un objeto de la clase

Una clase es semejante a un tipo:

- Atributos: estructura de datos
- Métodos: operaciones sobre el tipo

¿Qué es un Objeto?

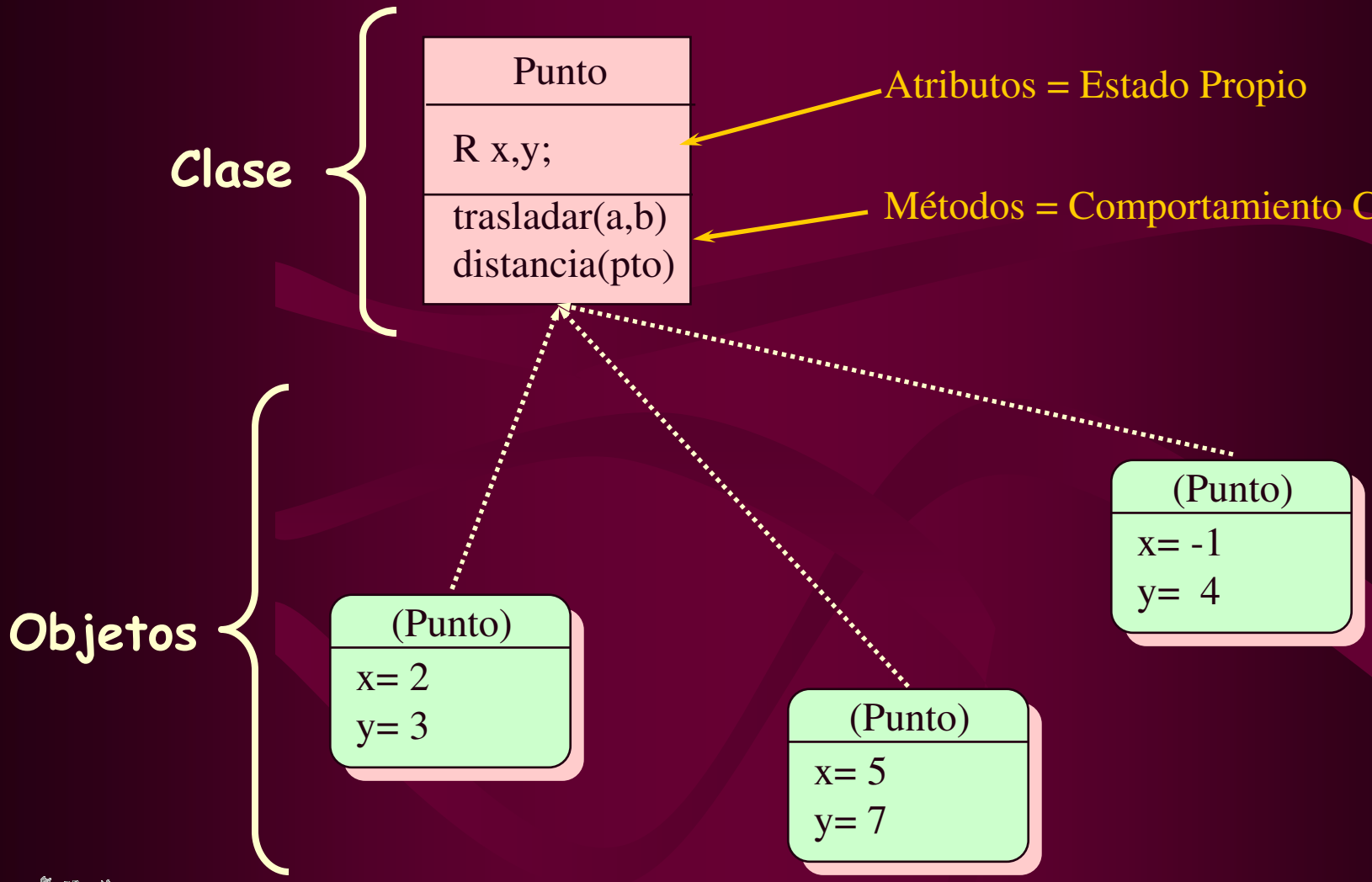
Instancia de una clase:

- Cada objeto de una clase tiene su propia copia de los atributos (**estado propio**)
- Todos los objetos de una clase comparten los mismos métodos (**comportamiento común**)

Implementador vs. Usuario

- Las clases son cajas negras con
  - Interfaz (uso)
  - Implementación (funcionamiento)
- El **implementador** se encarga de definir el interfaz y de desarrollar la implementación
- El **usuario** empleará los objetos de la clase exclusivamente a través del interfaz

Un ejemplo: la clase Punto



Definiendo la clase Punto

Comportamiento

```
INTERFAZ CLASE Punto
    MÉTODOS
        cambiar_x(E R nx);
        cambiar_y(E R ny);
        trasladar(E R dx,dy);
        R distancia(E Punto p);
FIN Punto;
```

Definición de Métodos

- El objeto afectado no aparece como argumento del método:

```
trasladar(E R dx,dy);
```

- En realidad, el objeto afectado es un argumento de **entrada/salida implícito** llamado **éste**:

```
ALGORITMO trasladar(ES Punto éste,E R dx,dy);
```

Implementando la clase Punto (I)

```
IMPLEMENTACIÓN CLASE Punto
  ATRIBUTOS
    R x,y;

  MÉTODOS
    cambiar_x(E R nx)
    INICIO
      x = nx;    // x == éste.x
    FIN

    cambiar_y(E R ny)
    INICIO
      y = ny;    // y == éste.y
    FIN
```

Estado

acceso al argumento implícito

Implementando la clase Punto (II)

```
trasladar(E R dx,dy)
INICIO
  x = x+dx;
  y = y+dy;
FIN

R distancia(E Punto p)
INICIO
 DEVOLVER sqrt(pow(x-p.x,2) +
              pow(y-p.y,2))
FIN

FIN Punto;
```

acceso total a otros objetos de la misma clase

Usando la clase Punto

- El **usuario** declara objetos como cualquier otra variable:

```
Punto p1, p2;
```

- Como **usuario**, no se puede acceder a la parte privada de los objetos:

```
p1.X = 2;
```

Error

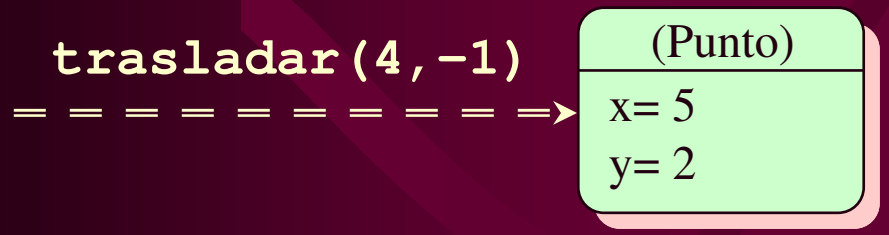
- El **usuario** sólo puede manipular un `Punto` invocando a los métodos del interfaz de la clase `Punto` (comportamiento)

Invocación de Métodos

- Los métodos se invocan mediante paso de **mensajes**:

```
P.trasladar(4,-1) = trasladar(P,4,-1)
```

- El objeto `P` es el receptor del mensaje:



Relación de Composición

- la composición permite expresar una relación de tipo "*está compuesto por*"
- Por ejemplo, un segmento *está compuesto por* dos puntos: origen y extremo

Punto
R x,y;
trasladar(a,b) distancia(pto)

Segmento
Punto Orig, Ext;
trasladar(a,b) longitud()

Implementando la Composición (I)

```
INTERFAZ CLASE Segmento
  MÉTODOS
       trasladar(E R dx,dy);
       R longitud();
FIN Segmento;
```

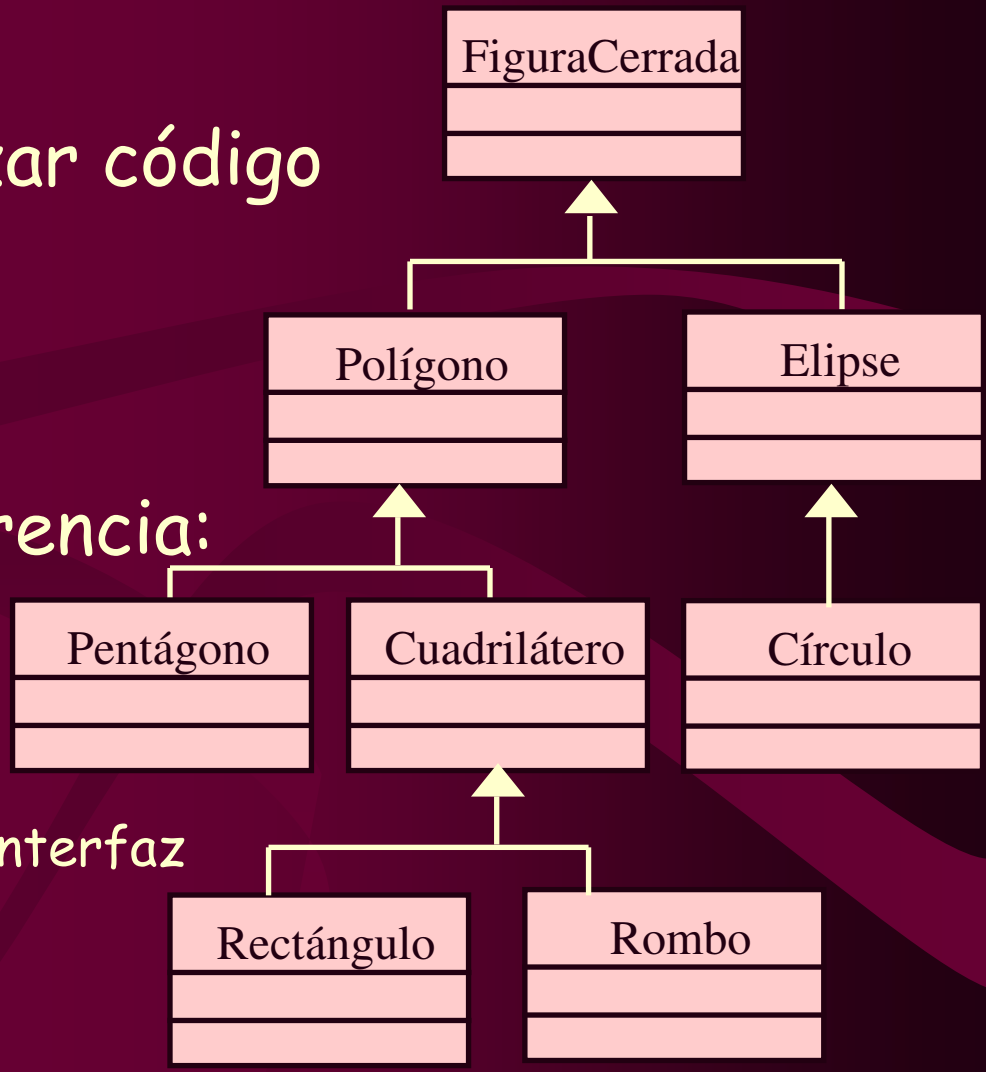
Implementando la Composición (II)

```
IMPLEMENTACIÓN CLASE Segmento
  ATRIBUTOS
    Punto Orig, Dest;
  MÉTODOS
    trasladar(E R dx,dy)
    INICIO
      Orig.trasladar(dx,dy);
      Dest.trasladar(dx,dy);
    FIN
    R longitud();
    INICIO
      DEVOLVER Orig.distancia(Dest);
    FIN
FIN Segmento;
```

Conceptos Avanzados de la Programación O.O.

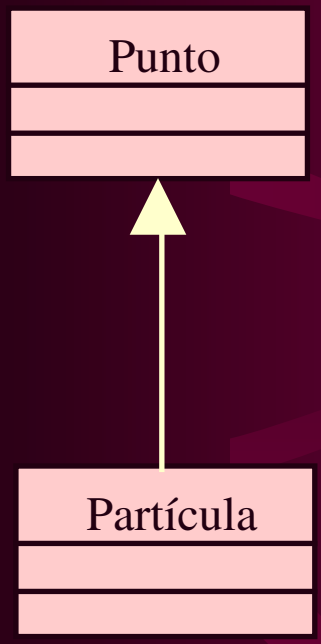
Herencia

- Posibilidad de reutilizar código
- Algo más que:
  - ✓ incluir ficheros, o
  - ✓ importar módulos
- Distintos tipos de herencia:
  - ✓ simple / múltiple
  - ✓ estricta
  - ✓ selectiva
  - ✓ de implementación/de interfaz



Herencia

Padres / Ascendientes

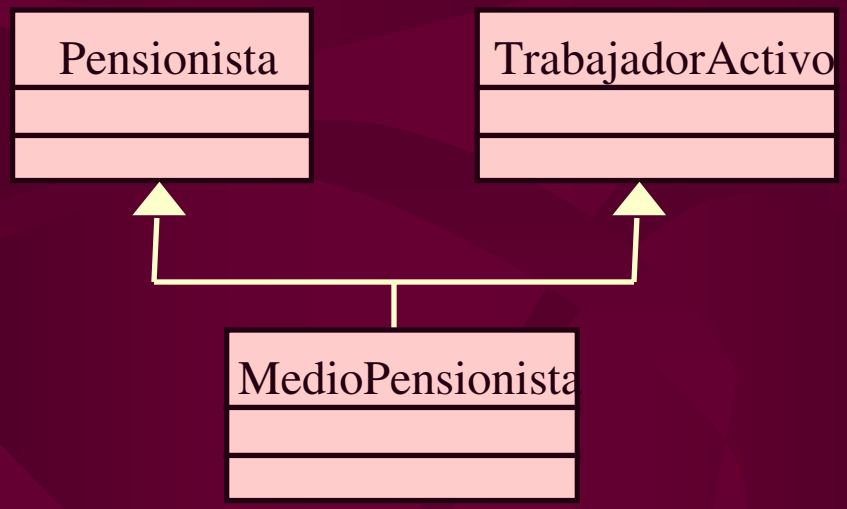


Hijos / Descendientes

- Una clase heredera proporciona los atributos y métodos de la clase heredada, y puede añadir otros nuevos.
- La clase heredera puede modificar el comportamiento heredado (por ejemplo, redefiniendo algún método heredado) .
- La herencia es transitiva.
- Los objetos de una clase que hereda de otra pueden verse como objetos de esta última.

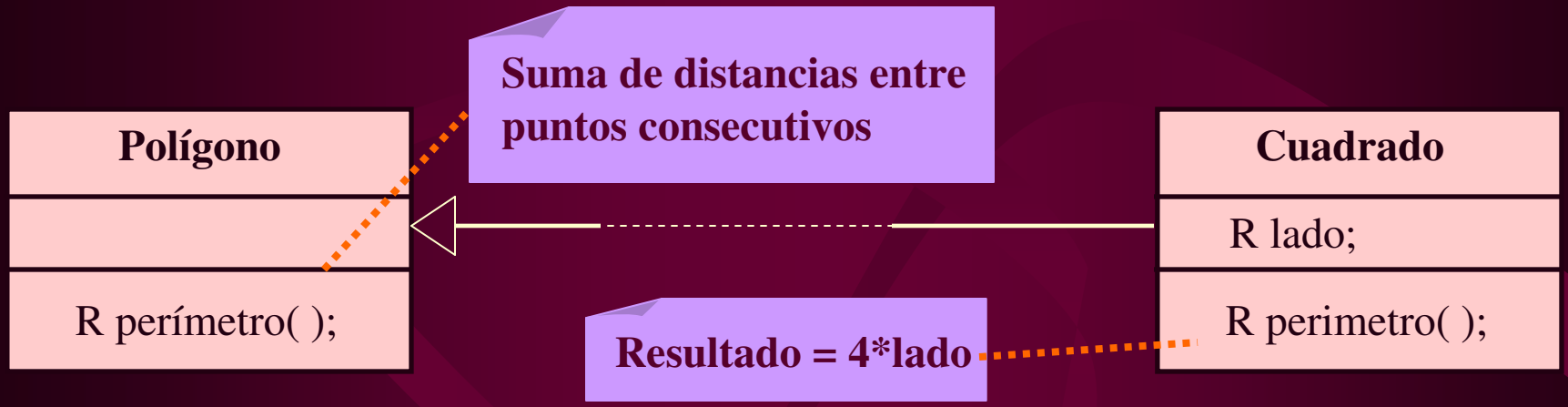
Herencia Múltiple

- Existen lenguajes con herencia múltiple, lo que permite que una clase reutilice la funcionalidad ofrecida por varias clases.



Herencia y Redefinición

- En la herencia las clases herederas pueden heredar un método o servicio, y luego redefinirlo, modificando su implementación.



Polimorfismo sobre los datos

- Una variable puede referirse a objetos de clases distintas de la que se ha declarado.
- La restricción dada por la herencia, permite construir estructuras con elementos de naturaleza distinta, pero con un comportamiento común: