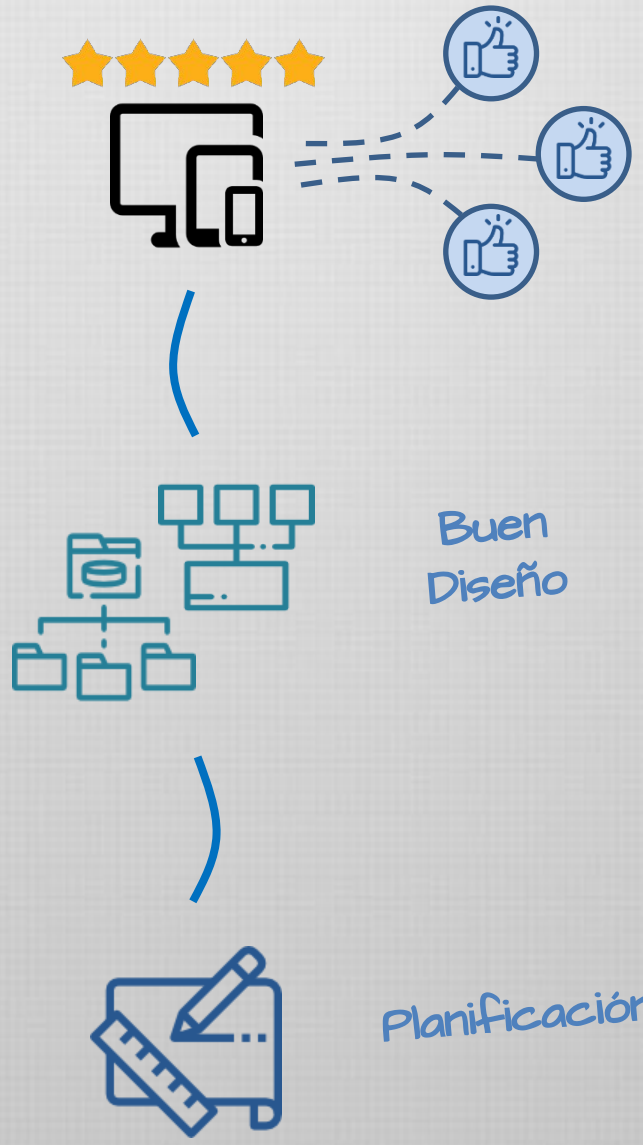
Tecnología de Programación

Martín L. Larrea

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Construir



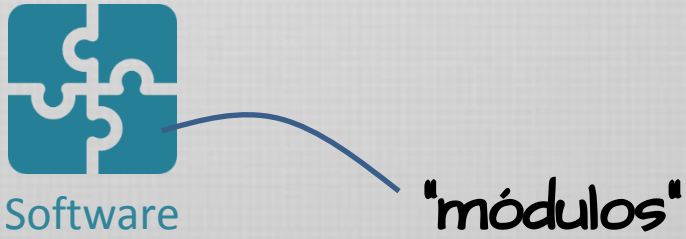
Un buen diseño...

¿Qué significa un buen diseño de software?

No se puede armar una definición o trazar una línea entre buenos y malos diseños en general.

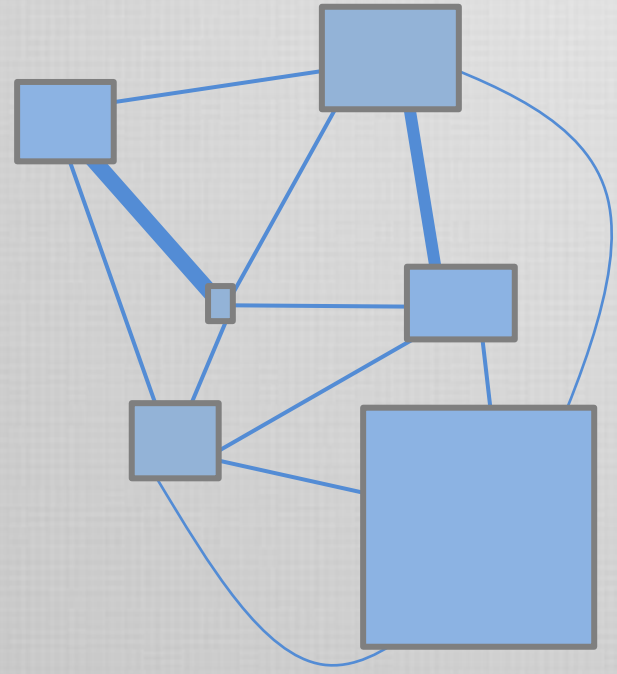
Pero hay muchas características que un buen diseño debería cumplir.

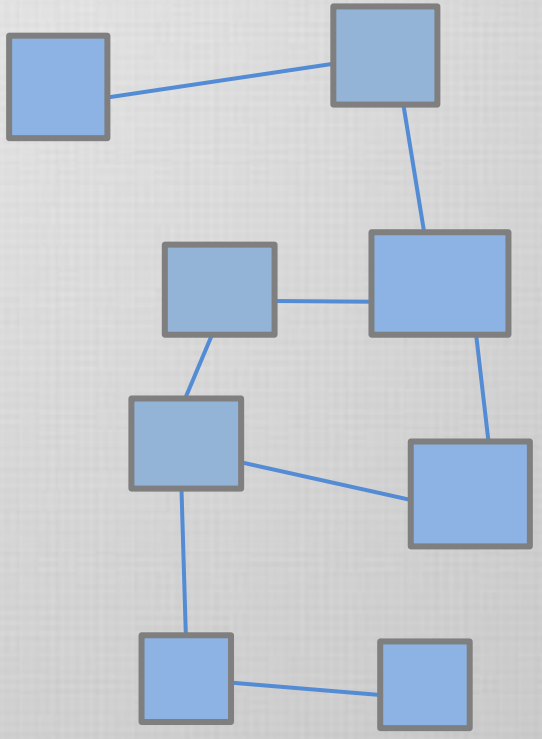
Principalmente, observar una buena organización en módulos.



Las partes se focalizan en un aspecto del sistema
Las partes son intercambiables
Las partes son independientes

¿*Buen* diseño modular?





Un buen diseño modular

Pocas Interfaces

Todo módulo debe comunicarse con pocos módulos,
tanto como sea posible.

Interfaces Pequeñas

Si dos módulos se comunican, deben intercambiar poca información,
tanto como sea posible.

Interfaces explícitas

La forma de comunicación entre dos módulos debe ser
obvia a partir de sus códigos.

Ocultamiento de Información

El diseño de todo módulo debe contener un subconjunto de propiedades que son públicas y un subconjunto de propiedades que son privadas.

Mapeo Directo

La estructura modular del sistema de software debe ser compatible con los elementos generados en el proceso de modelamiento del problema

Diseño modular y orientación a objetos

La programación orientada a objetos es un
paradigma de programación
que procura
favorecer un buen diseño modular

Propone una forma de observar el mundo real, sus elementos y sus relaciones, y plasmarlos en un diseño modularizado.

Esa forma de *modelar* favorece un *buen grado* de modularidad en el sistema.

(digamos que arrancamos bien,
pero no es suficiente)

Las ideas de la POO no se limitan al código sino que procuran gobernar
todo el proceso de creación del software.

Orientación a objetos

El método de diseño debe ser *modular*.
Debe satisfacer:

Descomposición modular

Ayuda a descomponer el problema en subproblemas simples y suficientemente independientes.

Composición modular

Favorece la producción de elementos de software que pueden combinarse para producir nuevos sistemas.

Comprensión modular

Ayuda a producir software en el cual el humano puede comprender cada módulo sin tener que comprender mucho de los restantes.

Continuidad modular

Favorece que un cambio pequeño en la especificación impacte en uno o pocos módulos.

Protección modular

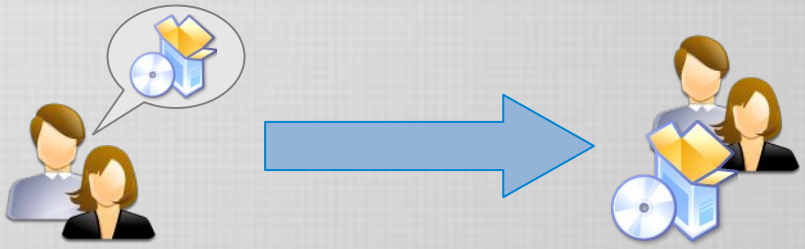
Favorece la contención de situaciones anormales en ejecución a uno o pocos módulos.

Orientación a objetos

La programación orientada a objetos es un paradigma de programación que procura favorecer un buen diseño.

Es un conjunto de ideas que acompaña todo el ciclo de vida del software.

Deben acompañar al paradigma:
- el método para diseñar y construir,
- las herramientas de soporte,
- el lenguaje de programación.



La "orientación a objetos" no es una condición booleana.

Un escenario A, aunque no 100% orientado a objetos, puede ser "más" OO que un escenario B.

No todos necesitan todas las propiedades OO todo el tiempo.

OO puede ser uno de los factores que guían la construcción del software.

Orientación a objetos

Las *orientación a objetos* es un método de construcción de software que satisface los principios anteriores.

Recordemos los objetivos principales:

confiabilidad – extendibilidad - reusabilidad

Es evolución de las ideas de abstracción de datos y modularización

Conceptos claves de OO

Los tres conceptos claves en la orientación a objetos forman la estructura estática y dinámica de todo software OO:

Los objetos, representaciones de los objetos del mundo real o conceptos.
Son la unidad de modularidad del software.
Empaquetan datos y operaciones
Son el centro del diseño del software

Los mensajes, por medio de los cuales los objetos se comunican entre sí.
Son requerimientos de que un objeto ejecute un procedimiento específico, conocidos como métodos o servicios.
Similares a las llamadas a funciones de lenguajes convencionales.

Las clases, que definen la estructura y el comportamiento de los objetos
Es la combinación del concepto de módulo y tipo de dato

Mecanismos claves en OO

Los tres mecanismos claves en la orientación a objetos:

Encapsulado, el mecanismo para empaquetar datos y procedimientos en los objetos.
Los objetos sustentan el encapsulado.

Polimorfismo, la habilidad de implementar el mismo mensaje en formas diferentes en objetos diferentes.
Los mensajes sustentan el polimorfismo.

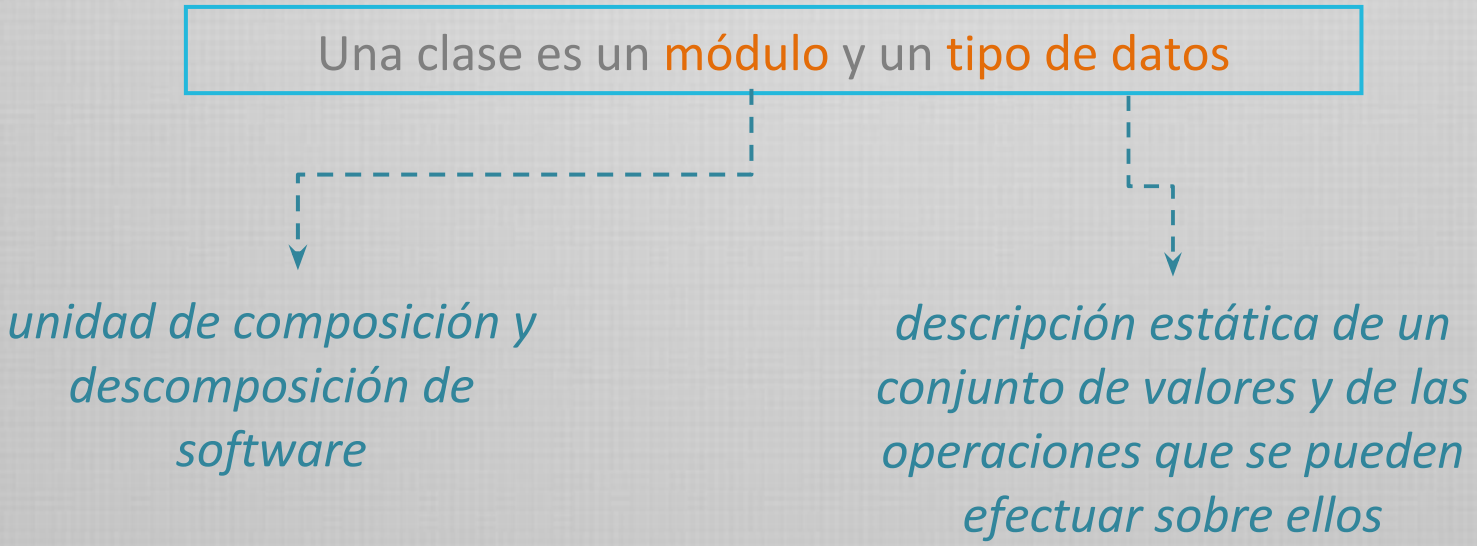
Herencia, el mecanismo para diseminar la información definida en clases *genéricas* a otras clases consideradas casos especiales de las anteriores.
Las clases sustentan la herencia.

Estos tres mecanismos son generalmente aceptados como los ingredientes necesarios de la orientación a objetos (no los conceptos!).

Conceptos de Orientación a Objetos - Clases

El concepto de clase es fundamental en OO, ya que conforman los módulos del sistema de software...

La construcción de software orientado a objetos es un método de desarrollo que que basa la construcción de sistemas de software en módulos obtenidos a partir de los tipos de objetos que manipula.



Conceptos de Orientación a Objetos - Clases

Una clase es un patrón o esquema que especifica los atributos y servicios compartidos por todos los objetos que pertenecen a ella.

Una clase es un tipo de dato abstracto posiblemente equipado con una implementación total o parcial

Un objeto es una instancia de una clase

Decir

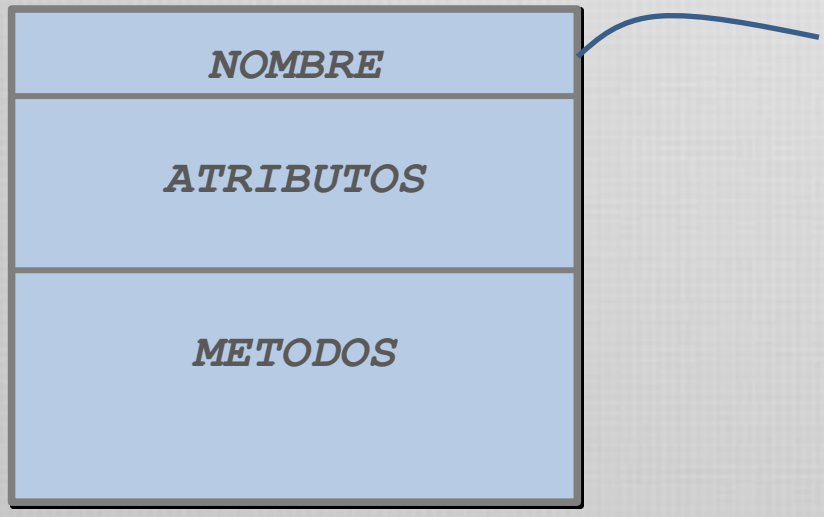
"x es instancia de la clase T"

es lo mismo que decir

"x es objeto de tipo T"

¿Cómo escribimos o especificamos una clase?

Toda clase tiene tres elementos básicos:

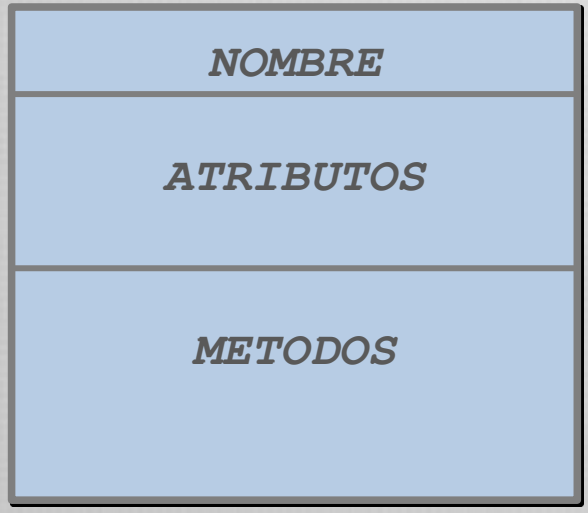


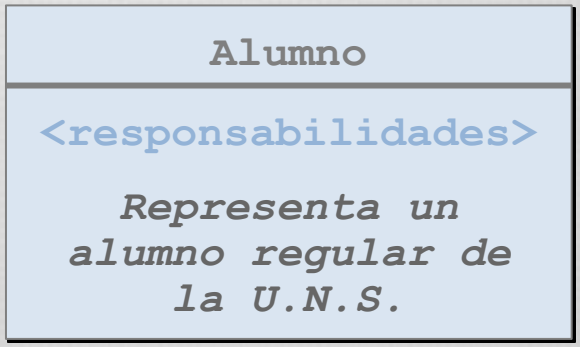
El nombre debe representar a la abstracción del conjunto de las instancias de la clase, y debe ser único en todo el sistema

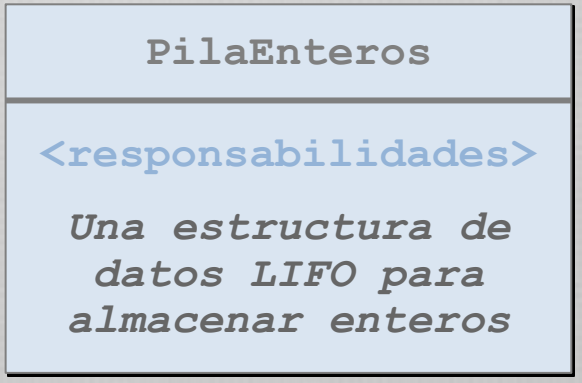
Generalmente, está acompañada de un texto denominado responsabilidades, que explica informalmente el objetivo de la clase

¿Cómo escribimos o especificamos una clase?

Toda clase tiene tres elementos básicos:

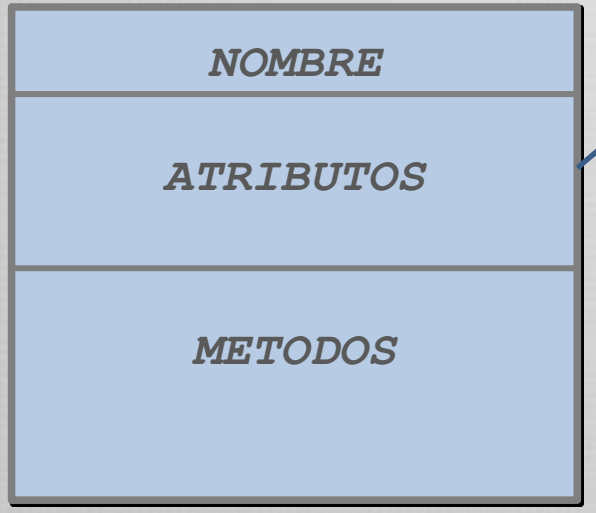






¿Cómo escribimos o especificamos una clase?

Toda clase tiene tres elementos básicos:

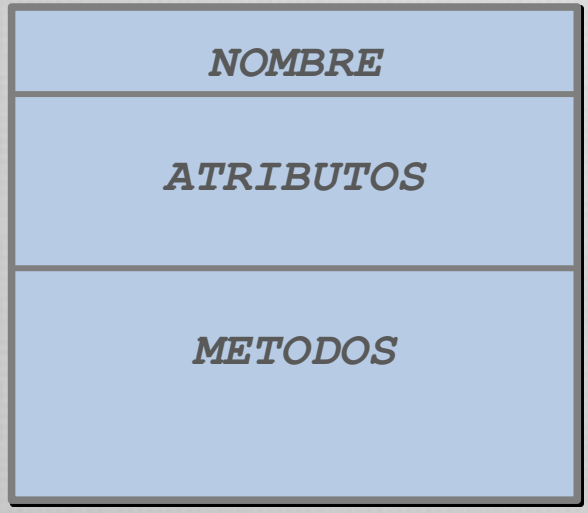


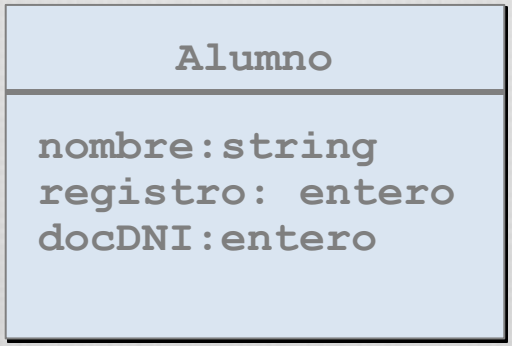
Los atributos describen la representación interna de los valores que son instancia de la clase.

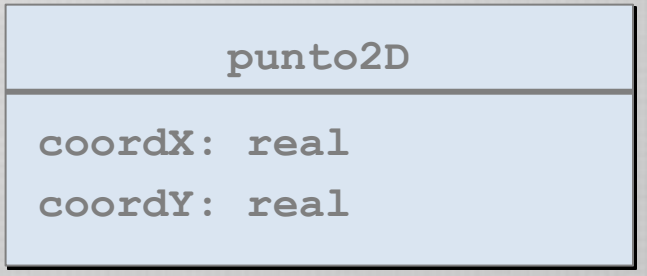
Según el lenguaje y la política de diseño, pueden ser accesibles desde afuera de la clase

¿Cómo escribimos o especificamos una clase?

Toda clase tiene tres elementos básicos:

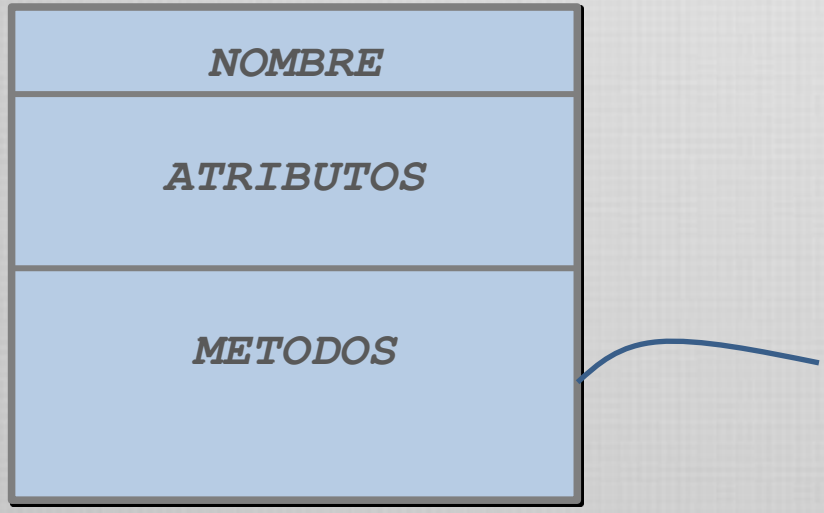






¿Cómo escribimos o especificamos una clase?

Toda clase tiene tres elementos básicos:

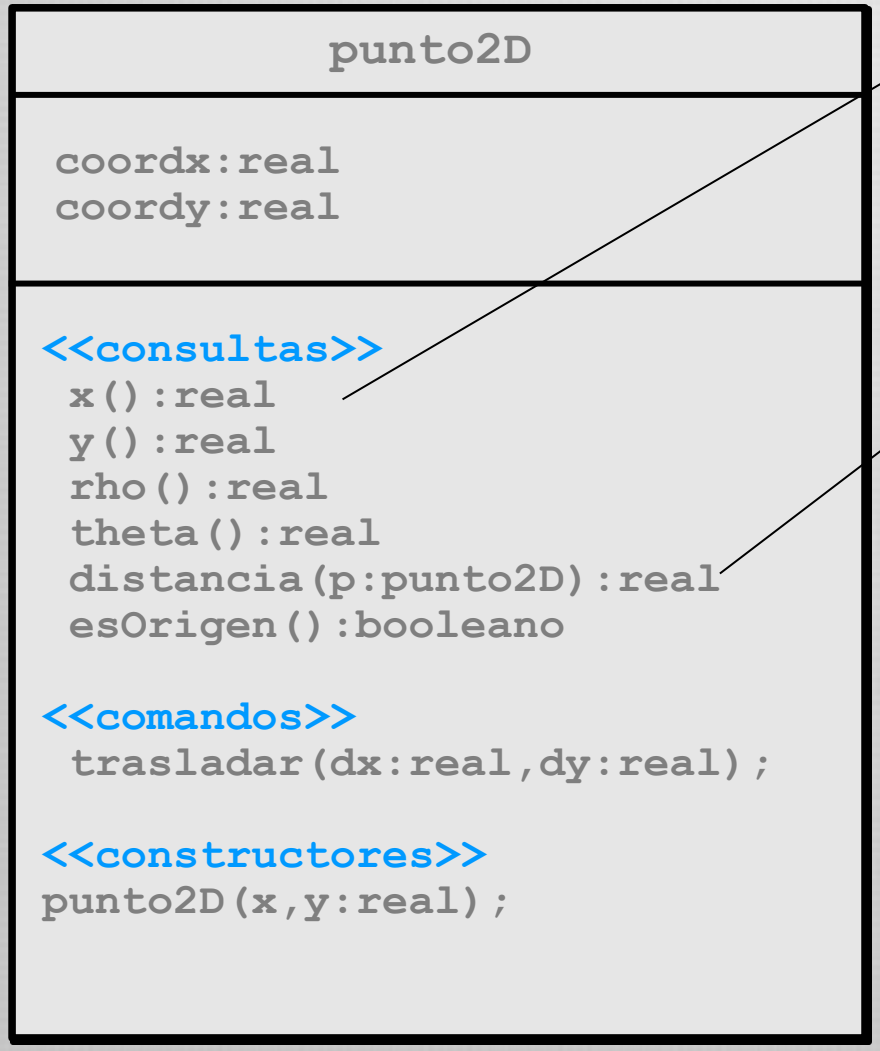


Los métodos o servicios son las operaciones que se pueden efectuar sobre las instancias de la clase.

(Preferentemente) son las únicas interfaces definidas entre módulos

Ejemplo - Clases

Clase Punto en dos dimensiones




```
x(): real
{
  Resultado ← coordx

}
```

```
distancia(p:punto2D): real
{
 dx ← coordx-p.x()
 dy ← coordY = p.y()
 Resultado = raizCuad(xd*xd + yd*yd)
}
```

Modelo de computación

El modelo de computación de OO se basa en las siguientes reglas:

- En ejecución, un sistema de software esta conformado únicamente por objetos.
- Todo objeto es instancia de una clase
- En todo momento existe un único objeto en ejecución, denominado instancia actual o corriente.
- Toda computación se realiza cuando un objeto resuelve un pedido de un servicio de otro objeto
- El objeto que llama se denomina cliente, y el que realiza la acción se denomina servidor.
- El cliente realiza un pedido al servidor de la siguiente forma:

```
<nombre_servidor>.<nombre_operación><parámetros>
```

```
unTriangulo.perimetro()
empleado7G.legajo()
```

La operación es uno de los métodos (o servicios) declarados en la clase del objeto servidor.

Objetos

- Todo objeto O es instancia de una clase C.
  C es la clase generadora de O.
- Cada clase puede tener *una*, *varias* o *ninguna* instancia en un momento determinado de la ejecución
- Algunos objetos serán una representación directa de objetos del mundo real (*mapeo directo*!).

  Ejemplo: clase EMPLEADO

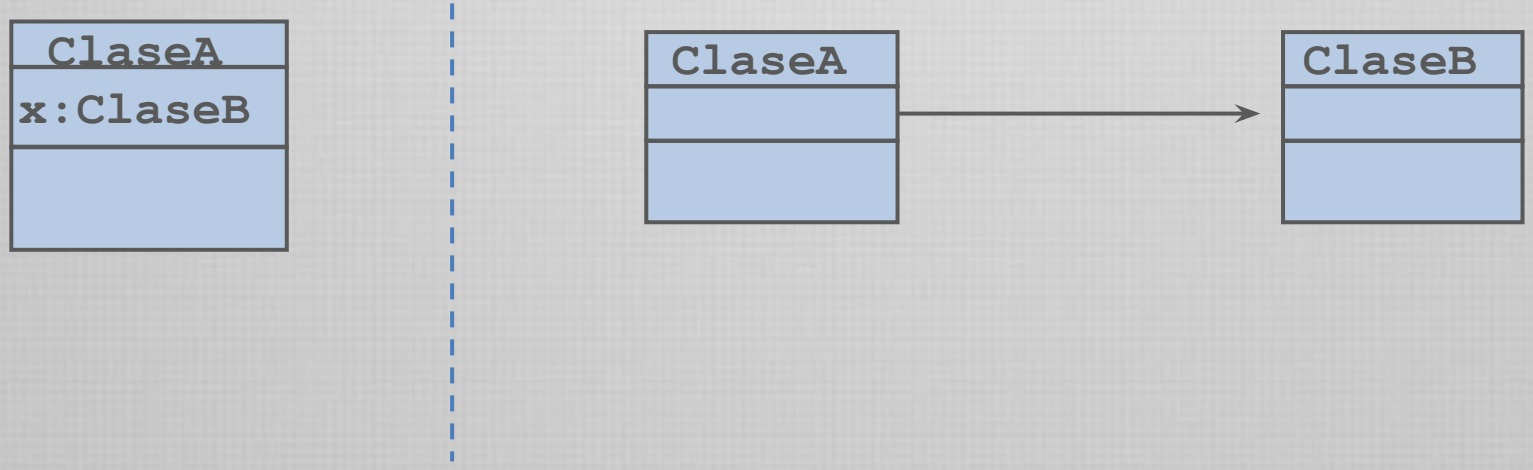
- Otros serán representaciones de abstracciones correspondientes a la solución del problema (*estructuras de datos, archivos, bases de datos, etc*)
- Otros se originarán en la implementación de la solución: componentes gráficos, manejadores de eventos, etc.

Relaciones entre clases - *Asociación*

Las clases que conforman un sistema pueden relacionarse de diferentes maneras.

Típicamente, si uno de los atributos de una clase A es de clase B,
las clases A y B están asociadas.

Una de las clases "conoce" a la otra y puede comunicarse con ella.



Relaciones entre clases

No es la única relación existente....

Una clase puede ser una extensión de otra clase

Una clase PilaRápida con las mismas operaciones de Pila pero ademas con una operación desapilarDos() que desapila dos elementos.

Una clase puede ser una especialización de otra clase

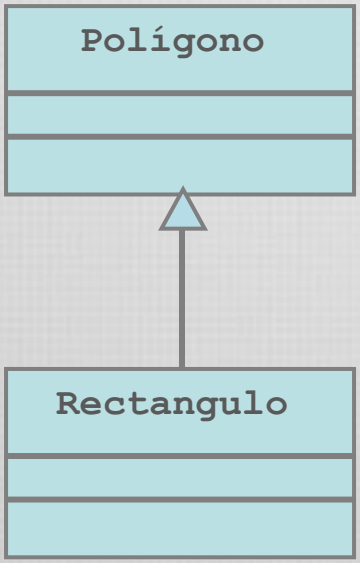
Un cuadrado es un caso especial de polígonos. La clase Cuadrado es una especialización de la clase Poligono.

Una clase puede ser una combinación de otras clases

Un ayudante B es un alumno de la Universidad y un docente de la Universidad. Es una combinación de la clases Alumno y Docente

Relaciones entre clases - Herencia

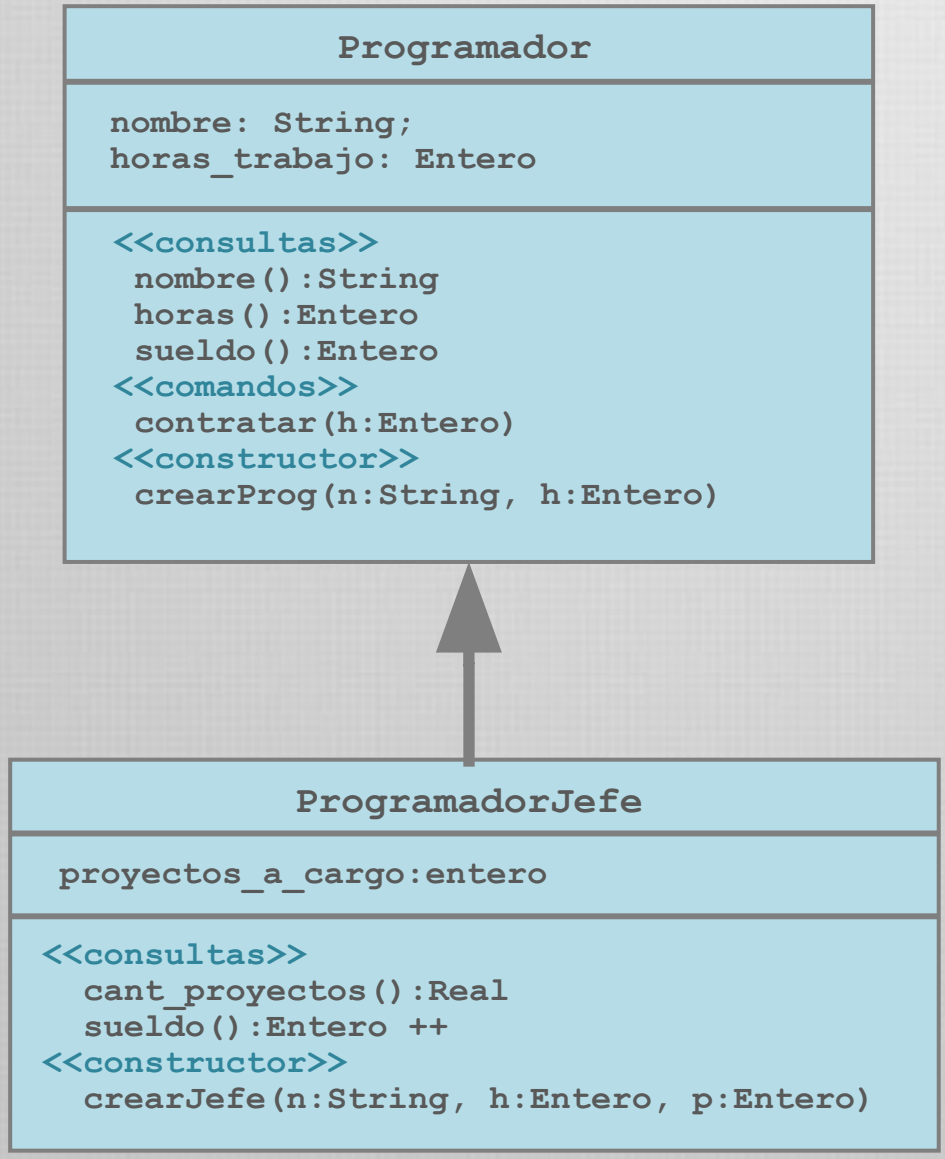
El concepto de herencia permite modelar estas relaciones.



"es un"

La herencia es un componente central en la orientación a objetos.
Es inconcebible un lenguaje OO sin un mecanismo de herencia.

Relaciones entre clases - Herencia



Como indicativo de una operación redefinida utilizamos el símbolo "++"

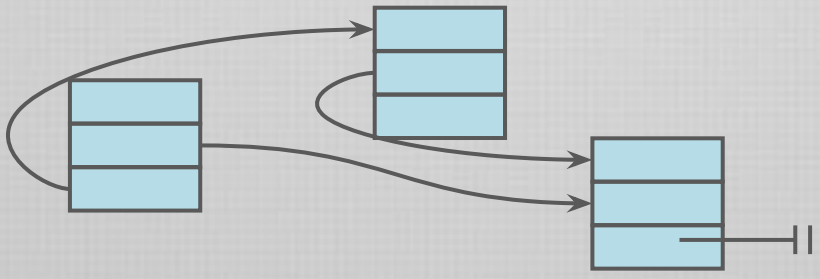
La nueva versión de la operación debe coincidir en el signature con la original

Objetos y referencias

Una referencia es un valor que puede estar nulo o asociado.

Si la referencia está asociada, entonces identifica un único objeto entre los demás que componen el sistema.

Usaremos flechas para indicar referencias asociadas a otros objetos en el sistema



Objetos y polimorfismo

Las instancias de una clase son los objetos que son instancia de algún descendiente de la clase.
Las instancias propias son las instancias de la misma clase

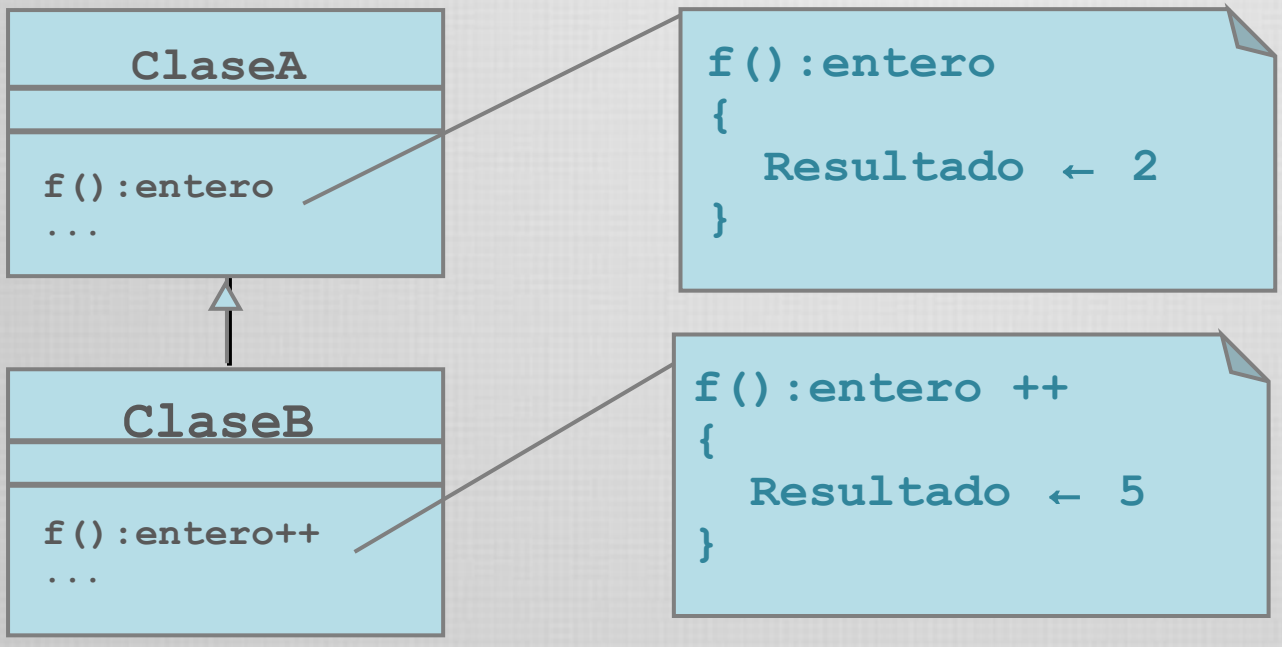
Una asociación polimórfica ocurre cuando a una referencia de una clase se le asocia una referencia a una instancia no propia.

```
p: Poligono ; re:Rectangulo ; tr:Triangulo
```

Es válido realizar las siguientes asignaciones:

```
p ← re;
p ← tr;
```

Vinculación (ligadura) dinámica




```
x:ClaseA; y:ClaseB;

x ← new ClaseA();
y ← new ClaseB();
a ← x.f();
x ← y;
a ← a + x.f();
```

a finaliza con valor 7