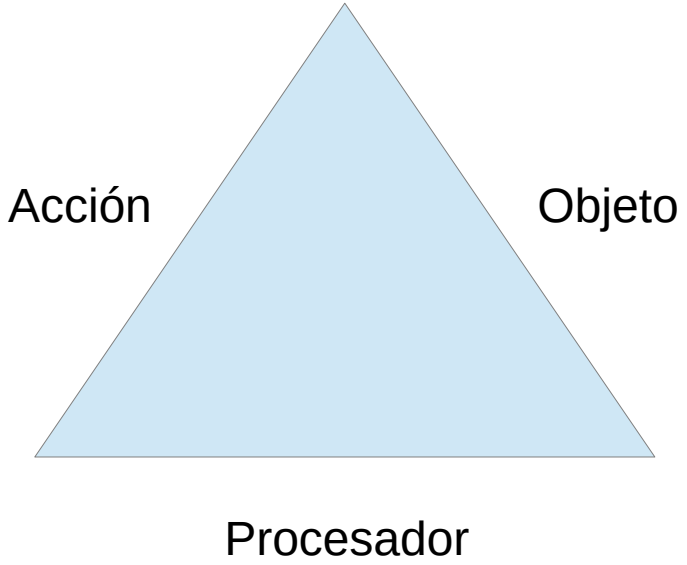
Programación Orientada a Objetos

Tema 6: 4 Pilares de la Tecnología OO

Tiángulo básico de la computación

- Las tres fuerzas de la computación:

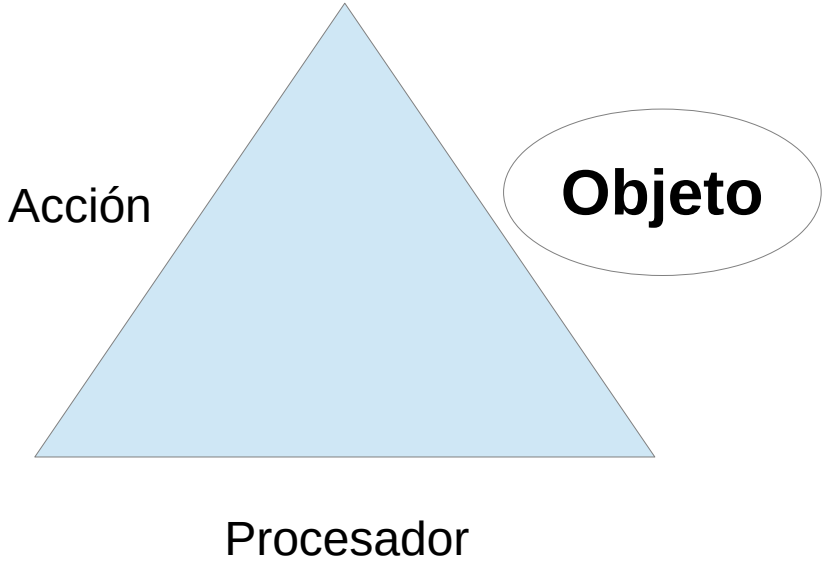


(A veces se denota como: acción = funciones, objeto=dato)

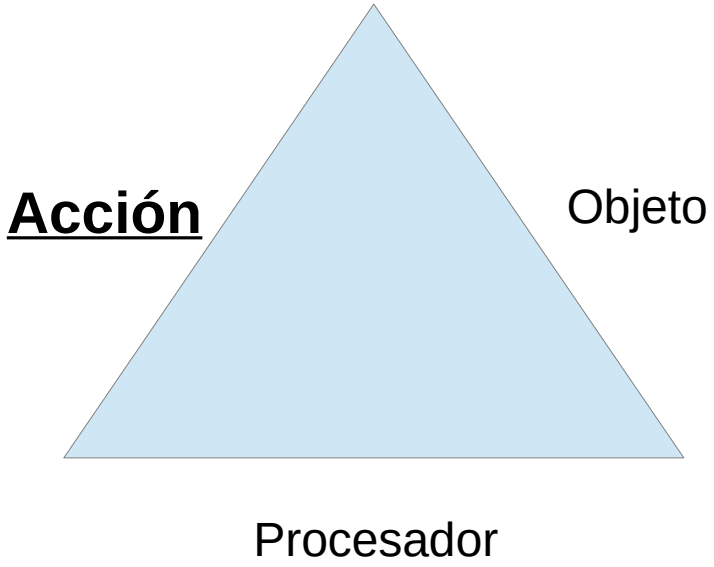
[Bertrand Meyer 99]

Visión Orientada a Objetos

... es la modularidad centrada en los objetos



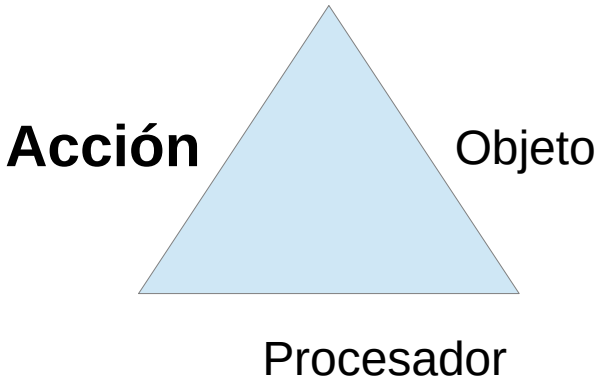
Enfoque Tradicional



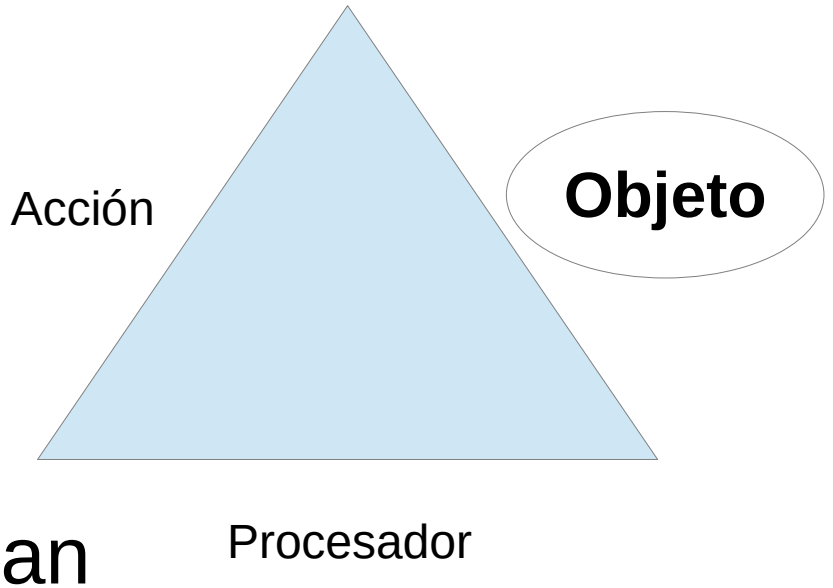
modularidad basada en la acción

Enfoque Tradicional

- Basada en funciones
- Decisiones de diseño prematuras y estratégicamente poco dispuestas a características de calidad deseables
- Excesiva importancia a la interfaz
- Se ha demostrado poco adecuado para sistemas de medio/gran tamaño

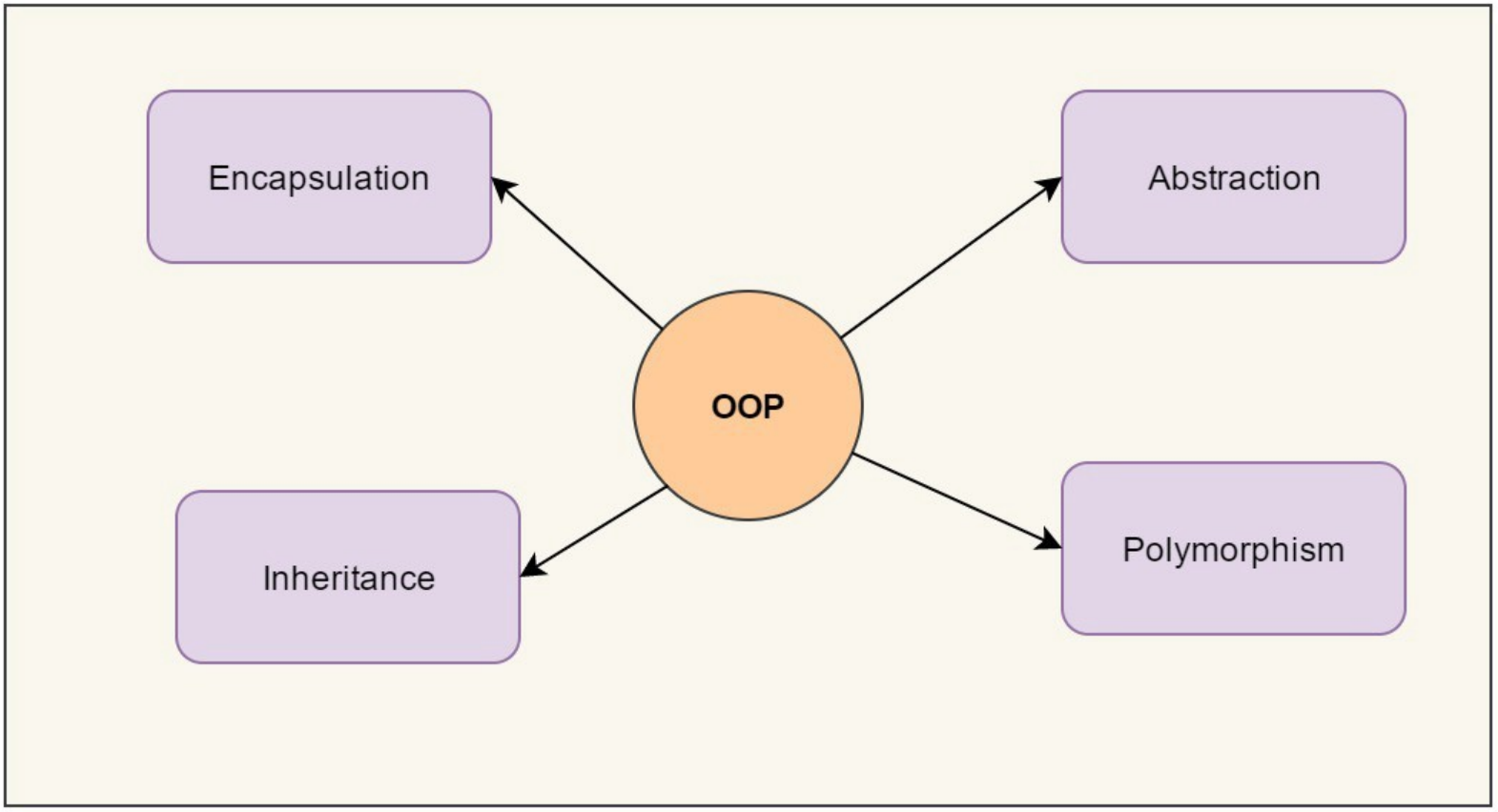


Enfoque OO



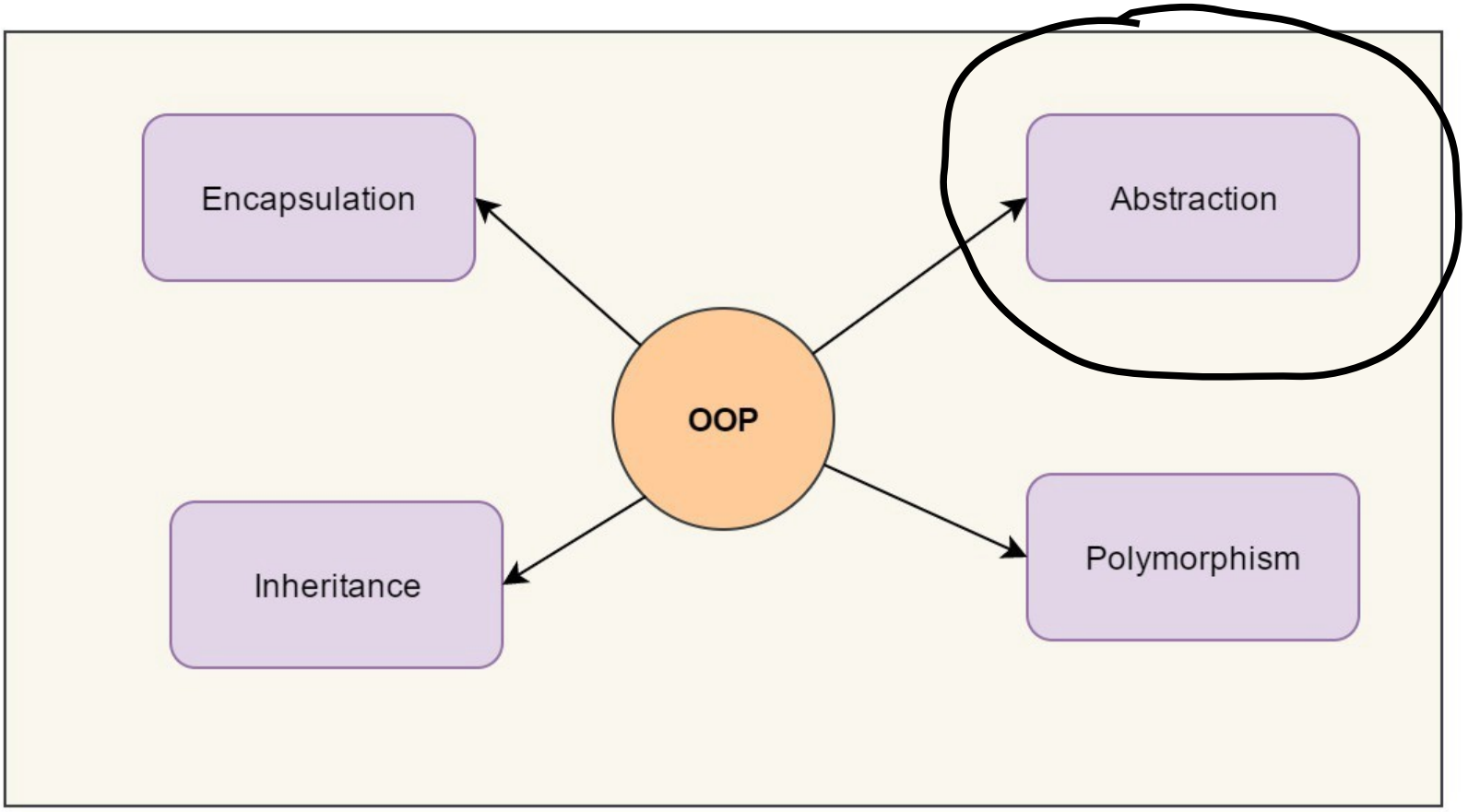
- Basada en los objetos
- Los módulos se deducen de los objetos que se manipulan
  - No hay diseño prematuro basado en una función
  - Los objetos permanecen
  - La interfaz, una función aparte más
- **No es:** *qué hace el sistema,* **es:** *¡a qué se lo hace!*
- Diseño pensando en reutilización
- … y en la extensibilidad, mantenimiento...
- Mayor simplicidad
- Más adecuado para grandes sistemas

4 pilares de la OO



Four Pillars of Object Oriented Programming

4 pilares de la OO



Four Pillars of Object Oriented Programming

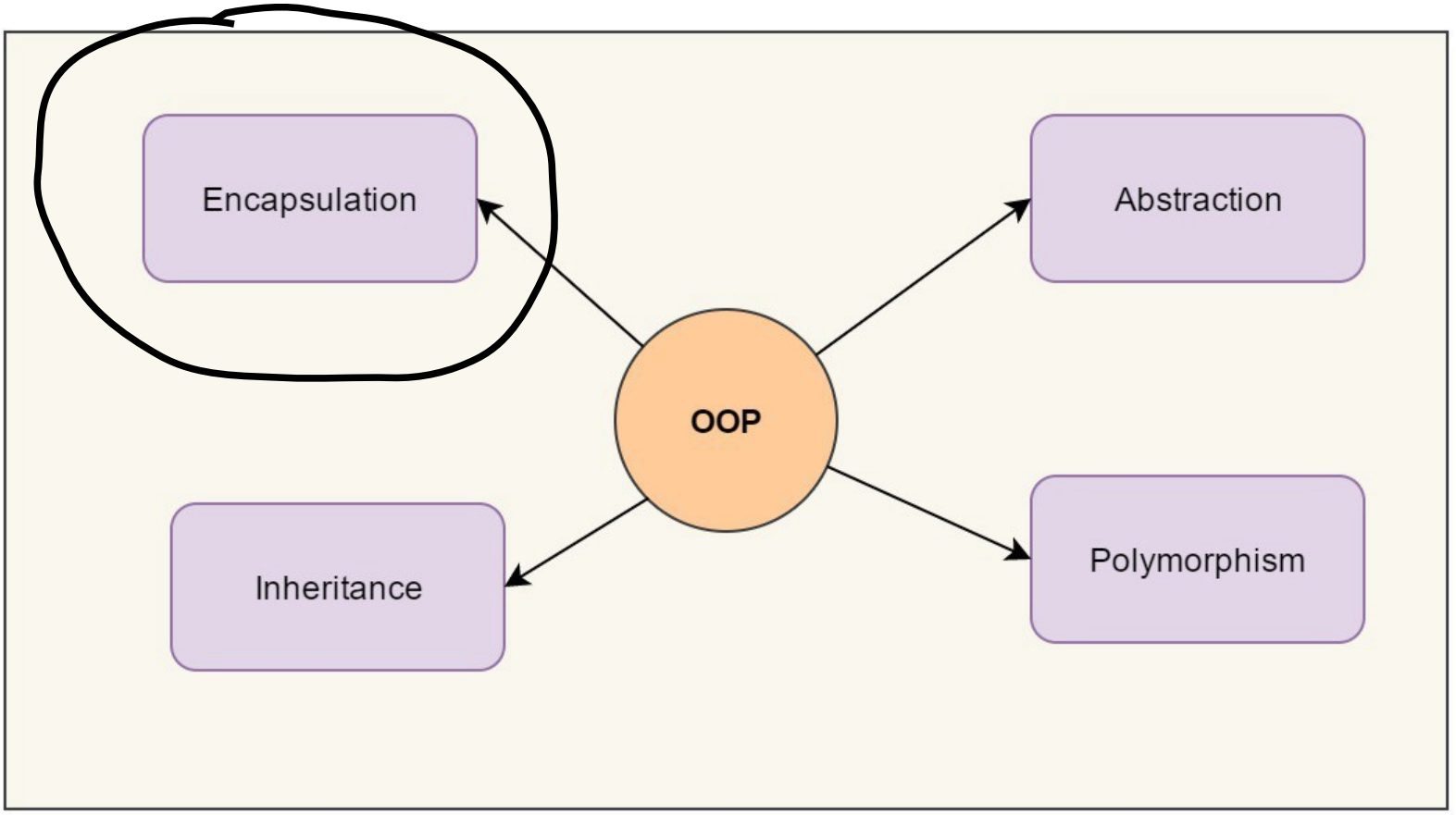
Abstracción: las clases y los objetos

- TAD *una clase* >---------- *separado de* < implementación *equipado con una*

- El cliente entiende y usa la clase de forma abstracta

- Acceso a una clase mediante métodos (mensajes)
  - La interfaz. También se denominan métodos exportados.
  - Un método puede invocar una operación compleja o devolver un simple atributo interno: en ambos casos se usan igual. Esto evita que el usuario acceda a la representación interna y simplifica el uso.

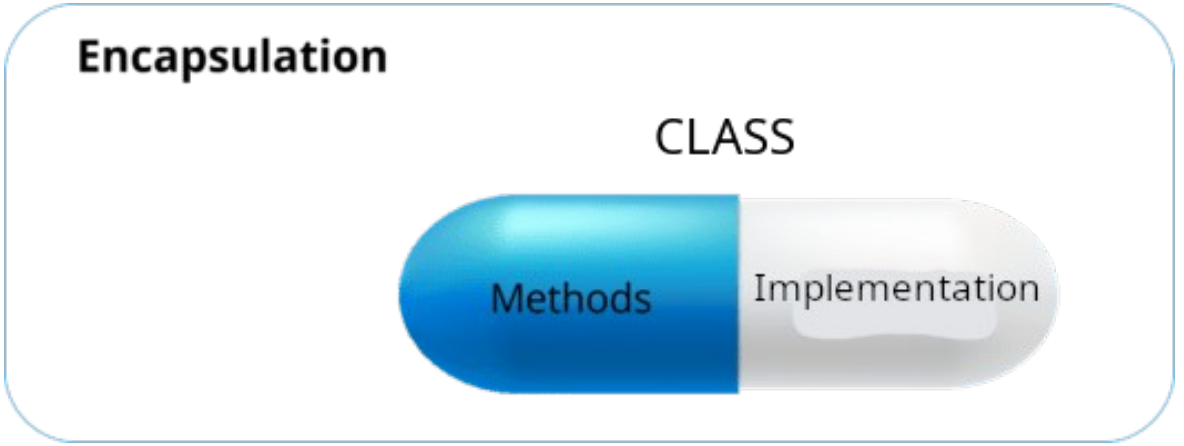
4 pilares de la OO



Four Pillars of Object Oriented Programming

Encapsulamiento. Ocultación/protección

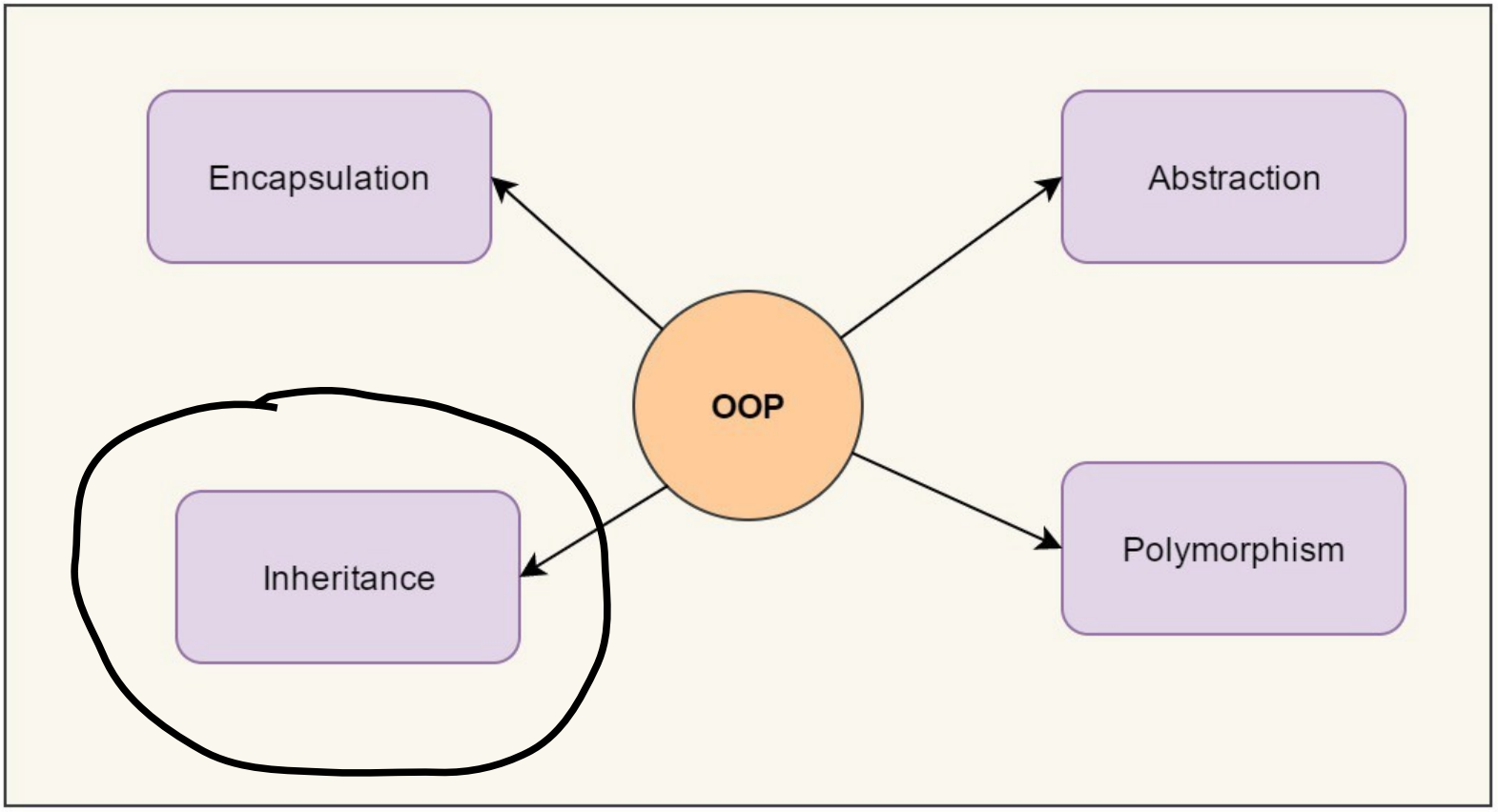
- La implementación queda oculta en una “cápsula”. Protegida de accesos indebidos.
- Ocultar la visión interna del tipo/clase/objeto a modo de "cápsula"
- Solo está permitido el acceso a través de los métodos



Encapsulamiento. Ventajas

- Impide acceso directo al estado interno de un objeto:
  - Impidiendo operaciones no permitidas.
  - Simplifica su comprensión ya que no será necesario conocer como está hecho internamente.
  - Si en el futuro se modifican sus datos internos, no afectará a ningún programa que use dicho objeto ya que nunca se accede a ellos.
- Solo podrá interactuarse con el objeto mediante su interfaz pública.

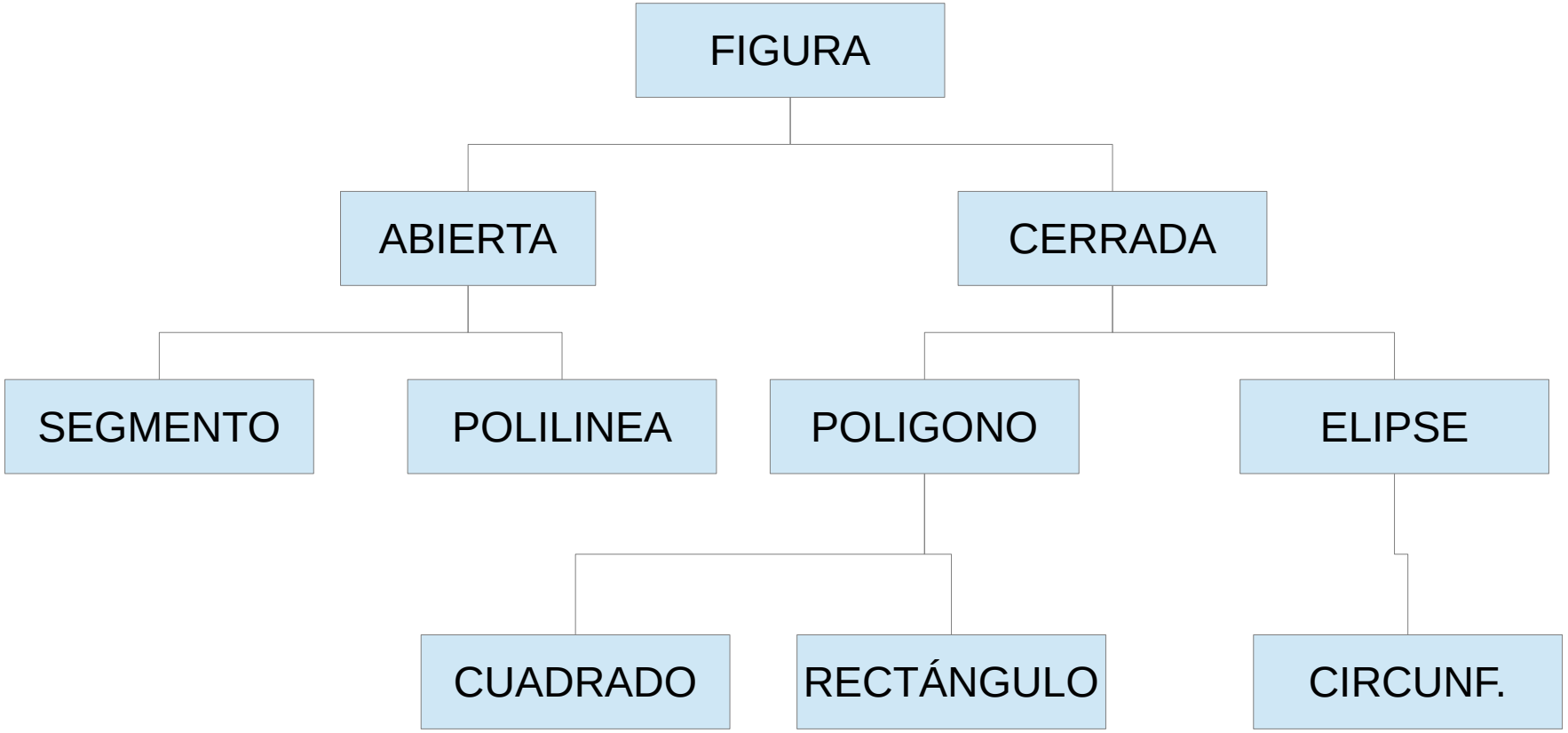
4 pilares de la OO



Four Pillars of Object Oriented Programming

Herencia (inheritance)

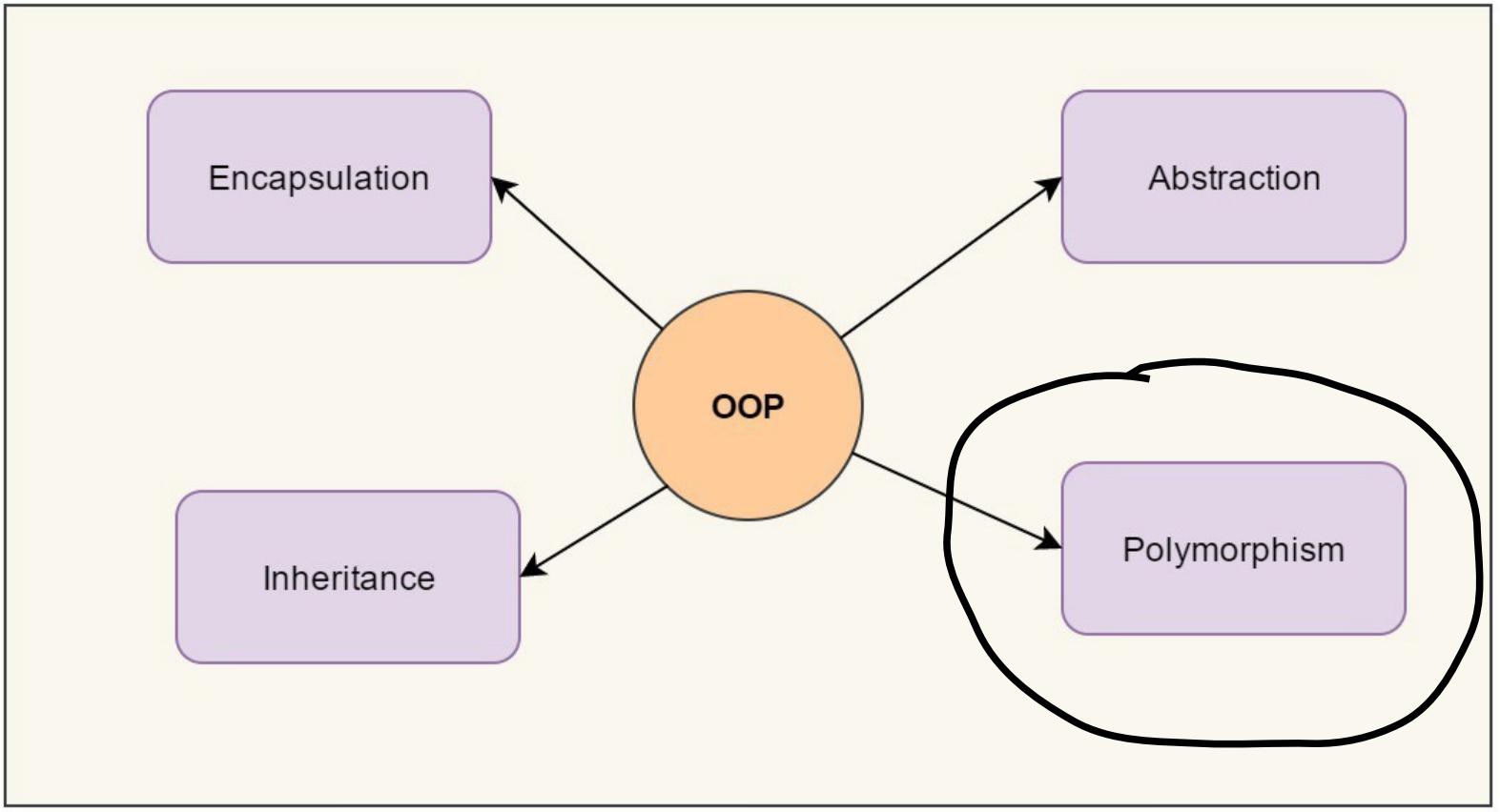
- Estructura de reutilización jerárquica específica de la POO.
- Definiendo subclases se facilita la reutilización y la extensibilidad.
- Útil en la descomposición modular: asigna comportamientos por niveles.
- Clase base (ancestro, superclase, padre, madre...) y clase derivada (descendientes, subclase...)



Tipos de clases

- Clase **diferida** (abstracta, ...):
  - Alguno de sus componentes es diferido: no está implementado en la propia clase
  - Determinan la interfaz/comportamiento de sus descendientes
  - Describen un interfaz genérico
  - C++: tiene funcionas virtuales o funciones virtuales puras (clases abstractas)
- La clase no diferida es una clase **efectiva**

4 pilares de la OO



Four Pillars of Object Oriented Programming

Polimorfismo (polymorphism)

- **Definición:** ocurre cuando se dispone del mismo interfaz sobre entidades (objetos) de distinto tipo.
- Hay varios tipos de polimorfismo
  - Estático (sobrecarga de funciones y op. en C++)
  - Estático paramétrico (plantillas/templates en C++)
  - Dinámico (de subtipo, polimorfismo en tiempo de ejecución en C++, vinculación dinámica)

Polimorfismo

- Polimorfismo estático (o *ad hoc*)
  - Forma más simple de polimorfismo
  - También llamado: sobrecarga de funciones, sobrecarga de operadores
  - Se define una función/operador diferente para cada tipo

la función/operador se hace caso a caso, *ad hoc*, para cada tipo

Polimorfismo

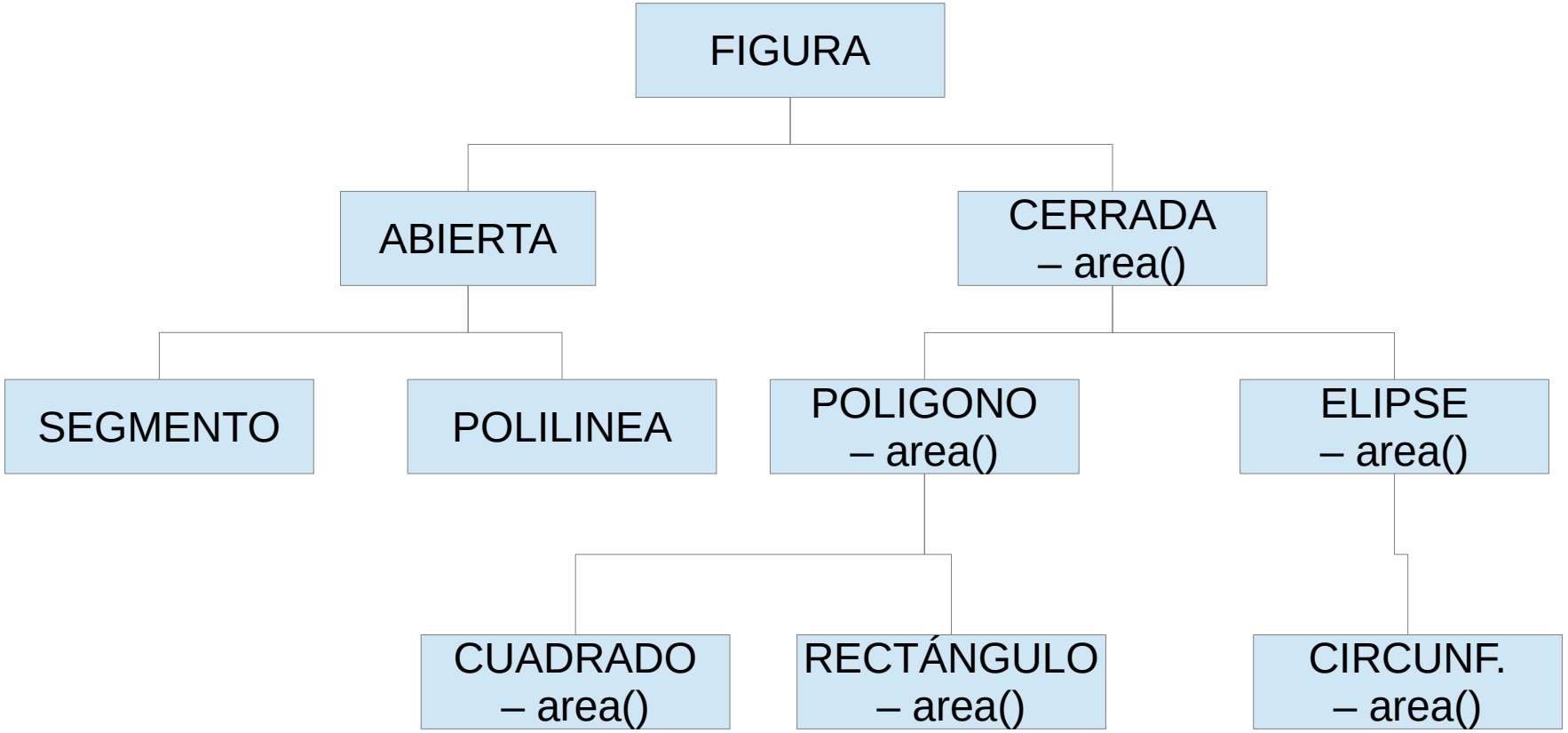
- Polimorfismo paramétrico (estático)
  - El código usa un tipo genérico que se especifica como parámetro cuando se instancia
  - La interfaz es la misma indep. del tipo instanciado
  - Plantillas de clase y de función en C++
    - C++ *Standard Template Library* (STL)
    - *Function templates*
    - *Class templates*

Polimorfismo

- Polimorfismo de subtipo (subtipado)
  - Vinculación dinámica
  - En tiempo de ejecución
  - Permite construir código genérico indep. del tipo
  - C++: punteros a la clase base y funciones virtuales
  - Ej: la clase *Figura* que hemos visto en anteriores temas

Polimorfismo

- En el polimorfismo de subtipo una familia de clases tienen el mismo comportamiento:
  - Los subtipos comparten interfaz
  - Potencia semántica, expresiva, sencillez
  - Acceso uniforme, etc.
  - Se puede escribir el mismo código para todas ellas



Sistema Software Orientado a Objetos

- Conjunto de clases (ensamblado de clases)
- Desde la **clase raíz** hasta el **despliegue** de todo el sistema
- ¿Y el programa principal?
  - En OO no juega un papel tan “principal”
  - Puede ser un simple método que inicie el despliegue
  - ¿Cuál es el programa principal de un SO, una compleja aplicación, una web, etc. ?
  - Son más bien los servicios que ofrece, lo que describe el sistema
- Sistema **cerrado**: si contiene todas las clases que necesita la clase raíz

Gestión de memoria

- Memoria Estática
- Memoria basada en pila
- Memoria Basada en montículo (libre, dinámica)
- Desconexión/*detachment* de memoria
- Objetos inalcanzables:
  - Problemas de memoria
- LP modernos incorporan: **recolector automático de basura**
  - Todos los objetos que se recolecten deben ser inalcanzables
  - Todos los objetos inalcanzables deben ser recolectados

Conclusiones

- La Tecnología OO facilita la reutilización, y otras muchas características de calidad deseables en el software.
- Formación en OO para explotar bien y en profundidad la orientación a objetos para lograr los objetivos de reutilización y calidad del software.