Introducción a OOP

Programación Orientada a Objeto

Evolución

- Programación no Estructurada,
- Programación procedimental,
- Programación modular y
- Programación orientada a objetos.

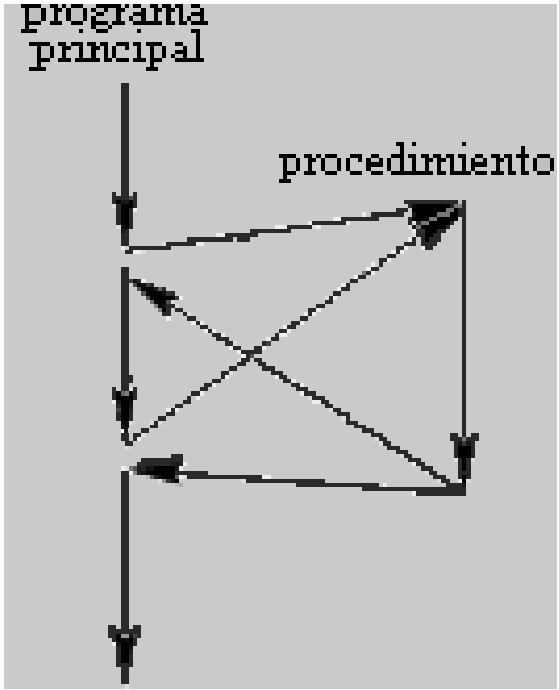
Programación no Estructurada

- Un solo bloque principal
- Variables globales

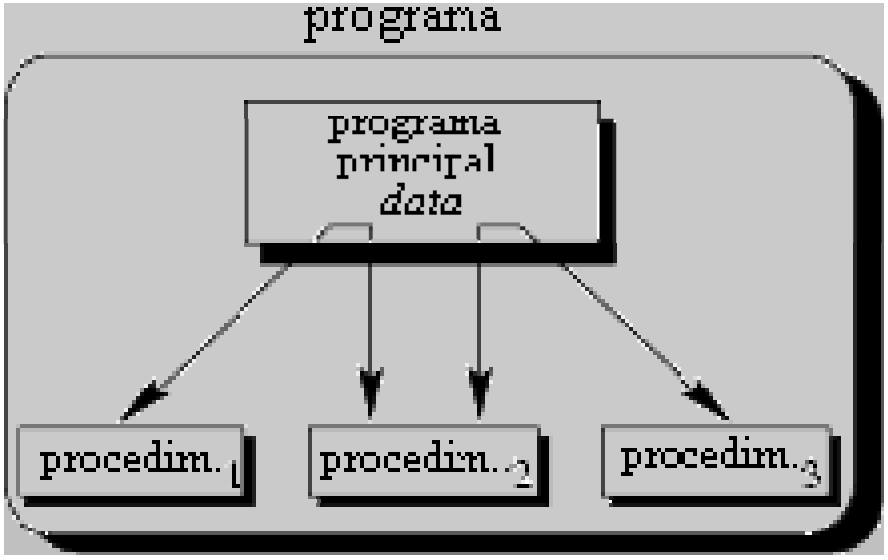
Cuando se repiten las secuencias→ Procedimientos

Programación Procedimental

- *llamada de procedimiento*
- Variables globales vs Locales y p*arámetros*



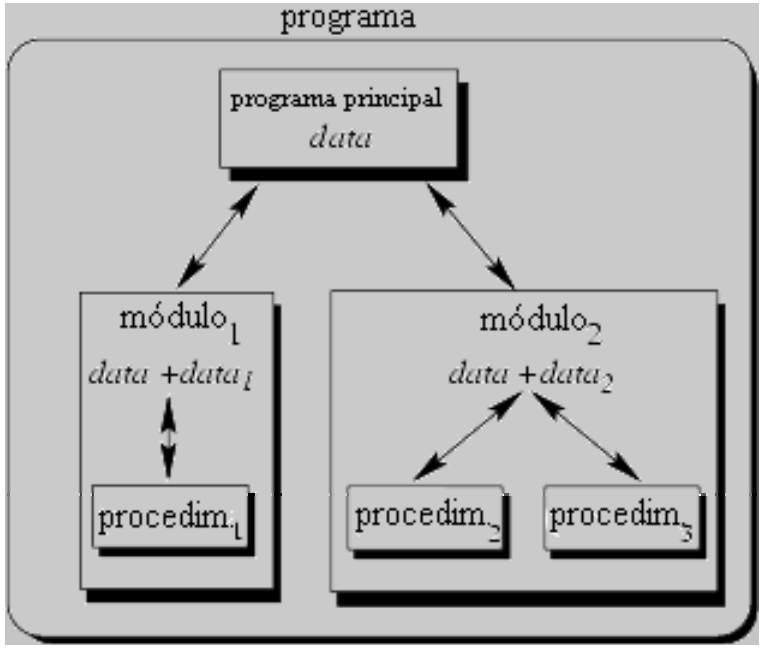
Necesidad de Módulos



Programación Modular

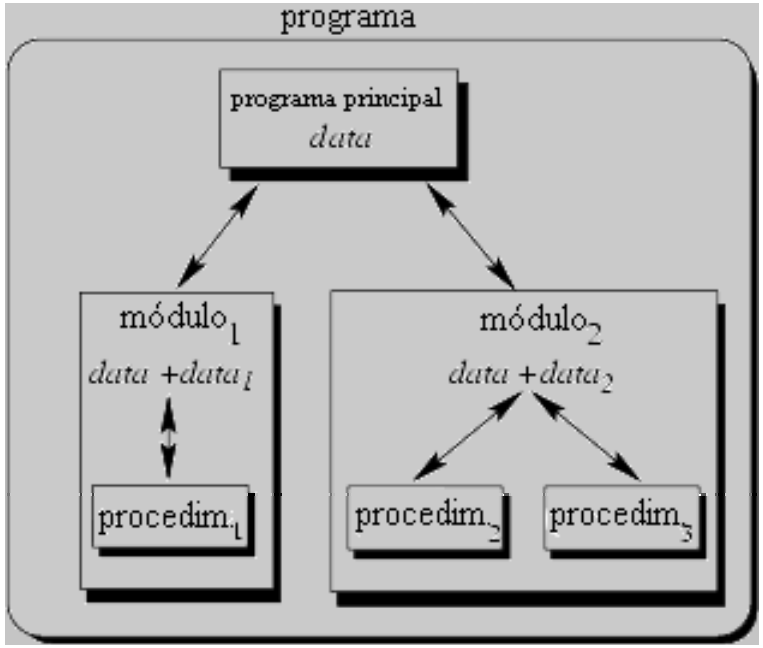
- *Procedimientos con una funcionalidad común*
- Módulo puede contener sus propios datos
- Espacio de Nombres

Problemas?



Problemas

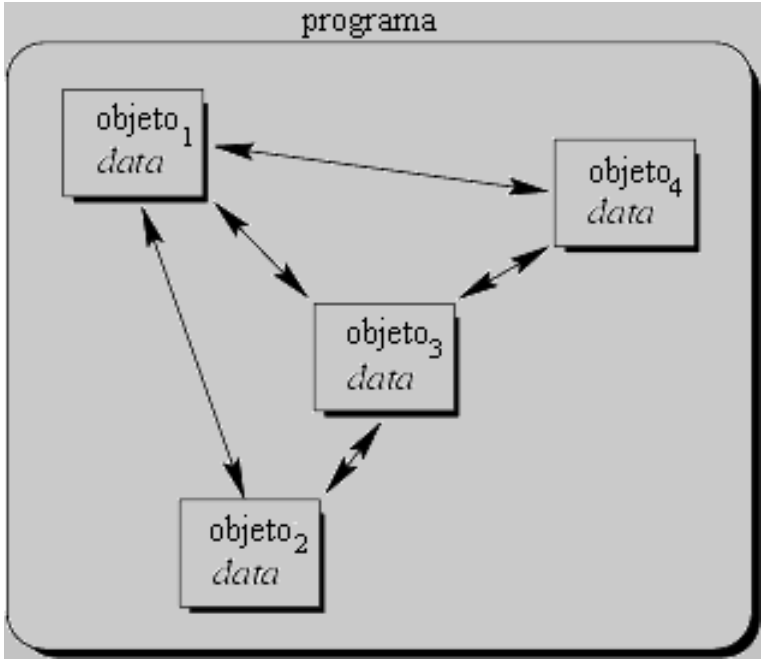
- Creación y Destrucción Explícitas
- Datos y Operaciones Desacoplados
- Omisiones en la Consistencia de Datos



Programación Orientada a Objetos

Los objetos del programa interactúan mandando mensajes

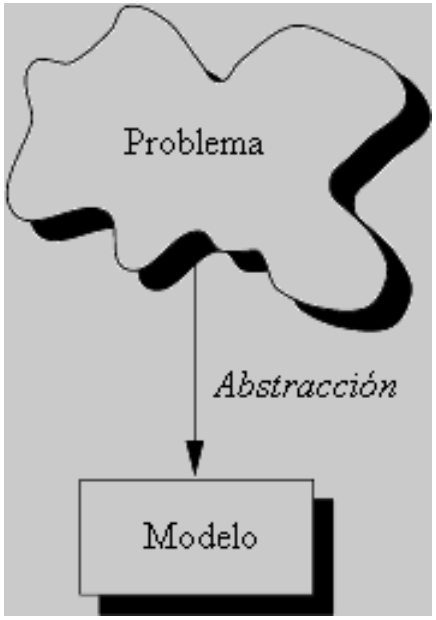
- Contienen el Qué → Datos
- Contienen el Cómo → Métodos

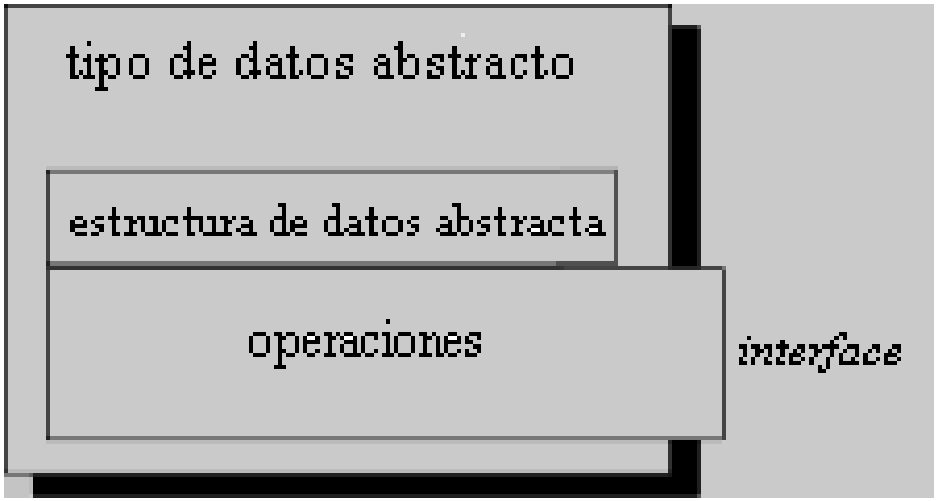


Manejar el Mundo Real

proceso de modelado se llama *abstracción*

- los *datos* que son afectados
- las *operaciones* que son identificadas





Construcción de un Programa OOP

Visión general:

- Objetos de diferentes clases que piden servicios a otros objetos.
- Cada servicio ofrecido se realizará el guión marcado por un método utilizando la información enviada en el mensaje
- Los objetos son entes con información y funcionalidad.

El diseño orientado a objetos consistirá en:

1: **Delimitar las clases** de objetos que existirán en el universo de nuestro programa, y cómo se pueden relacionar los objetos de las diferentes clases entres sí.
2: **Implementar los métodos** y los atributos necesarios.
3: **Construir un programa** con las instrucciones que instancian los objetos necesarios para iniciar la ejecución (situación inicial del universo), inicializando estos objetos a los estados necesarios para que el sistema comience a funcionar correctamente.

¿Cuáles son las ventajas de un lenguaje orientado a objetos?

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Construcción de prototipos
- Agiliza el desarrollo de software
- Facilita el trabajo en equipo
- Facilita el mantenimiento del software

Características de Lenguaje OOP

- Reutilización
  - Responsabilidades y contratos
  - Abstracción, ocultación y encapsulamiento
  - Herencia
  - Polimorfismo
- Abstracción
  - Diseño orientado a objetos provee la abstracción
  - Polimorfismo
- Ocultación
  - Principio de ocultación
  - Ámbitos
- Encapsulamiento
  - Empaquetar información y funcionalidad, (qué y cómo)

La Clase

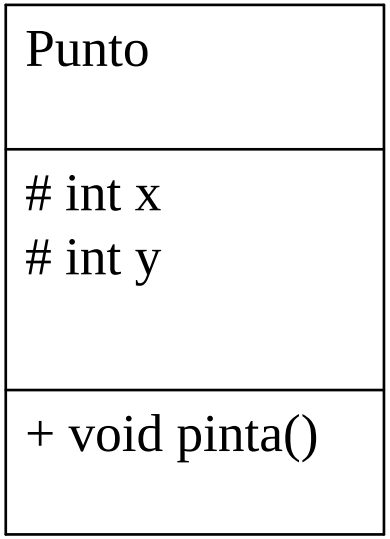
Los **objetos** son de una **clase y** las **clases** son **plantillas** que definen un comportamiento común a todos los objetos de la misma.

Las clases **amplían** el concepto de **TDA** con la inclusión de la **herencia**

- Clase como patrón
- Responsabilidades/contratos
- Diferencia entre objeto y clase
- Herencia
- Clase abstracta

La Clase

- **UML**
  - Atributos
  - Métodos



De clase y de instancia

Clases estáticas

Principio de ocultación: ámbitos

- Público
- Privado
- Protegido (se verá con la herencia)

El Objeto

Objeto

- ¿Qué es? (datos y procesamiento)
- Concreción de clase
- Estado concreto y propio (no compartido)
- Se comparten los atributos de clase (especie de globales)

- Consciencia de sí mismo: **this**
- Consciencia de su clase
- Ciclo de vida

Constructor

Vida (sucesión de estados)

Destructor

El Objeto II

Mensaje

Invocación de método

Método

Código que indica cómo debe responder a un mensaje

- De clase
- De instancia
- Abstracto (¿qué pasa con la instanciación?) (clase abstracta)
- Final

Atributo

Dato que ayuda a mantener el estado de un objeto.

- De clase
- De instancia
- Valores por defecto
- Constantes

Errores comunes en Diseño OOP

- Clases que hacen modificaciones directas a otras clases
- Clases con demasiada responsabilidad
- Clases sin responsabilidad
- Clases con responsabilidades que no se usan
- Nombres engañosos
- Responsabilidades desconectadas
- Uso inadecuado de la herencia
- Funcionalidad repetida

La Herencia

Definición:

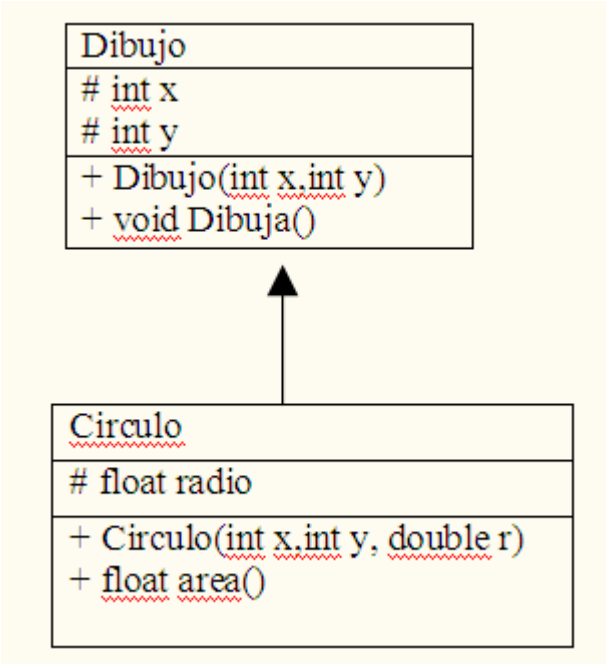
- **Reutilización y encapsulamiento: programación por responsabilidades, delegación.**
- **"Una clase A hereda de otra B cuando los objetos de A son una especialización de B (relación es-un)"**

Efectos básicos:

- **Una Subclase hereda los atributos y métodos de su padre**
- **Un método puede ser sobreescrito en una subclase cambiando sólo ese comportamiento**
- **Un método heredado puede ser llamado en una instancia de subclase y ejecutará el comportamiento heredado**

La Herencia

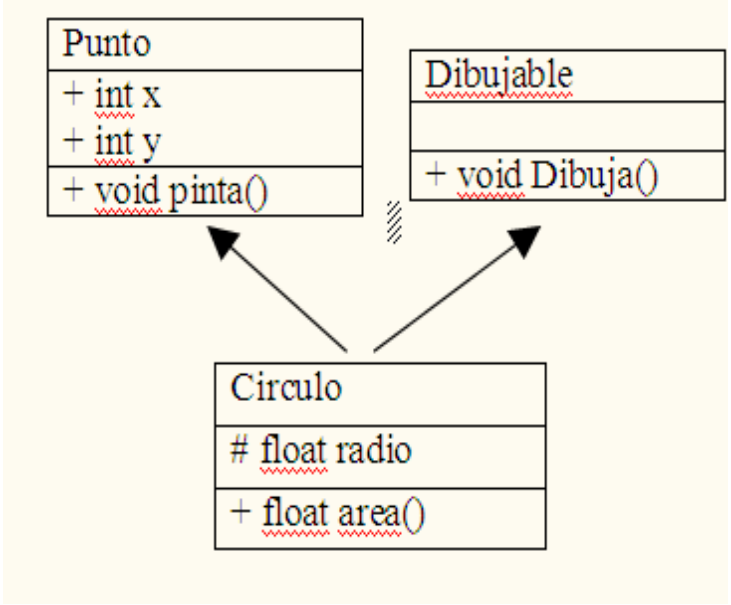
Dibujo establece las características básicas y comportamiento común
Circulo añade características y comportamiento asociado a estas características



- **Los atributos de Dibujo son accesibles desde Circulo**
- **Los atributos de Circulo y Dibujo no son accesibles desde otras clases que no estén por debajo de su herencia**

La Herencia múltiple

Dibujable y Punto establecen las características básicas y comportamiento común
Circulo añade características y comportamiento asociado a estas características



- **Los atributos de Dibujable y Punto son accesibles desde Circulo**
- **Los atributos de Circulo, Dibujable y Punto no son accesibles desde otras clases que no estén por debajo de su herencia**
- **Circulo puede utilizar métodos de cualquiera de sus ancestros:**

```
Circulo miCirculo = new Circulo();
miCirculo.pinta();
miCirculo.Dibuja();
miCirculo.area();
```

La Herencia múltiple vs Interfaces

Si no se tiene cuidado pueden establecerse ambigüedades con métodos y atributos heredados de más de un ancestro → Necesidad de especificar el ámbito del espacio de nombre (nombrar que clase específicamente contiene el método).

Otros lenguajes que no disponen de Herencia Múltiple implementan el uso de Interfaces: Similar a una clase completamente abstracta, sólo determina la obligación de establecer un conjunto de prototipos.

- **Provee generalización de clases por comportamiento**
- **Permiten a su vez heredar de otros interfaces**

Polimorfismo

“Varias formas” = distintos comportamientos dependiendo de las clases.
Principio de abstracción.
Amplía el concepto de procedimiento y suaviza la asignación de tipos fuerte.

Significa al menos:

1) Sobrecarga de operadores
2) Sobrecarga de funciones
3) Tipos genéricos o parametrización de tipos
4) Ampliación de las reglas de compatibilidad de tipos y de la asignación en lenguajes de tipo
5) Polimorfismo de mensajes

Polimorfismo

Contexto:
En todos los casos existe un contexto que determina el valor semántico exacto de cada identificador.

- **Estático**: (en tiempo de Compilación/Enlazado)
  Compilación/Enlazado
    Contexto = tipo definido en la declaración de objetos
    Contexto = tipo de los identificadores
    Contexto = tipo de operandos
    Contexto = número y tipo de los argumentos
  Comprobación de asignaciones
    comprobación de operaciones (funciones, operadores, métodos) implementados en las fuentes disponibles

- **Dinámico**: (en tiempo de ejecución)
    Contexto = clase a la que pertenece el objeto, se resuelve en el momento de la utilización (de método o de variable). Sólo es posible conocer la clase concreta en tiempo de ejecución.

Polimorfismo: Ejemplos

- Varias funciones con el mismo nombre en una clase y distinto número de parámetros/tipos.

- Sobrecarga de operadores en la que los operandos pueden ser de diferentes clases.

- Llamadas a métodos en una superclase no conocida en tiempo de compilación con diferentes implementaciones en las subclases que puede contener