CLASES Y OBJETOS

Programación Orientada a Objetos

La **Programación Orientada a Objetos (OOP)** es un **método** de programación en el cual los programas se organizan en colecciones cooperativas de **objetos**, cada uno de los cuales representa una ***instancia*** de alguna **clase**, y cuyas clases son, todas ellas, miembros de una **jerarquía de clases** unidas mediante **relaciones de herencia**.

Ventajas de la POO

- Proximidad de los conceptos modelados respecto a objetos del mundo real
- Facilita la reutilización de código
  - Y por tanto el mantenimiento del mismo
- Se pueden usar conceptos comunes durante las fases de análisis, diseño e implementación
- Disipa las barreras entre el *qué* y el *cómo*

Desventajas de la POO

- Mayor complejidad a la hora de entender el flujo de datos
  - Pérdida de linealidad
- Requiere de un lenguaje de modelización de problemas más elaborado:
  - *Unified Modelling Language* (UML)
  - Representaciones gráficas más complicadas

Conceptos de la OOP

Conceptos básicos

- Objeto
- Clase

Características de la OOP

- Abstracción:
- Encapsulamiento:
- Modularidad:
- Jerarquía

Otros conceptos OOP

- Tipos
- Persistencia

Tipos de relaciones

- Asociación
- Herencia
- Agregación
- Instanciación

Representaciones gráficas

- Diagramas estáticos (de clases, de objetos...)
- Diagramas dinámicos (de interacción...)

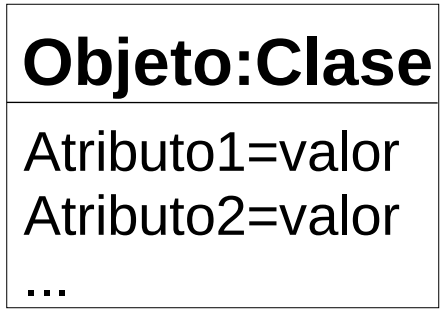
Objeto y Clase

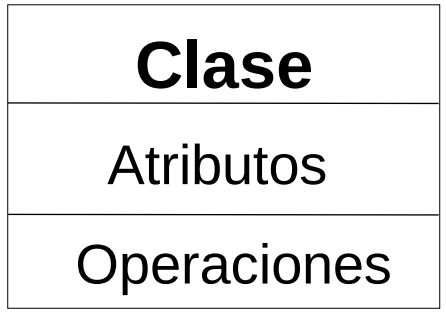
Un **objeto** es algo de lo que hablamos y que podemos manipular

- Existen en el mundo real (o en nuestro entendimiento del mismo)

Una **clase** describe los objetos del mismo tipo

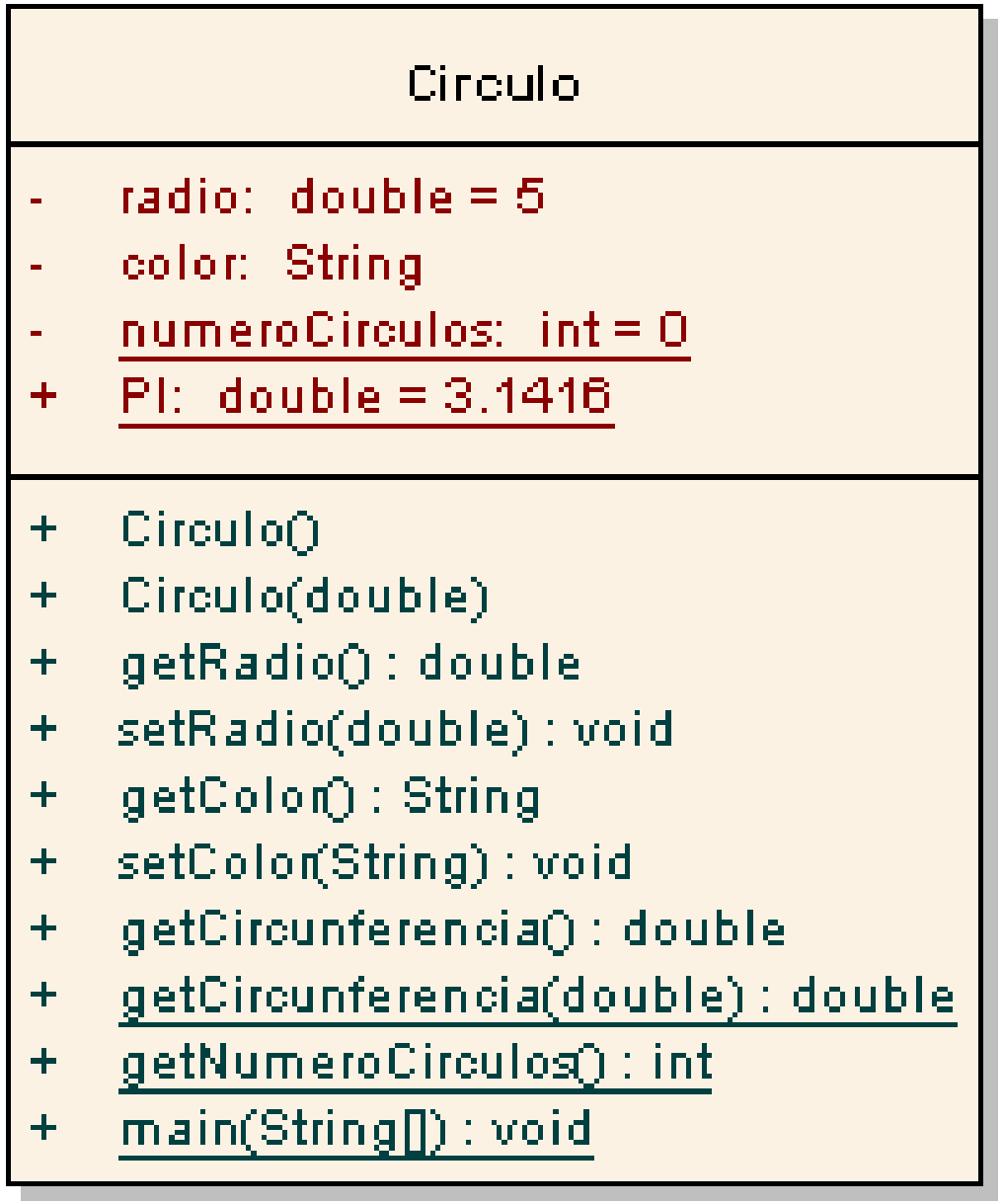
- Todos los objetos son instancias de una clase
- Describe las propiedades y el comportamiento de un tipo de objetos





Definición de una Clase

```
class Circulo {
    // campos
    // métodos
    // constructores
    // main()
}
```



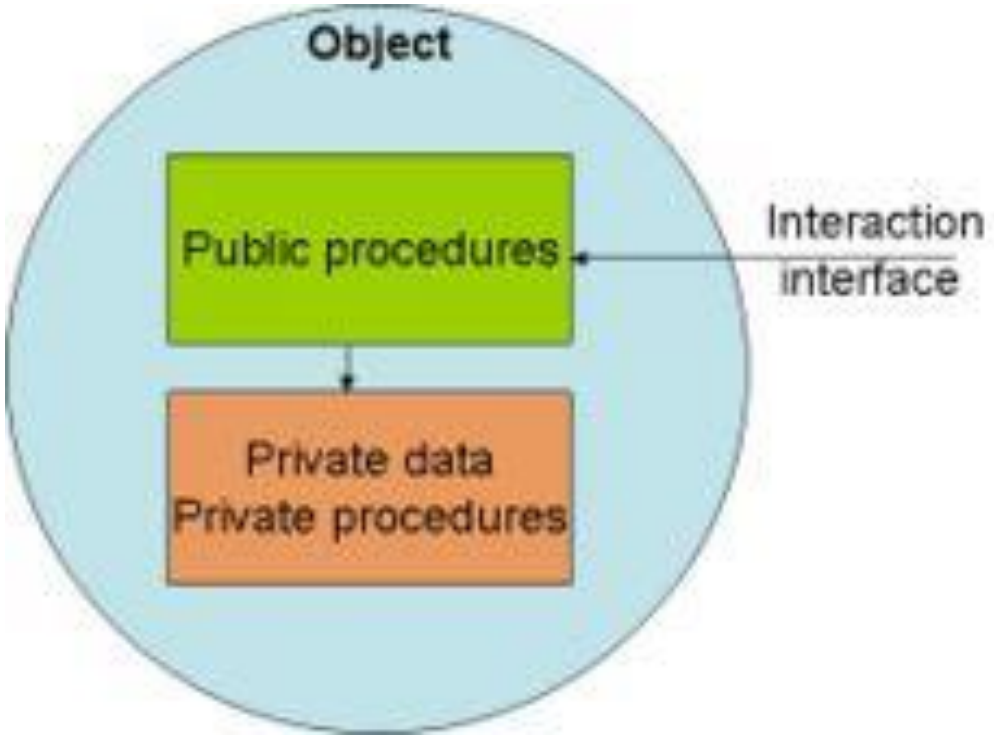
Conceptos OOP: Abstracción

- Nos permite trabajar con la complejidad del mundo real
  - Resaltando los aspectos relevantes de los objetos de una clase
  - Ocultando los detalles particulares de cada objeto
- Separaremos el comportamiento de la implementación
- Es más importante saber qué se hace en lugar de cómo se hace:

  Un sensor de temperatura
  - Se define porque...
    - mide la temperatura
    - nos muestra su valor
    - se puede calibrar...
  - No sabemos... (no nos importa)
    - cómo mide la temperatura
    - de qué está hecho
    - cómo se calibra

Encapsulamiento

- La clase es el espacio donde se empaquetan atributos y métodos



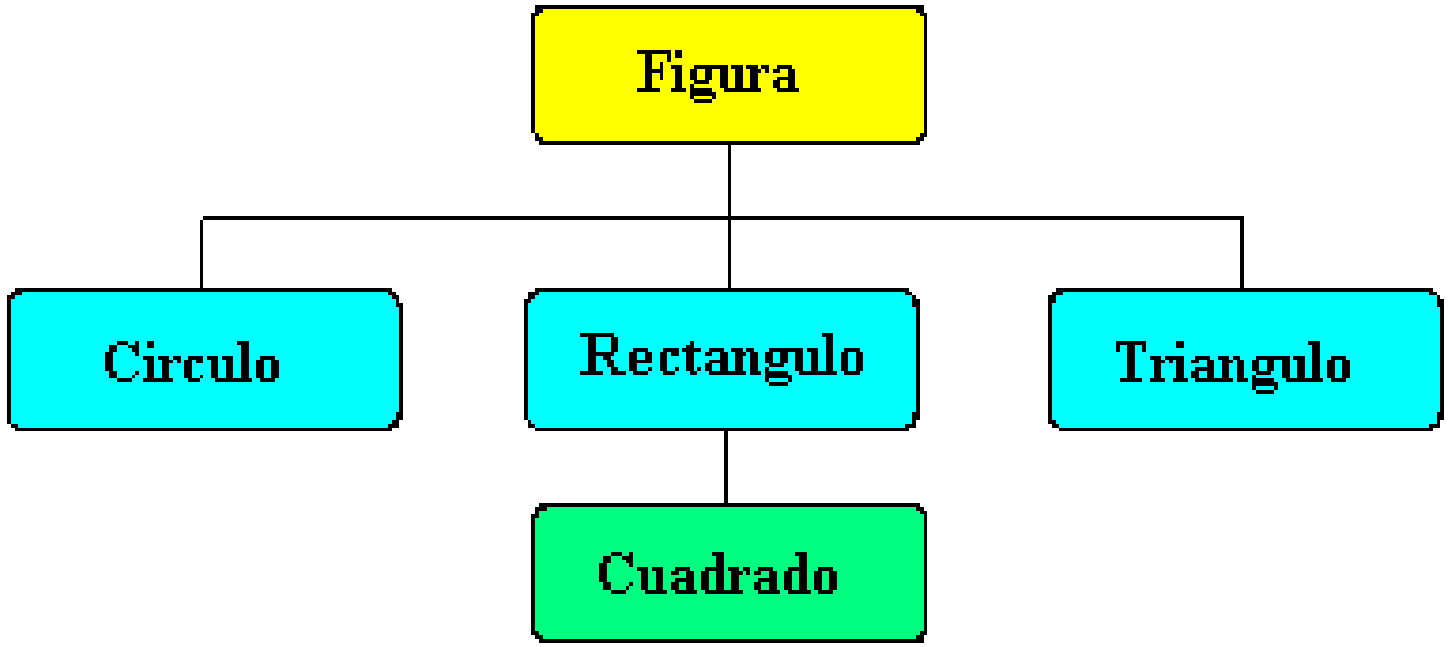
Conceptos OOP: Modularidad

- Consiste en separar el sistema en bloques poco ligados entre sí: módulos.
  - Organización del código
- Es una especie de encapsulamiento de más alto nivel.

Conceptos POO: Jerarquía

- Es una clasificación u ordenamiento de las abstracciones
- Hay dos jerarquías fundamentales:
  - Estructura de clases:
    - Jerarquía "*es un/a*"
    - Relaciones de herencia
  - Estructura de objetos:
    - Jerarquía "*parte de*"
    - Relaciones de agregación
    - Está implementada de manera genérica en la estructura de clases

Conceptos POO: Jerarquía



Conceptos OOP: Tipo

- Es el reforzamiento del concepto de clase
- Objetos de tipo diferente no pueden ser intercambiados
- El C++ y el Java son lenguajes fuertemente “tipeados”
- Ayuda a corregir errores en tiempo de compilación
  - Mejor que en tiempo de ejecución

Conceptos OOP: Persistencia

- Propiedad de un objeto de trascender en el tiempo y en el espacio a su creador (programa que lo generó)
- No se trata de almacenar sólo el estado de un objeto sino toda la clase (incluido su comportamiento)
- No está directamente soportado por el C++
  - Existen librerías y sistemas completos (OODBMS) que facilitan la tarea
  - Frameworks (entornos) como ROOT lo soportan parcialmente (reflex)
- El concepto de serialización del Java está directamente relacionado con la persistencia

Relaciones

- Están presentes en cualquier sistema
- Definen como se producen los intercambios de información y datos
- También ayudan a comprender las propiedades de unas clases a partir de las propiedades de otras
- Existen 4 tipos de relaciones:
  - Asociación
  - Herencia
  - Agregación
  - Instanciación

Relación de Asociación

- Relación más general
- Denota una dependencia semántica
- Es bidireccional
- Primer paso para determinar una relación más compleja

  Ejemplo: Relación entre un producto y una venta. Cualquier venta está asociada a un producto, pero no es, ni forma parte de, ni posee ningún producto… al menos en una primera aproximación.

- **Cardinalidad:** multiplicidad a cada lado
  - Uno a uno: Venta-Transacción
  - Uno a muchos: Producto-Venta
  - Muchos a muchos: Comprador-Vendedor

Relación de Herencia

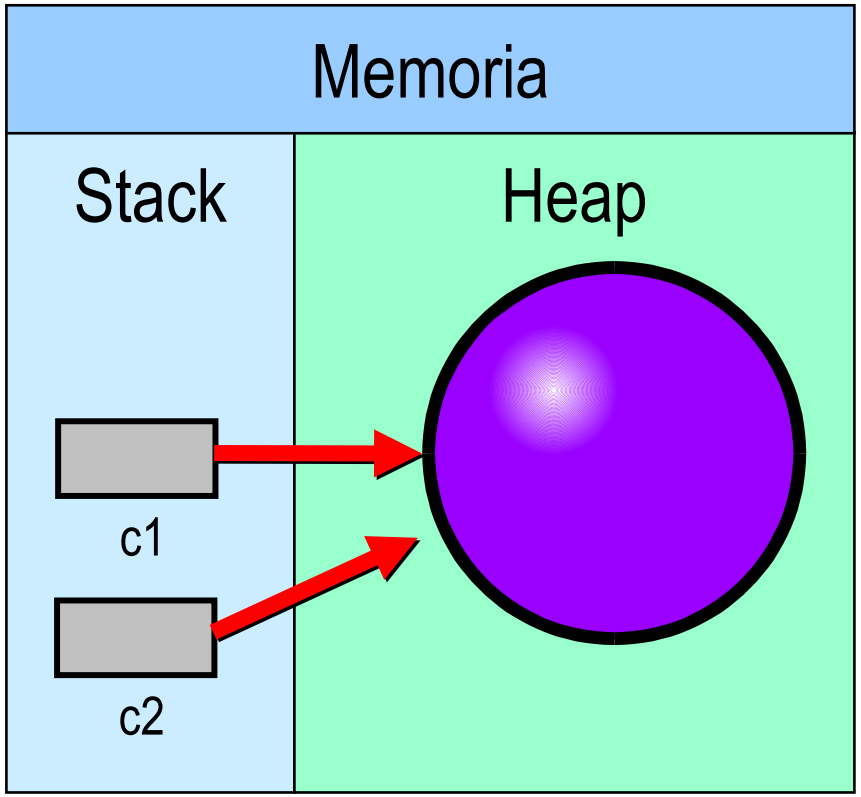
- ¡Relación característica de la OOP!
- Puede expresar tanto especialización como generalización
- Evita definir repetidas veces las características comunes a varias clases
- Una de las clases comparte la estructura y/o el comportamiento de otra(s) clase(s).
- También se denomina relación “*es un/a*” (*is a*)

Relación de Herencia (vocabulario)

- **Clase base o superclase**: clase de la cual se hereda
- **Clase derivada o subclase**: clase que hereda
- **Herencia simple**: Hereda de una sola clase
- **Herencia múltiple**: Hereda de varias clases
  - Java solo la soporta parcialmente
  - Presenta diversos problemas (¿qué hacer cuando se hereda más de una vez de la misma clase?)
- **Clase abstracta**: La que no lleva, ni puede llevar, ningún objeto asociado
- **Polimorfismo**: Posibilidad de usar indistintamente todos los objetos de un clase y derivadas.

Instanciación y Referencias

- Los objetos se crean con el operador new, y se manejan mediante referencias
- Los objetos se crean en el área de memoria dinámica conocida como el heap
- Una referencia contiene la dirección de un objeto (es similar a los *punteros* de otros lenguajes)
- Una asignación entre objetos es una asignación de referencias
  - Circulo c1 = new Circulo();
  - Circulo c2 = c1;



Paso de Parámetros

- En Java el paso de parámetros se realiza "por valor"
  - Argumentos de tipos primitivos
    - Si un método modifica el valor de un parámetro, este cambio sólo ocurre al interior del método; al retornar el método, se mantiene el valor original
  - Argumentos de tipo referencia (objetos)
    - Al retornar el método, la referencia pasada como parámetro sigue referenciando al mismo objeto; sin embargo, los campos del objeto podrían haber sido modificados por el método

Constructores

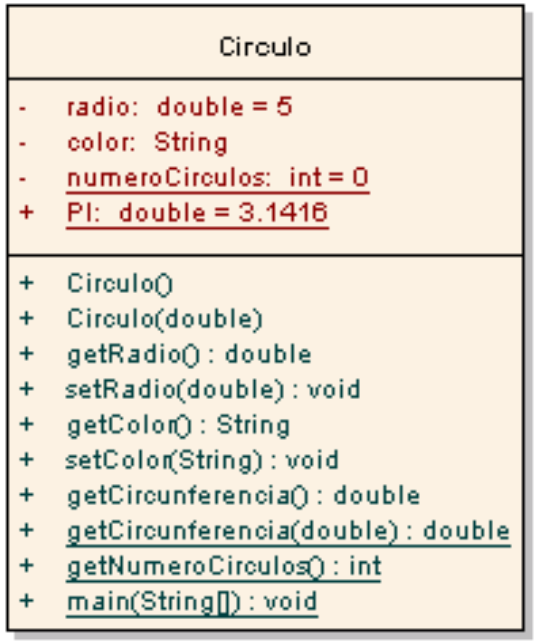
- Un constructor es un método especial invocado para instanciar e inicializar un objeto de una clase
  - Invocado con la sentencia new
  - Tiene el mismo nombre que la clase
  - Puede tener cero o más parámetros
  - No tiene tipo de retorno, ni siquiera void
  - Un constructor no público restringe el acceso a la creación de objetos
- Si la clase no tiene ningún constructor, el sistema provee un constructor default, sin parámetros
- Si la clase tiene algún constructor, debe usarse alguno de los constructores definidos al instanciar la clase (el sistema no provee un constructor default en este caso)

Ejemplo

```
class Circulo {
   ...

   // constructores
   public Circulo() {
      radio = 1;
   }
   public Circulo(double r) {
      radio = r;
   }

   void f() {
      Circulo c = new Circulo(30);
      ...
   }
}
```



Invocación entre Constructores

- La palabra this puede ser utilizada en la primera línea de un constructor para invocar a otro constructor

```
class Circulo {
    private double radio;
    private static int numeroCirculos = 0;
    Circulo(double radio) {
        this.radio = radio;
        Circulo.numeroCirculos++;
    }
    Circulo() {
        this(10);   // radio default: 10
    }
}
```