

---

---

# Introducción a la Programación Orientada a Objetos

— Tecnicatura Universitaria en Web —

---

---

# Introducción a la Programación Orientada a Objetos



- ***Simulación:***

- Creación del Lenguaje de Programación SIMULA-67. Años 60. Objetivo: desarrollar modelos del mundo real y sobre ellos ejecutar simulaciones.
- Objetos: Identidad, Estructura, Comportamiento, Interacción.
- SIMULA-67 introduce el concepto de Clase debido a la necesidad de crear muchos objetos con una misma estructura pero con Identidad diferente.
- Objetos: entidades reactivas capaces de responder requerimientos del exterior a través de la realización de operaciones sobre su estructura interna.

# Introducción a la Programación Orientada a Objetos



- ***Ingeniera de Software (I):***

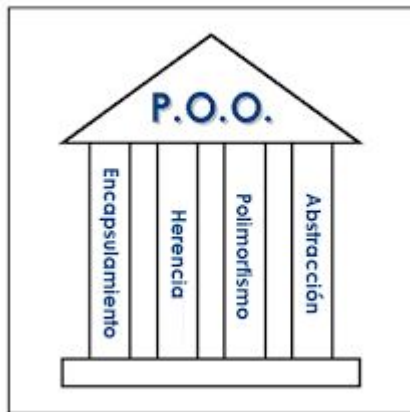
- Programación Estructurada. Refinamientos sucesivos.
- Reutilización de Software (Procedimientos y Funciones).
- Paso siguiente en los lenguajes para facilitar la reutilización: los Módulos.
- Información Oculta: datos locales a un módulo y procedimientos y funciones son servicios disponibles en el módulo para acceder a los datos desde el exterior. Módulos como abstracción de datos y no de control.

# Introducción a la Programación Orientada a Objetos



- ***Ingeniera de Software (II):***
  - Independencia del Contexto (Permite la reutilización).
  - Abstracción de Datos (Garantiza abstracción).
  - Encapsulamiento (Garantiza abstracción y protección).
  - Modularidad (Garantiza la composición de las partes).

# Introducción a la Programación Orientada a Objetos



- ***Significado de Orientado a Objetos:***

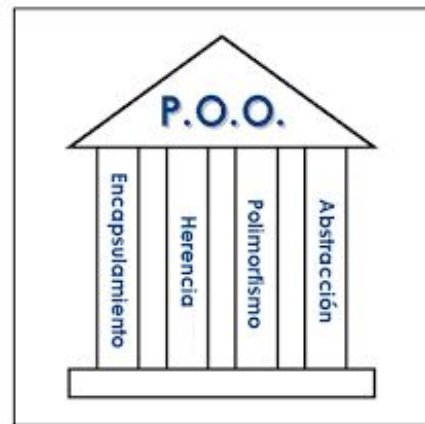
- Los seres humanos perciben el mundo como si estuviera formado por objetos: mesas, sillas, computadoras, coches, cuentas bancarias, etc.
- El significado de Orientado a Objetos nace como un conjunto de prácticas que definen un estilo de programación.

# Introducción a la Programación Orientada a Objetos

**Definición:** es un estilo de programación, donde todos los elementos que forman parte del problema se conciben como objetos, definiendo cuáles son sus atributos y comportamiento, cómo se relacionan entre sí y cómo están organizados.

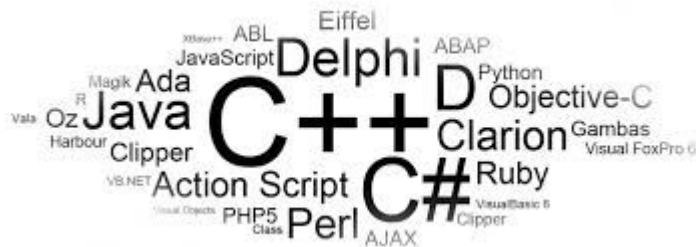
## **Estructura Interna de un Objeto:**

- *Atributos:* Define el estado del objeto
- *Métodos:* Define el comportamiento del objeto

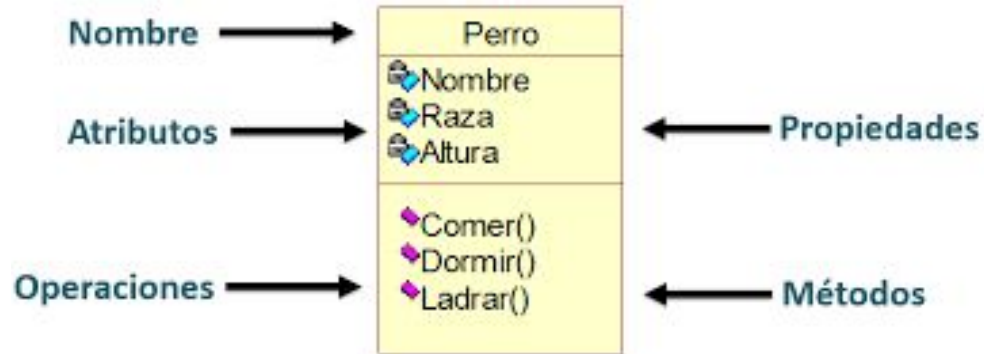


# Introducción a la Programación Orientada a Objetos

- Clase
  - Objeto
  - Atributos
  - Métodos
  - Instancia
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - Jerarquía
  - Generalización
- Herencia
  - Polimorfismo
  - Constructor
  - Destructor
  - Miembro Público
  - Miembro Privado
  - Miembro
  - Protegido



# Introducción a la Programación Orientada a Objetos



**Clase:** se puede definir como una descripción abstracta de un grupo de objetos, cada uno de los cuales tiene una serie de **atributos**, un **estado específico** y es capaz de realizar una serie de **operaciones**.



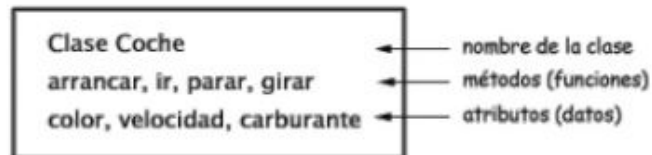
# Introducción a la Programación Orientada a Objetos

**Objeto:** es una instancia de una clase. La instancia de una clase significa definir un objeto dándole valores a sus atributos y comportamiento, y realizando operaciones permitidas por la clase.

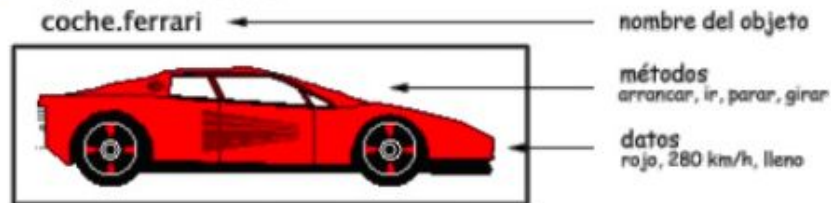
Un objeto tiene:

- Valores de los atributos
- Estado
- Identidad

**Clase:**  
*Coche*



♦ **Objeto:** *Ferrari*



# Introducción a la Programación Orientada a Objetos

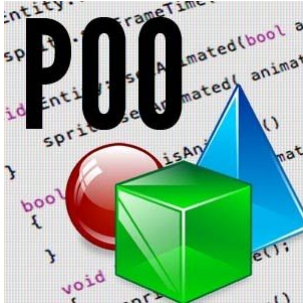


***Modularidad:*** es la propiedad que permite dividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.

# Introducción a la Programación Orientada a Objetos

## *Modularidad:*

- Los módulos definen estructuras de datos y operaciones.
- Las operaciones: son las únicas que pueden acceder a las estructuras de datos internas del módulo.
- No acceden a otras variables que sean externas al módulo.

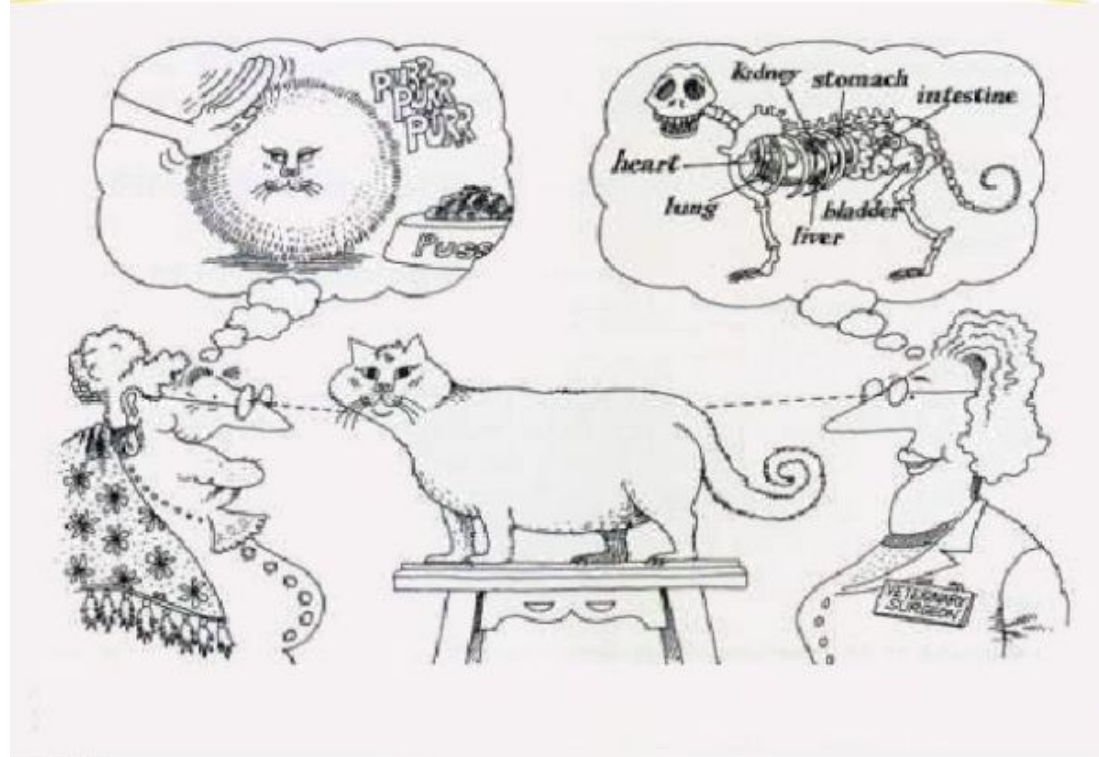


# Introducción a la Programación Orientada a Objetos

## ***Abstracción:***

- La abstracción es la capacidad que permite representar las características esenciales de un objeto sin preocuparse de las restantes características (no esenciales).
- Operación intelectual que ignora selectivamente partes de un todo para facilitar su comprensión.
- Surge del reconocimiento de similitudes (concentración de las similitudes y olvidarse de las diferencias).
- Barrera de abstracción: concentrarse en la separación del comportamiento de su implementación (Visión externa de un objeto).

# Introducción a la Programación Orientada a Objetos

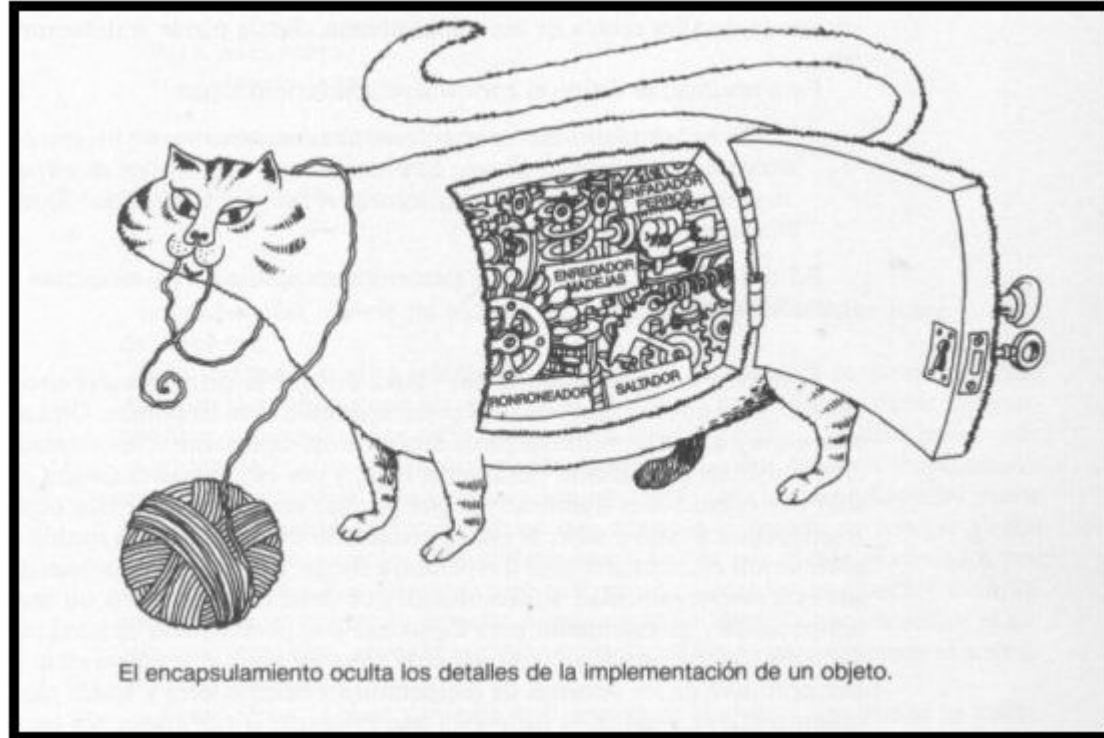


# Introducción a la Programación Orientada a Objetos

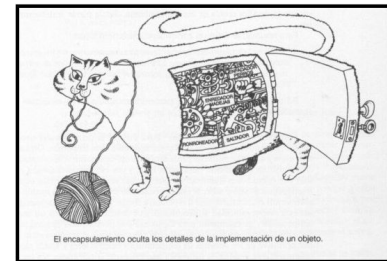
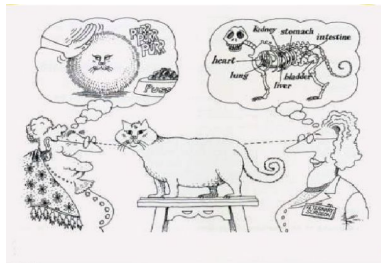
## *Encapsulación (Ocultamiento de la Información):*

- Es la propiedad que permite asegurar que los aspectos externos de un objeto se diferencien de sus detalles internos.
- Poder separar la interfaz de una clase de su implementación, o dicho en otras palabras: no es necesario conocer los detalles de cómo están implementadas las propiedades para poder utilizarlas. Los objetos funcionan a modo de caja negra en la que están empaquetados los datos y las instrucciones para su manipulación, de las que se conoce sólo lo necesario para utilizarla.

# Introducción a la Programación Orientada a Objetos



# Introducción a la Programación Orientada a Objetos



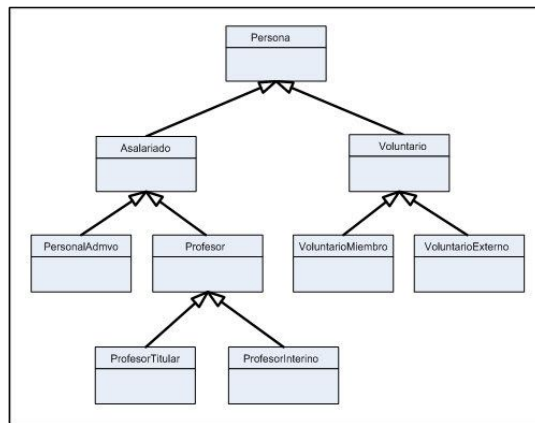
## *Abstracción de Datos y el Ocultamiento de la Información:*

- **Módulo**

- La entidad de un módulo que pasa a estar protegida y escondida y que idealmente debe ser inaccesible desde el exterior es la estructura de datos local al módulo.
- Las operaciones representan servicios disponibles por el módulo para que desde el exterior se pueda acceder a sus estructuras de datos.
- Los módulos con las características previamente mencionadas son vistos como mecanismos de abstracción de datos.



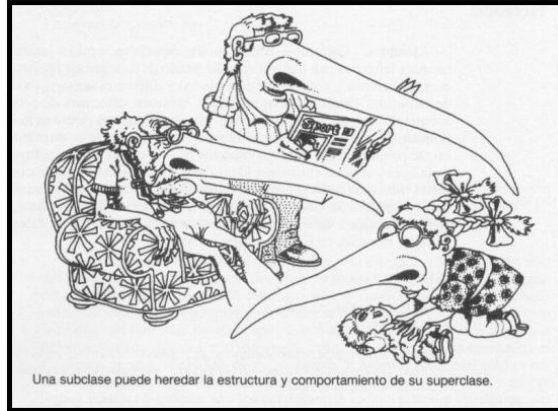
# Introducción a la Programación Orientada a Objetos



## ***Jerarquía:***

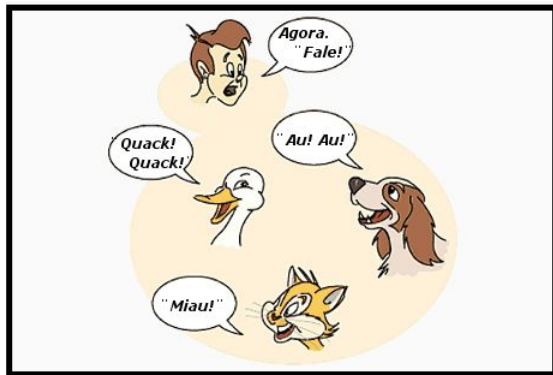
- Es una clasificación u ordenación de las abstracciones.
- Las dos jerarquías más importantes de un sistema complejo son:
  - Estructura de clases (jerarquía "es-un")
  - Estructura de objetos (jerarquía "parte de")

# Introducción a la Programación Orientada a Objetos



- **Generalización:** una clase que comparte atributos y métodos similares con otras clases se le llama superclase o clase padre. Cuando se define una clase padre se está generalizando.
- **Herencia:** del mismo modo, cuando se define una clase a partir de una clase padre se está creando una subclase.

# Introducción a la Programación Orientada a Objetos



- **Polimorfismo:** es el mecanismo de definir un mismo método en varios objetos de diferentes clases pero con distintas formas de implementación.
- **Constructor:** es un método que se invoca cuando un objeto es construido
- **Destructor:** es un método que se invoca cuando un objeto es destruido.

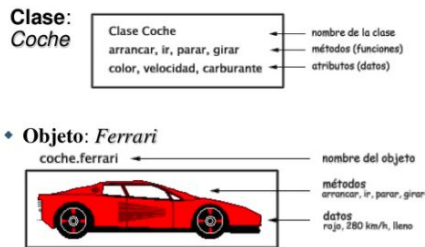
# Introducción a la Programación Orientada a Objetos

Visibilidad	Public	Private	Protected	Default*
Desde la misma clase	✓	✓	✓	✓
Desde una subclase	✓	✓	✓	✗
Desde otra clase (no subclase)	✓	✗	✗	✗

\*Con default, me refiero a omitir el modificador de acceso.

- **Miembro Público:** atributo o método de una clase que puede ser accedido desde cualquier parte del programa.
- **Miembro Privado:** atributo o método de una clase que puede ser accedido solo dentro de esa clase.
- **Miembro Protegido:** atributo o método de una clase que puede ser accedido desde esa clase y sus clases heredadas.

# Clases y Objetos



- Generalmente un programa utiliza muchos objetos durante su ejecución.
- Estos objetos intercambian mensajes para cumplir con la funcionalidad programada.
- Un objeto es entidad encapsulada, protegida y accesible a través de su interfaz y cuyo comportamiento definido es accesible por medio de un mecanismo de envío de mensajes.
- Las clases son un mecanismo que permite garantizar que todas sus instancias (objetos):
  - Tengan la misma estructura y el mismo comportamiento.
  - Accedan a sus datos por medio de operaciones destinadas para tal fin.

# Clases y Objetos

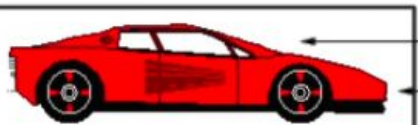
- Una clase es un molde para la creación de objetos o sea sus instancias.
- Una clase es una abstracción de entidades del mundo real que cumplen con las siguientes características:
  - Todas las entidades del mundo real tienen las mismas características.
  - Todas las entidades siguen las mismas reglas.



## Clase: *Coche*

Clase Coche	← nombre de la clase
arrancar, ir, parar, girar	← métodos (funciones)
color, velocidad, carburante	← atributos (datos)

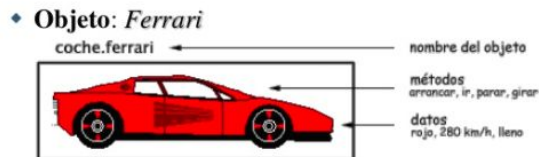
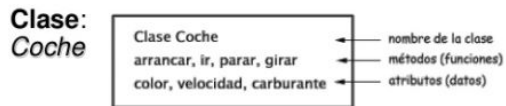
## ♦ Objeto: *Ferrari*

coche.ferrari	← nombre del objeto
	← métodos arrancar, ir, parar, girar
	← datos rojo, 280 km/h, lleno

# Clases y Objetos

## Importancia de las Clases:

- Las clases permiten la creación de objetos con una misma estructura (variables de instancia) y con un mismo comportamiento (métodos de instancia).
- Las clases pueden ser vistas como mecanismos para la implementación de tipos de datos abstractos.
- Las clases representan un mecanismo muy importante para la reutilización de código.



# Clases y Objetos

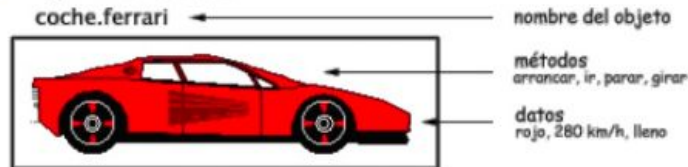
**Creación de Clases:** para la creación de una clase se debe especificar:

- La estructura (Variables de Instancia)
- El comportamiento (Métodos de Instancia)

**Clase:**  
*Coche*

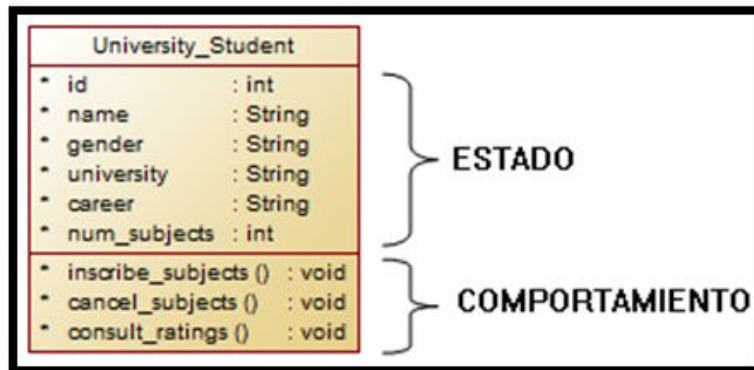


♦ **Objeto:** *Ferrari*





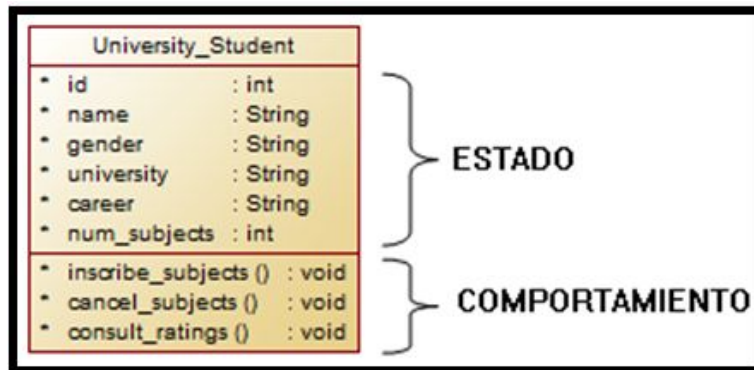
# Clases y Objetos



**Definición de la Estructura:** esta tarea se lleva a cabo definiendo los nombres y los tipos de las variables de instancia que serán utilizadas.

**Importante:** la estructura de una clase mantiene el estado de los objetos.

# Clases y Objetos



**Definición del Comportamiento:** para definir el comportamiento de una clase se deben especificar:

- Los Constructores.
- Los Métodos de Instancia.
- Los Destruyores.

# Clases - Definición del Comportamiento

**Constructores:** son métodos especiales que son utilizados en la creación de instancias de una clase.

- En Java, para cada clase C se le asocia un constructor C() capaz de crear instancias de la clase. Esto se lleva a cabo cuando dicho constructor se usa junto con la palabra reservada New en expresiones New C().
- Java permite que el programador defina tantos constructores como crea necesario. Solo se debe cumplir la siguiente regla:
  - Todos los constructores poseen el mismo nombre de la clase y solo difieren en el número y tipo de sus parámetros.



# Clases - Definición del Comportamiento

- Ejemplo: Constructores Clase Punto:
  - `Punto p1=New Punto();`
  - `Punto p1;`
  - `p1=New Punto();`
- En ambos casos se declara p1 de tipo punto. p1 es capaz de referenciar un objeto de tipo punto. Luego la variable se asocia a la ejecución de una expresión `New Punto()`, la cual crea una instancia de Punto por la ejecución del método especial `Punto()` (El constructor).



# Clases - Definición del Comportamiento

**Ejemplo:** Constructores Clase Punto:

- `Punto p1=new Punto();`
- `p1` es una instancia de `Punto`.
- `p1` tiene sus variables de instancia `x` e `y` inicializadas en 0 por ser de tipo `int`.



# Clases - Definición del Comportamiento

## ***Constructores:***

- Llevan el mismo nombre de la clase.
- Una clase puede tener más de un constructor.
- En el caso de haber más de un constructor los mismos se diferencian por el número de argumentos.



## ***Función de los Constructores:***

- La función principal de un constructor es la creación de instancias de un clase y por lo tanto no tiene sentido especificar el tipo de resultado.

# Clases - Definición del Comportamiento

*Constructores: Declaración*

*nombreDeLaClase ( listaDeParámetros ) {*

*Sentencias*

*}*



# Clases - Definición del Comportamiento

## ***Constructores: Declaración***

Es siempre posible y generalmente muy útil definir más de un constructor. En este caso, generalmente los constructores difieren en los valores asignados a las variables de instancia, o también en la forma de asignar esos valores.





# Clases - Definición del Comportamiento

## *Constructores: Declaración*

```
class Punto {  
    //Variables de Instancia  
  
    double x;  
    double y;  
  
    // Constructores  
  
    Punto() { x = 0; y = 0; }  
  
    Punto(double cx, double cy) {x = cx; y =cy;}  
  
    ...  
}
```



# Clases - Definición del Comportamiento

**Regla:** la única forma de garantizar que se programan entidades independiente del contexto y por lo tanto reutilizables es garantizando que ninguna clase accede directamente a las variables de instancia de otra clase. Para modificar estas variables se invoca a través del envío de mensajes a métodos de acceso a tales variables. Dichos métodos deben ser programados en las clases.

```
public class CuentaBancaria {  
    //Declaración de Atributos  
    private String numeroCuenta;  
    private String nombreCliente;  
    private float interesAnual;  
    private float saldo;  
  
    /**...*/  
    → public String getNumeroCuenta() {  
        return numeroCuenta;  
    }  
    /**...*/  
    → public void setNumeroCuenta(String numeroCuenta) {  
        this.numeroCuenta = numeroCuenta;  
    }  
}
```

# Clases - Definición del Comportamiento

## *Sugerencia:*

Es altamente recomendado utilizar métodos que permitan:

- Acceder al valor de las variables de instancia. Estos métodos son conocidos con el nombre de ***observadores***.
- Modificar el valor de las variables de instancia. Estos métodos son conocidos con el nombre de ***modificadores***.

```
public class CuentaBancaria {  
    //Declaración de Atributos  
    private String numeroCuenta;  
    private String nombreCliente;  
    private float interesAnual;  
    private float saldo;  
  
    /**...*/  
    → public String getNumeroCuenta() {  
        return numeroCuenta;  
    }  
  
    /**...*/  
    → public void setNumeroCuenta(String numeroCuenta) {  
        this.numeroCuenta = numeroCuenta;  
    }  
}
```

# Clases - Definición del Comportamiento



## *Métodos de Instancia - Sintaxis*

*<Mod. Acc> <Tipo de Resultado> Ident. (Tipo Par- 1 .. Tipo Par-N) Cuerpo*

- **Modificador de Acceso:** se estudiará luego.
- **Tipo de Resultado:** es un tipo simple o el nombre de una clase.
- **Identificador:** es una secuencia de letras y dígitos debiendo la primera ser una letra.
- **Lista de Parámetros Formales:** está constituida por una lista de cero o más pares «Tipo Par» que definen los tipos e identificadores de los parámetros formales del método.

# Clases - Definición del Comportamiento

- **Métodos de Instancia - Cuerpo:** consiste de un grupo de sentencias delimitados por llaves: { sentencias }
- **Importante:** si el método devuelve un resultado este será el valor resultante de la expresión que está asociada a la primera sentencia return que sea ejecutada.



# Clases - Definición del Comportamiento

## *Métodos de Instancia - Observadores/Modificadores*

// Observadores

```
double getX() { return x;}
```

```
double getY() { return y;}
```

// Modificadores

```
void incCoord(double deltaX, double deltaY) {
```

```
    x = x + deltaX; y = y + deltaY ;
```

```
}
```



# Clases - Definición del Comportamiento

```
class Estudiante {  
    private:  
        static int numero_estudiantes = 0;  
        char* nombre;  
        int edad;  
        char* direccion;  
        char* carrera;  
    public:  
        Estudiante();  
        void estudiar();  
        void almorzar();  
        ~Estudiante();  
};
```

```
Estudiante::Estudiante() {  
    numero_estudiantes++;  
}  
  
Estudiante::~~Estudiante() {  
    numero_estudiantes--;  
}
```

- ***Destructores:*** se utiliza cuando se necesitan realizar acciones de limpieza cuando se destruye el objeto.
- ***Importante:*** en Java no se definen destructores. Esta sección de la clase es sólo a nivel informativo.

# Clases - Definición del Comportamiento

```
class Estudiante {  
    private:  
        static int numero_estudiantes = 0;  
        char* nombre;  
        int edad;  
        char* direccion;  
        char* carrera;  
    public:  
        Estudiante();  
        void estudiar();  
        void almorzar();  
        ~Estudiante();  
};
```

```
Estudiante::Estudiante() {  
    numero_estudiantes++;  
}  
  
Estudiante::~~Estudiante() {  
    numero_estudiantes--;  
}
```

## ***Destruyores – Sintaxis***

### ***símbolo identificador (Lista de Parámetros Formales) Cuerpo***

**Destruyores:** un destructor siempre lleva el nombre de la clase pero se distingue del constructor porque va precedido por algún símbolo especial. En el caso de C++ dicho símbolo es: ~.



# Clases - Definición del Comportamiento

```
class Estudiante {  
    private:  
        static int numero_estudiantes = 0;  
        char* nombre;  
        int edad;  
        char* direccion;  
        char* carrera;  
    public:  
        Estudiante();  
        void estudiar();  
        void almorzar();  
        ~Estudiante();  
};
```

```
Estudiante::Estudiante() {  
    numero_estudiantes++;  
}  
  
Estudiante::~~Estudiante() {  
    numero_estudiantes--;  
}
```

## ***Destruyores:***

Los destructores son útiles en clases que usan punteros a bloques de memoria ya que cuando el objeto se destruye se debe liberar el bloque de memoria. Un destructor puede realizar esa tarea a la perfección.