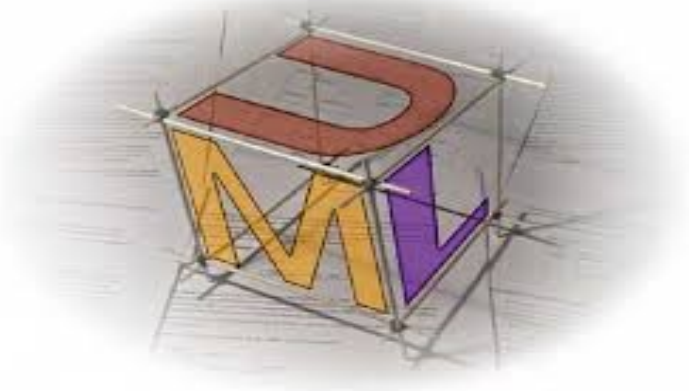


Ingeniería de Software y Programación Orientada a Objetos



Profesor: A.S. Carlos P. Ríos
lawlercarlospatricio@gmail.com



Mapa de la presentación

Ingeniería del Software

Ciclo de vida del Software

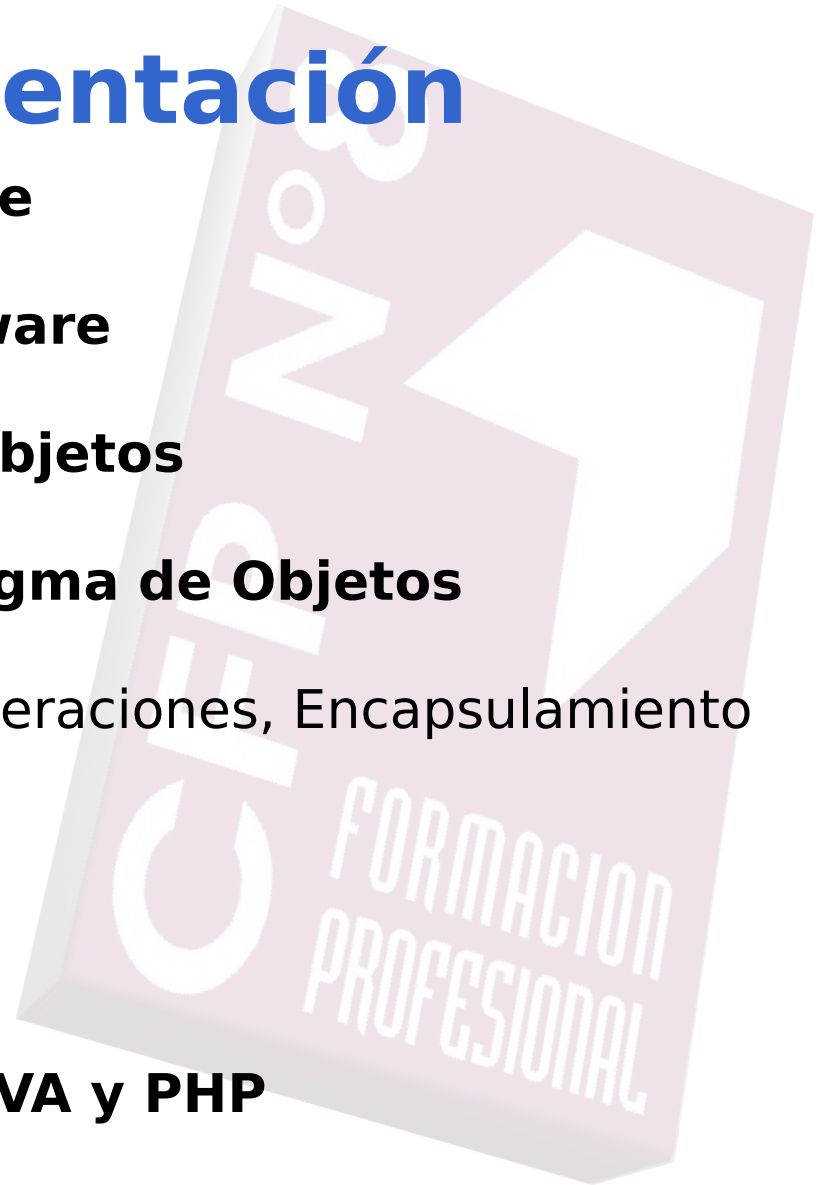
Análisis Orientado a Objetos

Introducción al Paradigma de Objetos

- Clases y Objetos
- Atributos, Tipos, Operaciones, Encapsulamiento
- Constructores
- Herencia

Introducción a UML

Ejemplos de Código JAVA y PHP

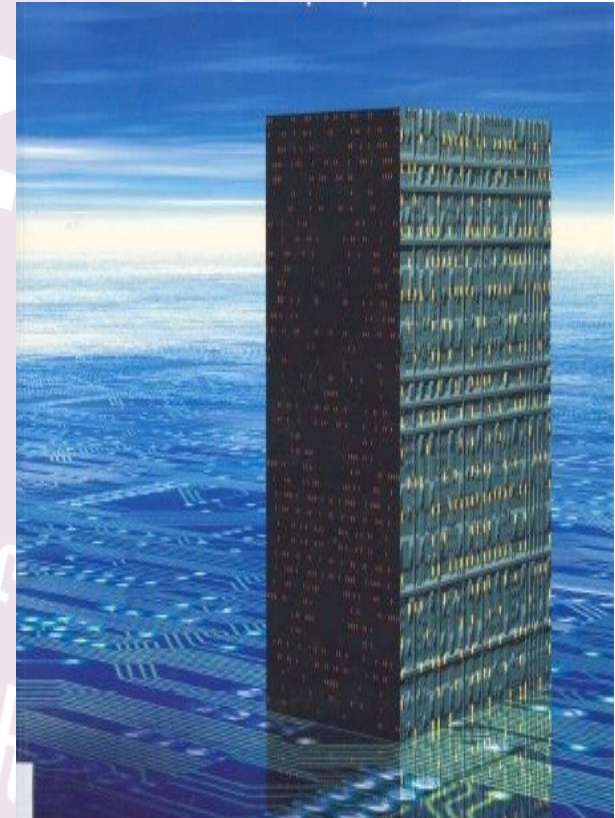


Ingeniería del Software

Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.

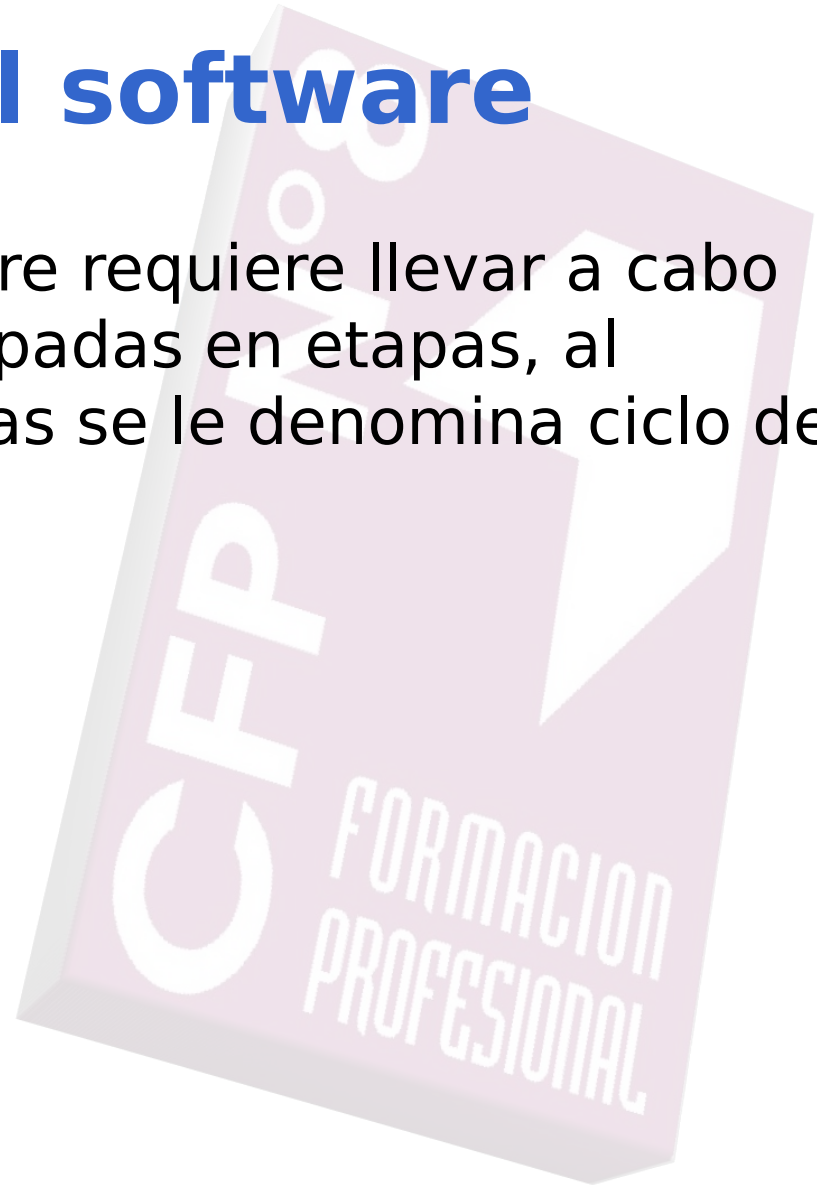
La ingeniería al software integra matemáticas, ciencias de la computación y prácticas cuyos orígenes se encuentran en la ingeniería.

El término "ingeniero de software", se utiliza en forma genérica en el ambiente empresarial, y no todos los ingenieros de software poseen realmente títulos de ingeniería de universidades reconocidas.



Ciclo de vida del software

La ingeniería de software requiere llevar a cabo numerosas tareas agrupadas en etapas, al conjunto de estas etapas se le denomina ciclo de vida.



Las etapas comunes a casi todos los modelos del ciclo de vida son:

- Análisis de requerimientos.
- Especificación de requerimientos.
- Arquitectura de software y datos.
- Desarrollo.
- Prueba.
- Documentación.
- Implementación.
- Mantenimiento.



Análisis de requerimientos

Extraer los requisitos de un producto de software es la primera etapa para crearlo. Mientras que los clientes piensan que ellos saben lo que el software tiene que hacer, se requiere habilidad y experiencia para reconocer requisitos incompletos, ambiguos o contradictorios. El resultado del análisis de requisitos con el cliente se plasma en el documento ERS, Especificación de Requisitos del Sistema, cuya estructura puede venir definida por varios estándares. La captura, análisis y especificación de requisitos (incluso pruebas de ellos), es una parte crucial; de esta etapa depende en gran medida el logro de los objetivos finales.

No siempre en la etapa de "análisis de requisitos" las distintas metodologías de desarrollo llevan asociado un estudio de viabilidad y/o estimación de costes.

Especificación de requerimientos

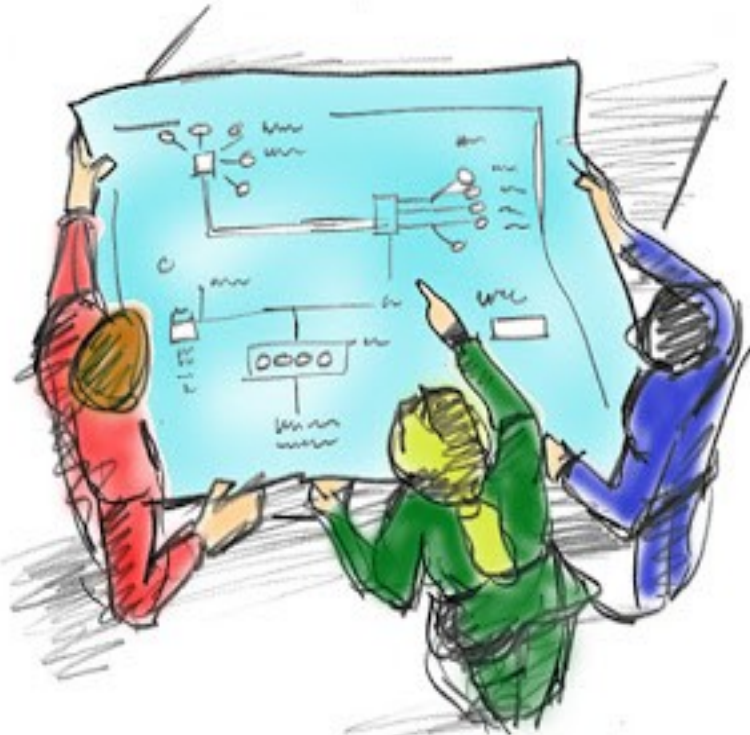
La especificación de requisitos describe el comportamiento esperado en el software una vez desarrollado. Gran parte del éxito de un proyecto de software radicará en la identificación de las necesidades del negocio (definidas por la alta dirección), así como la interacción con los usuarios funcionales para la recolección, clasificación, identificación, priorización y especificación de los requisitos del software.

Entre las técnicas utilizadas para la especificación de requisitos se encuentran:

- Caso de uso.
- Historias de usuario.

Siendo los primeros más rigurosas y formales, los segundos más ágiles e informales.

Arquitectura de software y datos



Arquitectura de software y datos

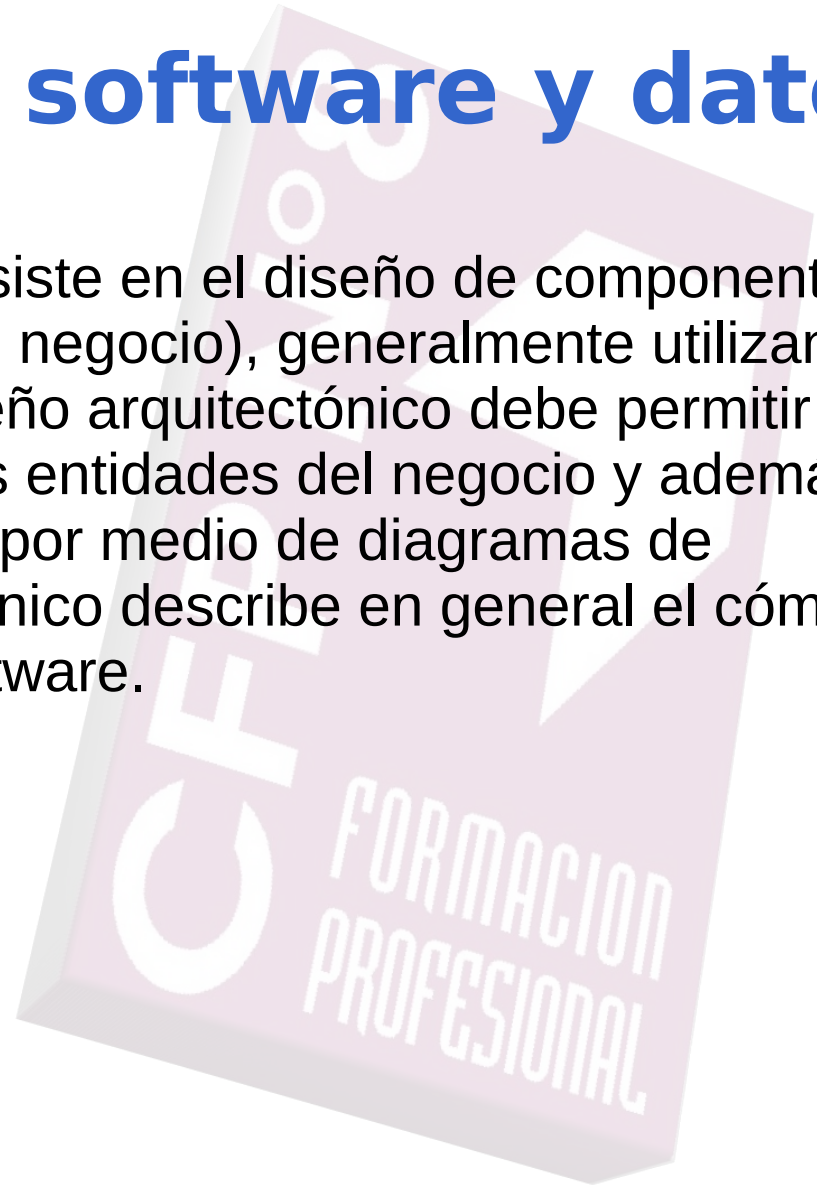
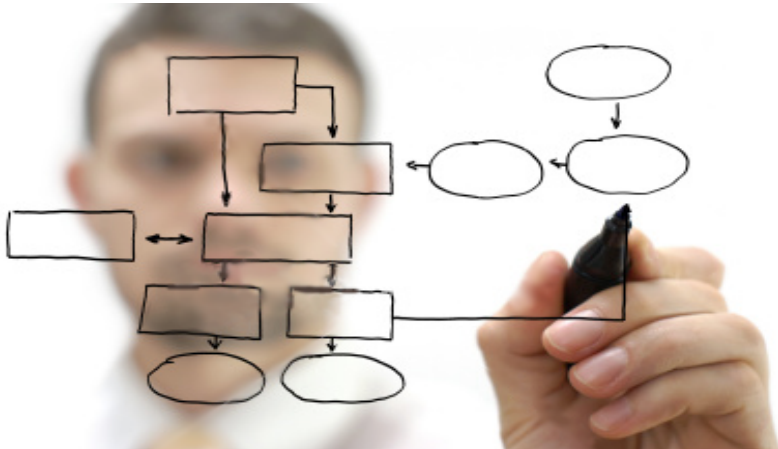
La integración de infraestructura, desarrollo de aplicaciones, bases de datos y herramientas gerenciales, requieren de capacidad y liderazgo para poder ser conceptualizados y proyectados a futuro, solucionando los problemas de hoy. El rol en el cual se delegan todas estas actividades es el del Arquitecto.

El arquitecto de software es la persona que añade valor a los procesos de negocios gracias a su valioso aporte de soluciones tecnológicas.

La arquitectura de sistemas en general, es una actividad de planeación, ya sea a nivel de infraestructura de red y hardware, o de software.

Arquitectura de software y datos

La arquitectura de software consiste en el diseño de componentes de una aplicación (entidades del negocio), generalmente utilizando patrones de arquitectura. El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del negocio y además poder ser validado, por ejemplo por medio de diagramas de secuencia. Un diseño arquitectónico describe en general el cómo se construirá una aplicación de software.



Arquitectura de software y datos

Para ello se documenta utilizando diagramas, por ejemplo:

- Diagramas de clases
- Diagramas de base de datos
- Diagrama de despliegue
- Diagrama de secuencia

Siendo los dos primeros los mínimos necesarios para describir la arquitectura de un proyecto que iniciará a ser codificado. Depende del alcance del proyecto, complejidad y necesidades, el arquitecto elige qué diagramas elaborar.

Las herramientas para el diseño y modelado de software se denominan CASE, (Computer Aided Software Engineering) entre las cuales se encuentran:

Enterprise Architect

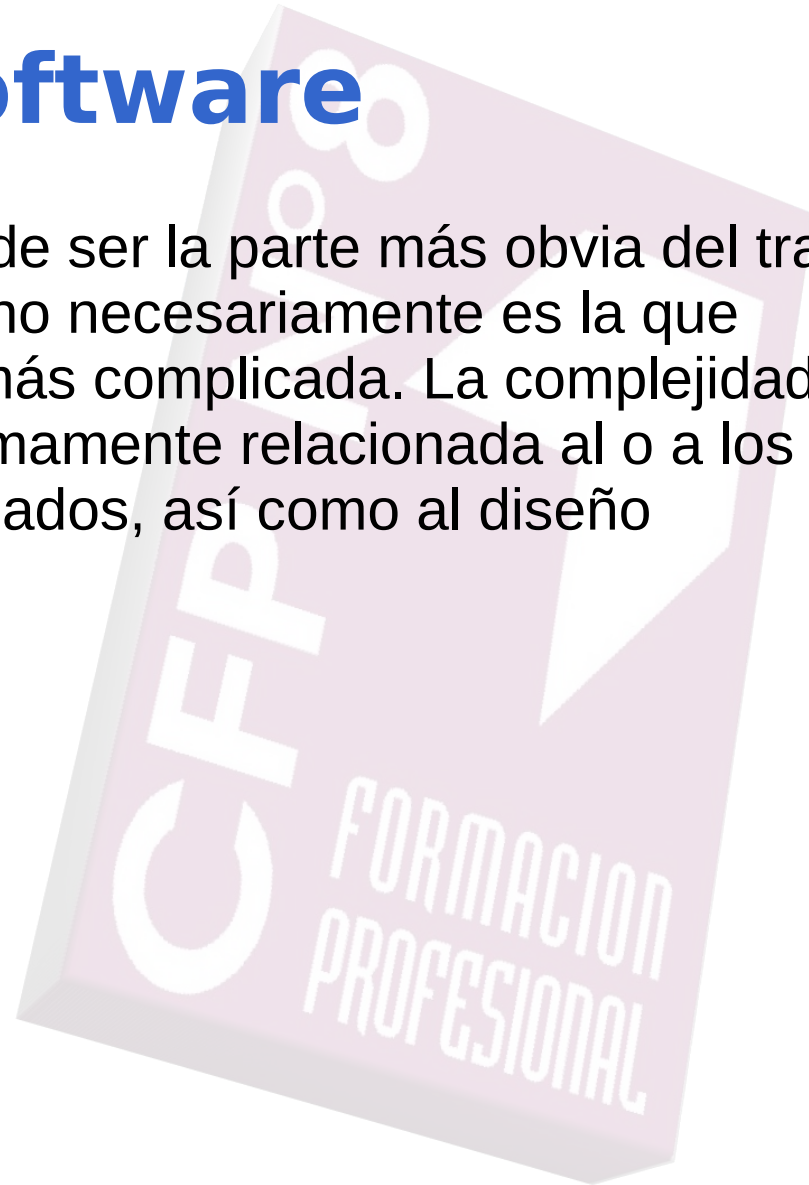
Microsoft Visio for Enterprise Architects

Desarrollo de software



Desarrollo de software

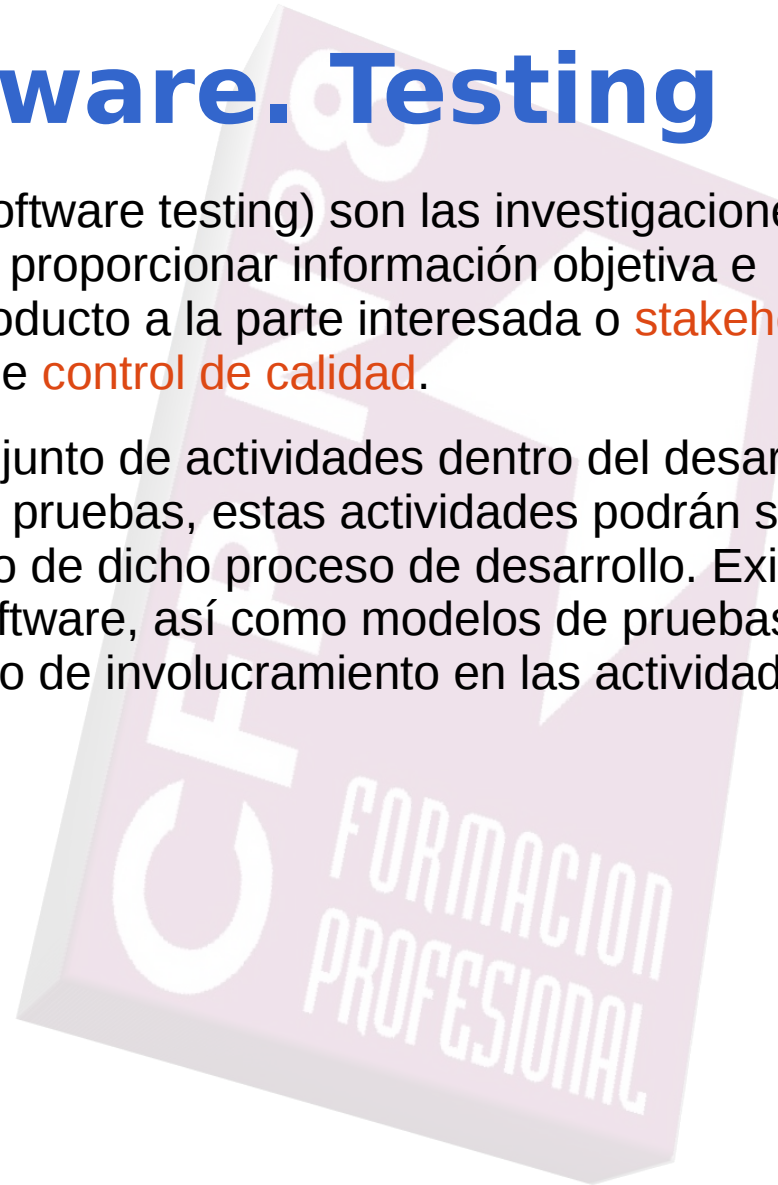
Reducir un diseño a código puede ser la parte más obvia del trabajo de ingeniería de software, pero no necesariamente es la que demanda mayor trabajo y ni la más complicada. La complejidad y la duración de esta etapa está íntimamente relacionada al o a los lenguajes de programación utilizados, así como al diseño previamente realizado.



Pruebas de Software. Testing

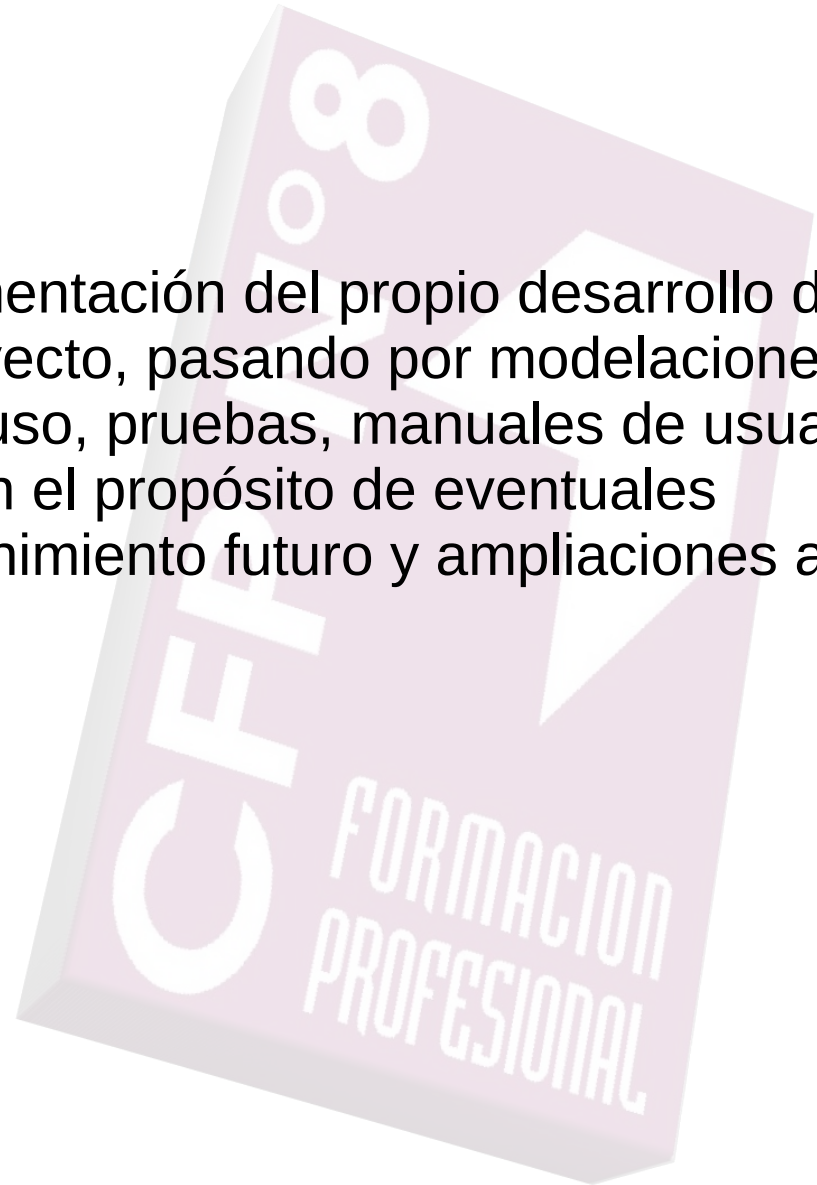
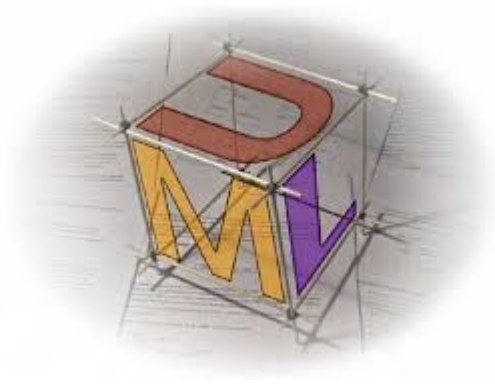
- Las pruebas de software (en **inglés** software testing) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o **stakeholder**. Es una actividad más en el proceso de **control de calidad**.

Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de **software**. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno corresponde un nivel distinto de involucramiento en las actividades de desarrollo.



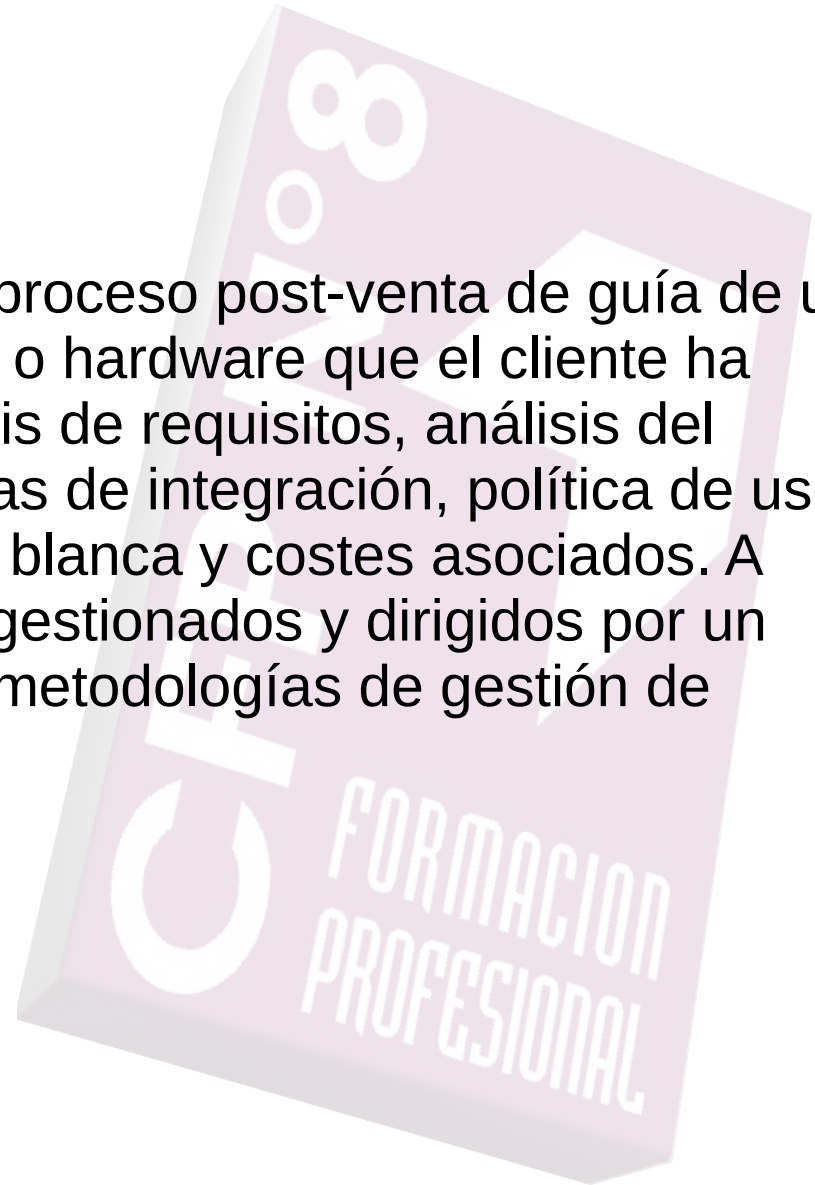
Documentación

Todo lo concerniente a la documentación del propio desarrollo del software y de la gestión del proyecto, pasando por modelaciones (UML), diagramas de casos de uso, pruebas, manuales de usuario, manuales técnicos, etc; todo con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema.



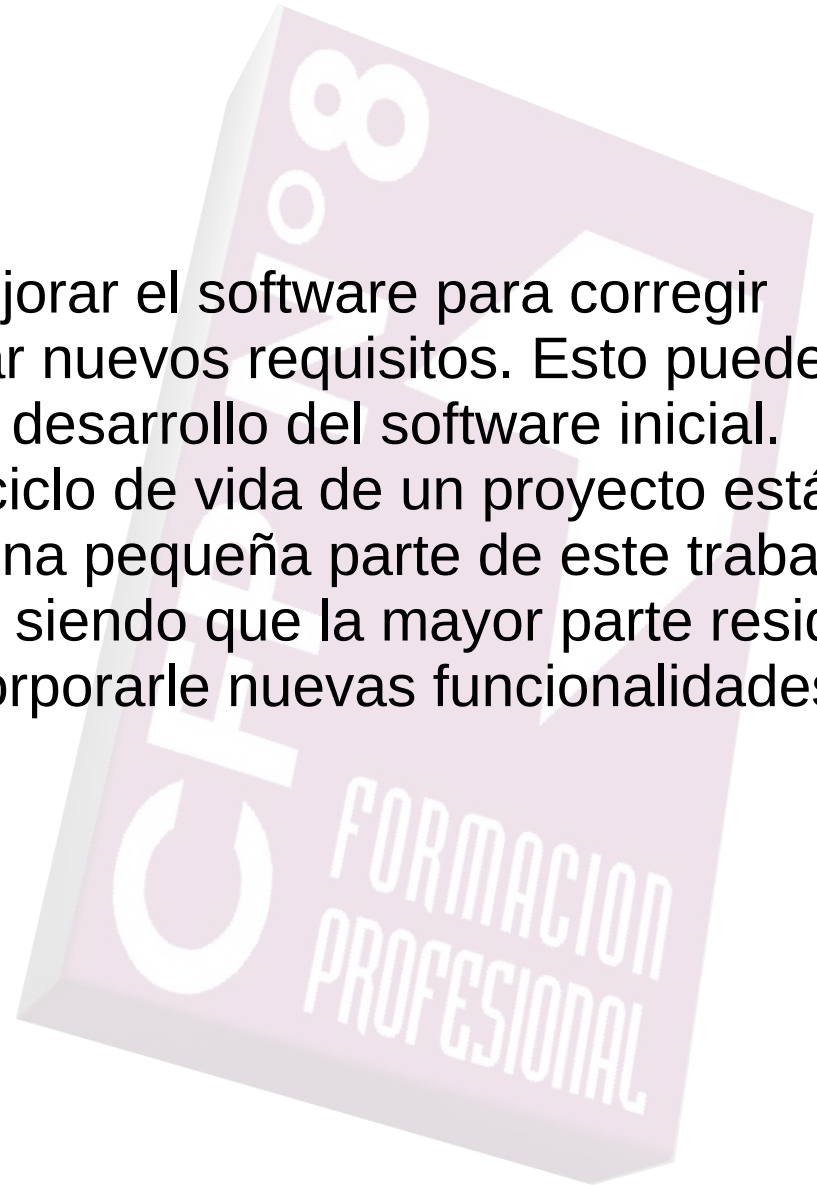
Implementación

La implementación se refiere al proceso post-venta de guía de un cliente sobre el uso del software o hardware que el cliente ha comprado. Esto incluye el análisis de requisitos, análisis del impacto, optimizaciones, sistemas de integración, política de uso, aprendizaje del usuario, marcha blanca y costes asociados. A menudo todos estos pasos son gestionados y dirigidos por un Director de Proyecto que utiliza metodologías de gestión de proyecto.



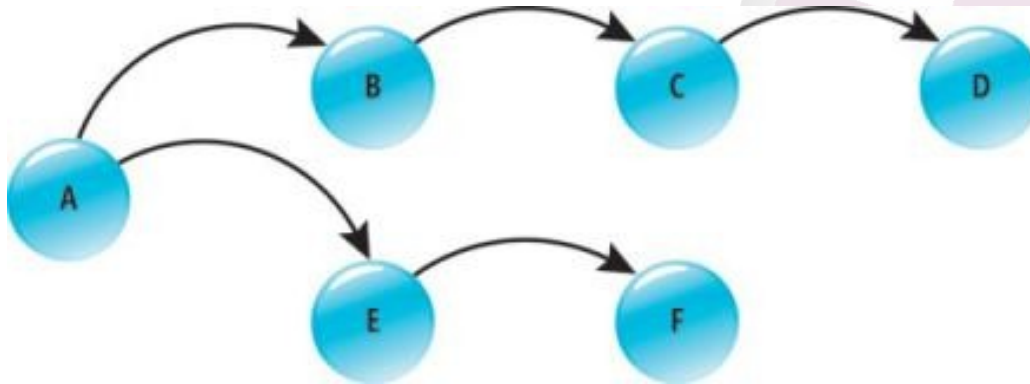
Mantenimiento

Fase dedicada a mantener y mejorar el software para corregir errores descubiertos e incorporar nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo del software inicial. Alrededor de 2/3 del tiempo de ciclo de vida de un proyecto está dedicado a su mantenimiento. Una pequeña parte de este trabajo consiste eliminar errores (bugs); siendo que la mayor parte reside en extender el sistema para incorporarle nuevas funcionalidades y hacer frente a su evolución.



Análisis Orientado a Objetos

Consiste en interpretar un sistema como partes independientes que se comunican entre sí

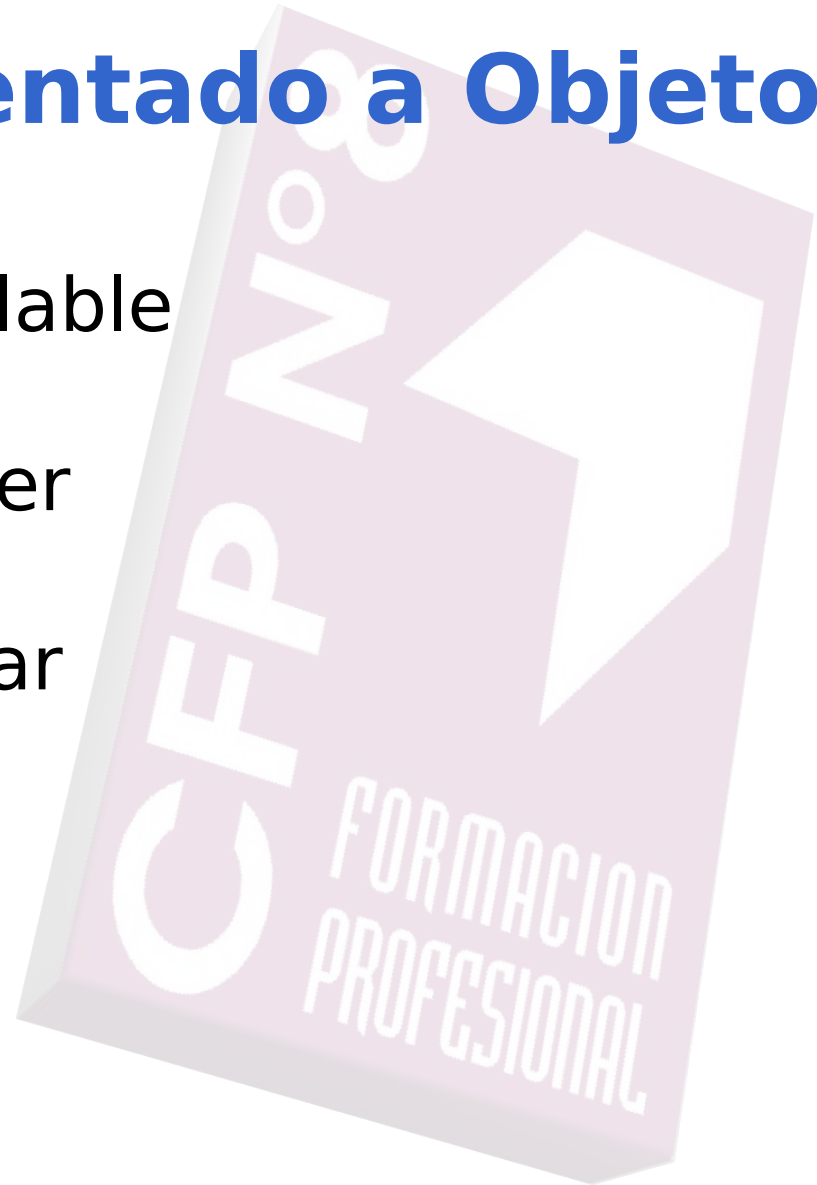


Las partes independientes se denominan **Objetos**

La comunicación entre los objetos se realiza a través de **Mensajes**

El Software Orientado a Objetos

- ✓ Altamente Escalable
- ✓ Fácil de Mantener
- ✓ Fácil de Reutilizar
- ✓ Muy Simple



Que es una Clase

Es una plantilla, es un molde que permite construir objetos

Representa ideas del mundo real, en forma genérica

Dentro de un sistema, las clases suelen detectarse como **sustantivos en singular**

Poseen atributos y métodos

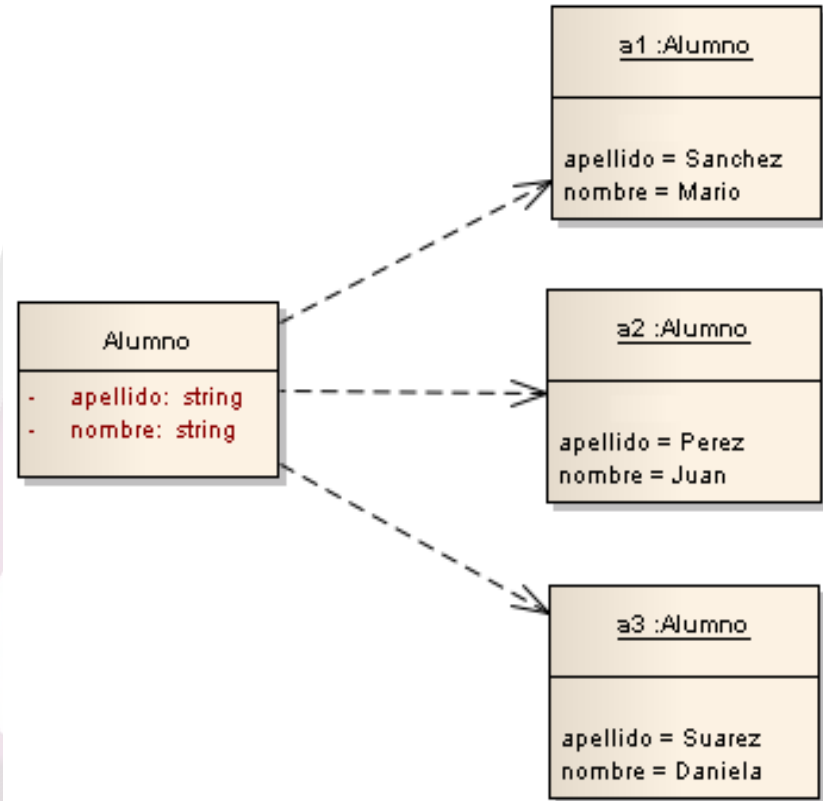
Ejemplos de clases: Auto, Empleado, CajaDeAhorro, Alumno

Que es un Objeto

Un objeto es una la **instancia de una clase**, podemos decir que el objeto representa algo en particular

Poseen un **estado** (de acuerdo a sus atributos)

Poseen un **comportamiento** (realizan operaciones de acuerdo a sus métodos)





Clases - Codificación

```
class Banco {  
  
    // Atributos aquí  
  
    // Métodos aquí  
  
}
```



Que son los Atributos

Automovil
color
modelo
precio
usado

Son **características** que posee una clase

Son variables contenidas y establecidas por los objetos, y normalmente cuentan con un tipo de dato asociado

Las atributos de una clase definen las características de sus objetos

Las clases definen los atributos, y los objetos “los completan”

Que es un Tipo de Dato

Automovil
color: string modelo: string precio: int usado: boolean

Es la forma de describir y/o almacenar un dato

Los tipos de datos numéricos mas conocidos son: **int**, **long**, **float**, **double**

Los tipos de datos de tipo caracter mas conocidos son: **String**, **char**

Para valores true/false el tipo de dato utilizado es **boolean** y para fechas se utiliza **Date**

Pueden ser otras clases!

Atributos - Codificación

```
// JAVA o C++
class ClientePyme {
    // Atributos aquí
    String razonSocial;
    String direccion;
    Date fechaDeAlta;
    CuentaCorriente cuenta;

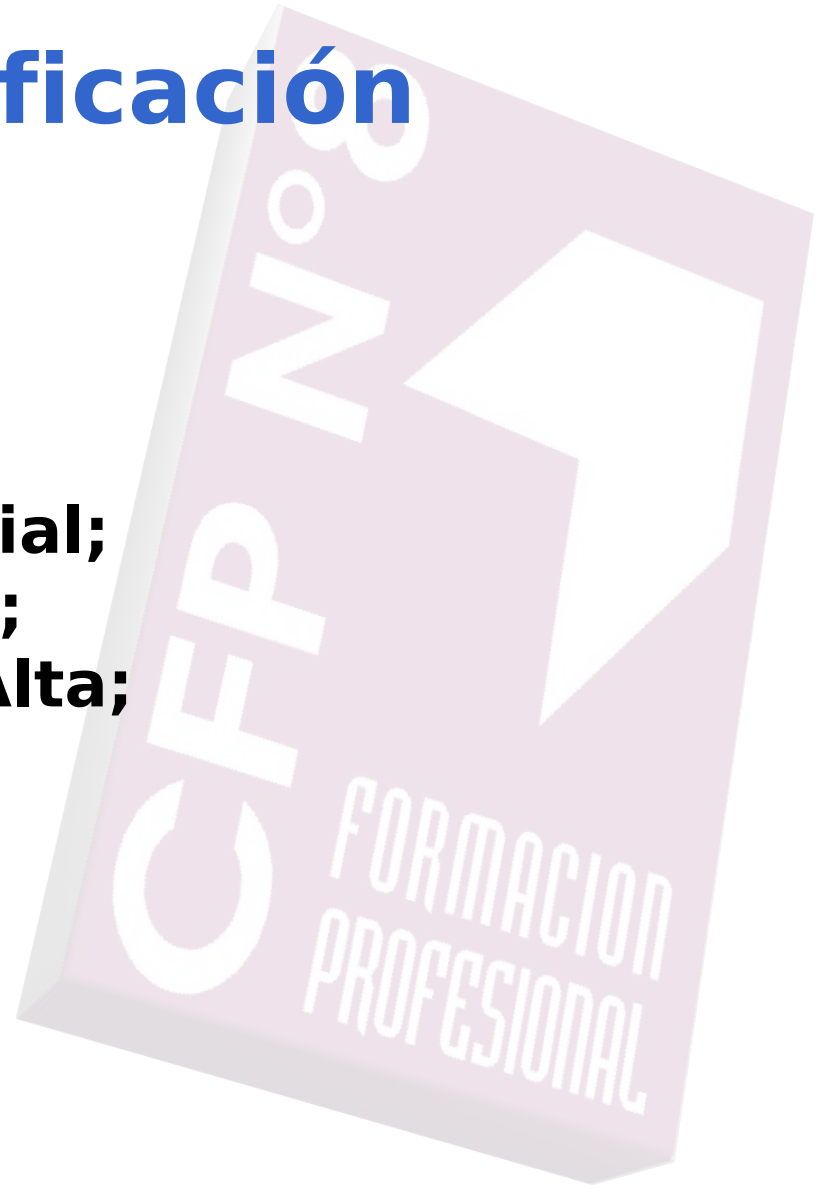
    // Métodos aquí
}
```



Atributos - Codificación

```
// PHP
class ClientePyme {
    // Atributos aquí
    private $razonSocial;
    private $direccion;
    private $fechaDeAlta;
    private $cuenta;

    // Métodos aquí
}
```



Que es una Operación

Las operaciones son acciones contenidas en una clase, y definen su comportamiento

Auto
color: string
encender() acelerar() frenar()

Dentro de un sistema, las operaciones suelen detectarse como **verbos**

Desde la perspectiva de Diseño y Programación, se denominan **Métodos**

Desde la perspectiva de Análisis, se denominan **Operaciones**

Puede tener opcionalmente valores de entrada (Parámetros) y valores de salida (Valores de Retorno)

Procedimientos (no retornan un valor) vs. **Funciones** (retornan un valor)

Que es un Valor de Entrada o Parámetro

Los parámetros son valores enviados a una operación

La operación toma los parámetros como **valores de entrada**, y así puede realizar las acciones necesarias

Todos los parámetros deben tener un tipo de dato asociado

Auto
color: string
encender() acelerar(int) frenar(int)

- Método encender() - sin parámetros.
- Método acelerar(int) - recibe como parámetro la cantidad de “km” a acelerar.
- Método frenar(int) - recibe como parámetro la cantidad de “km” que debe bajar de velocidad.

Que es un Valor de Salida o Valor de Retorno

El **valor de salida** de una operación es un **valor retornado** por la operación luego de realizar cierto procesamiento

Los valores de entrada son **datos**, y los valores de salida son considerados **información**

Todos los valores de salida deben tener un tipo de dato asociado

Es posible retornar un **único valor de salida**

Métodos - Codificación

```
// Java o C++  
class CajaDeAhorro {  
    // Atributos aquí  
    float saldo;  
  
    // Métodos aquí  
    void informarSaldo() {  
        // Imprime el atributo saldo  
        print(saldo);  
    }  
  
    float obtenerSaldo(){  
        // Retorna el saldo  
        return saldo;  
    }  
}
```



Métodos - Codificación

```
//PHP
class CajaDeAhorro {
    // Atributos aquí
    private $saldo;

    // Métodos aquí
    public function informarSaldo() {
        // Imprime el atributo saldo
        echo $this->saldo;
        echo '<br>';
    }

    public function obtenerSaldo(){
        // Retorna el saldo
        return $this->saldo;
    }
}
```



Métodos - Codificación

```
// Java o C++  
class CajaDeAhorro {  
    // Atributos aquí  
    float saldo;  
  
    // Métodos aquí  
    void depositarDinero(float unMonto) {  
        // Actualiza el valor del atributo saldo  
        saldo = saldo + unMonto;  
    }  
  
    void extraerDinero(float unMonto){  
        // Actualiza el valor del atributo saldo, NO controla si  
        monto > saldo  
        saldo = saldo - unMonto;  
    }  
}
```

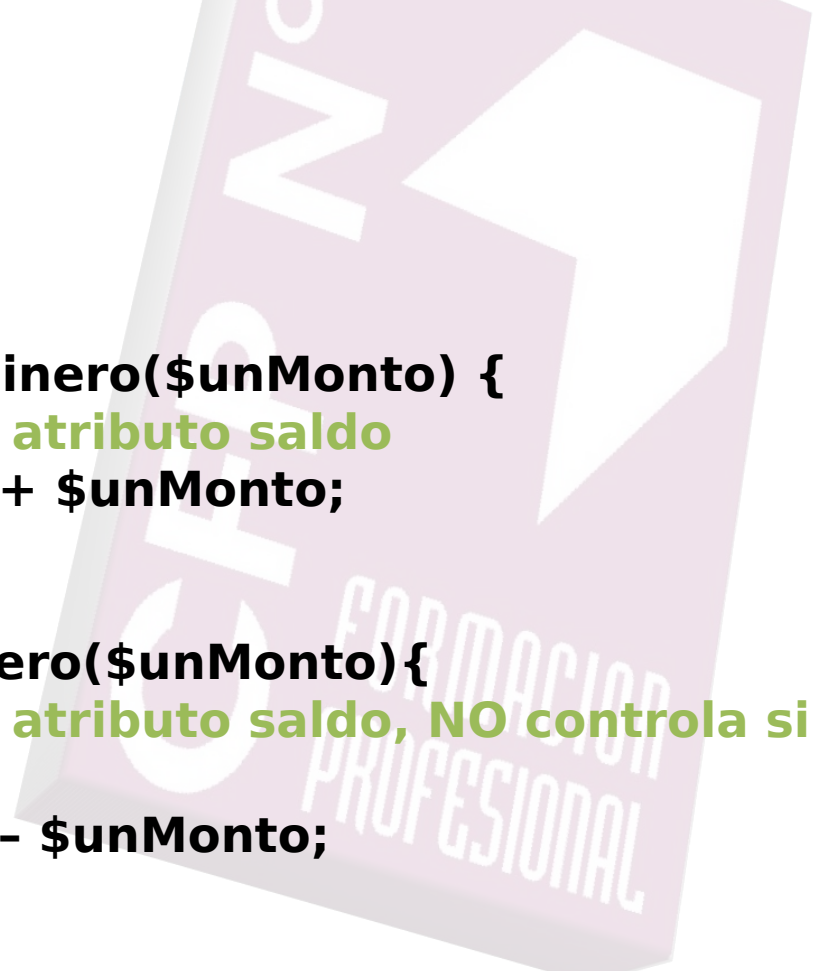


Métodos - Codificación

```
// PHP
class CajaDeAhorro {
    // Atributos aquí
    private $saldo;

    // Métodos aquí
    public function depositarDinero($unMonto) {
        // Actualiza el valor del atributo saldo
        this → $saldo = $saldo + $unMonto;
    }

    public function extraerDinero($unMonto){
        // Actualiza el valor del atributo saldo, NO controla si
        monto > saldo
        this → $saldo = $saldo - $unMonto;
    }
}
```



Que es la Sobrecarga de Operaciones

Es la aparición de métodos dentro de una misma clase que se llaman igual, pero realizan acciones (levemente) diferentes

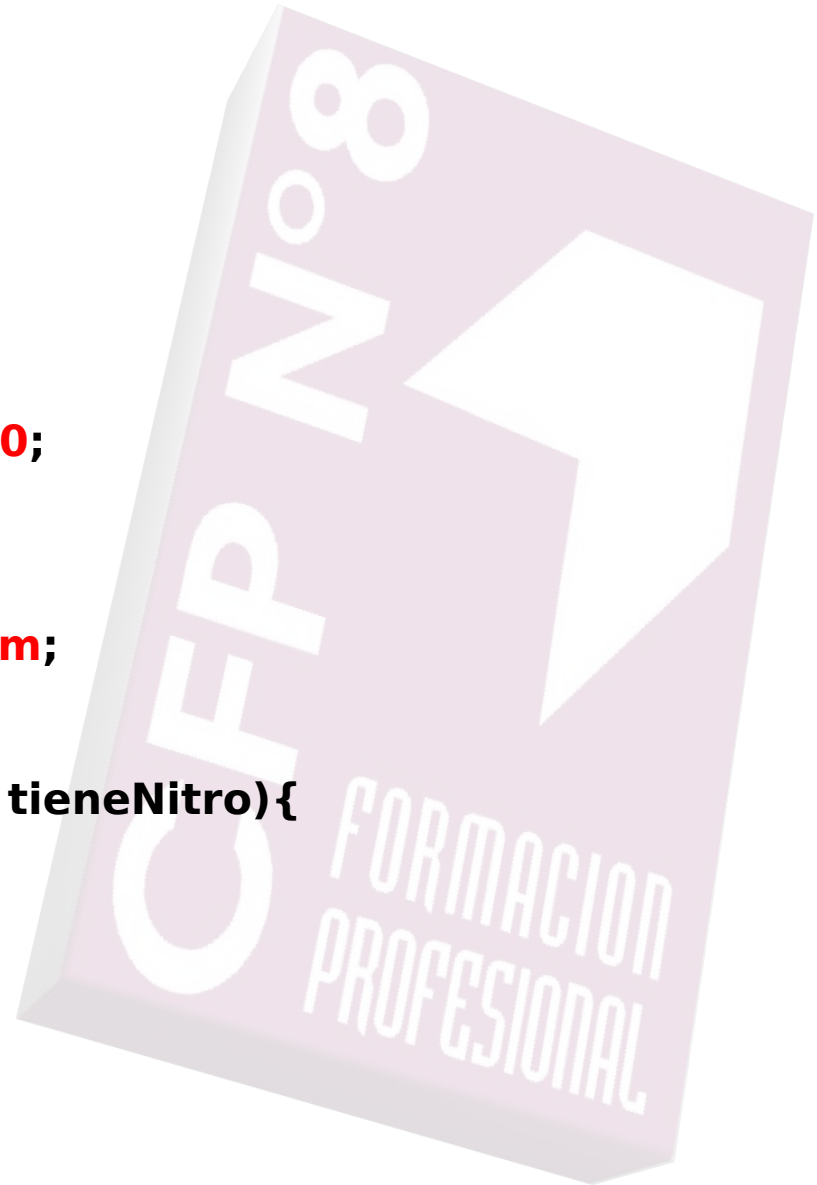
Normalmente varían en cantidad y/o tipo de parámetros

Por ejemplo, en la clase Auto podríamos encontrar variantes del método acelerar(), por ejemplo:

- acelerar() - acelera 10km/h
- acelerar(int km) - acelera de acuerdo al parametro “km”
- acelerar(int km, boolean tieneNitro) - idem caso anterior, pero si el parámetro “tieneNitro” es verdadero acelera el doble!

Nota: no soportada por PHP.

```
class Auto {  
    // Atributos aquí  
    int velocidad;  
  
    // Métodos aquí  
    void acelerar() {  
        velocidad = velocidad + 10;  
    }  
  
    void acelerar(int km) {  
        velocidad = velocidad + km;  
    }  
  
    void acelerar(int km, boolean tieneNitro){  
        if(tieneNitro == false){  
            acelerar(km);  
        } else {  
            acelerar(km*2);  
        }  
    }  
}
```



Que son las Relaciones Simples

Se produce cuando una clase se relaciona con otra clase

La relación **es única**

Por ejemplo: Auto tiene una relación simple con Motor, por que un auto puede tener un motor únicamente

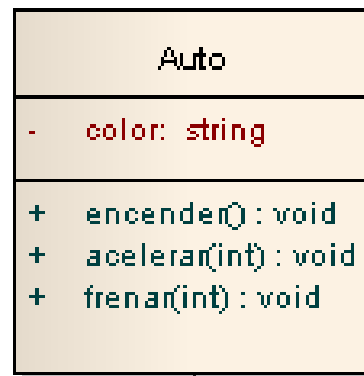
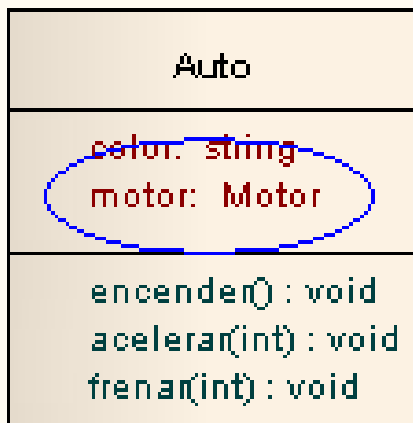
Se puede presentar como:

“... un Auto tiene un Motor ...”

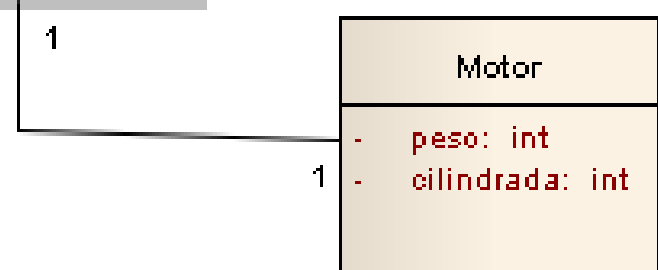
Que son las Relaciones Simples

2 Formas de representar relaciones simples gráficamente:

CASO #1



CASO #2



Relaciones simples - Codificación

```
// JAVA o C++  
class ClientePyme {  
    // Atributos aquí  
    String razonSocial;  
    CuentaCorriente cuenta;  
}
```

```
class Banco {  
  
    // Atributos aquí  
    String nombre;  
    GerenteGeneral gerente;  
}
```



Relaciones simples - Codificación

```
// PHP
class ClientePyme {
    // Atributos aquí
    private $razonSocial;
    private $cuenta;
}
```

```
class Banco {

    // Atributos aquí
    private $nombre;
    private $gerente;

}
```



Que son las Relaciones Múltiples

Se produce cuando una clase se relaciona con una o muchas otras clases. La relación **es múltiple**

Por ejemplo: Auto tiene una relación múltiple con Rueda, por que un auto puede tener varias ruedas

Se puede presentar como:

“... un Auto tiene de una a muchas Rueda(s) ...”

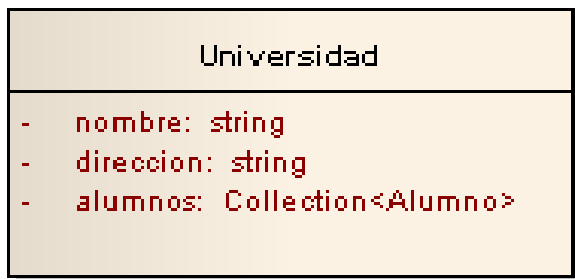
“... una Universidad tiene de uno a muchos Alumno(s) ...”

Se dice que una Universidad tiene **una colección** de alumnos

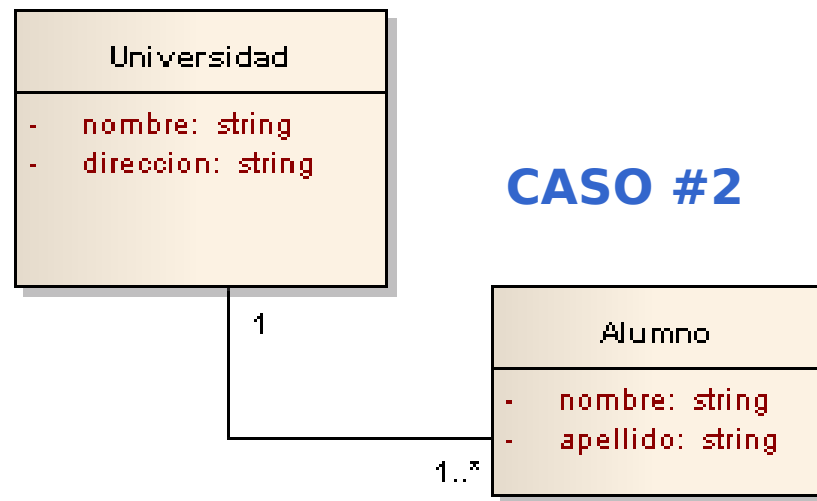
Que son las Relaciones Múltiples

2 Formas de representar relaciones simples gráficamente:

CASO #1



CASO #2



Relaciones multiples – Codificación

```
class ClienteCorporacion {
```

```
    // Atributos aquí
```

```
    String razonSocial;
```

```
    Collection<CuentaCorriente> cuentas;
```

```
}
```

```
class Banco {
```

```
    // Atributos aquí
```

```
    String nombre;
```

```
    GerenteGeneral gerente;
```

```
    Collection<Sucursal> sucursales;
```

```
    Collection<DirectorRegional> directores;
```

```
}
```

Que es la Visibilidad

Es la posibilidad de “ver” un atributo o método

Si un atributo o método es **privado (-)** solo puede verse dentro de la clase

Si un atributo o método es **publico (+)** puede verse desde otras clases

La visibilidad es establecida por los **modificadores de visibilidad: `private` y `public`**

Que es la Visibilidad

Auto
- velocidad: int
+ <code>acelerar(): void</code> + <code>frenar(): void</code>

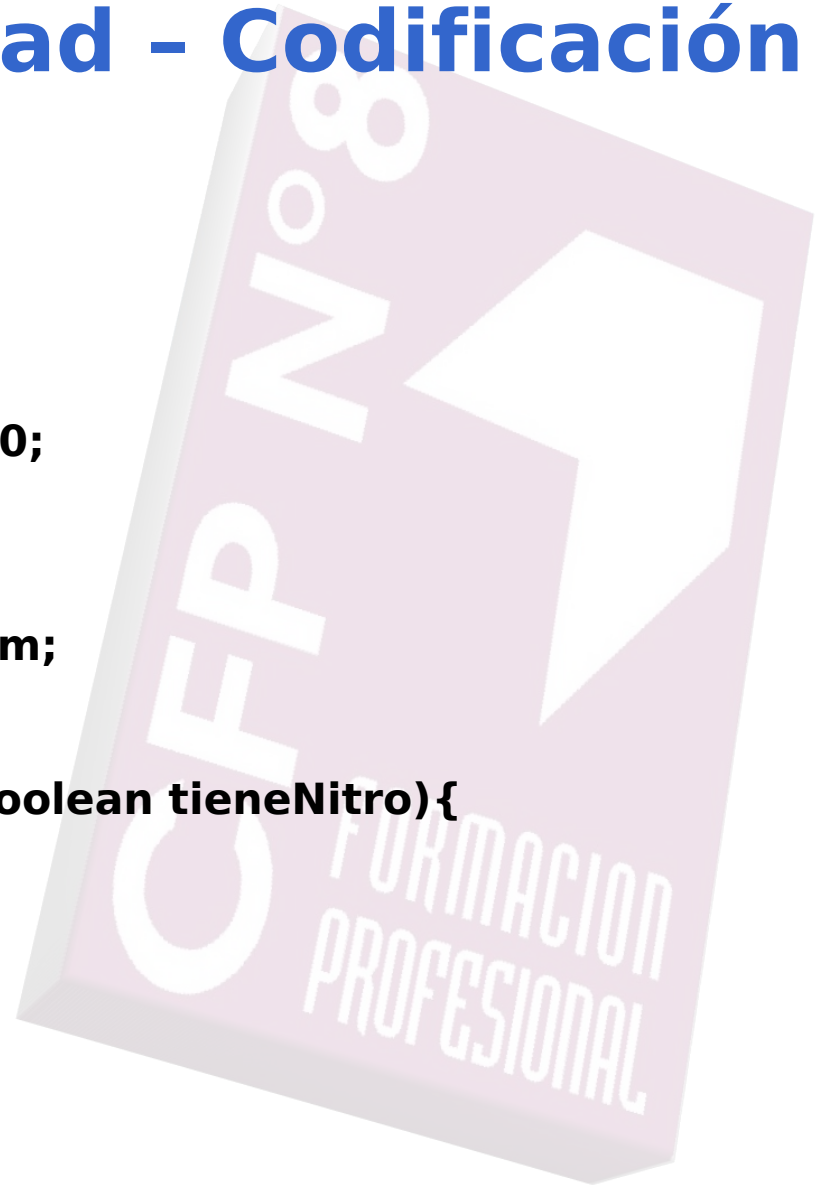
El atributo velocidad es privado, solo puede modificarse a través de los métodos `acelerar()` y `frenar()`

CuentaBancaria
- saldo: int
+ <code>depositar(int): void</code> + <code>extraer(int): void</code>

El atributo saldo es privado, solo puede modificarse a través de los métodos `depositar()` y `extraer()`

Que es la Visibilidad - Codificación

```
class Auto {  
    // Atributos aquí  
    private int velocidad;  
  
    // Métodos aquí  
    public void acelerar() {  
        velocidad = velocidad + 10;  
    }  
  
    public void acelerar(int km) {  
        velocidad = velocidad + km;  
    }  
  
    public void acelerar(int km, boolean tieneNitro){  
        if(tieneNitro == false){  
            acelerar(km);  
        } else {  
            acelerar(km*2);  
        }  
    }  
}
```



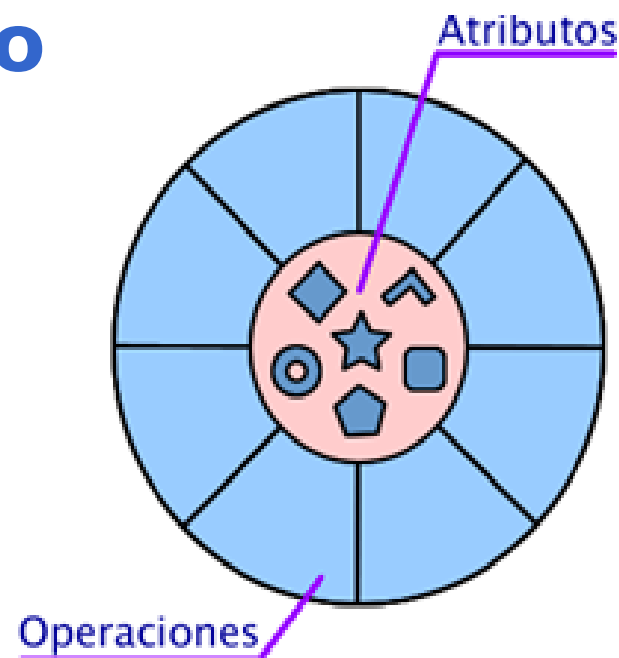
Que es el Encapsulamiento

Es el ocultamiento del estado de un objeto

El estado (atributos) podrá accederse únicamente a través de sus operaciones (métodos)

En la clase, los **atributos** deben ser **privados** y los **métodos** para acceder a los atributos deben ser **públicos**

El atributo **saldo** esta **encapsulado**, solo puede accederse a través de los métodos depositar() y extraer()



CuentaBancaria	
-	saldo: int
+	depositar(int): void
+	extraer(int): void

Que son los Setters y los Getters

Son **métodos de acceso públicos** a atributos privados

Representan la única forma de acceder a los atributos

Setter : método utilizado para **setear** un valor a un atributo

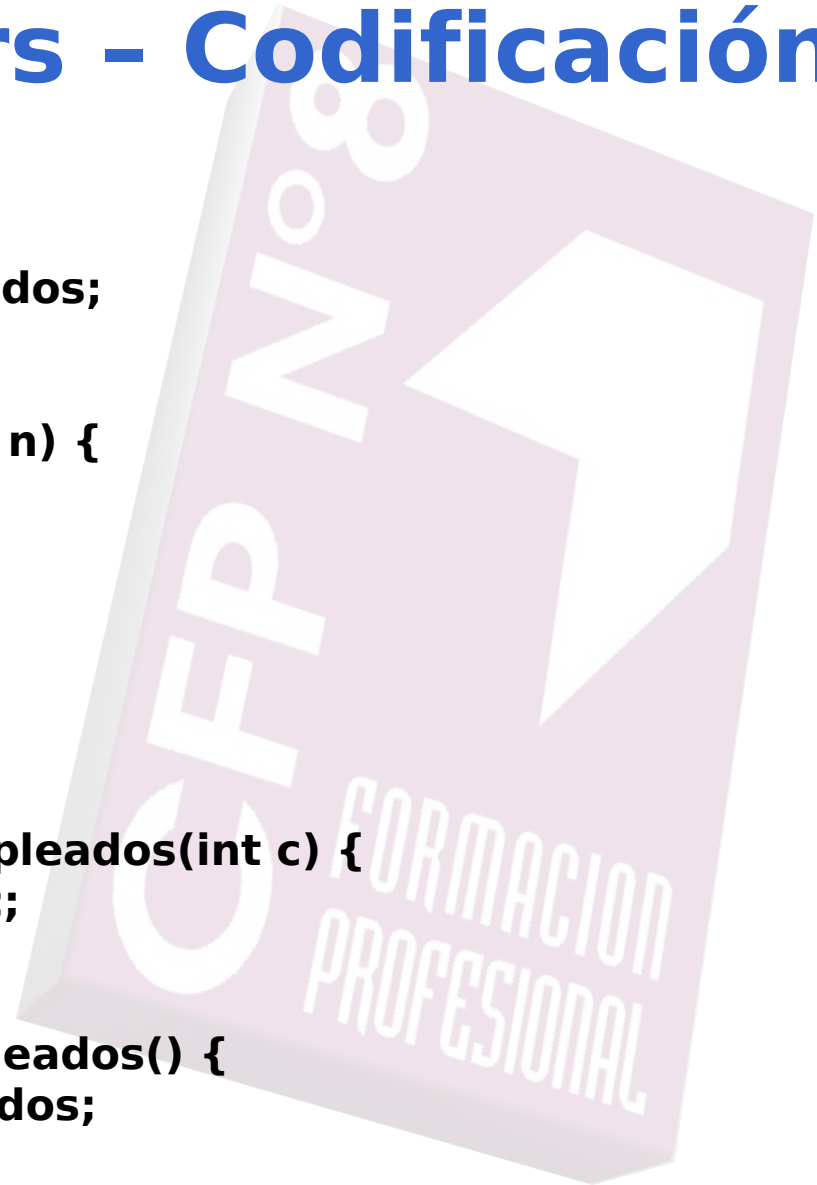
Getter : método utilizado para **obtener** un valor de un atributo

Los IDEs generalmente permiten generar los setters y los getters de forma automática!

Banco
- nombre: String - cantidadDeEmpleados: int - cantidadDeSucursale: int - fechaDeConstitucion: Date
+ getCantidadDeEmpleados() : int + setCantidadDeEmpleados(int) : void + getCantidadDeSucursales() : int + setCantidadDeSucursales(int) : void + getFechaDeConstitucion() : Date + setFechaDeConstitucion(Date) : void + getNombre() : string + setNombre(string) : void

Setters y Getters - Codificación

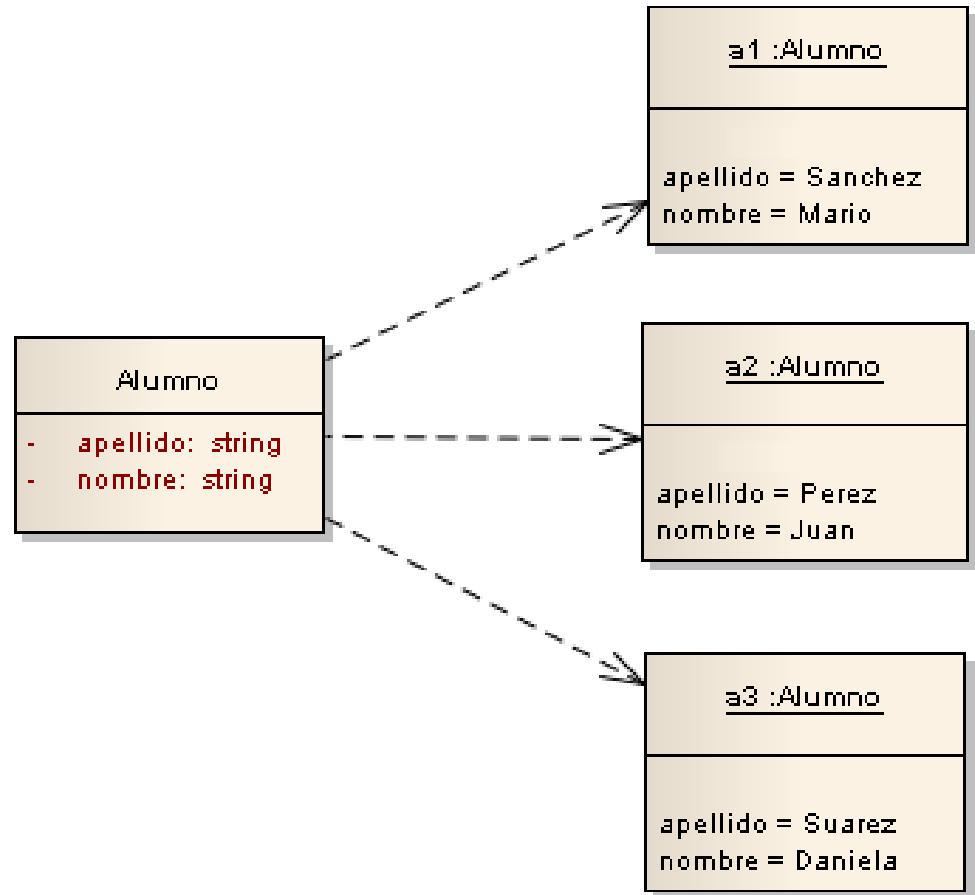
```
class Banco {  
    // Atributos aquí  
    private String nombre;  
    private int cantidadDeEmpleados;  
  
    // Métodos aquí  
    public void setNombre(String n) {  
        nombre = n;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setCantidadDeEmpleados(int c) {  
        cantidadDeEmpleados = c;  
    }  
  
    public int getCantidadDeEmpleados() {  
        return cantidadDeEmpleados;  
    }  
}
```



Objetos vs. Clases

La **clase** representa un concepto, es un molde, una plantilla

Los **objetos** representan instancias de una clase. Sería como tomar una plantilla (una clase) y personalizarla (completar sus atributos)



Constructores para Construir Objetos

Los objetos son contruidos a partir de una clase.

Todos los objetos dependen de una clase

Para construir un objeto es necesario utilizar un **constructor**

El constructor es “un método” de la clase que se invoca al construir un objeto, y en su interior tiene un conjunto de acciones a realizar

El constructor tiene el mismo nombre que la clase, y para invocarlo hay que utilizar una palabra clave del lenguaje de programación que se denomina **new**

FORMA GENERICA: **NombreDeClase nombreDeObjeto = new Constructor();**

EJEMPLO: **Auto a = new Auto();**

Constructores - Conceptos Avanzados

Como se llama cuando una clase tiene mas de un constructor?

Sobrecarga de constructores

Una clase debe **tener al menos un constructor**. Si no se agrega un constructor, normalmente se asume que posee el constructor vacío

Una clase puede tener todos los constructores que sean necesarios

Que ocurre si hay dos constructores con la misma firma? Por ejemplo:

```
Alumno(String n){  
    nombre = n;  
}  
  
Alumno(String a){  
    apellido = a;  
}
```

NO FUNCIONA!
***Al igual que los métodos, si tienen
misma cantidad de parámetros, deben
tener diferente tipo de dato***

La clase “Programa”

Quien instancia el primer objeto?

La clase Programa a través de un método llamado **main()**, que se invoca automáticamente al ejecutar nuestra aplicación

```
class Programa {  
  
    public static void main(String[] args) {  
  
        // código aquí  
  
        // todo el código ubicado aquí se ejecuta  
        automáticamente  
        // al ejecutar nuestra aplicación  
  
    }  
  
}
```

Interacción entre “Programa” y alumnos

```
class Alumno {  
    // Atributos aquí  
    private String nombre;  
    private String apellido;
```

// Constructores

```
Alumno() {  
}
```

```
Alumno(String n, String a){  
    nombre = n;  
    apellido = a;  
}
```

// Métodos aquí

```
public void informar(){  
    print(nombre + apellido);  
}
```

```
class Programa {
```

```
    public static void main(String[] args) {
```

```
        Alumno a1 = new Alumno();
```

```
        Alumno a2 = new Alumno("Juan",  
                                "Perez");
```

```
        a1.informar();  
        a2.informar();
```

```
    }  
}
```

1

2

3

Mecanismo de Herencia

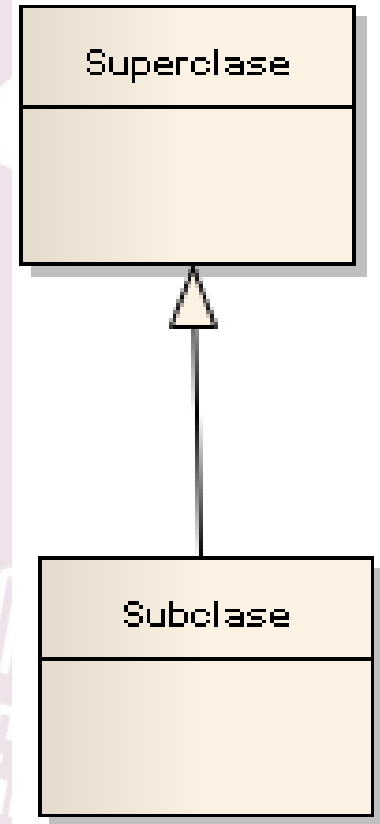
Mecanismo que permite que una clase "herede de otra clase" o "extienda otra clase", **recibiendo o heredando atributos y operaciones** de su clase "padre".

La **clase principal** se denomina: superclase o clase padre

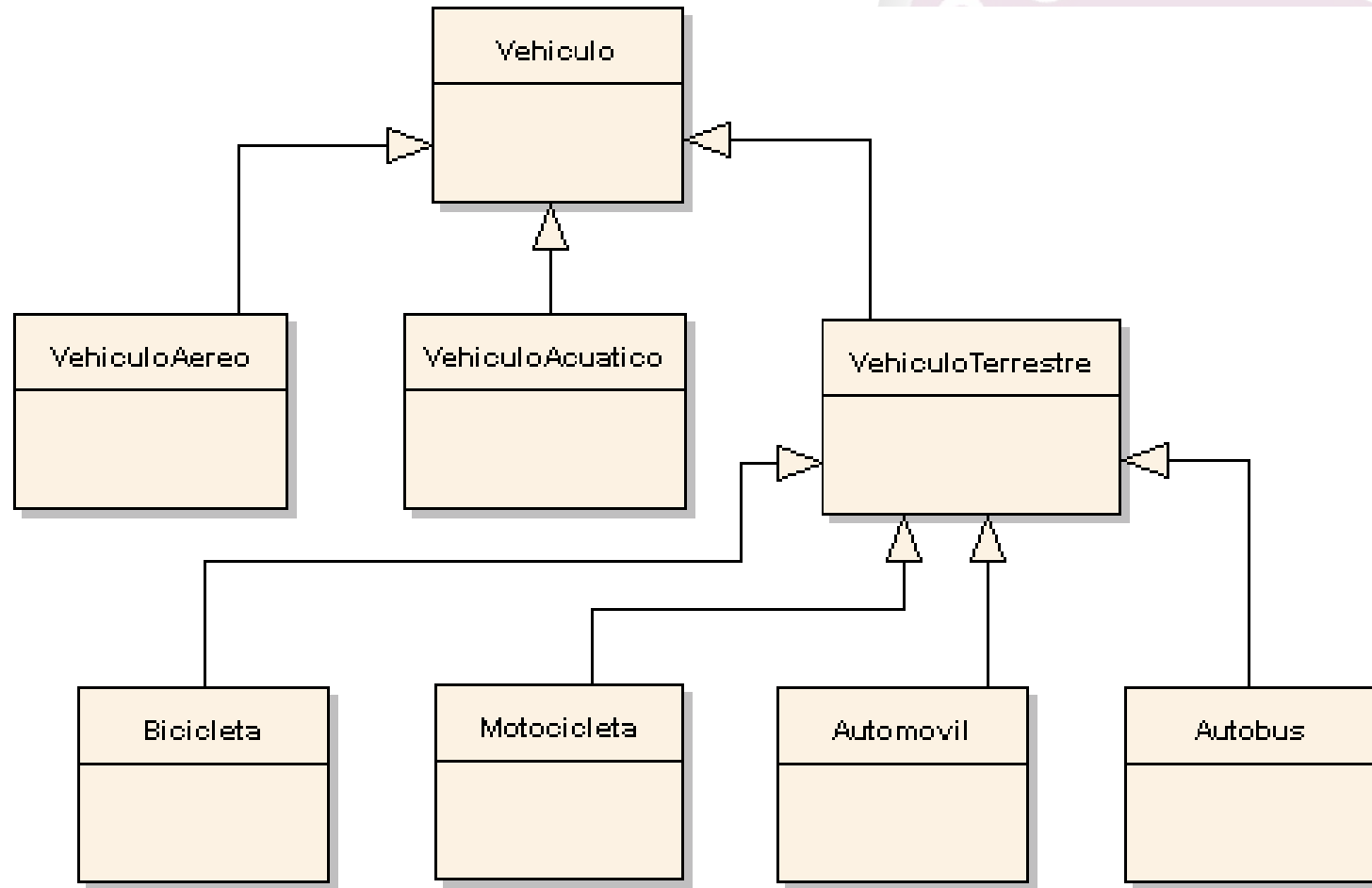
La **clase que hereda** se denomina: subclase o clase hija o clase derivada

La relación se interpreta como "ES UN"

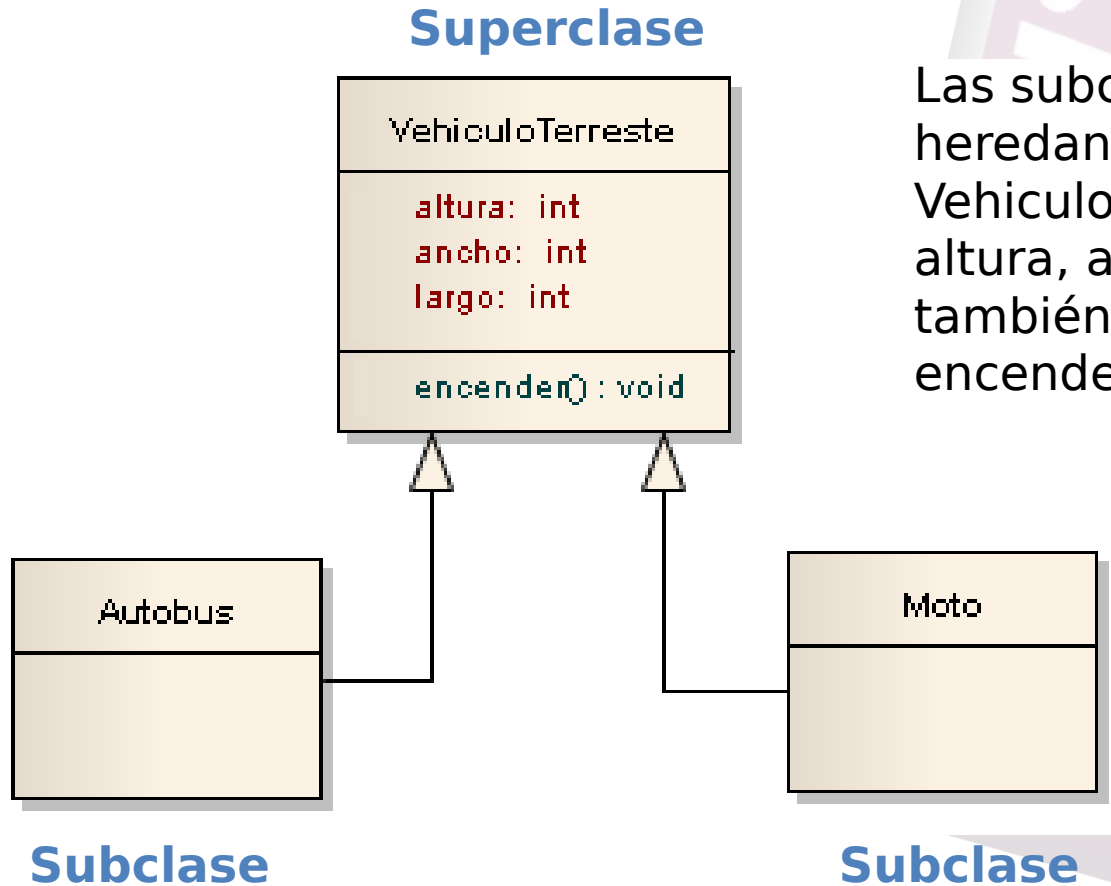
Ejemplo: un Auto **ES UN** VehiculoTerrestre



Mecanismo de Herencia - Ejemplo

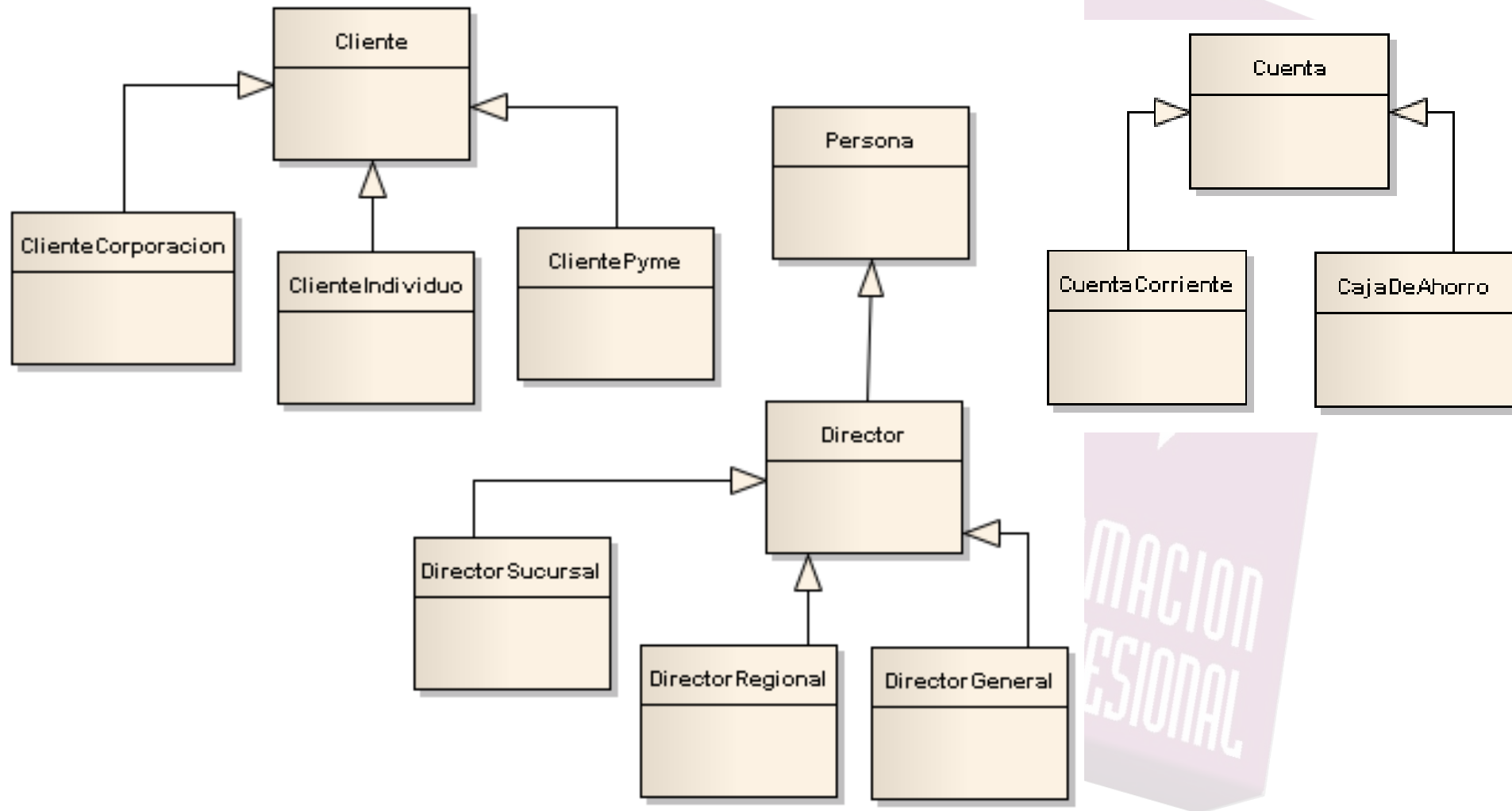


Mecanismo de Herencia - Ejemplo



Las subclases Autobus y Moto heredan de la superclase VehiculoTerreste los atributos altura, anchura, largo, y también la operación encender()

Ejercicio #3 - Mapa de Herencia



Herencia - Codificación

```
class Cliente {  
    // Atributos  
    String cuit;  
    String direccion;  
  
    // Constructores  
    // Métodos  
}
```

```
class ClientePyme extends Cliente {  
    // Atributos - Los atributos heredados no se vuelven a codificar!  
    String razonSocial;  
  
    // Constructores  
    // Métodos  
}
```



Que es el Polimorfismo

Es la posibilidad de que una clase presente un comportamiento distinto de acuerdo a una situación

2 Tipos de Polimorfismo: sin redefinición y con redefinición

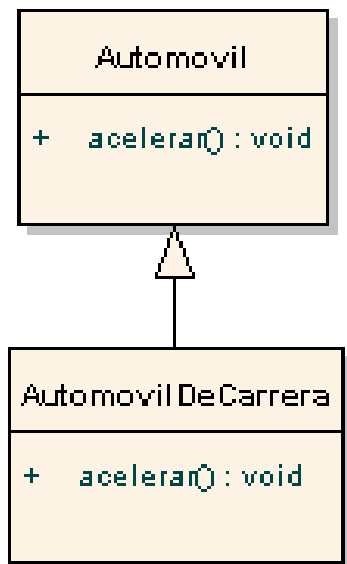
Polimorfismo sin redefinición: Una clase que posee varios métodos llamados iguales pero con diferentes firmas. También llamado *Sobrecarga de Operaciones*

Auto	
-	color: string
+	acelerar(): void
+	acelerar(int): void
+	acelerar(int, boolean): void
+	frenar(int): void
+	encender(): void

3 versiones del método acelerar()

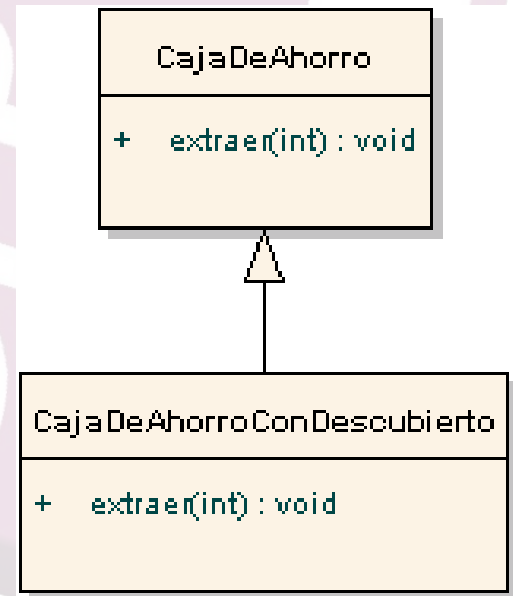
Que es el Polimorfismo

Polimorfismo con redefinición: Una subclase hereda métodos de su superclase pero decide modificarlos por que debería comportarse de forma diferente. También llamado *Redefinición de Métodos* o *Method Override*



Cualquier tipo de Automovil acelera igual que un AutomovilDeCarrera ?

Extraer dinero de una caja de ahorro sin descubierto se realiza de la misma forma que desde una caja con descubierto ?



Que es el Polimorfismo - Codificación

```
class CajaDeAhorro {  
  
    // Atributos  
    float saldo;  
  
    // Métodos  
    public void extraer(int monto) {  
        saldo = saldo - monto;  
    }  
}
```

```
class CajaDeAhorroDescubierto extends CajaDeAhorro {  
  
    // Atributos  
    float saldoDescubierto;  
  
    // Métodos  
    public void extraer(int monto) {  
        // este método se vuelve a escribir, misma firma pero diferente cuerpo  
        // aquí el código que contemple la extracción con saldo descubierto  
    }  
}
```

Atributos de Instancia y de Clase

Atributos de Instancia: son atributos que pertenecen a un objeto en particular

Atributos de Clase: son atributos que pertenecen a la clase y no a un objeto o instancia de clase. Esto significa que son atributos compartidos por todos los objetos. También llamados *atributos estáticos*

Para definir un atributo estático se utiliza la palabra clave **static**

```
class CajaDeAhorroConDescubierto
{

    // Atributos aquí
    public float saldo;
    public static int descubierto = 5000;

}
```

CajaDeAhorroConDescubierto

descubierto: int

Asumimos en este caso que todas las cajas de ahorro cuentan con el mismo descubierto

Métodos de Instancia y de Clase

Métodos de Instancia: son métodos que pertenecen a un objeto en particular e impactan en el comportamiento de ese objeto únicamente

Métodos de Clase: son métodos que pertenecen a la clase y no a un objeto o instancia de clase. Esto significa que son métodos compartidos por todos los objetos. También llamados *métodos estáticos*

Para definir un método estático se utiliza la palabra clave **static**

```
class CajaDeAhorroConDescubierto
{
    // Atributos aquí
    public float saldo;
    private static int descubierto = 5000;

    // Metodos aquí
    public static int leerDescubierto() {
        return descubierto;
    }
}
```

CajaDeAhorroConDescubierto	
-	<u>descubierto: int</u>
+	<u>leerDescubierto(): int</u>
+	<u>modificarDescubierto(int): void</u>

Clases Abstractas y Clases Concretas

Las Clases Concretas

- son clases que se pueden instanciar
- por ejemplo la clase Alumno existe para generar diversos objetos del tipo Alumno

Las Clases Abstractas

- son clases que no se pueden instanciar
- representan conceptos muy genéricos de la realidad
- por ejemplo la clase Vehiculo o la clase Persona son conceptos muy abstractos, seria difícil pensar en armar un objeto a partir de estas clases

Codificación:

```
abstract class Persona {
```

```
// Atributos aquí
```

```
// Métodos aquí
```

```
}
```

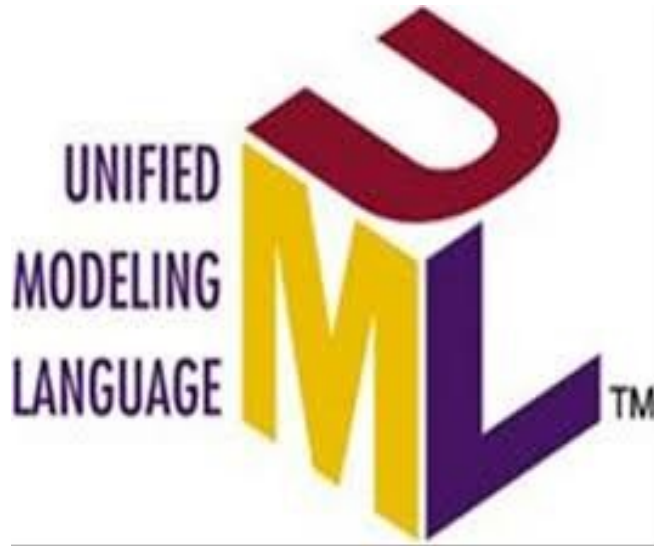
Generación Automática de Código

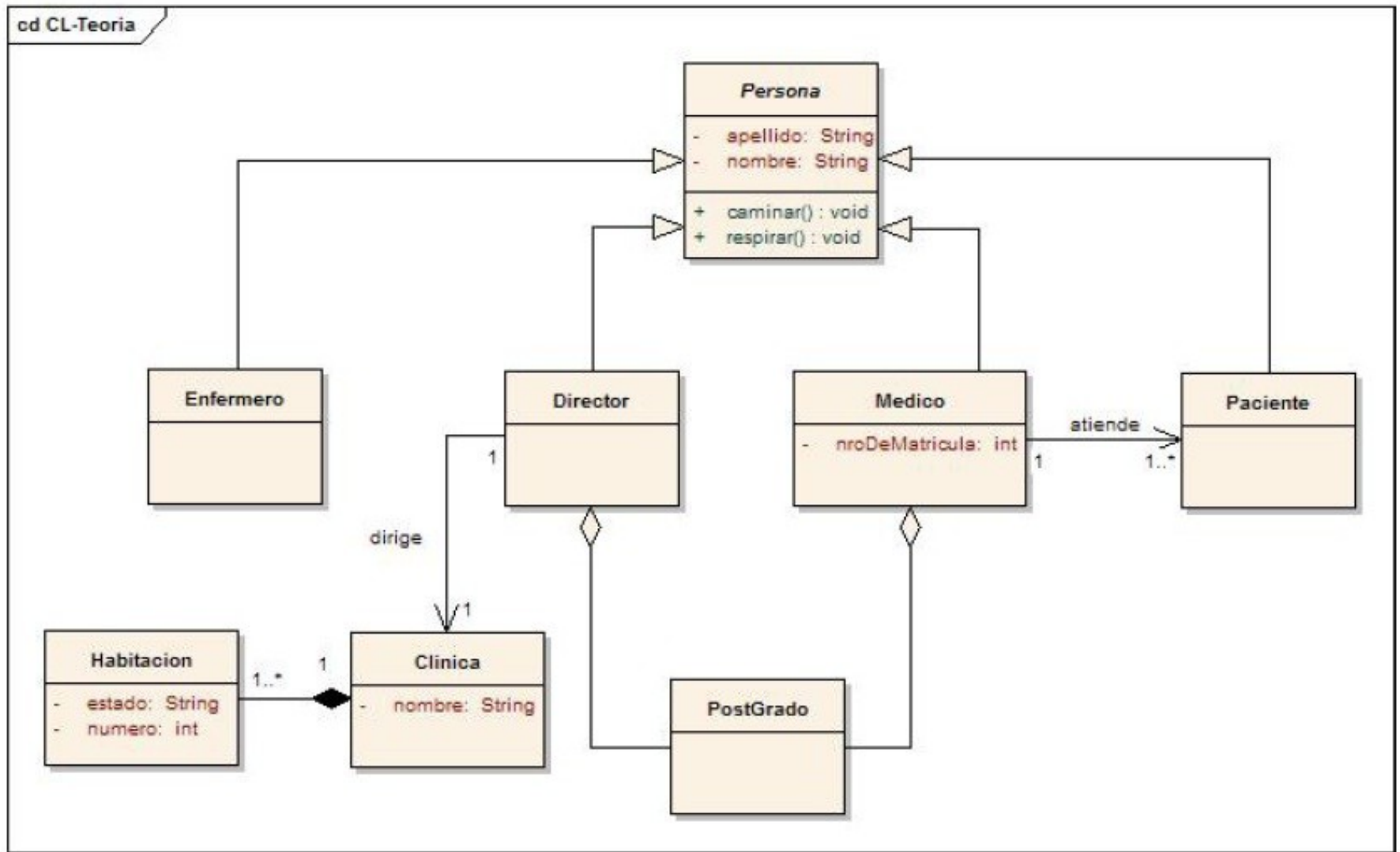
El mecanismo de generación automática de código se denomina ***Ingeniería Directa***

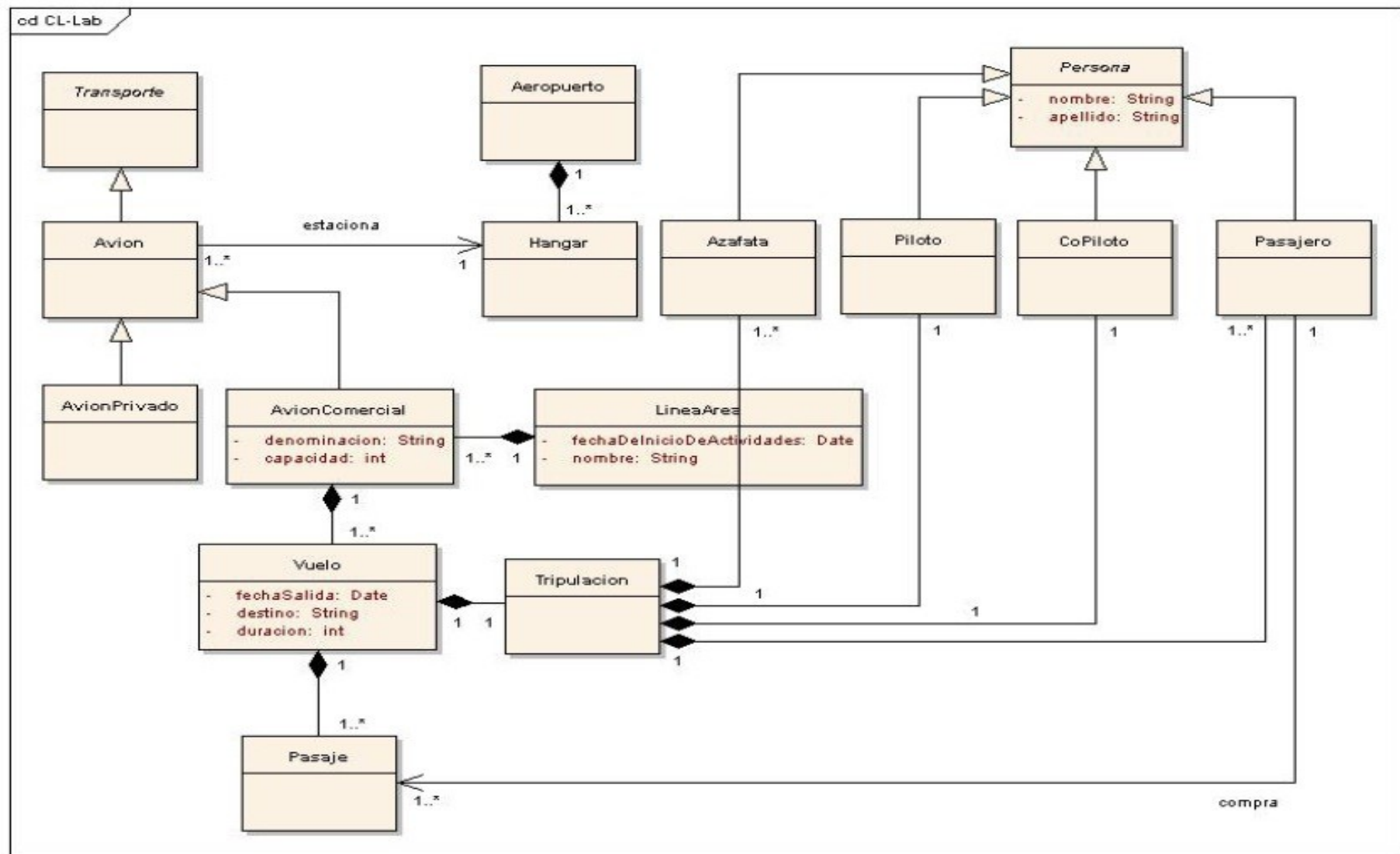
Existen herramientas que permiten generar código fuente a partir de diagramas, por ejemplo el Enterprise Architect

Estas aplicaciones permiten generar código fuente en diversos lenguajes, como ser: C#, VB.NET, PHP, Java, Actionscript, y mas

UML Casos de Ejemplos







FIN

Muchas Gracias!

