¿Qué resuelve la Programación Orientación a Objetos?

- Principalmente esos problemas y huecos que nos deja la programación estructurada tales como:
1. **Código muy largo:**
2. **Si algo falla todo se rompe**
3. **Código Spaguetti:** Muchas sentencias de control anidadas y pérdida de control sobre el código.

- Simplifica la programación, la hace más práctica, divertida y amigable.

Lenguajes Orientados a Objetos

No te cases con ningún lenguaje.

Lenguajes con los que podemos aprender a programar orientados a objetos son:

- Java
- PHP
- Python
- JavaScript
- C#
- Ruby
- Kotlin

Java

Java nació como un lenguaje OOP, su paradigma principal es este. Este lenguaje es muy utilizado para creación a Apps móviles o del lado de servido como lenguaje Backend.

- Extensión: .java

PHP

PHP es un lenguaje pensado para web y es un lenguaje interpretado, esto quiere decir que el navegador es quién interpreta php.

- Extensión: .php

Python

Python es un lenguaje fácil de usar, y tiene múltiples usos como del lado Web, Server side, Análisis de datos, Machine Learning, ect. Además, es un lenguaje OOP.

- Extensión: .py

JavaScript

Javascript es un lenguaje interpretado, también es un lenguaje OOP pero este está basado en prototipos. Esta pensado para web pero podemos trabajar el backend Node.js.

- Extensión: .js

—

Diagramas de Modelado

Nos permiten plasmar de forma gráfica a través de diagramas nuestro análisis. Servirá de intermediario para poder entender el problema y la solución con la orientación a objetos.

- Existen dos herramientas de diagramación:

OMT (Object Modeling Techniques)

Metodología que se basa estrictamente en identificar los objetos, sus métodos, sus atributos y cuales son las relaciones que tienen. Pero ya está descontinuada.

UML (Unified Modeling Languaje)

Toma todas las bases y técnicas del **OMT**, las unifica y permite una ampliación de apicaciones, no solo objetos y clases. Sino que incluye casos de uso, actividades, iteración, estados e implementación.

Un buen desarrollador debe manejar y dominar con fluidez, conceptos de UML ya que es lo que nos van a entregar cuando empecemos a trabajar en un proyecto que se haya construido bajo la arquitectura POO.

UML
Unified Modeling Language

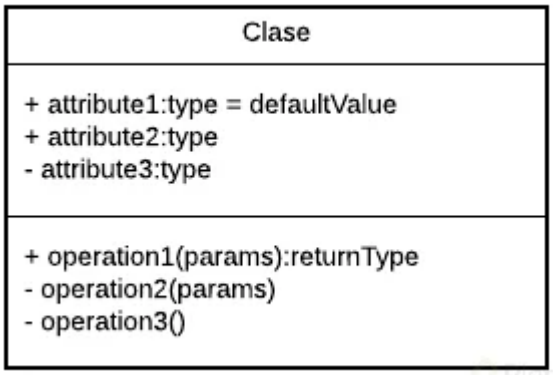
+ Clases
+ Casos de Uso
+ Objetos
+ Actividades
+ Iteración
+ Estados
+ Implementación

—

UML significa Unified Modeling Language el cual es un lenguaje estándar de modelado de sistemas orientados a objetos.

—

Las clases se representan así:



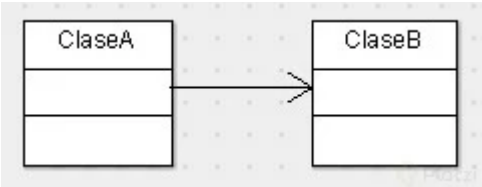
En la parte superior se colocan los atributos o propiedades, y debajo las operaciones de la clase. Notarás que el primer caracter con el que empiezan es un símbolo. Este denotará la visibilidad del atributo o método, esto es un término que tiene que ver con Encapsulamiento y veremos más adelante a detalle.

Estos son los niveles de visibilidad que puedes tener:

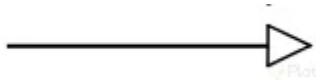
- private
+ public
# protected
~ default

Una forma de representar las relaciones que tendrá un elemento con otro es a través de las flechas en UML, y aquí tenemos varios tipos, estos son los más comunes:
Asociación

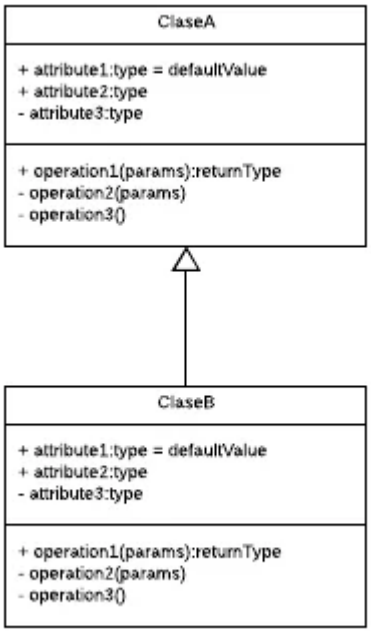
Como su nombre lo dice, notarás que cada vez que esté referenciada este tipo de flecha significará que ese elemento contiene al otro en su definición. La flecha apuntará hacia la dependencia.



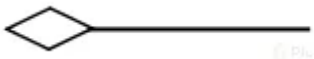
Con esto vemos que la ClaseA está asociada y depende de la ClaseB.
Herencia



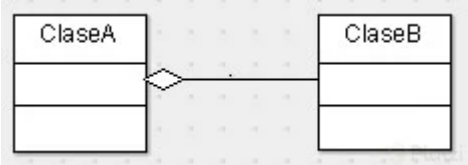
Siempre que veamos este tipo de flecha se estará expresando la herencia.
La dirección de la flecha irá desde el hijo hasta el padre.



Con esto vemos que la ClaseB hereda de la ClaseA
Agregación

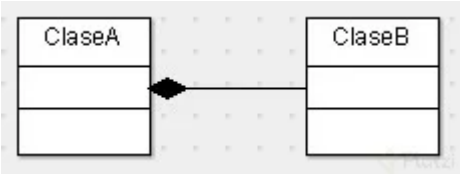


Este se parece a la asociación en que un elemento dependerá del otro, pero en este caso será: Un elemento dependerá de muchos otros. Aquí tomamos como referencia la multiplicidad del elemento. Lo que comúnmente conocerías en Bases de Datos como Relaciones uno a muchos.



Con esto decimos que la ClaseA contiene varios elementos de la ClaseB. Estos últimos son comúnmente representados con listas o colecciones de datos.
Composición

Este es similar al anterior solo que su relación es totalmente compenetrada de tal modo que conceptualmente una de estas clases no podría vivir si no existiera la otra.



—

Objetos

Los Objetos son aquellos que tienen propiedades y comportamientos, también serán sustantivos.

- Pueden ser Físicos o Conceptuales

Las **Propiedades** también pueden llamarse atributos y estos también serán sustantivos. Algunos atributos o propiedades son nombre, tamaño, forma, estado, etc. Son todas las características del objeto.

Los **Comportamientos** serán todas las operaciones que el objeto puede hacer, suelen ser verbos o sustantivos y verbos. Algunos ejemplos pueden ser que el usuario pueda hacer login y logout.

—

Abstracción

La abstracción es un concepto que usamos de forma natural para describir los objetos que nos rodean. Consiste en definir un objeto a partir de sus características más representativas (atributos y métodos), permitiendo describir de manera simple un objeto mucho más complejo. Por ejemplo, si pensamos en un carro, naturalmente pensamos en una máquina

de 4 ruedas y un timón (atributos) que se desplaza hacia adelante o hacia atrás a diferentes velocidades(métodos). Sin embargo, un carro es un objeto muchísimo más complejo, con varios sistemas interconectados que le dan su funcionalidad y apariencia. Abstraemos esa compleja máquina en un "carro" mediante las características más representativas, simplificando su definición. Esto nos permite modelar el objeto y representarlo mediante código, sin tener que definir la gran complejidad implícita detrás de él. Mientras más atributos y métodos definamos de un objeto, mayor grado de abstracción tendrá, y naturalmente, su definición será más cercana al objeto real.

—

Modular: Dividir un sitema y así crear módulos independientes, lo que permite evitar un colapso masivo en nuestro código y mejorar la legibilidad.

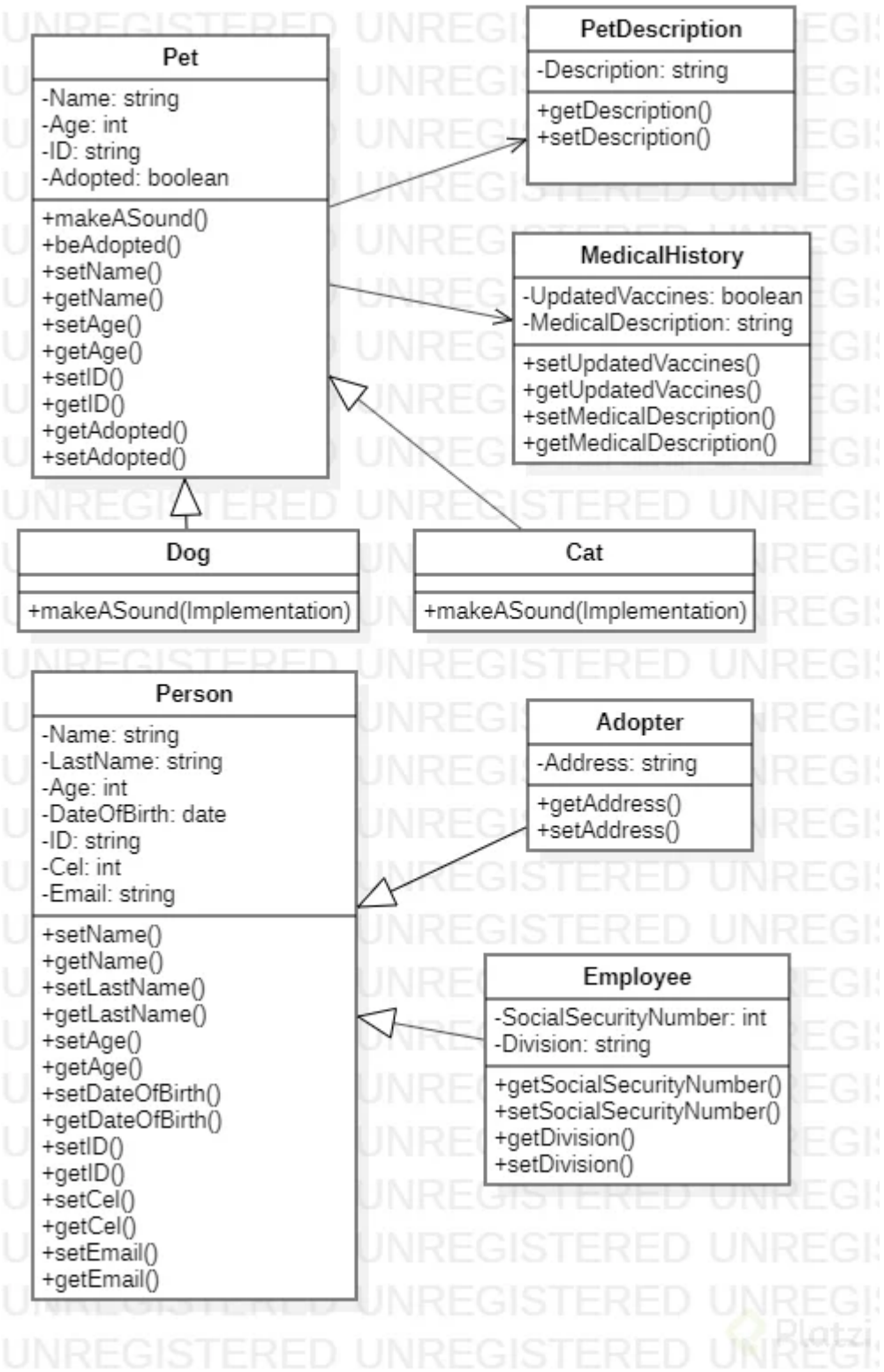
—

Objetos identificados Uber:

- **User:** El usuario, persona, cliente.
- **Car:** *UberX, UberPool, UberBlack, UberVan.*
- **Route:** Ruta de un punto **A** al **B**, tu ruta de traslado.
- **Driver:** Conductor del auto, del objeto car.
- **Payment:** *Card, Paypal, Cash,* método de pago por el servicio.
- **Trip:** Viaje realizado del punto **A** al **B**.

—

Reto 1



—

Clases en UML y su sintaxis en código

.

Recuerda que el proceso es:

- Identificar el problema, y objetos
- Definir las clases
- Plasmarlas en un diagrama

Se lo puede plasmar en UML :

- **Identidad,** que será el nombre de la clase.
- **Estado,** que serán los atributos de la clase.
- **Comportamiento,** que serán las operaciones de la clase.

—

Don’t repeat yourself es una filosofía que promueve la reducción de duplicación en programación, esto nos va a inculcar que no tengamos líneas de código duplicadas.

Toda pieza de información nunca debería ser duplicada debido a que incrementa la dificultad en los cambios y evolución

La **herencia** nos permite crear nuevas clases a partir de otras, se basa en modelos y conceptos de la vida real. También tenemos una jerarquía de **padre e hijo**.

—

Los **objetos** nos ayudan a crear instancia de una clase, el objeto es el resultado de lo que modelamos, de los parámetros declarados y usaremos los objetos para que nuestras clases cobren vida.

Los **métodos constructores** dan un estado inicial al objeto y podemos añadirle algunos datos al objeto mediante estos métodos. Los atributos o elementos que pasemos a través del constructor serán los datos mínimos que necesita el objeto para que pueda vivir.

—

No se puede usar un ArrayList con dos parámetros de entrada

super hace referencia a la clase padre

this hace referencia a la clase hija

—

El **encapsulamiento** es hacer que un dato sea inviolable, inalterable cuando se le asigne un **modificador de acceso** (no se trata solo de ocultar el dato sino también de protegerlo). Un modificador de acceso define el alcance y visibilidad de un miembro de clase.

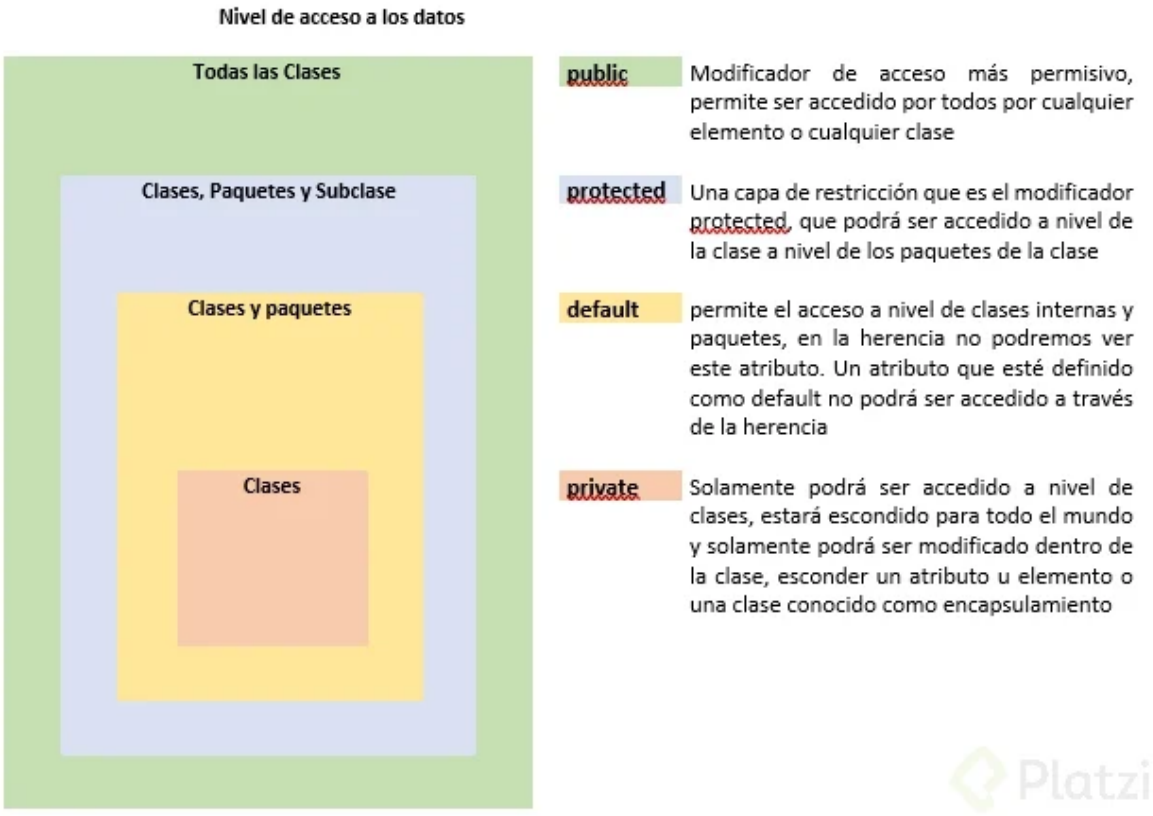
La encapsulación es también llamada **ocultamiento de información**.

Public: Todas las Clases.

Protected: En la misma Clase, paquetes, subclases.

Default: En la misma Clase y paquetes internos.

Private: Solo en la misma Clase



—-