**UNIFIED MODELING LANGUAGE (UML) DIAGRAMA DE CLASES** [**https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagramas-de-clases-con-uml/**](https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagramas-de-clases-con-uml/)

El Lenguaje Unificado de Modelado ([UML](https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado)) es una vía (lenguaje) de modelado visual, una combinación de varias notaciones orientadas a objetos: diseño orientado a objetos, técnica de modelado de objetos e ingeniería de software orientada a objetos.

Según su función existen modelos:

**De Comportamiento (Dinámicos):** contienen el flujo de información entre los objetos del modelo, durante la ejecución de un proceso.

**Estructurales (Estáticos):** El más utilizado entre los estructurales o estáticos, el diagrama de clases asigna clases a las instancias de objeto en función de sus propiedades, lo que implica una cierta dependencia jerárquica. Al mismo tiempo existen relaciones entre las diferentes clases o entre objetos.

1. **COMPONENTES**

CLASES

En los diagramas de clases, la clase es un elemento-modelo y una especialización del clasificador encapsulado (*EncapsulatedClassifier*) y del clasificador de comportamiento (*BehavioredClassifier*). **Comprende un conjunto de instancias**. Las instancias-objeto dentro de una clase comparten las mismas propiedades (*Attributes*) y comportamientos (*Methods*), así como la misma semántica, es decir, utilizan los mismos caracteres con el mismo significado. Esto convierte a la clase en una especie de plantilla para sus objetos, “instanciando” los objetos y definiendo su conducta en el sistema.

* **+** = pública
* **-** = privada
* **#** = protegida
* **/**= derivada (
* **~**= paquete
* **\***= aleatoria

INSTANCIAS(OBJETOS)

INTERFACES

* Estructura de un sistema
* Clases
* Modelos
* Atributos
* Métodos (operaciones)
* Relaciones entre objetos

<https://kodigoswift.com/uml/>

**HERENCIA**

* Se designa por una flecha con la punta blanca o hueca
* Se destina a la clase base o padre

**PROPIEDAD**

* Se representa con una flecha negra en su punta
* Denota

**PARADIGMAS DE PROGRAMACION**

* **IMPERATIVO** INSTRUCCIONESCAMBIANDO EL PROGRAMA A MEDIDA QUE VOY NECESITANDO COSAS (EL COMO) HASTA LLEGAR A LA SOLUCIÓN.

PROCIDEMENTAL  
ORIENTADO A OBJETOS (producto nombre, apellido, correo contraseña precios,datos) (seleccionar())

Carrito almacenar producto cantidad

Usuario  
Pago

* **FUNCIONAL (DECLARATIVOS)**
* **LÓGICO (DECLARATIVOS, EL QUE QUIERO)**

**PROGRAMACION ORIENTADA A OBJETOS** [**https://www.youtube.com/watch?v=DlphYPc\_HKk**](https://www.youtube.com/watch?v=DlphYPc_HKk)

* Datos igual a ATRIBUTOS
* Funciones igual METODOS
* CLASE- class plantilla base con atributos y métodos, se crea en código estructura básica del objeto. Cuando una clase crea nuevos objetos de usuarios, se le llama **INSTANCIAR** y a los objetos creados se les llama instancia.
* Clase OrdendeCompra.

**4 PILARES**

* **ABSTRACCION**: USUARIO, cada clase tener métodos y atributos
* **ENCAPSULAMIENTO**: datos protegidos, privados, para no alterar cada clase.
* **POLIMORFISMO**: por ejemplo cambiar un estado, o comprobante de compra a través de slack, web y correo el método notificar(); las 3 tienen propias api lógicas y métodos. Dar la misma orden y que todos los objetos la cumplan a su manera
* **HERENCIA:** Clases padres y clases hijxs. Clase Usuario (Administración, Cliente, profesores)

**PROGRAMACION ORIENTADA A OBJETOS** [**https://www.youtube.com/watch?v=TGfRjZL6YfU**](https://www.youtube.com/watch?v=TGfRjZL6YfU)

**CLASE:** unión entre los datos y la operación o funcionalidad

 La clase es la esencia de Java. Es la base sobre la cual se construye todo el lenguaje Java porque **la clase define la naturaleza de un objeto**. Como tal, la clase forma la base para la programación orientada a objetos en Java.

Dentro de una clase se definen los datos y el código que actúa sobre esos datos. El código está contenido en métodos

<https://javadesdecero.es/poo/que-es-una-clase-ejemplos/>

**INSTANCIAS U OBJETOS:** EJEMPLOS DE UNA CLASE

**RELACIONES:**

* **Herencia: “**Es un”
* **Instancia:** Oposición “Tiene un”

1. **Encapsulamiento:** Asociar Dato y operaciones// método

**Ejemplo:** **Pato (nombre, color, energía – volar(), comer(), Caminar(),Dormir()**

Cliente (Nombre, Fecha, Rut) (pagar(), seleccionar(), ingresar()

Administrador (Nombre, datos personales) (IngresarStock(), ingresar fechas, visualizarpedidos, visualizar clientes(), ingresar stock reserva()

1. **Abstracción: Concepto o idea sin estar asociada a una instancia especifica la clase**
2. **Herencia:** Compartir características entre otros objetos similares. Animal superclase operaciones comer dormir comparten metodo
3. **Polimorfismo:** Muchas formas, mezcla de objetos ANIMAL (super clase) AVE (clase) pato, paloma, cisne, lista de aves pueden operar de manera conjunta. Cliente Invitado Registrado

**PRINCIPIOS SOLID** [**https://www.youtube.com/watch?v=5flyAn9IvAU&t=29s**](https://www.youtube.com/watch?v=5flyAn9IvAU&t=29s)

1. **Single Responsability (**Responsabilidad Unica)

Una clase debe tener pocas o una única responsabilidad

1. **Open/closed** (Abierto/cerrado)

Este principio promueve la reutilización. Abierto para extensión y cerrado para modificación. Ejemplo superclase vehículo y extendemos de esa clase sus métodos como acelerar() a las otras clases como motocicleta, camión, camioneta, etc.

1. **Liskov Substitution** (Sustitución de Liskov)

Debe ser capaz de cambiar una instancia de una clase por otra de un subtipo y el código debiese seguir funcionando bien.

1. **Interface Segregation** (Segregación de interfaces)

Mas interfaces es mejor.

1. **Dependency Inversion** (Inversión de dependencias)

Depender sobre abstracciones y no sobre instancias especificas. Una clase abstracta es Calendar y no se puede instanciar

**PRINCIPIOS SOLID**

**Producto**

**Attributos: nombre o id**

**Métodos: sumar(),descontar(),**

**relacion de herencia**

* **PanAmasado**
* **PanPita**

**Producto**

**Nombre  
Seleccionar()**

**Descontar()**

**Sub clase PanPita y PanAmasado**

* Diagrama

  Descripción generada automáticamente

**CONSTRUCTORES**

Un constructor es un método perteneciente a la clase que posee unas características especiales:

* Se llama igual que la clase.
* No devuelve nada, ni siquiera void.
* Pueden existir varios, pero siguiendo las reglas de la sobrecarga de funciones.
* De entre los que existan, tan sólo uno se ejecutará al crear un objeto de la clase.

**public**: indica que es un método accesible a través de una instancia del objeto.

**private**: indica que a través de una instancia no es accesible el método. Al heredar el método se convierte en inaccesible.

**protected:** indica que a través de una instancia no es accesible el método. Al heredar si se puede usar desde la clase derivada.

**Sin especificar**: indica visibilidad de paquete, se puede acceder a través de una instancia, pero sólo de clases que se encuentren en el mismo paquete.