Diagrama de clases

Preparación de parcial

El paradigma orientado a objetos.

Objetos y clases.

Características.

Propiedades.

Conceptos básicos.

Elementos fundamentales y secundarios de los objetos.

Teórico y Práctico.

El paradigma orientado a objetos.

Construcción de Sistemas:

* Más complejos
* Más confiables
* Más grandes
* En menos tiempo
* A menos costo

¿Cómo logramos manejar la **complejidad** del Software, reducir los **costos** de su construcción, y aumentar la **velocidad** del desarrollo?

**COMPLEJIDAD, COSTOS y VELOCIDAD**

* Construyendo el Software a partir de componentes reusables.
* Ensamblando el Software a partir de componentes de muchos proveedores.
* Creando una enorme biblioteca de componentes.

¿Qué es un paradigma?

* Paradigma es una forma de “**ver**” y “**entender**” el mundo, es una forma de abstraerlo dentro de nuestra cabeza.
* Thomas Khun definió la palabra paradigma como “un conjunto de **teorías**, **estándares** y **métodos** que juntos representan una forma de organizar el conocimiento, es decir, una forma de ver el mundo”.

Un **paradigma** es un **modelo o marco de referencia** que usamos para interpretar la realidad y organizar el conocimiento.  
Es como un **lente**: no cambia al mundo en sí, pero cambia **cómo lo percibimos, entendemos y explicamos**.

Un paradigma es **la forma en que un grupo de personas elige mirar, organizar y explicar la realidad**.  
No es la realidad en sí, sino **el marco mental que determina cómo la entendemos**.

¿Qué es un paradigma orientado a objetos?

* Componentes independientes y altamente reutilizables.
* Comunicación mediante colaboraciones.
* Bloques de construcción -> objetos y clases.
* Los objetos administran sus propios datos y son responsables de sus propias funciones.
* Ayuda a combatir la complejidad.

Se basa en la idea de **objetos**. Un objeto es como una "cajita" que junta dos cosas:

* **Datos** (atributos, propiedades, características).
* **Comportamientos** (métodos, acciones que puede hacer).

La clave es que en vez de pensar los programas como una lista de pasos (como en el paradigma imperativo), se los piensa como un conjunto de **objetos que interactúan entre sí**, enviándose mensajes, como si fueran actores en una obra de teatro.

Características de los objetos.

**“Los objetos son entidades del mundo real que combinan estado, comportamiento e identidad”**

• El comportamiento está definido por los métodos con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él.

• El estado representa uno o varios atributos a los que se habrán asignado unos valores concretos (datos).

• La identidad es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador.

* **Estado = qué datos tiene.**
* **Comportamiento = qué puede hacer.**
* **Identidad = quién es, lo que lo diferencia de los demás.**

Características de las clases.

* Una clase está compuesta por características (atributos o propiedades) y por comportamientos (acciones o métodos).
* Las características y comportamientos de una clase están determinados por el contexto del problema o escenario.
* A los valores que tienen los atributos de un objeto se los conoce como el estado que tiene el objeto, a los atributos y métodos que ofrece se les conoce como interfaz y al código utilizado para construir las clases se le conoce como la implementación de la clase.

Una **clase** es como la **receta** y los **objetos** son las **tortas hechas con esa receta**.  
La clase define los ingredientes (atributos) y pasos (métodos).

🔹 **Abstracción**  
La clase define lo esencial, dejando de lado lo irrelevante.  
Ejemplo: la clase *Auto* define “marca, color, velocidad, arrancar(), frenar()…”, pero no se fija en si tiene calcomanías en la luneta.

🔹 **Abstracción**

Es el orden de abstracción organizado por niveles.

🔹 **Encapsulación**  
En la clase se agrupan juntos los atributos (estado) y métodos (comportamiento).  
Ejemplo: la clase *CuentaBancaria* guarda dentro de sí el saldo y los métodos para depositar() o extraer(), y no se accede al saldo “a lo bruto”, sino a través de esos métodos.

* Ocultación de detalles.
* Implementación.
* Concepto de interfaz.
* Independencia.

🔹 **Modularidad**  
Cada clase es un módulo independiente, se puede desarrollar, probar o modificar por separado.  
Ejemplo: *Clase Cliente* y *Clase CuentaBancaria* pueden evolucionar por separado, siempre que respeten cómo se comunican entre sí.

Sistema descompuesto en módulos coherentes e independientes.

Alta cohesión, bajo acoplamiento.

La modularidad reduce la complejidad porque permite dividir un sistema grande en partes manejables, donde cada módulo tiene un propósito claro (alta cohesión) y depende lo mínimo posible de otros (bajo acoplamiento). Esto facilita mantenimiento, escalabilidad y reutilización.

🔹 **Herencia**  
Una clase puede derivar de otra y heredar sus atributos y métodos.  
Ejemplo: *Clase Auto* → *Clase AutoEléctrico* hereda todo de *Auto*, pero agrega batería y recargar().

* Tipo de jerarquía.
* Generalización/Especialización.
* Los objetos "heredan" las características de la clase.
* Las clases pueden heredar de otras clases.
* Principal utilidad en Abstracción y Reutilización.

🔹 **Reutilización**  
Las clases permiten crear múltiples objetos con las mismas características básicas.  
Ejemplo: de la clase *Persona* podés instanciar a *María, Juan, Ana…* todos con nombre, edad y hablar(), pero con distintos valores.

🔹 **Polimorfismo** (ligado a herencia)  
Una clase hija puede redefinir comportamientos de la clase padre.  
Ejemplo: *Clase Ave* tiene método volar(), pero *Clase Pingüino* redefine volar() para que no funcione igual.

Beneficioso para la reutilización del código.

El polimorfismo es la propiedad de los objetos de distintos tipos de comportarse de manera diferente ante un mismo mensaje o llamada a un método, permitiendo la reutilización y flexibilidad del código

**🔹 Los conceptos básicos del modelo OO:**

* **Objeto**
* Es la entidad concreta (estado + comportamiento + identidad).
* Ejemplo: *el auto rojo de Juan*.
* **Clase**
* Es el molde del que salen los objetos.
* Ejemplo: *la receta Auto* que define atributos (color, marca, velocidad) y métodos (arrancar(), frenar()).
* **Herencia**
* Permite crear nuevas clases a partir de otras, heredando y especializando comportamientos.
* Ejemplo: *AutoEléctrico* hereda de *Auto*, pero agrega batería y recargar().
* **Envío de mensajes**
* Es la forma en que los objetos se comunican.
* Un objeto le “manda un mensaje” a otro para que ejecute uno de sus métodos.
* Ejemplo: el objeto *Conductor* envía al objeto *Auto* el mensaje arrancar().

**🔹 Parte estructural (estática)**

* **Objeto, Clase, Herencia**  
  Son los ladrillos con los que armamos el sistema. Definen la “estructura” de cómo se organiza el software.

**🔹 Parte dinámica (comportamiento)**

* **Envío de mensajes**  
  Es cómo los objetos **interactúan y colaboran** en tiempo de ejecución, dando vida al sistema.

**🔹 Elementos secundarios.**

* **Tipificación:** Características precisas que comparten una serie de objetos. Puesta en vigencia de la clase de los objetos.
* **Concurrencia:** Permite a diferentes objetos actuar al mismo tiempo. Distingue objetos activos de inactivos.
* **Persistencia:** Cantidad de espacio que ocupa y tiempo que dura un objeto. Conservación del estado del objeto en el espacio y en el tiempo.

**La propiedad Abstracción me permite representar lo esencial de un objeto ocultando lo irrelevante. Encapsulamiento se relaciona con ocultar datos y exponerlos mediante métodos controlados, modularidad divide en módulos y polimorfismo es la respuesta distinta a un mismo mensaje.**

Términos fuertes sacados del libro.

Las clases son bloques de construcción de UML.

Las clases no son objetos individuales, representan un conjunto de objetos.

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

Las clases son: Cosas tangibles, Roles de personas, Lugares, Transacciones, Eventos, Organizaciones o áreas.

Un objeto posee y combina un estado, comportamiento e identidad.

Un atributo es una propiedad de una clase identificada con un nombre que describe un rango de valores que pueden tomar las instancias de la propiedad.

Una operación es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento.

A menudo, la invocación de una operación sobre un objeto cambia los datos o el estado del objeto.

En la implementación de atributos y operaciones en una clase, estos pueden esconderse del usuario y solo mostrar los más "importantes".

Una responsabilidad es un contrato o una obligación de una clase. Por ej: la clase "pared" está obligada a saber sobre su anchura, alto y densidad.

El vocabulario del sistema son las abstracciones extraídas del problema que se intenta solucionar, cada una de ellas representa el conjunto de los elementos que son importantes para los usuarios.

RELACIONES

Una relación es una conexión entre elementos.

Dependencia --> Representan relaciones de uso entre clases. EJ: Las tuberías depende del calentador para calentar el agua que conducen.

Generalizaciones --> Conectan clases generales con sus especificaciones. Se conoce como relaciones superclase/clase o hijo/padre, relación entre un elemento general y un caso más específico de ese elemento. A veces se llama relación "es-un-tipo-de".

Asociaciones --> Representan relaciones estructurales entre objetos. EJ: las habitaciones constan de paredes y otras cosas; las paredes a su vez pueden contener puertas y ventanas, las tuberías pueden atravesar las paredes. Especifica que los objetos de un elemento están conectados con los objetos del otro.

La multiplicidad representa un rango de enteros que especifican el tamaño posible del conjunto de objetos relacionados. De cero a muchos (0..\*), de uno a muchos (1..\*), de cero a uno (0..1) o un número exacto, por ejemplo, tres (3..3).

Se denomina relación de AGREGACION cuando se desea moldear una relación "todo/parte" en la cual una clase representa una cosa grande(todo) que consta de elementos más pequeños(parte). Representa una relación del tipo "tiene un…" o sea un objeto TODO tiene objetos de la PARTE.

Una nota es un símbolo grafico para representar restricciones o comentarios asociados a un elemento o a una colección de elementos.

Un patrón describe la estructura interna y la relación entre dos clases.

Un patrón no busca lo particular, sino que destaca la esencia.

Los patrones son soluciones expresadas en términos de objetos e interfaces se expresan en base a la estructura y/o comunicación de objetos y clases para resolver los problemas del contexto.

Componentes de un patrón:

1. Nombre
2. Propósito
3. Sinónimo
4. Colaboraciones
5. Contexto
6. Explicación
7. Fuerzas
8. Motivación
9. Ejemplos
10. Patrones relacionados
11. Aplicabilidad
12. Estructura
13. Participantes
14. Consecuencia

**Patrón Fundamental**: Colección – Trabajador. Es la plantilla que todos los patrones siguen.

**Patrones Transaccionales**: Tienen un jugador de transacción o tienen jugadores que comúnmente juegan con un jugador de transacción.

* Actor – Participante
* Participante – Transacción
* Lugar – Transacción
* Ítem especifico – Transacción
* Transacción – Detalle de transacción
* Transacción – Transacción subsiguiente
* Detalle de transacción – Detalle de transacción subsiguiente
* Ítem – Detalle de transacción
* Ítem especifico – Detalle de transacción
* Ítem – Ítem especifico
* Asociación – Otra asociación
* Ítem especifico – Jerarquía de ítem

**Patrones de Agregación**: Se interconectan con otros Patrones, algunas veces con la ayuda de Patrones transaccionales “Asociación – Otra asociación”

* Contenedor – Contenido
* Contenedor – Detalle de contenedor
* Grupo – Miembro
* Todo – Parte
* Parte de compuesto – Parte
* Paquete – Componente de paquete

**Patrones de Plan:**

* Plan – Paso
* Plan – Ejecución de plan
* Plan – Ejecución – Paso de ejecución
* Paso – Paso de ejecución
* Plan – Versión de plan