IDAÑEZ, LUCIA MARÍA

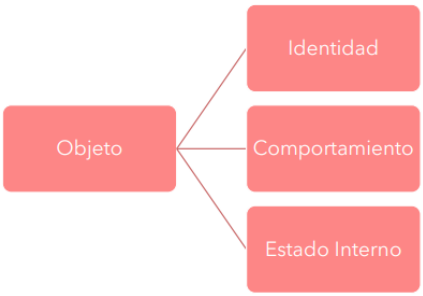
Paradigma Orientado a Objetos

El **paradigma orientado a objetos** nos propone pensar nuestras soluciones basadas en el mundo de objetos. Este paradigma nos hace pensar en la realidad, estudiarla, entenderla, conceptualizarla y luego modelarla; muchas veces puede suceder que nuestros modelos sean representaciones abstractas de nuestra realidad.
A su vez, nos hace pensar en la mantenibilidad, flexibilidad, cohesión y acoplamientos y otras cualidad y atributos de calidad que se buscan maximizar constantemente.
→Nos ahorra pensar en cuestiones de más bajo nivel
→Está a favor de la reutilización de código

Acoplamiento: es el grado en el que dos o más componentes o clases dependen unas de otras. Un buen diseño o una buena arquitectura tratará de mantener un nivel de acoplamiento bajo.

Objeto

Un **objeto** es una representación computacional de un ente (una abstracción con razón de ser) que exhibe comportamiento. Tiene identidad, puede tener un estado interno y expone una interfaz (comportamiento). Instancia de una clase.



Un objeto es una abstracción que tiene una razón de ser porque tiene una o varias responsabilidades asociadas; estas responsabilidades quedan en evidencia cuando alguien, otro actor, le pide algo al objeto.
Un objeto define un TIPO DE DATO, esto quiere decir que el tipo de una variable puede ser **objeto**.

IDENTIDAD

La **identidad** es la propiedad que permite diferenciar a un objeto y distinguirse de otros.
Se considera que dos objetos son idénticos si son el mismo, es decir que si las referencias están apuntando a la misma dirección de memoria.

COMPORTAMIENTO

El **comportamiento** es el conjunto de mensajes que entiendo un objeto. Este mismo está ligado con la responsabilidad del objeto que le da sentido a su existencia.
El **comportamiento de un objeto** está ligado a qué hace un objeto cuando recibe un mensaje, esto quiere decir que dos objetos pueden tener el mismo mensaje y, sin embargo, hacer cosas distintas.

ESTADO INTERNO

El **estado interno** es el conjunto de atributos que tiene el objeto. Como los atributos son variables, también se puede decir que el estado interno es el conjunto de variables que contiene el objeto.

MENSAJES Y MÉTODOS

Los **mensajes** son la forma que tiene los objetos de comunicarse entre sí y, cuando un objeto recibe un mensaje, ejecuta el **método** asociado a ese mensaje.
En un método se especifica la lógica que va a ejecutar el objeto cuando reciba el mensaje asociado.

ATRIBUTOS

Los **atributos** son las características que le pertenecen a un objeto y lo diferencian del resto, estas características se representan mediante variables. Un objeto puede tener uno o más atributos que conformaran su estado interno.

->Un atributo no se guarda en memoria, sino que hace referencia a otro objeto.

RESPONSABILIDAD

Cada objeto debe tener una **responsabilidad** lo cual será la razón de ser de ese objeto, sin embargo, un objeto no debería tener muchas responsabilidades asociadas, ya que eso provocaría que no sea mantenible y, a su vez, esto es un indicio de que no estamos modelando bien nuestro dominio y tengamos abstracciones por resolver.

ENCAPSULAMIENTO

El **encapsulamiento** hace referencia a que un objeto no debe permitir que otros objetos modifiquen sus atributos.

IDAÑEZ, LUCIA MARÍA

Este encapsulamiento se rompe cuando un objeto externo modifica los atributos de otro objeto directamente (ej.: por seteo de un valor), también se rompe cuando se "sabe" muchas cosas de otro objeto, esto quiere decir cuando se comienza a "preguntar" específicamente, en vez de directamente. Interactuar con un objeto mediante una anidación de mensajes también es romper el encapsulamiento.

DECLARATIVIDAD

La **declaratividad** hace referencia a ocultar el detalle algorítmico, es decir que no me importa el cómo sino el qué, implica dejar de lado el pensamiento imperativo y algorítmico para concentrarnos en aquello que necesitamos.
En el paradigma orientado a objetos este concepto hace referencia a pensar qué objetos necesitamos para resolver el dominio presentado, qué mensajes deben entender esos objetos y qué responsabilidades deberían tener.

DELEGACIÓN

Pateamos responsabilidades al mismo o a otro objeto.

CLASE

Las **clases** son **MOLDES** que abarcan a objetos que tienen características similares como comportamiento (interfaz) y atributos, por lo tanto, para no repetir lógica/código puedo agruparlos en clases.
Una **clase** nos sirve para modelar las abstracciones de mi dominio (conceptos que nos interesan), permitiéndome definir el comportamiento y atributos de las instancias. Por lo tanto, aquellos objetos que responden a un mensaje de la misma manera y que tiene igual estructura interna, se los clasifica de la misma "clase".

INSTANCIA

Al proceso de crear un objeto a partir de una clase se lo conoce como instanciación y decimos que el objeto obtenido como resultado de este proceso es una instancia de esa clase, esto quiere decir que **un objeto es una instancia de una clase**.

CLASE ABSTRACTA

Las **clases abstractas** (las SUPER CLASES) en particular no las queremos instanciar, ya que no nos interesan, aquello que nos interesa son las clases que heredan de ellas dado que, a partir de estas, instanciaremos objetos.
Las **clases abstractas** nos sirven para extraer comportamiento a partir de ellas. Dentro de las clases abstractas habrá métodos obligatorios para todas aquellas clases que hereden de ellas, estos métodos se denominan **MÉTODOS ABSTRACTOS**. Solo las clases abstractas pueden tener métodos abstractos.

POLIMORFISMO

El **polimorfismo** es la capacidad que tiene un objeto de poder hablar indistintamente a otros objetos que sean distintos; un objeto trata polimórficamente a otros objetos cuando les envía los mismos mensajes y estos pueden entender y responder ya que tienen una interfaz común.
Para que dos objetos sean polimórficos deben entender el mismo mensaje y que haya un t
indistintamente.

> Una clase concreta no es abstracta, las **clases concretas** se pueden instanciar

INTERFACE

La **interface** es el conjunto métodos (solo las firmas del método, no su comportamiento/cuerpo/desarrollo) que posee un objeto, dado que a través de ellos podremos interactuar con el objeto mediante mensajes.
Un objeto puede implementar una, muchas o ninguna interface.
La **INTERFAZ** de un objeto define el **TIPO** del objeto.

COLECCIONES

Las **colecciones** son objetos que nos permiten agrupar/contener y manejar un conjunto de objetos.
Las colecciones tienen definidos un conjunto de métodos, los cuales ellas entienden, y les permite manipular su contenido.
Si bien una colección es un conjunto de objetos, es mejor pensar que es un conjunto de referencias, ya que la colección no tiene "adentro suyo" a los elementos, sino que la colección los conoce mediante una referencia.

IDAÑEZ, LUCIA MARÍA

Diagrama de Clases

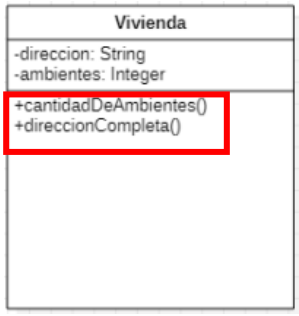
Un **diagrama de clases** (UML) es un diagrama estático que muestra y describe una parte de la estructura de un sistema mediante la representación de las clases con los atributos y los métodos más importantes y, a su vez, cómo se relacionan las clases entre sí.
No existe un único diagrama de clases para representar un sistema, esto depende de la arquitectura del sistema.

Clase

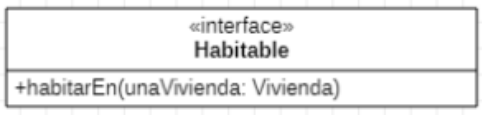
Atributos

+ public → público, todos pueden ver y acceder al método/atributo
– private → privado, solo de la clase donde se crea puede acceder al método/atributo
protected → protegido, la misma clase y sus clases hijas puede acceder al método/atributo

Mensajes/Métodos



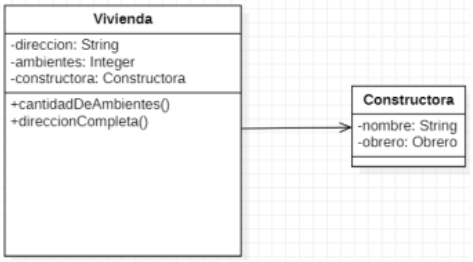
Interfaces



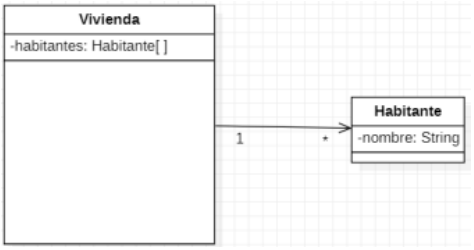
Relaciones

Asociación simple dirigida

Una clase A tiene como atributo (un atributo objeto) a un objeto de clase B.
Se lee “A tiene un B”.



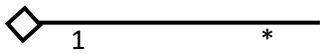
También se puede especificar la multiplicidad de las relaciones con un número o utilizando “*” en caso de que sea una colección.



IDAÑEZ, LUCIA MARÍA

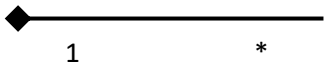
Agregación

Relación de tipo Todo-Parte, el objeto “parte” forma parte de uno o varios objetos Todo, pero aun así debe tener existencia en sí mismo.



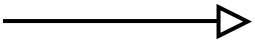
Composición

Asociación donde un objeto “parte” existe si y solo si forma parte de un objeto Todo y ese objeto Todo existe. En este caso, los elementos que forman parte no tienen sentido de existencia cuando el Todo no existe



Generalización – Herencia

Se utiliza cuando una clase hereda de otra



Dependencia- Uso

Se utiliza para representar que una clase requiere de otra para ofrecer cierta funcionalidad.

Realización- Implementación

Se utiliza para representar que una clase Implementa una Interface