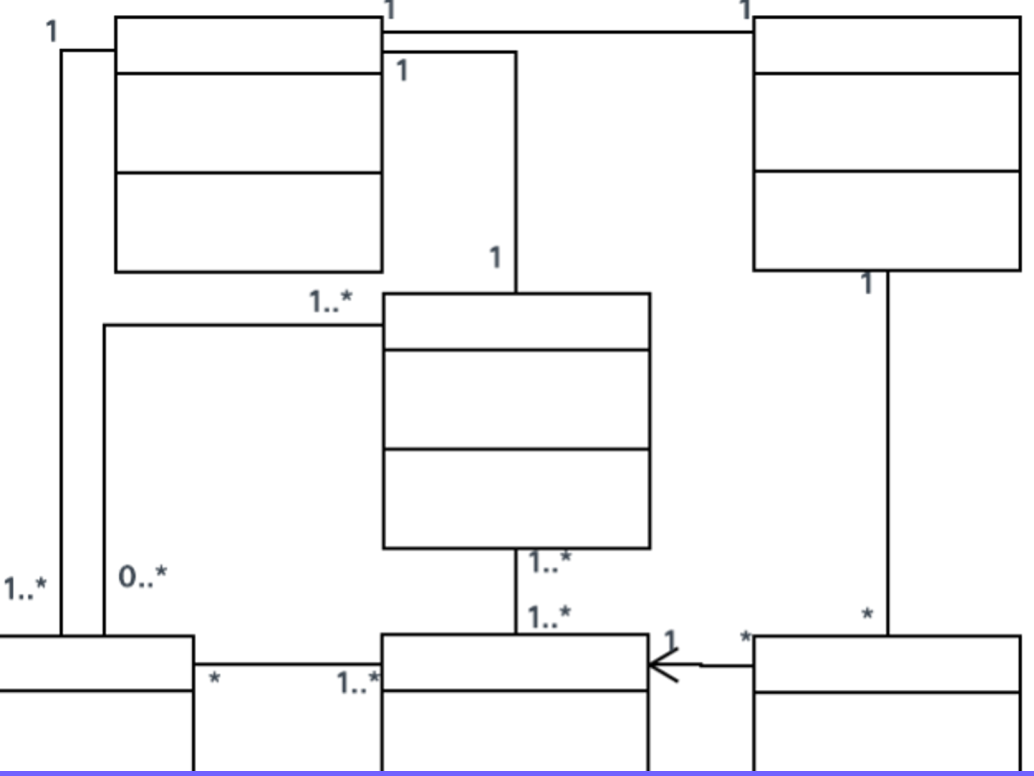
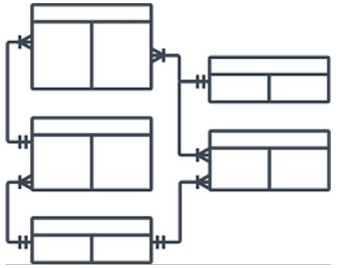
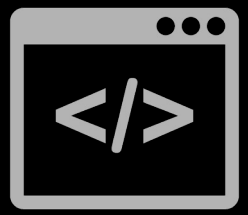
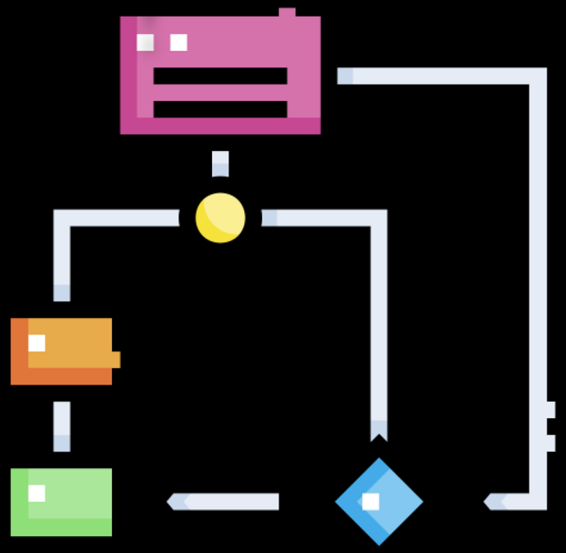




Diseño de Sistemas

Paradigma Orientado a Objetos





UNIFIED MODELING LANGUAGE™

Paradigma Orientado a Objetos

- El paradigma orientado a objetos nos hace pensar en la realidad, estudiarla, entenderla, conceptualizarla y luego modelarla. Claro está que nuestro entendimiento de la realidad está limitado y acotado porque nuestra mirada es totalmente subjetiva.

- Este paradigma nos hace replantear varias cuestiones sobre cómo diseñar y desarrollar un sistema.

Paradigma Orientado a Objetos

- Nos hace pensar en la mantenibilidad, flexibilidad, cohesión, y muchos otras tantas cualidades y atributos de calidad que se buscan maximizar constantemente.

- Nos abstrae de pensar en cuestiones de más bajo nivel como reserva dinámica de memoria, liberación de memoria o sentencias muy primitivas.

- Está a favor de la reutilización de código y cree en que las cosas deben hacerse una sola vez.

POO - Objeto

¿Qué es un objeto?

POO - Objeto

- Para este paradigma, un objeto es *toda "cosa" del mundo real, o no, que puede representarse computacionalmente*.

- Decimos también que 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 solicita a un objeto que haga algo o le pide algo.

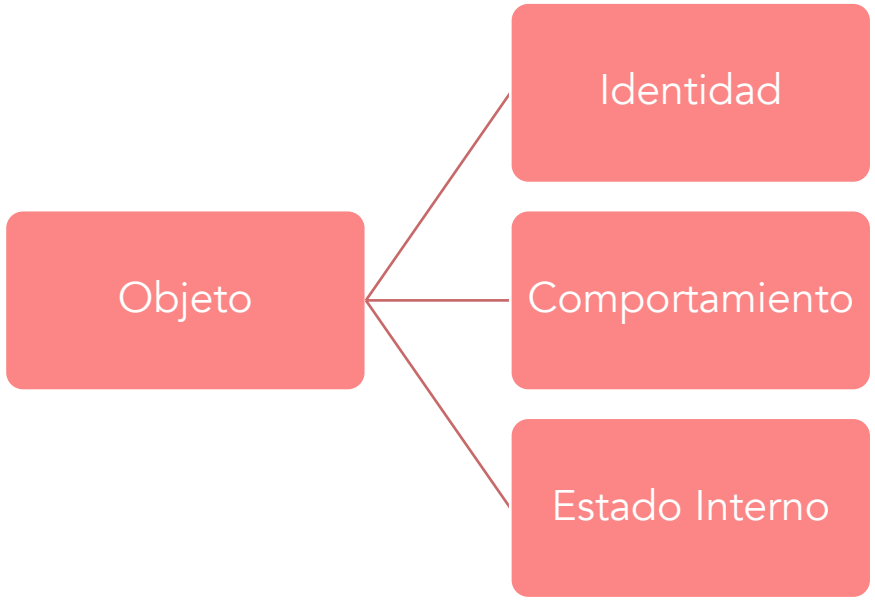
POO - Objeto

- Además, cada objeto tiene características que le pertenecen y que lo diferencian del resto.

- Es muy importante saber que un objeto define un tipo de dato, lo que significa que el tipo de una variable puede ser un objeto.

POO - Objeto

Todo objeto cuenta con tres características que lo diferencia del resto:



POO - Objeto

Identidad

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

POO - Objeto

Comportamiento

- En la jerga cotidiana se suelen escuchar preguntas como: ¿cuál es el comportamiento de este objeto? ¿qué hace? ¿para qué existe? Al conjunto de mensajes que entiende un objeto se lo conoce como comportamiento.

- El comportamiento que tiene un objeto está ligado con la responsabilidad que le dá sentido de existencia.

POO - Objeto

Estado interno

- Al conjunto de atributos que tiene un objeto se lo conoce como estado interno. Y como los atributos son variables, también podríamos afirmar que el estado interno es el conjunto de variables que contiene un objeto.

POO – Mensajes y Métodos

- Hemos dicho que un objeto puede tener una o varias responsabilidades asociadas y que éstas quedan en evidencia cuando alguien le dice al mismo que haga algo o bien cuando le pide algo.

- ¿Quién puede decirle a un objeto que haga algo o pedirle algo? ¡Otro objeto!

- Decimos entonces que ***los objetos se comunican enviándose mensajes*** entre sí.

POO – Mensajes y Métodos

- Como en todo proceso de comunicación, existe un emisor, un mensaje y un receptor.
- ***Cuando un objeto recibe un mensaje, ejecuta el método asociado***.
- En el método se especifica lo que el objeto debe hacer frente a esa petición. Por ejemplo, si yo le digo a un gato que camine, el gato se desplazará un par de metros.

POO – Atributos

- Hemos dicho que un objeto tiene características que le pertenecen y que lo diferencian del resto.
- Estas características se materializan en variables, más conocidas como atributos.
- Es decir, un objeto puede tener una o más variables, desde ahora y para siempre atributos, que conforman su estado interno.
- Estos atributos sobreviven mientras el objeto sobreviva, es decir, mientras el mismo esté presente en la memoria.

POO – Atributos

- Cabe destacar que un atributo no está haciendo referencia a una porción de memoria, sino que está guardando una referencia a otro objeto.

POO – Responsabilidad

- Cuando vamos a un kiosko a comprar golosinas, ¿qué deberíamos hacer para conseguirlas? Siendo respetuosos, deberíamos pedírselas al kioskero.
- El kioskero debería agarrar aquellas golosinas que le hayamos indicado y luego debería cobrarnos.
- Una vez que entreguemos el dinero, el kioskero nos debería dar la bolsita con las golosinas.

Si tuviéramos que modelar este pequeño dominio (pensando en el mundo real), ¿qué objetos identificaríamos?

POO – Responsabilidad

- En principio, Cliente, Kioskero y Golosina (sin importancia por el momento).

¿Y qué responsabilidad tendría cada uno?

- El Cliente tendría la responsabilidad de pedirle al kioskero una o varias golosinas.
- El kioskero debería buscar las golosinas y luego cobrarle al cliente.
- El cliente debería pagar por la compra.

¿Qué acabamos de hacer?

POO – Responsabilidad

- Establecimos las responsabilidades principales a nuestros objetos.
- Cada objeto debería tener una responsabilidad asociada, una razón de ser, algo que resolver.
- Un objeto no debería tener muchas responsabilidades asociadas porque ello llevaría a que se vuelva inmantenible.
- Si un objeto tiene muchas responsabilidades, probablemente no hayamos modelado bien el dominio y todavía tengamos más abstracciones por encontrar.

POO – Encapsulamiento

- Este término hace referencia a que un objeto no debería saber cómo está implementado otro objeto que conoce.

- En otras palabras, un objeto A debería solamente conocer los mensajes que puede enviarle a un objeto B, pero no debería saber la implementación de los métodos que se ejecutan cuando le envía un mensaje.

POO – Encapsulamiento

- Tampoco se debería dar a conocer el estado interno de un objeto. Un objeto A no debería acceder directamente a los atributos de un objeto B, porque si lo hiciera de esa forma, A estaría conociendo exactamente las cosas que tiene B.

- Las formas posibles interactuar con un objeto deberían ser acotadas y explícitas y es por eso que el objeto debe definir qué mensajes se deberían utilizar para interactuar con él.

POO – Declaratividad

- Una primera, y suficiente por ahora, aproximación a este concepto nos dice que ***debemos pensar en el qué y no en el cómo***.

- Deberíamos dejar de pensar los problemas algorítmicamente, imperativamente, y comenzar a concentrarnos en lo que realmente necesitamos.

- Para aplicar este concepto en el paradigma orientado a objetos deberíamos pensar qué objetos necesitamos para resolver el dominio presentado, qué mensajes deberían entender, qué responsabilidades deberían tener.

POO – Delegación

Este término es conocido como "***patear la pelota***" y viene de la mano con la declaratividad, la cual nos pide que pensemos en el qué y no en el cómo. Las preguntas que deberían surgirnos son:

- ¿Qué es patear la pelota? Patear la pelota significa hacer las cosas (calcularlas/obtenerlas/pedirlas) cuando realmente las necesitamos.

- ¿A quién se la pateamos? Puede ser al mismo objeto u otro objeto.

- ¿Qué le pateamos? ¡La pelota! ¿Pero qué es la pelota? Como dijimos, todo lo que necesitemos calcular, hacer, pedir. ***Pateamos responsabilidades***.

POO – Clase

- ¿Qué sucede si muchos de mis objetos de dominio se comportan igual y tienen características iguales?
- Probablemente estemos repitiendo exactamente el mismo código y/o la misma lógica en todos los objetos.
- Una forma de evitar esto es crear moldes para nuestros objetos, o más conocido como Clases.
- Las clases nos permiten definir moldes para nuestros objetos, para que todos ellos se comporten de la misma manera.
- ¿Qué es lo que queremos que compartan nuestros objetos?

POO – Clase

- Gráficamente podríamos decir que una clase es el contorno de la figura y un objeto es el relleno de ese contorno.
- Una clase es una forma de clasificar a los objetos que "se parecen".
- Los objetos que responden a los mismos mensajes de igual manera y que tienen una estructura interna igual (los mismos atributos, con diferente valor para cada objeto), se clasifican juntos en una determinada "clase".

POO – Instancias

- Hemos dicho que las clases son los moldes de los objetos, pero la gran pregunta es: ¿cómo obtengo un objeto a partir de una clase?

- 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.

- Entonces, de forma resumida, podríamos decir que ***un objeto es una instancia de una clase***.

POO – Instancias

¿Pero cómo se hace?

Instanciemos, por ejemplo, a Gary. Gary será una instancia de la clase Gato:

```
Gato gary = new Gato();
```

Esta sentencia es válida en Java, pero la realidad es que en muchos lenguajes también se usa la palabra ***new*** para pedir una instancia de una clase.

POO – Clase Abstracta

- Una clase abstracta es una clase que no puede ser instanciada.
- Decimos que las clases abstractas tienen clases hijas, o bien que algunas clases heredan de ella.
- Estas clases, las hijas, sí pueden ser instanciadas siempre y cuando sean concretas.
- Una clase es concreta cuando no es abstracta.

POO – Clase Abstracta

¿Para qué sirven?

- Así como evitamos repetir código y lógica entre los objetos cuando muchos de ellos tenían el mismo comportamiento generando clases, ahora podemos evitar repetir código y lógica entre clases haciéndolas heredar de una clase abstracta en común.

- En las clases abstractas podemos escribir el código en común que tengan muchas clases, ya sea métodos o atributos.

POO – Clase Abstracta

Es importante tener en cuenta que en la mayoría de los lenguajes solo se soporta la herencia simple. Se dice que un lenguaje soporta herencia simple cuando solamente permite que una clase herede de una sola clase, es decir, permite que una clase tenga como máximo un padre.

POO – Polimorfismo

- Es una característica que nos permite trabajar con objetos distintos en forma transparente.
- Decimos que dos o más objetos son polimórficos para un tercero si éste puede trabajar indistintamente con cualquiera de ellos.
- Una forma distinta de definir el polimorfismo es decir que dos o más objetos son polimórficos si entienden los mismos mensajes.

POO – Polimorfismo

Lo cierto es que dependiendo del tipo de lenguaje en el que estemos programando veremos este concepto de forma diferente.

Por ejemplo, sabemos que Java es un lenguaje de fuertemente tipado y, por lo tanto, debemos definir el tipo de cada parámetro en un método. ¿Cómo hacemos para definir un tipo y aplicar el concepto de polimorfismo si lo que puedo recibir son muchos objetos distintos?

Una estrategia podría ser definir una interface y hacer que varias clases implementen la misma. Para el método en cuestión será transparente recibir cualquier objeto, siempre y cuándo cumplan con dicha interface.

POO – Interface

- Una interface es un contrato en el cual se establecen los métodos que debe implementar un objeto.
- En la interface solo se escriben las firmas de los métodos, es decir, sin cuerpo ni desarrollo de los mismos.
- Un objeto puede implementar ninguna, una o muchas interfaces al mismo tiempo. Además, es importante destacar que una interface define un nuevo tipo de dato.
- En general, cuando objeto implementa una interface se dice que cumple con esa interface.

POO – Interface

Por ejemplo, supongamos que establecemos la interface desplazable que define el método desplazar(metros). Podrían existir varios objetos que cumplan con esa interface, como insectos, animales, personas, vehículos, etc. Todos estos objetos, si implementan la interface, estarían obligados a desarrollar el método desplazar(metros), pero cada uno lo hará de forma distinta. Una persona podría caminar, una serpiente se podría arrastrar, una mariposa podría volar, etc.

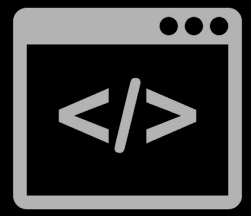
POO – Colecciones

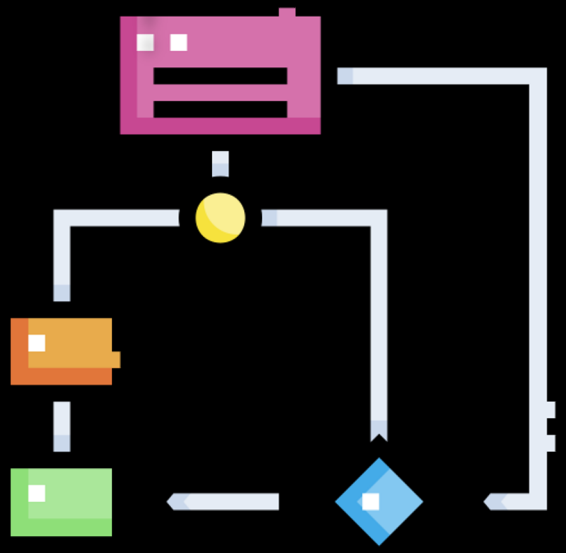
- Una colección es un objeto que dentro suyo contiene referencias a otros objetos.
- Su responsabilidad es contener y manejar un grupo de objetos.
- Las colecciones, por lo general, pueden contener objetos de distintas clases siempre y cuando compartan al menos una interface.
- Las colecciones varían en las tareas que pueden realizar y en los mensajes que saben responder, por lo tanto, hay distintas clases de colecciones, aunque hay algunas tareas que las puede realizar cualquier colección.

POO – Colecciones

- Hay colecciones que tienen organizados sus elementos.
- Esta organización puede consistir en un índice numérico que permite acceder a una posición determinada de la colección, una clave de acceso más compleja para búsqueda directa o tener algún criterio de orden interno.
- También tienen definidos un abanico amplio de métodos que permiten manipular su contenido evitando el uso de estructuras de control adicionales.

Diagrama de Clases





UNIFIED MODELING LANGUAGE™

Diagrama de Clases

Un diagrama de clases (UML) es un diagrama estático que muestra y describe cierta porción de la estructura de un sistema mostrando las clases con sus atributos, sus métodos y sus relaciones con otras clases.

Diagrama de Clases

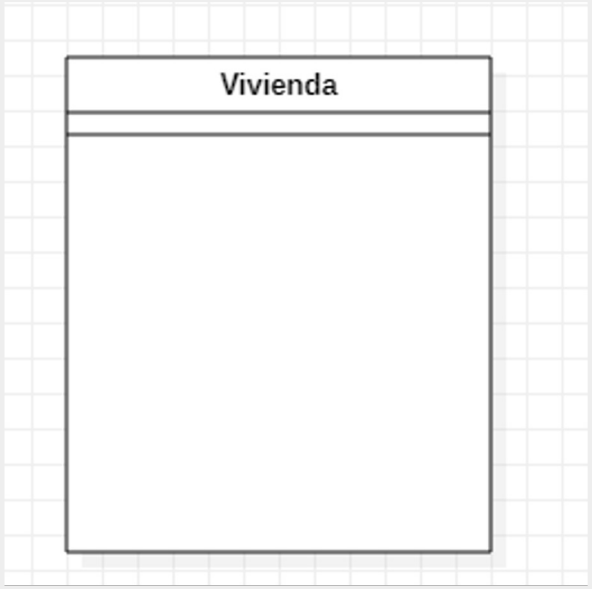
Es importante darse cuenta que no necesariamente existe un único diagrama de clases para representar un sistema; sino que pueden existir varios.
Esto dependerá de la Arquitectura del Sistema.

Diagrama de Clases

Por ejemplo, -casi- toda aplicación mobile cuenta con un backend que contiene -casi- toda la lógica de negocio. La aplicación mobile, en este caso, tendrá su propio diagrama de clases que muestre y exponga su estructura; así como también lo tendrá el backend, si es que éste está diseñado e implementado bajo el paradigma orientado a objetos.

Diagrama de Clases

Clases



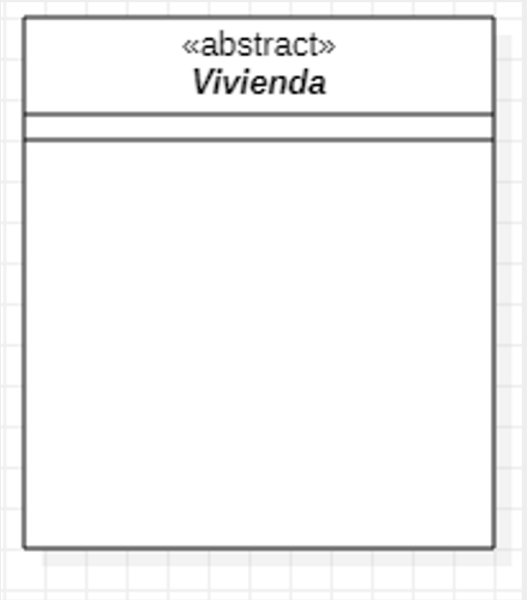
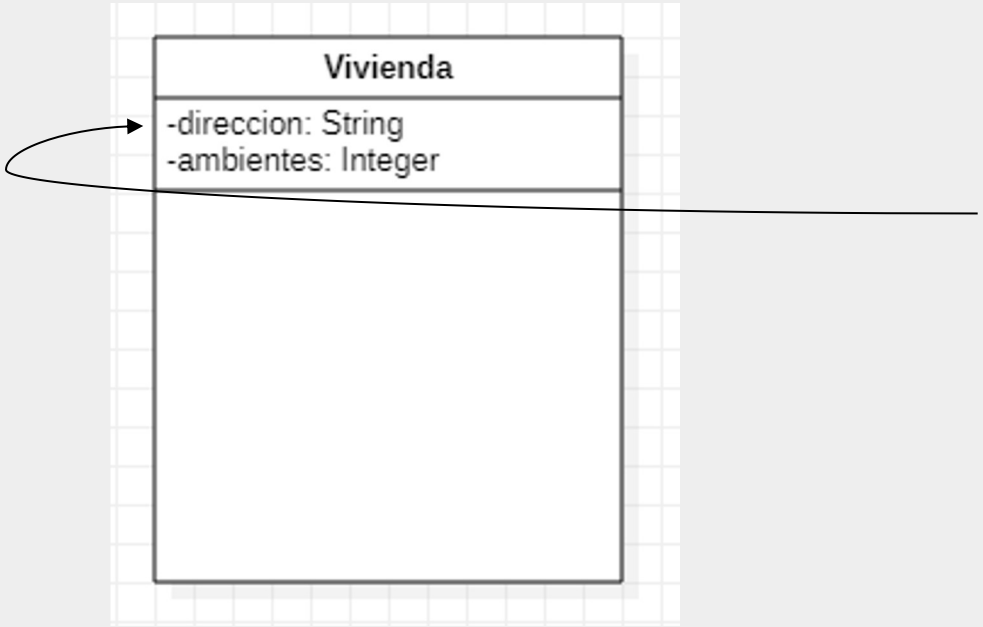


Diagrama de Clases

Atributos



+	-	#
• Público • Todos pueden ver y acceder al atributo/método	• Privado • Únicamente la misma clase puede ver y acceder al atributo/método	• Protegido • La misma clase y las clases hijas pueden ver y acceder al atributo/método

Diagrama de Clases

Mensajes/Métodos

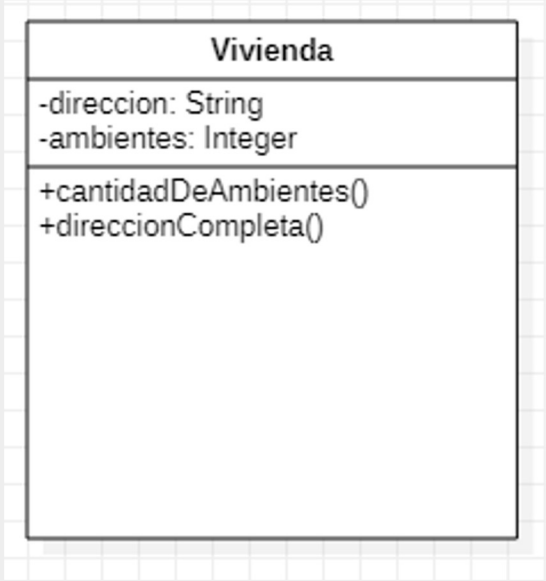


Diagrama de Clases

Interfaces

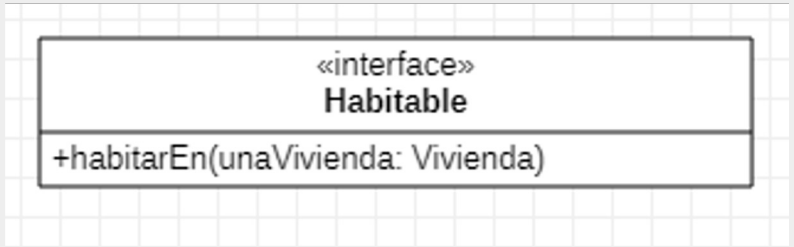


Diagrama de Clases

Relaciones – Asociación Simple Dirigida

- Una clase A tiene como atributo (de objeto) a un objeto de la clase B.
- Se suele leer *"A tiene un B"*
- En el ejemplo, "*Una vivienda tiene una constructora*"

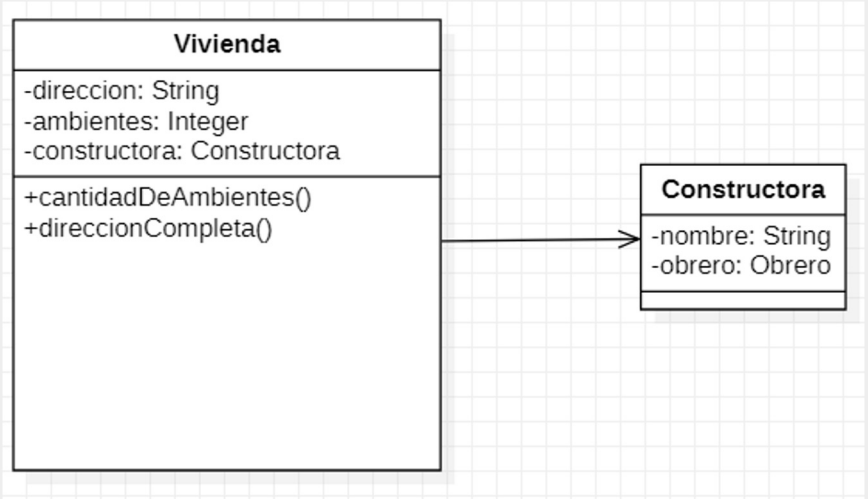


Diagrama de Clases

Relaciones – Asociación Simple Dirigida

- Se puede especificar la multiplicidad de las relaciones ya sea dejando explícito el número o utilizando un "*" en caso de que sea una colección.
- En el ejemplo, "*Una vivienda tiene (o puede tener) muchos habitantes, y un habitante solamente vive (o puede vivir) en una única casa*".
- "*Una vivienda tiene una colección de habitantes*"

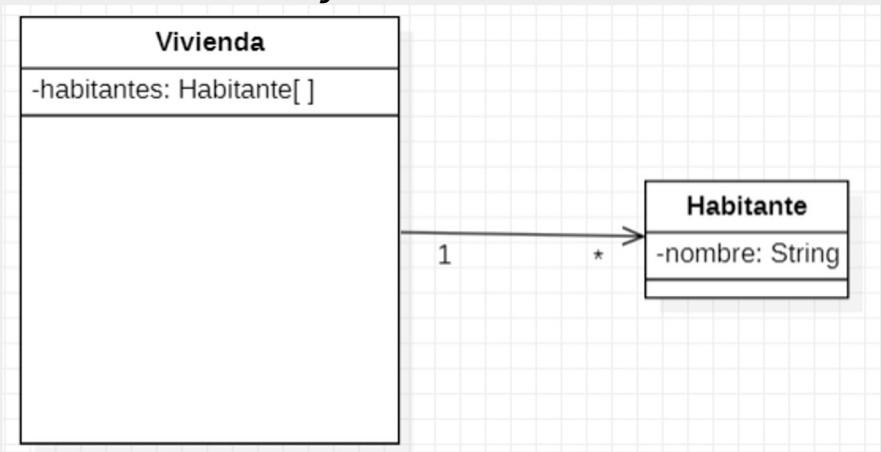


Diagrama de Clases

Relaciones – Agregación

- Es una representación jerárquica que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que un objeto es parte de otro, pero aun así debe tener existencia en sí mismo.

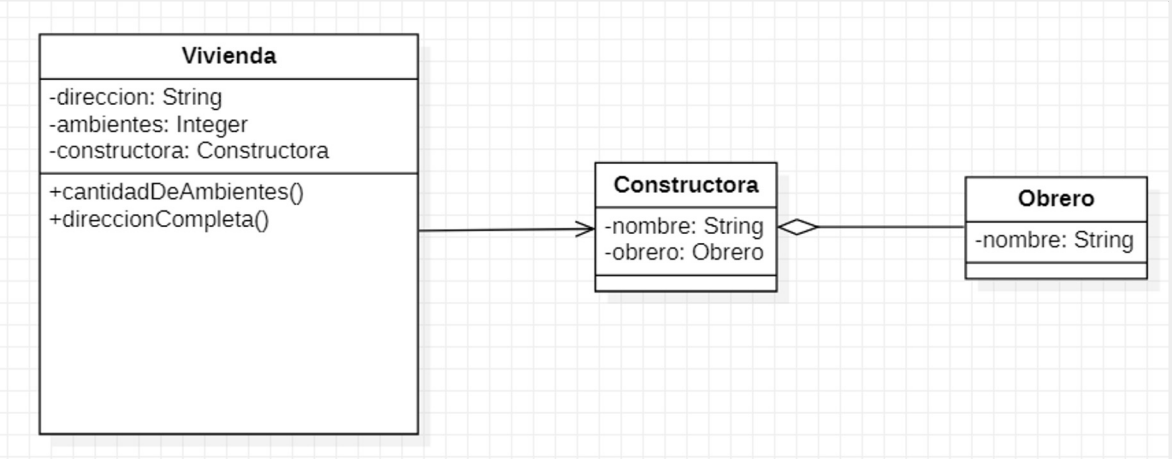


Diagrama de Clases

Relaciones – Composición

- La composición es similar a la agregación, representa una relación jerárquica entre un objeto y las partes que lo componen, pero de una forma más fuerte. En este caso, los elementos que forman parte no tienen sentido de existencia cuando el primero no existe.

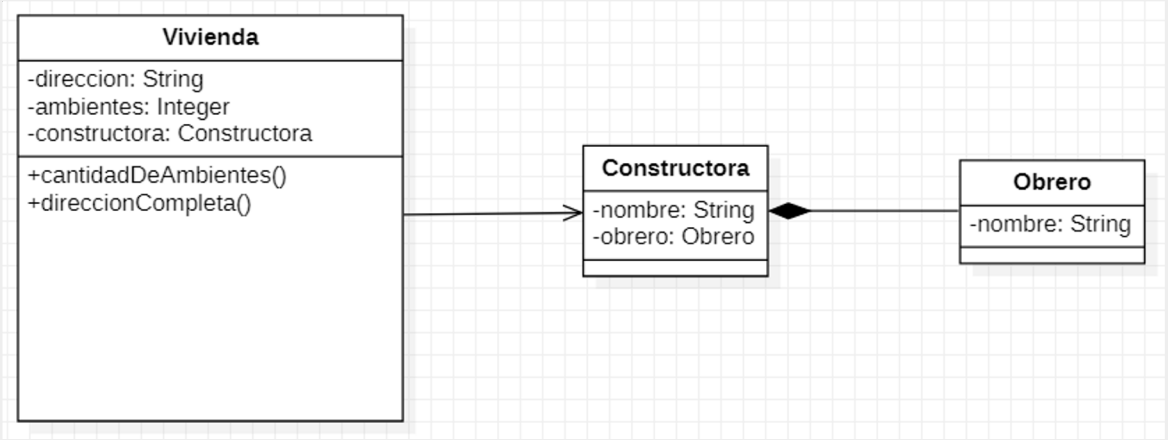


Diagrama de Clases

Relaciones – Generalización ("Herencia")

- Se utiliza cuando una clase hereda de otra
- Se suele leer "*A hereda de B*"

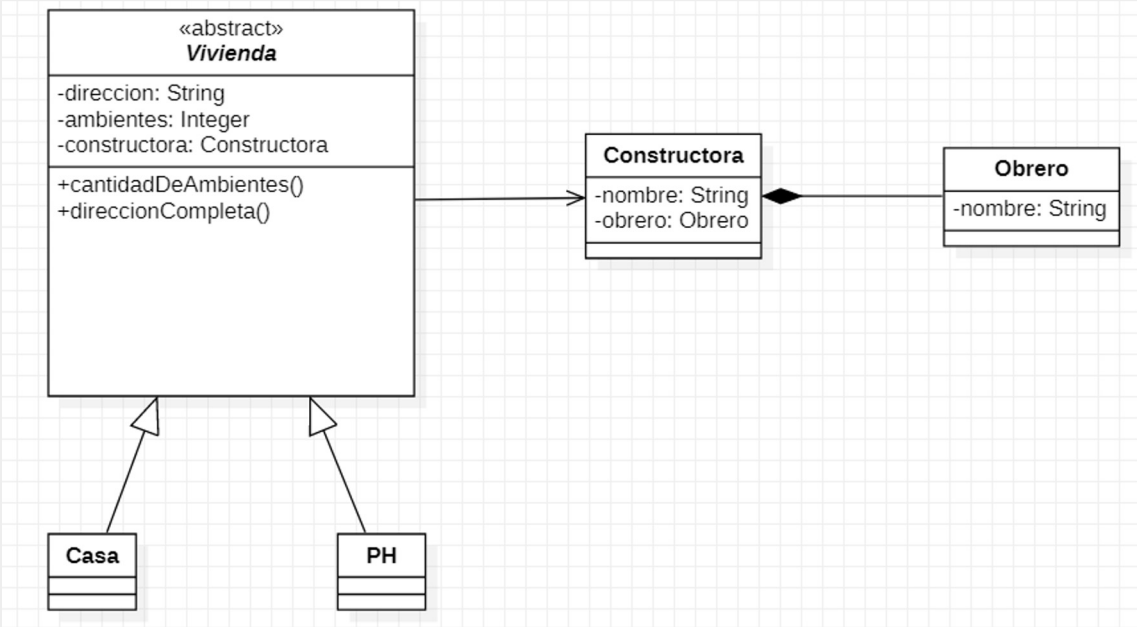


Diagrama de Clases

Relaciones – Dependencia ("Uso")

- Se utiliza para representar que una clase requiere de otra para ofrecer cierta funcionalidad.
- Se suele leer "*A usa o conoce a B*"

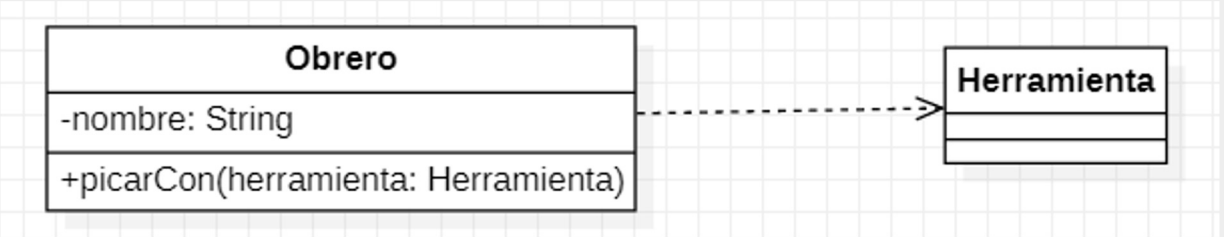


Diagrama de Clases

Relaciones – Realización ("Implementación")

- Se utiliza para representar que una clase Implementa una Interface
- Se suele leer "*A implementa a B, siendo B una interface*"

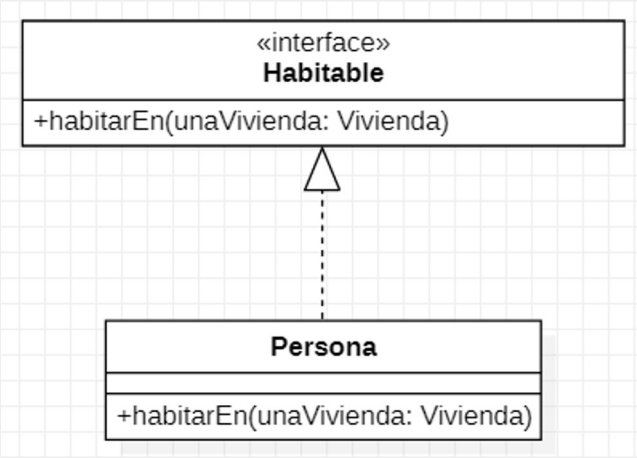
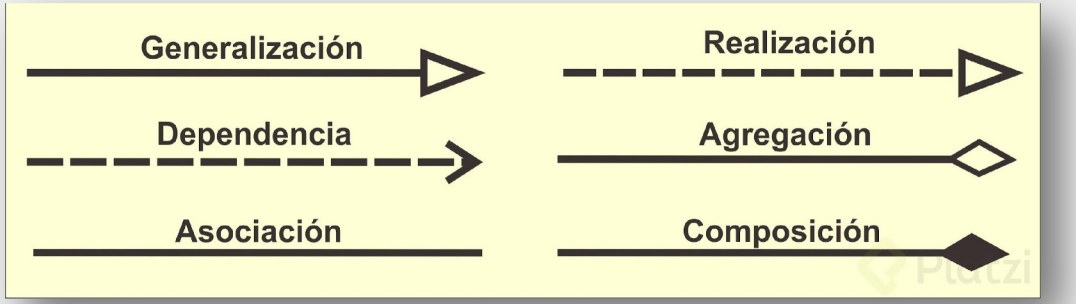


Diagrama de Clases

Resumen de relaciones



Gracias