# Programación orientada a objetos

**Lina Sofia Villarraga Gómez**

**ID:969366**

**Uniminuto**

**Estructura de datos**

**William Alexander Matallana Porras**

**Marzo 19, 2025**

TABLA DE CONTENIDO

## Introducción

Desde la década de los 90 la programación orientada a objetos (POO) se ha consolidado como el paradigma más común en la comunidad de programación. Su evolución surgió como una respuesta a las limitaciones de los paradigmas anteriores, ofreciendo una forma más estructurada de organizar el código. En lugar de centrarse en funciones y lógica, la POO se basa en la creación de objetos que representan elementos del mundo real, cada uno con atributos (características) y métodos (acciones), lo que facilita la reutilización y el mantenimiento del software.

Este documento explica de manera general la programación orientada a objetos en Java, explicando conceptos esenciales como clases, métodos y principios fundamentales. También se explorarán elementos más avanzados, como clases abstractas e interfaces, junto con herramientas gratuitas para la creación de diagramas de clases, los cuales ayudan a visualizar, diseñar y comprender mejor la estructura de un sistema.

## Objetivo

Explicar los conceptos fundamentales de la programación orientada a objetos, incluyendo los diagramas de clases, su estructura y relaciones, así como los métodos, las clases abstractas y las interfaces en Java, para facilitar su comprensión y aplicación.

**¿Qué es programación orientada a objetos?**

La programación orientada a objetos es un modelo de desarrollo de software que organiza el diseño en torno a objetos en lugar de funciones y lógica. Un objeto es una unidad que combina atributos y métodos, permitiendo representar elementos del mundo real dentro del código. Este enfoque facilita la creación de programas grandes y complejos, ya que permite estructurarlos de manera más organizada y flexible. Además, favorece el trabajo colaborativo, ya que el desarrollo puede dividirse en grupos.

Entre sus principales beneficios están la reutilización, la escalabilidad y la eficiencia del código. En este paradigma, los desarrolladores identifican los objetos que desean manipular y establecen las relaciones entre ellos, en un proceso llamado modelado de datos.

**¿Qué son clases en java?**

En Java, una clase es una plantilla o modelo a partir del cual se crean objetos, definiendo las características y comportamientos que estos tendrán. Estas clases también permiten la encapsulación, la herencia y el polimorfismo, lo que facilita la reutilización del código y la creación de sistemas escalables.

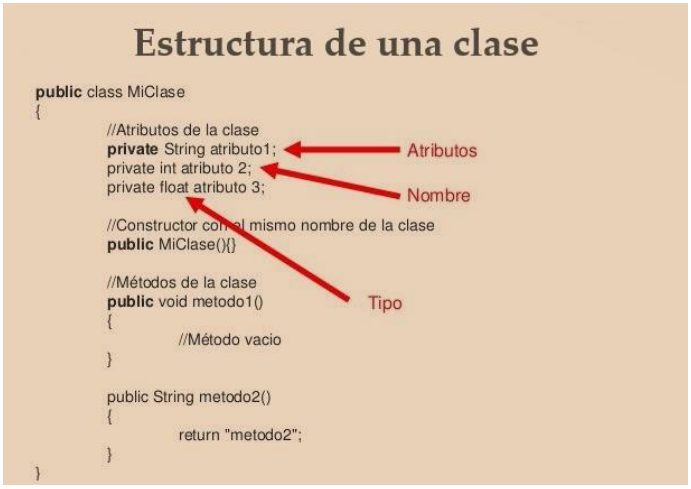
Cada clase en Java está compuesta por:

Variables de instancia: Son las variables que pertenecen a cada objeto creado a partir de la clase.

Métodos: Son las funciones que definen el comportamiento de los objetos de la clase.

Constructores: Son métodos especiales que se utilizan para inicializar los objetos.

Clases internas: Son clases que están definidas dentro de otra clase.




**Diagramas de clases**

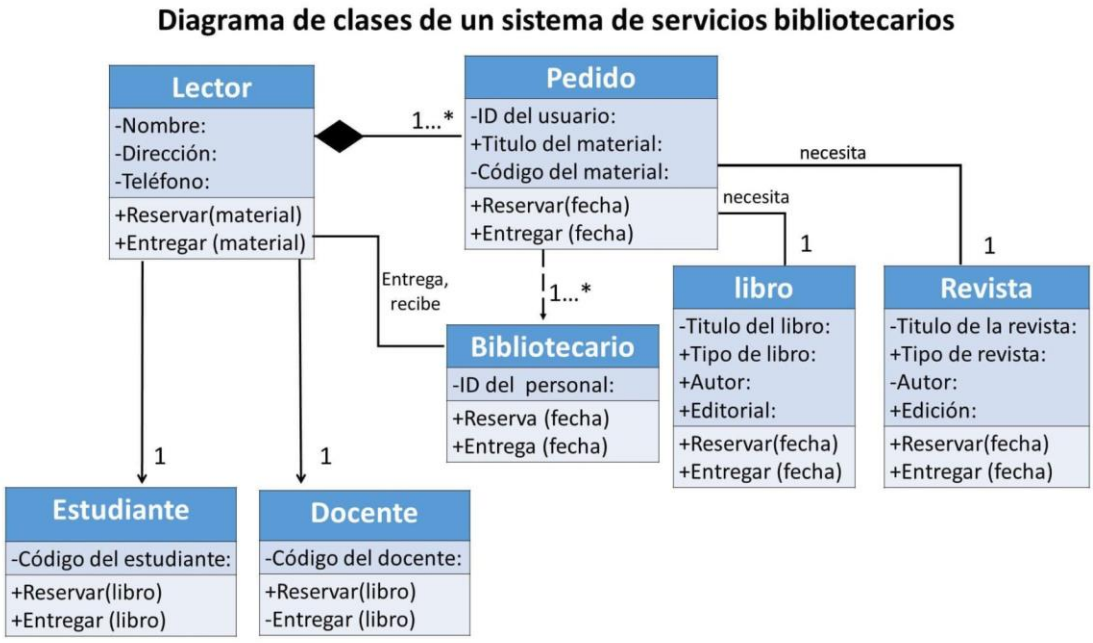
Los diagramas de clases son un diagrama de flujo representado con cuadros rectangulares que ayuda a los desarrolladores a comprender la arquitectura de un sistema ya que este proporciona una descripción general de un sistema en cuanto a los métodos, atributos y también describe su relación entre ellos.

**Estructura:**

Un diagrama de clases tiene de tres secciones:

1. **Sección superior**: Contiene el nombre de la clase.
2. **Sección central**: Describe los atributos de la clase.
3. **Sección inferior**: Muestra las operaciones o métodos asociados a la clase.

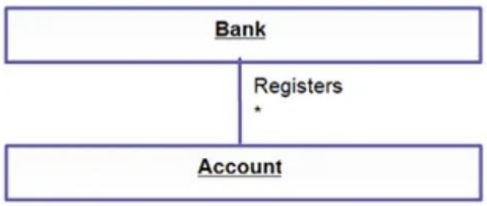
Ejemplo:



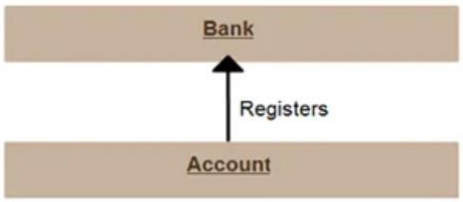

**Relaciones en los diagramas de clases**

1. **Asociación:** Entre otras dos clases en una relación de asociación, una clase de asociación forma parte de ella. Se podría obtener información adicional sobre la relación adjuntando la relación de asociación con la clase de asociación. Varias operaciones, atributos, etc., están presentes en la clase de asociación.

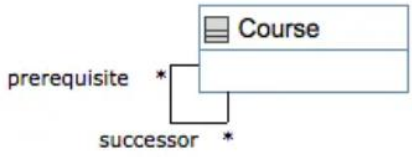
2. **Multiplicidad:** Es una de las relaciones más incomprendidas que describe el número de instancias permitidas para un elemento en particular al proporcionar un intervalo entero no negativo inclusivo. Tiene límite inferior y superior.




3. **Asociación dirigida:** La asociación dirigida es una relación unidireccional en un diagrama de clase que define el flujo de control clasificador a otro. Su navegabilidad se especifica en uno de los extremos y se representa con una flecha.



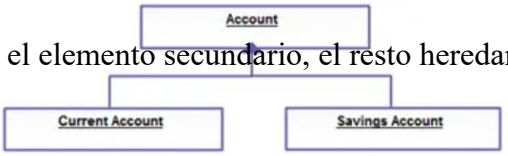

4. **Asociación reflexiva:** Se puede dividir en simétrico en donde la semántica de cada extremo de la asociación no tiene una diferencia lógica, o en asimétrico en la cual la clase asociada es la misma, pero hay una diferencia semántica entre los extremos de la asociación.



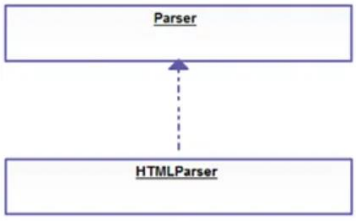
5. **Agregación:** Este tipo de relación, se crea un objeto más complejo mediante el ensamblaje de diferentes objetos juntos. La integridad de los objetos está protegida y el objeto de control decide la respuesta de los objetos ensamblados.


6. **Composición:** La composición es un tipo de agregación que indica una relación de dependencia total entre los objetos, donde la vida útil de las partes depende del todo. Representa un ciclo de vida fuerte y un flujo de datos unidireccional, generalmente indicado con una línea continua.


7. **Generalización:** Es un tipo de relación secundaria que se utiliza para describir varios diagramas de casos de uso y garantiza que la clase secundaria reciba las propiedades presentes en el elemento primario. Por lo tanto, los atributos distintos deben definirse solo en el elemento secundario, el resto heredaría del elemento primario.




8. **Realización:** El comportamiento de un elemento modelo se realiza mediante el comportamiento especificado de otro elemento modelo.



**Herramientas en línea gratuitas que se pueden utilizar para realizar diagramas de clases**

1. **Draw.io:** Aplicación de código abierto para hacer cuadros, diagramas, circuitos y varios tipos de mapas. Además ofrece una serie de plantillas para avanzar y se puede guardar en varios formatos, incluyendo PDF, PNG o JPG.
2. **GitMind:** Una herramienta todo en uno que te permite crear diagramas, mapas conceptuales variados, asignar prioridades de tareas, un medidor de progreso con símbolos y que ofrece una gran cantidad de plantillas para tus esquemas.
3. **LucidChart:** Se trata de una herramienta online que te permite crear varios tipos de diagramas, organigramas, planos, mapas y esquemas varios. Es una herramienta de pago, pero tiene una versión gratuita que incluye las figuras básicas y 100 plantillas esenciales.
4. **Creately:** web enfocada al trabajo empresarial colaborativo, y que te permitirá hacer varios tipos de esquemas y diagramas, y mapas conceptuales. u funcionamiento también busca ser sencillo, con una columna lateral con unos elementos que puedes pasar a la página central, y en los que podrás cambiar el tamaño, tipografía, color y estilo.

**Tipos de métodos en java**

- **Métodos estáticos:** Se definen con static y pueden ser llamados sin crear una instancia de la clase. Se usan para agrupar funcionalidades relacionadas.
- **Métodos de instancia:** Requieren una instancia de la clase para ser invocados y operan sobre los atributos de la clase. Son clave para la encapsulación.
- **Métodos de retorno de valores:** Devuelven un resultado después de realizar una operación mediante la palabra clave return. Son útiles para cálculos y procesamiento de datos.

- **Métodos sin retorno de valores:** No devuelven ningún resultado, simplemente ejecutan una acción sin necesidad de un valor de salida.
- **Métodos con argumentos:** Reciben valores al ser invocados, permitiendo realizar operaciones personalizadas según los datos proporcionados.
- **Métodos sobrecargados:** Comparten el mismo nombre, pero tienen diferentes listas de parámetros, permitiendo múltiples versiones de un método según los argumentos recibidos.

**Principios de la programación orientada a objetos**

- **Encapsulación:** La implementación y el estado de cada objeto se mantienen de forma privada dentro de la clase. Otros objetos no tienen acceso a esta clase o la autoridad para realizar cambios, pero pueden llamar a una lista de métodos públicos.
- **Abstracción:** Los objetos solo revelan mecanismos internos que son relevantes para el uso de otros objetos, ocultando cualquier código de implementación innecesario.
- **Herencia:** Permite reutilizar código al establecer relaciones entre clases y subclases, optimizando el desarrollo y asegurando una estructura jerárquica eficiente.
- **Polimorfismo:** Permite que los objetos adopten diferentes comportamientos según el contexto, reduciendo la duplicación de código.

**¿Qué son clases abstractas?**

Las clases abstractas son similares a las clases convencionales, ya que pueden tener métodos, atributos y constructores. La principal diferencia es que una clase abstracta debe tener al menos un método abstracto, es decir, un método vacío que no posee cuerpo ni puede

realizar ninguna acción. Tiene como propósito definir qué se debe hacer, mas no el cómo se debe hacer.

Es importante mencionar que las clases abstractas pueden ser heredadas por tantas clases como sea necesario, pero no pueden ser instanciadas. Para heredar de una clase abstracta, se utiliza la palabra reservada “ extends ”.

**¿Qué es interfaz y cómo se hace?**

Una interfaz en Java es una colección de métodos abstractos y constantes que establecen qué se debe hacer, pero no cómo, actuando como un contrato que las clases que la implementen deben seguir. Para crear una interfaz, se utiliza la palabra “ interface “, seguida del nombre de la interfaz. Dentro de ella, se declaran métodos abstractos sin implementación, los cuales deben ser definidos por las clases que la implementen utilizando la palabra “ implements “. A diferencia de una clase abstracta, una interfaz no puede contener implementación de métodos y no puede hacer nada por sí sola. La principal ventaja de las interfaces es que una clase puede implementar múltiples interfaces, mientras que solo puede heredar de una clase abstracta, lo que proporciona mayor flexibilidad en la estructura del código.

**Conclusión**

La programación orientada a objetos en Java facilita la creación de software un reutilizable y escalable. Sus principios fundamentales, como encapsulación, abstracción, herencia y polimorfismo, permiten una mejor organización del código. Además, las clases, métodos, diagramas de clases, clases abstractas e interfaces brindan herramientas esenciales para estructurar sistemas eficientes y flexibles. Comprender estos conceptos es clave para desarrollar aplicaciones más eficientes y mantenibles.

**Referencias**

Rodríguez, R., Sosa, E., & Prieto, Á. (2004). Programación orientada a objetos.

Programacion-Orientada-Objetos-2012.pdf

TechTarget**.** Programación orientada a objetos (OOP). *Computer Weekly*. https://www.computerweekly.com/es/definicion/Programacion-orientada-a-objetos-OOP

Conceptos.es. (s.f.). *Clase en Java*.

https://conceptos.es/clase-en-java

Education Wiki. *Class Diagram*.

https://es.education-wiki.com/7917237-class-diagram

Xataka. (s.f.). *Herramientas gratis para hacer esquemas y diagramas online*. https://www.xataka.com/basics/herramientas-gratis-para-hacer-esquemas-diagramas-online

Tiposde.net. (s.f.). *Tipos de método en Java*. Recuperado el 19 de marzo de 2025, de https://tiposde.net/tipos-de-metodo-en-java/

García, E. I.. *Diferencia entre clases abstractas e interfaces en Java*.

Diferencia entre clases abstractas e interfaces en Java - Blog de Código Facilito