Requerimientos de Software

Tarea 8

Profesor:

Jose Angel Campos Aguilar

Estudiantes:

Jimena Méndez Morales - 2023113347

Elder León Pérez - 2023166120

Entrega: 20/10/2024

II Semestre 2024

Contenido

Objetivos

Objetivo general:

Analizar el uso e importancia de los diagramas de clases en el desarrollo de software, destacando su papel en la representación de la estructura del sistema.

Objetivos específicos:

1. Explicar los componentes principales del diagrama de clases.
2. Describir los tipos de relaciones en entre clases en un diagrama.
3. Identificar y enumerar los pasos para desarrollar un diagrama de clases, asegurando que cumpla con los principios de diseño orientado a objetos.

Introducción

Un diagrama de clases es una herramienta desarrollada para la comunicación del diseño estructural de un sistema de forma gráfica y unificada permitiendo reducir la dificultad para entender diversos sistemas que pueden llegar a ser complejos. Esta técnica fue desarrollada por los ingenieros (Grady Booch, Ivar Jacobson y James Rumbaugh) entre los años 1994-1996 posteriormente se introdujeron gráficos de estado que fueron desarrollados por David Harel, debido a la viabilidad de esta técnica de trabajo, fue reconocida como estándar ISO para crear modelos orientados a objetos por Object Management Group (OMG) en 1997.

El diagrama de clases es una herramienta clave en el diseño de software orientado a objetos. Permite visualizar de forma clara las clases que componen un sistema, sus atributos, métodos y cómo se relacionan entre sí. Es como un mapa que muestra la estructura interna del sistema, ayudando a los desarrolladores a entender y comunicar mejor el diseño del software.

Cada elemento del diagrama tiene su función: las clases representan las entidades principales, los atributos describen sus características, y los métodos indican lo que pueden hacer. Además, las relaciones entre clases, como la herencia o la dependencia, muestran cómo interactúan entre sí. En conjunto, un diagrama de clases bien hecho facilita no solo el desarrollo, sino también la evolución del sistema, asegurando que siga principios como el modularidad, el bajo acoplamiento y el encapsulamiento.

Desarrollo

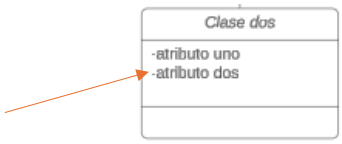
¿Qué es un diagrama de clases?

Se define como el conjunto de elementos gráficos que permiten modelar las clases, atributos y visibilidad de los métodos de un sistema, así como las relaciones que tienen las diversas clases entre ellas mismas.

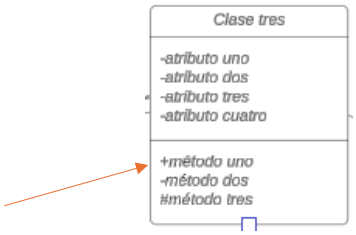
¿Cuáles son los elementos que componen un diagrama de clases?

- **Clases**: Es el nombre que se lo otorga a una entidad se coloca en la parte superior del área grafica se recomienda implementar estándares como camelCase o PascalCase para la escritura, es de carácter obligatorio integrar llenar este espacio en el desarrollo de un diagrama de clases.

- **Atributos**: Se ubican en la sección del medio del área gráfica, estos describen las instancias de objetos que serán definidos por esa clase.



- **Métodos**: Se redactan en la parte inferior del área grafica de la clase, en este apartado se



indican cuales son las funciones o acciones que la instancia del objeto podrá realizar.

- **Visibilidad**: Se ubican a la par de los métodos o atributos con la intención de controlar quien puede acceder a los atributos y funciones de la clase, existen tres niveles de visibilidad que determinan el acceso en diversas áreas del código.
    - **(+) Pública.** Representa que se puede acceder al atributo o función desde cualquier lugar de la aplicación.
    - **(-) Privada.** Representa que se puede acceder al atributo o función únicamente desde la misma clase.
    - **(#) Protegida.** Representa que el atributo o función puede ser accedida únicamente desde la misma clase o desde las clases que hereden de ella (clases derivada

- **Cardinalidad**: Indica cuantas instancias de una clase pueden estar relacionadas con otra ayuda a definir cuántos objetos de una clase pueden asociarse con objetos de otra clase.
    - **Uno a Uno (1..1)**: Cada instancia de una clase está relacionada con una y solo una instancia de otra clase.

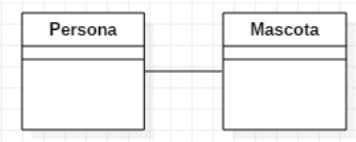
- **Uno a Muchos (1..*)**: Una instancia de una clase está relacionada con muchas instancias de otra clase.
- **Muchos a Uno (*..1)**: Muchas instancias de una clase están relacionadas con una única instancia de otra clase.
- **Muchos a Muchos (..)**: Muchas instancias de una clase están relacionadas con muchas instancias de otra clase.
- **Cero o Uno (0..1)**: Una instancia de una clase puede estar relacionada con ninguna o una instancia de otra clase.
- **Cero o Muchos (0..*)**: Una instancia de una clase puede estar relacionada con cero o muchas instancias de otra clase.

Tipos de relaciones entre clases

Las clases de cada sistema tiene un tipo de relación con otras estas indican el comportamiento que tendrá el sistema entre las clases.

- **Asociación**: Es la relación más básica entre clases indica que una clase se relaciona con otra, se dibuja como una línea simple conectando las dos clases.

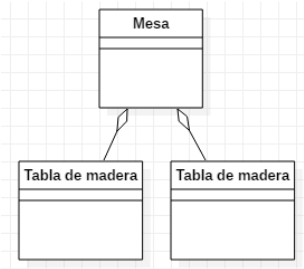
  En este caso una persona se puede asociar (relacionar) con una o muchas mascotas sin necesidad de vincular sus atributos.



- **Agregación**: Es una representación donde un objeto o clase es parte de otro es decir donde se vuelven un conjunto, sin embargo, si el ¨Todo¨ es destruido sus partes siguen existiendo por si solas. Se representan con una línea con un rombo en la parte de la clase que une todas sus partes.

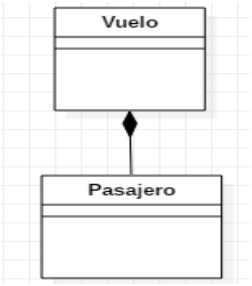
Comentado [JM1]: @ELDER JOSHUA LEON PEREZ explique los ejemplos, no solo es poner lo que es la asociación

Un ejemplo objetivo de agregación donde se juntan clavos y madera es una mesa sin embargo si la mesa se desarma el clavo y la madera siguen existiendo por aparte sin necesidad el uno del otro.



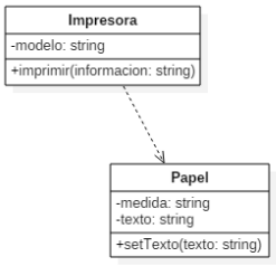
- **Composición**: Es una relación entre un objeto o clase y las partes que lo componen, Tiene similitud con la agregación sin embargo en este caso si la clase que compone todos los elementos no existe tampoco existirán sus partes. Se representa con una flecha con un rombo hacia la clase principal.

  Un vuelo de una compañía aérea está compuesto por pasajeros, sin los mismo el vuelo no tiene razón lógica para existir.



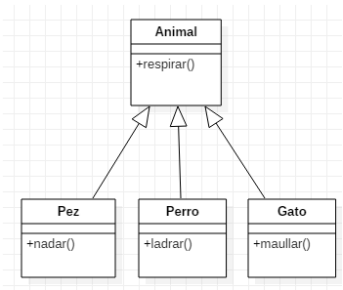
- **Dependencia**: Relación implementada para reflejar que una clase necesita estrictamente de otra para poder ofrecer todas sus funcionalidades. Se representa como una flecha discontinua que apunta de la clase hacia la que necesita para funcionar.

Un ejemplo de dependencia es la relación entre una impresora y el papel, sin papel la impresora no puede ofrecer la funcionalidad de imprimir.

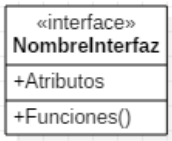


- **Herencia**: Esta relación permite recibir los atributos y métodos de la clase padre (clase de la que se hereda). Se representa como una fecha de la clase hija apuntando a la clase padre con la punta triangular vacía.

  Un animal sin importar su especie necesita respirar, sin embargo, no todos ladran así se ve la herencia todo animal hereda el método de respirar y lo incorpora a si mismo juntos a sus métodos individuales.

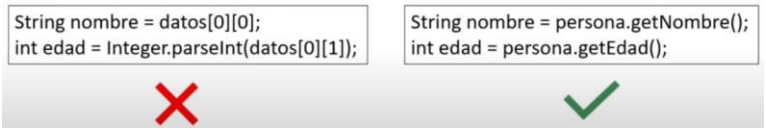


- **Interfaces**: Es una entidad donde se declara una seria de atributos y funciones además de obligaciones, donde toda clase asociada a la misma debe cumplir con lo que se estipula en la interfaz, no es posible instanciar esta clase.

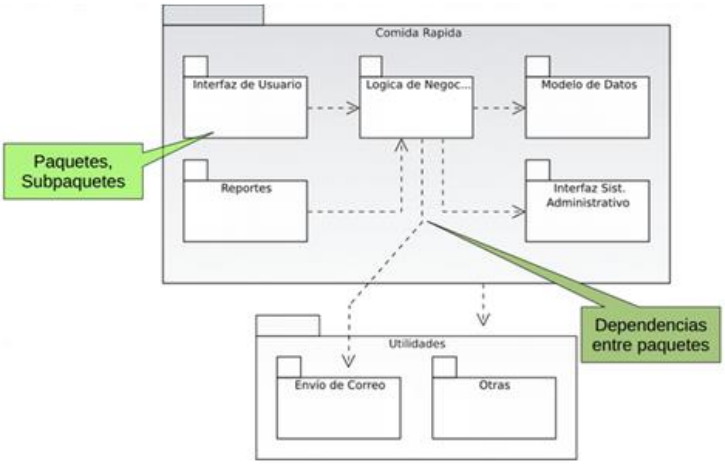


Principios de diseño de los diagramas orientados a objetos

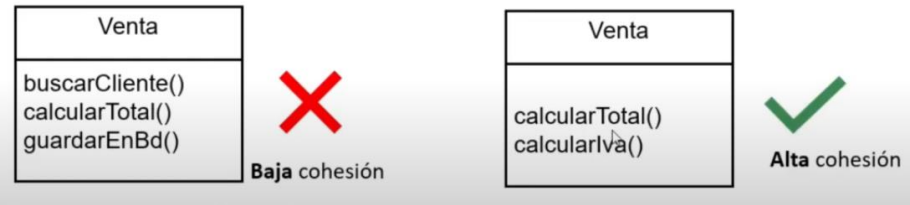
- Abstracción: Representar la información relevante de un elemento, para disminuir la complejidad. Un ejemplo es trabajar con una lista que tiene aspectos detallados de código, pero eso se puede abstraer en una clase llamada persona que se puede manejar de forma más sencilla a la hora de obtener los datos:



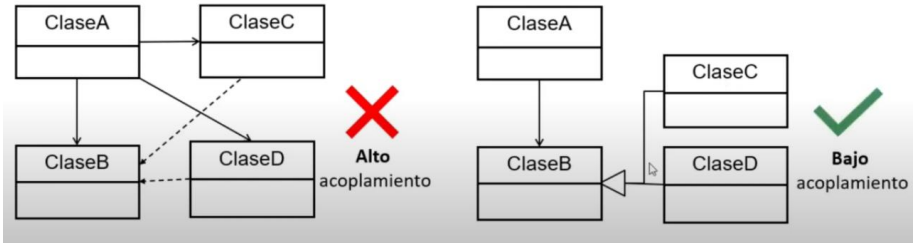
- Modularidad: Un programa como un conjunto de subprogramas (módulos), cada módulo tiene una función particular, suele ser un poco independiente e incluso puede encontrarse en ficheros de código distintos. Un ejemplo para realizarlo son los diagramas de paquetes que modularizan un programa separando cada paquete con subclases o subpaquetes y este paquete representa un solo módulo, en el ejemplo podemos ver que muestra un sistema de envío de comida rápida, donde tenemos el paquete de comida rápida que se puede representar como un solo módulo y el otro de utilidades que pueden representar el proceso de envío de la comida.



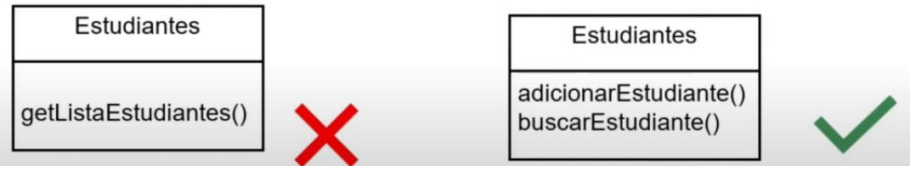
- Alta cohesión: Representa los métodos que pueden ser reutilizados y que no deben de estar dentro de varias clases porque no son parte de ellas, y pueden ser parte de una sola. Un ejemplo de esto son un sistema de ventas que tenga una baja cohesión porque en la clase ventas incluye un método que dice guardarEnBD y esta puede estar desarrollada por aparte en una clase que se llame BaseDeDatos y en esta incluya el guardado, es una forma en la que el guardado se puede reutilizar en varias clases, no solo en Venta y así se evita la repetición de código innecesario.



- Bajo acoplamiento: Baja dependencia entre clases, se deben agregar lo menos posible las relaciones entre clases, solo las necesarias, para evitar muchas dependencias. Ejemplo:



- Encapsulamiento: Cuando se diseña una clase se encapsulan sus métodos y atributos que solo pueden ser accedidos por la misma clase, pero no por las demás, esto se hace para que las clases externas no tengan acceso a cierta información. Ejemplo la siguiente clase de estudiantes, no se debe acceder a la lista de los estudiantes en otra clase, solo se accede a la agregación y búsqueda del estudiante.



Pasos por seguir para hacer un diagrama de clases

1. **Identificación de clases clave:**

   Identifica todas las clases que formarán parte del sistema, asegurándote de que reflejan los componentes principales de la solución. Considera las entidades que se modelarán en el dominio del problema.

2. **Definición de atributos:**

   Para cada clase, identifica los atributos que describirán sus características y propiedades. Asegúrate de definir la visibilidad de cada atributo (público, privado, protegido) según su accesibilidad dentro del sistema.

3. **Definición de métodos:**

   Determina las funcionalidades o comportamientos de cada clase, especificando sus métodos. Define también la visibilidad de estos métodos, tomando en cuenta el nivel de acceso necesario para interactuar con ellos.

4. **Establecimiento de relaciones entre clases:**

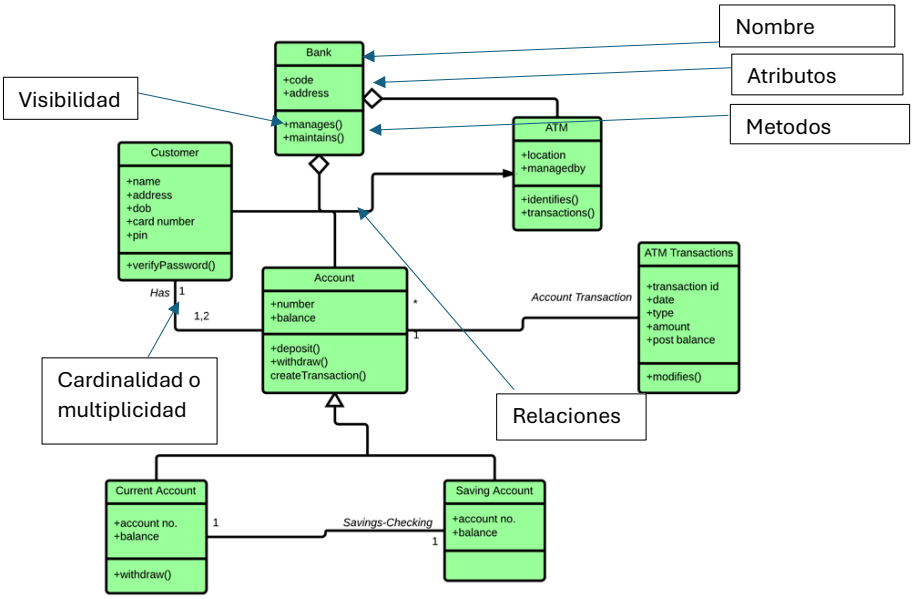
   Define cómo interactúan las clases entre sí, identificando los tipos de relaciones (asociación, herencia, agregación, composición, dependencia). Asegúrate de que las relaciones reflejan correctamente las interacciones y dependencias del sistema.

5. **Determinación de la multiplicidad:**

   Especifica la multiplicidad en las relaciones entre clases (uno a uno, uno a muchos, muchos a muchos), para reflejar con precisión cuántas instancias de una clase pueden estar asociadas con instancias de otra clase.

6. **Revisión y validación:**

   Revisa el diagrama completo para asegurarte de que sea claro, legible y cumpla con los principios de diseño orientado a objetos. Verifica que todas las clases, atributos, métodos y relaciones estén correctamente definidos y sigan una estructura lógica.


Nombre
Atributos
Metodos
Visibilidad
Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedby
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Account
+number
+balance
+deposit()
+withdraw()
createTransaction()
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Has
1
1,2
*
1
Account Transaction
Cardinalidad o multiplicidad
Relaciones
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance

Conclusión

En resumen, los diagramas de clases son herramientas esenciales en el desarrollo de software orientado a objetos, ya que permiten visualizar y comunicar de manera efectiva la estructura del sistema. Mediante la representación de clases, atributos y métodos, junto con sus relaciones, estos diagramas facilitan la comprensión tanto para desarrolladores como para otros interesados, asegurando que todos los aspectos del diseño sean claramente definidos y entendidos desde el principio.

Además, los diagramas de clases son cruciales para aplicar correctamente principios de diseño como la abstracción, la modularidad y el encapsulamiento. Esto permite que el sistema sea más organizado, con un enfoque en minimizar dependencias innecesarias y fomentar la reutilización de código. También ayudan a evitar redundancias y a detectar áreas donde el diseño puede ser mejorado, promoviendo un código más limpio y eficiente.

Por último, su uso no solo facilita la implementación inicial del software, sino que también mejora su mantenimiento y escalabilidad a largo plazo. Los diagramas de clases ofrecen una vista clara de cómo interactúan los componentes del sistema, lo que simplifica futuras modificaciones y garantiza que el software evolucione de manera ordenada y sostenible.

Bibliografía

Miro. (s.f.). ¿Qué es un diagrama de clases UML? *Miro*. https://miro.com/es/diagrama/que-es-diagrama-clases-uml/

Diagramas UML. (s.f.). *Diagrama de clases*. https://diagramasuml.com/diagrama-de-clases/

OK Diario. (2018, Noviembre 09 .f.). ¿Qué es un diagrama de clases? *OK Diario*. https://okdiario.com/curiosidades/que-diagrama-clases-3323710

Diagramas UML. (s.f.). Diagrama de clases. *Diagramas UML*. https://diagramasuml.com/diagrama-de-clases/

Parker, A.(2024,Septiembre 26). Historia Diagramas UML: Historia, tipos, características, versiones, herramientas. *Guru99*. https://www.guru99.com/es/uml-diagrams.html

Normas ISO. (s.f.). Normas ISO. *Normas ISO*. https://www.normas-iso.com/

Parker, A.(2024,Septiembre 26). Diagrama de clases UML: Elementos esenciales de un diagrama de clases UML. *Guru99*. https://www.guru99.com/es/uml-class-diagram.html#essential-elements-of-a-uml-class-diagram

Guru99. (2018). *UML Class Diagram*. https://www.guru99.com/images/1/051818_1150_UMLClassDia10.png

Lucid Software Español. (4 de febrero de 2019). *Tutorial - Diagrama de clases UML* [Video]. YouTube. https://www.youtube.com/watch?v=Z0yLerU0g-Q

Ingeniería de Software. (25 septiembre 2023). *Principios básicos del diseño* [Video]. YouTube. https://www.youtube.com/watch?v=BRDfPswf5kc