TRABAJO UML

Integrantes: Juan, Mikhael, Daniel y Raul

índice

Introducción a UML

¿Qué es UML?

UML, por sus siglas en inglés Unified Modeling Language (Lenguaje de Modelado Unificado), es un lenguaje de modelado visual que se utiliza para especificar, construir, visualizar y documentar los componentes de un sistema de software. No es un lenguaje de programación, sino una herramienta de comunicación estandarizada que utiliza diagramas y símbolos gráficos para representar conceptos complejos de forma comprensible.

UML permite modelar tanto la estructura como el comportamiento de sistemas, y aunque fue creado originalmente para aplicaciones de software, también puede aplicarse a otros dominios, como flujos de trabajo empresariales o procesos industriales.

Fue desarrollado en los años 90 como una unificación de los principales métodos de modelado orientado a objetos existentes en ese momento, y su objetivo principal es facilitar la comprensión del sistema por parte de todos los actores involucrados: analistas, diseñadores, desarrolladores, clientes, entre otros.

Historia y evolución

Orígenes:

Durante la década de 1990, existían múltiples metodologías para el análisis y diseño orientado a objetos, lo que generaba confusión e inconsistencia entre los equipos de desarrollo. Entre las más populares estaban:

OMT (Object Modeling Technique) de James Rumbaugh

Booch Method de Grady Booch

OOSE (Object-Oriented Software Engineering) de Ivar Jacobson

Conscientes de la necesidad de unificar estas metodologías, estos tres expertos se unieron para crear un lenguaje común que reuniera lo mejor de cada enfoque. Así nació UML.

- **1994-1996:** Se fusionan los métodos de Booch, OMT (Object Modeling Technique) y OOSE (Object-Oriented Software Engineering).
- **1997:** Se publica UML 1.0 como estándar para el modelado de software.

- **1999-2005:** Aparecen mejoras y nuevas versiones, destacando UML 2.0, que introduce mayor capacidad de modelado para sistemas complejos.
- **Actualidad:** UML sigue siendo un estándar ampliamente utilizado en la ingeniería de software, especialmente en metodologías orientadas a objetos y desarrollo ágil.

¿Para qué se utiliza?

- Modelar sistemas de software desde el análisis de requisitos hasta el diseño detallado.
- Visualizar la estructura y comportamiento de un sistema.
- Comunicar ideas técnicas entre desarrolladores, analistas y clientes de manera clara.
- Documentar procesos y arquitecturas de sistemas.
- Diseñar sistemas complejos, incluyendo sistemas en tiempo real, distribuidos y empresariales.

Beneficios del uso de UML

- Lenguaje estándar: Comprensible por diferentes profesionales del desarrollo de software.
- Mejora la comunicación entre los equipos de trabajo
- Facilita la documentación técnica del sistema.
- Permite detectar errores conceptuales antes de la codificación.
- Favorece la reutilización de modelos y componentes.
- Independencia del lenguaje de programación, lo que lo hace versátil en distintos entornos de desarrollo.

Conceptos Fundamentales

Modelado

Es el proceso de crear representaciones visuales de un sistema de software. Es como dibujar planos para un software. Se usan diagramas los cuales se utilizan para:

- Ver: hacer el sistema más fácil de entender visualmente
- Decir: Especifica cómo debe ser el sistema sin confusiones
- Hacer: Dar una guia para construir el software
- Guardar: Documentar el sistema para el futuro

Elementos de Estructurales de UML

Atributos

Representan los datos o propiedades de los objetos de la clase. Por ejemplo puede haber una clase llamada “Persona” que tenga los atributos “nombre”, “edad”, “profesión”.

Estos pueden tener varios atributos que describen una clase:

- Visibilidad: Quién puede acceder al atributo (+público, -privado, #protegido, ~paquete)
- Tipo: La clase de datos que almacena (ej:String, integer)
- Estaticidad: Si pertenece a la clase en sí o a cada objeto
- Derivación: Si su valor se calcula de otros atributos (/ esto se debe de poner al inicio)

Actor

Un actor es una entidad externa que interactúa con el sistema practicando en un caso de uso. Los actores pueden ser personas de la vida real, sistemas de computadoras o eventos externos.

Tipos de Actores

Actores principales: Los actores principales son esenciales para el funcionamiento del sistema. Estos suelen es los usuarios o el sistema que inicia los casos de uso.

Actores Secundarios: Estos actores no son necesarios para el funcionamiento básico del sistema, pero contribuyen a la eficiencia o fluidez de este. Estos pueden ser otros sistemas, componentes o usuarios.

Clase

Una clase es una plantilla para crear objetos, definiendo atributos y comportamientos. Es un elemento modelado que representa un conjunto de objetos que comparten las mismas características y acciones

Hay varios tipos de clases:

- **Clase Concreta:** Son las clases normales que puedes usar para crear objetos
- **Clases Abstracta:** Son las clases que no puede instanciar directamente, sirven como base para otras clases. Se identifican en cursiva o con <<abstract>>
- **Interfaz:** Define un contrato de comportamiento que otras clases deben implementar. Se representa con circulo o con <<interface>>

Objeto

Un objeto es una instancia de una clase. Los objetos se pueden agregar a un diagrama de clases para representar instancias. Hay varios tipos de objetos pero les voy a mostrar los 3 más importantes:

- **Objeto de Entidad**
  - Representa datos importantes en el proyecto
- **Objeto de control**
  - Coordina la lógica, decisiones o flujo de un caso de uso
  - Se encarga de organizar la interacción entre objetos frontera y de entidad
- **Objeto de frontera**
  - Representa la interfaz entre el sistema y el actor externo
  - Este tipo de objeto puede ser una pantalla, formulario, API

Métodos

Los métodos son las acciones o comportamientos que pueden hacer una clase u objetos existen varios tipos de métodos:

1. Según su propósito o funcionalidad

- **Constructor**: Método especial que crea una instancia (no se suele detallar en UML, pero puede agregarse si es importante)
- **Destructor**: Libera recursos (poco usado en UML, pero puede mencionarse)
- **Getter/Setter**: Acceden o modifican atributos privados
- **Abstracto**: Declarado sin implementación (solo en clases abstractas; se escribe en *cursiva* en UML)
- **Interfaz (sin implementación)**: Declarado en una interfaz; la clase que la implementa lo define.
- **Sobrecargado**: Varios métodos con el mismo nombre pero diferente número o tipo de parámetros (no se distingue directamente en UML, pero se pueden listar)

Interfaces

Una interfaz es un elemento que define operaciones (métodos) que una clase debe implementar. Esto especifica que debe hacer la clase, pero no como lo hace.

Características:

- **Definición de contratos:** la interfaz actúa como un contrato que las clases deben cumplir
- **Independencia de implementación:** las interfaces no definen el comportamiento real de los métodos, solo su firma. Esto permite que las clases implementan la misma interfaz de distintas maneras
- **Polimorfismo:** Una interfaz permite que diferentes clases puedan ser tratadas como si fueran instancias de una clase común

Nodo

Componente

Un componente es una unidad de software autónoma que puede ser desarrollada, reemplazada, desplegada y reutilizada de manera independiente

Elementos de Agrupación

Paquetes

Un paquete es un elemento estructural que se utiliza para agrupaciones lógicas de otros elementos, como clases, casos de uso, componentes, entre otros. Su principal propósito es organizar el sistema.

- **Organización:** Los paquetes agrupan elementos relacionados entre sí, ayudando a organizar el sistema
- **Jerarquía**: los paquetes pueden contener otros paquetes, esto facilita la organización de componentes
- **Modularidad**: Permite dividir el sistema para que este sea más claro o manejable
- **Visibilidad**: Los paquetes definen las relaciones entre los elementos

Elementos de comportamiento

Caso de uso

Es una representación que describe la interacciones de los actores con el sistema. Esta se representa mediante diagramas los cuales permiten describir las funcionalidades de un sistema desde la perspectiva de los actores.

Un ejemplo sería en los actores:

- **Actor Cliente**: Es una persona que usa la aplicaciones
- **Actor Administrador**: Persona que gestiona el sistema
- **Actor Sistema externo**: Un sistema con el que el sistema princ

Interacción

Define las acciones que los objetos de la clase pueden realizar. Por ejemplo, una clase “Persona” puede tener los métodos “hablar”, “comer”, “trabajar”.

Estado

Son los valores actuales de los atributos de una clase en un momento específico. Es decir que los atributos deben ser introducidos por alguien. Un ejemplo de esto sería de la clase “Coche” sería “Marca”, “Modelo”, “Velocidad”.

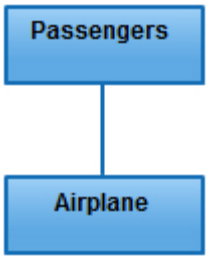
Actividad

La actividad es la representación gráfica del flujo de trabajo o del comportamiento del sistema, mostrando cómo se desarrollan las actividades de un proceso o tarea y cómo actúan las distintas acciones

Relaciones

Asociación

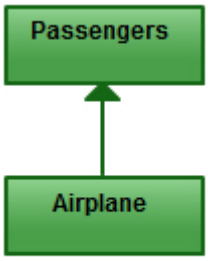
Es un término que abarca casi cualquier conexión o relación lógica entre clases. Por ejemplo, el alumno puede estar vinculado a un aula. Esto se representa solo con una línea



Asociación

Asociación Dirigida

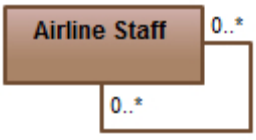
Se refiera a una relación direccional, esta se representa por una línea con punta de flecha rellena, esta representa el flujo direccional



Asociación dirigida

Asociación Reflexiva

Es una tipo de relación en la que la entidad se relaciona consigo misma. Es decir que un objeto o clase está relacionado con el mismo



Asociación Reflexiva

Multiplicidad

Esta relación describe cuántas instancias de una clase pueden estar asociadas con una instancia de otra clase en una relación.

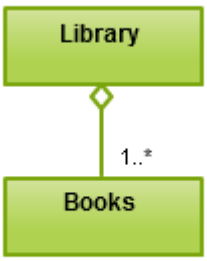
En esta hay varios valores:

- **0:1**: la clase es opcional que permite que haya uno o dero empleados
- **1:1**: la clase es obligatoria y esta debe estar relacionada a un sola instancia de otra clase
- **0:*** : la clase debe estar asociada con 0 o más instancias de otra clase. Esto indica que en la relación no es necesario que haya objetos asociados, pero si hay una relación cabe la posibilidad de que sean múltiples
- **1:*:** la clase debe estar asociada mínimo a una instancia de otra clase pero puede tener una o varias instancias asociadas
- **N:M**: La clase debe estar asociada con un número entre N y M instancias de otra clase

Multiplicidad

Agregación

Es un tipo de relación entre clases que representa una asociación “parte” o “contiene”, donde un objeto (clase contenedora) contiene otros objetos (la parte o componentes), pero los objetos parte pueden existir de forma independiente al objeto contenedor. Esta se representa mediante una línea y un rombo blanco en la parte de la clase contenedora.



La agregación

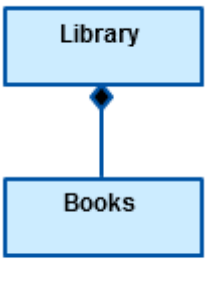
En esta relación la clase contenedora es librería y la clase parte es libro

Características

- Esta debe tener siempre una clase asociada eso significa que la relacion sera 1:1 o 1:*
- Si el objeto contenedor se destruye, las instancias de los objetos parte pueden seguir existiendo de manera independiente

Composición

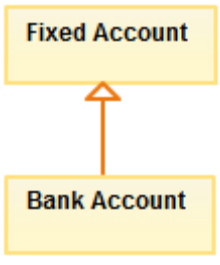
Es una relación en la cual una clase contiene completamente a la otra como parte de su estructura interna. Esta se usa cuando una no puede existir sin la otra. Esta se representa mediante una línea y un rombo relleno. Y su utilidad puede ser la de asociar las clases más importantes como “Coche” y “Motor”.



Composición

Herencia/Generalización

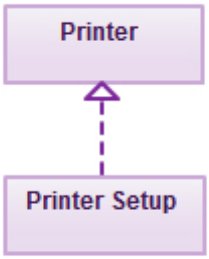
Es una relación en la cual una clase asociada es hija de otra con el objetivo de asumir las mismas funcionalidades de la clase padre, es decir tener los mismos atributos y comportamientos. Esta se representa mediante una punta de flecha rellena de blanco



La herencia

Realización

Es una relación la cual se da cuando una clase implementa una interfaz. Es decir que la clase debe cumplir un conjunto de métodos en una interfaz, pero no hereda directamente su implementación, como en la relación de herencia.



Realización

Visibilidad y alcance

La visibilidad controla quién puede acceder a un método o atributo

- **+ Público**: Accesible desde cualquier parte
- **- Privado**: Solo accesible dentro de la clase
- **# Protegido**: Accesible desde la clase y sus subclases
- **~ De paquete**: Accesible dentro del mismo paquete

El alcance indica si un atributo o método pertenece a la clase o a las instancias

- **De instancia**: Pertenece a un objeto específico
- **Estático / de clase**: Pertenece a la clase y se representa en UML con el **nombre subrayado**

Abstracción

La abstracción consiste en mostrar solo los aspectos esenciales de un objeto o sistema ocultando los detalles internos que no son relevantes para quien lo usa

Esto se usa mediante interfaces o clases abstractas y permite definir qué hace un elemento sin explicar cómo lo hace

Encapsulamiento

El encapsulamiento es el acto de ocultar los detalles internos de una clase (como atributos o implementaciones de métodos) y restringir el acceso a ellos.

Esto se utiliza para proteger los datos de una clase, de modo que no puedan acceder o modificarlos directamente desde fuera.

Y solo permite el acceso mediante métodos públicos como getters y setter.

Ej: getters: getNombre (): String | setters : setNombre(n: String): void

Diagramas de casos de uso

Un diagrama de casos de uso es una forma de diagrama de comportamiento (un diagrama de comportamiento es un tipo de diagrama de modelado de sistemas para representar como se comporta un sistema o algunas de sus partes a ciertos eventos o estímulos) mejorado. Este tipo de diagrama forma parte del modelado orientado a objetos y es muy común en el análisis y diseño de software.

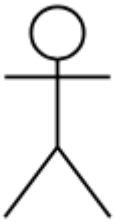
El uso de este tipo de diagramas nos mostrara quien y como interactúa con el sistema, define la funcionabilidad del sistema y nos proporciona una base para posteriores diseños entre otras ventajas-

ELEMENTOS DE LOS DIAGRAMAS DE CASOS DE USO

Los elementos básicos de un diagrama de casos de uso son los siguientes:

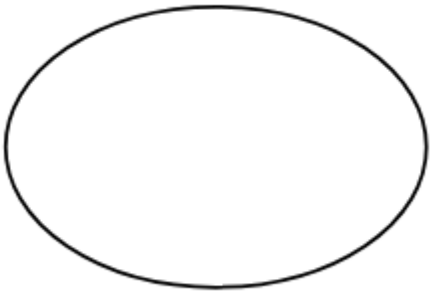
Actores

Representan a las personas, otros sistemas, o entidades externas que interactúan con el sistema. Se dibujan como muñecos (stick figures).



Casos de uso

Son acciones o funciones que el sistema pueden realizar. Se representan como óvalos.



Relaciones

Conectan actores con casos de usos o casos de usos entre si.

RELACIONES EN LOS DIAGRAMAS DE CASO DE USO

RELACIONES ENTRE ACTOR Y CASO DE USO

La asociación entre un actor y un caso de uso se denomina relación de comunicación. Un actor puede ser:

- **Activo:** Cuando el actor inicia es uso de un caso de uso. Si eso usa un flecha como relación esta debe apuntar al caso de uso, si no hay flecha el actor es activo de por si.
- **Pasivo:** Cuando el caso de uso es iniciado por el software y no por el actor. La flecha debe apuntar al actor.

RELACIONES ENTRE CASO DE USO Y CASO DE USO

Existen tres tipos de relaciones:

- **Asociación:** Esta relación se representa mediante un línea y se usa para representar la comunicación entre un actor y un caso de uso, entre casos de uso o entre actores.

- **Generalización o especialización**: Se usa cuando existe un comportamiento común a varios casos de uso, pero que presentan una diferencia. De esta forma los casos de uso específicos “heredan” el comportamiento del caso de uso generalizado. La relación se representa con una línea con flecha. Esta relación solo se puede realizar entre dos casos de uso o entre dos actores, nunca entre un actor y un caso de uso.

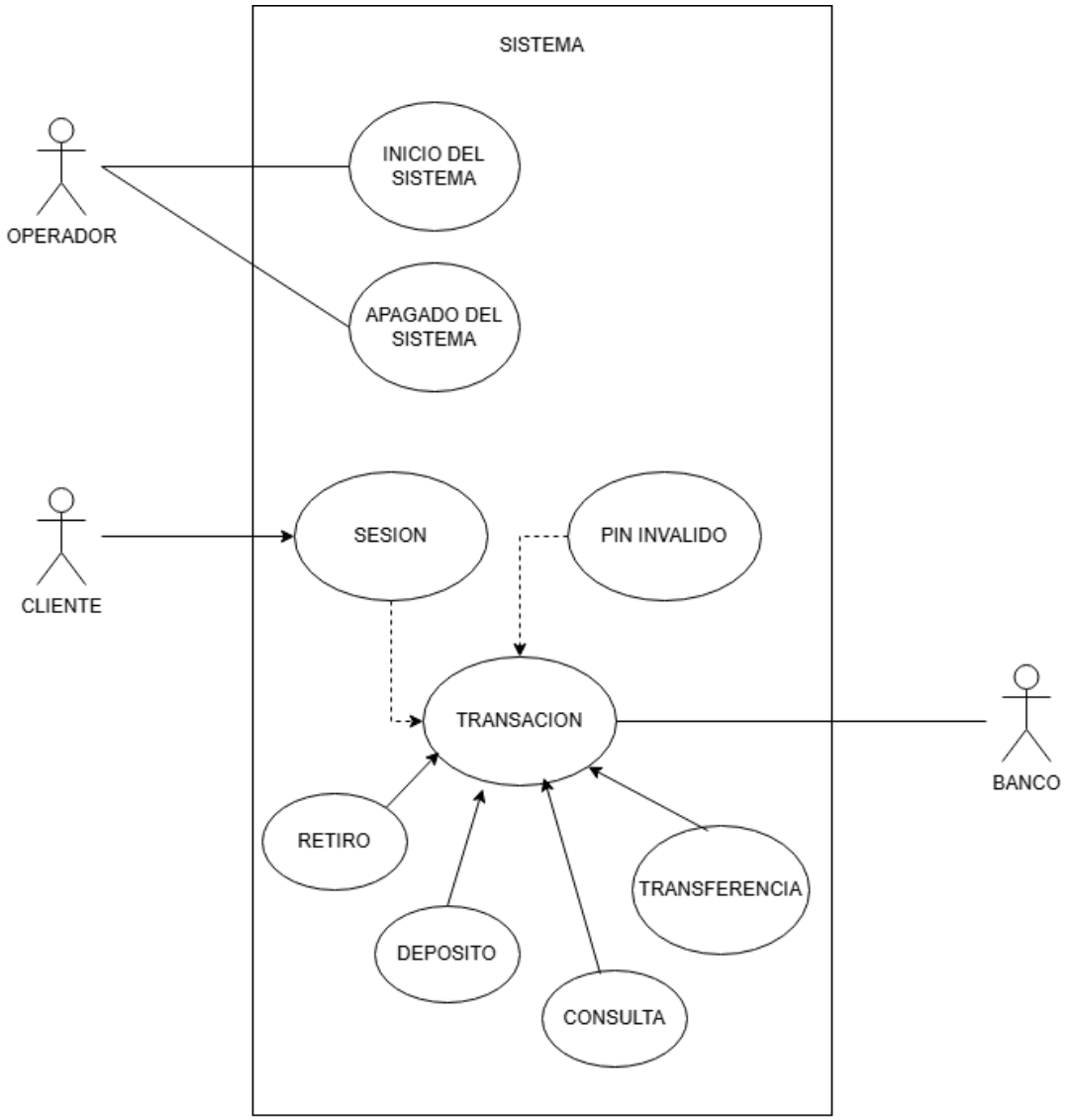
    - **Por ejemplo,** una tienda que maneja varios sistemas de realizar pedidos: realizar pedido en tienda, online o por teléfono. Todos tiene pasos diferentes para el poder realizar el pedido pero se puede definir un caso de uso generalizado que sea “Realizar pedido”.

- **Include:** Utilizada cuando existen una serie de pasos comunes y de observación obligatoria entre varios casos de uso. Se representa mediante una flecha punteada que parte del caso de uso base hacia el caso de uso incluido. Esta relación solo se puede establecer entre dos casos de uso.

    - **Por ejemplo**, un sistema de ventas con tres casos de uso: Ingresar pedido, facturar y registrar pago. Para estos tres casos de uso necesitamos primero buscar a que cliente asociaremos el pedido, creando así un caso de uso llamado “Seleccionar cliente” con una relación de inclusión con los casos de uso anteriores.

- **Extend**: Se utiliza para describir comportamientos adicionales que añaden actividades a uno o varios casos de uso. Un caso de uso puede funcionar como añadido a otro o varios casos de uso. Esta relación se representa igual que una relación de include solo que esta vez la flecha parte del caso de uso extendido al caso de uso base. Esta relación solo se puede establecer entre casos de uso.
    - **Por ejemplo**, un sistema de banca donde tenemos un procedimiento de autentificación. Si el usuario se encuentra en otro país el banco pedirá otro sistema para autentificarse. Este otro sistema puede ser un caso de uso extendido con el nombre de "Sistema de autentificación alternativo" ya que no siempre será utilizado. Por último este caso de uso extendido podrá también ser utilizado por otros casos de uso como por ejemplo uno que permita hacer una transacción.

EJEMPLO



En este ejemplo podemos observar el funcionamiento de un cajero automático. Vemos tres actores los cuales interactúan con los diferentes casos de uso del sistema, que son las diferentes actividades que puede ejecutar el cajero automático.

Diagramas de clases

Un diagrama de clases es un tipo de diagrama estructural (un diagrama estructural es un tipo de diagrama que representa la estructura estática de un sistema, es decir, una estructura que no va a cambiar con el tiempo) el cual representa una clase de un sistema y sus relaciones.

En el diseño de software orientado a objetos, las clases son las encargadas de crear y manipular objetos. Por lo tanto, las clases son elementos esenciales dentro de este desarrollo.

Estos diagramas de clases no existen por si solos ya que dependen de los diagramas de casos de uso y están vinculados con los diagramas de objetos y de comunicación. Gracias a la combinación de estos diagramas podemos tener una visión mas clara del software.

Estos diagramas tienen dos propósitos principales:

- Visualizar las clases de un sistema y sus propiedades.
- Mostrar y analizar las relaciones entre clases.

Además este tipo de diagramas son la base para los diagramas de componentes y despliegue que muestran los aspectos de hardware y software de un sistema.

Estos diagramas nos muestran el funcionamiento, el comportamiento y las relaciones de un sistema. Nos ayudara con la creación de clases, sus atributos, sus métodos y sus relaciones. También podremos ver como afectará a las clases existentes el añadir una nueva clase.

ELEMENTOS DE LOS DIAGRAMAS DE CLASE

Estos diagramas contienen clases y sus interacciones. Estas se representan mediante un rectángulo que contiene el nombre de la clase, sus atributos y sus métodos. Según el nivel de detalle que necesitaremos determinaremos si mostrar los atributos y los métodos.

Las interacciones de estos diagramas muestran las relaciones entre las clases utilizando líneas y flechas. Las flechas tienen significados específicos para diferenciar las relaciones, como por ejemplo las de herencia, las bidireccionales o las unidireccionales. Estas clases también pueden agruparse en paquetes.

NOTACIONES DEL DIAGRAMA DE CLASES

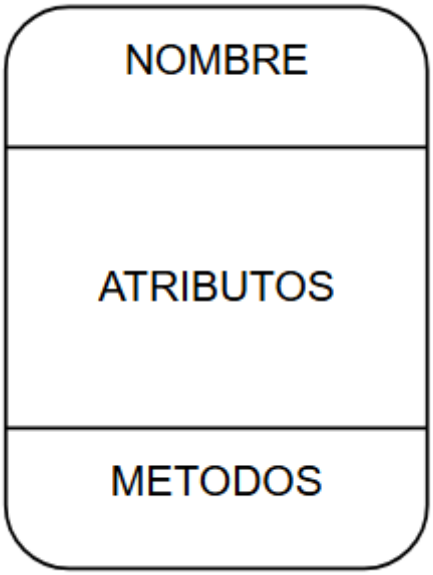
Estas notaciones incluyen las etiquetas de lo miembros de la clase, su visibilidad y los paquetes.

- **Clases:** El nombre de una clase refleja el nombre de un objeto. Este nombre va en el centro en la sección superior arriba, en negrita y en mayúsculas. El nombre de una clase es obligatorio.
- **Atributos:** La sección del centro incluye los atributos de una clase. Estos describen las características de una clase, las cuales se aplicaran a los objetos.
    - **Por ejemplo**, en una clase que sea “**TARJETA DE DEBITO**”, los atributos serian el número de tarjeta y el propietario. Para cada objeto creado de la clase, los atributos tendrán diferentes valores.

- **Métodos**: La última parte del rectángulo es para los métodos de la clase. Los métodos son las acciones que la clase realiza.
    - **Por ejemplo**, en la clase "**TARJETA DE DEBITO**", podemos tener un método que sea el poder cambiar un número de tarjeta y su propietario.

- **Visibilidad**: Los signos de mas y menos representan si los atributos de una clase o métodos son visibles o no. La almohadilla se puede usar para proporcionar una visibilidad protegida. La visibilidad nos permite decidir si una clase y sus métodos y atributos se pueden utilizar fuera de la propia clase.

- **Paquete**: Los paquetes sirven para agrupar clases y se visualizan mediante una caja con una etiqueta con el nombre en mayúsculas del paquete. El uso de paquetes nos ayudar a determinar una mejor visibilidad.

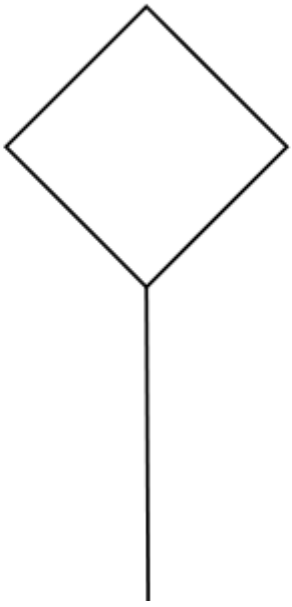


NOTACIONES DE LAS RELACIONES DE LOS DIAGRAMAS DE CLASES

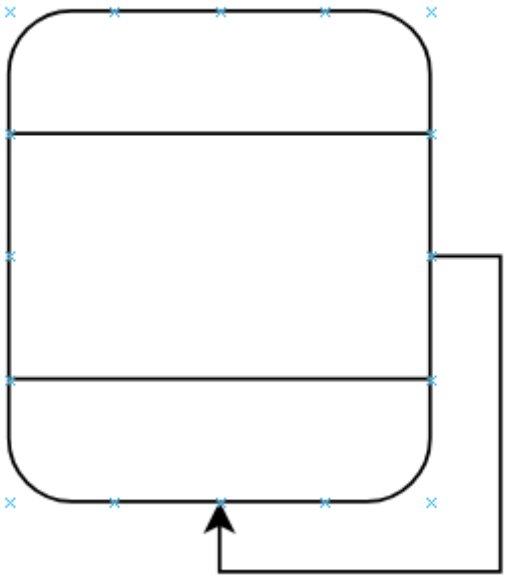
Los diagramas de clases se conectan mediante líneas, notaciones textuales o números de multiplicidad para poder observar sus relaciones.

- **Asociación**: Una asociación conecta dos clases en cualquier conexión lógica. Es la relación mas utilizada y se ve como una línea recta. Las asociaciones pueden ser bidireccionales, es decir, que ambas clases saben que la otra existe o unidireccionales, las cuales se representan mediante una punta de flecha plana.

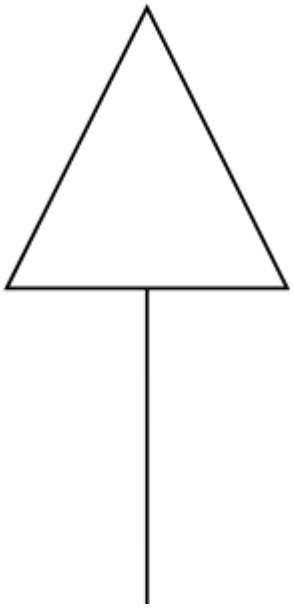
- **Agregación y composición**: La agregación y la composición son subconjuntos de la relación de asociación.
    - En la agregación las clases secundarias siguen existiendo independientemente de la principal y se representan con un diamante vacío junto a la clase principal.
    - La composición es lo opuesto y se representa mediante un diamante relleno junto a la clase principal.



- **Asociación reflexiva**: La asociación reflexiva es la única relación unilateral y está representada mediante una línea que hace un bucle junto al modelo de la clase. Su propósito es representar una relación entre una instancia de una clase y otra instancia de la misma clase.



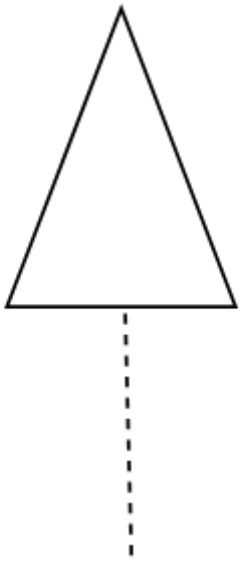
- **Herencia y generalización**: Este tipo de relaciones representan como clases heredan propiedades de otras clases. Esta relación se representa mediante una línea recta y una forma de triángulo hueco apuntando a la clase independiente.



- **Dependencia**: Esta relación se representa mediante una línea escalonada con una flecha que parte desde la clase dependiente. Esta

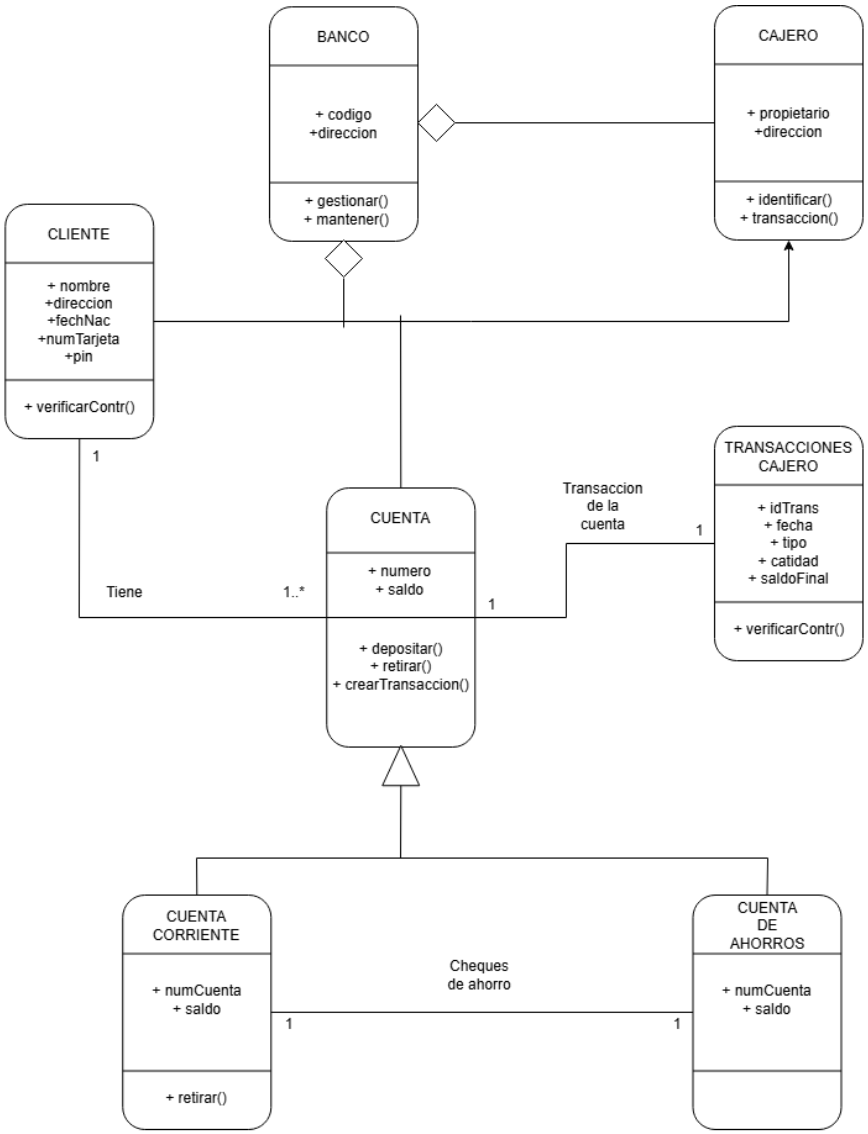
relación representa que si hay un cambio en la superclase lo habrá en la subclase.

- **Realización**: Las clases conectadas por este tipo de relación indica que una clase utiliza un comportamiento definido por otra clase. Hay dos elementos en esta relación, la superclase y la subclase. Esta relación se representa mediante una línea escalonada y una forma de triángulo hueco.



- **Multiplicidad**: Esta relación es un conjunto de números colocados en las intersecciones de líneas. La multiplicidad define cuantas instancias participan en una relación. Los dígitos muestran el número de instancias de una clase que están vinculadas a una instancia de otra clase.

EJEMPLO



En este ejemplo podemos observar la representación de las diferentes clases de un sistema de un banco. Las clases se ven representadas mediante un rectángulo con las esquinas redondeadas divididas en secciones para los atributos, el nombre y los métodos. Estas clases están conectadas mediante relaciones diferentes como de herencia o de asociación. También podremos observar una descripción de la relación junto a su multiplicidad.

Diagramas de actividades

Un diagrama de actividades es una representación visual de acciones, restricciones, requisitos y otros factores que intervienen en la realización de tareas.

Una actividad en UML es un método para crear representaciones visuales que muestran las relaciones entre diferentes elementos. Estos diagramas representan el flujo de lo que sucede en el sistema, visualizando el comportamiento dinámico de un sistema mostrando el flujo de una actividad a otra. Cada componente de un diagrama de actividades se llama elemento y la actividad es el elemento de mas alto nivel dentro del diagrama.

Este tipo de diagramas se pueden usar para rastrear la lógica de un algoritmo, representar los pasos en un caso de uso, modelar el proceso de negocio o flujo de trabajo o describir el flujo de acciones para cualquier actividad. También son muy útiles en las etapas de planificación para aclarar un flujo de trabajo o para las operaciones de un proyecto que ya está completo para poder entenderlo o mejorarlo.

Mapear actividades de esta manera puede revelar nueva información, ayudar a identificar ineficiencias y proporcionar otros beneficios importantes.

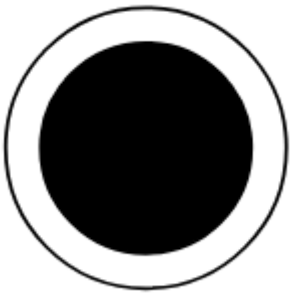
ELEMENTOS DE LOS DIAGRAMAS DE ACTIVIDADES

Estos son los elementos que componen un diagrama de actividades:

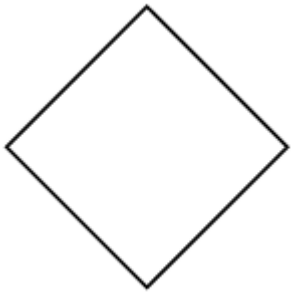
- **Acción**: Este elemento representa un paso dentro de una actividad. Las acciones suelen representarse con rectángulos con esquinas redondeadas.

- **Nodo de inicio**: Esto es el punto de partida o el evento desencadenante de una actividad. Este se representa por un círculo negro sólido.

- **Nodo final**: Este es el paso final de una actividad, cuando el flujo dinámico terminado es retomado por otra actividad. Se representa mediante un círculo negro solido dentro de un círculo blanco mas grande.



- **Nodo de control**: Este elemento controla el flujo entre otros nodos. Se representa mediante un diamante con un flujo de entrada y dos o mas flujos de salida.



- **Flujos de control**: Estos también se conocen como bordes de control. Representa el flujo de control de un elemento a otro, con una línea sólida.

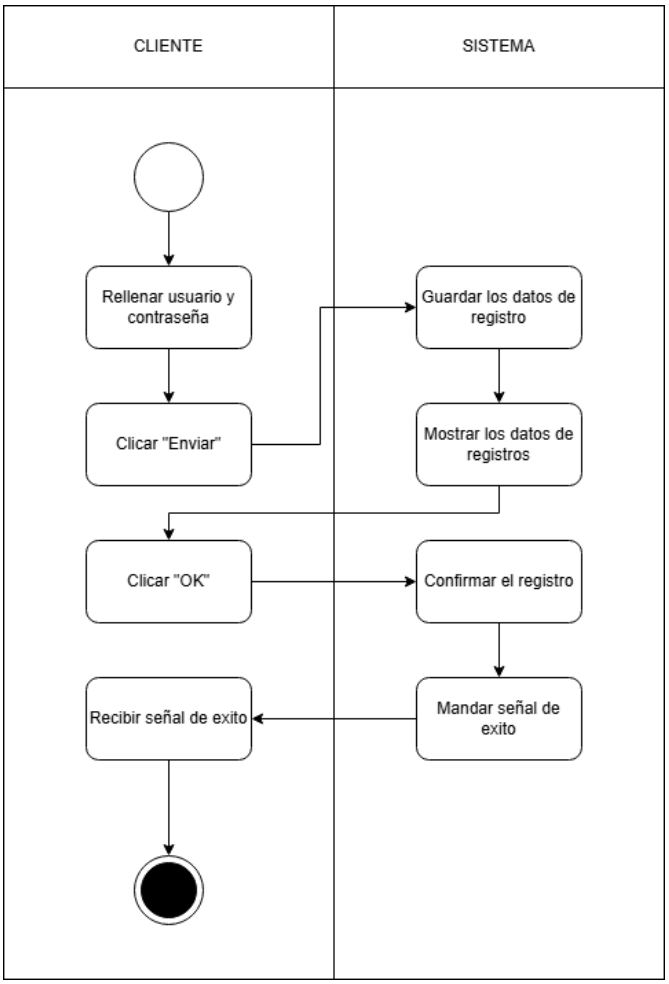
- **Flujos de objetos**: También se conocen como bordes de objeto. Estos representan el flujo dirigido de objetos de un elemento a otro y se representan mediante una línea punteada.

- **Partición de actividad**: Es una columna o fila que se utiliza para mostrar áreas de responsabilidad para diferentes actores. Estos también se conocen como diagramas de carriles.

COMPARACIÓN DE LOS DIAGRAMAS DE ACTIVIDADES CON OTROS DIAGRAMAS

Un diagrama de actividades describe el flujo de actividad de sistema a sistema, pero dependiendo del diseño y de las necesidades puede que no aborde a los usuarios ni a los actores. Sin embargo, en los diagrama de caso de uso si que se preocupa por los usuarios y los actores.

EJEMPLO



En este ejemplo podemos ver como es el proceso de registro de un cliente dividido en dos columnas. En la primera podemos ver la acciones del cliente y en la segunda las acciones del sistema.

Conclusión

El Lenguaje de Modelado Unificado (UML) es una herramienta fundamental en el desarrollo de software moderno, permitiendo a los equipos visualizar, documentar y comunicar claramente la estructura y comportamiento de un sistema. A lo largo de este trabajo, se exploraron sus conceptos clave, elementos estructurales y de comportamiento, así como sus principales diagramas, como los de casos de uso, clases y actividades.

El uso de UML no solo facilita la comprensión entre los distintos actores involucrados en un proyecto, sino que también permite detectar errores tempranamente, fomentar la reutilización de componentes y mejorar la organización del sistema. En definitiva, UML es una herramienta poderosa y versátil que, cuando se utiliza correctamente, contribuye significativamente al éxito de proyectos de software complejos.