Curso de UML

Diagrama de Clases desde Cero

Christian Ramirez
@christian_ramireezz

Temario General

- Introducción a UML
- Diagramas de Clases
- Relaciones entre Clases
- Asociaciones y Multiplicidad
- Herencia y Generalización
- Dependencias
- Agregación y Composición
- Visibilidad y Ámbito
- Ejercicios y Práctica

Importante

Conocimientos previos

- Objetos, atributos y Clases
- 4 Principios de POO
- Clases abstractas e Interfaces
- Contratos y realización

POO

https://www.youtube.com/watch?v=m-whwkqBKgk

@latecnologiaavanza

UML

UML

- UML (Lenguaje Unificado de Modelado, por sus siglas en inglés Unified Modeling Language)
- Es un lenguaje visual estándar utilizado para especificar, visualizar, construir y documentar los componentes de sistemas de software
- Es un lenguaje gráfico que se utiliza en la ingeniería de software para modelar y diseñar sistemas de software

UML

- **Modelo:** Es una representación simplificada de un sistema o proceso real. Se utiliza para entender, analizar y comunicar cómo funciona el sistema o proceso en cuestión

- **Modelado:** Es el proceso de crear modelos. En el desarrollo de software, modelar significa crear representaciones gráficas y abstractas del sistema que estás construyendo

- Un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema

UML

- **Unificado:** es un término que hace referencia a la idea de integrar o combinar varios elementos, sistemas o componentes en uno solo, de manera que funcionen como una unidad coherente

- Antes de UML, existían varias metodologías de modelado que usaban diferentes notaciones y enfoques

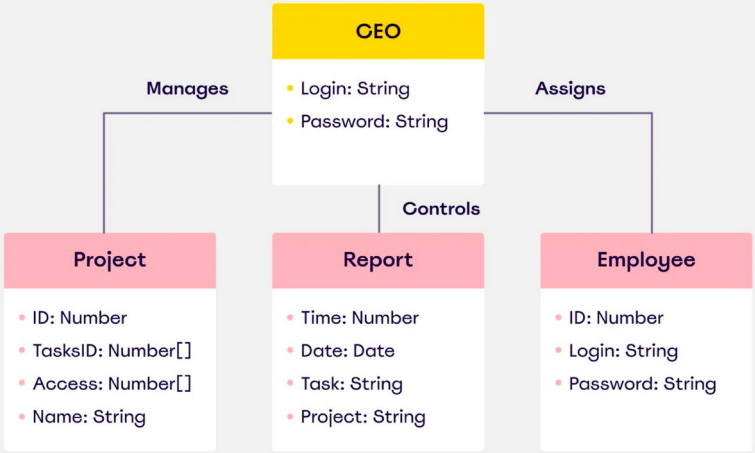
- UML fue diseñado para unificar estas diferentes notaciones en un solo lenguaje, integrando conceptos y técnicas de estas metodologías para ofrecer una solución estándar

¿Por qué es necesario?

- **Comunicación efectiva:** UML proporciona un lenguaje común para los desarrolladores, diseñadores, analistas y otros miembros del equipo
- **Documentación visual:** Ofrece una representación gráfica de los sistemas, lo que facilita la comprensión de estructuras complejas
- **Planificación y diseño:** Ayuda en la planificación y el diseño detallado de los sistemas antes de su implementación

Diagramas UML

Los diagramas en UML se agrupan en dos categorías principales: diagramas estructurales y diagramas de comportamiento.



Diagramas Estructurales

Diagramas estructurales: Representan los componentes estáticos del sistema

- Diagrama de Clases
- Diagrama de Componentes
- Diagrama de Objetos
- Diagrama de Despliegue
- Diagrama de Paquetes

Diagramas de Comportamiento

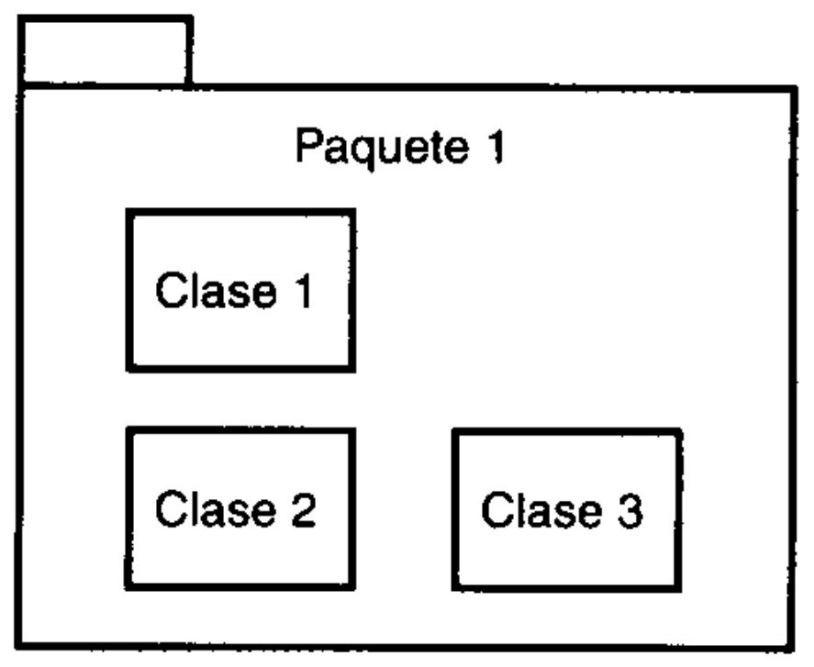
Diagramas de comportamiento: Representan la interacción entre los componentes y el flujo de control dentro del sistema

- Diagrama de Casos de Uso
- Diagrama de Secuencia
- Diagrama de Actividades
- Diagrama de Estados

Otras Características

- **Paquetes:** Un paquete es una forma de organizar elementos del modelo, como clases o diagramas completos, en grupos lógicos
- Ayudan a reducir la complejidad de los diagramas y a gestionar grandes sistemas dividiéndolos en subcomponentes más manejables
- **Ejemplo:** Un paquete GestiónDeUsuarios podría contener clases relacionadas con usuarios, roles y permisos dentro de un sistema

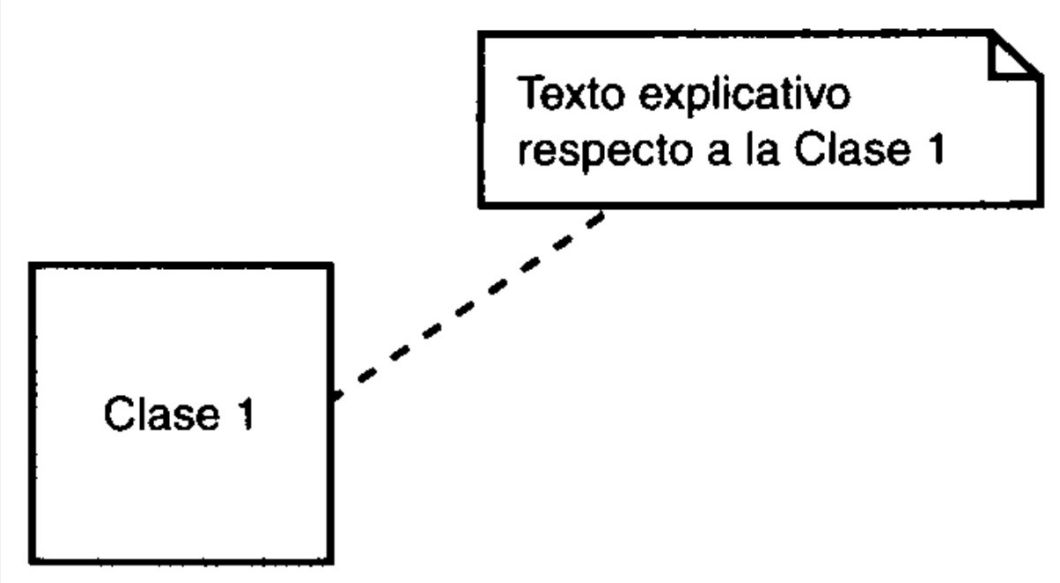
Paquetes



Otras Características

- **Notas:** Las notas son comentarios adicionales que se pueden agregar a los diagramas UML para proporcionar explicaciones, aclaraciones o detalles extra que no se reflejan directamente en el diagrama
- **Ejemplo:** Una nota puede explicar un cálculo complejo de un atributo o justificar una relación entre clases

Notas



Otras Características

- **Estereotipo:** Son una forma de extender o personalizar los elementos estándar del lenguaje
- UML define ciertos elementos básicos(como clases, asociaciones, interfaces, etc)
- A veces necesitas agregar información o características que no están incluidas de manera predeterminada en estos elementos

Estereotipos

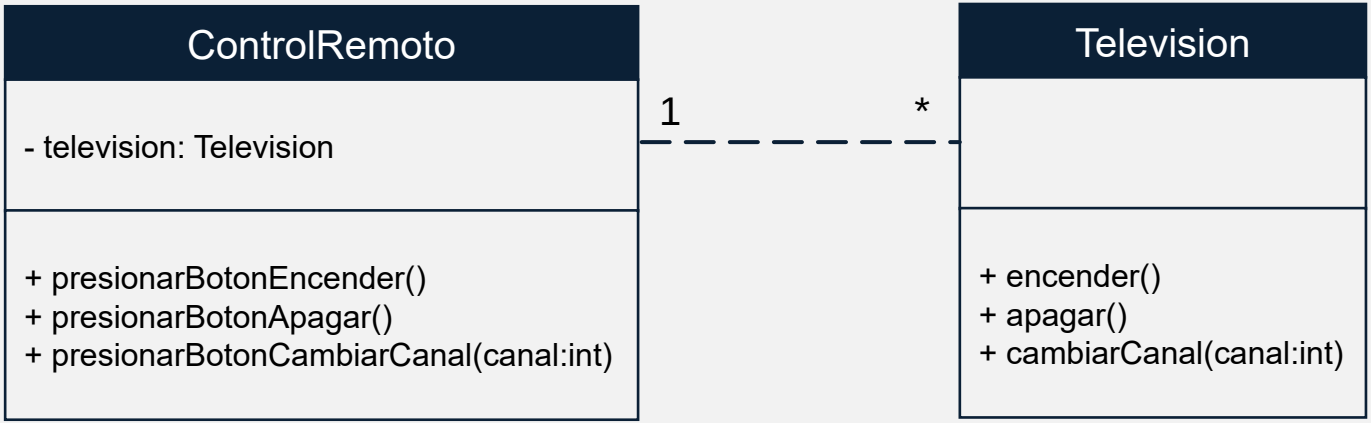
Un estereotipo permite crear nuevos elementos a partir de otros existentes

POO

Envío de Mensajes

- El envío de mensajes es una forma de representar cómo los objetos (o instancias de clases) en un sistema se comunican entre sí para realizar operaciones y coordinar acciones
- El envío de mensajes se refiere a la transferencia de información o solicitud de acción de un objeto a otro

Envío de Mensajes



Envío de Mensajes

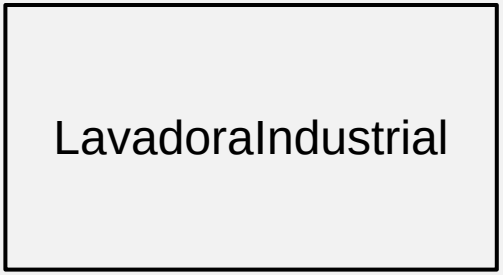
- Un control remoto está diseñado para controlar una sola televisión (al menos en este ejemplo simple)
- Una televisión puede ser controlada por uno o varios controles remotos
- Entre las clases ControlRemoto y Television, podríamos representar una asociación. ControlRemoto depende de Television para ejecutar sus métodos

Orientación Objetos

@latecnologiaavanza

Concepción de una Clase

Un Rectángulo es el símbolo que representa una clase

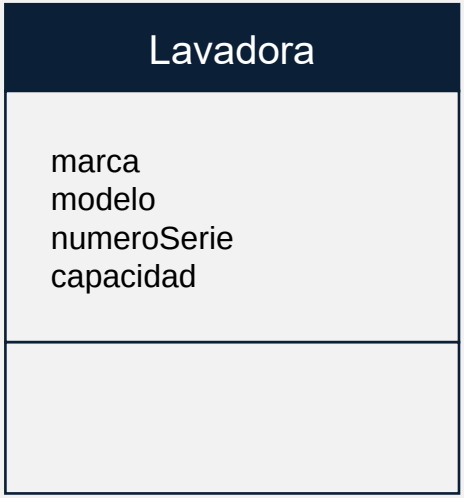


Concepción de una Clase

Si la clase “Lavadora” es parte de un paquete llamado “Electrodomesticos”, el nombre será “Electrodomesticos::Lavadora”

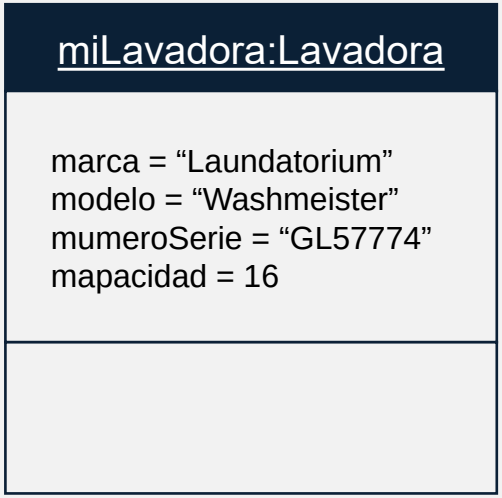
Atributos

Un atributo es una propiedad o característica de una Clase



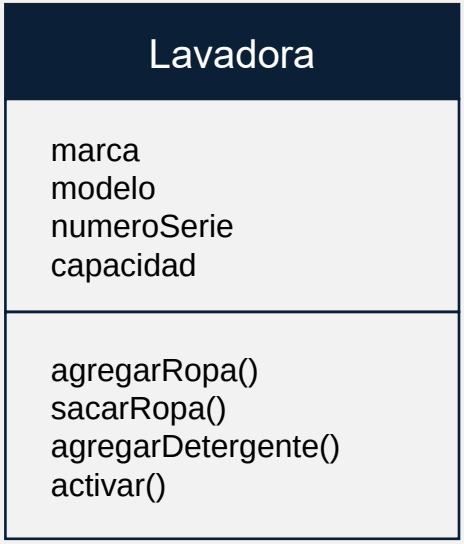
Atributos

El nombre **miLavadora** es una instancia con nombre; pero también es posible tener una instancia anónima, como **:Lavadora**



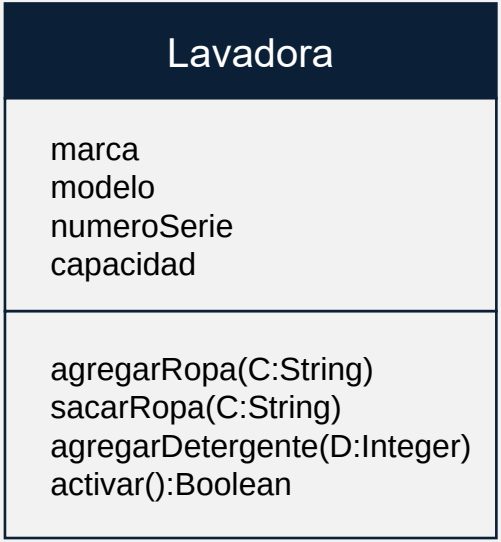
Operaciones

Una operación es algo que la clase pueda realizar. El nombre de una operación se escribe en minúsculas si consta de una sola palabra



Firma de Operación

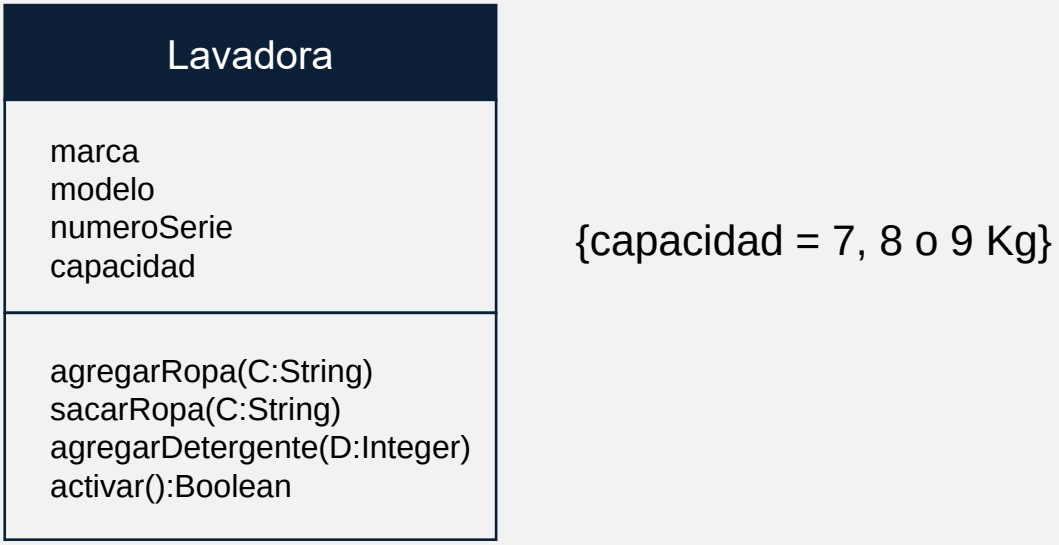
Firma De Operación: Se refiere a la descripción completa de una operación o método de una clase, incluyendo su nombre, parámetros que recibe, los tipos de esos parámetros, y el tipo de valor que devuelve



Atributos, operaciones y concepción

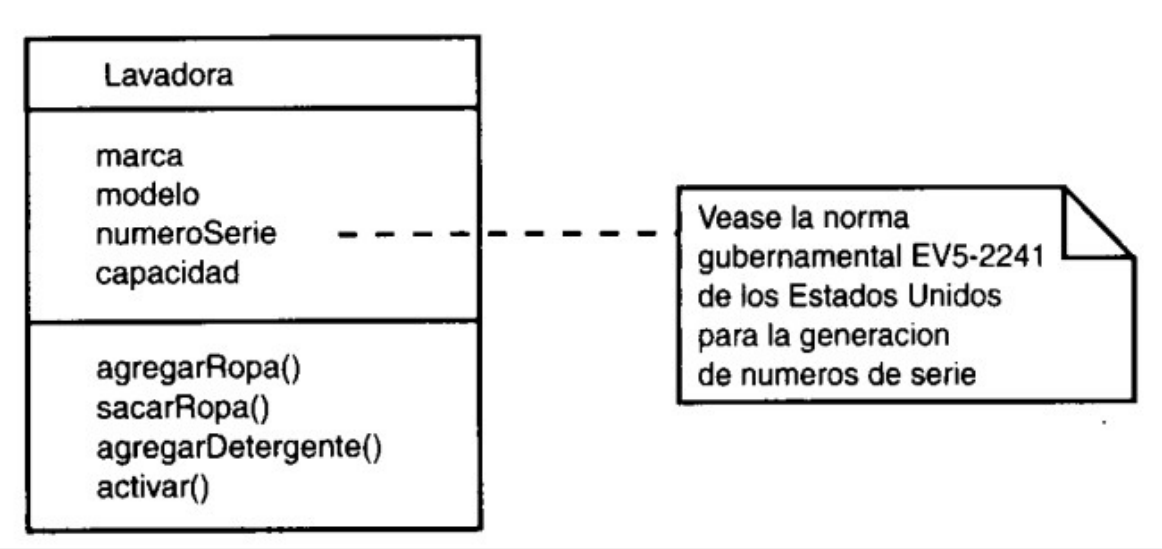
Restricciones

Una manera más formal es agregar una **restricción**, un texto libre bordeado por llaves; este texto especifica una o varias reglas que sigue la clase



Notas Adjuntas

Las notas adjuntas en UML son elementos gráficos utilizados para agregar comentarios, aclaraciones o información adicional a un diagrama

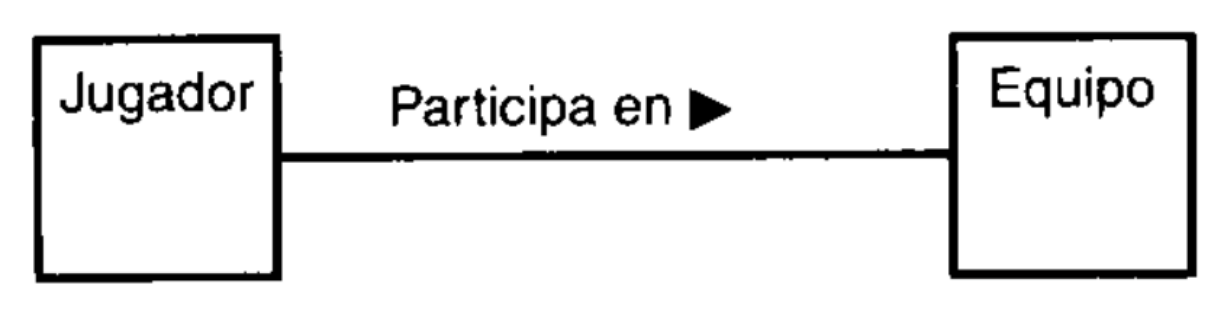


Práctica

Relaciones

Asociaciones

- Relación básica que conecta dos o más clases
- Ejemplo: La asociación entre un jugador y un equipo
- Un jugador participa en un equipo

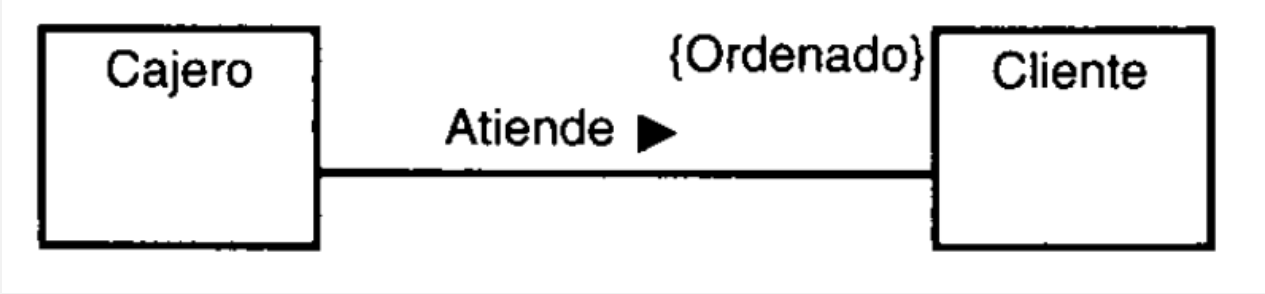


Restricciones Asociaciones

- Son condiciones o reglas específicas que limitan o definen el comportamiento de una asociación entre clases
- Ayudan a representar de manera clara las reglas específicas que deben cumplirse en las relaciones entre objetos, garantizando que el modelo sea fiel a los requisitos del sistema

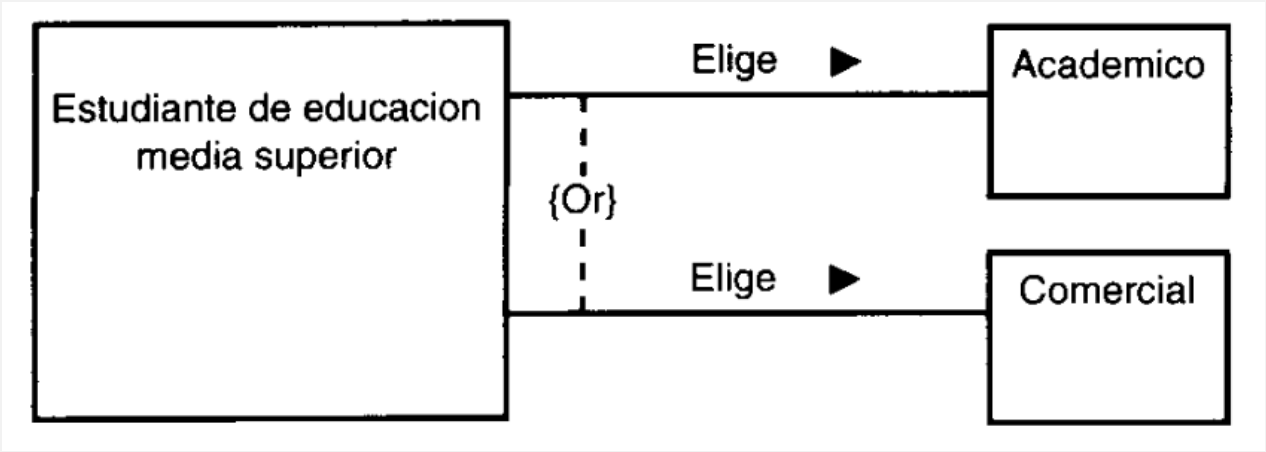
Ejemplos prácticos

- Un **Cajero** solo puede atender a un **Cliente** si sigue un orden establecido (En el orden en que se encuentre en la formación). Esto se representa con la palabra {ordenado} junto a la línea de asociación.



Ejemplos prácticos

- Un estudiante elige entre un curso **Académico** o **Comercial**. La línea punteada con {Or} indica que solo puede seleccionar una de las dos opciones



Clases de Asociación

- Son clases que modelan atributos y operaciones de una asociación entre dos o más clases.
- Se representan con una línea discontinua que conecta la clase de asociación a la asociación principal
- Permiten enriquecer una relación entre clases al añadir atributos y comportamientos específicos que pertenecen exclusivamente a esa relación

Multiplicidad

- La multiplicidad define la cantidad de objetos de una clase que pueden estar asociados con un objeto de otra clase
- Permite especificar las cardinalidades de las relaciones, como "uno a muchos" o "muchos a muchos", garantizando que el modelo respete las reglas de negocio

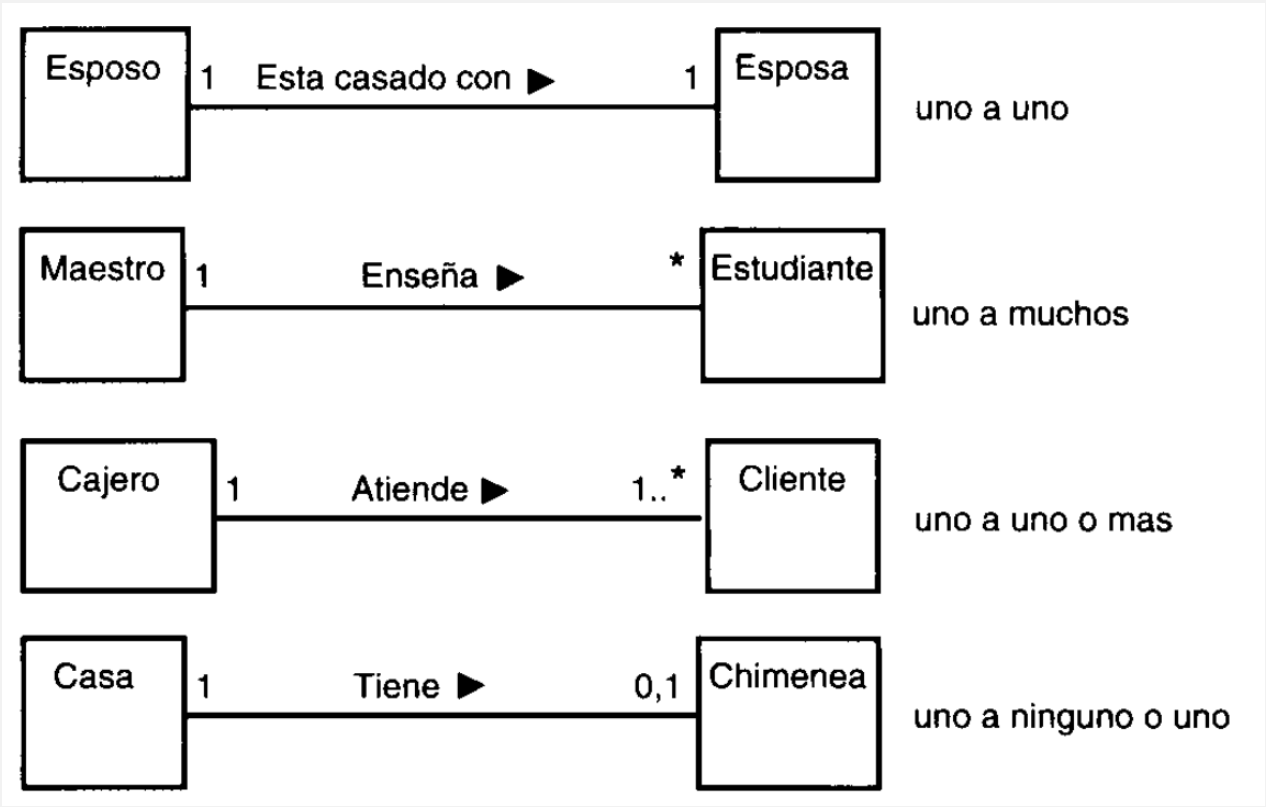
Multiplicidad

- Un Jugador puede participar en varios Equipos, pero cada equipo tiene al menos un jugador
- Se establece que un Jugador participa en 0 o más equipos (0..*)

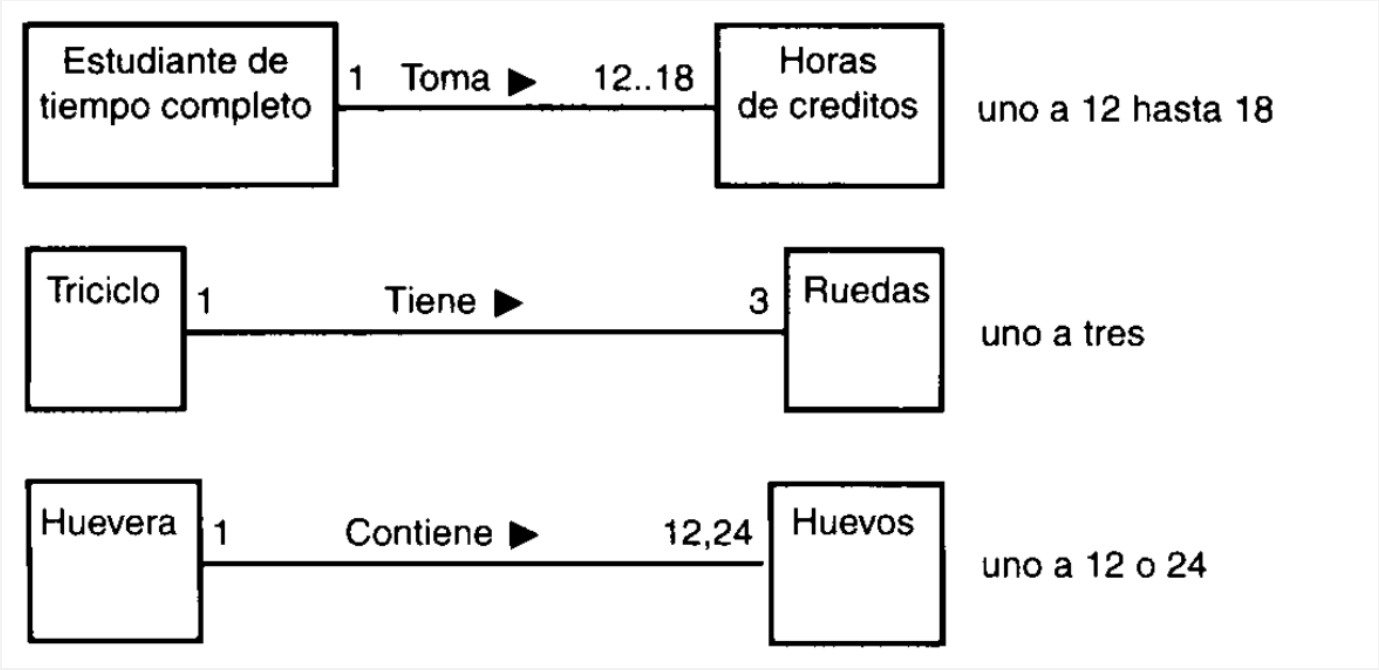
Tipos de Multiplicidad

- Una clase puede relacionarse con otra en un esquema de uno a uno, uno a muchos, uno a uno o más, uno a ninguno o uno
- En UML se utiliza el asterisco (*) para representar más y representar muchos
- En un contexto O se representa por dos puntos, como en “1..*” (uno o muchos)

Tipos de Multiplicidad



Tipos de Multiplicidad

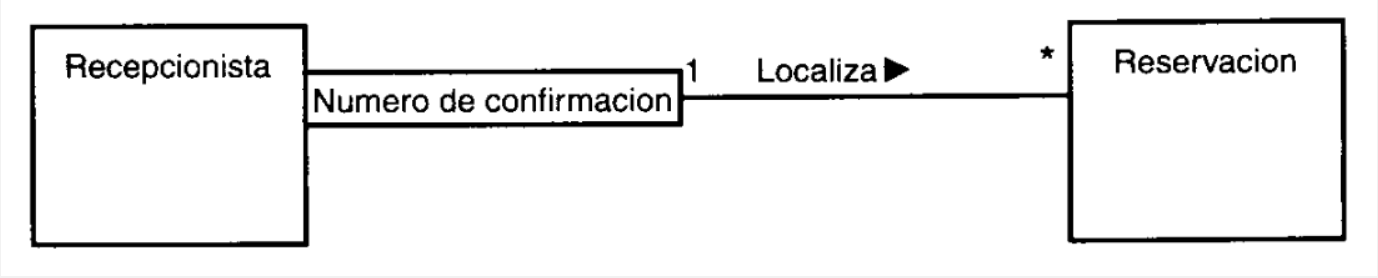


Asociaciones Calificadas

- Son asociaciones que utilizan un atributo identificador (calificador) para localizar un objeto relacionado de forma más eficiente
- Permiten optimizar las búsquedas en relaciones donde una clase puede estar asociada con muchas instancias de otra clase.

Asociaciones Calificadas

- Se identifica la relación: Recepcionista y Reservación
- Se utiliza un atributo (Número de confirmación) para calificar la relación
- Se representa el calificador como un pequeño rectángulo adjunto a la clase asociada

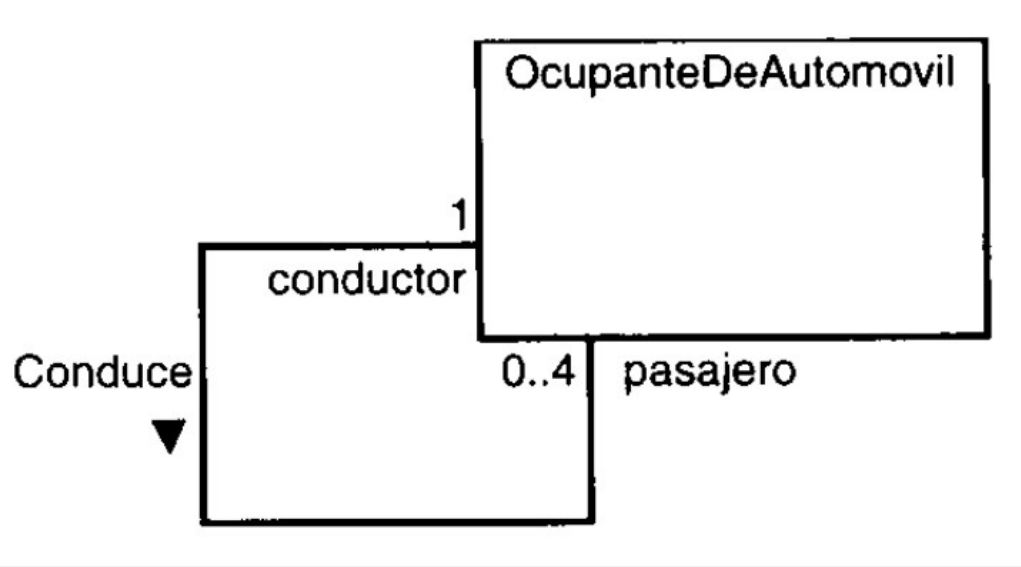


Asociaciones Reflexivas

- Son asociaciones donde una clase se relaciona consigo misma
- Representan relaciones jerárquicas o recursivas, como empleados que supervisan a otros empleados

Asociaciones Reflexivas

- Se define la relación: OcupanteDeAutomóvil puede tener distintos roles
- Se establece la multiplicidad: un conductor puede llevar a varios pasajeros (0..*)
- Se dibuja la relación reflexiva con una línea que regresa a la misma clase



Práctica

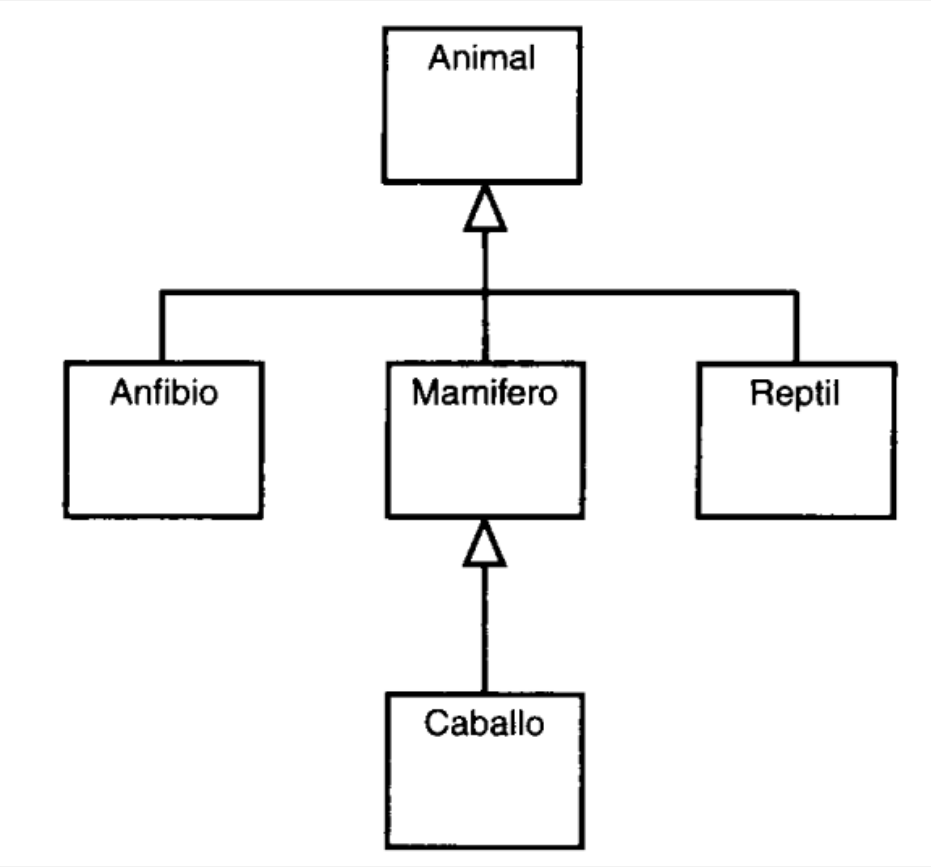
Herencia y Generalización

- Es un mecanismo de modelado que permite que una clase (llamada clase hija o derivada) herede atributos y métodos de otra clase (llamada clase padre o base)
- Este mecanismo permite representar relaciones jerárquicas y promover la reutilización de código

Herencia y Generalización

- La **generalización** es el proceso inverso a la herencia: identifica las características comunes de varias clases específicas para crear una clase más general
- La **generalización** es el concepto abstracto que da lugar a la herencia en UML
- Se representan de la misma manera en los diagramas: con una línea conectada a un triángulo en la parte superior

Herencia y Generalización



Herencia y Generalización

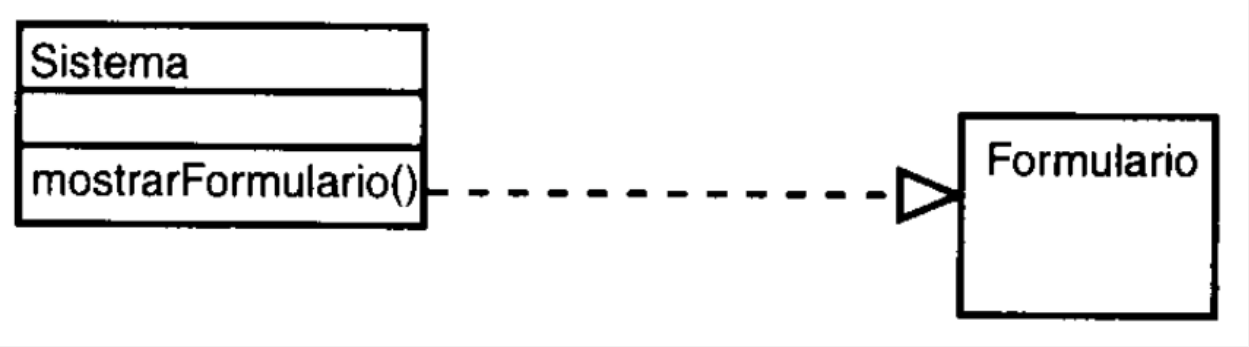
Aspecto	Herencia	Generalización
Definición	Relación jerárquica entre clases, donde una hija hereda de la padre	Proceso de identificar clases comunes para generalizar
Enfoque	Implementación en el diseño del sistema	Modelado conceptual antes del diseño
Uso	Define atributos y métodos en la clase padre que son reutilizados	Organiza las clases y simplifica el modelo

Dependencia

- Es una relación en la que un elemento (generalmente una clase) utiliza a otro para cumplir con una función o tarea
- Esta relación indica que un cambio en el elemento dependido (proveedor) podría impactar al elemento dependiente (consumidor)
- Se representa como una línea discontinua con una flecha que apunta hacia el elemento del cual se depende

Dependencia

- La Clase **Sistema** tiene **mostrarFormulario(f:Form)**
- El formulario que el sistema desplegará, dependerá, obviamente, del que elija el usuario



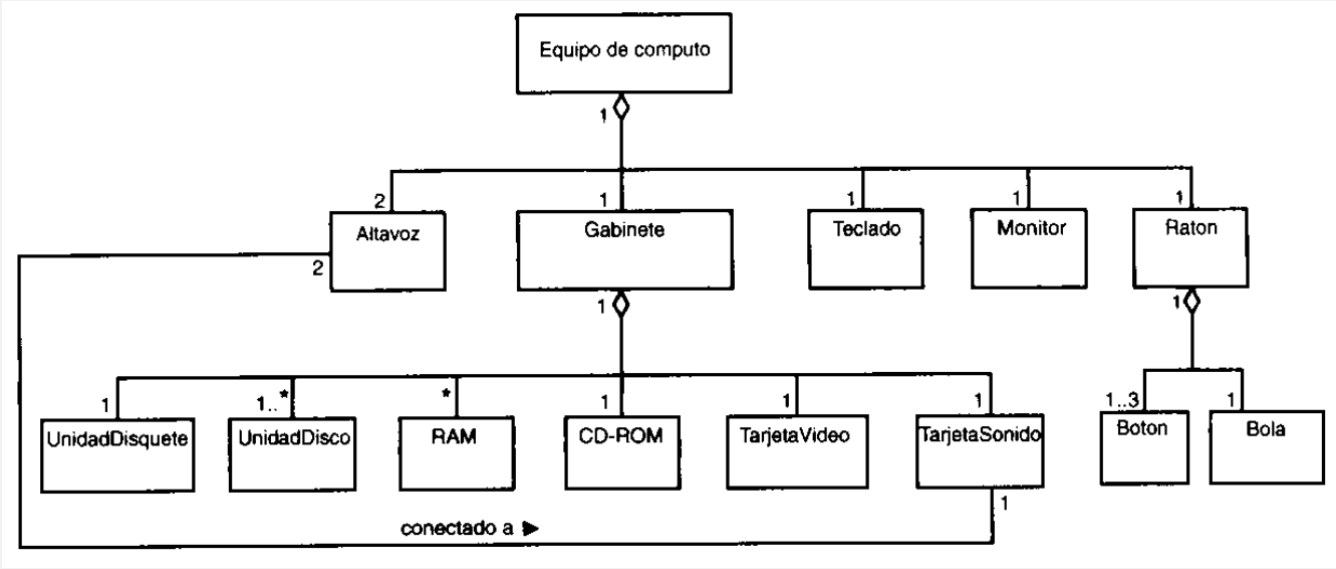
Dependencia y Asociación

Criterio	Dependencia	Asociación
Relación	Se usa de manera momentánea	se mantiene mientras ambas clases existan
Uso	La clase dependiente utiliza métodos o atributos de otra clase	Una clase está relacionada estructuralmente con otra
Representación	Línea discontinua con flecha	Línea continua con multiplicidades

Agregaciones

- La agregación es una relación especial entre clases donde una clase (el "todo") está compuesta por otras clases (los "componentes")
- Se utiliza para modelar relaciones de "tiene un"
- **Ejemplo:** Un sistema de cómputo que incluye componentes como la unidad de disco, monitor, RAM, entre otros

Agregaciones

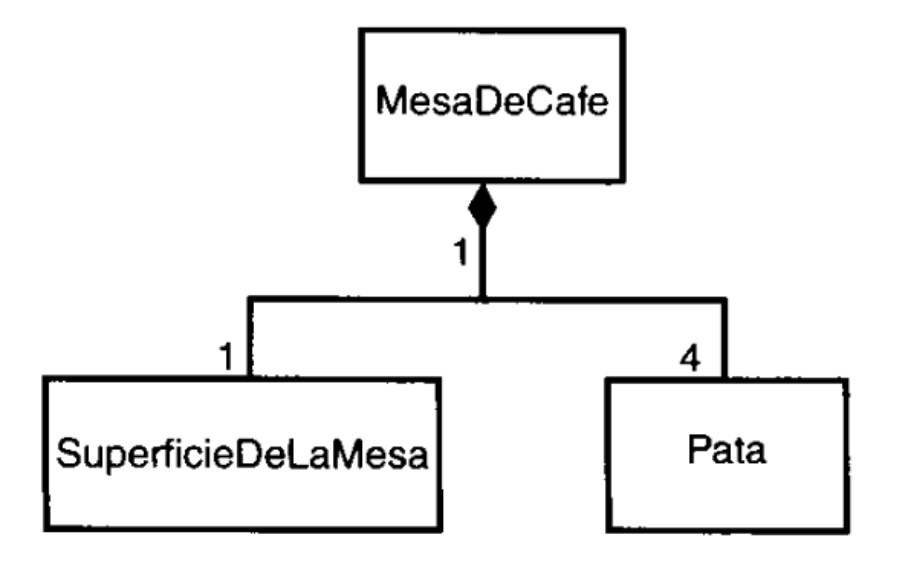


Composiciones

- La composición es un tipo más fuerte de agregación. Los componentes dependen completamente del "todo"; si el "todo" deja de existir, los componentes también
- **Ejemplo:** Una mesa con componentes como la superficie y las patas
- Diferencia clave con agregación: En la composición, la relación es de dependencia total

Composiciones

- En una composición, cada componente pertenece solamente a un todo. Un rombo relleno representa esta relación



Contextos

- Representación gráfica que muestra de manera simplificada el sistema que estás modelando en relación con los actores externos o entidades que interactúan con él
- Se enfoca en cómo se conecta y comunica con otros sistemas o actores externos, sin detallar la estructura interna del sistema
- **Contexto**: Se refiere al entorno o las circunstancias que rodean un evento, una actividad o un objetivo específico

Diagrama de Contexto de Composición

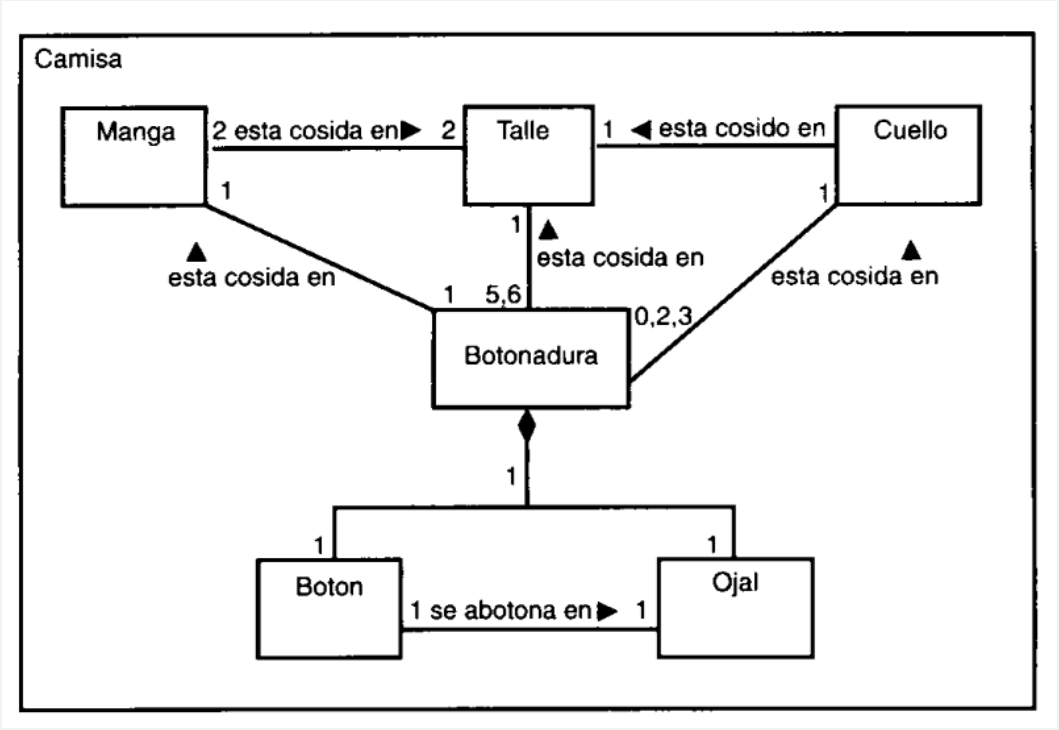
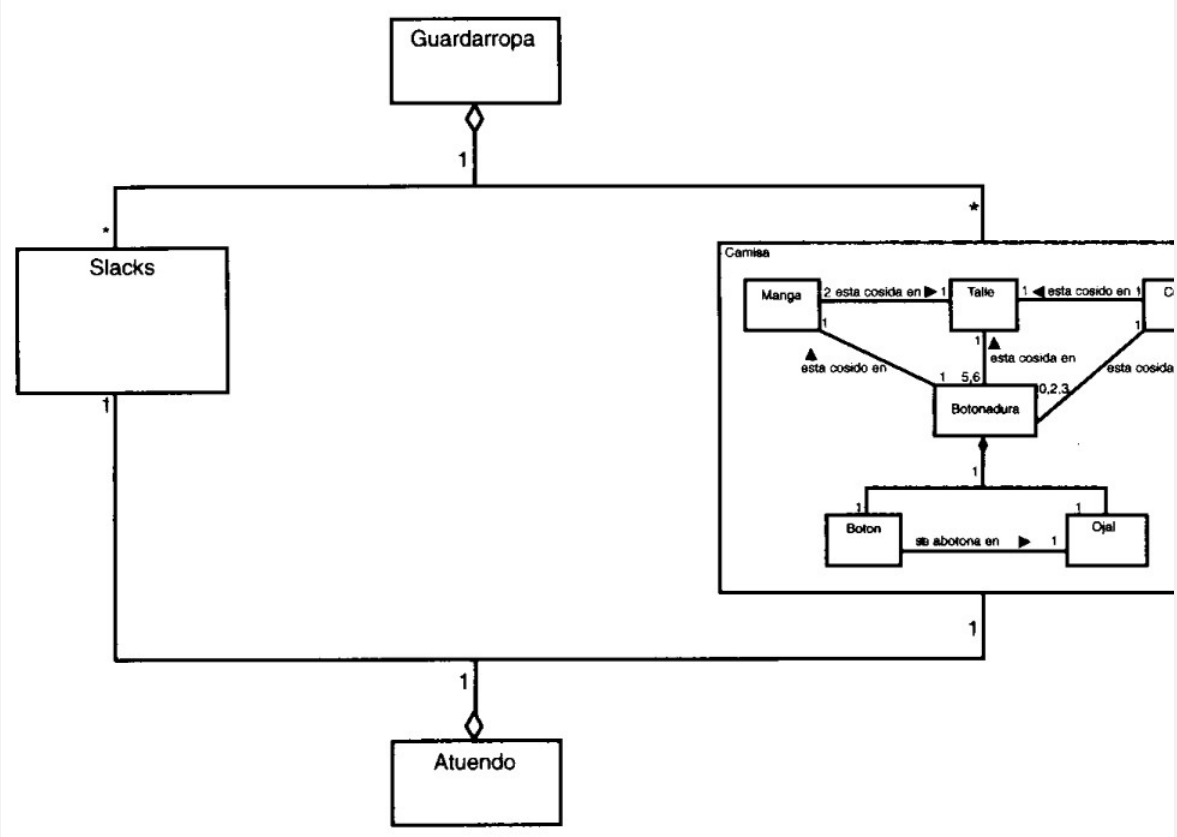


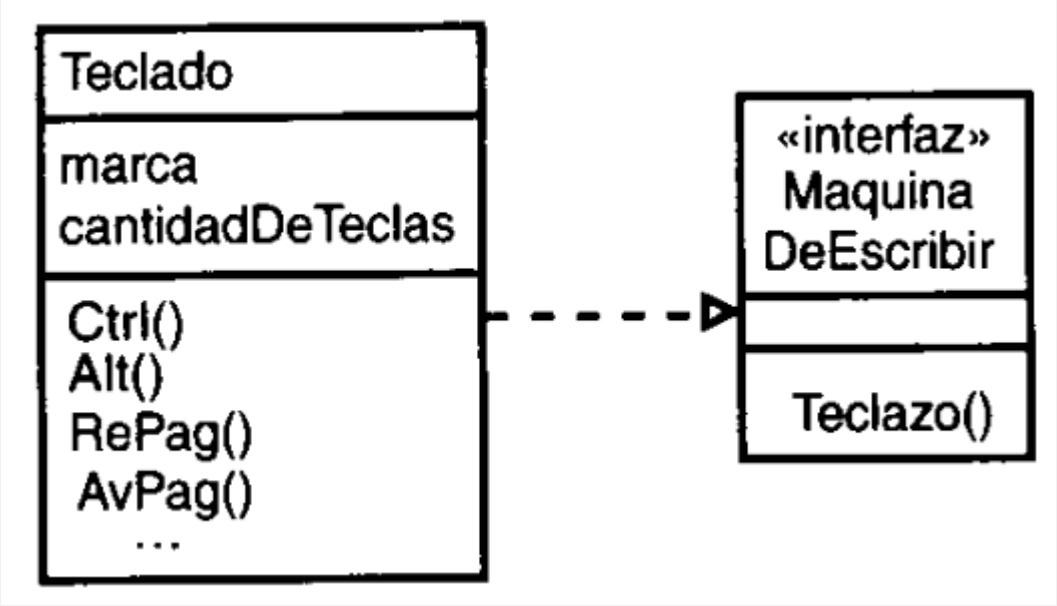
Diagrama de Contexto del Sistema



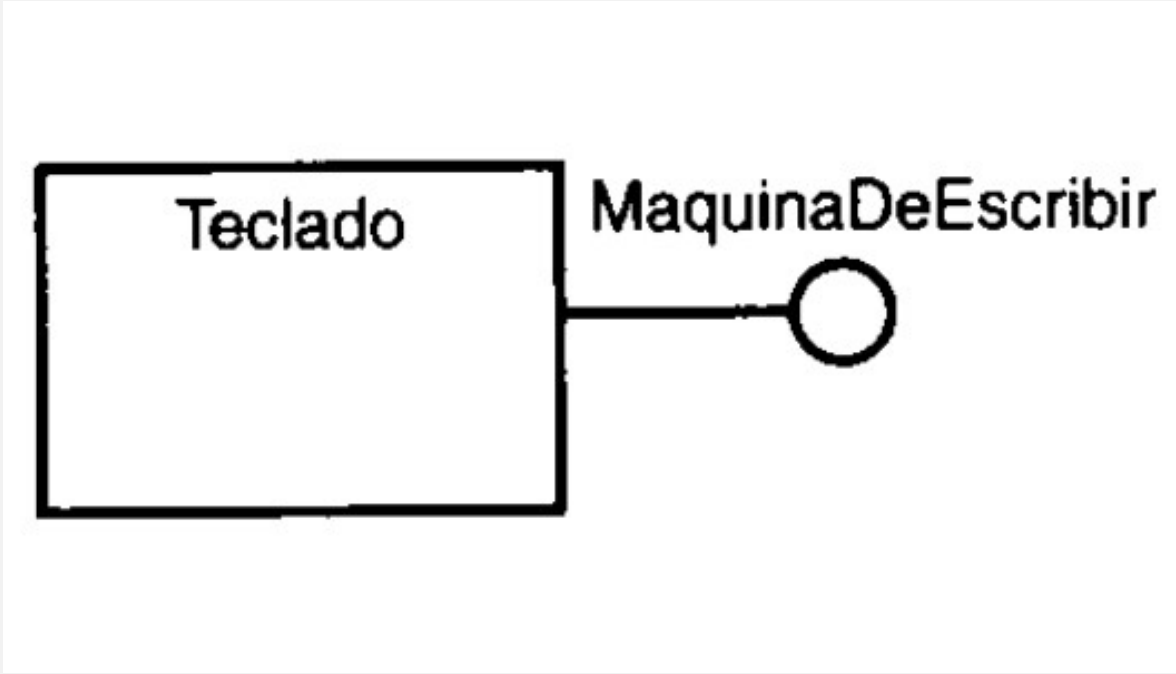
Interfaces y Realización

- Una interfaz define un conjunto de operaciones reutilizables por otras clases
- **Realización:** Es la implementación de esas operaciones en una clase concreta
- Una interfaz se muestra como un rectángulo con una línea discontinua conectada a las clases que la implementan
- **Ejemplo:** Un teclado que implementa operaciones como escribir, borrar y tabular

Interfaces y Realización



Interfaces y Realización



Visibilidad

- Se refiere al nivel de acceso que tienen otros objetos o clases a los atributos (propiedades) y métodos de una clase
- Controlar la visibilidad es fundamental para el principio de encapsulamiento, ya que permite ocultar detalles internos de una clase y exponer solo lo necesario

Modificadores de Acceso

- Palabra clave que se utiliza para controlar el nivel de visibilidad y el alcance de los atributos (variables) y métodos (funciones) de una clase
- **Controlar el acceso:** Permitir acceso únicamente a las partes del programa que realmente lo necesitan
- **Aumentar la seguridad:** Restringir la modificación o el uso indebido de los datos
- **Encapsular datos:** Ocultar detalles internos de una clase para proteger su integridad

Visibilidad

- **public:** Son accesibles desde cualquier parte del programa, es decir, desde cualquier otra clase u objeto
- **protected:** Son accesibles desde la propia clase, sus subclases y otras clases dentro del mismo paquete (en lenguajes como Java)
- **private:** Accesibles desde dentro de la misma clase, lo que significa que ninguna otra clase u objeto puede interactuar directamente con ellos

Ámbito

- Se refiere al contexto o alcance en el que una variable, método, o clase es visible y puede ser accesible dentro de un programa
- En otras palabras, el ámbito determina en qué partes del código puedes usar o referenciar un elemento específico

Visibilidad y Alcance

- **Visibilidad:** Está determinada por los modificadores de acceso como public, protected, private, o el ámbito por defecto en algunos lenguajes (como Java o C++)
- **Alcance:** Es el rango dentro del cual ese elemento existe o es válido. Una vez fuera de ese rango, el elemento ya no es accesible ni tiene efecto

Tipos de Ámbito

Tipo	Definición	Ejemplo
Local	El elemento es visible solo dentro del bloque donde se define	Una variable declarada dentro de un método o bloque if
De Clase	El elemento es visible para todos los métodos de la misma clase (dependiendo del modificador)	Un atributo declarado directamente dentro de una clase con modificador private, public o protected
De Paquete	El elemento es visible solo dentro del mismo paquete	Una clase o método sin modificador explícito (default en Java)
Global	El elemento es visible en todo el programa (no común en OO, pero sí en JavaScript o Python)	Variables declaradas en el nivel más alto del programa, como var x = 10 en JavaScript.

Práctica

Nuestra comunidad

@latecnologiaavanza

@latecnologiaavanza

latecnologiavanza

La Tecnología Avanza