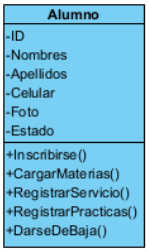
1. Definición de clase

Una clase en UML es un elemento que representa un conjunto de objetos que tienen las mismas propiedades y comportamientos. Se utiliza para modelar la estructura de un sistema, especificando las características de los objetos que se encuentran en él. se representa gráficamente mediante un rectángulo dividido en tres secciones:1. Nombre de la clase 2. Atributos de la clase (propiedades) 3. Métodos de la clase (operaciones)



2. Definición de clase de Análisis

Se utilizan en la fase de análisis del proceso de desarrollo de software y se centran en describir el comportamiento, las propiedades y las relaciones de las entidades relevantes del sistema.

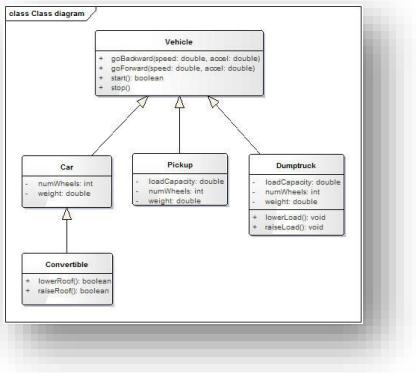
Son una forma de representar los requisitos del usuario y los conceptos del dominio del problema no necesariamente están vinculadas a la implementación del sistema sino para entender el problema

3. Definición de clase Abstracta

Proporciona una plantilla para las clases que la extienden y define la estructura y el comportamiento común que deben tener las clases derivadas.

Las clases abstractas se utilizan en el análisis de sistemas para representar conceptos del mundo real que tienen características comunes

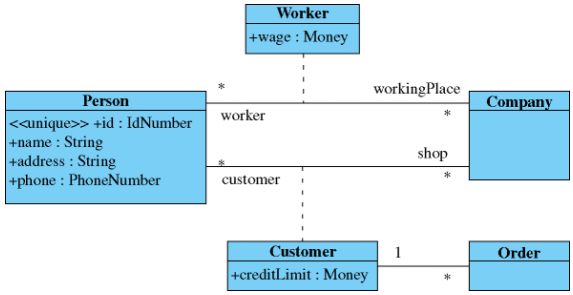
En UML, se utiliza el estereotipo <<abstract>> para indicar que una clase es abstracta.



4. Definición de clase Asociativa

Representan una relación compleja entre dos o más clases en un diagrama de clases. Esta clase actúa como intermediaria entre las otras clases, lo que permite simplificar la relación y modelarla de manera más eficiente.

En otras palabras, una clase asociativa de análisis se utiliza para modelar relaciones *que no pueden ser expresadas directamente entre dos clases*. En lugar de agregar atributos o métodos a las clases relacionadas, *se agrega una clase asociativa* que contiene la información necesaria para describir la relación.



5. Definición de clase Boundary

Es una clase que representa la interacción entre el sistema y su entorno externo

Describe las entradas y salidas que el sistema intercambia con entidades externas, como usuarios, otros sistemas o dispositivos

Las clases de análisis de límites son una parte importante del modelado UML, ya que ayudan a definir las interfaces entre el sistema y su entorno y brindan una comprensión clara de las entradas y salidas del sistema.

6. Definición de clase de Entidad

Es una clase que representa un objeto o concepto en el dominio del problema que el sistema está diseñado para modelar. Representa una entidad del mundo real como una persona o una cosa y captura los datos y el comportamietno

Al identificar las entidades que son relevantes para el sistema y modelarlas como clases, los diseñadores pueden crear un modelo más preciso y efectivo del sistema.

7. Definición de clase de Interfaz

<<interface>>

IPrintable

+ print(): void

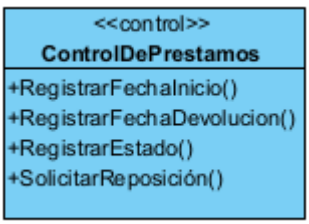
Describe la funcionalidad que la clase puede ofrecer, pero no especifica cómo se realiza esta funcionalidad

las clases de análisis de tipo interfaz no son clases concretas que se puedan instanciar en un programa, sino que se utilizan para modelar la interacción entre objetos y las funcionalidades que ofrecen en un nivel más abstracto.

8. Definición de clase de control

Estas clases no representan objetos concretos, sino que describen cómo se llevan a cabo las operaciones y cómo se controla el flujo de información y eventos

Estas clases suelen estar relacionadas con las clases de análisis de tipo entidad



9. Definición de colaboración entre clases

Ninguna clase está aislada.

Cada una trabaja en colaboración con las demás.

Para modelar una colaboración:

Hay que identificar los comportamientos del sistema que resultan de la interacción de un grupo de clases.

Para cada comportamiento hay que identificar las clases, interfaces y otras colaboraciones que intervienen.

Hay que emplear escenarios para recorrer la interacción entre los elementos.

Hay que repartir las responsabilidades entre las clases para después convertirlas en atributos y operaciones.

Las colaboraciones se modelan generalmente mediante diagramas de secuencia

10.Definición de diagrama de [máquina de] estados

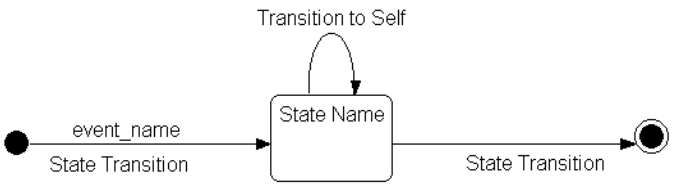
Muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro.

Son útiles sólo para los objetos con un comportamiento significativo.

Cada objeto sigue el comportamiento descrito en el Diagrama de Estado asociado a su clase.

Mientras que las interacciones modelan el comportamiento de una "sociedad de objetos", un diagrama de estado modela el comportamiento de un único objeto.

Cada objeto está en un estado en cierto instante



11.Elementos de un diagrama de [máquina de] estados

Evento:

Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Que puede ser:

Condición que toma el valor de verdadero o falso

Recepción de una señal de otro objeto en el modelo

Recepción de un mensaje

Paso de cierto período de tiempo, después de entrar al estado o de cierta hora y fecha particular

El nombre de un evento tiene alcance dentro del paquete en el cual está definido, no es local a la clase que lo nombre.

Estado:

Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto está esperando alguna operación

Se representa mediante un rectángulo con los bordes redondeados, que puede tener tres compartimientos:

uno para el nombre

otro para el valor característico de los atributos del objeto en ese estado,

y otro para las acciones que se realizan al entrar, salir o estar en un estado (entry, exit o do, respectivamente).

transacción simple

Es una relación entre dos estados que indica que un objeto en el primer estado puede entrar al segundo estado y ejecutar ciertas operaciones, cuando un evento ocurre y si ciertas condiciones son satisfechas.

event-signature es la descripción del evento que da lugar la transición,

guard-condition son las condiciones adicionales al evento necesarias para que la transición ocurra,

action-expression es un mensaje al objeto o a otro objeto que se ejecuta como resultado de la transición y el cambio de estado

send-clause son acciones adicionales que se ejecutan con el cambio de estado, por ejemplo, el envío de eventos a otros paquetes o clases.

transacción interna

Es una transición que permanece en el mismo estado, en vez de involucrar dos estados distintos. Representa un evento que no causa cambio de estado. Se denota como una cadena adicional en el compartimiento de acciones del estado.

Transacción temporizada

Las esperas son actividades que tienen asociada cierta duración.

La actividad de espera se interrumpe cuando el evento esperado tiene lugar.

Este evento desencadena una transición que permite salir del estado que alberga la actividad de espera

Sub estados

Un estado puede descomponerse en sub-estados, con transiciones entre ellos y conexiones al nivel superior.

Se representa como una línea sólida entre dos estados

12.Definición de Relación de agregación

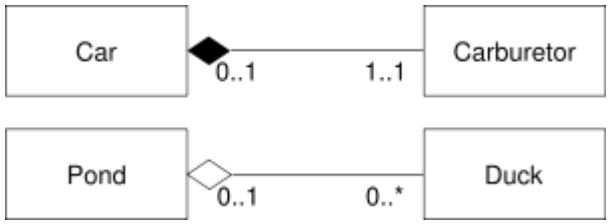
Representa una relación de "todo-parte" entre dos o más clases. En una agregación, una clase es el todo y las otras clases son las partes. La agregación se representa con un diamante vacío en el extremo de la clase del todo y una línea que conecta con la clase de la parte

Un ejemplo de agregación podría ser la relación entre una clase "Departamento" y una clase "Empleado", donde un departamento puede tener varios empleados, pero un empleado solo puede pertenecer a un departamento.

Otro ejemplo es que podemos tener un conjunto de objetos “árboles”, por ejemplo, que cuando se agregan entre sí pueden formar un bosque. Podemos tener un bosque sin árboles, y podemos tener un solo árbol. Los árboles también pueden pertenecer a otros bosques. Lo mismo es cierto en el caso de Ducks and Pond

Por lo tanto, NO completamos el diamante con agregación.

El Diamante siempre va del lado del objeto 'grupo'.

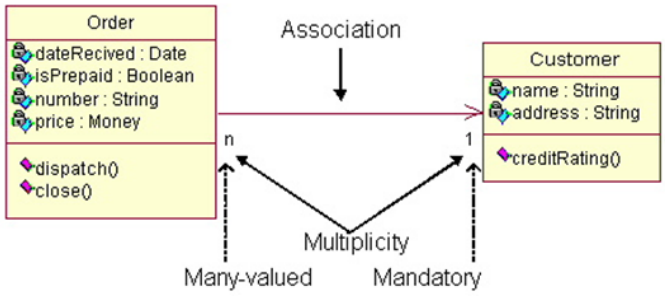


13.Definición de Relación de asociación

Representa una relación entre dos o más clases en la que una instancia de una clase está relacionada con una o más instancias de otra clase.

La relación de asociación es la relación más común en un diagrama de clases.

La asociación puede ser unidireccional o bidireccional y puede ser nominativa, en la que la relación entre las clases tiene un nombre, o anónima, en la que la relación no tiene un nombre explícito.



14.Definición de Relación de composición

Es similar a la agregación, pero con una relación más fuerte. En la composición, una clase es responsable de crear y administrar las instancias de otra clase.

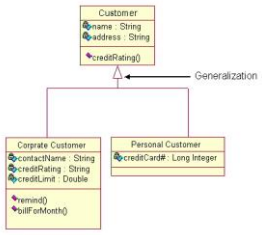
La composición es donde se tiene una cantidad de objetos que forman otro objeto: no puedes tener las piernas, el hígado y la lengua de un perro, etc., sin tener el objeto del perro. Los objetos están compuestos. Si eliminamos un objeto perro, perdemos todos los objetos que lo componen en una eliminación en cascada. El diamante "lleno" indica que esos objetos están contenidos dentro de ese objeto perro. Lo mismo es cierto para un objeto automóvil y el objeto motor/carburador

15.Definición de Relación de generalización

Representa una relación de especialización entre dos clases, donde una clase se deriva de otra clase existente.

Una generalización se usa cuando dos clases son similares, pero tienen algunas diferencias.

La clase derivada hereda los atributos y métodos de la clase base, pero también puede agregar o modificar estos elementos.



La herencia se representa con una flecha que apunta desde la clase derivada a la clase base.

16.Definición de Requerimientos no-funcionales

Son aquellos que no están directamente relacionados con los servicios específicos que el sistema entrega a sus usuarios

Pueden relacionarse con propiedades emergentes del sistema como la confiabilidad, los tiempos de respuesta. También pueden definir restricciones de implementación

No cumplir con un requisito no funcional puede significar que todo sea inutilizable

Surgen a través de las necesidades del usuario

17.Definición de Requerimientos no-funcionales externos

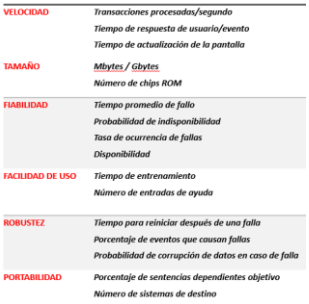
Se derivan de factores externos al sistema y su proceso de desarrollo.

Requerimientos legales: establecen lo que se debe de hacer para que un regulador como hacienda apruebe el uso de sistema ,deben asegurarse para que el sistema opere dentro de la ley

Requisitos éticos: aseguran que el sistema será aceptable para sus usuarios y el publico

18.Definición de Requerimientos no funcionales de producto

Estos requisitos especifican o restringen el comportamiento del software. Los ejemplos pueden ser que tan rápido debe ejecutarse el producto o cuanta memoria requiere



19.Definición de Requerimientos no funcionales organizacionales

Son requisitos generales del sistema derivados de políticas y procedimientos en la organización del cliente y el desarrollador.