Clase 6

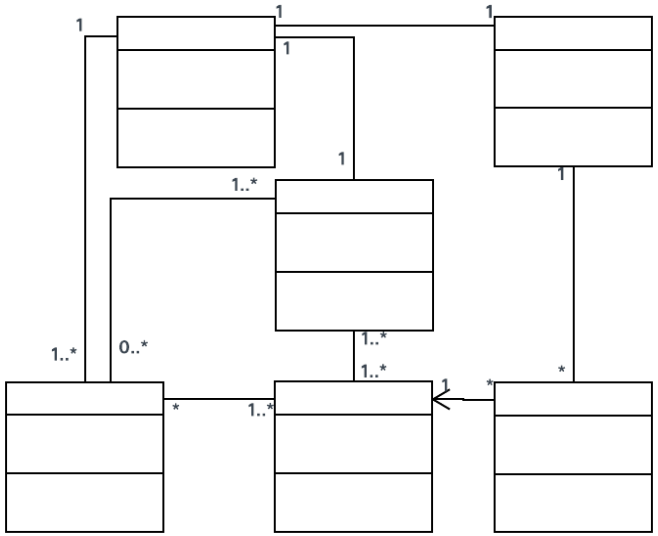
UML y Serialización

Agenda

- Avisos
- UML
- Diagramas de clases
- Herencia
- Serialización
- Des-Serialización

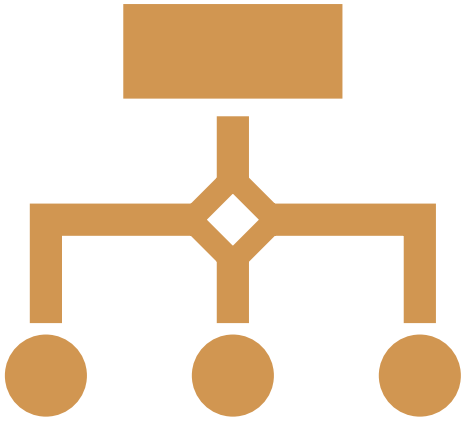
UML

- ¿Qué es?
- Tipos de diagramas
- Diagramas de clases
- Beneficios de diagramas de clases
- Componentes básicos
- Modificadores de acceso



¿Qué es?

El Lenguaje Unificado de Modelado (UML) fue creado para forjar un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento. UML tiene aplicaciones más allá del desarrollo de software, p. ej., en el flujo de procesos en la fabricación.



Tipos de Diagramas

UML usa elementos y los asocia de diferentes formas para formar diagramas que representan aspectos estáticos o estructurales de un sistema, y diagramas de comportamiento, que captan los aspectos dinámicos de un sistema.

Tipos de diagramas

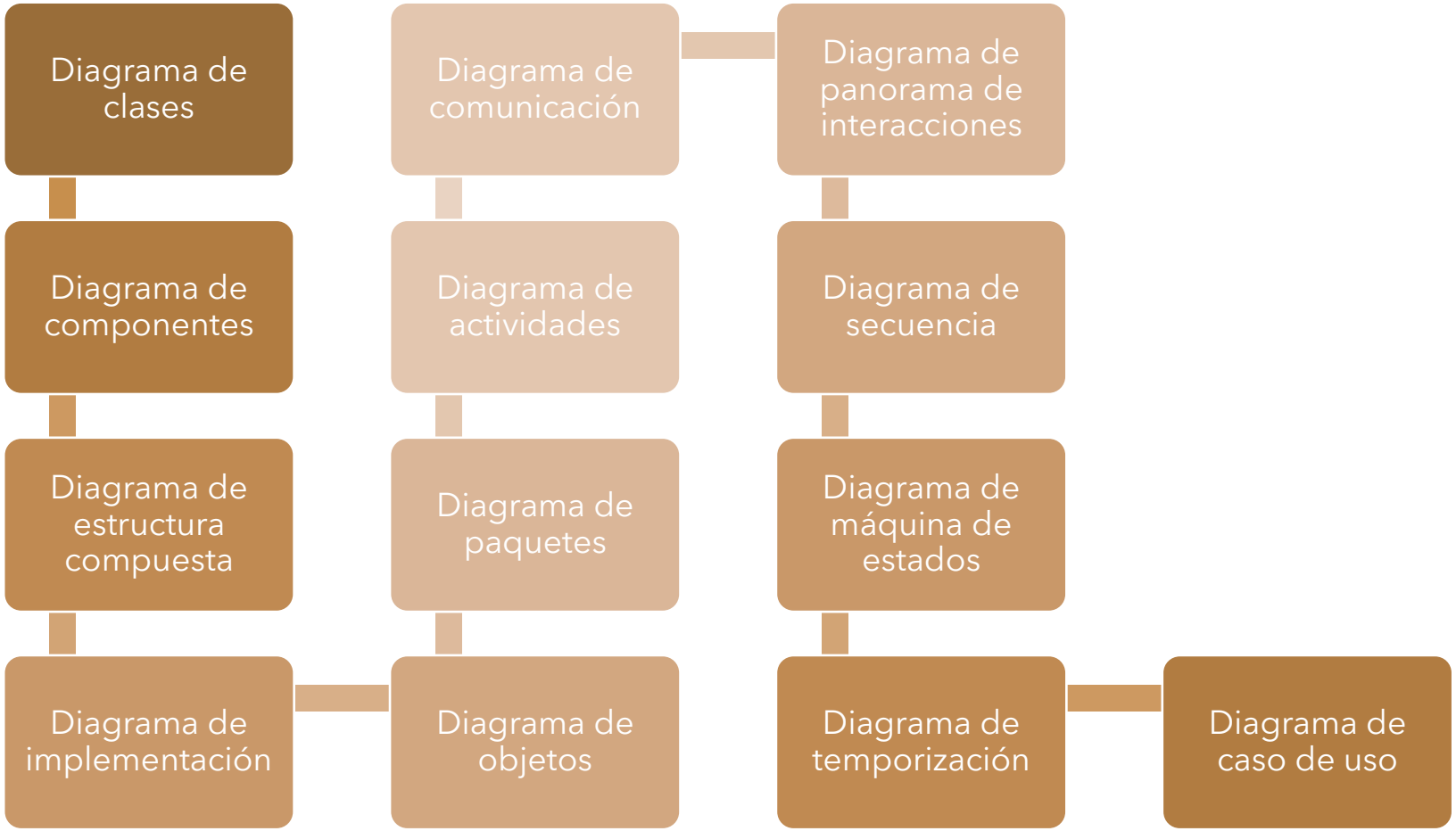


Diagrama de Clases

El diagrama UML más comúnmente usado, y la base principal de toda solución orientada a objetos. Las clases dentro de un sistema, atributos y operaciones, y la relación entre cada clase. Las clases se agrupan para crear diagramas de clases al crear diagramas de sistemas grandes.

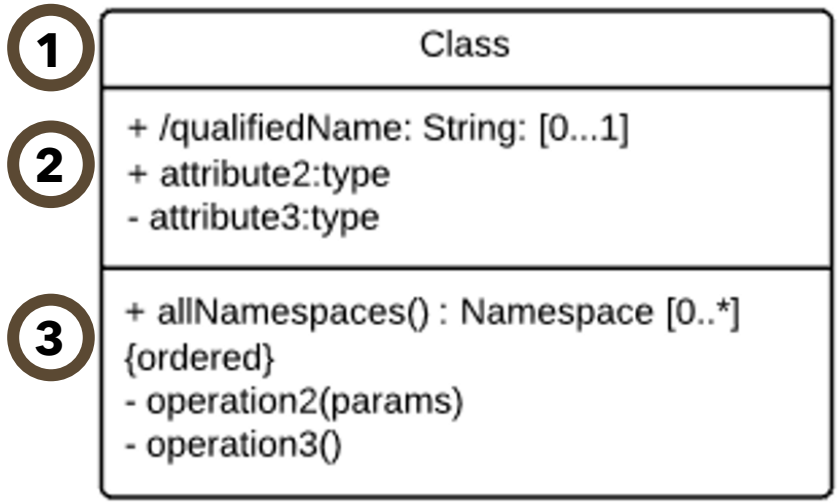
El UML se estableció como un modelo estandarizado para describir un enfoque de programación orientada a objetos (POO). Como las clases son los componentes básicos de los objetos, los diagramas de clases son los componentes básicos del UML. Los diversos componentes en un diagrama de clases pueden representar las clases que se programarán en realidad, los objetos principales o la interacción entre clases y objetos.

Beneficios del Diagrama de Clases

Los diagramas de clases ofrecen una serie de beneficios para toda organización. Usa los diagramas de clases UML para:

- Ilustrar modelos de datos para sistemas de información, sin importar qué tan simples o complejos sean.
- Comprender mejor la visión general de los esquemas de una aplicación.
- Expresar visualmente cualesquier necesidades específicas de un sistema y divulgar esa información en toda la empresa.
- Crear diagramas detallados que resalten cualquier código específico que será necesario programar e implementar en la estructura descrita.
- Ofrecer una descripción independiente de la implementación sobre los tipos empleados en un sistema que son posteriormente transferidos entre sus componentes.

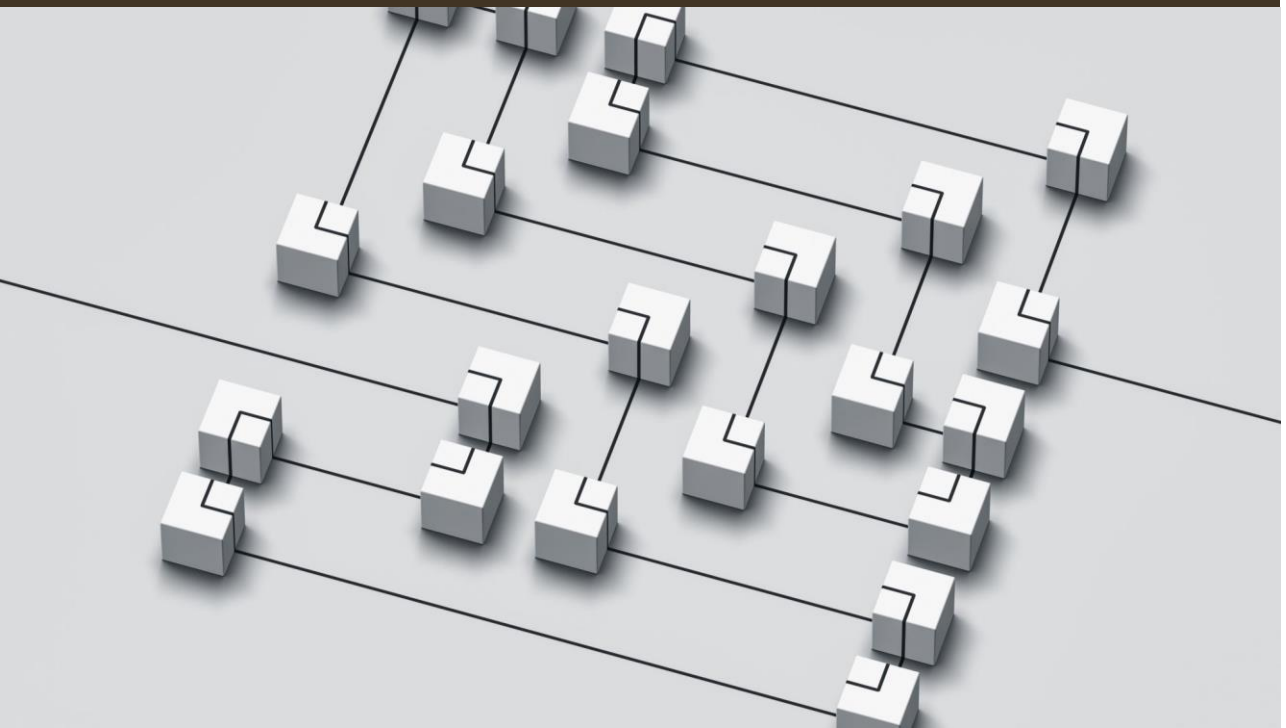
Componentes Básicos



El diagrama de clases estándar está compuesto por tres partes:

1. **Sección superior:** Contiene el nombre de la clase. Esta sección siempre es necesaria, ya sea que estés hablando del clasificador o de un objeto.
2. **Sección central:** Contiene los atributos de la clase. Usa esta sección para describir cualidades de la clase. Esto solo es necesario al describir una instancia específica de una clase.
3. **Sección inferior:** Incluye operaciones de clases (métodos). Esto está organizado en un formato de lista. Cada operación requiere su propia línea. Las operaciones describen cómo una clase puede interactuar con los datos.

Modificadores de Acceso

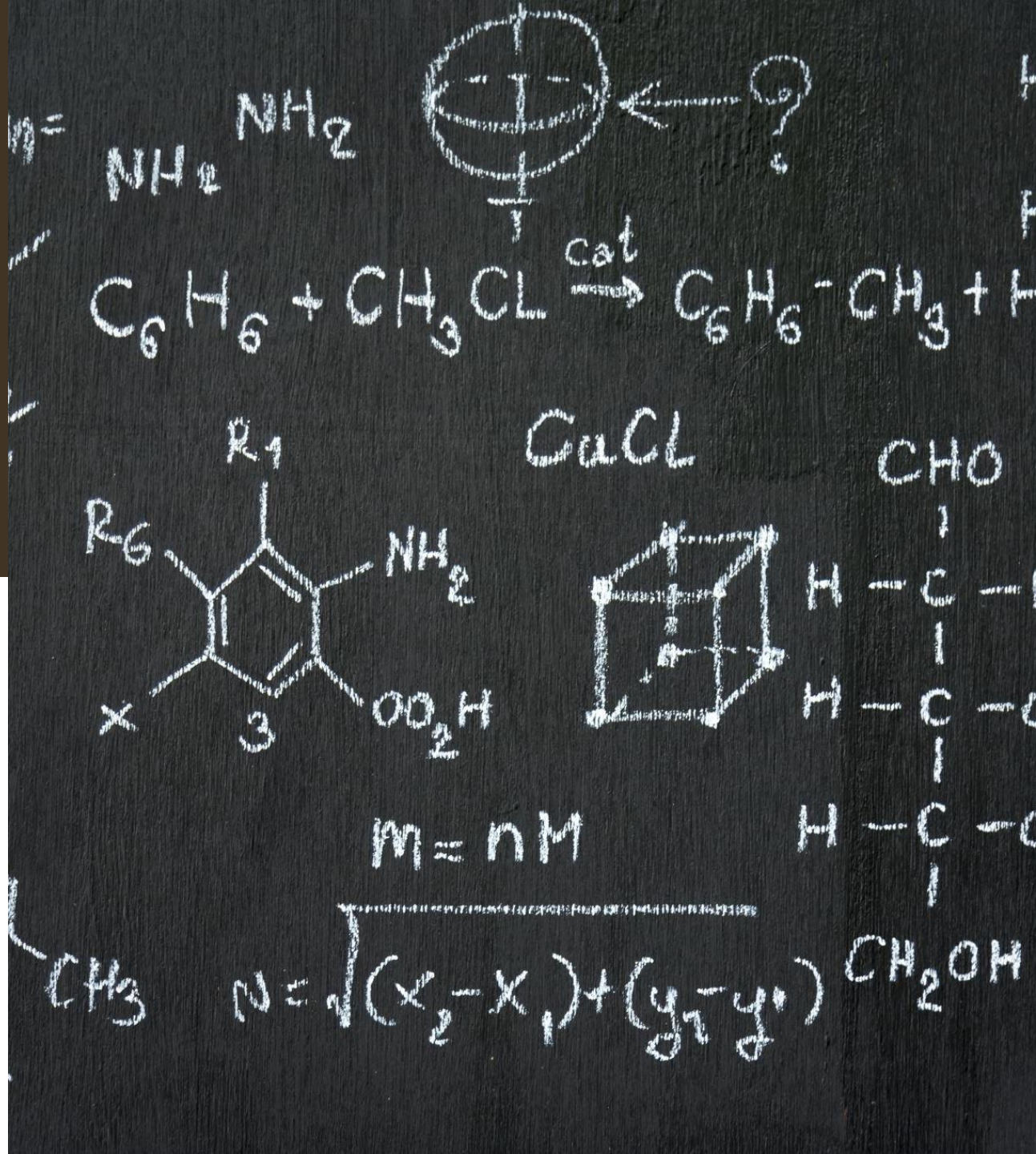


Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad). A continuación te mostramos los niveles de acceso con sus símbolos correspondientes:

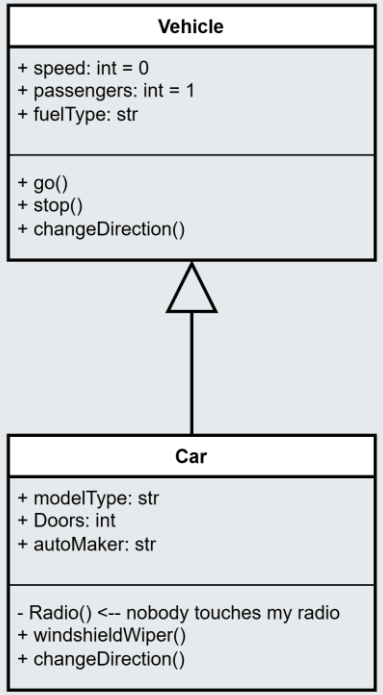
+	Público
-	Privado
#	Protegido
~	Paquete
/	Derivado
Subrayado	Estático

Interacciones

El término "interacciones" se refiere a múltiples relaciones y enlaces que pueden existir en diagramas de objetos y de clases. Algunas de las interacciones más comunes incluyen:



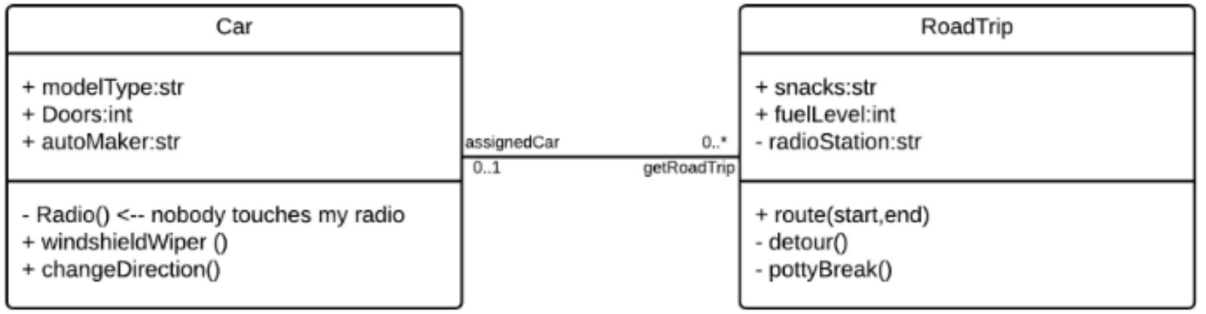
Herencia



El proceso en el que una subclase o clase derivada recibe la funcionalidad de una superclase o clase principal, también se conoce como "generalización". Se simboliza mediante una línea de conexión recta con una punta de flecha cerrada que señala a la superclase.

En este ejemplo, el objeto "Auto" heredaría todos los atributos (velocidad, números de pasajeros, combustible) y los métodos (arrancar(), frenar(), cambiarDirección()) de la clase principal ("Vehículo"), además de los atributos específicos (tipo de modelo, número de puertas, fabricante del auto) y métodos de su propia clase (Radio(), limpiaparabrisas(), aireacondicionado/calefacción()). La herencia se muestra en un diagrama de clases por medio de una línea continua con una flecha cerrada y vacía.

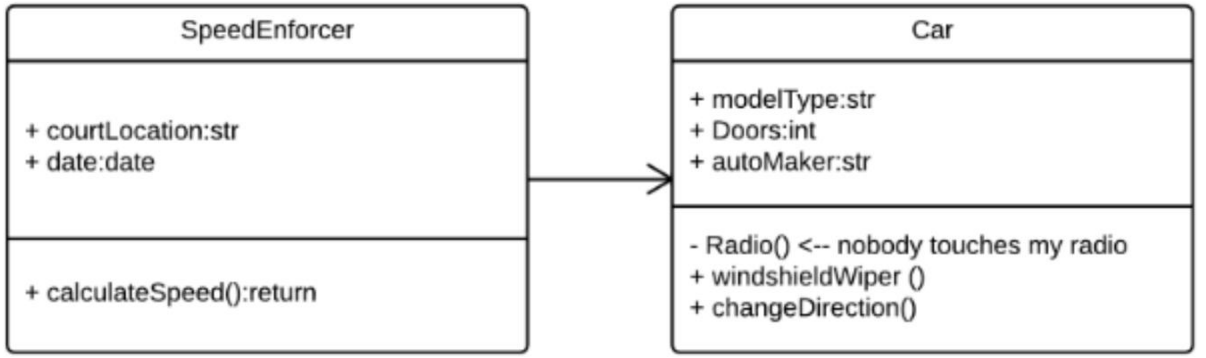
Asociación Bidireccional



La relación predeterminada entre dos clases. Ambas clases están conscientes una de la otra y de la relación que tienen entre sí. Esta asociación se representa mediante una línea recta entre dos clases.

En el ejemplo anterior, la clase Auto y la clase Viaje están interrelacionadas. En un extremo de la línea, el Auto recibe la asociación de "autoAsignado" con el valor de multiplicidad de 0..1, de modo que cuando la instancia de Viaje existe, puede tener una instancia de Auto asociada a ella o no tener instancias de Autos asociadas a ella. En este caso, una clase CasaRodante separada con un valor de multiplicidad de 0..* es necesaria para demostrar que un Viaje puede tener múltiples instancias de Autos asociadas a ella. Dado que una instancia de Auto podría tener múltiples asociaciones "iniciarViaje", en otras palabras, un auto podría realizar múltiples viajes, el valor de multiplicidad se establece en 0..*

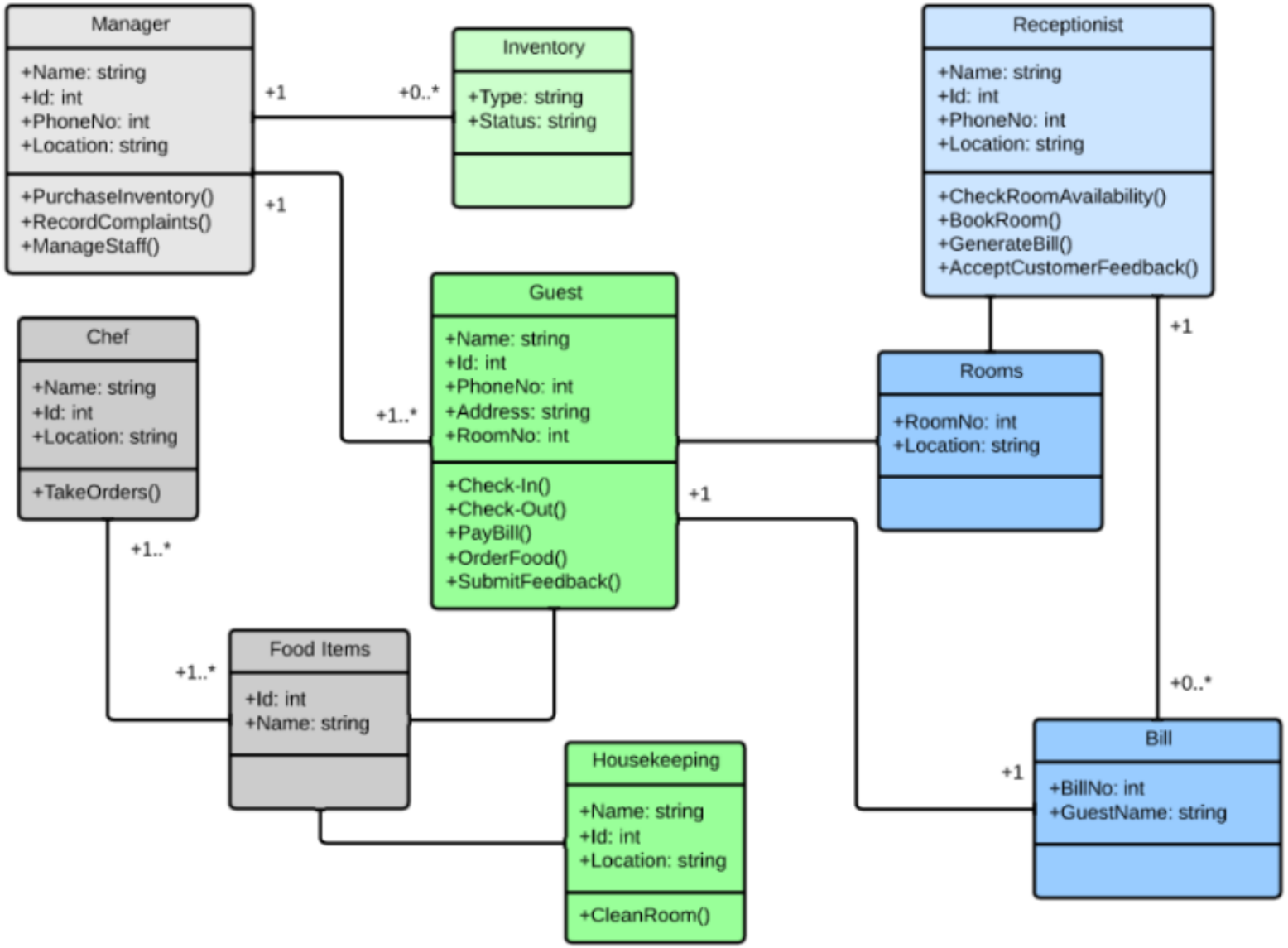
Asociación Unidireccional



Una relación un poco menos común entre dos clases. Una clase está consciente de la otra e interactúa con ella. La asociación unidireccional se dibuja con una línea de conexión recta que señala una punta de flecha abierta desde la clase "knowing" a la clase "known".

Como ejemplo, en tu viaje por Arizona, podrías encontrarte con una trampa de velocidad donde un radar de tráfico registra la velocidad a la que conducías, pero no lo sabrás hasta que recibas la notificación por correo. Esto no está dibujado en la imagen, pero en este caso, el valor de multiplicidad sería 0..* en función de cuántas veces hayas conducido frente al radar de tráfico.

Diagrama de Clases

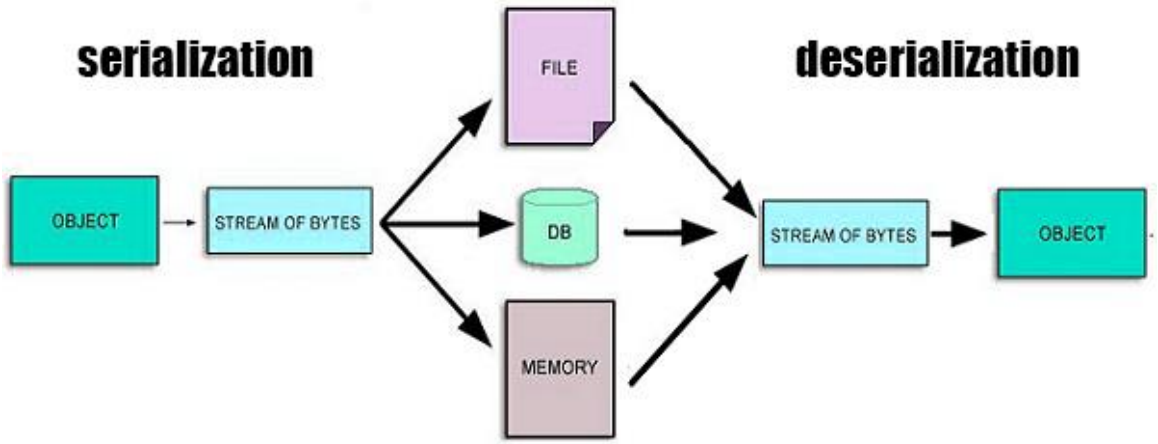


Video de Apoyo

https://www.youtube.com/watch?v=Z0yLerU0g-Q&feature=youtu.be

Ejemplo

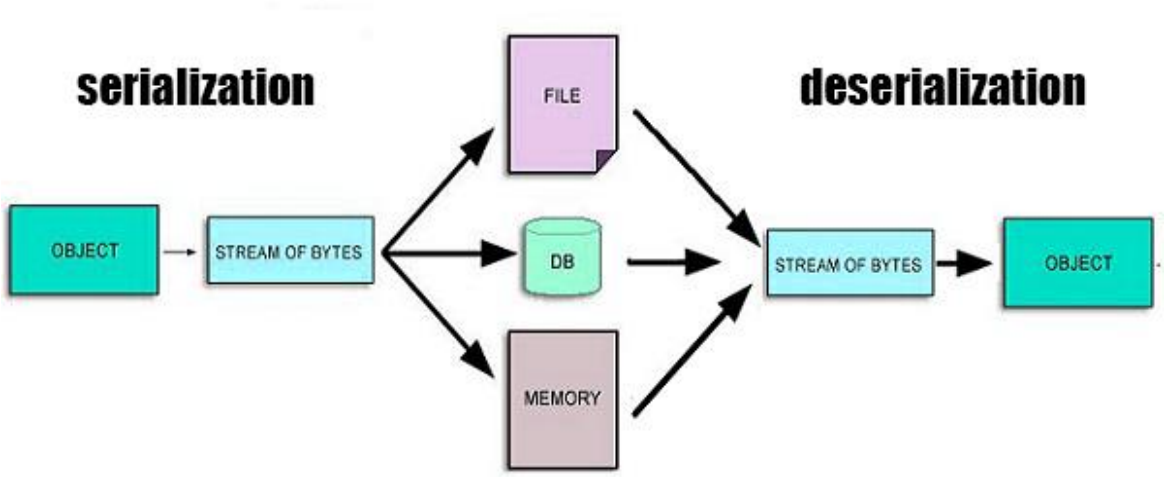
Serialización



La serialización es el proceso de convertir un objeto en una secuencia de bytes que puede ser almacenada, transmitida o guardada en un archivo. Para que un objeto sea serializable, su clase debe implementar la interfaz Serializable. Una vez que un objeto es serializado, puede ser guardado o transmitido utilizando clases como ObjectOutputStream.

Des-Serialización

Es el proceso inverso de la serialización. Convierte una secuencia de bytes (que representa un objeto serializado) de vuelta en un objeto. Esto se hace para reconstruir objetos previamente serializados, por ejemplo, cuando se lee un objeto de una archivo. Para des-serializar un objeto se utiliza la clase ObjectInputStream.



Ejemplo

Dudas o Comentarios