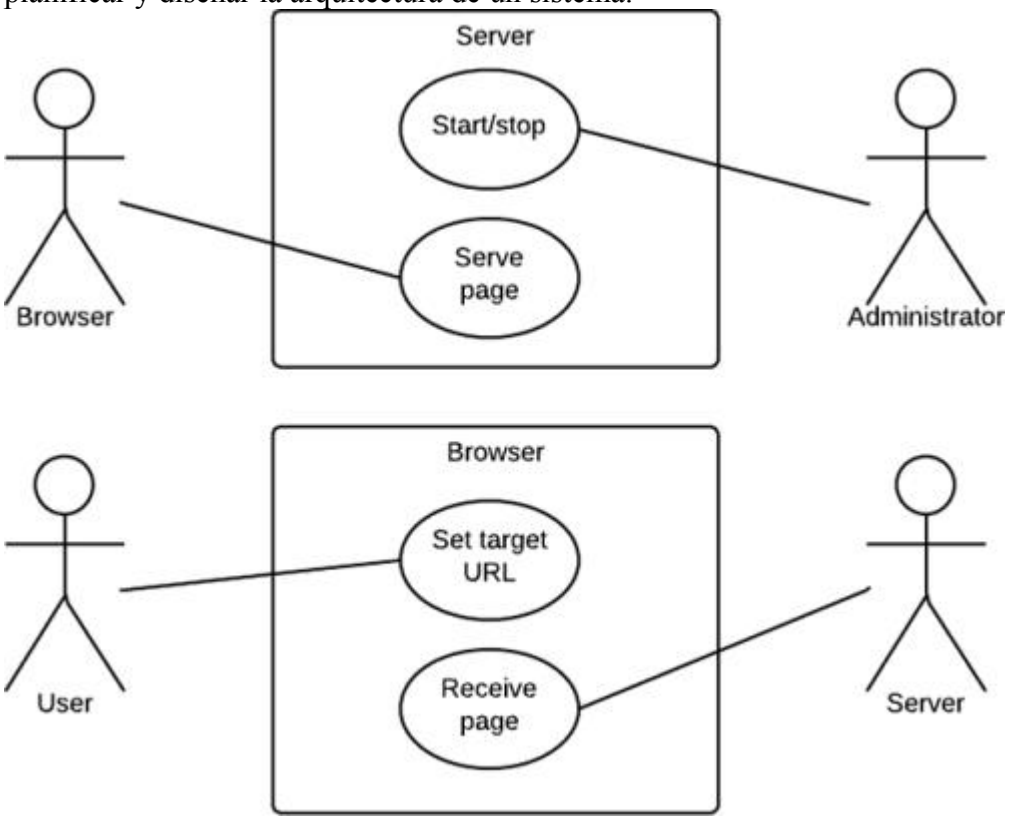
Axel David Arévalo Gómez
ISW-25
Optativa

Actividad 6

Desarrollo de la práctica

1. ¿Qué es el lenguaje UML?

El Lenguaje de Modelado Unificado (UML) es un lenguaje gráfico y estandarizado, utilizado principalmente en la ingeniería de software para visualizar y documentar un sistema. Sirve como una notación abstracta que ayuda a los desarrolladores a comprender la estructura y el comportamiento de un software, independientemente del lenguaje de programación que se vaya a usar. Su enfoque en la Programación Orientada a Objetos (POO) lo convierte en una herramienta fundamental para planificar y diseñar la arquitectura de un sistema.



2. ¿Para qué sirve?

UML sirve para analizar y modelar problemas de manera estática (como estructuras de clases) o dinámica (como interacciones y flujos). Permite abstraer el diseño de un sistema antes de pasar a la implementación en código, facilitando la comprensión del comportamiento del sistema, la estimación de tiempos de desarrollo y la aplicación de patrones de diseño o arquitecturas. En el contexto de diagramas de clases, ayuda a representar modelos de dominio, análisis y diseño refinados durante el proceso de desarrollo de software.

3. ¿En qué consiste un diagrama de clases?

Un diagrama de clases es el diagrama estático más importante de UML. Consiste en la representación de las clases que componen un sistema, incluyendo sus atributos y métodos, así como las relaciones entre ellas. Se construye de forma iterativa:

- Modelo de dominio: Se identifican las clases y relaciones básicas a partir de la descripción del problema.
- Modelo de análisis: Se añaden atributos y métodos preliminares.
- Modelo de diseño: Se detallan elementos como tipos de datos, visibilidad (pública, privada), parámetros y tipos de retorno, preparando el diagrama para la implementación.

Es una herramienta poderosa para modelar la estructura del sistema sin las complejidades del lenguaje de programación.

4. ¿Qué elementos se deben considerar para crear un diagrama de clases?

Para crear un diagrama de clases, se deben considerar los siguientes elementos clave:

- **Clases**: Representadas como rectángulos divididos en tres partes: nombre de la clase (arriba), atributos (medio) y métodos (abajo).
- **Atributos**: Información que almacena la clase (ej. nombre, edad), con visibilidad (pública [+], privada [-], protegida [#]), tipos de datos (ej. string, integer) y valores por defecto si aplica.
- **Métodos**: Comportamientos o operaciones (ej. constructores, getters, setters, métodos como agregar/eliminar elementos), incluyendo visibilidad, parámetros, tipos de retorno y si son abstractos o estáticos.
- **Relaciones**: Tipos de conexiones entre clases (ver pregunta 5), con navegabilidad (dirección de la flecha), multiplicidad (ej. 1, *, 1..n) y nombres descriptivos.
- **Otros detalles**: Instancias (objetos), modificadores de acceso (público, privado), clases abstractas o interfaces para herencia parcial, y responsabilidades (cada clase debe tener una sola razón para cambiar). Se puede empezar identificando sustantivos en el problema para definir clases, y refinar iterativamente para incluir patrones de diseño o capas arquitectónicas.

5. ¿Qué relaciones se representan en un diagrama de clases?

En un diagrama de clases se representan varios tipos de relaciones entre clases, cada una con su símbolo gráfico:

- **Asociación**: Conexión básica entre clases (línea sólida con flecha para navegabilidad). Indica que una clase conoce o usa a otra (ej. un estudiante registra materias). Puede ser unidireccional o bidireccional, con multiplicidad (ej. * para muchos) y nombres (ej. "registra").
- **Agregación**: Relación de "todo-parte" donde el todo contiene partes, pero las partes pueden existir independientemente (rombo vacío hacia el todo). Ej. un grupo contiene estudiantes; si el grupo desaparece, los estudiantes persisten. Incluye multiplicidad (ej. 1..*).

- **Composición**: Similar a la agregación, pero más fuerte: las partes no existen sin el todo (rombo lleno hacia el todo). Ej. una materia se compone de temas; si la materia desaparece, los temas también.
- **Generalización/Herencia**: Relación "es un tipo de" donde subclases heredan atributos y métodos de una superclase (flecha con triángulo vacío hacia la superclase). Ej. materias presenciales y virtuales heredan de materia. Promueve reutilización de código.
- **Interfaces o Herencia Parcial**: Subclases implementan métodos de una interfaz o clase abstracta (flecha punteada con triángulo). No hereda todo, solo especificaciones. Otras consideraciones incluyen clases asociativas (para atributos en relaciones) y dependencias (conocimiento temporal).