Consigna 2

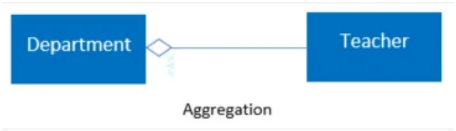
Motivación:

Esta actividad tiene la finalidad de definir varias clases con relaciones entre ellas, de modo de resolver un requerimiento sencillo e implementarla en lenguaje C++, considerando aspectos de comunicación entre objetos.

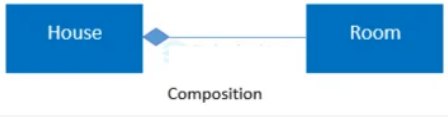
Asociación, agregación, composición, dependencia, herencia o realización, en Programación Orientada a Objetos, son formas de mostrar cómo se relacionan las clases.

La ***asociación*** es una conexión simple entre objetos sin propiedad alguna. Se refiere a la relación entre dos o más clases que interactúan entre sí. Define cómo los objetos de diferentes clases trabajan juntos, pero permanecen independientes entre sí. Como cuando varios alumnos pueden asociarse con un mismo profesor, y un mismo alumno puede asociarse con varios profesores. Sin embargo, los objetos no tienen propiedad y ambos tienen su propio ciclo de vida. Ambos pueden crearse y eliminarse de forma independiente.

La ***agregación*** significa que un objeto forma parte de otro, pero puede existir por sí solo. Se refiere a un tipo de relación donde una clase contiene una referencia a otra, pero ambas pueden existir independientemente. Muestra una relación de tipo "tiene un", con la característica que el objeto contenido no depende del objeto contenedor. Como cuando una escuela tiene profesores, pero estos pueden existir sin la escuela.

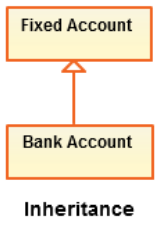


La ***composición*** es una relación fuerte (rígida) donde un objeto posee a otro, y si el objeto principal se elimina, la parte también. Es una forma de construir una clase utilizando objetos de otras clases, también muestra una una relación de "tiene un", pero con la intención de “está formado por”. Por ejemplo, una casa tiene habitaciones, cada habitación no se encuentra de manera independiente, ninguna habitación puede pertenecer a 2 casas. Y una característica básica, es que al destruir la casa se destruyen las habitaciones.

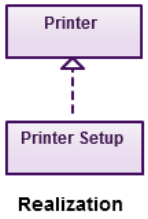


La ***dependencia*** significa que una clase necesita la ayuda de otra para funcionar correctamente, lo que muestra una relación de "uso". Por ejemplo, un cliente remoto necesita de un servicio para funcionar, por lo que la clase cliente usa temporalmente una clase de servicio. Es como tomar prestado algo de otra clase para completar una tarea.

La ***herencia*** es una relación de tipo jerárquica. Se refiere al mecanismo por el cual una clase, denominada hija o subclase), adquiere/hereda los atributos y comportamientos de otra clase, denominada madre o superclase. Este mecanismo permite aprovechar definiciones ya existentes. La herencia está ligada a principios de diseño conocidos como generalización y especialización. Es el caso de una cuenta bancaria para un comercio, que aprovecha la definición ya existente de una cuenta común. La herencia presenta diferentes variantes en función de la cantidad de clases afectadas y sus relaciones.



La ***realización*** es un tipo especial de relación donde una clase concreta implementa una interfaz o una clase abstracta, la cual sólo enuncia un cierto comportamiento. Se dice que la clase realiza la interfaz porque provee la implementación de los métodos definidos en ella. Es el caso de una clase de configuración de impresora que implementa métodos generales (vacíos) enunciados en la clase impresora.



Tanto la herencia como la realización son relaciones que, si bien se pueden estimar en las primeras fases del desarrollo, lo habitual es que se modelen/incluyan durante el diseño (en segundas iteraciones del proceso).

Requerimientos:

Se desea contar con una clase genérica para “transportar” mensajes, que pueda ser utilizada entre 2 nodos de una red de datos.

La particularidad es que, uno de los extremos requiere conocer un código de operación (resultado) y una colección de datos organizados, mientras que en el otro extremo se requiere además de lo anterior, un id de usuario y un id de dispositivo.

Se asume que la colección de datos es variable en su cantidad de elementos, en sus tipos de datos y en el nombre de los campos asociados a los mismos.

Los datos deben poderse guardar y recuperar.

En la solución propuesta se debe reutilizar código.

Si bien se debe realizar sólo una implementación, proponga 2 modelos diferentes.

Procedimiento general:

1) Diseñar en Umbrello 2 (dos) modelos OO usando UML, que muestre y defina la/s clase/s fundamentales. Realice las mismas consideraciones que las utilizadas en consignas previas.

2) Generar los módulos que correspondan, con el código base en lenguaje C++

3) Editar y completar la implementación de uno de los modelos, de modo de obtener una aplicación funcional que resuelva el requerimiento.

4) Considerar que el módulo/clase principal tiene la finalidad de mostrar el uso y comportamiento de las clases de mensajes propuestas. Es de interés, incorporar instrucciones que manejen diferentes cantidades y tipos de datos dentro del mensaje.

5) Realizar un informe similar a lo ya pautado.

6) Agregue al informe una comparativa con las ventajas y desventajas de cada modelo propuesto y una justificación (técnica, conceptual) acerca del modelo adoptado.

7) Preparar una carpeta para entrega, similar a lo ya pautado.

Recursos complementarios (disponibles en el aula virtual)**:**

- Apuntes de clase.
- https://www.scholarhat.com/tutorial/oops/understanding-association-aggregation-composition-and-dependency-relationship
- https://www.scholarhat.com/tutorial/oops/understanding-inheritance-and-different-types-of-inheritance
- https://www.scholarhat.com/tutorial/oops/difference-between-generalization-and-specialization