Trabajo Practico N°6

Pruebas Unitarias con JUnit

3.- Responda las siguientes preguntas:

a) ¿Por qué no se dibuja la relación de agregación entre la clase CollectionProducto y Producto?

b) ¿Cuántos atributos tiene la clase Factura? ¿Cuáles son los nombres de esos atributos?

c) ¿Cómo se llama la relación que se establece entre Factura y Detalle?

d) ¿Cómo se llama la relación entre las clases Factura y Cliente?

e) ¿Por qué los atributos de las clases Collections son públicos?

f) Describa las características de todos los métodos de la clase CollectionClientes.

DESARROLLO

a) Porque la relación de agregación sí está representada. El diagrama utiliza una versión simplificada para indicar la multiplicidad 1..* en el atributo "productos" de la clase CollectionProducto. Esto significa que CollectionProducto contiene una colección de objetos Producto (uno o más). Aunque no se utiliza el símbolo del diamante específicamente, la semántica de agregación está presente.

b) La clase Factura tiene dos atributos:

- fecha: de tipo LocalDate, que representa la fecha de la factura.
- nroFactura: de tipo long, que representa el número de la factura.

c) La relación entre Factura y Detalle es de **composición**. Esto significa que el ciclo de vida de los objetos Detalle está ligado al de la Factura. Si se elimina la Factura, también se eliminan los Detalles asociados. La multiplicidad 1..* indica que una factura puede tener uno o más detalles.

d) La relación entre Factura y Cliente no está directamente representada en el diagrama. Sin embargo, podemos inferir una relación indirecta a través de la clase CollectionFactura, que gestiona las facturas, y la relación entre Cliente y CollectionCliente. Se podría decir que existe una **asociación indirecta** entre Cliente y Factura.

e) Porque es importante destacar que el diagrama **no muestra explícitamente el modificador de acceso** (público, privado, protegido) de los atributos. Se asume que los atributos de las clases "Collection" (CollectionProducto, CollectionCliente, etc.) son privados o protegidos, siguiendo buenas prácticas de encapsulamiento. El acceso a estos atributos se realizaría mediante métodos públicos (no representados en el diagrama) como getters, setters o métodos específicos para manipular las colecciones.

f) La siguiente clase CollectionClientes tiene los métodos:

- **agregarCliente(in cliente: Cliente):** Permite agregar un nuevo objeto Cliente a la colección de clientes.
- **buscarCliente(in dni: long): Cliente:** Busca un cliente en la colección utilizando su DNI como criterio. Devuelve el objeto Cliente si lo encuentra, de lo contrario retorna un valor nulo o lanza una excepción.
- **precargarClientes():** Este método probablemente se encarga de cargar clientes iniciales desde alguna fuente de datos (como un archivo o una base de datos) al iniciar la aplicación.