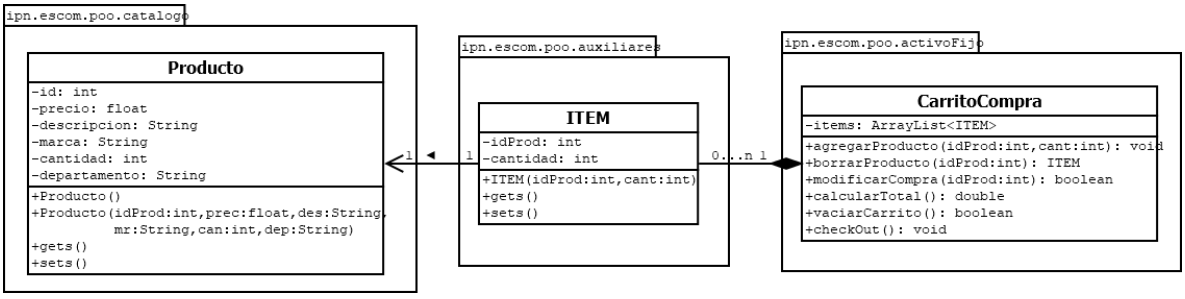
PRACTICA 4: DISEÑO DE APLICACIONES UTILIZANDO RELACIONES ENTRE CLASES, SOBRECARGA DE MÉTODOS Y CONSTRUCTORES Y PAQUETES.

Agregar los métodos y atributos que considere necesarios en cada clase para hacerlas más útiles, además de sus **gets** y **sets** correspondientes. Utilizar ampliamente sobrecarga de constructores y métodos.

Ejercicio 1: Tienda virtual

Utilizando el siguiente diagrama de clases, implemente una App llamada **TiendaVirtual** con las siguientes consideraciones:

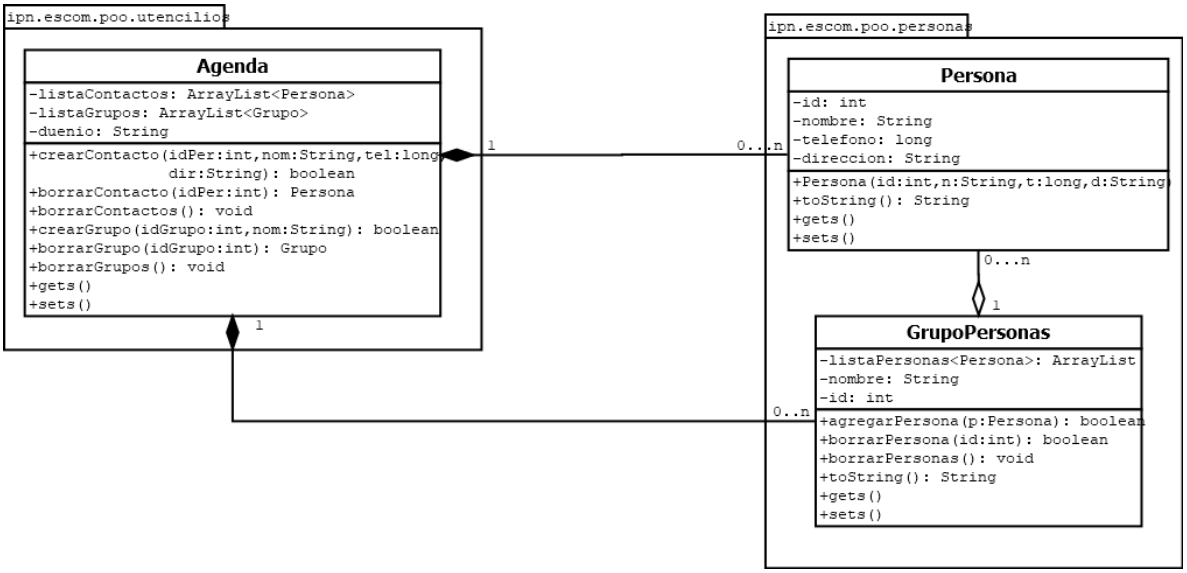


1. Crear dos arreglos, el primero de 10 objetos **Producto** y el segundo 5 objetos **Carrito**. Cada **Producto** contendrá un **id** y una **cantidad** que será el stock en almacén.
2. La relación entre las clases **Producto** y **Carrito** es muchos a muchos, para evitar esta relación, ya que puede causar muchos problemas, es recomendable utilizar una clase intermedia llamada **ITEM** que permita manipular los datos sin tanto problema al momento de realizar la operación **agregarProducto**.
    - La relación entre **ITEM** y **Producto** es de asociación y es uno a uno, ya que en el **ITEM** se debe colocar el id del **Producto** y la cantidad que se desea comprar.
    - La relación entre **ITEM** y **Carrito** es composición 0…muchos a uno, es decir, un carrito puede contener 0 o muchos **ITEM**, la cantidad de **ITEM** en el **Carrito** dependerá de la cantidad de **Productos** que se quieran comprar. **OJO** no se agregan **Productos** al **Carrito**, sino que cada que se quiera agregar un **Producto** al carrito se crea un **ITEM** que contiene el **id** del **Producto** y la **cantidad** que se desea comprar.
3. La operación **checkOut**, se debe realiza lo siguiente:
    - La cantidad de cada **Producto** se debe actualizar ya que el usuario puede decidir comprar o no el **Producto**.
    - Crear un ticket que describa los **Productos** que se compraron, la cantidad de cada uno de ellos, el precio unitario, el costo total por cada producto y el costo total de la compra.
    - Eliminar los **ITEM** del carrito para que el carrito pueda ser utilizado nuevamente.

4. **Borrar Producto:** Esta operación elimina el **`ITEM`** que describe al producto que se quiere eliminar del carrito
5. **Modificar compra:** Recibe el id del producto que se desea modificar la cantidad que se va a comprar.

Ejercicio 2: Agenda

Utilizando el siguiente diagrama de clases, implemente una App llamada **`Agenda`** con las siguientes consideraciones:



- ## Puntos a desarrollar en este ejercicio:

1. La clase **`Agenda`** debe estar compuesta por objetos **`Persona`** y objetos **`Grupo`**, para que se cumpla la relación de composición en las operaciones:
    - **`crearContacto`**: Se debe enviar los datos del contacto, crear el objeto **`Persona`** y agregarlo a **`listaContactos`** dentro de este método.
    - **`crearGrupo`**: Se debe enviar los datos del grupo, crear el objeto **`Grupo`** y agregarlo a **`listaGrupos`** dentro de este método.
2. A la clase **`Grupo`** se le agregan o envían las referencias de objetos **`Persona`**, para que se cumpla la relación de agregación en la operación:
    - **`agregarPersona`**: Se debe enviar la referencia del objeto **`Persona`**, esta referencia se toma de la lista de contactos y se agrega a **`listaPersonas`**, es decir, en **`listaPersonas`** se deben manejar las mismas referencias de la **`listaContactos`** de la agenda.

Operaciones que debe soportar la aplicación que utiliza la **Agenda**, menú principal:

1. **Crear contacto:** utiliza el método **crearContacto** de **Agenda**

2. **Elegir un contacto:** toma el **id** de un **Persona,** busca en la **listaContactos** ese **id** y despliega el siguiente submenú:
   - **Modificar sus datos:** modifica los datos del contacto utilizando los **gets** y **sets** de la clase **Persona**.
   - **Mostrar sus datos:** muestra los datos del contacto utilizando el método **toString** de la clase **Persona**.
   - **Afiliarlo a un grupo:** utiliza el método **agregarPersona** de la clase **Grupo** para agregar la referencia del objeto **Persona** a la **listaPersonas** de la clase **Grupo**.
   - **Desafiliar de un grupo:** toma el **id** del objeto **Persona** y utiliza el método **borrarPersona** e la clase **Grupo**
   - **Borrarlo de la agenda:** utiliza el método **borrarContacto** de **Agenda** y también lo borra de los grupos a los que pertenece.
3. **Eliminar contactos:** usa el método **borrarContactos** de la clase **Agenda**. En esta operación se borran los objetos **Persona** de la **Agenda** y de los grupos a los que pertenecen. Los objetos **Grupo** siguen existiendo pero después de esta operación estarán vacíos.

4. **Crear Grupo:** crea un grupo sin contactos y lo agrega a la lista de grupos de la agenda

5. **Elegir Grupo:** toma el **id** de un **Grupo,** busca en la **listaGrupos** ese **id** y despliega el siguiente submenú:
   - **Modificar datos del grupo:** modifica los datos del grupo utilizando los **gets** y **sets** de la clase **Grupo**.
   - **Mostrar los datos del grupo:** muestra los datos del grupo y los contactos que contiene utilizando el método **toString** de la clase **Grupo**.
   - **Afiliar contacto:** toma el **id** de una **Persona**, lo busca en **listaContactos** de la clase **Agenda** y lo afilia a este grupo.
   - **Desafiliar contacto:** toma el **id** de la **Persona** lo busca en la **listaPersonas** de la clase **Grupo** y lo elimina.
   - **Borrar grupo:** utiliza el método **borrarGrupo** de la clase **Agenda** para eliminar el actual grupo de la **listaGrupos**.
   - **Modificar contacto:** toma el **id** de la **Persona** lo busca en la **listaPersonas** de la clase **Grupo** y modifica sus datos utilizando los **gets** y **sets** de la clase **Persona**.

6. **Eliminar grupos:** usa el método **`borrarGrupos`** de la clase **`Agenda`**. En esta operación los contactos no se eliminan de la agenda, solo los grupos. Recuerden que la relación entre **`Grupo`** y **`Persona`** es de agregación.

Consideraciones a tomar en cuenta:

Código bien escrito y documentado	Buen uso de las relaciones entre las clases	Sobrecarga de métodos y constructores	Implementación completa	Pregunta
10 %	20%	20%	30%	20%

NOTA: No se revisaran prácticas con códigos sin documentar.

Requerimientos de entrega de la práctica:

- La práctica será desarrollada en equipos de 3 personas.
- La revisión de la implementación de la práctica:
- Grupo 2CV3 lunes 30 de abril.
- Grupo 2CM2 miércoles 02 de mayo.
- La entrega del reporte es el viernes 04 de mayo hasta las 23:00. Ver en la plataforma Moodle el formato de entrega.

Enviar un solo archivo comprimido que contenga el reporte de la práctica, los códigos fuente, los archivos jar. NO ANEXAR ARCHIVOS CON EXTENSIÓN .class