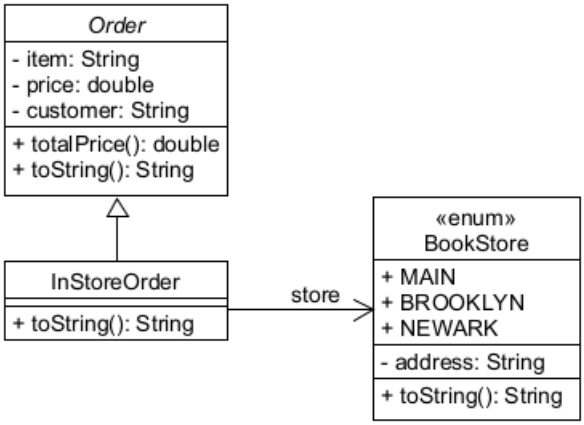
Análisis y Diseño de Software
Curso 2024-2025
Práctica 2
Diseño Orientado a Objetos

Inicio: A partir del 10 de febrero.
Duración: 2 semanas.
Entrega: En Moodle, una hora antes del comienzo de la siguiente práctica según grupos (semana del 24 de febrero)
Peso de la práctica: 15%

El objetivo de esta práctica es introducir los conceptos básicos de orientación a objetos, desde el punto de vista del diseño (usando UML), así como de su correspondencia en Java.

Apartado 1 (2 puntos):

Se quiere diseñar una aplicación para la gestión de pedidos de una cadena de librerías. El diseño inicial es el de la figura de la derecha.



Cada pedido (clase abstracta `Order`) tiene una descripción del elemento que se pide (`item`), el precio base (`price`), y el nombre del cliente que lo pide (`customer`). El método `totalPrice` calcula el precio final del pedido, y `toString` devuelve una representación del pedido en forma de cadena de texto.

La aplicación debe considerar dos tipos de pedidos: en la tienda (`InStoreOrder`) y online (que debes completar). Los pedidos en tienda guardan la tienda (`store`) en que se ha hecho el pedido (y que es también donde se recogerá). La cadena tiene tres sucursales que el diseño representa con los literales `MAIN`, `BROOKLYN` y `NEWARK` del enumerado `BookStore`. En Java, al igual que las clases, los enumerados pueden tener atributos, métodos y constructores. En particular, `BookStore` tiene un atributo `address` que guarda la dirección de la sucursal, y sobreescribe el método `toString` para devolver la dirección cuando se imprime el literal del enumerado.

Nota: Los diagramas de clases son abstracciones del código, así que normalmente y para facilitar su comprensión, no suelen incluir todos los detalles necesarios para la codificación. En concreto, se suelen omitir los métodos `setters, getters` y los constructores.

El siguiente listado proporciona una implementación para el diagrama de clases (cada clase va en un fichero .java que se llama como la clase). Como ya sabes, los constructores tienen el mismo nombre que la clase, y se invocan mediante el operador `new` para construir un objeto (crear una instancia de la clase) e inicializar sus atributos. Por ejemplo, el constructor de `Order` recibe tres parámetros e inicializa sus atributos. El constructor de la clase `InStoreOrder` llama al constructor de la superclase `Order` mediante `super(...)`, y ejecuta código adicional. La anotación opcional *`@Override`* indica que la intención del programador es sobrescribir un método de alguna superclase, y así el compilador puede comprobar que el método tiene igual firma que el método sobrescrito.

```
package orders;
public abstract class Order {
  private String item;
  private double basePrice;
  private String customer;

  public Order(String item, double price, String customer) {
      this.item = item;
      this.basePrice = price;
      this.customer = customer;
  }
  public double totalPrice() {
      return this.basePrice;
  }
  @Override
  public String toString() {
      return "Order of '"+this.item+"' for "+this.customer+" ("+this.totalPrice()+"$)";
  }
}
```

```java
package orders;
public class InStoreOrder extends Order {
  private BookStore store;

  public InStoreOrder(String item, double price, String customer, BookStore store){
      super(item, price, customer);
      this.store = store;
  }

  @Override
  public String toString() {
      return super.toString()+"\nStore: "+this.store;
  }
}
```

```java
package orders;
public enum BookStore {
  MAIN("Fifth Avenue. 73, Manhattan, NYC"),
  BROOKLYN("Bedford Avenue 24, Brooklyn, NYC"),
  NEWARK("Broad Street 11, Newark, NJ");

  private String address;

  private BookStore(String address) {
      this.address = address;
  }
  public String toString() {
      return this.address;
  }
}
```

Se pide: Extender el **diseño UML** y el **programa Java** para:

- Incluir una clase para los pedidos online, que guarde el email del cliente. Al mostrar un pedido de este tipo debe mostrarse también el email. El precio total de este tipo de pedidos tiene un recargo de 5$.
- El precio total de un pedido en tienda debe ser igual al precio base menos un descuento que depende de la sucursal donde se realizó el pedido: ninguno en MAIN, 1$ en BROOKLYN y 2$ en NEWARK.
- Debe poder hacerse un descuento (que será un porcentaje) sobre el precio base de cualquier pedido. Si se trata de descontar un porcentaje mayor que el 100.0% o menor que el 0.0%, el descuento debe igualarse a 100.0% o 0.0%, respectivamente.

Nota: Las clases del enunciado declaran un paquete `orders`, que es simplemente una carpeta en el sistema de ficheros donde debes incluir los ficheros *.java*. Más adelante aprenderás en detalle a usar paquetes, y su utilidad.

A modo de ejemplo, el siguiente programa crea tres pedidos (uno de ellos online), y aplica un descuento del 5% a los que tienen un precio total superior a 50 dólares.

```java
package orders;

public class Main {
  public static void main(String[] args) {
      Order[] orders = {
            new OnlineOrder("The Java bible", 50, "Peter Smith", "smith@gmail.com"),
            new InStoreOrder("Basic programming", 30, "Sue Rogers", BookStore.MAIN),
            new InStoreOrder("Functional Pearls", 50, "Donald Knuth", BookStore.NEWARK),
      };

      for (Order o: orders) {
        System.out.println(o+"\n----------------------");
        if (o.totalPrice()>=50) {
            o.setDiscount(5.0);
            System.out.println("With 5.0% discount on base price:\n"+
                               o+
                               "\n----------------------");
        }
      }
  }
}
```

Salida esperada:

```
Order of 'The Java bible' for Peter Smith (55.0$)
Email: smith@gmail.com
--------------------
With 5.0% discount on base price:
Order of 'The Java bible' for Peter Smith (52.5$)
Email: smith@gmail.com
--------------------
Order of 'Basic programming' for Sue Rogers (30.0$)
Store: Fifth Avenue. 73, Manhattan, NYC
--------------------
Order of 'Functional Pearls' for Donald Knuth (48.0$)
Store: Broad Street 11, Newark, NJ
--------------------
```

Apartado 2 (3 puntos)

Se quiere diseñar una plataforma de música en *streaming*. La plataforma debe organizar la música en géneros de manera jerárquica (p.ej., un género puede ser *música clásica*, con subgéneros como *barroco* o *romanticismo*, y éstos pueden tener otros subgéneros). La plataforma contiene canciones, que se pueden agrupar en álbumes (y una canción puede pertenecer a cero o más álbumes), así como *playlists*, que pueden contener canciones, álbumes completos, y otras playlists. Los géneros de una playlist vienen dados por el género de cada uno de los elementos que la componen. Los álbumes y las canciones tienen una fecha de creación, y las canciones tienen una duración en segundos. La duración de álbumes y playlists se calcula como la suma de la duración de los elementos que lo componen. Los intérpretes y los autores de álbumes y canciones son bien músicos individuales o grupos musicales. Un grupo musical está formado por músicos individuales, y la aplicación debe guardar su rol en el grupo (p.ej., bajista), y el periodo (fecha inicio y fin) en el que ejerció ese rol en el grupo.

Los usuarios de la plataforma se identifican con un nombre de usuario y una clave, y tendrán acceso a toda la música contenida en la plataforma. Los usuarios pueden marcar canciones, álbumes y playlists como favoritas ("*likes*"), y crear playlists, que podrán ser públicas o privadas. La aplicación deberá guardar las reproducciones de canciones, álbumes o playlists de cada usuario, junto con la fecha y hora de cada reproducción.

Se pide:

a) Realiza un diagrama de clases UML que refleje el diseño de la aplicación. No incluyas constructores, *getters* o *setters*, a no ser que se pidan en el próximo apartado **(2 puntos)**.

b) Añade en las clases los métodos y atributos necesarios para realizar las siguientes funcionalidades **(0.6 puntos)**:

1. Obtener el top 10 de las canciones y el top 10 de los álbumes más escuchados entre dos fechas en toda la plataforma.
2. Obtener la duración de una canción, álbum o playlist.
3. Obtener los géneros de una playlist.
4. Obtener el número de "likes" de una canción, álbum o playlist; y los usuarios que la han marcado como favorita.

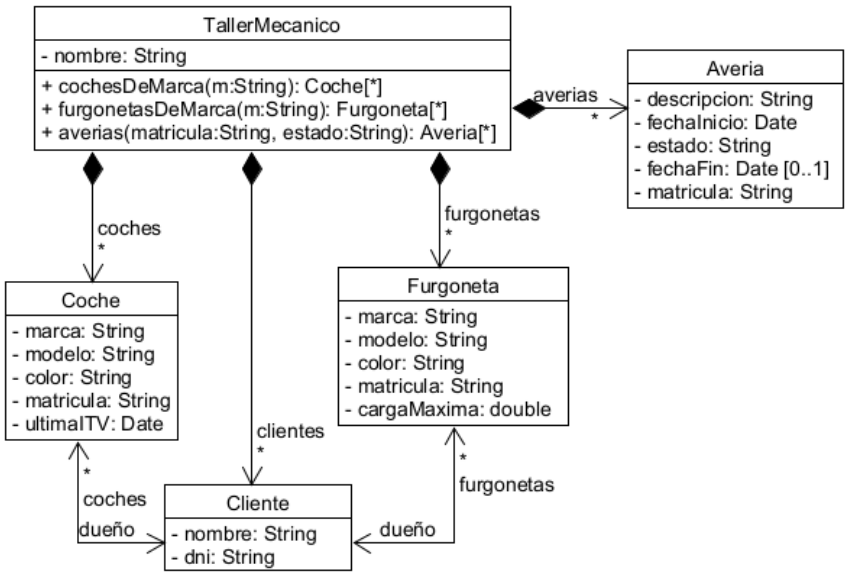
Ten en cuenta que en algún caso puede ser necesario añadir métodos auxiliares. No es necesario que incluyas el código del método.

c) Proporciona pseudocódigo (o código Java) de el/los métodos que calcula/n la duración de una canción, álbum o playlist **(0.4 puntos)**.

Apartado 3 (2 puntos):

El siguiente diagrama muestra un diseño (de mala calidad) de una aplicación para la gestión de un taller mecánico que repara coches y furgonetas. El sistema almacena también clientes, dueños de los vehículos, y las averías de estos. Actualmente, cada avería referencia al vehículo correspondiente por su matrícula. Cada avería tiene un estado, actualmente representado mediante una cadena de texto, y que puede tener un valor entre: `EN_ESPERA`, `REVISADO`, `EN_PROCESO`, `FINALIZADO` o `ENTREGADO`.

La aplicación necesita obtener de manera *eficiente* todos los vehículos de una determinada marca, funcionalidad que implementan los métodos `cochesDeMarca` y `furgonetasDeMarca`. También es necesario obtener todas las averías de un vehículo que estén en determinado estado, para lo que actualmente se usa el método `averias`.



Se pide:
Explica qué problemas tiene el diseño proporcionado y mejora el diseño para que refleje buenas prácticas de orientación a objetos. Debes crear un nuevo diagrama con el diseño mejorado y explicar cada uno de los cambios que has hecho, indicando qué problema resuelve.

Apartado 4 (3 puntos):

Se quiere diseñar una plataforma de comercio electrónico. Ésta ofrecerá productos organizados en categorías, permitirá a los clientes realizar pedidos sobre los mismos, y gestionará su entrega, siempre que la dirección de entrega se encuentre en un código postal soportado por la plataforma.

Cada producto tendrá un nombre, descripción, precio y número de unidades en stock. Los clientes (identificados mediante nombre y contraseña) podrán realizar pedidos, cada uno con entrega a una determinada dirección, con un código postal (que debe estar entre los soportados por la aplicación). Cada pedido especificará las unidades de los productos deseados. Los productos podrán tener descuentos de tres tipos: de una cantidad fija de euros, de un porcentaje del precio, y descuentos por la compra de múltiples productos del mismo tipo (p.ej., comprando 3 se pagan 2). La aplicación debe guardar el estado por el que pasan los pedidos, y la fecha/hora de dichos estados. En particular, un pedido puede estar en elaboración, confirmado, pagado, enviado y entregado.

La aplicación debe soportar dos tipos más de usuarios: operarios y gestores. Ambos se identifican con nombre y contraseña. Cada operario tiene asignado una serie de códigos postales, y se encargará de gestionar (marcar como enviados y entregados) los pedidos asignados por el gestor. Los gestores dan de alta y baja códigos postales soportados, asignan la gestión de pedidos a operarios, y la aplicación debe comprobar que sólo es posible asignar un pedido si su código postal está entre los controlados por el operario.

La aplicación debe permitir obtener el precio de los pedidos, evitar que se pidan más productos de los que se tienen en stock, pagar los pedidos, y permitir que estos sigan su flujo, hasta llegar a entregarse.

Se pide:

a) El diseño de la plataforma usando diagramas de clases UML. Añade los métodos necesarios en las clases para obtener la funcionalidad que menciona el enunciado. (**2.25 puntos**)

b) Crea un diagrama de objetos (**0.75 puntos**) que modele una plataforma con las siguientes características:

- Tenga un usuario que ha realizado un pedido de dos productos distintos, que está siendo gestionado por un operario.
- Uno de los productos debe tener un descuento.
- El pedido debe haberse enviado.

Normas de Entrega:

- Crea un directorio por cada apartado.
- La entrega la realizará uno de los estudiantes de la pareja, a través de Moodle.
- Si el ejercicio pide código Java, además del código, se debe entregar la documentación generada con *javadoc*. Si el ejercicio pide un diagrama de diseño, se debe entregar en PDF junto con una breve explicación (dos o tres párrafos a lo sumo) del mismo.
- Se debe entregar un único fichero ZIP / RAR con todo lo solicitado, que deberá llamarse de la siguiente manera: `GR<numero_grupo>_<nombre_estudiantes>.zip`. Por ejemplo, Roberto y Carlos, del grupo 2261, entregarían el fichero: `GR2261_RobertoCarlos.zip`.