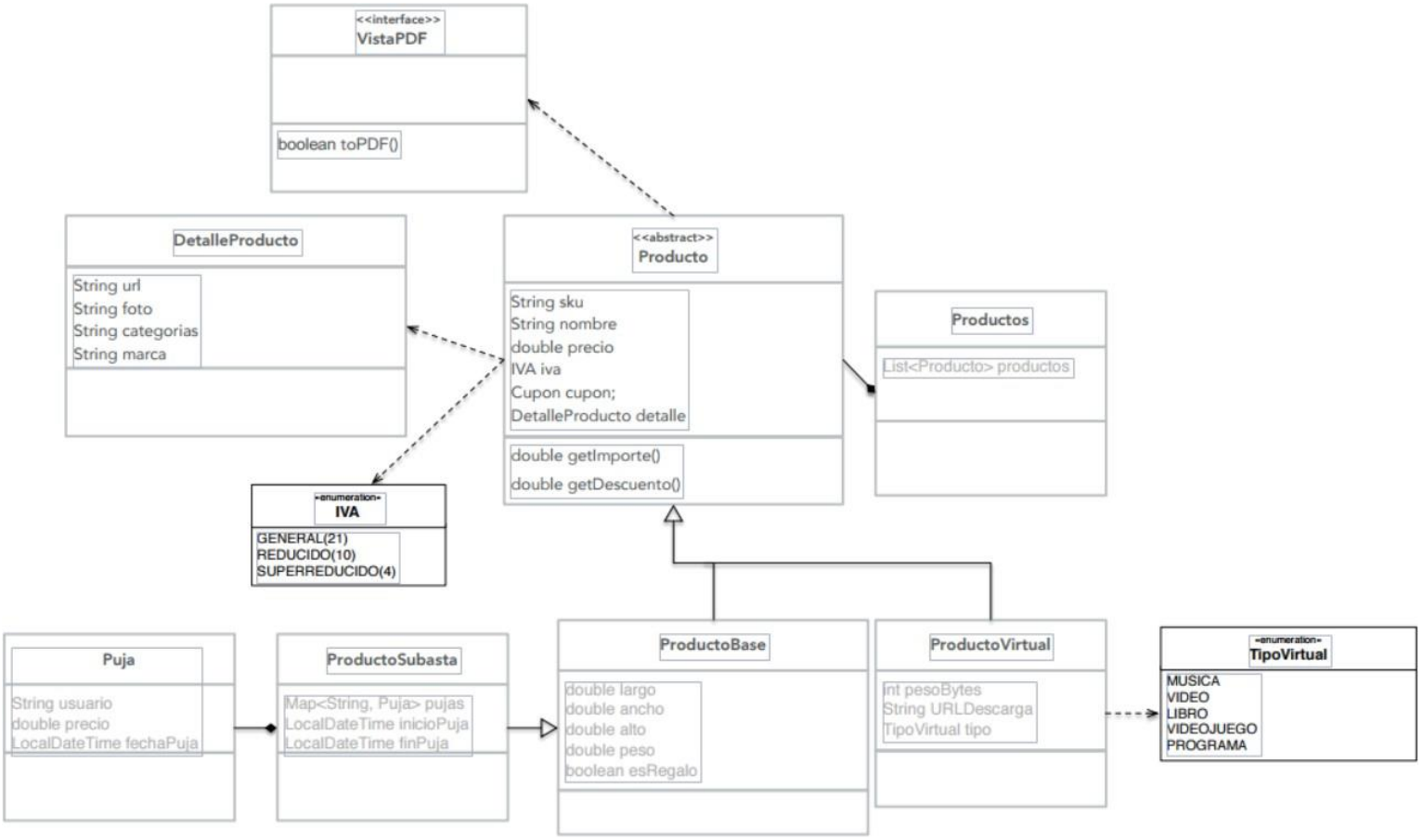
TIENDA

Una empresa tiene una web dedicada a la venta de productos por Internet. Dicha empresa desea realizar una aplicación para comercializar sus productos y nos ha encargado únicamente la gestión de productos.



PRODUCTOS

Los productos se van a modelar mediante tres clases: la principal que es Producto, y luego para el iva de un producto una clase IVA, y para el detalle de un producto DetalleProducto.

Todos los productos soportan **IVA**. Para almacenar el IVA se tiene:

public enum IVA{GENERAL(21), REDUCIDO(10), SUPERREDUCIDO(4)}

Se tiene una clase **Producto** que recoge información más relevante de un producto. Un producto se caracteriza por:

1. sku: es un código alfanumérico que identifica un producto. (Puedes utilizar **UUID**).
2. precio base: un número real.
3. iva: De tipo ***IVA*** (ver enum anterior)
4. detalleProducto de tipo ***DetalleProducto***

El orden natural de un producto es por sku, para ordenar, vamos.

También disponemos de una clase llamada **DetalleProducto** que amplía la información de un Producto y tiene los siguientes campos:

1. url de tipo cadena
2. foto de tipo cadena
3. categorías de tipo cadena
4. marca de tipo cadena
5. descripción de tipo cadena

La clase Producto implementa la interfaz **VistaPDF**, con el método toPDF(). Este método únicamente creará un fichero cuyo nombre será su sku y extensión .txt (<sku>.txt).

La empresa comercializa los siguientes productos en su web:

- **Producto base**: son productos normales que se caracterizan por que tienen dimensiones (largo, ancho y alto), peso y un boolean esRegalo. Esta información es interesante para el cálculo de los costes de transporte. El atributo esRegalo indica si se incluyen los gastos para envolver el regalo. Tienen un coste fijo de 2€. ProductoBase **hereda** de la Clase Producto.

- **Producto virtual**: Se trata de un producto descargable como una canción, un libro digital, una película, un videojuego, fotos, etc… Obviamente este tipo de productos no tiene dimensiones. Este tipo de objetos se caracterizan por:
    1. public enum TipoVirtual {MUSICA, VIDEO, IMAGEN, VIDEOJUEGO, PROGRAMA}
    2. long pesoBytes
    3. String URLDescarga
    4. TipoVirtual tipo

- **Productos subasta**: este producto se vende por los clientes en modo subasta al resto de clientes de la siguiente forma. Se fija un precio inicial (el valor de precio de la clase Producto), una fecha inicial y otra de finalización. Cada vez que un usuario realiza una Puja se comprueba que el precio de su puja es igual o mayor al precio inicial del producto o a la puja máxima realizada y que la fecha de finalización de la puja (finPuja) no es superior a la fecha y hora actual. Entonces se recoge el id de cliente (email), la cantidad pujada(puja) y la fecha y hora actual (fechaPuja). Cuando pasa la fecha de la puja, se cierra y el usuario que haya pujado más (última puja) se lleva el producto. Se desea almacenar las pujas (**colección** de Puja) realizadas para ello tenemos:
    - Clase **Puja** con los atributos: cliente de tipo cadena, puja de tipo double y fechaPuja de tipo LocalDateTime

Por último, para almacenar todos los productos disponemos de la clase **Productos**. Los atributos de esta clase son:

productos: colección de Producto

ACTIVIDADES A REALIZAR:

1. [3 puntos] Crea todas las clases, tipos enumerados, e interfaces del Diagrama de clases, con sus respectivos atributos, métodos getters y setters, constructores, toString, equals y la interfaz Comparable de acuerdo a la documentación aportada en este documento.

2. Implementa los siguientes métodos en la clase indicada:
    a. Producto [0.5 puntos]
        i. public boolean **toPDF**(): deberá generar un fichero llamado <sku>.txt que contenga toda la información de Producto (incluido DetalleProducto)
        ii. public abstract double **getImporte**(). Método *abstracto*
    b. ProductoBase [0.5 puntos]
        i. public boolean **toPDF**(): sobreescribe ese método de la clase padre añadiendo la información de largo, alto, ancho, peso y si es regalo.
        ii. public double **gastoTransporte**(double costeFlete, double pesoFlete): Calcula el coste de transporte de un producto. Para ello multiplicará el peso de un producto por (5€ si el peso es menor de 10kg, por 10€ si el peso está entre 10 y 50kg, por 20€ si el peso es mayor a 50kg).
        iii. public double **getImporte**(): Calcula el importe de un producto teniendo en cuenta el precio base, los impuestos, si es regalo y los gastos de transporte. Para ello seguiremos la siguiente formula: importe = (precioBase + transporte) + calculoIVA()
    c. ProductoSubasta [1.5 puntos]
        i. public boolean **toPDF**(): que además de lo anterior aparezcan el precio inicial de la puja y el precio final de la puja adjudicada.
        ii. public double **getImporte**(): idéntico al padre
        iii. Desde el atributo pujas (Map) se debe permitir: **añadir, modificar, borrar y consultar** pujas.
        iv. public Puja **buscarPuja**(Predicate<Puja> criterio): Permite filtrar pujas.
        v. public Puja **getPujaMaxima**(): Obtiene la puja más alta
        vi. public boolean **pujar**(Puja puja): Puja por un producto, o sea, *añade* esa puja.
    d. ProductoVirtual [0.5 puntos]
        i. public boolean **toPDF**(): sobrescribe el método padre de modo que salga también la información de peso en bytes, url, tipo.
        ii. public double **getImporte**(): no tiene gastos de envío, pero sí IVA.
    e. Productos [2.5 puntos]
        i. public void **abrirFromCSV**(): carga los dos ficheros .csv de la prueba en la lista de productos, cada uno creando un objeto de ProductoBase o ProductoVirtual antes de meterlo en la lista. Coloca los ficheros en tu proyecto.

ii. public void **buscar**(Predicate<Producto> criterio): Permite filtrar productos. Como se le pasa un Predicate, buscará sobre la lista de productos pero pasando a Stream.
iii. public Producto **getProducto**(String sku): Obtiene un producto por su sku
iv. public List<Producto> **getProductosSubasta**(): Obtiene una lista de solo los productos subasta
v. public List<Producto> **getProductosPrecio**(): obtiene una lista de todos los productos ordenados por precio de menor a mayor.
vi. public HashMap<String, List<Producto>> **getProductosPorCategoria**() devuelve para cada categoría los productos que hay en ella ordenados por sku descendente.
vii. public HashMap<String, Long> **getProductosTotalPorCategoria**() devuelve para cada categoría el número de productos que hay en ella.

3. [1.5 punto] Fichero **App.java** con un **main** que haga:

Productos catalogo = new Productos();
catalogo.abrirFromCSV();

//Acciones a realizar

1. Muestra todos los productos cuyo precio sea superior a 3000€
2. Muestra las categorías con el número de productos que hay en cada una.
4. Muestra los productos ordenados por precio
5. Obtén los 3 productos más baratos
6. Crea y añade un producto Subasta al catalogo
7. Realiza 3 pujas para un producto

ENTREGA: debes entregar el proyecto completo, incluidas las librerías, generado el Javadoc, generado un fichero .jar con el main indicado y, si haces la parte de bases de datos, pues el fichero .sql de la base de datos.