T3 PROGRAMACIÓN

Juan Marcial Portilla Zambrano
1º DAM

2022 - 06 - 12

Índice

CUESTIÓN 1. POO en Java y colecciones

- **Diseña un ejemplo de clases en donde puedas demostrar la herencia en Java. Explica si es posible realizar herencia múltiple en Java y por qué.**

Primero de todo empezaremos con recordar que es la herencia en POO:

La herencia es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente. La herencia permite compartir automáticamente métodos y datos entre clases, subclases y objetos.

Con un ejemplo muy sencillo :

```java
//Clase de la que se va a heredar
public class Padre {
    String nombre;
    String Apellidos;

    public Padre(String nombre, String Apellidos) {
        this.nombre = nombre;
        this.Apellidos = Apellidos;
    }

    public void saludar(){
        System.out.println("Soy Padre");
    }
}
```

```java
//clase que hereda
public class Hijo extends Padre{ // se usa exteds(oalabra reservada) para especificar de quien se hereda

    int edad; //atributo propio de la clase hijo (no la tiene padre)

    public Hijo(int edad, String nombre, String Apellidos) { // si no tiene constructor da error
        super(nombre, Apellidos); //atributos heredados de Padre
        this.edad = edad;
    }
```

Y en el main hacemos unas comprobaciones:

```java
 */
public class Principal {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Padre padre = new Padre("santa","claus");
        Hijo hijo = new Hijo(22,"juan","portilla"); //comparten atributos
        Padre hijo2 = new Hijo(12,"lura","portilla"); // tmb se podria inicializar de esta manera(polimorfismo)

        //Hijo hijo3 = new Padre(12,"lura","portilla"); // no se puede aplicar el contructor del padre

        padre.saludar(); //metodo saludar de padre
        hijo.saludar();// estamos llamdo al metodo que hay en la clase padre desde hijo
        hijo2.saludar();
        //padre.edad -> da error

    }
```

```
--- exec-maven
  Soy Padre
  Soy Padre
- Soy Padre
```

Java no admite herencia múltiple con clases, la herencia híbrida tampoco es posible con clases, pero podemos lograr el mismo resultado a través de Interfaces.

¿por qué no herencia múltiple? -> Las razones para omitir la herencia múltiple(MI) en el lenguaje Java en su mayoría se derivan del objetivo "simple, orientada a objetos, y familiar " que tenían los desarrolladores, además de pensar que agregar MI agregaba demasiada complejidad a los lenguajes mientras proporcionaba muy poco beneficio.

hay algunos artículos disponibles en la web con entrevistas de algunos de los diseñadores de java sobre este tema:
https://www.infoworld.com/article/2077394/why-not-multiple-inheritance.html

- **Siguiendo el ejercicio anterior, puedes proponer un ejemplo para diferenciar sobrecarga y sobreescritura. El ejemplo funcionará en consola.**

Override -> Son métodos pertenecientes a la clase "padre" de una Herencia, los cuales son declarados en la clase "hija" para modificar algún tipo de comportamiento más específico

Para explicar Overload primero hay que saber que es la **firma** de un método:
Es la combinación del nombre del método y la lista de parámetros(número, tipos y orden) .

Overload -> Consiste en poder crear un mismo método tantas veces como se quiera, pero han de tener diferentes

Diferencia-> override tiene que tener los mismo parámetros y del mismo tipo mientras que override tiene que cambiar la firma del método

Para el ejemplo reciclaremos las clases del anterior punto:
-La clase Padre se mantiene igual pero la hija tendrá un ligero cambio:

```java
//clase que hereda
public class Hijo extends Padre{ // se usa exteds(oalabra reservada) para especificar de quien se hereda

    int edad; //atributo propio de la clase hijo (no la tiene padre)

    public Hijo(int edad, String nombre, String Apellidos) { // si no tiene constructor da error
        super(nombre, Apellidos); //atributos heredados de Padre
        this.edad = edad;
    }

    @Override //siempre que hay sobreescrituar aparece esta etiqueta
    public void saludar() {
        System.out.println("hola soy HIJO");
    }

    //sobrecarga del metodo saludar añadiendole dos paramatros (por lo que cambia la firma)
    public void saludar(String saludo, int edad){
        System.out.println("Hola buenas soy "+saludo+" y  tengo "+edad+"años");
    }

    //sobreescritura cambiando el orden
    public void saludar(int edad, String saludo){
        System.out.println("Cambiando el orden");
    }

}
```

Prueba de que los métodos esta Ok

```java
 */
public class Principal {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Padre padre = new Padre("santa","claus");
        Hijo hijo = new Hijo(22,"juan","portilla"); //comparten atributos

        padre.saludar(); //metodo saludar de padre
        hijo.saludar();// estamos llamdo al metodo sobreescrito de hijo
        hijo.saludar("pedro",22);
        hijo.s
    }

}
```

```
saludar()                                  void
saludar(String saludo, int edad) void
saludar(int edad, String saludo) void
com.mycompany.t3_p_hi_juan_portilla.herencia.Hijo
```

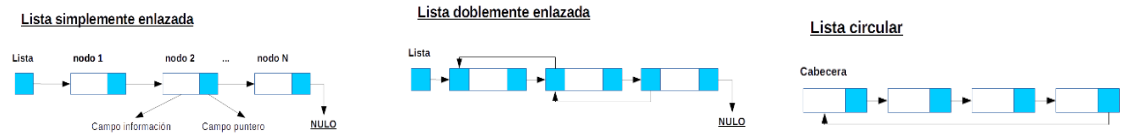
Ejecutamos pruebas en main :

```java
public class Principal {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Padre padre = new Padre("santa","claus");
        Hijo hijo = new Hijo(22,"juan","portilla"); //comparten atributos

        padre.saludar(); //metodo saludar de padre
        hijo.saludar();// estamos llamdo al metodo sobreescrito de hijo
        hijo.saludar("pedro",22);
        hijo.saludar(12, "jjjj");

    }

}
```

```
--- exec-maven-plugin:3.0.0:exec (default-
Soy Padre
hola soy HIJO
Hola buenas soy pedro y  tengo 22años
Cambiando el orden
------------------------------------------------------------
```

- En Java existen varias opciones para almacenar datos en colecciones. Explica con un ejemplo qué diferencia hay entre colecciones de tipo lista, pila, cola y tabla.

Lista (List)-> Estructura dinámica de datos que contiene una colección de elementos del mismo tipo de manera que se establece entre ellos un orden. Tipo:



Existen en Java principalmente dos implementaciones de List, las dos útiles en distintos casos:

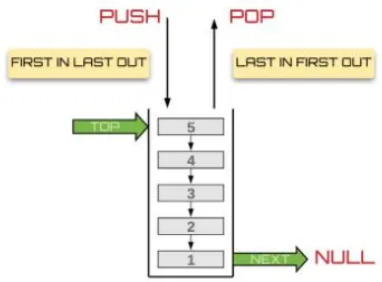
ArrayList-> Como un array pero redimensionable.
LinkedList-> En ésta, los elementos son mantenidos en una serie de nodos atados entre sí como eslabones de una cadena.

En este caso trabajaré con arrayList

```
List<String> listal = new ArrayList<String>();
```

```
//Listas / List
List<String> lista = new ArrayList<String>();
//La lista administra objetos de la clase String, luego mediante el método 'add' añadimos componentes al final:
lista.add("juan");
lista.add("Luis");
lista.add("Carlos");
//tambien se pùede añadir a una posicion en concreto y el resto se muevo
lista.add(2,"Sebas");

//Para eliminar un nodo de la lista debemos llamar al método 'remove' y pasar la posición del nodo a eliminar:
lista.remove(0);

//También podemos eliminar los elementos que coinciden con cierta información:
lista.remove("Carlos");

//La cantidad de elementos nos lo suministra el método 'size':
System.out.println("Cantidad de elementos en la lista:" + lista.size());

//Para conocer si la lista almacena cierto valor disponemos del método 'contains':
if (lista.contains("ana"))
    System.out.println("El nombre 'ana' si esta almacenado en la lista");
else
    System.out.println("El nombre 'ana' no esta almacenado en la lista");

//Para recuperar el dato de un nodo sin eliminarlo podemos hace uso del método 'get'
//(en un ArrayList este método es muy rápido y no depende de la cantidad de elementos):
System.out.println("El segundo elemento de la lista es:" + lista.get(1));

//Eliminamos todos los nodos de la lista mediante el método 'clear':
lista.clear();

//Podemos consultar si la lista se encuentra vacía mediante el método 'isEmpty':¡
if (lista.isEmpty())
    System.out.println("La lista se encuentra vacía");
```

Pila (Stack) -> Estructura lineal en la que los elementos (del mismo tipo) pueden ser añadidos o eliminados sólo por el final. (LIFO -> Last In First Out)



Una forma de trabajar con pilas es mediante nodos enlazados , a mi parecer es una forma muy enrevesada de tratar con ellos por lo que no voy a meterme en es tema,
por que Java ya nos aporta un colección para poder trabajar con pilas:

```
Stack <String> pila = new Stack<>();
```

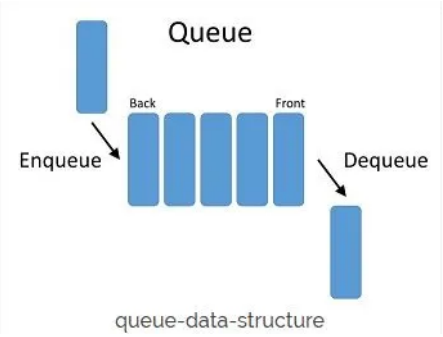
Las operaciones básicas con una pilas son:

.

```
//PILA / STACK
Stack <String> pila = new Stack<>();
//Para agregar elementos en la pila lo hacemos a través del método push:
pila.push("primero");
pila.push("segundo");
pila.push("tercero");
pila.push("cuarto");
pila.push("quinto");

//Para conocer la cantidad de elementos almacenados en la pila llamamos al método size:
System.out.println("Cantidad de elementos en la pila:" + pila.size());

//Para retirar un elementos de la pila disponemos del método pop: (ademas devuelve el elemneto retirado)
System.out.println("Extraemos un elemento de la pila:" + pila.pop());

//En cambio si queremos conocer el valor que hay en el primer elemento(cima) de la pila pero sin eliminarlo hacemos uso del método peek:
System.out.println("Consultamos el primer elemento de la pila sin extraerlo:" + pila.peek());

//Para saber si la lista esta vacia tenemos el método isEmpty (true si esta vacia , false si no)
while (!pila.isEmpty())
    System.out.print(pila.pop() + "-");

//para vaciar la pila
pila.clear();
```

```
Cantidad de elementos en la pila:5
Extraemos un elemento de la pila:quinto
Consultamos el primer elemento de la pila sin extraerlo:cuarto
cuarto - tercero - segundo - primero -
```

Cola (Queue) -> Lista lineal en la que los elementos (del mismo tipo) sólo pueden ser añadidos por un extremo y eliminados por el otro. (FIFO).



Con colas también se pueden usar nodos ,pero una vez más java tiene un tipo de colección especial para trabajar con colas. Las operaciones básicas con una colas son:

```
//La diferencia con respecto a la administración de pilas en Java
//es que para trabajar con colas debemos crear un objeto de la clase LinkedList e implementar la interfaz Queue:
Queue<Integer> cola = new LinkedList<>();
//Añadimos elementos al final de la cola mediante el método 'add':
cola.add(1);
cola.add(2);
cola.add(3);
cola.add(4);

//Para conocer la cantidad disponemos del método size():
System.out.println("Cantidad de elementos en la cola:" + cola.size());

//Para extraer el nodo de principio de la lista tipo cola lo hacemos mediante el método 'poll': (devolviendo el item)
System.out.println("Extraemos un elemento de la cola:" + cola.poll());

//i queremos conocer el objeto primero de la cola sin extraerlo lo hacemos mediante el método 'peek':
System.out.println("Consultamos el primer elemento de la cola sin extraerlo:" + cola.peek());

//Para conocer si la cola se encuentra vacía llamamos al método 'isEmpty':
while (!cola.isEmpty())
    System.out.print(cola.poll() + "-");

//Para eliminar todos los elementos de una pila empleamos el método clear:
cola.clear();
```

```
Cantidad de elementos en la cola:4
Extraemos un elemento de la cola:1
Consultamos el primer elemento de la cola sin extraerlo:2
2-3-4-
```

Tabla (Map) -> Disposición de datos en filas y columnas, es decir tendrá dos dimensiones. Hay dos tipos de tablas , mutables o inmutables

La clase HashMap utiliza datos genéricos tanto para la clave como para el valor, en este ejemplo la clave y el valor son datos de tipo String.

```java
//tablas (Map)
Map<String, String> mapa1 = new HashMap<String, String>();

//Mediante el método put agregamos un elemento a la colección de tipo HashMap:
mapa1.put("rojo", "red");
mapa1.put("verde", "green");
mapa1.put("azul", "blue");
mapa1.put("blanco", "white");

//Para imprimir todos los valores del mapa lo recorremos mediante un for:
for (String valor : mapa1.values())
    System.out.print(valor + "-");

//De forma similar si queremos recorrer todas las claves del mapa:
for (String clave : mapa1.keySet())
    System.out.print(clave + "-");

//Para recuperar un valor para una determinada clave llamamos al método 'get'
//y le pasamos la clave a buscar, si dicha clave no existe en el mapa se nos retorna el valor 'null':
System.out.println("La traducción de 'rojo' es:" + mapa1.get("rojo"));

//Si queremos verificar si una determinada clave existe en el mapa lo hacemos mediante el método 'containsKey':
if (mapa1.containsKey("negro"))
    System.out.println("No existe la clave 'negro'");

//Para eliminar un elemento de la colección debemos hacer uso del método 'remove', pasamos una clave del mapa:
mapa1.remove("rojo");

//Para imprimir el mapa completo en la Consola podemos hacer uso del método 'println':
System.out.println(mapa1);
```

```
red-white-green-blue-
rojo-blanco-verde-azul-
La traducción de 'rojo' es:red
{blanco=white, verde=green, azul=blue}
------------------------------------------------.
```

CUESTIÓN 2. Acceso a información con ficheros

- **Crea un archivo de texto con el enunciado de 10 preguntas. Desde consola muestre cada pregunta aleatoriamente. Cuando el usuario responde, se muestra la siguiente pregunta.**

- **En otro fichero de texto, o en el mismo anterior, añadimos las respuestas a las preguntas. Ahora, cuando el usuario responde, indica si la respuesta es correcta o no.**

- **Muestra la puntuación obtenida por el usuario. Cada respuesta acertada suma un punto, y cada errónea resta 0.5. La nota necesaria para aprobar es un 5**

Para este ejercicio yo he empleado dos ArrayList:

```
ArrayList<String> preguntasRespuestas = new ArrayList<>(); //tc
ArrayList<Integer> aleatorio = new ArrayList<>(); //indice
```

El primero almacena todas las preguntas y respuestas.
El segundo almacena los índices de las preguntas.

También declaro las variables que usaré después:

```
int posicion=0, acierto=1 ;
float fallo=0.5f, puntos = 0f;
String yo, pregunta ,respuesta;
```

añado todo el contenido del documento al arrayList preguntasRespuestas usando un scanner.

```
Scanner sc = new Scanner(Preguntas.class.getResourceAsStream("preguntsaYRespuestas.txt"));
```

Al tratar de leer el documento si no lo encuentra generará una excepción y parará el programa ,por lo que hacemos un try catch:

```
} catch (NullPointerException ne) {
    System.out.println("No se encontro el archivo. Erorr: "+ne);
}catch (Exception e){
    System.out.println(e);
}
```

También guardo las posiciones de las preguntas en el arrayList aleatorio (que serán las posiciones pares)

```
//traspasar los datos de preguntsaYRespuestas.txt a la lista pr
while (sc.hasNext()){
    //añado las pregunats y respuestas a pr
    preguntasRespuestas.add(sc.nextLine());

    if(posicion%2 == 0)
        aleatorio.add(posicion);

    posicion++;
}


   //remuevo la pòsicion de las preguntas
   Collections.shuffle(aleatorio);


  //lo convierto en iterator para ecorrerlo mas facil
  Iterator <Integer> preguntasAleatorias = aleatorio.iterator();
```

Como necesito leer por teclado creo otro Scanner:

```
Scanner lc = new Scanner(System.in);
```

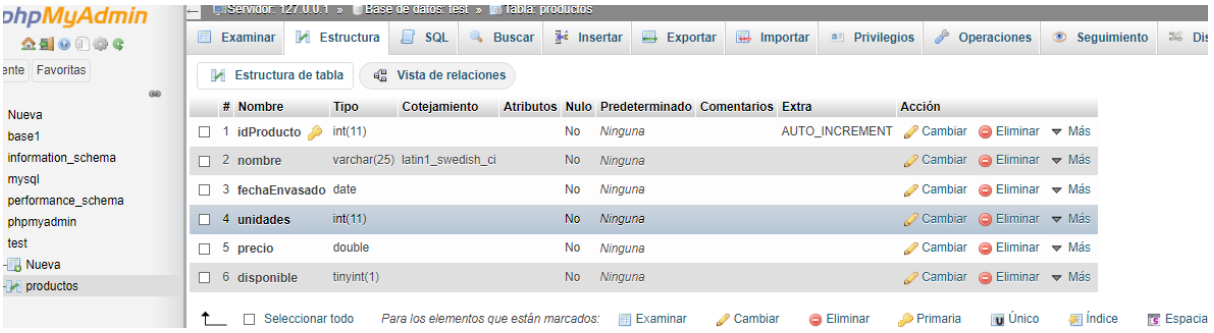
Y ahora recorro el iterador que contiene las posiciones de las preguntas (preguntasAleatorias) , lo guardo en la variable posición para que no quede un codigo muy difícil de entender , ya al final si acierta la pregunta se suma un punto y si falla se le resta 0.5

```
while(preguntasAleatorias.hasNext()){
    //para no tener que escribir mas y que el codigo quede mas limpio reutilizo esta variable
    posicion=preguntasAleatorias.next();

    //saca las preguntas aleatoriamente
    pregunta = preguntasRespuestas.get(posicion);
    respuesta = preguntasRespuestas.get(posicion + 1);
    //la convierto a mayusculas
    System.out.println(pregunta);
    yo = lc.nextLine();

    if(yo.equalsIgnoreCase(respuesta)){
        System.out.println("Correctoo!!! -> +"+acierto+"\n");
        puntos += acierto;
    }else{
        System.out.println("falaste.... -> -"+fallo+"\n");
        puntos -= fallo;
    }
}
```

Finalmente muestro por consola los puntos obtenidos ,si ha aprobado o no y cierro los dos Scanners que he usado.

CUESTIÓN 3. Java CRUD

- **Gestión de base de datos relacionales. Utilizando MySQL crea una base de datos en donde dispongamos de una tabla que permita almacenar los siguientes datos de productos (idProducto, nombre, fechaEnvasado, unidades, precio, disponible).Disponible será un valor boleano.**

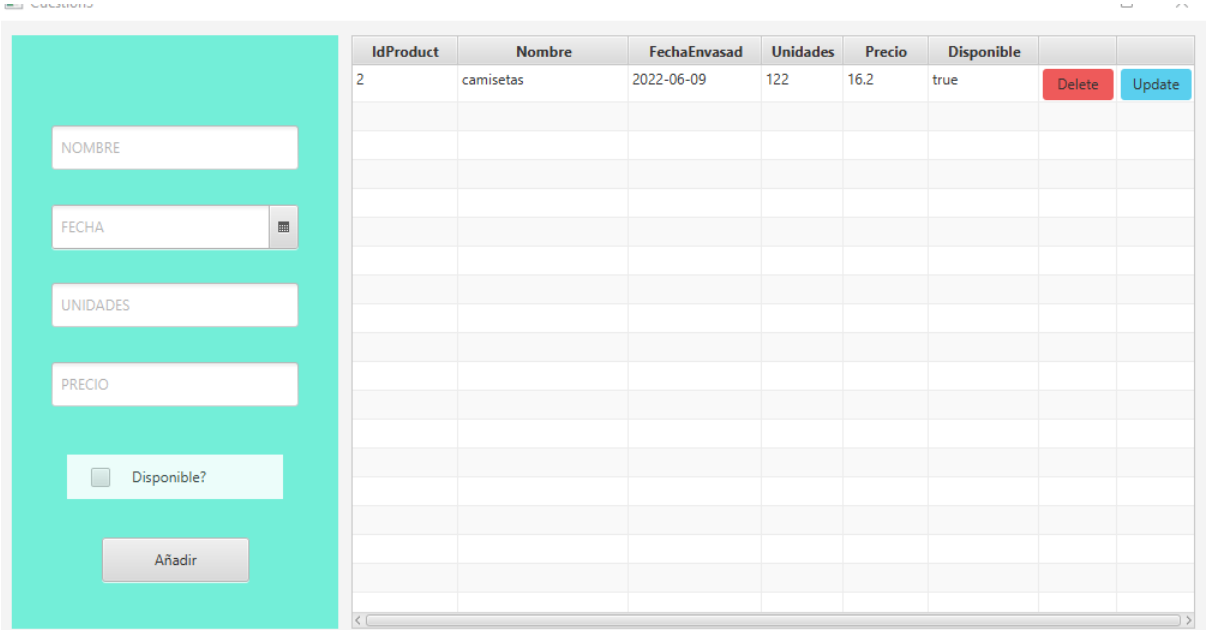


- **Realiza una aplicación que nos permita añadir, eliminar, actualizar y mostrar los productos. Es válido realizar esta tarea por consola.**

- **Desarrolla una aplicación de escritorio con Java Swing o Java FX para diseñar la capa visual del ejercicio.**

-El proyecto se ha realizado en Intellij Community IDEA V. 2022.1.2
-Java 11
-JavaFx 18.0.1
-Scene Builder 18.0.0

Primera Visualización:

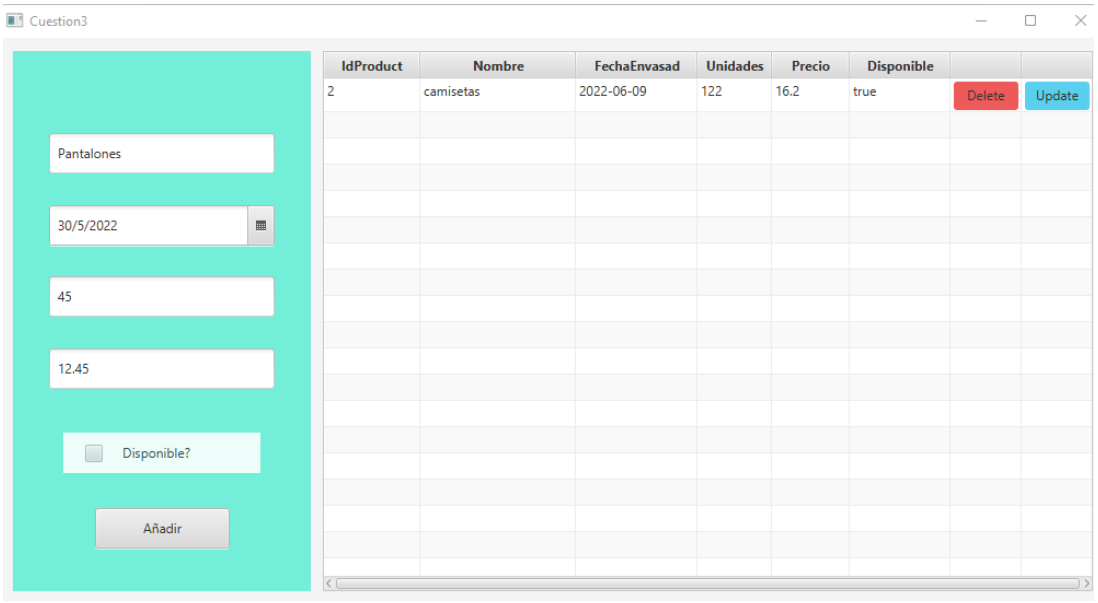


En este documento se explicará la funcionalidad incluyendo los trozos de código más relevantes , se adjuntará aparte el zip del código.

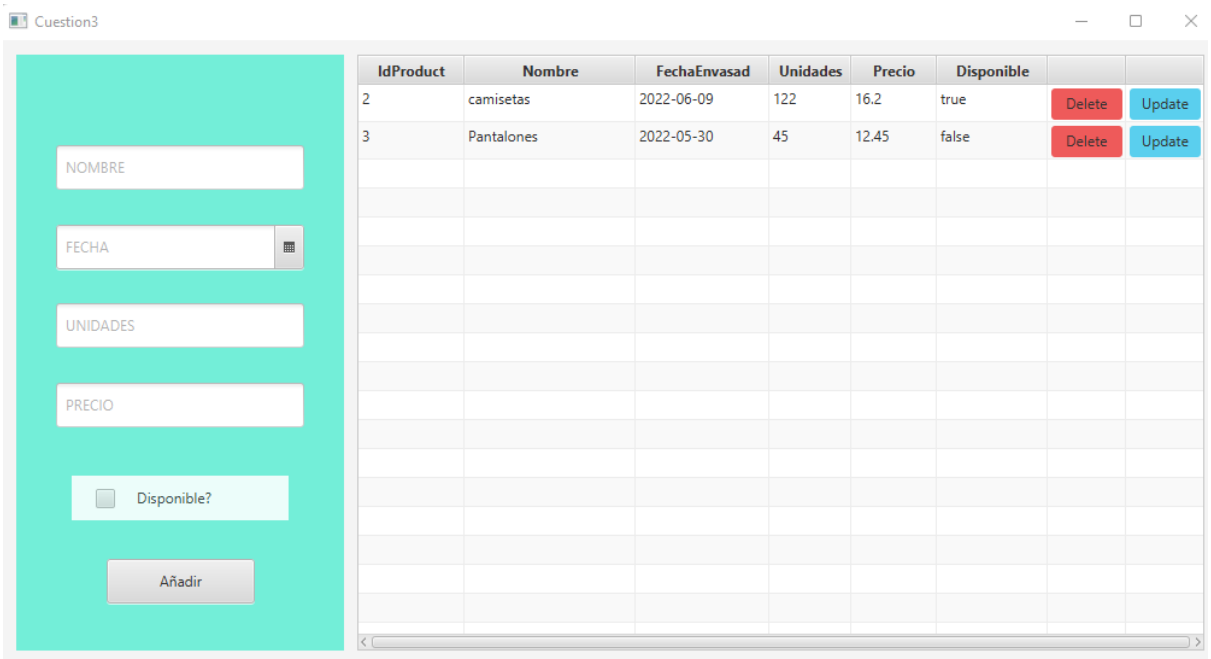
Añadir (Insertar en base de datos y ver en tabla):

Desde el panel de fondo azul introducimos los datos del producto , los cuales se guardaran en la base de datos (local) PhpMyAdmin cuando pulsemos el botón “Añadir”,, y en la tabla se añadirá una fila con este producto insertado además de los botones de “Delete” y “Update”.

Como se puede ver en la siguiente imagen añadimos Pantalones



Y al dar al botón ”Añadir” además de insertar la fila, restablece los campos a introducir.



Código:

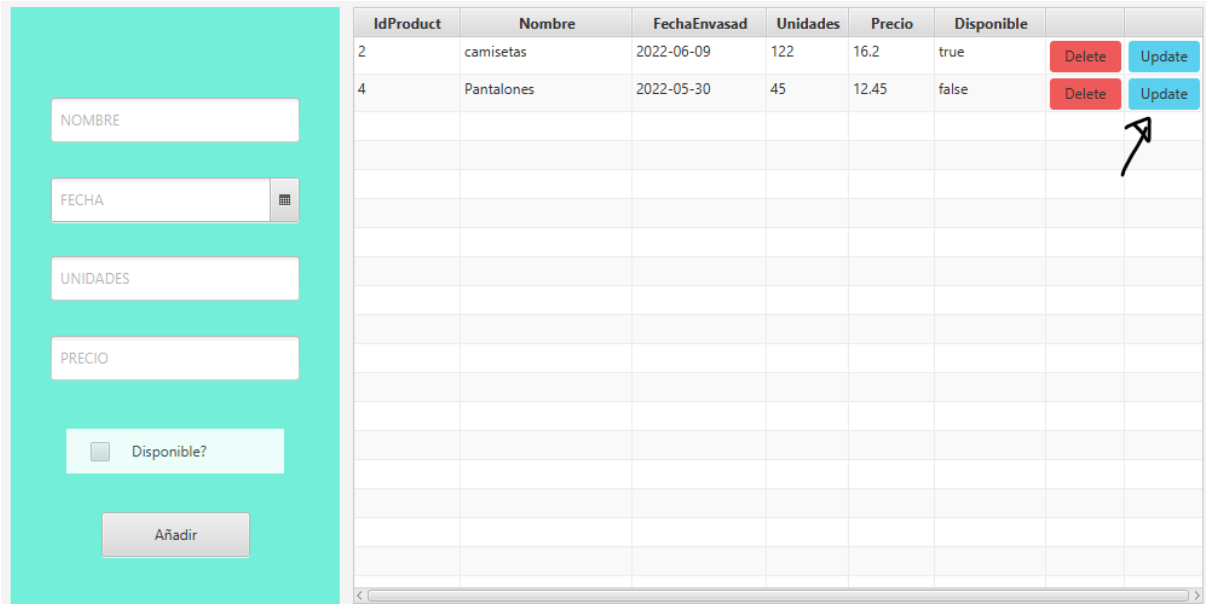
El botón ”Añadir” tiene el evento “add”:

```java
@FXML
protected void add() {

    //Se conecta a la base de datos
    this.connect.conectar();

    //parsea algunos campos necesarios
    Date date = Date.from(pickerFecha.getValue().atStartOfDay(ZoneId.systemDefault()).toInstant());
    int unidades = Integer.parseInt(txtUnidades.getText());
    double precio =  Double.parseDouble(txtPrecio.getText());

    // extrae los elementos de los TextFields y crea un objeto producto para luego insertarlo
    Product producto = new Product(txtNombre.getText(), date, unidades, precio, checkDisponible.isSelected());
    this.connect.insertar(producto); //inserta el producto

    //Refresca la tabla para mostrar la nueva fila añadida
    Refresh();

    //cierra conexion
    this.connect.cerrar();

    // reestablece los campos
    undo();
```

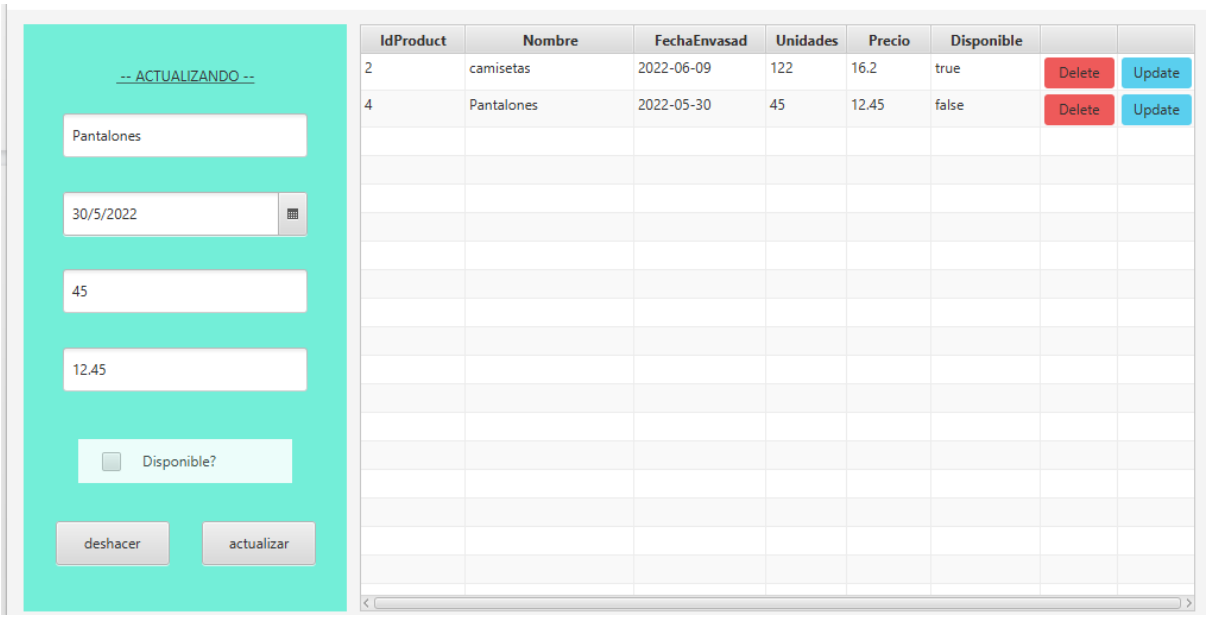
Método insertar de la clase conexion:

```java
public void insertar(Product p){
    //try catch para capturar una excepcion Sql con un mensaje personalizados
    try {
        //parseamos campo java.util.date a java.sql.date
        Date sqlDate =  new java.sql.Date(p.getFechaEnvasado().getTime());

        //CONSULTA Insert
        String consulta="INSERT INTO `productos` VALUES (?, ?, ?, ?, ?, ?)";
        PreparedStatement  st =this.conn.prepareStatement(consulta);

        //los parametros(?) de la consulta insert
        st.setString ( parameterIndex: 1, x: null);
        st.setString ( parameterIndex: 2, p.getNombre());
        st.setDate   ( parameterIndex: 3, sqlDate);
        st.setInt    ( parameterIndex: 4, p.getUnidades());
        st.setDouble ( parameterIndex: 5, p.getPrecio());
        st.setBoolean( parameterIndex: 6, p.getDisponible());

        //Ejecutamos setencia
        st.execute();

    }catch (SQLException e) {
        System.out.println("HA FALLADO EN HACER LA CONSULTA insertar");
        e.printStackTrace();
    }
}
```

Update (Actualizar producto en la base de datos y actualiza en la tabla):
El botón “Update” muestra en el modo actualización la línea a la que pertenece



Al Clicar el botón “Update” cambia ligeramente la ventana(modo actualizar) , añadiendo la información a los campos de la izquierda y cambia los botones de abajo para facilitar la actualización:



Si le das a “deshacer” , sales de este modo y si le das “actualizar” actualiza el producto y restablece los campos.

Código:

```
//creamos boton Update y lo añadimos a la tabla
3 usages
private final Button btn = new Button( s: "Update");
{
    //le ponemkos un poco de estilo
    btn.setStyle("-fx-background-color: #5ACFEE; -fx-pref-width: 200px;");
    //añadimos el evento
    btn.setOnAction((ActionEvent event) -> {

        //deshabilitar el boton de Añadir
        btnAdd.setDisable(true);
        btnAdd.setVisible(false);

        //habilita los botones de deshacer y actualizar
        btnActualizar.setDisable(false);
        btnActualizar.setVisible(true);
        btnActualizar.setOpacity(1);

        btnDeshacer.setDisable(false);
        btnDeshacer.setVisible(true);
        btnDeshacer.setOpacity(1);

        //habilita la label para avisar que estas en el modo actualizar
        lblMessage.setDisable(false);
        lblMessage.setText(" -- ACTUALIZANDO --");

        //el propio boton recoge a que producto nos referimos
        Product data = getTableView().getItems().get(getIndex());

        //rellena los campos de la isquierda
        visualizarUpdate(data);
    });
}
```

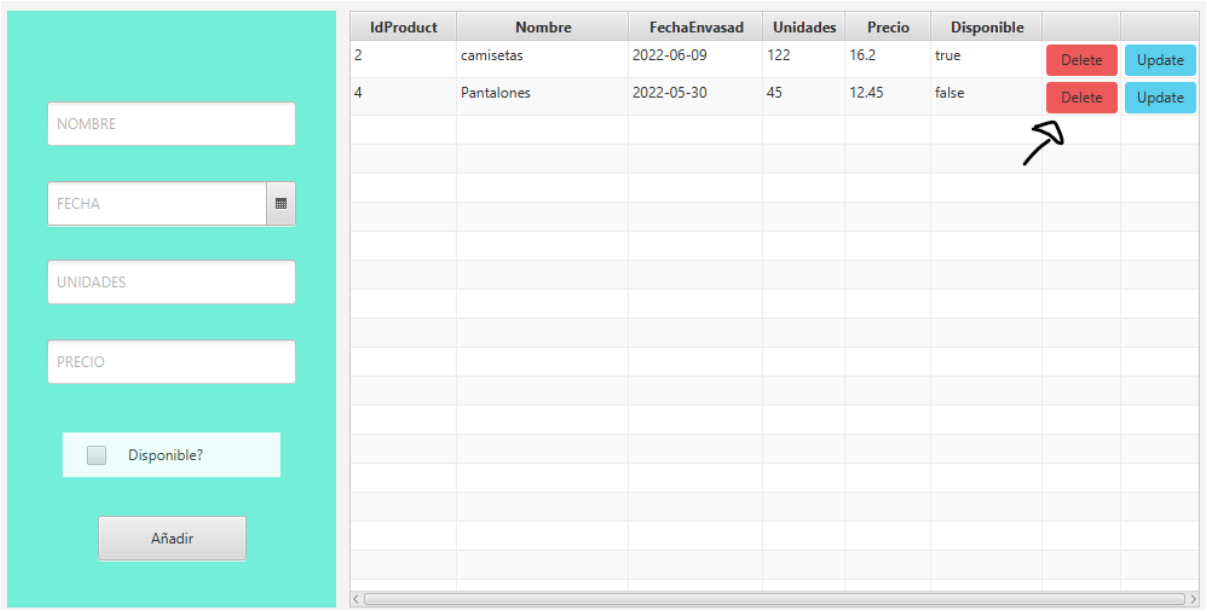
El botón actualizar de la derecha:

```java
@FXML
protected void update(){

    //se parsean algunos campos
    Date date = Date.from(pickerFecha.getValue().atStartOfDay(ZoneId.systemDefault()).toInstant());
    int unidades = Integer.parseInt(txtUnidades.getText());
    double precio =  Double.parseDouble(txtPrecio.getText());

    //creamos producto para actualizar
    Product p = new Product(this.key,txtNombre.getText(), date, unidades, precio, checkDisponible.isSelected());

    //conectamos a la base de datos
    this.connect.conectar();

    //pasa como parametro los datos del producto a actualizar
    this.connect.Update(p);

    //cerramos conexion
    this.connect.cerrar();

    //refrescamos y restablecmos campos
    Refresh();
    undo();
}
```

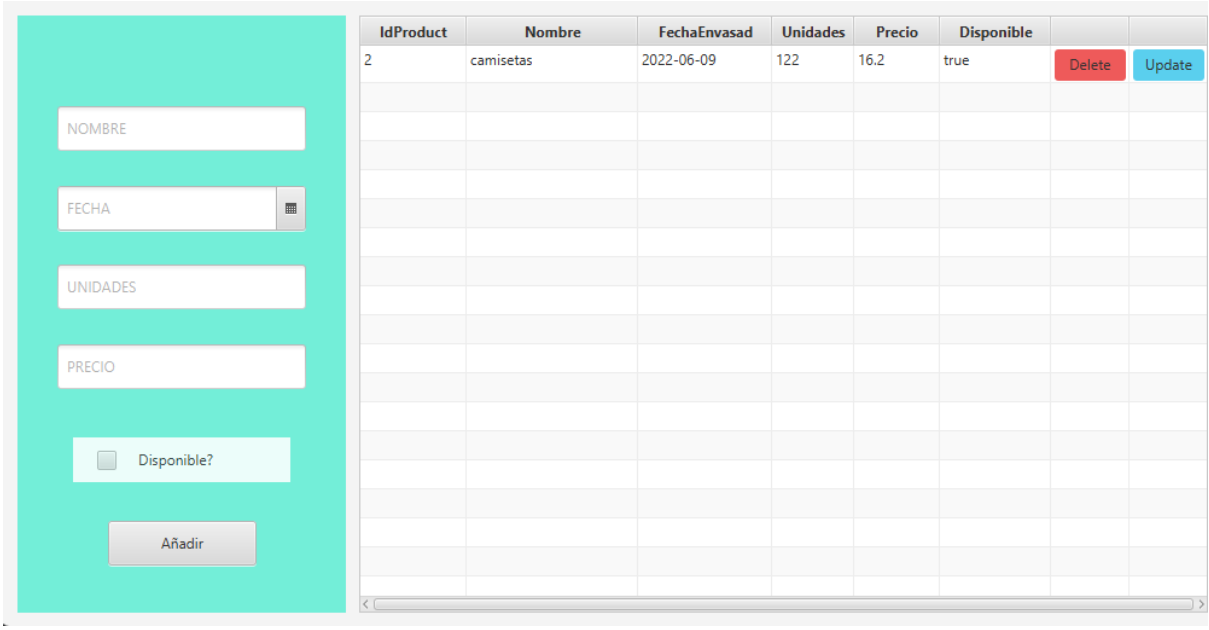
Método update de la clase conexion:

```java
public  void Update(Product p){
    try {

        Date sqlDate =  new java.sql.Date(p.getFechaEnvasado().getTime());

        //CONSULTA      updates
        String consultaUpdate = "UPDATE `productos` SET  `nombre` = ?, `fechaEnvasado` = ?, `unidades` = ?, `precio` = ?, `disponible` = ? WHERE `productos`.`idProducto` = ?";
        PreparedStatement st = this.conn.prepareStatement(consultaUpdate);

        st.setString ( parameterIndex: 1, p.getNombre());
        st.setDate   ( parameterIndex: 2, sqlDate);
        st.setInt    ( parameterIndex: 3, p.getUnidades());
        st.setDouble ( parameterIndex: 4, p.getPrecio());
        st.setBoolean( parameterIndex: 5, p.getDisponible());
        st.setInt    ( parameterIndex: 6, p.getIdProducto());

        st.execute();

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Delete (Eliminar de la base de datos y actualiza tabla)
El botón “Delete” elimina directamente el producto de la línea a la que pertenece



Elimina la esa fila y el registro en la base de datos.



Código:

el botón “Delete”:

```
//Creamos boton delete y lo añadimos al la tabla
@Override
public TableCell<Product, Void> call(final TableColumn<Product, Void> param) {
    final TableCell<Product, Void> cell = new TableCell<Product, Void>() {

        3 usages
        private final Button btn = new Button( s: "Delete");
        {
            //añadimos un poco de estilo
            btn.setStyle("-fx-background-color: #EE5A5A; -fx-pref-width: 200px;");
            //añadimos el evento Delete
            btn.setOnAction((ActionEvent event) -> {

                //el propio boton recoge a que producto nos referimos
                Product data = getTableView().getItems().get(getIndex());

                //Creamos un objeto de tipo conexion y conetamos con la base de datos
                Conexion c= new Conexion();
                c.conectar();

                //introduce el idProducto para luego buscarlo en la base datos y eliminarlo
                c.delete(data.getIdProducto());

                //cerramos conexion
                c.cerrar();

                //refrescamos tabla
                Refresh();
            });
        }
```

Método delete de la clase conexion:

```
1 usage
public  void delete(int id){
    try {
        //CONSULTA delete
        String consultaDelete="DELETE FROM `productos` WHERE `productos`.`idProducto` = ?";
        PreparedStatement st = this.conn.prepareStatement(consultaDelete);
        st.setInt       ( parameterIndex: 1,id);

        st.execute();

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```