Criterio C

I. Clases y Objetos

Se crearon 4 clases: Producto representa los artículos en inventario; EntregaProveedor representa las entregas; Venta representa las ventas a un cliente; y Cliente representa un cliente registrado. Usar cuatro clases permitió representar múltiples instancias de los cuatro tipos de elementos administrados, cada uno con características y comportamientos propios (una venta cuenta con atributos distintos a los de una entrega, por ejemplo).

Las clases están relacionadas (una venta está asociada a un cliente, por ejemplo) mediante variables de identificación Integers (como clienteVenta en la clase Venta, que hace referencia al código único que cada cliente tiene, haciendo posible identificar al comprador), permitiendo optimizar memoria, lo que no hubiera sido posible de haber utilizado agregación directamente (donde el objeto sería el atributo, y no solo el Integer), ya que sólo en ciertos casos son requeridos los datos del objeto relacionado con este identificador.

II. Guardado y acceso a la información

Para almacenar permanentemente la información se utilizaron cuatro archivos de texto, uno para cada clase. Cada línea en los archivos representa un registro, y la información de los atributos de cada objeto se separaró con un carácter especial.

Para las operaciones fueron diseñados métodos que generaban arreglos de cada tipo (según la clase del objeto) y los llenaban con la información de los archivos de texto para manejar la información como objetos (ejemplo en la Figura 1), y métodos que recibían los arreglos (después de haberse realizado una operación) y guardaban la información actualizada en los archivos de texto (ejemplo de la clase Entrega en la Figura 2). Ésto fue útil para mantener actualizada la información de objetos de distintas clases (pero que estuvieran relacionados) al guardar una operación. Por ejemplo, al registrarse una venta, se llenan los arreglos de Cliente, Venta y Producto. Además de que se crea un nuevo objeto de la clase Venta, se actualizan los datos en los arreglos de Cliente (el número y valor de compras), y de Producto (el número de stock restante), como se puede ver en las figuras 3 y 4, donde después de guardar una venta se actualiza el stock del producto vendido. Dado que el usuario elegía los objetos involucrados, se accedía directamente al número de registro en los arreglos que correspondiera a la variable de identificación, y se actualizaban los atributos de los objetos, ya que en los arreglos los objetos se encuentran ordenados por código de identificación de manera ascendente. Al finalizar la operación se envían los arreglos actualizados a los métodos que guardan los nuevos datos en los archivos de texto, como se observa en la figura 5.

```java
//Éste metodo leerá del archivo de texto que almacena los datos de las entregas, y los almacenará en un arreglo de EntregaProveedor, que será regresado para que pueda ser trabajado
public static EntregaProveedor[] llenarArregloEntrega() throws IOException{ // Inicio 2

  //Se crea el stream con el archivo y se cierra, sólo para crear uno en caso de que no exista.
  PrintWriter out = new PrintWriter(new FileOutputStream("Entregas.txt", true));
  out.close();

  //Se crean dos scanner al archivo de texto; uno leerá cuantas lineas tiene el archivo, y leerá su contenido.
  Scanner file = new Scanner(new File ("Entregas.txt"));
  Scanner file2 = new Scanner(new File ("Entregas.txt"));

  //Se define el tamaño del arreglo
  int lineas = 0;
  if (file2.hasNext()){
    do{
    file2.nextLine();
    lineas++;
    } while (file2.hasNext());
  }

  EntregaProveedor[] arr = new EntregaProveedor[lineas]; //Tamaño del numero de lineas que haya y es de la clase Producto

  if (file.hasNext()){ //Mientras existan líneas a leer en archivo se va a llenar el arreglo
    for (int x=0; x<lineas; x++){
    arr[x] = new EntregaProveedor(); //Se crea pendiente por casilla en arreglo
    String line = file.nextLine(); //Se lee línea del archivo
    String delimitador = "#"; //Se establece el delimitador entre la información para cada atributo
    String[] arregloTemporal = new String[6]; //Se crea arreglo temporal para guardar el resultado de .split
    arregloTemporal = line.split(delimitador); //Se parte la linea leida
    arr[x].setProductoEntrega(Integer.parseInt(arregloTemporal[0])); //Se guarda el codigo del producto entregado
    arr[x].setFolioEntrega(Integer.parseInt(arregloTemporal[1])); //Guarda el folio de la entrega
    arr[x].setNombreProveedor(arregloTemporal[2]); //Guarda el nombre del proveedor
    arr[x].setFechaEntrega(arregloTemporal[3]); //Guarda la fecha de la entrega
    arr[x].setCantidadProductoEntrega(Integer.parseInt(arregloTemporal[4])); //Guarda la cantidad de producto entregado
    arr[x].setValorEntrega(Double.parseDouble(arregloTemporal[5])); //Guarda el valor de la entrega
    }
  }

  //Se regresa el arreglo lleno
  return arr;
} // Fin 2
```

Figura 1

```java
//Éste metodo guardará en el archivo de texto la información en el arreglo de EntregaProveedor
public static void guardarArchEntrega(EntregaProveedor[] arr) throws IOException{ // Inicio 2

  //Se crea el stream con el archivo
  PrintWriter out = new PrintWriter(new FileOutputStream("Entregas.txt"));

  //Se guardan todos los datos de los objetos en el arreglo, separados por el caracter #
  for (int x=0; x<arr.length; x++){
   out.println(arr[x].getProductoEntrega() + "#" + arr[x].getFolioEntrega() + "#" + arr[x].getNombreProveedor() + "#" + arr[x].getFechaEntrega() + "#" + arr[x].getCantidadProductoEntrega() + "#"
  }

  //Se cierra el stream
  out.close();
} // Fin 2
```

Figura 2

Figura 3

Figura 4

```java
//Esta será la nueva venta
Venta estaVenta = new Venta();
//Se crea el objeto para poder accesar a los archivos de texto
ArchivosBinarios ab = new ArchivosBinarios();
//Se llena el arreglo con los datos de los productos en inventario
Producto[] arrProducto = ab.llenarArregloProducto();
//Se llena el arreglo con los datos de los clientes registrados
Cliente[] arrCliente = ab.llenarArregloCliente();

//Folio venta
Venta[] arrVenta = ab.llenarArregloVenta();
estaVenta.setFolioVenta(arrVenta.length+1);

//Guardado del código en el registro de ventas
estaVenta.setProductoVenta(arrProducto[codProducto-1].getCodigoProducto());

//Leer códgio cliente al que se vende
//Guardado del código en el registro de ventas
estaVenta.setClienteVenta(arrCliente[codCliente-1].getCodigoCliente());
//Se agrega una venta al cliente
arrCliente[codCliente-1].setNumeroComprasCliente(arrCliente[codCliente-1].getNumeroComprasCliente() + 1);

//Cantidad vendida
arrProducto[codProducto-1].setStockProducto(arrProducto[codProducto-1].getStockProducto() - cantidadVendida);
//Se guarda en el registro de esta venta
estaVenta.setCantidadProductoVenta(cantidadVendida);

//Valor de la venta (Se calcula multiplicando el precio del producto por la cantidad vendida)
double valorEstaVenta = estaVenta.getCantidadProductoVenta()*arrProducto[codProducto-1].getPrecioProducto();
estaVenta.setValorVenta(valorEstaVenta);
//Se guarda en el registro de clientes
arrCliente[codCliente-1].setValorComprasCliente(arrCliente[codCliente-1].getValorComprasCliente() + valorEstaVenta);

//Fecha Venta
estaVenta.setFechaVenta(fecha);

//Guardar la nueva venta en el archivo de ventas
PrintWriter out1 = new PrintWriter(new FileOutputStream("Ventas.txt", true));
out1.println(estaVenta.getClienteVenta() + "#" + estaVenta.getProductoVenta() + "#" + estaVenta.getFolioVenta() + "#" + estaVenta.getFechaVenta() + "#" + esta
out1.close();

//Guardar arreglo de Clientes con nueva info
ab.guardarArchCliente(arrCliente);

//Guardar arreglo de Venta
ab.guardarArchProducto(arrProducto);
```

Figura 5

III. Métodos de búsqueda

Para consultar los registros, el usuario debía seleccionar de una lista el objeto del cual deseaba consultar información (ver figura 6). Ya que en muchas ocasiones se desplegaba información contenida en distintos arreglos (pero relacionada con el objeto elegido), el diseño de las clases permitió que todas las búsquedas del programa se hicieran con base en las variables de identificación, obtenidas cuando el usuario elegía un objeto de la lista. Por ejemplo, en el registro de clientes, se desplegaban las compras del cliente elegido. Ésto se hacía con una búsqueda secuencial en el arreglo de Venta mediante un ciclo “while”, donde todos los objetos cuyo atributo clienteVenta fuera el mismo que la variable codigoCliente del cliente seleccionado por el usuario se imprimían en una lista (ver figura 7). Las búsquedas son todas secuenciales, dado que los arreglos que contienen los objetos de Venta y EntregaProveedor están ordenados por folio, ya que se guardan según ocurran y no por el código de producto, lo que implica que, por ejemplo, tanto el tercer como el vigésimo registro podrían ser ventas del cliente buscado, creando la necesidad de verificar cada registro.

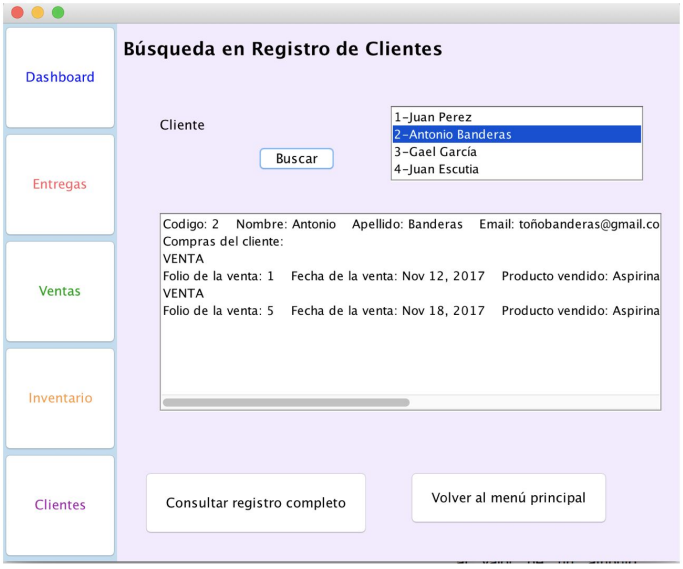

Figura 6

```
private void jButton33ActionPerformed(java.awt.event.ActionEvent evt) {
    //Este boton se encarga de desplegar el historial del cliente elegido

    //Se resetea el label de errores
    labelError8.setText("");

    //Se declaran los objetos necesarios
    int codigoCliente;
    String line;
    ArchivosBinarios ab = new ArchivosBinarios();
    Cliente[] arrCliente;

    try{
        //Se declaran los objetos necesarios
        arrCliente = ab.llenarArregloCliente();
        Venta[] arrVenta = ab.llenarArregloVenta();
        Producto[] arrProd = ab.llenarArregloProducto();

        //Se obtinene el código del producto vendido
        line = jList7.getSelectedValue();
        String delimitador = "-"; //Se establece el delimitador entre la información para cada atributo
        String[] arregloTemporal = new String[2]; //Se crea arreglo temporal para guardar el resultado de .split
        arregloTemporal = line.split(delimitador); //Se parte la linea
        codigoCliente = Integer.parseInt(arregloTemporal[0]); //Se establece el código del cliente

        //Se despliegan los atributos del producto seleccionado
        DefaultListModel dlm = new DefaultListModel();
        dlm.addElement("Codigo: " + arrCliente[codigoCliente-1].getCodigoCliente() + "    Nombre: " + arrCliente[codigoCliente-1].getNombreCliente() + "    A
        dlm.addElement("Compras del cliente:");

        //Se busca en el arreglo de ventas, si el codigo del cliente buscado es el de la venta, se va a desplegar.
        for(int x=0; x<arrVenta.length; x++){
            if(arrVenta[x].getClienteVenta() == codigoCliente){
                dlm.addElement("VENTA");
                dlm.addElement("Folio de la venta: " + arrVenta[x].getFolioVenta() + "    Fecha de la venta: " + arrVenta[x].getFechaVenta() + "    Producto
            }
        }
        jList6.setModel(dlm); //Se despliega la lista
    }
    catch(Exception e){
        labelError8.setText("Eliga un cliente"); //En caso de que no se eliga un cliente
    }
}
```

Figura 7

IV. Método de ordenamiento

Existió la necesidad de ordenar una lista de productos para desplegar solamente los cinco más vendidos (ver Figura 8). Para esto, se creó un arreglo de tipo Venta con tamaño del número de productos registrados, y se recorrió el arreglo del registro de ventas con para encontrar las ventas en las que se vendió cada producto (con una búsqueda secuencial, como se explicó en la parte III) y la cantidad vendida. Cada objeto de este nuevo arreglo representa un producto registrado, y uno de sus atributos (cantidadProductoVenta) es la cantidad de unidades vendidas del mismo. Después, este arreglo se ordenó de acuerdo al valor del atributo mencionado(ver Figura 9). Se usó el método *sort* (que funciona con el principio del *merge sort*, optimizado para ser más rápido) de la clase *Arrays* , y a la clase Venta se le implementó la

interfaz *Comparable*. Ya que el método usado ordena de manera ascendente los objetos, se tomaron los último cinco productos del arreglo ordenado (o todos en caso de que existieran menos de cinco productos en el registro) y se desplegaron en una lista.

```
//Se crea un nuevo arreglo de ventas, para que sea ordenada por el numero de productos vendidos (en todas las ventas)
Venta[] arrVenta2 = new Venta[arrProducto.length];

//Aquí se llena el segundo arreglo de ventas
for (int x=0; x<arrProducto.length; x++){
    //Cada casilla del nuevo arreglo se llena con las ventas que se hayan realizado del mismo
    arrVenta2[x] = new Venta();
    arrVenta2[x].setProductoVenta(x+1);
    for (int y=0; y<arrVenta.length; y++){
        if (arrVenta[y].getProductoVenta() == (x+1)){
            arrVenta2[x].setCantidadProductoVenta(arrVenta2[x].getCantidadProductoVenta() + arrVenta[y].getCantidadProductoVenta());
        }
    }
}

//Se ordena el arreglo de ventas que contiene las ventas totales de cada producto.
Arrays.sort(arrVenta2);

//Se llena el modelo de la lista que se desplegara
DefaultListModel dlm = new DefaultListModel();
if(arrVenta2.length<5){
    for (int x=arrVenta2.length; x>0; x--){
        dlm.addElement("Nombre del producto: " + arrProducto[arrVenta2[x-1].getProductoVenta()-1].getNombreProducto() + "    Marca: " + arrProducto[arrV
    }
}
else {
    for (int x=arrVenta2.length; x>arrVenta2.length-5; x--){
        dlm.addElement("Nombre del producto: " + arrProducto[arrVenta2[x-1].getProductoVenta()-1].getNombreProducto() + "    Marca: " + arrProducto[arrV
    }
}
jList13.setModel(dlm);
```

Figura 8

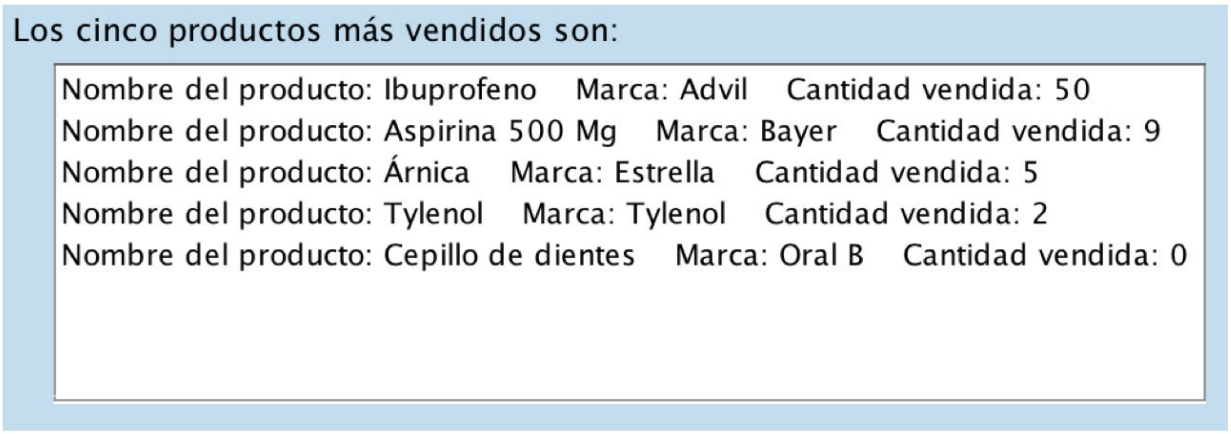

Figura 9

V. Validaciones

Las validaciones fueron una gran importancia para asegurarse de que los datos ingresados por el usuario cumplieran siempre con las condiciones requeridas para que el sistema funcionara. Para la implementación de la interfaz, al momento de que el usuario quisiera registrar una operación, se validaban todos los datos ingresados vía los campos de texto. Las dos principales validaciones que se llevaron a cabo en casi todas las posibles acciones del usuario fueron por formato, en las cuales se validaba que, por ejemplo, se ingresara un número en un campo de precio, y que éste fuera positivo (ver figura 10). También se validó que el usuario nunca dejara en blanco un campo de texto, para evitar que se registraran operaciones con

datos faltantes. Finalmente, una validación especial fue diseñada para registrar una venta, en donde se validaba que la cantidad de producto vendido no fuera mayor a la existente en stock, por lo que al querer registrar una venta se consultaba el inventario y se validaba la cantidad ingresada. Para todas las validaciones, en caso de que algún dato ingresado por el usuario fuera erróneo, se desplegaba un mensaje que indicaba el problema. La figura 11 es un ejemplo del código usado para validar un número en una venta.

Figura 10

```
cantidadVendida = Integer.parseInt(jTextField14.getText()); //Se obtiene la cantidad vendida

if (cantidadVendida <= 0) //Se valida que la cantidad vendida sea mayor que 0
    throw eNum;
ArchivosBinarios ab = new ArchivosBinarios();
Producto[] arrPro = ab.llenarArregloProducto();
for(int x=0; x<arrPro.length; x++){
    if ((arrPro[x].getCodigoProducto()) == (codigoProducto)){
        if (arrPro[x].getStockProducto() < cantidadVendida)
        throw eNum; //Se valida que la cantidad vendida no sea mayor que el stock del producto
    }
}

//Se obtiene y se valida la fecha
Date date = jDateChooser3.getDate();
    fechaVenta = DateFormat.getDateInstance().format(date);
if(fechaVenta.equals(""))
    throw e;
}
catch(NumberFormatException errorNum){
    labelError4.setText("Error al ingresar un número");
    status = false;
}
catch(Exception e){
   labelError4.setText("Llene todos los campos");
   status = false;
}
```

Figura 11

(1001 palabras)

Referencias

Apache POI - the Java API for Microsoft Documents. (n.d.). Retrieved October 20, 2017, from https://poi.apache.org/

Oracle. (n.d.). Arrays (Java Platform SE 7). Retrieved November 1, 2017, from https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

Oracle. (n.d.). Comparable (Java Platform SE 7). Retrieved November 1, 2017, from https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

Mkyong. (2010 July 7). Java object sorting example (Comparable and Comparator). Retrieved November 1, 2017, from https://www.mkyong.com/java/java-object-sorting-example-comparable-and-comparator/

Oracle.(n.d.). How to Use Lists. Retrieved October 15, 2017, from https://docs.oracle.com/javase/tutorial/uiswing/components/list.html

Gul. S. (n.d.). Introduction to GUI Building. Retrieved October 1, 2017, from https://netbeans.org/kb/docs/java/gui-functionality.html

Mkyong. (2009 December 24). Java - How to get current date and time. Retrieved November 2, 2017, from https://www.mkyong.com/java/java-how-to-get-current-date-time-date-and-calender/

Oracle. (n.d.). How to Write an Action Listener (Java Platform SE 7). Retrieved October 1, 2017, from https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html