**INTERNATIONAL BACCALAUREATE**

**INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY**

*Prepa Tec Santa Catarina*

**Computer Science SL**

*Ferretería*

Written by

Name: Mauro Amarante

Candidate Number: dvs580

School: Tecnologico de Monterrey-Campus Santa Catarina

School Number: 002267

**Table of Contents:**

**A1. Analysis of the Problem……………………………………………………………………3**

**A2. Criteria for Success…………………………………………………………………………8**

**A3. Prototype Solution…………………………………………………………………………..10**

**B1. Data Structures………………………………………………………………………………15**

**B2. Algorithms…………………………………………………………………………………….24**

**B3. Modular Organization………………………………………………………………………29**

**Code Listing………………………………………………………………………………………..31**

**C2. Handling Errors………………………………………………………………………………72**

**D1. Annotated Hard Copy………………………………………………………………………..77**

**Mastery Aspects……………………………………………………………………………………95**

**D2. Evaluating Solutions…………………………………………………………………………96**

**A1. Analysis of the Problem:**

My client owns a local hardware store that offers very diverse merchandise that could be used for home use, industrial use, construction use, etc. From my client’s diverse merchandise some of the products that he offers are screws, hammers, pipes, cement, sandpaper, light bulbs, electric utilities, paint, etc. My client has had some problems for a while now at his store with their organizational system for their merchandise. Currently my client is using a system-based organization that has been very useful for him in the past years, but has brought some problems that are affecting his business. This previously mentioned system (System Screenshot Appendix B) is very simple software that was previously bought and currently used by my client; he also mentioned “it has a simple but complicated interface for new or even current employees since it lacks friendliness for the user”. Mainly these problems with the system are causing losses in profits and merchandise shortages that only damage the business and disrupt the compliance of their main objectives as a business (full problem description later in the text). My client’s main objectives as a business are to never have a shortage of merchandise to avoid unhappy customers and to always have updated selling prices of their products to avoid loses in profits.

The problem with my client’s business as seen by him is “stressful”; the losses in profits and some shortages on merchandise always makes it hard for my client to keep him happy and his customers at all times. First the losses in profit happen some days when prices require an update or, as called by my client, “price updating” is required. Sometimes prices need to be updated to avoid a loss in profit, for example, cooper is a raw material that is sold by my client in diverse shapes or forms as required by the client. In the case of copper if it has a rise in its cost, prices must be changed, but copper is not the only raw material that changes its price daily. There are many raw materials that are used in my client’s products that are sold at the business that constantly change their cost. Therefore there is a daily process that is being done at the business due to this problem; a better description follows:

Here is a full description of each of the previously mentioned steps:

|  |  |
| --- | --- |
| Event | Description |
| Supplier arrives with invoice (Appendix A) at the business | Usually at 8:00 AM in the morning merchandise arrives at the business from one or several suppliers. |
| The business receives the invoice and revises all the merchandise and accepts the invoice | When the merchandise arrives my client’s employees need to check if all the merchandise that was ordered arrived in good state and that it was complete, after this the employees accept the invoice. |
| Merchandise is moved from the truck to the cellar. | Employees begin to move all the merchandise into the cellar behind the business. |
| Invoice is signed and copy is given to the employee (Merchandise is later paid) | When merchandise has being moved, my client signs the invoice and the copy is given to employee for him to proceed with the price updating. Merchandise is not paid at the moment; it is later paid through bank payment using the invoice. |
| Employee uses invoice for price updating in the system (Appendix B) | After receiving the copy of the invoice, an employee uses the invoice for the price updating by inputting he following data into the system:   * Invoice Number * Date * Supplier Number * For every product:   + Quantity   + Fabrication Number   + Product’s Description   + Product’s Cost |
| Invoice is archived | When price updating is finished, invoice is archived in folders. |

The information or data required for the system to work is the following:

|  |  |  |
| --- | --- | --- |
| Name | Description | Example (based on invoice in Appendix A) |
| Invoice Number | Identification number for every invoice that the business receives. | F39015 |
| Date | Date of merchandise arrival and price change | 10/01/2012 |
| Supplier Number | Number that identifies the supplier (printed list with identification numbers for suppliers in the business). | 07 |
| Quantity | Number of units that were received from the supplier. | 3 |
| Fabrication Number | Number given by the business for identification (product’s id number). | 7501206611654 |
| Product’s Description | (Automatic): Product description that was previously saved in the system. | R.T. 125 x 50 x 15.9 MM. IC4BPE SHELLAC VERDE CLASICA PIEDRA |
| Product’s Cost | Cost that is printed in the invoice. | 83.48 |

During this whole process of price updating there have been issues that my client has encountered many times already in the past and that is that the date must be written every day, which then is written wrong by some of his employees that ten creates a lot of difficulties when wanting to check some data with saved paperwork or invoices. Also product’s price has to be written every single time and employees input the prices wrong. Additionally when description is displayed it may be confusing since some descriptions are very similar between some products like screws since the only difference could be its diameter and mistakes are made sometimes too.

So there are lots of issues as seen by my client and he has previously tried to solve some of this issues by trying to make employees revise every thing that is written into the system twice, he told them to be very careful, he got them a calendar or the date writing, he also had two employees writing the data for them to check each others work but mistakes kept coming still since this is also a daily job. So one of his main problems is to reduce the human error caused by his employees.

Another main problem that my client has many times dealt with is merchandise shortage. At my client’s business they do not have a system that reports if there is about to be a shortage or if they are completely out of stock in some products so usually when there is a shortage of any product that an employee “notices”, they immediately report it so an order is placed for merchandise to arrive next week from the suppliers. Previously this was not a very big issue when all the data was managed in papers years ago since reports were received every day and the merchandise that was sold was removed from the books and if a product was coming to a shortage an order was made, but when the previously explained system was placed, it did not made the counting automatically every time a product is sold, but he needs to know when a shortage is about to happen. So the only way he has tried to solve this issue is by telling his employees to be careful with shortages and to report every time a product is about to come to a shortage which has worked but not as efficiently as he liked it would. Also he has tried to made an estimate of sells that are coming up and he would order in advance since he already knows the business works pretty well at every time in the year but having to much stock could also be expensive for my client. So a possible solution could be a printed list for making a revision during the day for shortages revision if needed.

Also in the client’s business obviously they sell their products. The current system that is implemented in the business is somewhat inefficient since it is complicated for the user to use in a fast way. As the client said, “It’s frustrating to see many clients wandering in the store waiting for their turn”. So then he decided to buy a numbered ticket roll where the clients would take a ticket and wait for their turn. So this make some change but not that much since the system still is not helping at all. Then the client said that even sometimes the employees mess up the sales and end up selling something way more cheaper and ten the business made a loss. Also the client noticed that most of his customers wont take the receipts and he thinks that is a waste of paper and of money too. This finally demonstrates that there are many issues in the store and trying to fix them is the client’s biggest desire.

**A2. Criteria for Success:**

* Improve price-updating process:
  + Implement log system to ensure a database for possible revision of this process if required by the user.
  + Optimize the process by making it easier for the user to complete by the use of simple questions to complete.
  + Implement automatic date inputting for eliminating the possibility of user mistyping.
* User Friendly:
  + Easy to add, remove, make lists and make price updates with simple path to follow to accomplish a certain task.
  + If an error is present, the user will be notified about his mistake so it can be corrected.
  + Avoid mistyping of prices to evade the loss of profits in the business by making the user input the product’s price twice.
* Inventory:
  + Implement shortage system section where the user will be notified if any product is currently out of stock.
  + Implement list printing method where the user can choose the group he wants the list of.
  + Give the user the capability to add or remove any desired product if there’s a need to include a new one or remove one.
* Selling system:
  + Optimize selling system:
    - Easier use for improving the efficiency while making a sale to avoid unhappy waiting customers.
  + Include a full description of products when making sale so the user can be sure that the desired product is being sold.
  + Implement receipt creation system where the client can decide if he/she will like to have the receipt printed when products are paid at cashier.
  + Implement reports system:
    - Give the user the capability to create a report that informs the user about the sales of the day.
    - Give the user the capability to create a report that informs the user about the sales of the week.

**A3. Prototype Solution**

“Store’s Name”

Buenos Días/Tardes

1. Inventario
2. Ventas
3. Cambios en Precios
4. Salir

Ferretería

Inventory

Sales

Price Updating

Exit

View List

Make Sale

Add/Remove Item

Daily Report

This is the opening menu for the program. It is written in spanish since my client’s main language is spanish.

**Feedback:** *“Me parece perfecto, es lo que necesito.”*

**Translation:** “*It seems perfect, just what I need.”*

1. Inventario

1. Ver Lista
2. Agregar/Remover Artículo

This is the menu for the inventory section.

**Feedback:** *“Esta bien pero creo que nos falto incluir un boton para avisos de productos que ya no tengo en el inventario, osea los que ya se me acabaron.”*

**Translation:** *“It looks good, but it seems we are missing a button for warnings for any products that I have out of stock, or that I have no more of this certain product.”*

Ver Lista

Archivo Creado

This only acknowledges the user that the list file has been created.

**Feedback:** *“Ok pero en donde me crea el archivo, es posible crearlo en el desktop para encontrarlo facilmente y luego imprimirlo; tambien si es posbile si puedo escoger el area de los productos que quiero ver la lista y no tener que imprimirla toda.”*

**Translation:** *“Ok but where is the file created, is it possible to create it in the desktop so it can be easy to find and then print it; also if its possible I would like to choose the area of products that I want to see in the list and not having to print the whole list.”*

Agregar/Remover Artículo

Agregar o Remover (A/R)

Agregar/Remover Artículo

Nombre del Artículo

# Codigo

Grupo

Costo

Cantidad

This window demonstrates the adding or removing any product from the inventory, it requires product name, product’s code number, group that it belongs to, like screws, pluming, etc., its cost and quantity.

**Feedback:** *“Muy bien parece que tiene todo lo que necesito ahi solo que cuando quiera remover un articulo no creo que necesite saber el precio o cuantos son solo lo quiero remover de la lista y creo que sería mucho mas facil si solo se escribe el codigo ya que cada product tiene un codigo unico.”*

**Translation:** *“Very good its seems it haves everything I need but when I want to remove a product I do not think that I need to know the price or quantity I just want to remove it from the list and I think that it would be a lot more easier if we only write the code since every product has its unique code.”*

1. Ventas

1. Hacer Venta
2. Reporte del Dia

This window shows the sales menu.

**Feedback:** *“Perfecto, me gustaria tener un reporte semanal tambien.”*

**Translation:** *“Perfect, I would also like a weekly report option.”*

Hacer Venta

# Codigo

Nombre del Producto

Costo

Cantidad

Otro?

This window shows the make a sale procedure. First code must be written, then product info appears, if its incorrect, the user writes an ‘n’ do the code can be written again, then it asks quantity and shows cost and the it asks if another product id going to be sold or not, if not total is showed.

**Feedback:** *“Bien, me gusto la opcion que me da si quiero cancelar el prodcuto por si me equivoco, muy bien.”*

**Translation:** *“Good, I liked the option for cancelling a mistake I made, very good.”*

Reporte del Dia

Archivo Creado

This only acknowledges the user that the list file has been created.

**Feedback:** *“Bien solo que me gustaria al igual que el otro archivo que se cree el archivo en el desktop para no tener que buscarlo.”*

**Translation:** *“Good but I would like the file to be created in the desktop as the other file so I can find it easily.”*

Cambios en Precios

# de Factura

Fecha

Numero de Proovedor

Cantidad

# Codigo

Costo

This window shows how the price changing procedure works, first invoice number is needed, then date which is automatically written, then supplier number, then quantity, then the code number and finally the cost will be written twice to avoid mistakes

**Feedback*:*** *“Bueno aqui primero me gustaria que al escribir el numero del provedoor que tengo en la lista me aparezca en pantalla para asegurarme que es tal provedoor, luego al escribir el codigo me gustaria ver la informacion del producto para saber que ese es el producto y luego el costo me gusto la idea de tener que escribirlo dos veces para evitar errores.”*

**Translation:** *“Well first I would like to write the supplier number that I have in my list and then to see on screen his name to ensure that its him, then when the code is written I would like to see the products info to ensure its that product and finally I liked the idea of writing the cost twice to avoid mistakes.”*

**B1. Data Structures**

The program my client requires will mainly manage inventory work and management and sales, this will demand many different data structures for it to work in the most proficient way due to its very extensive inventory. An array would be the best option to store the entire inventory since an array can hold many values in a single list so it will make it really easy to manipulate the inventory. This array will need to be an Object array that will hold the diverse data for every product in the inventory. As previously mentioned, every Object will have to hold all of the product’s data given that every product has unique and diverse data (different data types) and Object are the only data type that can hold many different data types. The different data types that the array will hold are the following:

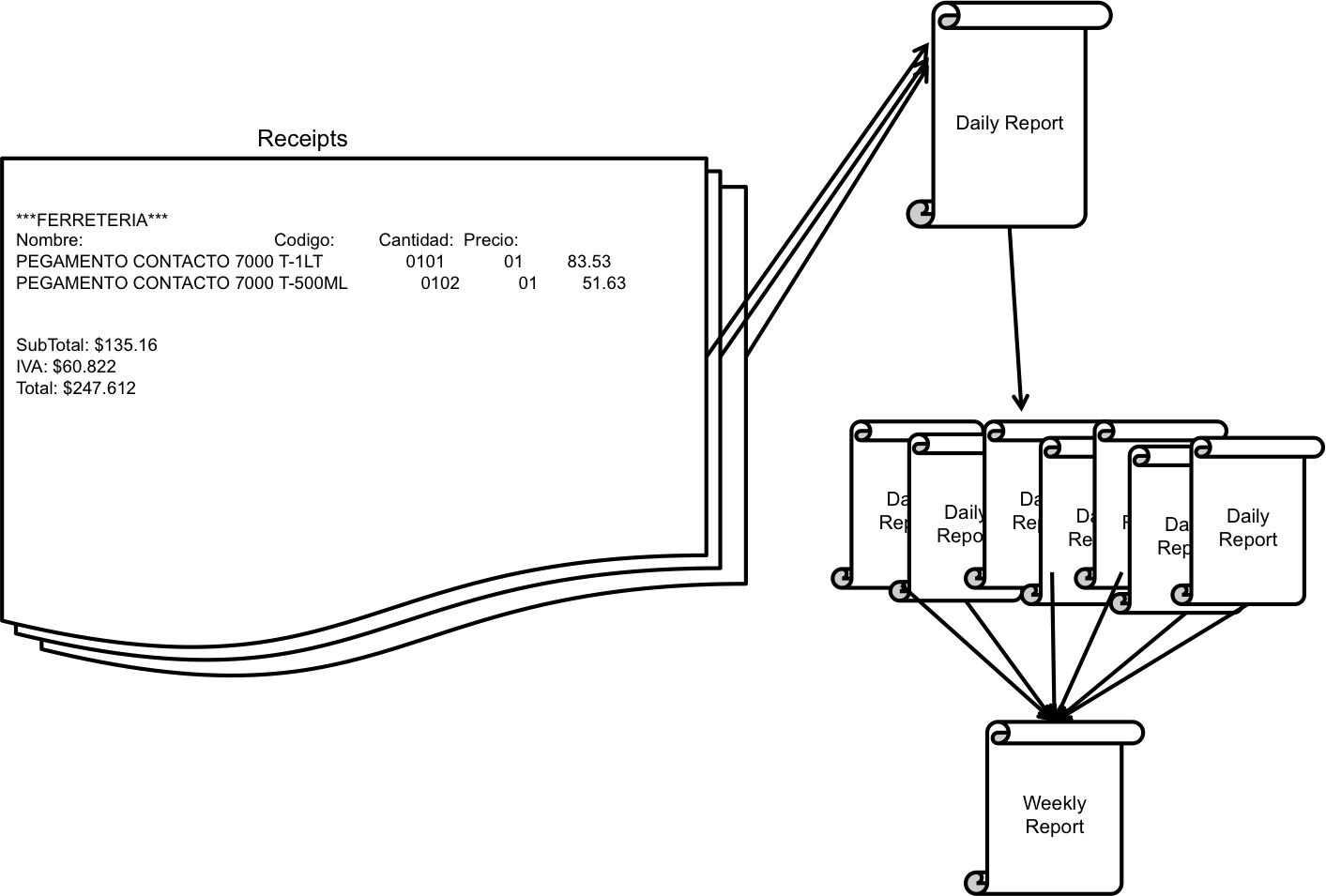
|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Name | Discussion | Sample Data |
| String | group | The group’s name that the product belongs to. String was the most suitable option since the name is pure text and numbers and the only data type that can hold a line of text is String. | PZA |
| String | productName | The product’s name. String is the best option since it's the only capable of holding a line of text in one variable. | PEGAMENTO CONTACTO 7000 T-1LT |
| int | pQuantity | The amount of this product that is in stock. The best data type that could be used in this case is int, since the quantity is only a whole number and there can’t be parts of product sold, the best option is int rather than double that handles decimal numbers that will not be used here. | 1 |
| String | pFabricationNumber | The number or code for a given product in the inventory. The most suitable option for this will be a String since some of the codes contain letters and numbers, rather than only numbers that could be handled using int. | 008FM |
| String | pDescription | The description of a certain product. This will definitely be a String since a description is a group of words that need to be hold in a variable and String is the only one capable of doing that. | TRUPER |
| double | pCost | The cost or what my client pays to the supplier for the product. This variable must be a double since the value that is being hold is a money amount and money is used with decimals rater than a whole number so the best option is double. | 441.6 |
| double | pSell | The selling price of the product. This variable must be also a double since it is also money that will be handled here and no whole numbers so also the best option is to use a double. | 552.0 |

 All of this previously mentioned data must be saved in a File for saving changes in inventory. A Binary File, which will be called completeInventory, would be the most fitting choice for this situation thanks to its ability to easily save a certain data type (array) in it since it transforms data into binary data and not literally as the text file would require only Strings. A Binary File would convert the data more efficiently for the program.

For the sales module the previously mentioned data structures will be used in this module. In this module the user will be capable of making a sale and also create the reports that were requested by my client. Some of the other main data structures that will be used are the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Name | Discussion | Sample Data |
| double | subTotal | The total amount before taxes. This variable will need to be a double and not an int because it is handling money number and these numbers need to be handled with decimal in case they are needed no whole numbers. | 135.16 |
| double | totalCost | The total amount after taxes. This variable also will need to be a double and not an int since decimals will be used | 247.612 |
| double | tax | The total amount that that needs to be paid for taxes. This will also be a double and not an int since decimals are required. | 60.822 |
| double | cashFlow | The sum of the total revenue of sales. This needs to be a double since it is the continuous sum of the previous variable totalCost that is a double, so if the previous variable is a double this one must be a double and not an int. | 247.612 |
| double | profit | The sum of the difference between price and cost of products. This variable must also be a double since previously price (pSell) and cost (pCost) were stated as doubles. So the data type needs to be the same and an int wont work since it's a different data type and we require decimals. | 15.2 |
| int | pSold | The total amount of products sold. This variable will be an int since the value that will be used is a whole number and decimal numbers will not be required so it will no be a double. | 4 |
| File | rDiario | Text file that will display the day’s revenue, number of products sold and profit. This file needs to be a text file since the user needs to see what is in the file and if the user requires a printed version the file must be text and not binary numbers from a binary file where the user cannot read the file. |  |
| File | rSemana | Text file that will display the week’s revenue, number of products sold and profit. As the previous file, this file also need to be a text file so the user can use this file for reading or for printing a copy if needed. |  |

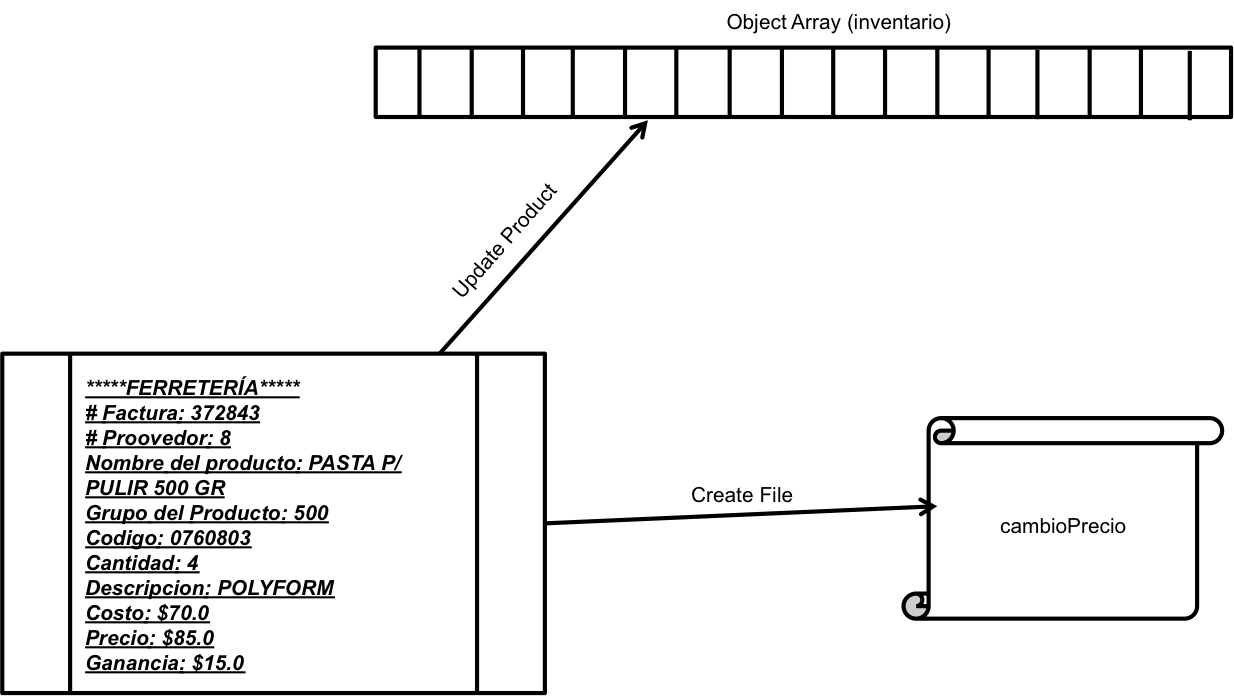
Next is an illustration of basically how the sales module works. The main process is making a sale, then getting the electronic receipts that will be used to create the daily reports with the stored data in the previously mentioned arrays and finally all of this data will then be used for the weekly reports.



Furthermore the Price Updating division will require some new data structures, which are the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Name | Discussion | Sample Data |
| int | invoiceNumber | The number that identifies every invoice. This variable needs to be an int since all of the invoice numbers are only big whole numbers, no decimals. This means that it will not be a double but an int. | 000010023020434343230 |
| int | supplierNumber | The number that identifies the supplier. This variable will also be an int since the supplier numbers are only whole numbers and no decimals; therefore it cannot be a double. | 12 |
| String | date | The date. This variable will be a String since the date is a combination of numbers and characters and the String can handle both in one variable. | 22.11.12 |
| File | cambioPrecio | Text file that displays all the information regarding the Price Updating process. This file needs to be a text because the user must be able to read it and a binary file cannot be read so a text file is the best choice. |  |

Also this is an illustration that basically explains what is being done in this section. First data is gathered for the system then to create the report and updating the inventory.

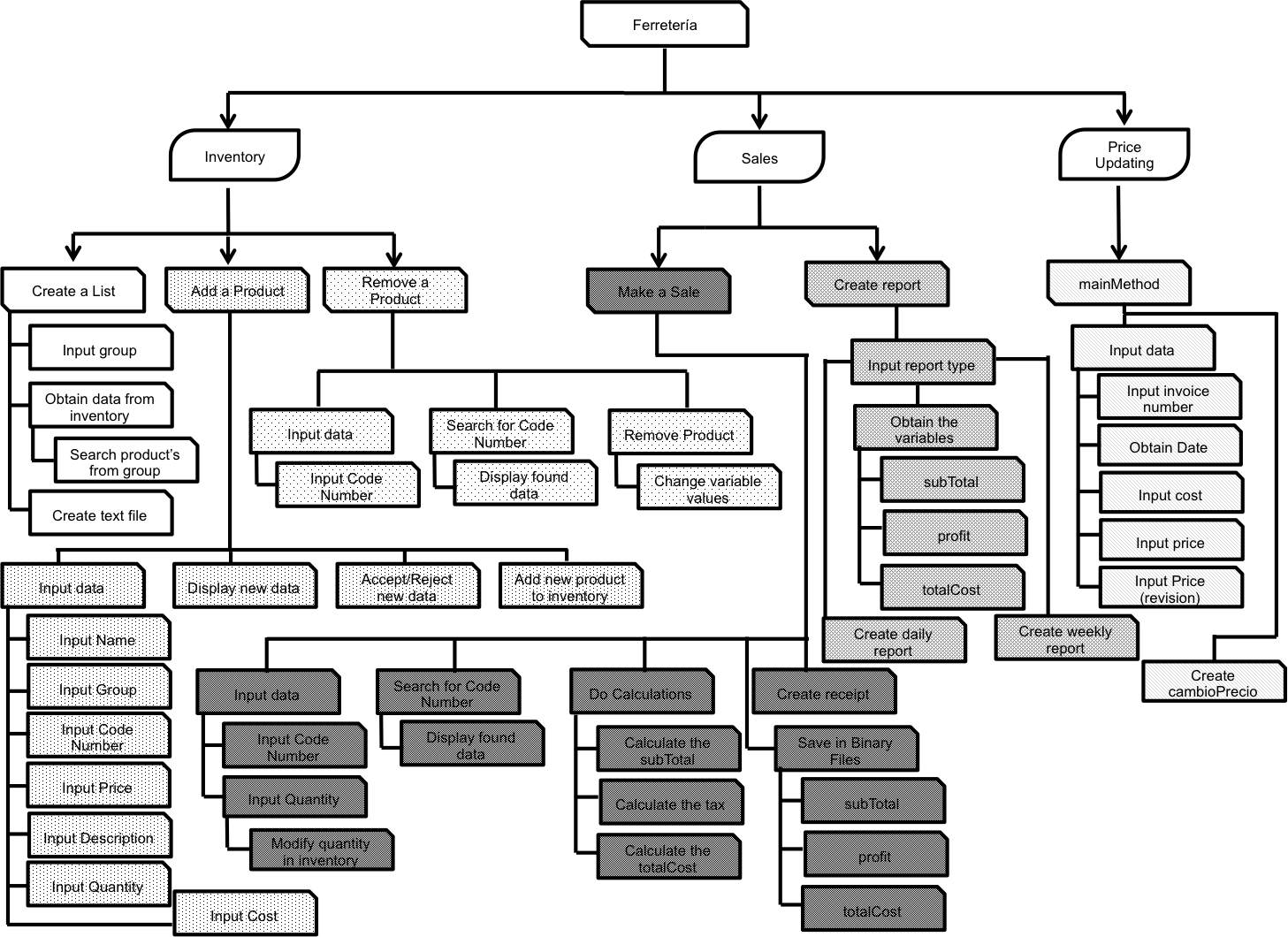


The following are the previously mentioned data types and other important data types:

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Name | Brief Description | Sample Data |
| Application Class | ferreteria.java | Main archive where all the code is saved. |  |
| Object Class | product.java | Archive where code for object is saved. |  |
| Product[] | inventory[] | Array that holds all the product’s data |  |
| String | group | The group’s name that the product belongs to. | PZA |
| String | productName | The product’s name. | PEGAMENTO CONTACTO 7000 T-1LT |
| int | pQuantity | The amount of this product that is in stock. | 1 |
| String | pFabricationNumber | The number or code for a given product in the inventory. | 008FM |
| String | pDescription | The description of a certain product. | TRUPER |
| double | pCost | The cost or what my client pays to the supplier for the product. | 441.6 |
| double | pSell | The selling price of the product. | 552.0 |
| double | subTotal | The total amount before taxes. | 135.16 |
| double | totalCost | The total amount after taxes. | 247.612 |
| double | tax | The total amount that that needs to be paid for taxes. | 60.822 |
| double | cashFlow | The sum of the total revenue of sales. | 247.612 |
| double | profit | The sum of the difference between price and cost of products. | 15.2 |
| int | pSold | The total amount of products sold. | 4 |
| File | rDiario | Text file that will display the day’s revenue, number of products sold and profit. |  |
| File | rSemana | Text file that will display the week’s revenue, number of products sold and profit. |  |
| int | invoiceNumber | The number that identifies every invoice. | 000010023020434343230 |
| int | supplierNumber | The number that identifies the supplier. | 12 |
| String | date | The date. | 22.11.12 |
| File | cambioPrecio | Text file that displays all the information regarding the Price Updating process. |  |
| boolean | exit | Decision that defines to exit or not the program. | True |
| int | choice | The number the user chose in a certain menu. | 2 |
| char | answer | The letter the user chose to anwer yes or no questions | Y |
| File | textInventory | Excel file that holds all of the inventory data |  |
| String | menu | The different menus that are displayed for the user | "\*\*\*\*VENTAS\*\*\*\*";  1) Hacer Venta |
| int [] | spaces[] | Array that holds amount of spaces for creating a nice and clean text file that contains several columns that need to be organized. | spaces[0] = |
| int [] | days[] | Array that holds day numbers. | days[0] = 22 |
| File | salesData1 | Binary file that contains the data for the variable cashFlow. |  |
| File | salesData2 | Binary file that contains the data for the variable profit. |  |
| File | salesData3 | Binary file that contains the data for the variable pSold. |  |
| File | daysReport | Binary file that contains the data for the array days[]. |  |
| int | month | The current month. | 11 |
| int | year | The current year. | 12 |
| int | day | The current day. | 22 |

**B2. Algorithms:**

|  |  |
| --- | --- |
| **Name:** createList | |
| **Description**: This method creates a Text file with the entire product’s from the groups specified by the user. | |
| **Parameters**: spaces[], inventory[] | **Return Value**: No return values |
| 1. Input desired group. 2. Obtain necessary data from inventory[]. 3. Create Text File using spaces[] to arrange text. | |
| **Name:** add | |
| **Description**: This method adds a product to the inventory. | |
| **Parameters:** inventory[], spaces[] | **Return Value**: inventory[] |
| 1. Input desired group. 2. Input the product’s name. 3. Input product’s code number. 4. Input product’s quantity. 5. Input product’s description. 6. Input product’s cost. 7. Input product’s price. 8. Input product’s price again. 9. If prices are the same. 10. If prices are the same.  * Continue with step 12.  1. Else  * Repeat step 8.  1. Display gathered information from the user using spaces[] to arrange the text. 2. Answer if displayed data is correct. 3. If data is correct  * Continue with step 16.  1. Else  * Repeat step 1.  1. Save new product data in inventory[]. 2. Return inventory[]. | |
| **Name:** remove | |
| **Description:** This method removes a product to the inventory. | |
| **Parameters:** inventory[], spaces[] | **Return Value**: inventory[] |
| 1. Input product’s code. 2. If found  * Continue with step 4.  1. Else  * Repeat step 1.  1. Display data using spaces[] to arrange text. 2. Answer if displayed data is correct. 3. If data is correct  * Define productName as “ZZZZZ” * Define pDescription as “ZZZZZ” * Define group as “ZZZZZ” * Define pFabricationNumber as “ZZZZZ” * Define pCost as 0 * Define pSell as 0 * Define pQuantity as 0 * Replace old product’s data with previous variables.  1. If data is incorrect  * Answer if redoing the process would be in order * If yes * Repeat step 1 * If not * Continue with step 7.  1. Return inventory[]. | |
| **Name:** notAvailable | |
| **Description:** This method displays any product that is currently out of stock. | |
| **Parameters:** inventory[], spaces[] | **Return Value**: No return values |
| 1. Search for products that have quantity in 0. 2. Display found products. | |
| **Name:** makeSale | |
| **Description:** This method makes a sale and creates a corresponding receipt. | |
| **Parameters:** inventory[], spaces[], salesData1, salesData2, salesData3 | **Return Value**: No return values |
| 1. Input code number. 2. Search for product. 3. If found  * Continue with step 5.  1. Else  * Repeat step 1.  1. Display product’s data using spaces[] to organize text. 2. Input Quantity. 3. Modify the product’s quantity in inventory. 4. Calculate the subTotal. 5. Calculate the tax. 6. Calculate the totalCost. 7. Create text file with used data (receipt). 8. Add totalCost to cashFlow. 9. Add the difference of pSell and pCost to profit. 10. Add the quantity to pSold. 11. Save cashFlow, profit and pSell in different Binary Files. | |
| **Name:** reports | |
| **Description:** This method controls the creation of daily and weekly sales in the business. | |
| **Parameters:** inventory[], spaces[], daysReport, cashFlow, profit, pSold | **Return Value**: No return values |
| 1. Input type of report D/W (daily or weekly) 2. If input is D  * Obtain cashFlow, profit and pSold. * Create daily report Text File.  1. If input is W  * Obtain cashFlow, profit and pSold. * Create weekly report Text File. | |
| **Name:** priceUpdating | |
| **Description:** This method is capable of changing/updating prices when it is needed (when merchandise is received) | |
| **Parameters:** inventory[] | **Return Value**: No return values |
| 1. Input invoiceNumber. 2. Obtain date. 3. Input supplierNumber. 4. Input product’s code number. 5. If found  * Display product’s data. * Input new quantity. * Input cost. * Input price. * Input price again * If price is not equal * Repeat step (“Input Price”). * Create cambioPrecio File.  1. Else  * Repeat step 1. | |

**B3. Modular Organization**

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Description | Input/Output | Data Structures |
| Inventory | In this module the basic processes that are being done are for the management of the inventory. A list can be created, a product can be added to an inventory and also a product could be removed. | Input:  inventory[]  Output:  list | inventory[]  productName  pQuantity  pFabricationNumber  pDescription  pCost  pSell  pQuantity  pFaabricationNumber |
| Sales | In this module all of the sales management is controlled including the reports data is controlled from here. | Input:  inventory[]  Output:  receipt  dailyReport  weeklyReport | subtotal  totalCost  tax  cashFlow  profit  pSold  rDiario  rSemana |
| Price Updating | In this module the “Price Updating” process is done and managed. It contains a series of easy steps for its completion. | Input:  inventory[]  Output:  Price Update Report | invoiceNumber  supplierNumber  date  cambioPrecio |

**Code Listing**

1. import java.io.\*;
2. import java.util.\*;
3. class Ferreteria
4. /\*\* Program that manages inventory and sales of a hardware store.
5. \* Author: Mauro Amarante Esparza
6. \* School: Tecnologico de Monterrey-Campus Santa Catarina
7. \* Candidate Number:dvs580
8. \* Date: November 2012
9. \*\*/
10. {
11. /\* This method controls all of the actions that are being done in the program. This method's main task is to read
12. and write the inventory file for the program to work and to distribute the work based on the user's will. \*/
13. public static void main (String [] args) throws IOException
14. {
15. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
16. PrintWriter stdOut = new PrintWriter(System.out, true);
18. int choice = 0;
19. boolean exit = false;
20. boolean approved = false;
21. int i = 1;
22. char answer = 'X';
24. File completeInventory = new File("Inventario.dat");
25. File textInventory = new File("Inventario FINAL.txt");
27. /\*The size of the array was stablished by my client who said "Currently I have many products but I don't think
28. that I will never pass the 40000 mark".\*/
29. Product [] inventory = new Product [40000];
31. //Invenotry file creation in case it does not exists.
33. if(!completeInventory.exists()){
34. ObjectOutputStream outFile = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream
35. (completeInventory)));
36. inventory = inventoryTextToSet(textInventory, inventory); //Method that converts Text File data to array.
37. outFile.writeObject(inventory);
38. outFile.close();
39. }
41. //Reading Inventory Binary File
43. else{
44. ObjectInputStream inFile = new ObjectInputStream(new BufferedInputStream(new FileInputStream
45. (completeInventory)));
47. try{
48. inventory = (Product [])inFile.readObject();
49. }
50. catch(ClassNotFoundException e){
51. stdOut.println("Error");
52. }
53. catch(EOFException e){
54. stdOut.println("Archivo Leido");
55. }
56. inFile.close();
57. }
59. //Array sorting, for easier manipulation of the array
61. inventory = arraySort(inventory);
63. //Menu
65. do{
66. ferreteriaMenu();
68. try{
69. stdOut.println("Escoger una opción"); //"Choose an option"
70. choice = Integer.parseInt(stdIn.readLine());
71. }catch(NumberFormatException e){
72. choice = 0;
73. }
75. switch(choice)
76. {
77. case 1:
78. {
79. inventory = manageInventory(inventory);
80. break;
81. }
82. case 2:
83. {
84. sales(inventory);
85. break;
86. }
87. case 3:
88. {
89. priceUpdating(inventory);
90. break;
91. }
92. case 4:
93. {
94. do{
95. stdOut.println("No se te olvide crear el reporte del dia. Deseas apagar el Sistema? S/N");
96. //"Do not forget to create the daily report. Do you wish to shut down you system? Y/N"
97. answer = stdIn.readLine().toUpperCase().charAt(0);
99. //Saving inventory in Binary File
101. if(answer == 'S'){
102. ObjectOutputStream outFile = new ObjectOutputStream(new BufferedOutputStream
103. (new FileOutputStream(completeInventory)));
104. outFile.writeObject(inventory);
105. outFile.close();
106. approved = true;
107. exit = true;
108. }
110. else if(answer == 'N'){
111. approved = true;
112. exit = false;
113. }
115. else
116. approved = false;
118. }while(!approved);
119. break;
120. }
121. default:
122. stdOut.println("Opción Invalida" + '\n'); //"Invalid Option"
123. }
124. }while(!exit);
125. }



130. /\*In this method, the inventory is the main data type used. Here you can manipulate the inventory to create a list
131. of a certain group in the inventory or you can add a product in the inventory or even remove one if required by
132. the user\*/
133. public static Product[] manageInventory(Product inventory [])throws IOException
134. {
135. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
136. PrintWriter stdOut = new PrintWriter(System.out, true);
138. String menu = "";
139. int choice = 0;
140. boolean exit = false;
141. boolean pass = true;
142. int spaces [] = new int [7];
144. /\*Method that returns an array with the number of spaces that need to be added to the words of different sizes for
145. creating an organized text.\*/
147. spaces = spaces(inventory);
149. //Inventory Menu
151. do{
152. stdOut.println('\n' + "\*\*\*\*INVENTARIO\*\*\*\*");
154. menu = mInventoryInitialMenu();
155. stdOut.println(menu);
157. try{
158. stdOut.println("Escoja alguna opción"); //"Choose an option"
159. choice = Integer.parseInt(stdIn.readLine());
160. }catch(NumberFormatException e){
161. choice = 0;
162. }
164. switch(choice)
165. {
166. case 1: //Create List
167. {
168. createList(spaces, inventory);
169. break;
170. }

173. case 2: //Add/Remove Menu
174. {
175. do{
176. try{
177. menu = addRemoveMenu();
178. stdOut.println(menu);
180. stdOut.println("Escoger una opción"); //"Choose an option"
181. choice = Integer.parseInt(stdIn.readLine());
182. pass = true;
183. }
184. catch(NumberFormatException e){
185. pass = false;
186. stdOut.println("Opción Invalida");
187. }
188. }while(!pass);
190. switch (choice)
191. {
192. case 1: //Add
193. {
194. inventory = add(inventory, spaces);
195. inventory = arraySort(inventory);
196. break;
197. }
199. case 2: //Remove
200. {
201. inventory = remove(inventory, spaces);
202. break;
203. }
205. case 3: //Cancel
206. {
207. stdOut.println("Cancelado");
208. break;
209. }
211. default:
212. stdOut.println("Opción Invalida");
213. }
214. break;
215. }
217. case 3: //Out of stock
218. {
219. notAvailable(spaces, inventory);
220. break;
221. }
223. case 4: //Cancel
224. {
225. stdOut.println("Cancelado");
226. exit = true;
227. break;
228. }
229. default:
230. stdOut.println("Opción Invalida");
231. }
232. }while(!exit);
233. return inventory;
234. }



239. /\*Method where the user can create a list of the products from the group that he chose from the menu or even all the
240. products in the invenotry\*/
241. public static void createList(int spaces[], Product inventory[]) throws IOException
242. {
243. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
244. PrintWriter stdOut = new PrintWriter(System.out, true);
246. int i = 0;
247. int j = 0;
248. int groupNum = 0;
249. String menu = "";
250. String name = "Nombre:";
251. String code = "Codigo:";
252. String quantity = "Cantidad:";
253. String description = "Descripción:";
254. String cost = "Costo:";
255. String group = "Grupo:";
256. String sell = "Precio:";
257. String cGroupNumber = ""; //Chosen group Number
258. String wGroupNumber = ""; //Wanted group Number
259. String allGroups = "TODOS";
260. boolean pass = true;
261. boolean listNotMade = false;
262. String [] list = new String [40000]; //Array where chosen products will be hold
264. menu = mInventoryDisplayGroups();
265. stdOut.println(menu);
267. //Preparing spaces required based on the longest String in th inventory of every section.
268. do{
269. for(i = name.length(); i < spaces[0]; i++)
270. name = name + " ";
272. for(i = code.length(); i < spaces[1]; i++)
273. code = code + " ";
275. for(i = quantity.length(); i < spaces[2]; i++)
276. quantity = quantity + " ";
278. for(i = description.length(); i < spaces[3]; i++)
279. description = description + " ";
281. for(i = cost.length(); i < spaces[4]; i++)
282. cost = cost + " ";
284. for(i = group.length(); i < spaces[5]; i++)
285. group = group + " ";
287. for(i = sell.length(); i < spaces[6]; i++)
288. sell = sell + " ";
290. do{
291. try{
292. stdOut.println("Crear lista del grupo:"); //"Create a list from the group:"
293. groupNum = Integer.parseInt(stdIn.readLine());
294. pass = false;
295. cGroupNumber = defineGroup(groupNum);
296. if(groupNum >= 0 && groupNum <= 25)
297. pass = true;
298. }
299. catch(NumberFormatException e){
300. pass = false;
301. stdOut.println("Opción Invalida");
302. }
303. }while(!pass);
305. j = 0;
306. i = 0;
308. do{
309. if(inventory[i] != null){
310. wGroupNumber = inventory[i].getGroup();
312. if(wGroupNumber.equals(cGroupNumber)){
313. list[j] = inventory[i].toString(spaces);
314. j++;
315. }
316. }
317. i++;
318. }while(inventory.length > i);
320. j = 0;
321. i = 0;
323. try{
324. //All groups list
325. while(inventory.length > j && cGroupNumber.equals(allGroups)){
326. list[j] = inventory[i].toString(spaces);
327. j++;
328. i++;
329. }
330. }catch(NullPointerException e){
331. }
333. //List creation
335. try{
336. PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter("lista.txt", false)));
337. outFile.println(name + code + quantity + description + cost + group + sell);
339. i = 0;
341. while(list.length > i){
342. if(list[i] != null)
343. outFile.println(list[i]);
344. i++;
345. }
347. outFile.close();
348. }catch(NullPointerException e){
349. listNotMade = false;
350. }
352. if(!listNotMade)
353. stdOut.println("La lista ha sido creada"); //"The list has been created"
355. else if(listNotMade)
356. stdOut.println("Error al crear lista"); //"Error while creating the list"
358. else
359. stdOut.println("Opción Invalida");
361. }while(listNotMade);
362. }



367. /\*This method adds a product to the inventory.\*/
368. public static Product[] add(Product inventory [], int spaces [])throws IOException
369. {
370. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
371. PrintWriter stdOut = new PrintWriter(System.out, true);
373. int groupNum = 0;
374. int pQuantity = 0;
375. int i = 0;
376. String menu = "";
377. String group = "";
378. String productName = "";
379. String pDescription = "";
380. String pFabricationNumber = "";
381. double pCost = 0;
382. double pSell2 = 0;
383. double pSell = 0;
384. boolean pass = true;
385. boolean correct = true;
386. boolean flag = true;
387. char revision = 'X';
389. //Group's List
390. do{
391. menu = mInventoryDisplayGroups();
392. stdOut.println(menu);
394. do{
395. try{
396. stdOut.println("Agregar producto al Grupo?"); //"Create a list from the group:"
397. groupNum = Integer.parseInt(stdIn.readLine());
398. group = defineGroup(groupNum);
399. pass = false;
400. if(groupNum >= 0 && groupNum <= 25)
401. pass = true;
402. }
403. catch(NumberFormatException e){
404. pass = false;
405. stdOut.println("Opción Invalida");
406. }
407. }while(!pass);
409. do{
410. stdOut.println("Nombre del Producto:"); //"Product's Name:"
411. productName = stdIn.readLine();
412. pass = true;
413. try{
414. for(i = 0; i < inventory.length; i++){
415. if(productName.equals(inventory[i].getProductName())){
416. pass = false;
417. stdOut.println("Nombre ya Existe"); //"Name already exists"
418. }
419. }
420. }catch(NullPointerException e){
421. }
422. if(productName.equals("")){
423. pass = false;
424. stdOut.println("Opción Invalida");
425. }
426. }while(!pass);
428. do{
429. stdOut.println("Codigo de Fabricacion:"); //"Fabrication Number:"
430. pFabricationNumber = stdIn.readLine();
431. pass = true;
432. try{
433. for(i = 0; i < inventory.length; i++){
434. if(pFabricationNumber.equals(inventory[i].getPFabricationNumber())){
435. pass = false;
436. stdOut.println("Codigo ya Existe"); //"Code already exists"
437. }
438. }
439. }catch(NullPointerException e){
440. }
441. if(pFabricationNumber.equals("")){
442. pass = false;
443. stdOut.println("Opción Invalida");
444. }
445. }while(!pass);
447. do{
448. try{
449. stdOut.println("Cantidad:"); //"Quantity:"
450. pQuantity = Integer.parseInt(stdIn.readLine());
451. pass = true;
452. }
453. catch(NumberFormatException e){
454. pass = false;
455. stdOut.println("Opción Invalida");
456. }
457. }while(!pass);
459. do{
460. stdOut.println("Descripción del Producto:"); //"Product's Description:"
461. pDescription = stdIn.readLine();
462. pass = true;
464. if(pDescription.equals("")){
465. pass = false;
466. stdOut.println("Opción Invalida");
467. }
468. }while(!pass);
469. do{
470. do{
471. try{
472. stdOut.println("Costo del Producto:"); //"Product's Cost"
473. pCost = Double.parseDouble(stdIn.readLine());
474. pass = true;
475. }
476. catch(NumberFormatException e){
477. pass = false;
478. stdOut.println("Opción Invalida");
479. }
480. }while(!pass);
482. do{
483. try{
484. do{
485. stdOut.println("Precio del Producto:"); //"Product's Price"
486. pSell = Double.parseDouble(stdIn.readLine());
488. stdOut.println("Vuelva a escribir el Precio del Producto: (verificación)");
489. //"Write the product's price again: verification)"
490. pSell2 = Double.parseDouble(stdIn.readLine());
492. if(pSell == pSell2)
493. correct = false;
495. else
496. stdOut.println("Error, precios diferenes"); //"Error, diferent prices"
498. }while(correct);
499. pass = true;
500. }
501. catch(NumberFormatException e){
502. pass = false;
503. stdOut.println("Opción Invalida");
504. }
505. }while(!pass);
507. flag = true;
509. if(pSell < pCost){
510. stdOut.println("Costo es mayor a precio de venta!"); //"Cost is higher than selling price!"
511. flag = false;
512. }
513. }while(!flag);
515. //Display
517. stdOut.println("Nombre del Producto: " + productName); //"Product's Name:"
518. stdOut.println("Codigo de Fabricacion: " + pFabricationNumber); //Fabrication Number:"
519. stdOut.println("Cantidad: " + pQuantity); //"Quantity:"
520. stdOut.println("Descripcion: " + pDescription); //"Description:"
521. stdOut.println("Costo: " + pCost); //"Cost:"
522. stdOut.println("Precio de Venta: " + pSell); //"Selling price:"
524. do{
525. do{
526. try{
527. stdOut.println("Esta todo Correcto?: S/N (Si/No)"); //"Is everything correct?: Y/N (Yes/No)"
528. revision = stdIn.readLine().toUpperCase().charAt(0);
529. pass = true;
530. }
531. catch(StringIndexOutOfBoundsException e){
532. pass = false;
533. stdOut.println("Opción Invalida");
534. }
535. }while(!pass);
537. if(revision == 'S'){
538. flag = false;
539. pass = true;
540. }
542. else if(revision == 'N'){
543. flag = true;
544. pass = true;
545. stdOut.println("Información incorrecta"); //"Incorrect information"
546. }
548. else{
549. flag = true;
550. pass = false;
551. stdOut.println("Opción Invalida");
552. }
554. }while(!pass);
555. }while(flag);

558. inventory[inventory.length-1] = new Product(group, productName, pFabricationNumber, pQuantity, pDescription,
559. pCost, pSell);
561. stdOut.println("Producto Agregado"); //"Product Added"
563. return inventory;
564. }



569. /\*This method removes a product to the inventory.\*/
570. public static Product [] remove(Product inventory[], int spaces []) throws IOException
571. {
572. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
573. PrintWriter stdOut = new PrintWriter(System.out, true);
575. int pQuantity = 0;
576. int i = 0;
577. String retrievedCode = "";
578. String searchCode = "";
579. String data = "";
580. String productName = "";
581. String pDescription = "";
582. String group = "";
583. String pFabricationNumber = "";
584. double pCost = 0;
585. double pSell = 0;
586. boolean pass = true;
587. boolean found = false;
588. boolean repeat = false;
589. boolean approved = false;
590. char revision = 'X';
592. do{
593. do{
594. try{
595. stdOut.println("Codigo del Producto:"); //"Product's Code:"
596. retrievedCode = stdIn.readLine();
598. //Product search
600. do{
601. searchCode = inventory[i].getPFabricationNumber();
602. if(searchCode.equals(retrievedCode)){
603. data = inventory[i].toString(spaces);
604. stdOut.println(data);
605. found = true;
606. pass = true;
607. }
608. i++;
609. }while(!found);
611. if(!found)
612. stdOut.println("Producto no encontrado"); //"Product not found"
614. } catch(NumberFormatException e){
615. pass = false;
616. stdOut.println("Opción Invalida");
617. }
618. catch(NullPointerException e){
619. stdOut.println("Lista Revisada"); //"List checked"
620. }
621. }while(!pass && !found);
623. if(found){
624. do{
625. do{
626. try{
627. stdOut.println("Es este el producto? S/N"); //"Is this the product? Y/N"
628. revision = stdIn.readLine().toUpperCase().charAt(0);
629. pass = true;
630. }
631. catch(NumberFormatException e){
632. pass = false;
633. stdOut.println("Opción Invalida");
634. }
635. }while(!pass);

638. if(revision == 'S'){
639. i--;
641. productName = "ZZZZZ";
642. pDescription = "ZZZZZ";
643. group = "ZZZZZ";
644. pFabricationNumber = "ZZZZ";
645. pCost = 0;
646. pSell = 0;
647. pQuantity = 0;
649. //Data replacement
651. inventory[i].setProductName(productName);
652. inventory[i].setGroup(group);
653. inventory[i].setQuantity(pQuantity);
654. inventory[i].setCost(pCost);
655. inventory[i].setPrice(pSell);
656. inventory[i].setFabricationNumber(pFabricationNumber);
657. inventory[i].setDescription(pDescription);
659. inventory = arraySort(inventory);
660. approved = true;
661. }
663. else if(revision == 'N'){
664. do{
665. do{
666. try{
667. stdOut.println("Buscar nuevamente un producto? S/N"); //"Search fo another product? Y/N"
668. revision = stdIn.readLine().toUpperCase().charAt(0);
669. pass = true;
670. approved = true;
671. }
672. catch(NumberFormatException e){
673. pass = false;
674. stdOut.println("Opción Invalida");
675. }
676. }while(!pass);
678. if(revision == 'S'){
679. repeat = true;
680. approved = true;
681. }
683. else if(revision =='N'){
684. repeat = false;
685. approved = true;
686. }
688. else{
689. repeat = false;
690. approved = false;
691. }
692. }while(!approved);
693. }
695. else{
696. stdOut.println();
697. approved = false;
698. }
699. }while(!approved);
700. }
702. else
703. stdOut.println("Producto no encontrado"); //"Product not found"
704. }while(repeat);
706. return inventory;
707. }

710. /\*This method displays any product that is currently out of stock.\*/
711. public static void notAvailable(int spaces[], Product inventory[])
712. {
713. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
714. PrintWriter stdOut = new PrintWriter(System.out, true);
716. int i = 0;
717. String name = "Nombre:";
718. String code = "Codigo:";
719. int quantity = 0;
720. String productName = "";
721. String sentence = "";
723. spaces = spaces(inventory);
725. for(i = name.length(); i < spaces[0]; i++)
726. name = name + " ";
728. for(i = code.length(); i < spaces[1]; i++)
729. code = code + " ";
731. stdOut.println("Productos no Disponibles:"); //"Unavailable products:"
732. stdOut.println(name + code);
734. //Product out of stock search
736. try{
737. for(i = 0; i < inventory.length; i++){
738. quantity = inventory[i].getPQuantity();
739. productName = inventory[i].getProductName();
740. if(quantity == 0 && productName != "ZZZZZ"){
741. sentence = inventory[i].toStringStock(spaces);
742. stdOut.println(sentence);
743. }
744. }
745. }catch(NullPointerException e){
746. stdOut.println("Lista Revisada"); //"List checked"
747. }
748. }

751. /\*In this method the sales module will be controlled. The user can make a sale or make a report\*/
752. public static void sales (Product inventory[])throws IOException
753. {
754. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
755. PrintWriter stdOut = new PrintWriter(System.out, true);
756. Calendar cal = new GregorianCalendar();
758. String menu = "";
759. boolean flag = true;
760. boolean pass = true;
761. int choice = 0;
762. double cashFlow = 0;
763. double profit = 0;
764. int pSold = 0;
765. int spaces [] = new int [7];
766. int days [] = new int [2];
767. File salesData1 = new File("salesData1.dat");
768. File salesData2 = new File("salesData2.dat");
769. File salesData3 = new File("salesData3.dat");
770. File daysReport = new File("daysReport.dat");
772. //Binary File craetion in case they dont exist
774. if(!salesData1.exists()){
775. ObjectOutputStream outFile = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream
776. (salesData1)));
777. outFile.writeObject(cashFlow);
778. outFile.close();
779. }
781. if(!salesData2.exists()){
782. ObjectOutputStream outFile = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream
783. (salesData2)));
784. outFile.writeObject(profit);
785. outFile.close();
786. }
788. if(!salesData3.exists()){
789. ObjectOutputStream outFile = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream
790. (salesData3)));
791. outFile.writeObject(pSold);
792. outFile.close();
793. }
795. if(!daysReport.exists()){
796. days[0] = cal.get(Calendar.DAY\_OF\_MONTH);
797. ObjectOutputStream outFile = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream
798. (daysReport)));
799. outFile.writeObject(days);
800. outFile.close();
801. }
803. else{
804. ObjectInputStream inFile = new ObjectInputStream(new BufferedInputStream(new FileInputStream
805. (daysReport)));
807. try{
808. days = (int [])inFile.readObject(); //Reading Binary File
809. }
810. catch(ClassNotFoundException e){
811. stdOut.println("Error");
812. }
813. catch(EOFException e){
814. stdOut.println("Archivo Leido");
815. }
816. inFile.close();
817. }
819. spaces = spaces(inventory);
821. //Sales menu
823. do{
824. menu = salesInitialMenu();
825. stdOut.println(menu);
827. do{
828. try{
829. stdOut.println("Escoja alguna opción"); //"Choose an option:"
830. choice = Integer.parseInt(stdIn.readLine());
831. pass = true;
832. }
833. catch(NumberFormatException e){
834. pass = false;
835. stdOut.println("Opción Invalida");
836. }
837. }while(!pass);

840. switch(choice)
841. {
842. case 1: //Make a sale
843. {
844. makeSale(inventory, spaces, salesData1, salesData2, salesData3);
845. break;
846. }
847. case 2: //Reports
848. {
849. //Read of binary files
851. ObjectInputStream inFile = new ObjectInputStream(new BufferedInputStream(new FileInputStream
852. (salesData1)));
853. try{
854. cashFlow = (double)inFile.readDouble(); //Reading Binary File
855. }
856. catch(EOFException e){
857. stdOut.println("Archivo Leido");
858. }
859. inFile.close();
861. ObjectInputStream inFile2 = new ObjectInputStream(new BufferedInputStream(new FileInputStream
862. (salesData2)));
863. try{
864. profit = (double)inFile2.readDouble(); //Reading Binary File
865. }
866. catch(EOFException e){
867. stdOut.println("Archivo Leido");
868. }
869. inFile.close();
871. ObjectInputStream inFile3 = new ObjectInputStream(new BufferedInputStream(new FileInputStream
872. (salesData3)));
873. try{
874. pSold = (int)inFile3.readInt(); //Reading Binary File
875. }
876. catch(EOFException e){
877. stdOut.println("Archivo Leido");
878. }
879. inFile.close();
881. reports(inventory, daysReport, cashFlow, profit, pSold);
882. break;
883. }
884. case 3: //Cancel
885. {
886. stdOut.println("Cancelado");
887. flag = false;
888. break;
889. }
890. default:
891. stdOut.println("Opción Invalida");
892. }
893. }while(flag);
894. }



899. /\*This method makes a sale and creates a corresponding receipt.\*/
900. public static void makeSale(Product inventory[], int spaces [], File salesData1, File salesData2, File salesData3)
901. throws IOException
902. {
903. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
904. PrintWriter stdOut = new PrintWriter(System.out, true);
905. Calendar cal = new GregorianCalendar();
907. int choice = 0;
908. int pQuantity = 0;
909. int quantity = 0;
910. int i = 0;
911. int chosen = 0;
912. int temp = 0;
913. int newQuantity = 0;
914. String name = "Nombre:";
915. String code = "Codigo:";
916. String quantity2 = "Cantidad:";
917. String sentence = "";
918. String pDescription = "";
919. String menu = "";
920. String group = "";
921. String productName = "";
922. String pFabricationNumber = "";
923. String sell = "Precio:";
924. char answer = 'A';
925. double pSell = 0;
926. double pCost = 0;
927. double subTotal = 0;
928. double tax = 1.6\*subTotal;
929. double totalCost = 0;
930. boolean found = false;
931. boolean pass = true;
932. boolean flag = true;
933. boolean approved = false;
934. double cashFlow = 0;
935. double profit = 0;
936. int pSold = 0;
938. //Spaces control
940. for(i = name.length(); i < spaces[0]; i++)
941. name = name + " ";
943. for(i = code.length(); i < spaces[1]; i++)
944. code = code + " ";
946. for(i = quantity2.length(); i < spaces[2]; i++)
947. quantity2 = quantity2 + " ";
949. for(i = sell.length(); i < spaces[6]; i++)
950. sell = sell + " ";
952. //Receipt creation
954. PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter("receipt.txt",false)));
955. outFile.println("\*\*\*FERRETERIA\*\*\*");
956. outFile.println(name +code + quantity2 + sell);
958. do{
959. do{
960. do{
961. try{
962. stdOut.println("Escribir Codigo:"); //"Write the code:"
963. pFabricationNumber = stdIn.readLine();
964. pass = true;
965. }
966. catch(NumberFormatException e){
967. pass = false;
968. stdOut.println("Opción Invalida");
969. }
970. }while(!pass);
972. //Product Search
974. try{
975. for(i = 0; i < inventory.length-1; i++){
976. if(pFabricationNumber.equals(inventory[i].getPFabricationNumber())){
977. found = true;
978. productName = inventory[i].getProductName();
979. pFabricationNumber = inventory[i].getPFabricationNumber();
980. pSell = inventory[i].getPSell();
981. pQuantity = inventory[i].getPQuantity();
982. pDescription = inventory[i].getPDescription();
984. stdOut.println("Nombre del Producto: " + productName);
985. stdOut.println("Codigo de Fabricacion: " + pFabricationNumber);
986. stdOut.println("Cantidad: " + pQuantity);
987. stdOut.println("Descripcion: " + pDescription);
988. stdOut.println("Precio de Venta: " + pSell);
989. chosen = i;
990. }
991. }
992. }catch(NullPointerException e){
993. stdOut.println("Termino revision");
994. pass = true;
995. }
996. if(!found)
997. stdOut.println("No se encontro producto");
998. }while(!found || !pass);

1001. if(found){
1002. do{
1003. try{
1004. stdOut.println("Cantidad:"); //"Quantity:"
1005. quantity = Integer.parseInt(stdIn.readLine());
1006. pass = true;
1007. }
1008. catch(NumberFormatException e){
1009. pass = false;
1010. stdOut.println("Opción Invalida");
1011. }
1012. }while(!pass);
1014. temp = inventory[chosen].getPQuantity();
1015. newQuantity = temp - quantity;
1017. if(newQuantity >= 0){
1018. inventory[chosen].setQuantity(newQuantity);
1019. pass = true;
1020. }
1022. else{
1023. stdOut.println("Ya no hay mas!"); //"There ar no more!"
1024. pass = false;
1025. }
1027. if(pass){
1028. sentence = inventory[chosen].toStringSell(spaces, quantity);
1029. outFile.println(sentence);
1030. }
1031. subTotal = subTotal + pSell;
1032. }
1034. do{
1035. do{
1036. try{
1037. stdOut.println("Otro Producto? S/N"); //"Another product? Y/N"
1038. answer = stdIn.readLine().toUpperCase().charAt(0);
1039. pass = true;
1040. }
1041. catch(NumberFormatException e){
1042. pass = false;
1043. stdOut.println("Opción Invalida");
1044. }
1045. }while(!pass);
1047. if(answer == 'S' || answer == 'N')
1048. approved = true;
1050. else
1051. approved = false;
1052. }while(!approved);
1054. tax = .45 \* subTotal;
1055. totalCost = tax + subTotal + pSell;
1057. cashFlow = cashFlow + totalCost;
1058. profit = profit + (pSell - pCost);
1059. pSold = pSold + quantity;
1060. }while(!found || answer == 'S');
1062. //Writing receipt
1064. outFile.println();
1065. outFile.println();
1066. outFile.println("SubTotal: $" + subTotal);
1067. outFile.println("IVA: $" + tax);
1068. outFile.println("Total: $" + totalCost);
1069. outFile.close();
1071. stdOut.println("SubTotal: $" + subTotal);
1072. stdOut.println("IVA: $" + tax);
1073. stdOut.println("Total: $" + totalCost);
1075. //Saving important variables in Binary Files
1077. ObjectOutputStream outFile2 = new ObjectOutputStream(new BufferedOutputStream
1078. (new FileOutputStream(salesData1)));
1079. outFile2.writeObject(cashFlow);
1080. outFile2.close();
1082. ObjectOutputStream outFile3 = new ObjectOutputStream(new BufferedOutputStream
1083. (new FileOutputStream(salesData2)));
1084. outFile3.writeObject(profit);
1085. outFile3.close();
1087. ObjectOutputStream outFile4 = new ObjectOutputStream(new BufferedOutputStream
1088. (new FileOutputStream(salesData3)));
1089. outFile4.writeObject(pSold);
1090. outFile4.close();
1091. }



1096. /\*This method controls the creation of daily and weekly sales in the business.\*/
1097. public static void reports (Product inventory [], File daysReport, double cashFlow, double profit, int pSold)
1098. throws IOException
1099. {
1100. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
1101. PrintWriter stdOut = new PrintWriter(System.out, true);
1102. Calendar cal = new GregorianCalendar();
1104. boolean pass = true;
1105. char type = 'X';
1106. int i = 0;
1107. int day = 0;
1108. int days [] = new int [2];
1110. do{
1111. try{
1112. stdOut.println("Cual tipo de Reporte desea, diario o semanal? D/S");
1113. //"Which type of report would you like, daily or weekly? D/W
1114. type = stdIn.readLine().toUpperCase().charAt(0);
1115. pass = true;
1116. }
1117. catch(StringIndexOutOfBoundsException e){
1118. pass = false;
1119. stdOut.println("Opción Invalida");
1120. }
1121. }while(!pass);
1123. switch(type)
1124. {
1125. case 'D':
1126. {
1127. day = cal.get(Calendar.DAY\_OF\_MONTH);
1128. if(days[0] != day){
1129. reportDay(cashFlow, pSold, profit);
1130. }
1131. break;
1132. }
1134. case 'S':
1135. {
1136. day = cal.get(Calendar.DAY\_OF\_MONTH);
1137. if(days[1] <= day){
1138. reportWeek(cashFlow, pSold, profit);
1139. }
1140. break;
1141. }
1143. default:
1144. stdOut.println("Opción Invalida");
1145. }
1147. //Saving binary file
1149. ObjectOutputStream outFile2 = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(daysReport)));
1150. outFile2.writeObject(days);
1151. outFile2.close();
1152. }



1157. /\*This method is capable of changing/updating prices when it is needed (when merchandise is received)\*/
1158. public static void priceUpdating(Product inventory[]) throws IOException
1159. {
1160. BufferedReader stdIn = new BufferedReader(new InputStreamReader (System.in));
1161. PrintWriter stdOut = new PrintWriter(System.out, true);
1163. int invoiceNumber = 0;
1164. int supplierNumber = 0;
1165. int i = 0;
1166. int quantity = 0;
1167. int pQuantity = 0;
1168. int chosen = 0;
1169. double pCost = 0;
1170. double pSell = 0;
1171. double pSell2 = 0;
1172. double profit = 0;
1173. String date = currentDate();
1174. String pFabricationNumber = "";
1175. String display = "";
1176. String productName = "";
1177. String pDescription = "";
1178. char answer = 'X';
1179. boolean found = false;
1180. boolean correct = false;
1181. boolean approved = true;
1182. boolean decision = true;
1183. boolean pass = true;
1184. File cambioPrecio = new File (date + ".txt");
1186. if(!cambioPrecio.exists()){
1187. PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter(cambioPrecio, true))) ;
1188. outFile.println("\*\*\*\*\*FERRETERÍA\*\*\*\*\*");
1189. outFile.close();
1190. }
1192. do{
1193. try{
1194. stdOut.println("Escribir numero de factura:"); //"Write the invoice number:"
1195. invoiceNumber = Integer.parseInt(stdIn.readLine());
1196. pass = true;
1197. }
1198. catch(StringIndexOutOfBoundsException e){
1199. pass = false;
1200. stdOut.println("Opción Invalida");
1201. }
1202. }while(!pass);
1204. stdOut.println("Fecha:" + date); //"Date"
1206. do{
1207. try{
1208. stdOut.println("Escribir numero de proovedor:"); //"Supplier number:"
1209. supplierNumber = Integer.parseInt(stdIn.readLine());
1210. pass = true;
1211. }
1212. catch(StringIndexOutOfBoundsException e){
1213. pass = false;
1214. stdOut.println("Opción Invalida");
1215. }
1216. }while(!pass);
1218. do{
1219. do{
1220. try{
1221. stdOut.println("Escribir codigo del producto"); //"Write th product's code"
1222. pFabricationNumber = stdIn.readLine();
1223. pass = true;
1224. }
1225. catch(StringIndexOutOfBoundsException e){
1226. pass = false;
1227. stdOut.println("Opción Invalida");
1228. }
1229. }while(!pass);
1231. try{
1232. do{
1233. for(i = 0; i < inventory.length-1; i = i + 1){
1234. if(pFabricationNumber.equals(inventory[i].getPFabricationNumber())){
1235. found = true;
1236. chosen = i;
1237. }
1238. }
1239. }while(!found);
1240. }catch(NullPointerException e){
1241. stdOut.println("Lista Revisada");
1242. }
1244. //Product Search
1246. if(found){
1247. productName = inventory[chosen].getProductName();
1248. pFabricationNumber = inventory[chosen].getPFabricationNumber();
1249. pSell = inventory[chosen].getPSell();
1250. pQuantity = inventory[chosen].getPQuantity();
1251. pDescription = inventory[chosen].getPDescription();
1252. stdOut.println("Nombre del Producto: " + productName);
1253. stdOut.println("Codigo de Fabricacion: " + pFabricationNumber);
1254. stdOut.println("Cantidad: " + pQuantity);
1255. stdOut.println("Descripcion: " + pDescription);
1256. stdOut.println("Precio de Venta: " + pSell);
1257. }
1259. do{
1260. try{
1261. stdOut.println("Nueva Cantidad:"); //"New quantity:"
1262. quantity = Integer.parseInt(stdIn.readLine());
1263. pass = true;
1264. }
1265. catch(NumberFormatException e){
1266. pass = false;
1267. stdOut.println("Opción Invalida");
1268. }
1269. }while(!pass);
1271. do{
1272. try{
1273. stdOut.println("Costo del Producto:"); //"Product's Cost"
1274. pCost = Double.parseDouble(stdIn.readLine());
1275. pass = true;
1276. }
1277. catch(NumberFormatException e){
1278. pass = false;
1279. stdOut.println("Opción Invalida");
1280. }
1281. }while(!pass);
1283. do{
1284. try{
1285. do{
1286. stdOut.println("Precio del Producto:"); //"Product's Price:"
1287. pSell = Double.parseDouble(stdIn.readLine());
1289. stdOut.println("Vuelva a escribir el Precio del Producto: (verificacion)");
1290. //"Write the product's price again: verification)"
1291. pSell2 = Double.parseDouble(stdIn.readLine());
1293. if(pSell == pSell2)
1294. correct = true;
1296. else
1297. stdOut.println("Error, costos diferenes"); //"Error, diferent prices"
1298. }while(!correct);
1299. pass = true;
1300. }
1301. catch(StringIndexOutOfBoundsException e){
1302. pass = false;
1303. stdOut.println("Opción Invalida");
1304. }
1305. }while(!pass);
1307. //Modyfing Product
1309. inventory[chosen].setAddQuantity(quantity);
1310. inventory[chosen].setCost(pCost);
1311. inventory[chosen].setPrice(pSell);
1313. profit = pSell - pCost;
1315. //Report creation
1317. PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter(cambioPrecio,true)));
1318. outFile.println("# Factura: " + invoiceNumber);
1319. outFile.println("# Proovedor: " + supplierNumber);
1320. outFile.println("Nombre del producto: " + productName);
1321. outFile.println("Grupo del Producto: " + inventory[chosen].getGroup());
1322. outFile.println("Codigo: " + inventory[chosen].getPFabricationNumber());
1323. outFile.println("Cantidad: " + quantity);
1324. outFile.println("Descripcion: " + inventory[chosen].getPDescription());
1325. outFile.println("Costo: $" + pCost);
1326. outFile.println("Precio: $" + pSell);
1327. outFile.println("Ganancia: $" + profit);
1328. outFile.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");
1330. do{
1331. stdOut.println("Desea modificar algun otro producto? S/N"); //"Do you wish to modify any other product? Y/N"
1332. answer = stdIn.readLine().toUpperCase().charAt(0);
1334. if(answer == 'S'){
1335. decision = true;
1336. approved = true;
1337. }
1339. else if(answer == 'N'){
1340. decision = false;
1341. approved = true;
1342. }
1344. else{
1345. decision = false;
1346. approved = false;
1347. stdOut.println("Opción Invalida");
1348. }
1349. }while(!approved);
1350. outFile.close();
1351. }while(decision);
1352. }



1357. /\*Method that returns the program's main menu\*/
1358. public static void ferreteriaMenu()
1359. {
1360. PrintWriter stdOut = new PrintWriter(System.out, true);
1362. stdOut.println("\*\*\*\*\*FERRETERÍA\*\*\*\*\*");
1363. stdOut.println("1) Inventario"); //Inventory
1364. stdOut.println("2) Ventas"); //Sales
1365. stdOut.println("3) Cambio en Precio"); //Change in Price
1366. stdOut.println("4) Cerrar Programa"); //Close program
1367. }



1372. /\*Method that sors the array according to their fabrication numbers(codes)\*/
1373. public static Product [] arraySort(Product inventory [])
1374. {
1375. PrintWriter stdOut = new PrintWriter(System.out, true);
1376. Product [] temp = new Product [1];
1377. boolean change = true;
1378. int result = 0;
1380. try{
1381. while(change){
1382. change = false;
1383. for (int i = 1; i < inventory.length; i ++){
1384. result = inventory[i - 1].getPFabricationNumber().compareToIgnoreCase(inventory[i].getPFabricationNumber());
1385. if(result > 0){
1386. temp[0] = inventory[i];
1387. inventory[i] = inventory[i -1];
1388. inventory[i -1] = temp[0];
1389. change = true;
1390. }
1392. else{
1393. temp[0] = inventory[i];
1394. inventory[i - 1] = inventory[i];
1395. inventory[i] = temp[0];
1396. change = true;
1397. }
1398. }
1399. }
1400. }catch(NullPointerException e){
1401. stdOut.print("Lista Reorganizada");
1402. }
1404. return inventory;
1405. }



1410. /\*Method that converts the .txt file into an array of Objects\*/
1411. public static Product [] inventoryTextToSet(File textInventory, Product inventory[]) throws IOException
1412. {
1413. BufferedReader readFile = new BufferedReader (new FileReader (textInventory));
1414. PrintWriter stdOut = new PrintWriter(System.out, true);
1416. String line = readFile.readLine();
1417. StringTokenizer tokenLine;
1418. File temp;
1419. String preGroup;
1420. String group;
1421. String productName;
1422. int pQuantity;
1423. String pFabricationNumber;
1424. String pDescription;
1425. double pCost;
1426. double pSell;
1428. int j = 0;
1429. try{
1430. while(line != null){
1431. tokenLine = new StringTokenizer(line, "\t");
1432. while(tokenLine.hasMoreTokens()){
1433. pFabricationNumber = tokenLine.nextToken();
1434. productName = tokenLine.nextToken();
1435. pDescription = tokenLine.nextToken();
1436. pCost = Double.parseDouble(tokenLine.nextToken());
1437. group = tokenLine.nextToken();
1438. pQuantity = Integer.parseInt(tokenLine.nextToken());
1439. pSell = Double.parseDouble(tokenLine.nextToken());
1440. inventory[j] = new Product(group, productName, pFabricationNumber, pQuantity, pDescription, pCost, pSell);
1441. j++;
1442. }
1443. line = readFile.readLine();
1444. }
1445. }catch(FileNotFoundException e){
1446. stdOut.println("Error");
1447. }
1448. catch(EOFException e){
1449. stdOut.print("");
1450. }
1451. catch(NoSuchElementException e){
1452. stdOut.println("");
1453. }
1454. readFile.close();
1455. return inventory;
1456. }



1461. /\*Method that returns the inventory menu\*/
1462. public static String mInventoryInitialMenu()
1463. {
1464. String sentence1 = "1) Crear Lista de Inventario";
1465. String sentence2 = "2) Agregar/Remover Producto";
1466. String sentence3 = "3) No disponibles";
1467. String sentence4 = "4) Regresar";
1469. String statement = sentence1 + '\n' + sentence2 + '\n' + sentence3 + '\n' + sentence4;
1471. return statement;
1472. }



1477. /\*Method that returns the list of the available groups in th inventory\*/
1478. public static String mInventoryDisplayGroups()
1479. {
1480. String sentence1 = "Grupos Disponibles:";
1481. String sentence2 = "1) 500";
1482. String sentence3 = "2) 1 K";
1483. String sentence4 = "3) A PZ";
1484. String sentence5 = "4) BOL";
1485. String sentence6 = "5) BTO";
1486. String sentence7 = "6) CAJ";
1487. String sentence8 = "7) CUB";
1488. String sentence9 = "8) GAL";
1489. String sentence10 = "9) JGO";
1490. String sentence11 = "10) KGS";
1491. String sentence12 = "11) KIT";
1492. String sentence13 = "12) LTO";
1493. String sentence14 = "13) MEI";
1494. String sentence15 = "14) MTO";
1495. String sentence16 = "15) PAQ";
1496. String sentence17 = "16) PAR";
1497. String sentence18 = "17) POR";
1498. String sentence19 = "18) PZA";
1499. String sentence20 = "19) QTO";
1500. String sentence21 = "20) ROL";
1501. String sentence22 = "21) TBO";
1502. String sentence23 = "22) TMO";
1503. String sentence24 = "23) TRA";
1504. String sentence25 = "24) ZA P";
1505. String sentence26 = "25) TODOS";
1507. String statement = ('\n' + sentence1 + '\n' + sentence2 + '\n' + sentence3 + '\n' + sentence4 + '\n' + sentence5 + '\n' +
1508. sentence6 + '\n' + sentence7 + '\n' + sentence8 + '\n' + sentence9 + '\n' + sentence10 + '\n' +
1509. sentence11 + '\n' + sentence12 + '\n' + sentence13 + '\n' + sentence14 + '\n' + sentence15 +
1510. '\n' + sentence16 + '\n' + sentence17 + '\n' + sentence18 + '\n' + sentence19 + '\n' +
1511. sentence20 + '\n' + sentence21 + '\n' + sentence22 + '\n' + sentence23 + '\n' + sentence24 +
1512. '\n' + sentence25 + '\n' + sentence26);
1514. return statement;
1515. }



1520. /\*Method that based on the group's list it returns the name of a given group\*/
1521. public static String defineGroup(int number)
1522. {
1523. PrintWriter stdOut = new PrintWriter(System.out, true);
1524. String choice = "";
1526. switch(number)
1527. {
1528. case 1:
1529. {
1530. choice = "500";
1531. break;
1532. }
1533. case 2:
1534. {
1535. choice = "1 K";
1536. break;
1537. }
1538. case 3:
1539. {
1540. choice = "A PZ";
1541. break;
1542. }
1543. case 4:
1544. {
1545. choice = "BOL";
1546. break;
1547. }
1548. case 5:
1549. {
1550. choice = "BTO";
1551. break;
1552. }
1553. case 6:
1554. {
1555. choice = "CAJ";
1556. break;
1557. }
1558. case 7:
1559. {
1560. choice = "CUB";
1561. break;
1562. }
1563. case 8:
1564. {
1565. choice = "GAL";
1566. break;
1567. }
1568. case 9:
1569. {
1570. choice = "JGO";
1571. break;
1572. }
1573. case 10:
1574. {
1575. choice = "KGS";
1576. break;
1577. }
1578. case 11:
1579. {
1580. choice = "KIT";
1581. break;
1582. }
1583. case 12:
1584. {
1585. choice = "LTO";
1586. break;
1587. }
1588. case 13:
1589. {
1590. choice = "MEI";
1591. break;
1592. }
1593. case 14:
1594. {
1595. choice = "MTO";
1596. break;
1597. }
1598. case 16:
1599. {
1600. choice = "PAQ";
1601. break;
1602. }
1603. case 17:
1604. {
1605. choice = "POR";
1606. break;
1607. }
1608. case 18:
1609. {
1610. choice = "PZA";
1611. break;
1612. }
1613. case 19:
1614. {
1615. choice = "QTO";
1616. break;
1617. }
1618. case 20:
1619. {
1620. choice = "ROL";
1621. break;
1622. }
1623. case 21:
1624. {
1625. choice = "TBO";
1626. break;
1627. }
1628. case 22:
1629. {
1630. choice = "TMO";
1631. break;
1632. }
1633. case 23:
1634. {
1635. choice = "TRA";
1636. break;
1637. }
1638. case 24:
1639. {
1640. choice = "ZA P";
1641. break;
1642. }
1643. case 25:
1644. {
1645. choice = "TODOS";
1646. break;
1647. }
1648. default:
1649. {
1650. stdOut.println("Opción Invalida");
1651. }
1652. }
1653. return choice;
1654. }



1659. /\*Method that returns an array with the number of spaces that need to be added to the words of different sizes for
1660. creating an organized text.\*/
1661. public static int[] spaces(Product inventory[])
1662. {
1663. int maxName = 9;
1664. int maxCode = 9;
1665. int maxQuantity = 11;
1666. int maxDescription = 14;
1667. int maxCost = 8;
1668. int maxGroup = 8;
1669. int maxSell = 9;
1670. int tempLong = 0;
1671. String temp = "";
1672. int spaces[] = new int [7];
1674. try{
1675. for(int i = 0; i < inventory.length; i++){
1676. temp = inventory[i].getProductName();
1677. tempLong = temp.length();
1678. if(tempLong > maxName)
1679. maxName = tempLong;
1680. }
1681. }catch(NullPointerException e){
1682. }
1684. try{
1685. for(int i = 0; i < inventory.length; i++){
1686. temp = inventory[i].getPFabricationNumber();
1687. tempLong = temp.length();
1688. if(tempLong > maxCode)
1689. maxCode = tempLong;
1690. }
1691. }catch(NullPointerException e){
1692. }
1694. try{
1695. for(int i = 0; i < inventory.length; i++){
1696. temp = "" + inventory[i].getPQuantity();
1697. tempLong = temp.length();
1698. if(tempLong > maxQuantity)
1699. maxQuantity = tempLong;
1700. }
1701. }catch(NullPointerException e){
1702. }
1704. try{
1705. for(int i = 0; i < inventory.length; i++){
1706. temp = inventory[i].getPDescription();
1707. tempLong = temp.length();
1708. if(tempLong > maxDescription)
1709. maxDescription = tempLong;
1710. }
1711. }catch(NullPointerException e){
1712. }
1714. try{
1715. for(int i = 0; i < inventory.length; i++){
1716. temp = "" + inventory[i].getPCost();
1717. tempLong = temp.length();
1718. if(tempLong > maxCost)
1719. maxCost = tempLong;
1720. }
1721. }catch(NullPointerException e){
1722. }
1724. try{
1725. for(int i = 0; i < inventory.length; i++){
1726. temp = inventory[i].getGroup();
1727. tempLong = temp.length();
1728. if(tempLong > maxGroup)
1729. maxGroup = tempLong;
1730. }
1731. }catch(NullPointerException e){
1732. }
1734. try{
1735. for(int i = 0; i < inventory.length; i++){
1736. temp = "" + inventory[i].getPSell();
1737. tempLong = temp.length();
1738. if(tempLong > maxSell)
1739. maxSell = tempLong;
1740. }
1741. }catch(NullPointerException e){
1742. }
1744. spaces[0] = maxName;
1745. spaces[1] = maxCode;
1746. spaces[2] = maxQuantity;
1747. spaces[3] = maxDescription;
1748. spaces[4] = maxCost;
1749. spaces[5] = maxGroup;
1750. spaces[6] = maxSell;
1752. return spaces;
1753. }



1758. /\*Method that creates the daily report\*/
1759. public static void reportDay(double salesAmount, int totalPSold, double totalProfit) throws IOException
1760. {
1761. File rDiario = new File("Reporte Diario.txt");
1762. PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter(rDiario,false)));
1763. outFile.println("\*\*\*\*\*FERRETERÍA\*\*\*\*\*");
1764. outFile.println("Ventas Totales: $" + salesAmount);
1765. outFile.println("Total Unidades Vendidas: " + totalPSold);
1766. outFile.println("Ganancia Total: $" + totalProfit);
1767. outFile.close();
1768. }



1773. /\*Method that creates the weekly report\*/
1774. public static void reportWeek(double salesAmount, int totalPSold, double totalProfit) throws IOException
1775. {
1776. File rSemana = new File("Reporte Semanal.txt");
1777. PrintWriter outFile = new PrintWriter(new BufferedWriter(new FileWriter(rSemana,false)));
1778. outFile.println("\*\*\*\*\*FERRETERÍA\*\*\*\*\*");
1779. outFile.println("Ventas Totales: $" + salesAmount);
1780. outFile.println("Total Unidades Vendidas: " + totalPSold);
1781. outFile.println("Ganancia Total: $" + totalProfit);
1782. outFile.close();
1783. }



1788. /\*Method that returns the add/remove menu\*/
1789. public static String addRemoveMenu()
1790. {
1791. String sentence1 = "\*\*\*\*AGREGAR/REMOVER PRODUCTO\*\*\*\*";
1792. String sentence2 = "1) Agregar Producto";
1793. String sentence3 = "2) Remover Producto";
1794. String sentence4 = "3) Cancelar";
1796. String statement = '\n' + sentence1 + '\n' + sentence2 + '\n' + sentence3 + '\n' + sentence4;
1798. return statement;
1799. }



1804. /\*Method that returns the sales initial menu\*/
1805. public static String salesInitialMenu()
1806. {
1807. String sentence1 = "\*\*\*\*VENTAS\*\*\*\*";
1808. String sentence2 = "1) Hacer Venta";
1809. String sentence3 = "2) Reportes";
1810. String sentence4 = "3) Cancelar";
1812. String statement = '\n' + sentence1 + '\n' + sentence2 + '\n' + sentence3 + '\n' + sentence4;
1813. return statement;
1814. }



1819. /\*Method that obtains the date and then returns it\*/
1820. public static String currentDate(){
1821. Calendar cal = new GregorianCalendar();
1822. int month = cal.get(Calendar.MONTH);
1823. int year = cal.get(Calendar.YEAR);
1824. int day = cal.get(Calendar.DAY\_OF\_MONTH);
1825. String date = (day + "." + (month + 1) + "." + year);
1826. return date;
1827. }

1828. }

1. import java.io.\*;
2. class Product implements Serializable
3. {
4. /\*Instance Variables\*/
5. static final long serialVersionUID = -649087714753943071L;
6. private String group;
7. private String productName;
8. private int pQuantity;
9. private String pFabricationNumber;
10. private String pDescription;
11. private double pCost;
12. private double pSell;

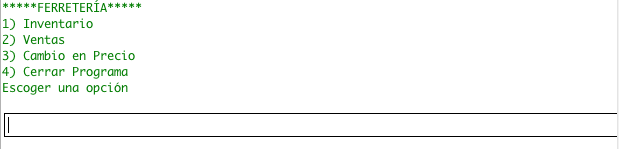

16. /\*Constructor\*/
17. public Product(String group, String productName, String pFabricationNumber, int pQuantity, String pDescription,
18. double pCost, double pSell)
19. {
20. this.group = group;
21. this.productName = productName;
22. this.pFabricationNumber = pFabricationNumber;
23. this.pQuantity = pQuantity;
24. this.pDescription = pDescription;
25. this.pCost = pCost;
26. this.pSell = pSell;
27. }
29. /\*Methods\*/
30. public void setGroup(String group)
31. {
32. this.group = group;
33. }
35. public void setProductName(String productName)
36. {
37. this.productName = productName;
38. }
40. public void setQuantity(int pQuantity)
41. {
42. this.pQuantity = pQuantity;
43. }
45. public void setAddQuantity(int addQuantity)
46. {
47. pQuantity = pQuantity + addQuantity;
48. }
50. public void setCost(double pCost)
51. {
52. this.pCost = pCost;
53. }
55. public void setPrice(double pSell)
56. {
57. this.pSell = pSell;
58. }
60. public void setFabricationNumber(String pFabricationNumber)
61. {
62. this.pFabricationNumber = pFabricationNumber;
63. }
65. public void setDescription(String pDescription)
66. {
67. this.pDescription = pDescription;
68. }
70. public String getGroup()
71. {
72. return group;
73. }
75. public String getProductName()
76. {
77. return productName;
78. }
80. public int getPQuantity()
81. {
82. return pQuantity;
83. }
85. public double getPCost()
86. {
87. return pCost;
88. }
90. public String getPFabricationNumber()
91. {
92. return pFabricationNumber;
93. }
95. public String getPDescription()
96. {
97. return pDescription;
98. }
100. public double getPSell()
101. {
102. return pSell;
103. }
105. public String toString(int spaces [])
106. {
107. String name = productName + " ";
108. String fabNumber = pFabricationNumber + " ";
109. String description = pDescription + " ";
110. String group2 = group;
111. String cost = "" + pCost;
112. String quantity = "" + pQuantity;
113. String sell = "" + pSell;
115. for(int i = name.length(); i < spaces[0]; i++)
116. name = name + " ";
118. for(int i = fabNumber.length(); i < spaces[1]; i++)
119. fabNumber = fabNumber + " ";
121. for(int i = quantity.length(); i < spaces[2]; i++)
122. quantity = quantity + " ";
124. for(int i = description.length(); i < spaces[3]; i++)
125. description = description + " ";
127. for(int i = cost.length(); i < spaces[4]; i++)
128. cost = cost + " ";
130. for(int i = group2.length(); i < spaces[5]; i++)
131. group2 = group2 + " ";
133. for(int i = sell.length(); i < spaces[6]; i++)
134. sell = sell + " ";
136. String sentence;
138. sentence = name + fabNumber + quantity + description + cost + group2 + sell;
140. return sentence;
141. }
143. public String toStringSell(int spaces [], int unitsSold)
144. {
145. String name = productName + " ";
146. String fabNumber = pFabricationNumber + " ";
147. String quantity = "" + pQuantity + unitsSold;
148. String sell = "" + pSell;
149. String sentence = "";
151. for(int i = name.length(); i < spaces[0]; i++)
152. name = name + " ";
154. for(int i = fabNumber.length(); i < spaces[1]; i++)
155. fabNumber = fabNumber + " ";
157. for(int i = quantity.length(); i < spaces[2]; i++)
158. quantity = quantity + " ";
160. for(int i = sell.length(); i < spaces[6]; i++)
161. sell = sell + " ";
163. sentence = name + fabNumber + quantity + sell;
165. return sentence;
166. }
168. public String toStringStock(int spaces[])
169. {
170. String name = productName + " ";
171. String fabNumber = pFabricationNumber + " ";
173. for(int i = name.length(); i < spaces[0]; i++)
174. name = name + " ";
176. for(int i = fabNumber.length(); i < spaces[1]; i++)
177. fabNumber = fabNumber + " ";
179. String sentence;
181. sentence = name + fabNumber;
183. return sentence;
184. }
185. }

**C2. Handling Errors**

|  |  |  |
| --- | --- | --- |
| Problem Description: | Fix: | Code: |
| The program requires reading a binary file that holds the array of the inventory that the program crucially needs for it to work. The problem appears when the binary file is not has not yet been created making the program to crash. | Using an if to check if the file already exits helps the program know if it requires to create the binary file for future use. | if(!completeInventory.exists()){  ObjectOutputStream outFile = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(completeInventory)));  inventory = inventoryTextToSet(textInventory, inventory);  outFile.writeObject(inventory);  outFile.close();  } |
| When the first menu is displayed it requests an answer if the chosen answer is to close the program so if the program is going to be closed all updated data would be deleted. | First, if the user tries to input an invalid offered option the switch will display that its an invalid option. Also when the exit option is selected the updated or changed inventory will be saved for next use. | case 4:  {  ObjectOutputStream outFile = newObjectOutputStream(new BufferedOutputStream(newFileOutputStream(completeInventr)));  outFile.writeObject(inventory);  outFile.close();  exit = true;  break;  }  default:  stdOut.println("Opción Invalida" + '\n');  }  }while(!exit); |
| When the program requires a user input the user can make mistakes of any kind, from clicking enter with no value or a letter when a number was required. | To solve this issue a try-catch is required so the program can “catch” a user error and make the input run again thanks to a loop. If an error is caught the program rejects the pass of the input and tells the user that its an invalid value. | do{  try{  menu = addRemoveMenu();  stdOut.println(menu);    stdOut.println("Escoger una opción");  choice = Integer.parseInt(stdIn.readLine());  pass = true;  }  catch(NumberFormatException e){  pass = false;  stdOut.println("Opción Invalida");  }  }while(!pass); |
| When the program is adding a new product and it asks for the fabrication number from the user there is the possibility that this number already exists or even an empty space could be inputted. | First, after the number has been entered the program will compare all possible numbers in the inventory until it finds one that is identical or until it has checked all the inventory array without the empty or null spaces, if a null space is found the error will be caught. If an identical number was found the loop will make the user to re-enter the number wanted also if the user inputted an empty space the loop will run again. | do{  stdOut.println("Codigo de Fabricacion:");  pFabricationNumber = stdIn.readLine();  pass = true;  try{  for(i = 0; i < inventory.length; i++){  if(pFabricationNumber.equals(  inventory[i].getPFabricationNumber())){  pass = false;  stdOut.println("Codigo ya Existe");  }  }  }catch(NullPointerException e){  }  if(pFabricationNumber.equals("")){  pass = false;  stdOut.println("Opción Invalida");  }  }while(!pass); |
| When adding a product into the inventory, the user must enter the price, but sometimes the price that was written is wrong. | To fix this issue the program requires the user to input the product’s price twice, if they are different the loop will repeat itself. | do{  stdOut.println("Precio del Producto:");  pSell = Double.parseDouble(stdIn.readLine());    stdOut.println("Vuelva a escribir el Precio del Producto: (verificacion)");  pSell2 = Double.parseDouble(stdIn.readLine());    if(pSell == pSell2)  correct = false;    else  stdOut.println("Error, costos diferenes");    }while(correct); |
| When adding a product into the inventory sometimes the user might confuse cost and price. | Usually the cost of the product is lower than the (selling) price of a product so an if condition will determine if the loop must repeat itself or accept the entered values. | flag = true;    if(pSell < pCost){  stdOut.println("Costo es mayor a precio de venta!");  flag = false  }  }while(!flag); |
| When making a sell of a certain product before checking its availability, the product might be out of stock. | When making a sell the program will get the defined quantity in stock and rest the desired quantity to be sold if the desired amount is higher than the available I repeats the loop. | temp = inventory[chosen].getPQuantity();  newQuantity = temp - quantity;    if(newQuantity >= 0){  inventory[chosen].setQuantity(newQuantity);  pass = true;  }    else{  stdOut.println("Ya no hay mas!");  pass = false;  } |

**D1. Annotated Hard Copy:**

When we first run the program the display will be the following:



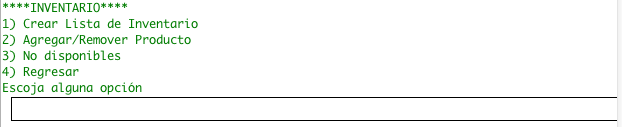
1) Inventory

2) Sales

3) Change in Price

4) Close Program

First we will go into the option called “Inventario” which means inventory by entering the number 1. In this section the inventory can be controlled inn many ways with the following options will be available:



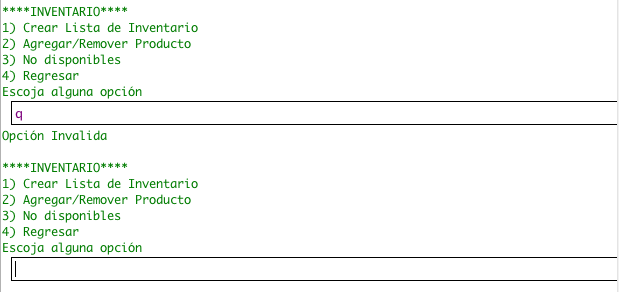
1) Create inventory List

2) Add/remove Product

3) Unavailable products

4) Go back

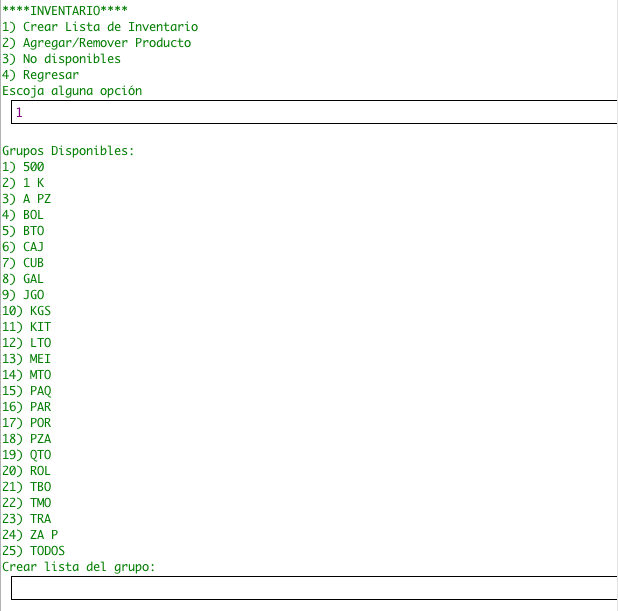
Here we can see that on the inventory section allows us to create and inventory list, add/remove a product, display not available products, or a go back option. In the create list option we can make the program create a list of a certain subgroup of the inventory or the whole inventory. We will now go to create a list section by entering 1:



INVALID VALUE

Same as previous menu

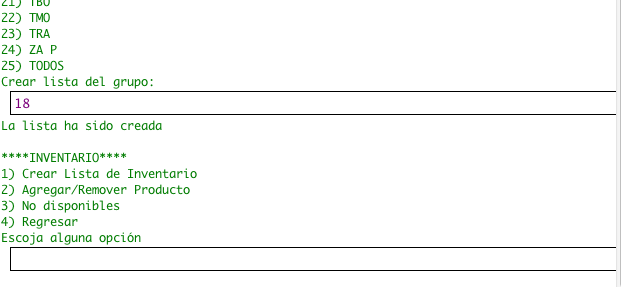
Accidentally the user entered an invalid value, he tried to input “q” but the program understood that it as an invalid option and it displayed the same previous menu options for the user to re-enter another correct value. Now we will go into the create a list section by entering 1:



Valid value

Inventory subgroups

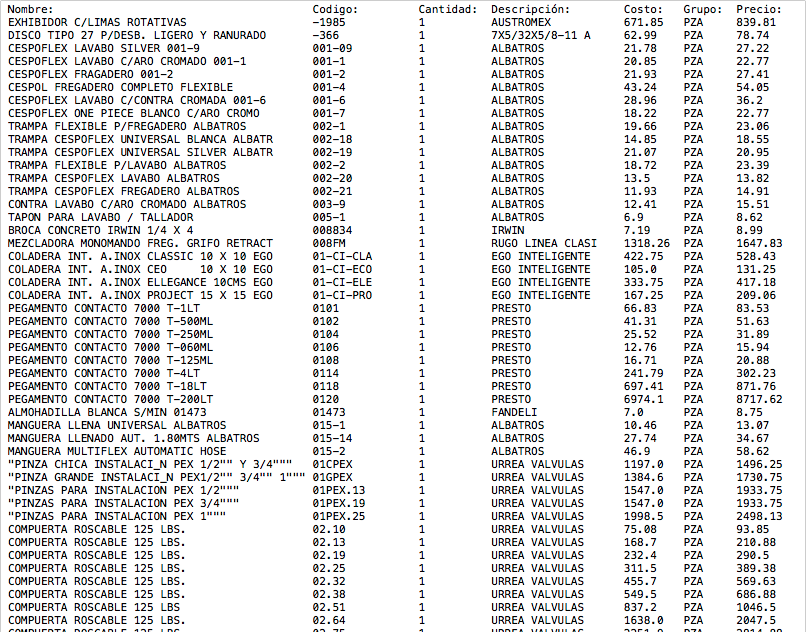
Now we have the option to choose which subgroup we want in our inventory list whish will show us the products that belong to that subgroup and the product’s information that will be included in the list is: name, code, quantity, description, cost, group and price.



Chosen group (PZA)

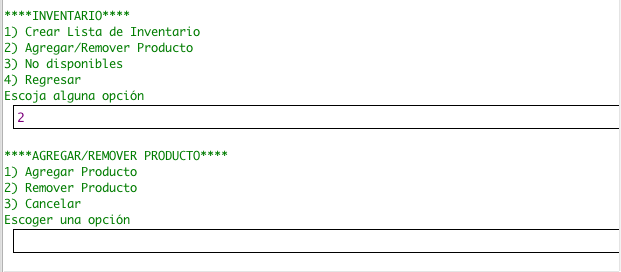
“List has been created”

When a list has been created it will look like the following:



Name Code Quantity Description Cost Group Price

As we can see the list has been created successfully with all the text arranged in columns for an easy reading for the user. Now lets continue on the program into the add/remove section by entering 2:

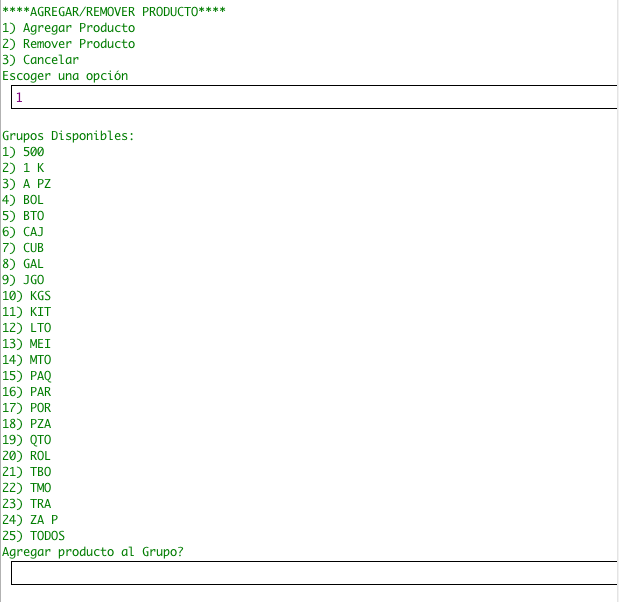


1) Add product

2) Remove Product

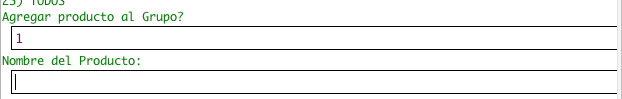
3) Go Back

In this section we have the options: add product, remove product and go back. Now lets add a new product into the inventory by entering 1:



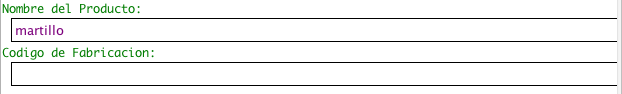
Inventory subgroups

Now for creating a new product we will be required to input a lot of information so the program can create the new product. First we have to choose a respective group for the new product. We will choose the group 1 (500):



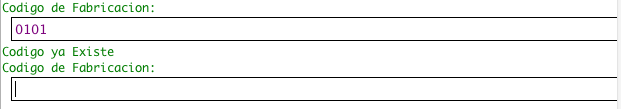
Product’s Name

Now we need to input the product’s name. Let’s call it “martillo” which means hammer:



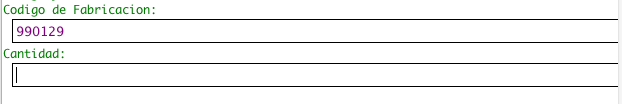
Product’s Code

Now the program requires a code for the new product and it can be any kind of code so let’s input 0101:



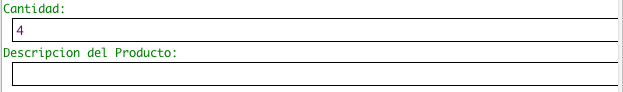
“Code already exists”

Immediately the program tells us that the code already exists! So we need to re-enter another code to see if it is available. Let’s input 990129:



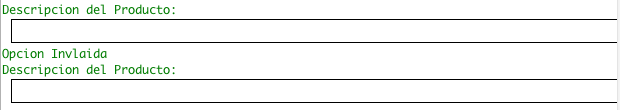
Product’s Quantity

Now as we can see the program has accepted the code and now is requesting for a quantity or the amount of the product that will be in the inventory. Let’s think we have 4 hammers:



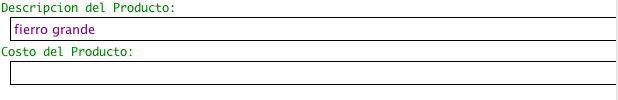
Product’s Description

After inputting the quantity, the program needs to know the description of the product. So lets write “fierro grande” which means “iron big”:



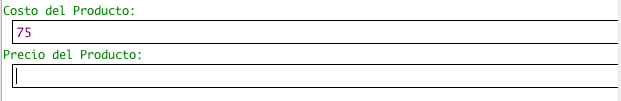
INVALID INPUT

Accidentally the user clicked the enter button but the program sees this as an error so it requires the description again:



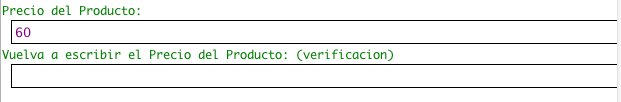
Product’s Cost

After entering the previously mentioned description, now are requested to write the cost of the product, the cost will be 75:



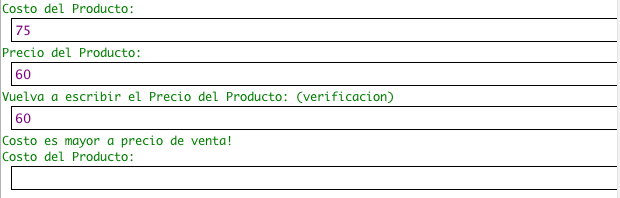
Product’s Price

Now we need to write in the price, let’s price it at 60:



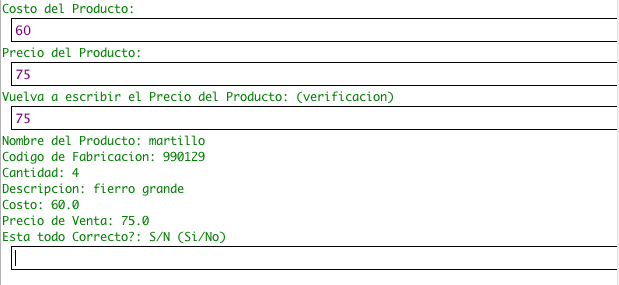
Product’s Price Verification

Immediately the program asks us to re-enter the price, this helps the user avoid mistakes when writing in the price of a product. Now let’s rewrite the price:



“Cost is greater than price!”

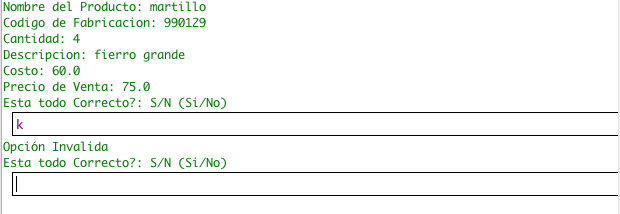
Now the program tells us that the cost is greater than the price! This means that maybe the user might have mixed up the cost and price since the selling price must always be higher than the product’s cost. Let’s renter all the previous data with the cost at 60 and the price at 75:



New Product’s Data

“Is everything correct?: Y/N (Yes/No)”

Now that the program has accepted our new cost and price it displays all of the previously inputted data for the user to check if there is any mistake. If here is no mistake we need to input the letter “s” which stands for yes in spanish, but if a mistake is spotted the letter “n” must be entered to restart the adding product process:



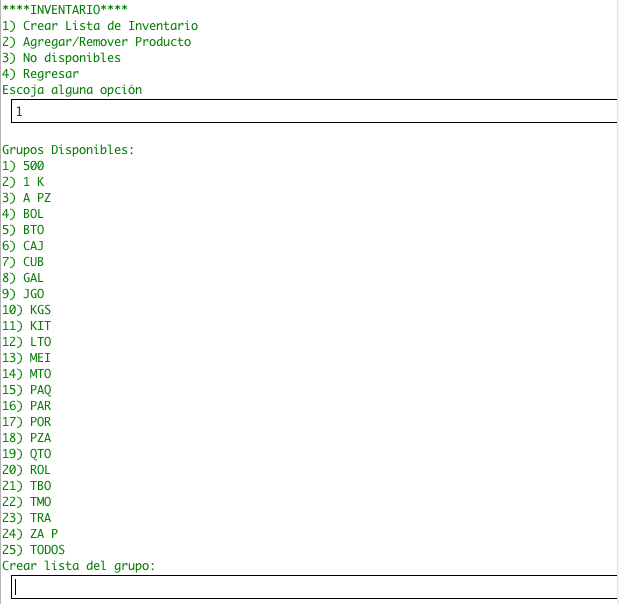
INVALID INPUT

Accidentally the user wrote an invalid letter and the program is requesting a valid input, let’s enter the letter “s” since it seems all correct:



“Product added”

Now that we have finished adding a product the program brought us into the inventory menu, let’s check in the list if our product was added by creating a list from the subgroup of the product (500):



Inventory subgroups

1) Create inventory List

2) Add/remove Product

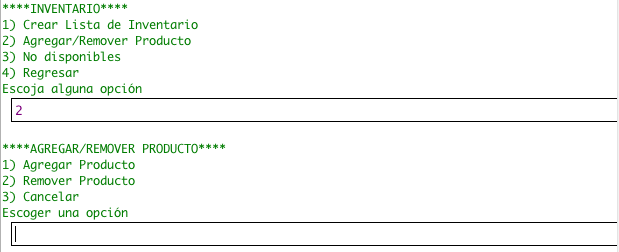
3) Unavailable products

4) Go back



Product is in the inventory

Now that we have successfully added a product in the inventory let’s remove the same product from the inventory by going into the add/remove product section by entering 2 in the inventory menu:



1) Add Product

2) Remove Product

3) Cancel

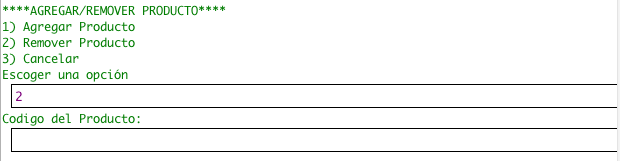
1) Create inventory List

2) Add/remove Product

3) Unavailable products

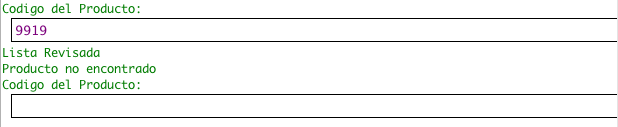
4) Go back

Now lets enter the remove product section by entering 2:



Product’s Code

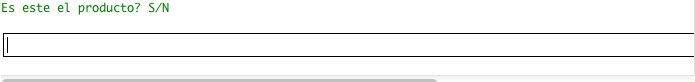
Now that we are in the remove product section the program requires the product’s code to find the desired product that will be removed from the inventory, our previously added product had the code “990129¨.



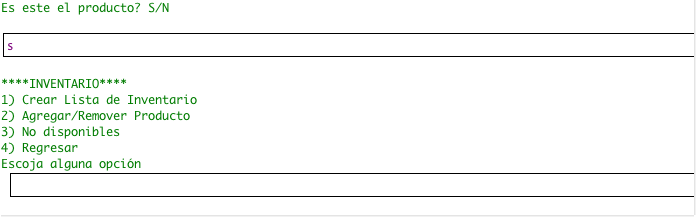
“List Revised”

“Product not found”

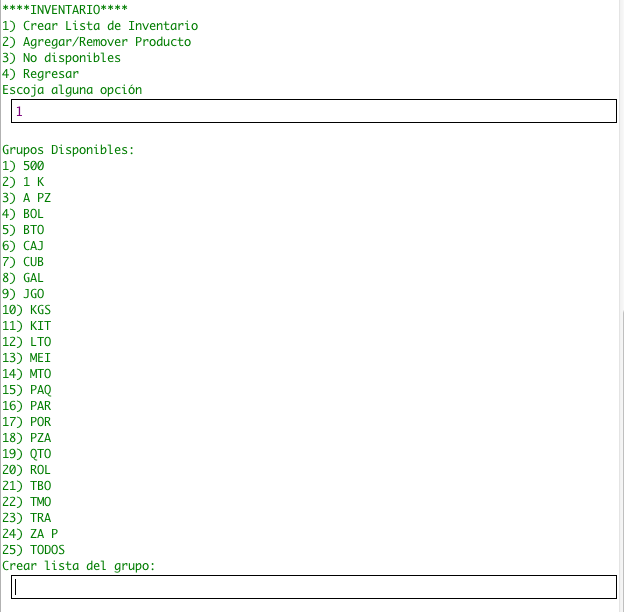
The user has accidentally written the code wrong but the program tells us that it has revised the list but no product was found so it requests another code:



Now that the program has found our product the program asks us if this is our product, let’s enter the letter “s” which stands for “yes”:



Now let’s check if the product was really removed from the inventory by creating another list from the group where the product belonged:



1) Create inventory List

2) Add/remove Product

3) Unavailable products

4) Go back

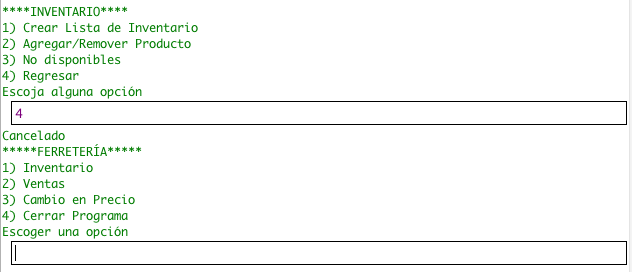
Inventory subgroups





Product is NOT in the inventory

As we can see the previously added product has now been removed from the inventory. Now let’s try to make a sale. First let’s go back into the main menu by entering 4:



1) Create inventory List

2) Add/remove Product

3) Unavailable products

4) Go back

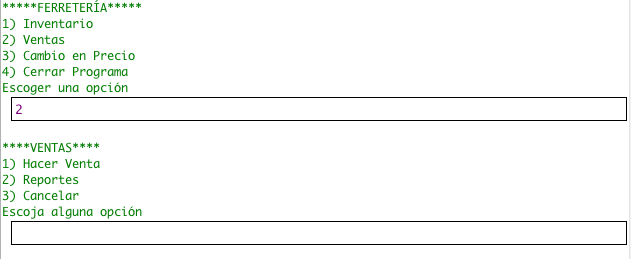
1) Inventory

2) Sales

3) Change in Price

4) Close Program

Now let’s input number 2 to go into the sales section:

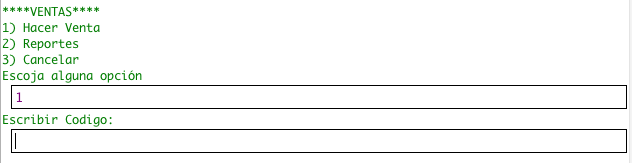


1) Make Sale

2) Reports

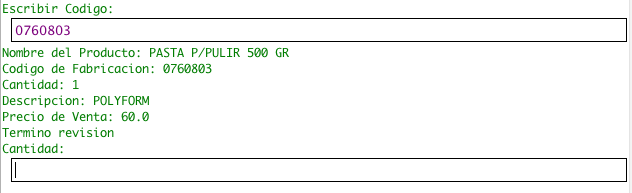
3) Cancel

Now we must enter 1 so we can now make a sale:



Product’s Code

Now the program is requesting a code from a product to make a sale. Let’s input the code from the remaining product in the previous group:



Product Name:

Product Code:

Quantity:

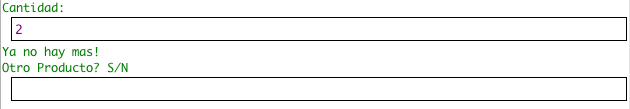
Description:

Price:

“Revision Finished”

Product’s Quantity

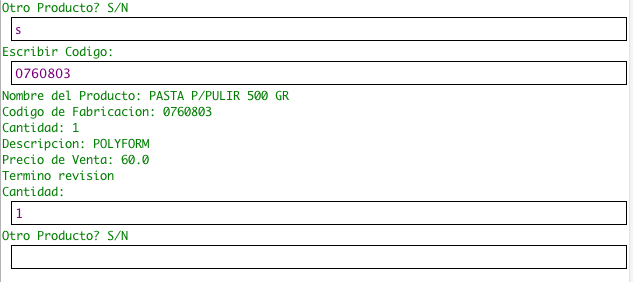
When the program has found our product it will display all of its important data and now we must input the quantity that will be sold:



“There are no more!”

Another Product?

Accidentally the user wanted to sale 2 of the same product but there is only one available, so let’s try it again but only selling one this time:



Another Product?

Product Name:

Product Code:

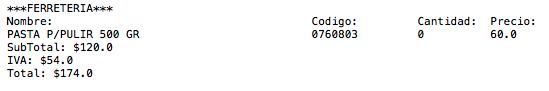
Quantity:

Description:

Price:

“Revision Finished”

After re-entering the required data now we can see that the program does not tells us that there not enough materials, but now its asking if here will b anything else sold besides the already chosen product. Let’s input “n” which means no to finish the sale:

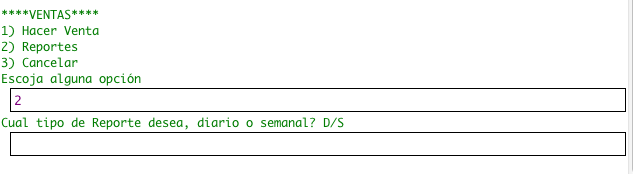


Subtotal:

Tax:

Total:

This is the receipt that was made when the sale was terminated. Now after making a sale let’s check our daily report. To do this we must enter 2 so we can go into the reports section:



1) Make Sale

2) Reports

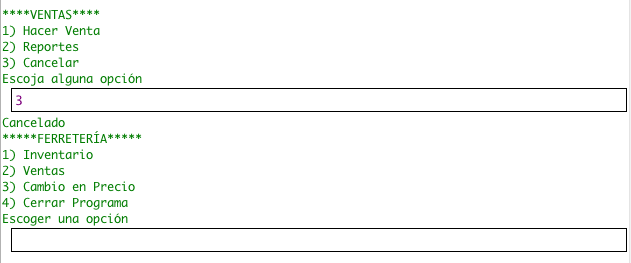
3) Cancel

Which type of report, daily or weekly? D/W

Now that we are in the reports section we can choose between daily report or weekly report but since a week has not obviously past, we will choose the daily report by entering “d”:

Macintosh HD:Users:biblioteca-13:Downloads:Screen Shot 2012-11-22 at 9.24.18 AM.png

This is the daily report. No lets check if there are any unavailable products in the business in the not available section, to go to this section first we must enter 3:



1) Make Sale

2) Reports

3) Cancel

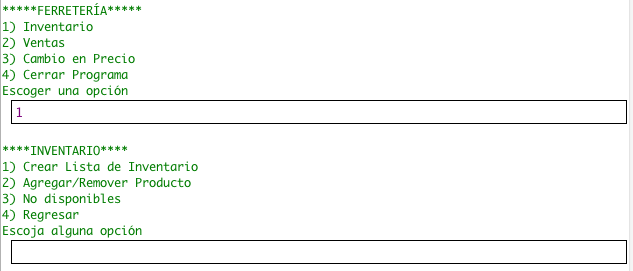
1) Inventory

2) Sales

3) Change in Price

4) Close Program

Then we must input 1:



1) Inventory

2) Sales

3) Change in Price

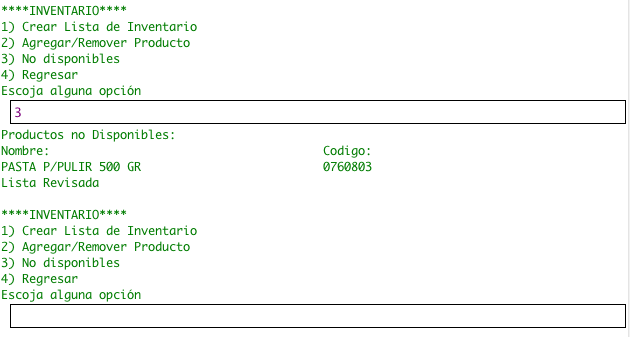
4) Close Program

1) Create inventory List

2) Add/remove Product

3) Unavailable products

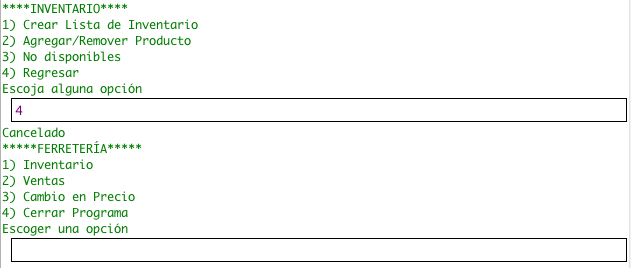
4) Go back



Unavailable Products

Now we must enter 3 to go to he not available section:

As we can see there is one product that is out of stock! So when merchandise arrives at the business we can go into the price updating section where we can add new ordered stock or changes in prices. To go to this section let’s enter 4:



1) Create inventory List

2) Add/remove Product

3) Unavailable products

4) Go back

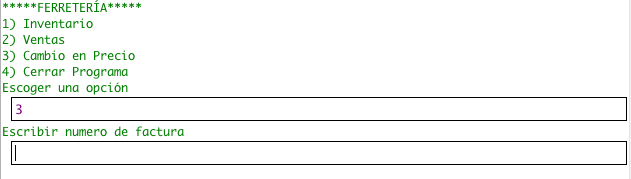
“Canceled”

1) Inventory

2) Sales

3) Change in Price

4) Close Program



1) Inventory

2) Sales

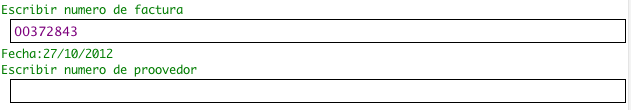
3) Change in Price

4) Close Program

Invoice Number:

Now let’s input 3 so we can finally access the price updating section:

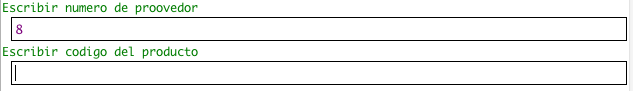
Now the program is requesting some data for he completion of this process. First we must input the invoice number, let’s suppose the number is 00372843:



Date

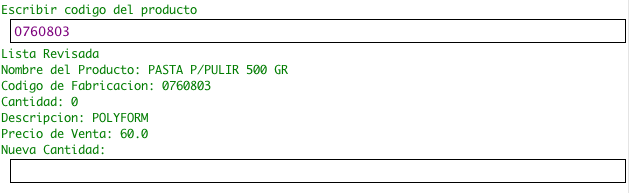
Supplier Number:

Immediately the program shows us the date that automatically is written in. Now the program requests the supplier number, let’s input supplier number 8:



Product’s Code

Now it is requesting the product’s code. If we check back in the not available list, we can see that our product’s code is 0760803:



Product Name:

Product Code:

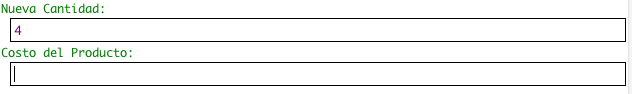
Quantity:

Description:

Price:

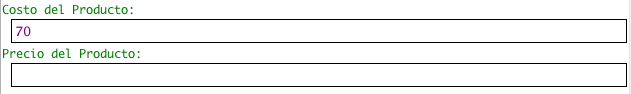
“Revision Finished”

Now that the program ahs found our product we can se that it is still out of stock but now we can input the new quantity that has arrived with new merchandise, let’s suppose that 4 have arrived:



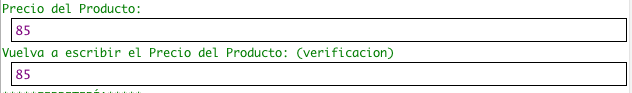
Product’s Cost

w the program is requesting for a new cost (in case there is) if not the user can type in the previous one, let’s supposes that there was a change and the new cost is $70:



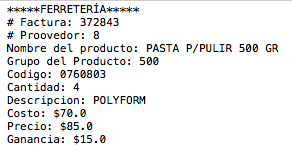
Product’s Price

Now the product’s price is requested and since the cost has changed that means that the price must also change, so let’s input the price at $85:



Product’s Price (Verification)

Now that the updating price process is finished a report is created for the user to see that all changes made were correct:



Invoice Number

Supplier Number

Product Name:

Product’s Group:

Code:

Quantity:

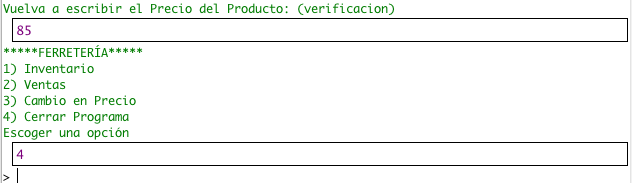
Description:

Cost:

Price:

Profit:

Now let’s suppose the user has finished the day and wants to close the program, so let’s input number 4 so we can finalize the program and save all previously modified data:



1) Inventory

2) Sales

3) Change in Price

4) Close Program

**Mastery Aspects**

|  |  |
| --- | --- |
| Mastery Aspect | Code line(s) |
| 1. Arrays | Ferreteria.java 🡪 Line 29, Page 32  Ferreteria.java 🡪 Lines 558-559, Page 42  Ferreteria.java 🡪 Lines 433-438, Page 40 |
| 2. User-defined objects | Ferreteria.java 🡪 Lines 558-559, Page 42  Product.java 🡪 Lines 17-27, Page 67 |
| 3. Objects as data records | Ferreteria.java 🡪 Lines 558-559, Page 42  Product.java 🡪 Lines 17-27, Page 67 |
| 4. Simple selection (if – else) | Ferreteria.java 🡪 Lines 400-401, Page 39 |
| 5. Complex selection (nested if, if with multiple conditions or switch) | Ferreteria.java 🡪 Lines 164-231, Page 35-36 |
| 6. Loops | Ferreteria.java 🡪 Lines 600-609, Page 43 |
| 7. Nested loops | Ferreteria.java 🡪 Lines 428-445, Page 40 |
| 8. User-defined methods | Ferreteria.java 🡪 Lines 368-564, Page 38-42 |
| 9. User-defined methods with appropriate parameters | Ferreteria.java 🡪 Lines 368, Page 38 |
| 10. User-defined methods with appropriate return values | Ferreteria.java 🡪 Lines 368, Page 38 |
| 11. Sorting | Ferreteria.java 🡪 Lines 1375-1405, Page 58 |
| 12. Searching | Ferreteria.java 🡪 Lines 600-609, Page 43 |
| 13. File I/O | Ferreteria.java 🡪 Lines 24, Page 32  Ferreteria.java 🡪 Lines 33-39, Page 32 |
| 14. Use of additional libraries (e.g. utilities and graphical libraries) | Ferreteria.java 🡪 Lines 1820-1827, Page 66 |
| 15. Use of sentinels or flags | Ferreteria.java 🡪 Lines 428-445, Page 39 |

**D2. Evaluating Solutions**

Finally the program has been done. After many hours and days of work the program is completed. Recalling the previous set of goals and objectives, many of the goals were accomplished. The system was improved in many ways that made the program more efficient, user friendly, and effective. Regarding the previously set goals the following were accomplished:

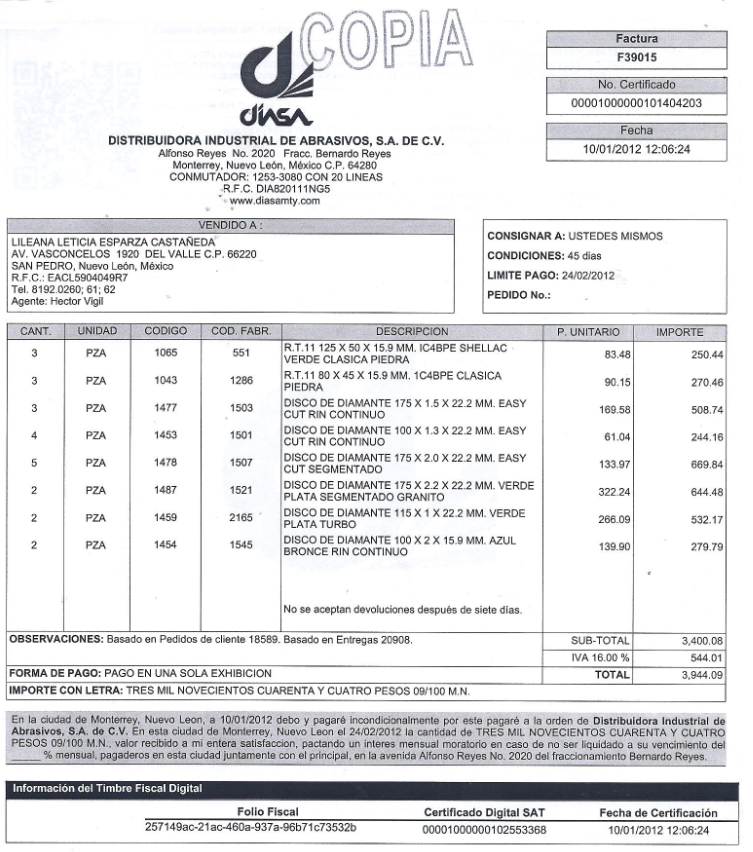
* Improve price-updating process:
* Implement log system to ensure a database for possible revision of this process if required by the user.
* Optimize the process by making it easier for the user to complete by the use of simple questions to complete.
* Implement automatic date inputting for eliminating the possibility of user mistyping.
* User Friendly:
* Easy to add, remove, make lists and make price updates with simple path to follow to accomplish a certain task.
* If an error is present, the user will be notified about his mistake so it can be corrected.
* Avoid mistyping of prices to evade the loss of profits in the business by making the user input the product’s price twice.
* Inventory:
* Implement shortage system section where the user will be notified if any product is currently out of stock.
* Implement list printing method where the user can choose the group he wants the list of.
* Give the user the capability to add or remove any desired product if there’s a need to include a new one or remove one.
* Selling system:
* Optimize selling system:
  + - Easier use for improving the efficiency while making a sale to avoid unhappy waiting customers.
* Include a full description of products when making sale so the user can be sure that the desired product is being sold.
* Implement receipt creation system where the client can decide if he/she will like to have the receipt printed when products are paid at cashier.
* Implement reports system:
* Give the user the capability to create a report that informs the user about the sales of the day.
* Give the user the capability to create a report that informs the user about the sales of the week.

Mainly after observing all of these accomplished and not accomplished goals. It is easy to notice that the system was really improved. Since it seems that most of the goals were accomplished, this means that the system was really optimized. This means that the program has become more efficient than its predecessor since the achieving of these goals will improve how things work in the client’s business. For example, employees will now have less difficulty understanding the system and will become very familiarized with the system very quick thanks to its simplicity. Also since the selling process has been made easier and faster people that usually wait at the business to be attended will be attended faster improving the workflow in the business. Also the inventory is now very easy to handle and control since previously changing it was a nightmare or almost impossible because of its complicated interface. Finally there was only one main goal that was not accomplished, the reports section. Sadly this section has flaws and it is not functional thus creating a limitation in the program. After understanding this the new system is more effective and efficient that will make this business have a better and easier workflow.

After analyzing the new system there could be more things that could be included that will help the system be even better. For starters, the first thing that could be included in the system would be a functional reports section that will allow the user to create the desired reports. Also another limitation that was found while testing the program was that new added products were not capable of being removed from he inventory, so another improvement could be fixing this issue. Furthermore another process that could be included into the system could be an alarm kind of system that informs the user when a low amount of stock is coming and it can be configurable for every single product since every product has a different demand. All of this will make the business have a more effective workflow since it will help it have a better control of inventory, reports that will inform the user if something needs to be done to improve or if things are being doing well.

After revising all of the previous data I think that the initial design used to create this system was appropriate since it covered the main areas that requires a considerable improvement that will make the business more effective than before, attending clients faster which means more profit. This system complies with its objectives and goals making it a successful achievement for the future use in the business. This making the business a more efficient and effective workforce that will continually grow to be one of the top hardware stores in the city.

**Appendix A: Invoice Copy**



Description

Fabrication Code

Cost per Unit

Total Cost

Quantity

**Appendix B: Current System**



Quantity

Fabrication Code

Description

Cost per Unit

Total Cost