Criterion C: Development

Techniques used to create the Project

- Created two main ArrayLists, one to store all the Items in the current Inventory, and one to record the sale movements.
- Built Algorithms to find key values about each Item
- Built unique GUI to display the up-to-date Inventory and Sales record, and be able to look into every item in detail.
- Used nested for loops to efficiently and effectively display all the plates of that item.
- Used for loops in algorithms to add Items to the inventory or alter them as well as to record sales
- Created searching algorithms to search through Inventory and Sales arrays.
- Created a method to filter the table's content depending on the content of a Text Field, in order to search through the Inventory or the Sales and get real-time results of your search.
- Used iText Library in order to generate ordered PDF reports of Inventory and Sales
- Created FileWriter and FileReader methods to save changes made to the Inventory and Sales

Created two main ArrayLists, one to store all the Items in the current Inventory, and one to record the sale movements

- The first ArrayList is instantiated in the Inventory class. It is public and static, in order to be accessed and manipulated from other classes, since the Inventory class will never be instantiated itself. Used Diamond Interface to make this ArrayList only contain Item objects, and doesn't have a fixed size as seen in *Figure 1*

```
public static ArrayList<Item> inventory = new ArrayList<>();
```

Figure 1

- The second ArrayList is instantiated in the Sales class. It is identical to the Inventory ArrayList in order to store those items which are removed from the Inventory ArrayList, and record them as sales. It is also public and static due to the fact that it must be accessed and manipulated from outside the Sales class, and the Sales class will never be instantiated itself. They can also only contain Item objects, and doesn't have a fixed size, as can be seen in *Figure 2.*

```
public static ArrayList<Item> sales = new ArrayList<>();
```

Figure 2

Built Algorithms to find key values about each Item

- Created a method in the Item object to return the average cost of all the plates in the ArrayList as seen in *Figure 3.*

Loops i from 0 to size of plates arraylist

```
public double avgCost(){
    double total = 0;
    int count = 0;
    for(int i = 0; i<plates.size();i++){
        total = total + plates.get(i).cost;
        count++;
    }
    double avgCost = total/count;
    return avgCost;
}
```

Adds the cost of the plate in index i to total.

Total divided by number of plates to get average

Figure 3

- Created a method in the Item object to return the total cost of having bought all the plates in the item, as can be seen in *Figure 4.*

Loops i from 0 to size of plates arraylist

```
public double getTotalCost(){
    double totalCost = 0;
    for(int i = 0; i<plates.size();i++){
        totalCost += plates.get(i).cost;
    }
    return totalCost;
}
```

Adds up all the costs

Figure 4

- Created a method in the Item object to return the total revenue of having sold the plates which have been sold, as can be seen in *Figure 5.*

Loops j from 0 to size of plates arraylist

```
public double getTotalRevenue(){
    double totalRevenue = 0;
    for(int j = 0; j<plates.size();j++){
        totalRevenue += plates.get(j).price;
    }
    return totalRevenue;
}
```

Adds up all the prices

Figure 5

- Created a method to calculate the total profit made on each individual plate, as can be seen in *Figure 6.*

Loops i from 0 to size of plates arraylist

```
public double getTotalProfit(){
    double totalProfit = 0;
    for(int i = 0; i<plates.size();i++){
        totalProfit += (plates.get(i).price - plates.get(i).cost);
    }
    return totalProfit;
}
```

Adds up all the differences of the price and the cost of every PlateOject

Figure 7

Built unique GUI to display the up-to-date Inventory and Sales record, and be able to look into every item in detail. Used nested for loops to efficiently and effectively display all the plates of that item.

- Page to see the contents of the Inventory seen in *Figure 8*

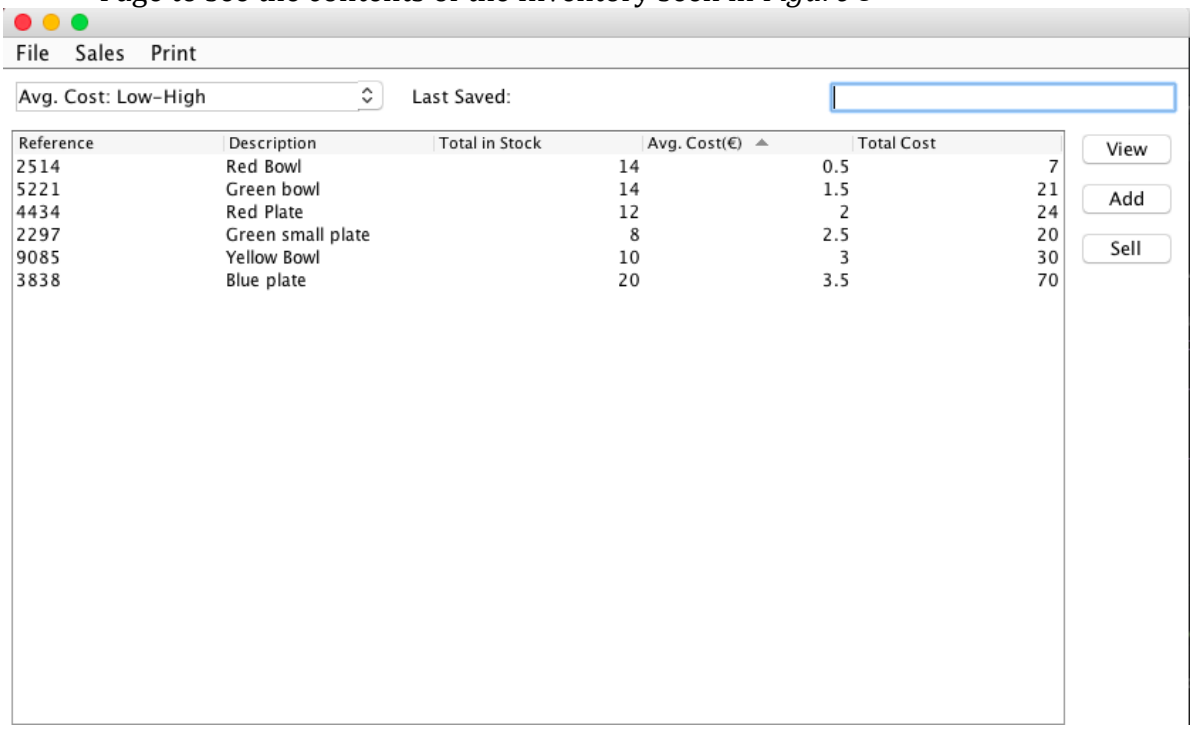
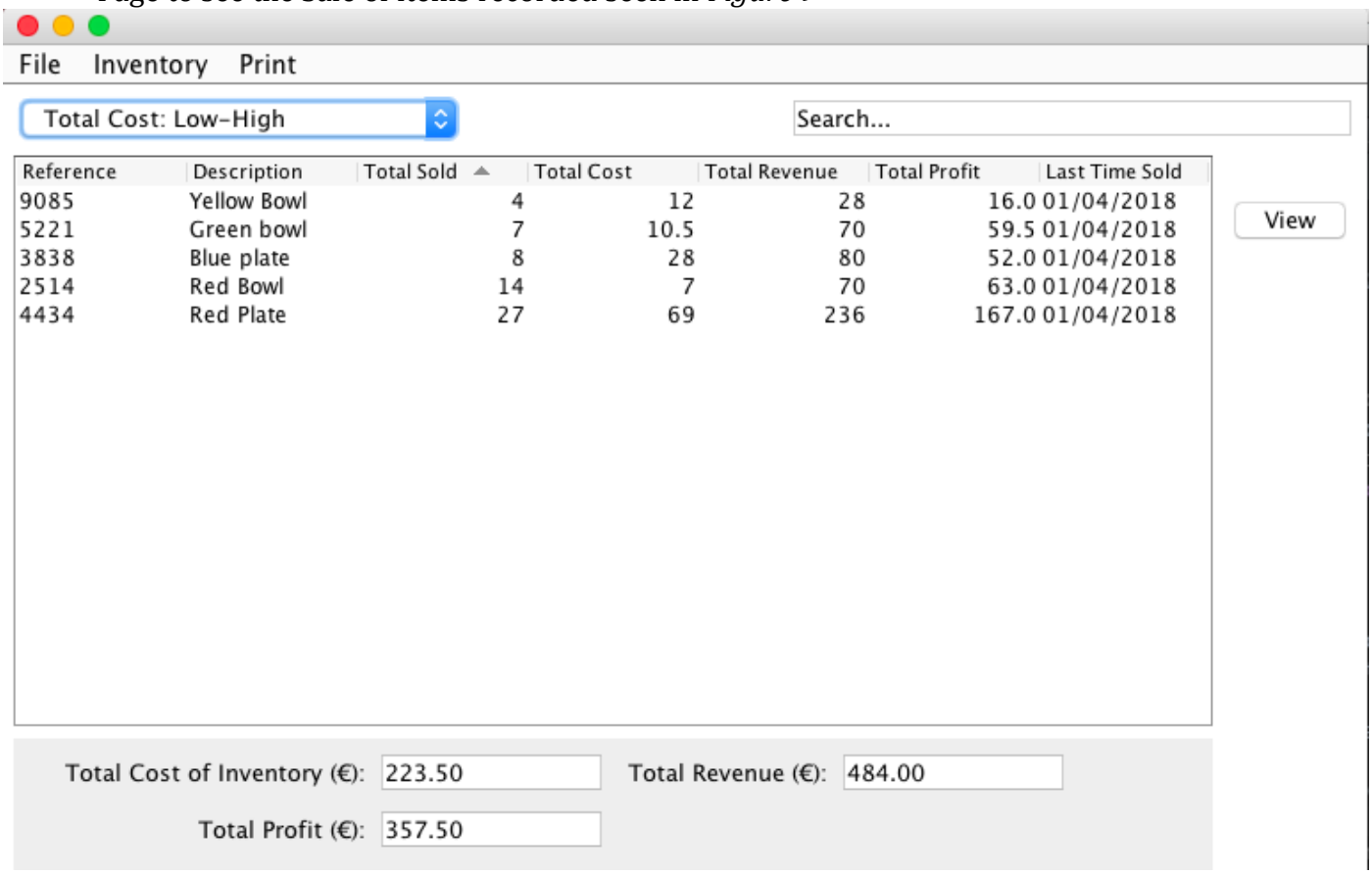


Figure 8

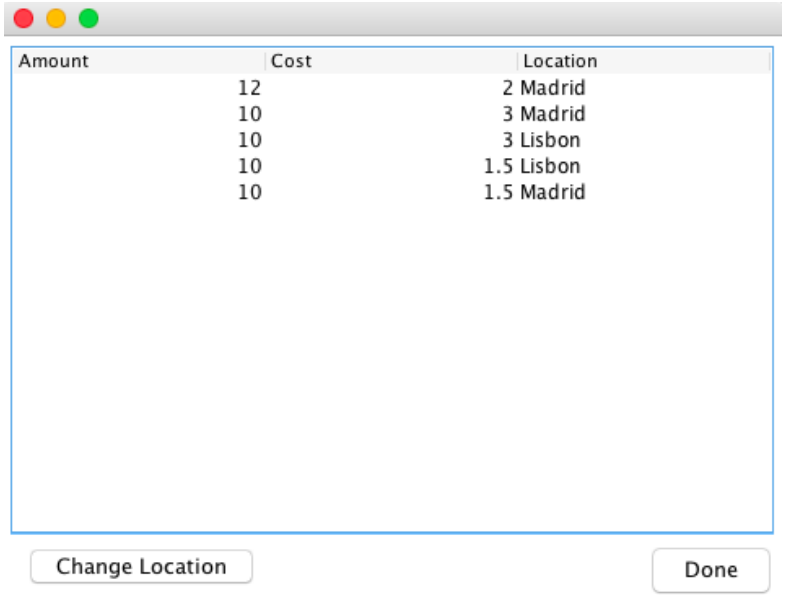
- Page to see the Sale of items recorded seen in *Figure 9*



Reference	Description	Total Sold	Total Cost	Total Revenue	Total Profit	Last Time Sold
9085	Yellow Bowl	4	12	28	16.0	01/04/2018
5221	Green bowl	7	10.5	70	59.5	01/04/2018
3838	Blue plate	8	28	80	52.0	01/04/2018
2514	Red Bowl	14	7	70	63.0	01/04/2018
4434	Red Plate	27	69	236	167.0	01/04/2018

Figure 9

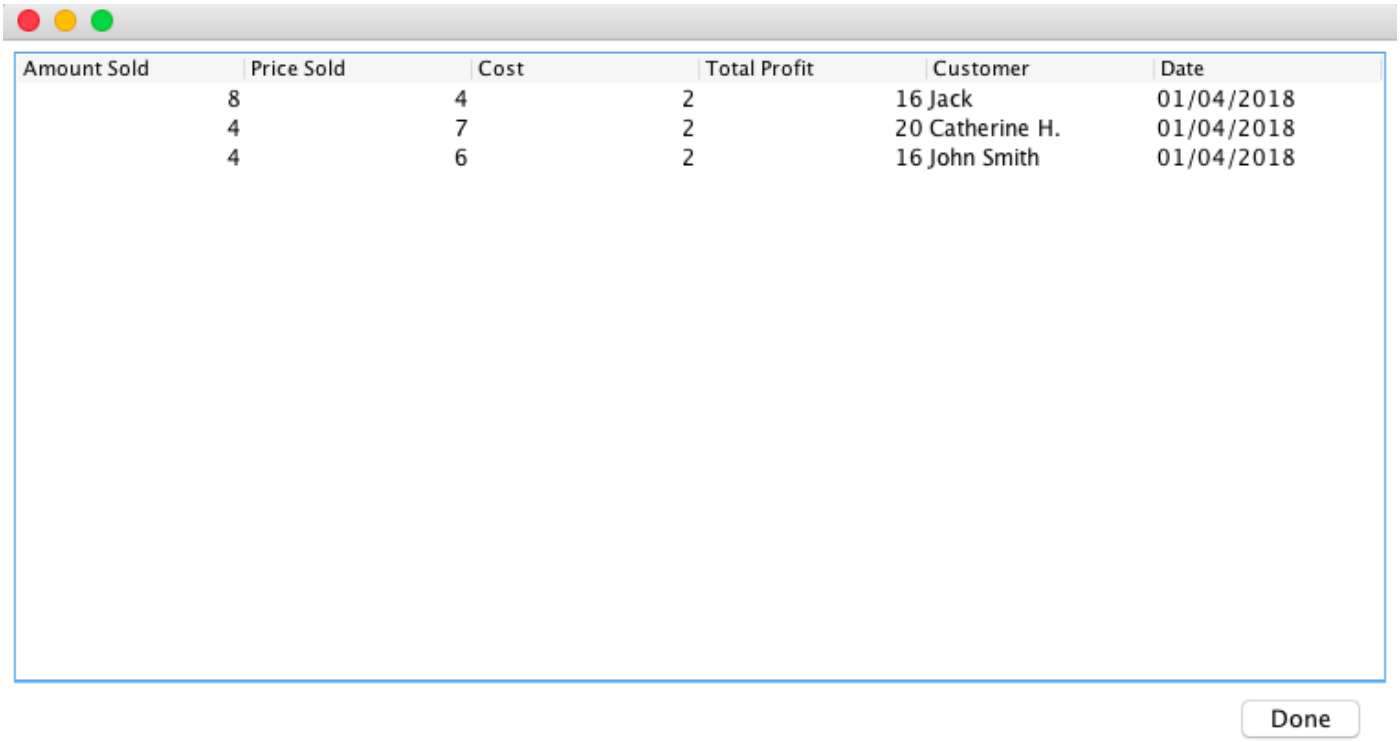
- Page to see every Item in the inventory in more detail seen in *Figure 10*



Amount	Cost	Location
12	2	Madrid
10	3	Madrid
10	3	Lisbon
10	1.5	Lisbon
10	1.5	Madrid

Figure 10

- Page to see the Sales of each item in more detail seen in *Figure 11*



Amount Sold	Price Sold	Cost	Total Profit	Customer	Date
8	4	2	16	Jack	01/04/2018
4	7	2	20	Catherine H.	01/04/2018
4	6	2	16	John Smith	01/04/2018

Done

Figure 11

Used nested for loops to efficiently and effectively display all the plates of that item.

- Method to show an up-to-date version of the Inventory. This method is called every time the class ColoresDaTerra runs, and shows the updated Inventory as can be seen in *Figure 12.* The method to display the Sales is exactly the same except for the data being added to the rows.

Private as it will only be accessed inside this class

Assigns the TableModel of the iTable in the GUI to the DefaultTableModel model

```
private void addPlatestoTable(){
    DecimalFormat df = new DecimalFormat("##.00");
    DefaultTableModel model = (DefaultTableModel) DisplayTable.getModel();
    Object rowData[] = new Object[4];
    for(int i = 0;i<Database.Inventory.inventory.size();i++)
    {
        rowData[0] = Database.Inventory.inventory.get(i).reference;
        rowData[1] = Database.Inventory.inventory.get(i).description;
        rowData[2] = Database.Inventory.inventory.get(i).totalStock();
        if(Database.Inventory.inventory.get(i).avgCost() == 0 ){
            rowData[3] = 0;
        }
        else{
            rowData[3] = Double.parseDouble(df.format(Database.Inventory.inventory.get(i).avgCost()));
        }
        model.addRow(rowData);
    }
}
```

Instantiates the rowData array, which is used to represent one row of Data in the table.

Loops i from 0 to size of inventory

Fills rowData array with values from the Inventory at index i

Adds that row of Data to the jTable in the GUI

Figure 12

- Method to display the items for View All Plates, as can be seen in Figure 13.

Private as it will only be accessed inside this class

Sets the class variable 'item' equal to the class variable 'item' in the ViewInfoInventory class

```
private void bundle(){
    item = ViewInfoInventory.item;
    ArrayList<Integer> unq = new ArrayList<>();
    DecimalFormat df = new DecimalFormat("##.00");
    unq.add(0);
    for(int a = 0; a<item.plates.size();a++){
        int countun = 0;
        for(int k = 0; k<unq.size();k++){
            if(!(item.plates.get(a).cost == item.plates.get(unq.get(k)).cost && item.plates.get(a).location.equals(item.plates.get(unq.get(k)).location))
                countun++;
            }
        }
        if(countun == unq.size()){
            unq.add(a);
        }
    }

    for(int u = 0; u<unq.size();u++){
        double cost = item.plates.get(unq.get(u)).cost;
        String location = item.plates.get(unq.get(u)).location;
        int count = 0;
        for(int i = 0;i<item.plates.size();i++){
            if(location.equals(item.plates.get(i).location) && cost == item.plates.get(i).cost){
                count++;
            }
        }
        DefaultTableModel model = (DefaultTableModel) DisplayTable.getModel();
        Object rowData[] = new Object[3];
        rowData[0] = count;
        rowData[1] = cost;
        rowData[2] = location;
        model.addRow(rowData);
    }
}
```

Creates Arraylist (unq) to store the indices of unique PlateObjects in the plates arraylist of any item

Adds 0 (the first object in the plates arraylist) to unq to compare to other PlateObjects in the plates Arraylist

Loops through plates Arraylist

Loops through unq arraylsit, and compares the values of cost and location of the PlateObject at index k, to the indices which are recorded in unq.

If the value of cost and location at k don't match exactly with the values at any of the indices recorded in unq, the value of k is added to the unq arraylist.

Loops the variable u through the unq arraylist.

Counts how many PlateObjects in plates arraylist have same cost and location as PlateObject in unq at index u.

Adds the unique combination of cost and location, as well as the number of PlateObjects with that combination as a row in the table.

Figure 13

Similar bundle() algorithm (to the one for the Inventory) to view sales items in more detail, but also takes into account the dateSold and the price variables of the PlateObjects to bundle PlateObjects. It also displays different variables of the PlateObjects, as can be seen in *Figure 11*

Created algorithms to add Items to the inventory or alter them and to record sales

- Algorithm to determine whether user can sell given amount of Plates according to the Total Stock of that item seen in *Figure 14*

```
public static boolean referenceVerif(Item r, int a){
    boolean sale = false;
    if(r.totalStock() >= a)
    {
        sale = true;
    }
    return sale;
}
```

Set sale to false

Compares number of plates trying to be sold to total stock, if it is less or equal, sale becomes true

Figure 14

- Algorithm to add a new Item to the Inventory seen in *Figure 15*

```
public static boolean firstTimeAdd(String d, int a, double c, String l, File image) //Algorithm when you add a new item
{
    boolean success = true;
    String description = d;
    int amount = a;
    double cost = c;
    String location = l;
    File imageLoc = image;
    ArrayList p = new ArrayList();//adds the plateobjects to this arraylist in order to pass it to the Item and add it to the inventory
    for(int i = 0; i<amount;i++){
        p.add(p.size(), new objects.PlateObject(cost, 0, location, null));
    }
    try{
        inventory.add(inventory.size(), new objects.Item(referenceGenerator(), description, null, imageLoc, p));
    }
    catch (Exception e){
        System.out.print(e);
        success = false;
    }
    return success;
}
```

Instantiates all variables needed to create Item

Creates new ArrayList p to put in the new Item

Loops from 0 to amount of plates being added, adds PlateObjects to p with given cost, given location and a price of 0

Tries to add item at the end of the inventory with description, new reference, null lastSold Date, given image, and created arrayList

If there is an error adding the item, it prints p and makes success false

Figure 15

- Algorithm to increase the stock of an existing Item seen in *Figure 16*

```
public static boolean addStock(int a, double c, String l, objects.Item item){
    boolean success = false;
    objects.Item select = item;
    int amount = a;
    double cost = c;
    String location = l;
    for(int i = 0;i<amount;i++){
        select.plates.add(select.plates.size(), new objects.PlateObject(cost, 0, location, null));
        success= true;
    }
    return success;
}
```

Instantiates value of success, select, amount, cost and location, with values passed by the user.

Loops I from 0 to amount of plates added by user, and adds new PlateObjects to the end of the 'select' Item plates ArrayList

Figure 16

- Algorithm to record the sale of an Item that has never been sold before (using referenceCheck algorithm seen in *Figure 19)* can be seen in *Figure 17*

```java
public static boolean recordFirstSale(double p, int a, String l, int r){
    double price = p;
    int amount = a;
    String location =l;
    int reference = r;
    int ii = Database.Inventory.findIndex(reference);
    boolean success;
    try{
        ArrayList<PlateObject> plts = new ArrayList<>(); //creates arraylist to hold plates being sold.
        SimpleDateFormat ft = new SimpleDateFormat("dd/MM/yyyy");
        Date d = new Date();
        ft.format(d);
        for(int i = 0; i<amount;i++){
            double cost = inventory.get(ii).plates.get(0).cost;
            plts.add(new PlateObject(cost,price,location,d));
            inventory.get(ii).plates.remove(0); //removes items from inventory
        }
        inventory.get(ii).lastSold = new Date();
        ft.format(inventory.get(ii).lastSold);
        String descrip = inventory.get(ii).description;
        File image = inventory.get(ii).image;
        sales.add(new Item(reference,descrip, inventory.get(ii).lastSold,image,plts));
        success = true;
    }
    catch(Exception e){
        System.out.println(e);
        success=false;
    }
    return success;
```

Instantiates parameters passed by user

This integer stores the index of the item in the inventory which the user is trying to sell

Creates new ArrayList to fill with PlateObjects

Tries to record the sale

Loops i from 0 to the amount of plates being sold

Gets cost from the PlateObject in the inventory

Adds new plate to plts ArrayList and removes PlateObject

Updates the lastSold Date of the item on the inventory.

Gets description and image from the item being sold from the inventory

Adds item to the sales ArrayList with description, cost, lastSold date, and image from the item in the inventory

Returns true if it recorded the sale successfully and false if there was a problem

Figure 17

- Algorithm to record the sale of an item which already exists in the Sales Arraylist, is the same, except for the fact that it doesn't have to create a new Item, simply add PlateObjects to the existing plates array of the Item in the Sales ArrayList.

Created searching algorithms to search through Inventory and Sales arrays.

- Algorithm to find the Index of an item based on its reference seen in *Figure 18*

```java
public static int findIndex(int r){
    int reference = r;
    int index = -1;
    for(int i = 0; i<inventory.size();i++){
        if (inventory.get(i).reference == reference){
            index = i;
            break;
        }
    }
    return index;
}
```

Instantiates reference with int value passed to method by user

Loops i from 0 to size of inventory

Compares value of reference of Item in inventory at index i with reference given by user. If equal, it returns breaks loop and returns index.

Figure 18

- Algorithm to determine whether an Item exists in the Sales ArrayList or not seen in *Figure 19*

Loops i from 0 to size of Sales ArrayList

```
public static boolean referenceCheck(int r){
    boolean present = false;
    for(int i = 0; i<sales.size();i++){
        if(sales.get(i).reference == r)
            present =true;
    }
    return present;
}
```

Instantiates present as false

Figure 19

Created a method to filter the table's content depending on the content of a Text Field, in order to search through the Inventory or the Sales and get real-time results of your search, as seen in *Figure 20*

Instantiates the TableRowSorter

```
private void SearchFilter(String search){
    TableRowSorter<TableModel> sorter = new TableRowSorter<>(DisplayTable.getModel());
    DisplayTable.setRowSorter(sorter);
    RowFilter rowf = RowFilter.regexFilter("(?i)"+search);
    sorter.setRowFilter(rowf);
```

Sets RowSorter to jtable in GUI

Figure 20

Applies Row Filter

Creates filter which filters for appearances of "search" once or many times and is case insensitive

Used iText Library in order to generate ordered PDF reports of Inventory and Sales

Used iTextPdf 5.4[1] library to print a PDF Amit Tuli and Surbhi Agarwal's guide.[2] A print method in the Inventory class prints n number of items from the inventory in a given order, and a print method in the sales class prints n number of items from the inventory in a given order. The code for the print method in the inventory can be seen in *Figure 21*.

[1] "IText 5, a PDF Library to Create Yor Smart Document Workflow." *, a PDF Library to Create Your Smart Document Workflow.*, itextpdf.com/itext-5-core.

[2] Tuli, Amit, and Surbhi Agarwal. "Generate PDF Files from Java Applications Dynamically." *IBM - United States*, 12 Dec. 2012, www.ibm.com/developerworks/library/os-javapdf/index.html.

```java
public static void print(int n, String o) throws FileNotFoundException, DocumentException, IOException{
    int numb = n;
    String order = o;
    Document pdfFile = new Document(PageSize.A4, 50, 50, 50, 50);
    String path = order+"("+numb+"items).pdf";
    PdfWriter writer = PdfWriter.getInstance(pdfFile, new FileOutputStream(path));
    pdfFile.open();
    pdfFile.add(new Paragraph("Colores Da Terra",
        FontFactory.getFont(FontFactory.TIMES, 24, Font.BOLD)));
    pdfFile.add(new Paragraph("Print out of "+n+" objects in order: "+o+"\n\n",
        FontFactory.getFont(FontFactory.TIMES, 16, Font.NORMAL)));
    PdfPTable table = new PdfPTable(4);
    PdfPCell reference = new PdfPCell(new Phrase ("Reference", FontFactory.getFont(FontFactory.TIMES, 14, Font.BOLD)));
    table.addCell(reference);
    PdfPCell description = new PdfPCell(new Phrase("Description", FontFactory.getFont(FontFactory.TIMES, 14, Font.BOLD)));
    table.addCell(description);
    PdfPCell totalStock = new PdfPCell(new Phrase("Total in Stock", FontFactory.getFont(FontFactory.TIMES, 14, Font.BOLD)));
    table.addCell(totalStock);
    PdfPCell avgCost = new PdfPCell(new Phrase("Avg. Cost(€)", FontFactory.getFont(FontFactory.TIMES, 14, Font.BOLD)));
    table.addCell(avgCost);
    ArrayList<Item> printlist = inventory;
   switch(order){
        case "Avg. Cost: Low-High":
            Collections.sort(printlist, new Comparator<Item>() {
                @Override
                public int compare( Item p1, Item p2){
                    return (int) (p1.avgCost() - p2.avgCost());
                }
            });
            break;
        case "Avg. Cost: High-Low":
            Collections.sort(printlist, new Comparator<Item>() {
                @Override
                public int compare( Item p1, Item p2){
                    return (int) (p2.avgCost() - p1.avgCost());
                }
            });
            break;
        case "Stock: Low-High":
            Collections.sort(printlist, new Comparator<Item>() {
                @Override
                public int compare( Item p1, Item p2){
                    return (int) (p1.totalStock() - p2.totalStock());
                }
            });
            break;
        case "Stock: High-Low":
            Collections.sort(printlist, new Comparator<Item>() {
                @Override
                public int compare( Item p1, Item p2){
                    return (int) (p2.totalStock() - p1.totalStock());
                }
            });
            break;
        default:
            break;

    }
    for(int i =0; i<numb;i++){
        table.addCell(Integer.toString(printlist.get(i).reference));
        table.addCell(printlist.get(i).description);
        table.addCell(Integer.toString(printlist.get(i).totalStock()));
        table.addCell(Double.toString(printlist.get(i).avgCost()));
    }
    table.addCell("");
    table.addCell("");
    table.addCell("");
    table.addCell(new Phrase("Total:", FontFactory.getFont(FontFactory.TIMES, 14, Font.BOLD))); //shows total cost of all items printed
    int total = 0;
    for(int i = 0; i<numb;i++){
        total += printlist.get(i).getTotalCost();
    }
    table.addCell(new Phrase(Double.toString(total), FontFactory.getFont(FontFactory.TIMES, 14, Font.BOLD)));
    pdfFile.add(table);
    pdfFile.close();
    Desktop.getDesktop().open(new File(path)); //opens PDF
}
```

Creates A4 document

Creates file using PdfWriter and the path String, created from order and number of items printed

Opens document

Creates and adds title and subtitle to document with appropriate font size

Creates headers for every column with bold font

Creates copy of Inventory called printlist

Creates Item Comparator to sort printlist ArrayList

Compares the two by subtracting them. Returns a negative integer, zero, or a positive integer to know if p2 is smaller, equal or greater respectively. Orders them in ascending order

Orders printlist in given order using a Collections and Comparator java libraries

By switching the order, it returns a negative integer, zero, or a positive integer to know if p1 is greater, equal or smaller respectively. Therefore, sorting printlist in descending order.

Loops through prinlist as many times as the number of items user wants to print

Adds values from Items in the prinlist at index into the table in the Document

Calculates Total Cost of displayed Items, and adds it to the table's last row

Adds table to document, closes document, and then open PDF document on user's computer

Figure 21

- In order to print the Sales records, the algorithm is similar, however, instead of creating a printlist and ordering it, I create a printlist, and find all the PlateObjects in the all the Items which have been

sold by using the Algorithm seen in *Figure 22*. Then the items are displayed like in the Inventory print, but showing the values of totalCost, totalRevenue, and totalProfit for each item, as well as a grand total.

```
public static void print(Date s, Date e)throws FileNotFoundException, DocumentException, IOException{
    Date start = s;
    Date end = e;
    SimpleDateFormat datef = new SimpleDateFormat("dd/MM/yyyy");
    ArrayList<Item> provi = new ArrayList<>();//This arrayList is going to contain all items which have plates between selected dates.
    for(int i = 0; i<sales.size();i++){
        int size = sales.get(i).plates.size();
        boolean contains=false;
        for(int j = 0; j<size;j++){ //checks if this item has any plates sold between the selected dates
            if(sales.get(i).plates.get(j).dateSold.after(start) && sales.get(i).plates.get(j).dateSold.before(end)){
                contains =true;
                break;
            }
        }
        if(contains){
            ArrayList<PlateObject> provp = new ArrayList<>();//this arraylist is going to contain all the plates that are sold between the bet
            for(int p = 0;p<size;p++){
                if(sales.get(i).plates.get(p).dateSold.after(start) && sales.get(i).plates.get(p).dateSold.before(end)){
                    provp.add(sales.get(i).plates.get(p));
                }
            }
            provi.add(new Item(sales.get(i).reference, sales.get(i).description, sales.get(i).lastSold, sales.get(i).image, provp));
        }
    }
```

Instantiates start and end dates

Creates provisional ArrayList to hold Items

Loops through Sales

Sees if any of the plates have been sold between given dates

Loops through plates of Item at index i

Runs if item has plates sold between dates

Creates provisional ArrayList to hold PlateObjects

Loops through plates of item, and adds any sold between dates to provp

Adds Item which contains plates between dates, and passes provp, which only contains plates sold between dates

Figure 22

Created FileWriter and FileReader methods to save changes made to the Inventory.

The FileWriter can be seen in *Figure 23* and the fileReader can be seen in *Figure 24*

```
public static boolean fileWriter(){
    boolean save;
    try{
    FileWriter fw = new FileWriter("resources/Files/SaveInventory.txt");
    PrintWriter pw = new PrintWriter(fw);
    String sv;
    SimpleDateFormat datef = new SimpleDateFormat("dd/MM/yyyy");
    sv = datef.format(lastSaved);
    pw.println(sv);
    sv = "";
    for(int i = 0;i < inventory.size();i++){
        if(inventory.get(i).lastSold == null){
            if(inventory.get(i).image == null){
             sv = inventory.get(i).reference +","+ inventory.get(i).description +","+" "+","+ " ";
            }
            else{
             sv = inventory.get(i).reference +","+ inventory.get(i).description +","+" "+","+ inventory.get(i).image.getCanonicalPath();
            }
        }
        else{
            sv = inventory.get(i).reference +","+ inventory.get(i).description +","+datef.format(inventory.get(i).lastSold)+","+inventory.get(i).image.getCanonicalPa
        }
        for(int p = 0;p<inventory.get(i).plates.size();p++){

            sv = sv + ","+inventory.get(i).plates.get(p).cost+","+inventory.get(i).plates.get(p).price+","+inventory.get(i).plates.get(p).location;
        }
        pw.println(sv);
    }
    save = true;
    fw.close();
    }
    catch(Exception e){
         System.out.println(e);
                save = false;
    }
    return save;
}
```

Tries to go through code with no erros

Create new FileWriter to SaveInventory .txt file in Project resources folder

Wraps FileWriter with PrintWriter, in order to be able to use println()

Creates local variable sv

Adds the formatted lastSaved variable from the Inventory to sv

Prints sv onto the txt file

Separates Item's variables using comas

Sets value of sv to different strings depending on the value of the lastSold and image variables of the Item at index i. It replaces variables which are null with " "

Loops through inventory

Prints sv as a line onto the txtfile, eachline represents an Item

Loops through plates ArrayList of Item at index i.

Adds the cost, price, and location of every Plateobject in the ArrayList to sv, separated by comas

Closes FileWriter

Prints error and sets variable save to false, to return that there has been an error

Figure 23

Catches any exception, and runs this code if any are caught

```java
public static void fileReader(){
    try{
        SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
        FileReader fr = new FileReader("resources/Files/SaveInventory.txt");
        BufferedReader br = new BufferedReader(fr);
        String temp = br.readLine();
        try{
        lastSaved = (java.util.Date) formatter.parse(temp);
        savedDate = true;
        }
        catch(Exception e){
            savedDate = false;
        }
        temp = br.readLine();
        while(temp != null){
        String store [] = temp.split(",");
        int reference = Integer.parseInt(store[0]);
        String description = store[1];
        java.util.Date lastSold;
        if(store[2].equals(" ")){
            lastSold = null;
        }
        else{
            lastSold = (java.sql.Date) formatter.parse(store[2]);
        }
        File image;
        if(store[3].equals(" ")){
            image = null;
        }
        else{
            String imageLocation= store[3];
            image = new File(imageLocation);
        }
        ArrayList p = new ArrayList();
        for(int j = 4; j<store.length;j+=3){
            double cost = Double.parseDouble(store[j]);
            double price = Double.parseDouble(store[j+1]);
            String location = store[j+2];
            p.add(p.size(), new objects.PlateObject(cost,price,location));
        }
        inventory.add(inventory.size(), new objects.Item(reference, description, lastSold, image, p));
        temp = br.readLine();
        }
        }
    catch(Exception e){
        System.out.println(e);
        }
}
```

Creates FileReader for .txt file in Project resources folder

Wraps FileReader with bufferedReader in order to use readLine()

Instantiates temp, assigning it the first line of the File

Parses temp (first line) into a Date, if succesfull, it sets value of classVariable savedDate to true

Tries to parse temp into a Date

If it fails, it sets savedDate to false, this is to prevent errors when first saved, as there would be no lastSaved date

Reads next line

Loops through this code until the next Line is empty (null)

Tries to read .txt File and catches any errors

Splits line into an array, splitting everytime it finds a coma

First two are always reference and dexcription as it is printed ref,

Checks if the third index in array is an empty space, if it isn't it parses into a Date

Checks if the fourth index in array is an empty space, if not, turns into file path and then image File

Creates a new ArrayList p to store read PlateObjects

Every index in store array which is a multiple of 3 when 4 is subtracted is a cost, when 5 is subtracted it's a price, and when 6 is subtracted it is a location

Loops j from 4 to the size of the store array, increasing by 3 everytime around

Use this logic to extract all costs, prices, and locations in that line and create PlateObjects to add them to

Reads next line

Adds Item created by the data in that line to end of inventory

Prints error, if there is any

Figure 24

- The FileWriter and FileReader of the sales ArrayList is identical, but it saves the contents of the sales ArrayList into another .txt file calles SaveSales.txt in which it also includes the dateSold of each PlateObject, which is not necessary in the Inventory, as they are all null.

Word Count: 956