

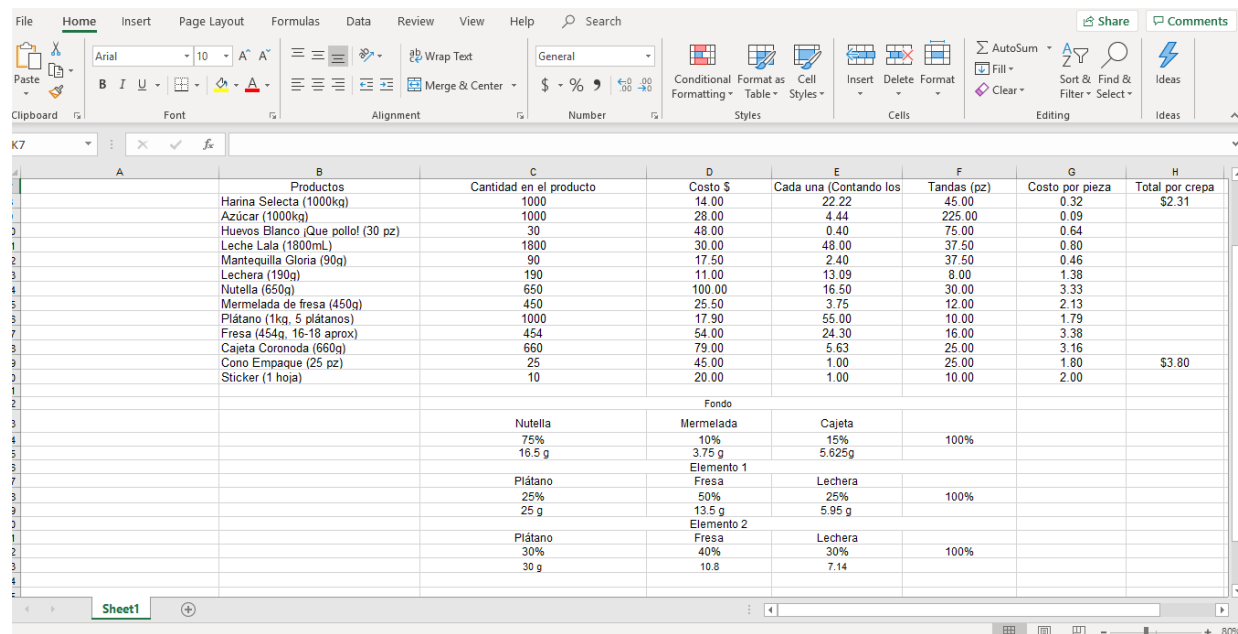
## Criterio C (Técnicas utilizadas)

### 1.-Manejo de Archivos

El manejo de archivos es parte fundamental del programa, ya que es necesario guardar información en el disco duro, a la que se acceda y modifique cuando se corra el programa, además de no borrarse al término de este. Se manejaron los siguientes tipos de archivos para funciones diferentes:

#### -Almacenamiento

Se utilizó la librería Apache.Poi para poder acceder a información dentro de un archivo .xlsx. La aplicación creada para Porticrepas obtiene el producto, cantidad y precio de diversas materias primas de este archivo para poder realizar el algoritmo que calcula e imprime los gastos para una cantidad de crepas digitada por el cliente. El archivo contiene los siguientes datos.



A	B	C	D	E	F	G	H
	Productos	Cantidad en el producto	Costo \$	Cada una (Contando los	Tandas (pz)	Costo por pieza	Total por crepa
1	Harina Selecta (1000kg)	1000	14.00	22.22	45.00	0.32	\$2.31
2	Azúcar (1000kg)	1000	28.00	4.44	225.00	0.09	
3	Huevos Blanco ¿Que pollo? (30 pz)	30	48.00	0.40	75.00	0.64	
4	Leche Lala (1800mL)	1800	30.00	48.00	37.50	0.80	
5	Mantequilla Gloria (90g)	90	17.50	2.40	37.50	0.46	
6	Lechera (190g)	190	11.00	13.09	8.00	1.38	
7	Nutella (650g)	650	100.00	16.50	30.00	3.33	
8	Mermelada de fresa (450g)	450	25.50	3.75	12.00	2.13	
9	Plátano (1kg, 5 plátanos)	1000	17.90	55.00	10.00	1.79	
10	Fresa (454g, 16-18 aprox)	454	54.00	24.30	16.00	3.38	
11	Cajeta Coronada (660g)	660	79.00	5.63	25.00	3.16	
12	Cono Empaque (25 pz)	25	45.00	1.00	25.00	1.80	\$3.80
13	Sticker (1 hoja)	10	20.00	1.00	10.00	2.00	
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							
36							
37							
38							
39							
40							
41							
42							
43							
44							
45							
46							
47							
48							
49							
50							
51							
52							
53							
54							
55							
56							
57							
58							
59							
60							
61							
62							
63							
64							
65							
66							
67							
68							
69							
70							
71							
72							
73							
74							
75							
76							
77							
78							
79							
80							
81							
82							
83							
84							
85							
86							
87							
88							
89							
90							
91							
92							
93							
94							
95							
96							
97							
98							
99							
100							

El código utilizado para leer estos datos es:

```

public String excel(int a, int b){
    try{
        FileInputStream File = new FileInputStream("C:\\Users\\Jorge Loreda\\Documents\\NetBeansProjects\\PortiApp\\src\\portiapp\\Costos PortiCrepa");
        XSSFWorkbook hoja = new XSSFWorkbook(File);
        if (hoja.getSheetAt(0).getRow(a).getCell(b).getCellType() == XSSFCell.CELL_TYPE_STRING)
            return hoja.getSheetAt(0).getRow(a).getCell(b).getStringCellValue();

        else if(hoja.getSheetAt(0).getRow(a).getCell(b).getCellType() == XSSFCell.CELL_TYPE_NUMERIC){
            return Double.toString(hoja.getSheetAt(0).getRow(a).getCell(b).getNumericCellValue());
        }
        hoja.close();
        File.close();
    }catch(Exception e){
        System.out.println("No se puede leer");
    } return "";
}

```

Cómo se puede observar el método recibe dos enteros como parámetros, el entero “a” dicta la fila y el entero “b” la columna. Primeramente, se crea un flujo de entrada de archivo y posteriormente una hoja del Excel. Las dos condicionales comparan si el dato en dicha celda es de tipo String o es un número, en cualquier caso, este valor se regresa como dato tipo String. Lo anterior ayuda a cumplir el segundo criterio de logro, para que los datos se mantengan actualizados. En el caso de que se necesite un número existe el método “excelnumeric” que regresa un dato double. Se instancian los valores de la siguiente manera:

```

String Mat[][] = new String [13][5];
for(int i =0; i<13; i++){
    Mat [i][0]= excel(i+7,1);
    Mat [i][1] = excel(i+7,4);
}

```

Finalmente se modela en una JTable. Esto quiere decir que si cualquier característica de alguna materia prima cambia, el programa se actualiza de la misma manera.

# ¿Cuántas crepas tienes en mente vender?

Digite el # de crepas que quieras

PRODUCTO	INGREDIENTES	CANTIDAD	PRECIO	SOBRA
Harina Selecta (1000kg)	22.22	3	\$42.0	778
Azúcar (1000kg)	4.44	1	\$28.0	555
Huevos Blanco ¡Que pollo! (30 pz)	0.4	2	\$96.0	20
Leche Lala (1800mL)	48.0	3	\$90.0	600
Mantequilla Gloria (90g)	2.4	3	\$53.0	30
Lechera (190g)	13.09	7	\$77.0	21
Nutella (650g)	16.5	3	\$300.0	300
Mermelada de fresa (450g)	3.75	1	\$26.0	75
Plátano (1kg, 5 plátanos)	55.0	6	\$107.0	500
Fresa (454g, 16-18 aprox)	24.3	6	\$324.0	294
Cajeta Coronoda (660g)	5.625	1	\$79.0	97
Cono Empaque (25 pz)	1.0	4	\$180.0	0
Sticker (1 hoja)	1.0	10	\$200.0	0

Los costos totales son de: **\$1602.0**

Los ingresos máximos son de: **\$2500**

## -Archivos de texto

Se utilizó un archivo de texto que guarda el historial de las crepas realizadas, este se modifica a través de otra técnica llamada ArrayLists (se utilizó dos veces), la primera (“listalocal”) guarda los datos que contiene el archivo de texto y la segunda (“Listapedidos”) los insertados en la JTable. Cuando se ejecuta la opción salir, ambas estructuras de datos se combinan y se vuelven a guardar en el archivo de texto:

```

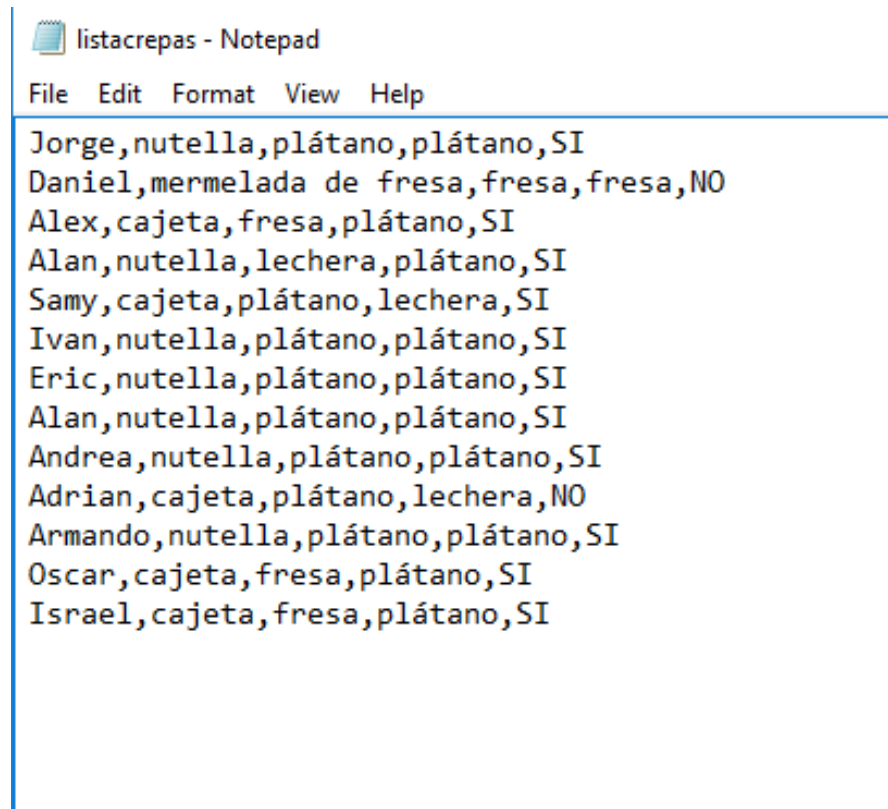
public void cargarr(){
    File ruta = new File("listacrepas.txt");
    try{

        FileReader File = new FileReader(ruta);
        BufferedReader bu = new BufferedReader(File);

        String linea = null;
        while((linea = bu.readLine())!=null){
            StringTokenizer Token = new StringTokenizer(linea, ",");
            p = new Crepa();
            p.setNombre(Token.nextToken());
            p.setFondo(Token.nextToken());
            p.setElemento1(Token.nextToken());
            p.setElemento2(Token.nextToken());
            p.setEnv(Token.nextToken());
            listalocal.add(p);
        }
        bu.close();
    }catch(Exception ex){
        System.out.println("error");
    }
}

```

P se instancia como una crepa auxiliar la cual copia cada atributo del archivo de texto, los cuales están separados por comas (",").



```

listacrepas - Notepad
File Edit Format View Help
Jorge,nutella,plátano,plátano,SI
Daniel,mermelada de fresa,fresa,fresa,NO
Alex,cajeta,fresa,plátano,SI
Alan,nutella,lechera,plátano,SI
Samy,cajeta,plátano,lechera,SI
Ivan,nutella,plátano,plátano,SI
Eric,nutella,plátano,plátano,SI
Alan,nutella,plátano,plátano,SI
Andrea,nutella,plátano,plátano,SI
Adrian,cajeta,plátano,lechera,NO
Armando,nutella,plátano,plátano,SI
Oscar,cajeta,fresa,plátano,SI
Israel,cajeta,fresa,plátano,SI

```

Las nuevas crepas se añaden cuando se oprime el botón “comprar”.

```
Listapedidos.add(new Crepa(nombre,Fondo,Elemento1,Elemento2,envoltura));
```

Finalmente se combinan y se guardan.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    gastos();  
    File ruta = new File("listacrepas.txt");  
    PrintWriter Escribe;  
    if(!ruta.exists()){  
        try{  
            ruta.createNewFile();  
        }catch(IOException e){  
        }  
    }  
    try{  
        Crepa aux;  
        for(int i= 0; i<Listapedidos.size(); i++){  
            listalocal.add(Listapedidos.get(i));  
        }  
    }catch(Exception ex){  
        System.out.println("error");  
    }  
    try{  
        Crepa aux;  
        Escribe = new PrintWriter(ruta, "utf-8");  
        for(int i= 0; i<listalocal.size(); i++){  
            aux =listalocal.get(i);  
            aux.guardar(Escribe);  
            Escribe.close();  
        }  
    }catch(Exception e){  
    }  
}  
  
void guardar(PrintWriter Escribe) {  
    Escribe.print(Nombre + ",");  
    Escribe.print(Fondo + ",");  
    Escribe.print(Elemento1 + ",");  
    Escribe.print(Elemento2 + ",");  
    Escribe.println(env);  
}
```

Primeramente, se accede al archivo (“listacrepas.txt”) mediante el objeto File, después se agregan todos los objetos de Listapedidos a listalocal, Finalmente, en un ciclo for y usando la función PrintWriter se escriben los datos en los archivos.

## 2.-Búsqueda Secuencial

El método búsqueda se utiliza para que el usuario pueda filtrar el historial de crepas dado un atributo específico. Por ejemplo, si desea visualizar todas las crepas que lleven cierto ingrediente se ejecuta el siguiente código.

```
private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {  
  
    int c = jComboBox1.getSelectedIndex();  
    String a = jTextField1.getText();  
    los = new ArrayList <>();  
    if(c==0){  
        for(int i = 0; i<lis.size(); i++){  
            if(lis.get(i).getNombre().equals(a))  
                los.add(lis.get(i));  
        }  
        verDatos1();  
    }  
    else if(c==1){  
        for(int i = 0; i<lis.size(); i++){  
            if(lis.get(i).getFondo().equals(a))  
                los.add(lis.get(i));  
        }  
        verDatos1();  
    }  
    else if(c==2){  
        for(int i = 0; i<lis.size(); i++){  
            if(lis.get(i).getElemento1().equals(a))  
                los.add(lis.get(i));  
        }  
        verDatos1();  
    }  
    else if(c==3){  
        for(int i = 0; i<lis.size(); i++){  
            if(lis.get(i).getElemento2().equals(a))  
                los.add(lis.get(i));  
        }  
        verDatos1();  
    }  
    else if(c==4){  
        for(int i = 0; i<lis.size(); i++){  
            if(lis.get(i).getEnv().equals(a))  
                los.add(lis.get(i));  
        }  
        verDatos1();  
    }  
    else {  
        for(int i = 0; i<lis.size(); i++){  
            if(lis.get(i).getNombre().equals(a))  
                los.add(lis.get(i));  
        }  
        verDatos1();  
    }  
}
```

Para esto, se recorre la lista “lis”, la cual contiene todos los objetos crepa. Dependiendo del índice de un ComboBox “c”, si el dato insertado por el usuario concuerda con algún atributo de cualquier crepa en el mismo índice, se guarda en otro ArrayList llamada “los”. Finalmente se ejecuta el método “verDatos1” y se modela en una JTable.

```

public void verDatos1(){
    String Mat[][] = new String [los.size()][5];
    Crepa aux;
    for (int i = 0; i<los.size(); i++){
        aux = los.get(i);
        Mat[i][0] = aux.getNombre();
        Mat[i][1] = aux.getFondo();
        Mat[i][2] = aux.getElemento1();
        Mat[i][3] = aux.getElemento2();
        Mat[i][4] = aux.getEnv();
    }
    jTable1.setModel(new javax.swing.table.DefaultTableModel(
        Mat,
        new String [] {
            "Nombre", "Fondo", "Elemento 1", "Elemento2", "Envoltura"
        }
    ));
}

```

### 3.- Programación Orientada a Objetos (POO)

Esta es la técnica que tiene más recurrencia en todo el código, ya que las crepas se manejan como objetos que fueron abstraídos, estos tienen cinco atributos que varían dependiendo del pedido de un cliente. Existe un accesor y mutator para cada atributo, así como diversos métodos que permiten su interacción y modificación por distintos algoritmos dentro del programa. Cuando se quiere crear un objeto crepa, primeramente, se obtienen los datos de estos a partir de la interfaz gráfica y posteriormente se mandan a la clase crepa.

```

public class Crepa {
    public String Nombre;
    public String Fondo;
    public String Elemento1;
    public String Elemento2;
    public String env;

    public Crepa(String Nombre, String Fondo, String Elemento1, String Elemento2, String env) {
        this.Nombre = Nombre;
        this.Fondo = Fondo;
        this.Elemento1 = Elemento1;
        this.Elemento2 = Elemento2;
        this.env = env;
    }
}

```

Manejar los datos de esta forma resulta útil, ya esto permite la filtración rápida en el historial, así como el fácil acceso para el cálculo de gastos.

El código utilizado para los getters y setters es el siguiente:

```

public String getNombre() {
    return Nombre;
}

public void setNombre(String Nombre) {
    this.Nombre = Nombre;
}

```

Se repite lo mismo para cada atributo diferente. Un método muy importante que se relaciona con los archivos de texto es el método cargar

```

public Crepa cargar(BufferedReader Almacen) {
    String Nom, Fon, E11, E12, en;
    try {
        Nom = Almacen.readLine();
        Fon = Almacen.readLine();
        E11 = Almacen.readLine();
        E12 = Almacen.readLine();
        en = Almacen.readLine();

        return new Crepa(Nom, Fon, E11, E12, en);
    } catch (Exception e) {
    }
    return null;
}

```

Este método lee los datos dado un Buffered instanciado en el código principal.

## 4.- Manejo de estructuras de datos

Durante todo el desarrollo de la aplicación, se utilizaron diversas estructuras de datos las cuáles tenían función diferente, así como acercamiento. Las dos más utilizadas fueron las siguientes.

### - ArrayLists

Se utilizaron alrededor de 4 ArrayLists diferentes en las ventanas del programa, todas fueron utilizadas para recopilar la información del archivo de texto, de maneras diferentes.

“Listapedidos” guarda las crepas compradas por los clientes en la última jornada laboral, “listalocal” guarda el historial de ventas, finalmente, “los” y “lis” asemejan el historial para su filtración.



```

private ArrayList<Crepa> listalocal;
private ArrayList<Crepa> Listapedidos;
public VentanaCliente() {

    listalocal = new ArrayList <>();
    Listapedidos = new ArrayList <>();
}

```

Siempre se instancia como nuevas, así como privadas ya que, no se permite que se accedan a estas desde otra clase. Es lo primero que se instancia al crear una nueva Ventana, debido a que, su modificación depende de la ejecución de botones. Por ejemplo, cuando se realiza una compra, la nueva crepa de la ArrayList se modela en la lista de pedidos.

```

public void verDatos() {
    String Mat[][] = new String [Listapedidos.size()][5];
    Crepa aux;
    for (int i = 0; i<Listapedidos.size(); i++){
        aux = Listapedidos.get(i);
        Mat[i][0] = aux.getNombre();
        Mat[i][1] = aux.getFondo();
        Mat[i][2] = aux.getElemento1();
        Mat[i][3] = aux.getElemento2();
        Mat[i][4] = aux.getEnv();
    }
    jTable1.setModel(new javax.swing.table.DefaultTableModel(
        Mat,
        new String [] {
            "Nombre", "Fondo", "Elemento 1", "Elemento2", "Envoltura"
        }
    ));
}

```

## - Matrices

Las matrices tuvieron dos funciones diferentes, la primera es para modelar las tablas dado a objetos de la clase crepa que son agregados, así como el historial de estas. Por ejemplo:

```

public void verDatos1(){
    String Mat[][] = new String [los.size()][5];
    Crepa aux;
    for (int i = 0; i<los.size(); i++){
        aux = los.get(i);
        Mat[i][0] = aux.getNombre();
        Mat[i][1] = aux.getFondo();
        Mat[i][2] = aux.getElemento1();
        Mat[i][3] = aux.getElemento2();
        Mat[i][4] = aux.getEnv();
    }
    jTable1.setModel(new javax.swing.table.DefaultTableModel(
        Mat,
        new String [] {
            "Nombre", "Fondo", "Elemento 1", "Elemento2", "Envoltura"
        }
    ));
}

```

En este fragmento de código, se aprecia cómo la Matriz Mat obtiene los atributos de las crepas y después son modelados en una tabla.

La segunda función se encuentra en la Ventana Ingredientes, donde la matriz utilizada guarda los campos de las columnas “Producto”, “Ingredientes”, “Cantidad”, “Precio” y “Sobra”, las cuáles se establecen mediante el acceso al archivo de Excel e iteración.

```

,
    String Mat[][] = new String [13][5];
    for(int i =0; i<13; i++){
Mat [i][0]= excel(i+7,1);
Mat [i][1] = excel(i+7,4);
    }

    for(int h = 0; h<13; h++){
        Mat[h][2] = Integer.toString(Cantidadd [h]);
    }

    for(int preciod = 0; preciod<13; preciod++){
        Mat[preciod][3] = "$" +Double.toString(Precio[preciod]);
    }

    for(int alt = 0; alt<Precio.length; alt++){
        Mat[alt][4] = Integer.toString(Sobra[alt]);
    }

```

## 5.- Manejo de excepciones

Es fundamental el manejo de excepciones, especialmente al abrir y cerrar archivos externos a Netbeans, cómo lo son Excel y los archivos de texto. Lo anterior se debe a que en ocasiones la ruta del archivo puede estar mal escrita por lo que el programa podría realizar sus demás funciones en vez de dejar de funcionar. Por ejemplo:

```
try{
    FileInputStream File = new FileInputStream("C:\\Users\\Jorge Loredó\\Documents\\NetBeans\\
    XSSFWorkbook hoja = new XSSFWorkbook(File);
    if (hoja.getSheetAt(0).getRow(a).getCell(b).getCellType() == XSSFCell.CELL_TYPE_STRING)
        return Double.parseDouble(hoja.getSheetAt(0).getRow(a).getCell(b).getStringCellValue());

    else if(hoja.getSheetAt(0).getRow(a).getCell(b).getCellType() == XSSFCell.CELL_TYPE_NUMERIC){
        return hoja.getSheetAt(0).getRow(a).getCell(b).getNumericCellValue();
    }

    hoja.close();
    File.close();
}catch(Exception e){
    System.out.println("No se puede leer");
}
```

De igual manera se presenta cuando el archivo no existe, por lo que se crea uno con el mismo nombre:

```
File ruta = new File("listacrepas.txt");
PrintWriter Escribe;
if(!ruta.exists()){
    try{
        ruta.createNewFile();
    }catch(IOException e){
    }
}
```

Finalmente, cuando se quiere eliminar una crepa y esta no es seleccionada:

```
try{
    c = jTable1.getSelectedRow();
    listalocal.add(Listapedidos.get(c));
    Listapedidos.remove(c);

}catch( Exception e){
    JOptionPane.showMessageDialog(null, "Favor de escoger una crepa");
}
```

**Número de palabras: 982 palabras**

## Referencias Bibliográficas:

- Ernesto, L. G. D. (2016, Marzo 30). Recuperado de:  
<https://www.youtube.com/watch?v=xGzeEUHcsj8&t=434s>
- ProgramacionATS. (2018, Abril 9). Recuperado de:  
<https://www.youtube.com/watch?v=RF7Ko3AgRf8>
- [Informativa C] (2018, Abril 24). Agregar, Modificar, Eliminar registro en Java Parte 5 [Archivo de video]. Recuperado de: <https://www.youtube.com/watch?v=57SYi-uhJTQ>
- Alex Pagoada (2015, Mayo 23). Leer archivos de Excel de tipo xlsx en Java. Recuperado de: <https://www.youtube.com/watch?v=nUnCSwIkss8&t=615s>