

## **Trabajo Practico Numero 4**

### **Programación Orientada a Objetos**

#### **Proyecto Candy Shop**

#### **Universidad Siglo 21**

##### **Docente:**

Ignacio Luis Picotto

##### **Alumno:**

Joaquin Padron

**Documento que explique los conceptos aplicados:**

El documento que describe la aplicación de los conceptos aplicados en el proyecto debe ser un archivo de texto Microsoft Word con la debida calidad de presentación en sus contenidos (ortografía, formatos y espaciados acordes y consistentes, enumeraciones, títulos, otros.). Se pueden utilizar herramientas que incluye el mismo procesador de texto que faciliten la comprensión de los contenidos expuestos, por ejemplo: utilización de tablas o inserción de imágenes. El documento debe indicar los integrantes que componen el grupo de trabajo.

**Consigna:**

En esta última instancia del trabajo práctico, cada alumno deberá trabajar en forma individual sobre el proyecto Java definido en la instancia anterior agregando la aplicación de los siguientes conceptos de programación orientada a objetos: polimorfismo, manejo de excepciones, interfaces. Esta última entrega del trabajo práctico consiste en la definición final del proyecto Java que da resolución a la situación problemática planteada inicialmente.

Implementar la aplicación de los siguientes conceptos en el proyecto Java iniciado en las instancias anteriores del trabajo práctico:

- Polimorfismo.
- Manejo de excepciones.
- Uso de interfaces en Java.
- Manipulación de archivos en Java (opcional).

Se deben definir nuevos requerimientos o mejoras al proyecto que permitan la aplicación de los conceptos mencionados anteriormente, en forma individual, tomando la base del proyecto Java definido en forma grupal en la tercera instancia del trabajo práctico.

- **Polimorfismo:**

La aplicación de polimorfismo en el proyecto Candy Shop esta en la clase MetodoDePago y las clases que la extienden PagoCredito y Pago efectivo.

Cada clase que extiende MetodoDePago implementa el método realizarPago a su manera y tomando ventaja del beneficio de poder utilizar otras clases de forma independiente.

```
package App;
public class PagoCredito extends MetodoDePago {
    2 usages
    @Override
    public double realizarPago() {
        double credito = Inventario.subtotal * 0.10;//el boton de pago con credito aumenta en un 10% el valor del subtotal
        double totalCredito = Inventario.subtotal + credito;
        return totalCredito;
    }
}
```

```
package App;

public abstract class MetodoDePago {
    2 usages 2 implementations
    public abstract double realizarPago(); //Clase abstracta
}
```

```
package App;

public class PagoEfectivo extends MetodoDePago {
    2 usages
    @Override
    public double realizarPago() {
        double efectivo = Inventario.subtotal * 0.05;//el pago con efectivo resta un 5% del valor al subtotal
        double totalEfectivo = Inventario.subtotal - efectivo;
        return totalEfectivo;
    }
}
```

- **Manejo de excepciones:**

En este programa se utilizan muy pocas excepciones ya que no se pensó de una forma de que el usuario interactúe por teclado lo que es un beneficio para evitar errores, lo que en una primera instancia parece beneficioso resulta una desventaja en este punto del trabajo dificultando la implementación de excepciones y la posibilidad de darle un uso práctico y no tanto demostrativo.

Las excepciones que se pudieron manejar fueron las que están relacionadas con eliminar el último producto agregado en la lista, el cual si se eliminaban todos los productos este método intentaba restar productos y al no haber ninguno se presentaba un error, este se manejaba con un condicional en trabajos anteriores, pero en mi caso no encontré otra implementación útil de las excepciones que agregar a los casos ya implementados en el trabajo.

Otra segunda excepción se necesitó implementar en un nuevo método para generar un archivo de texto para mostrar el recibo de la compra si se desease.

Si hay algún tipo de error durante la ejecución de tipo IOException se mostraría el error por consola.

(Este código se encuentra en la linea 281 de la clase CandyShop.java, se darán mas detalles en el apartado de “Manipulación de archivos Java”.)

```
private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) { //elimino la ultima fila si es que hay filas
    //utilizo el bloque try/catch/finally para excepciones
    // si se eliminan todos los productos se avisa que no hay mas filas a eliminar
    try {
        listaVentas.remove( index: modelo.getRowCount() - 1);
        modelo.removeRow(modelo.getRowCount() - 1);
        resetearContador();
        actualizarTabla();
    }catch (Exception ex){
        JOptionPane.showMessageDialog( parentComponent: null, message: "No existe un producto a eliminar", title: "Error",
            JOptionPane.ERROR_MESSAGE);
    }
}
```

- Uso de interfaces en Java:**

En mi caso no puede encontrar un uso de las interfaces en este caso, desde el trabajo practico numero 3 cambio la forma que los productos son almacenados y otras funciones así que debería eliminar cosas que hice en los trabajos anteriores y decidí profundizar en la manipulación de archivos.

- Manipulación de archivos Java:**

Esta función de Java para trabajar con archivos externos al programa es una de las funciones mas interesantes para mi, en mi caso lo utilice para generar un archivo de texto para que el usuario pueda imprimir el hipotético recibo de compra (este podría ser tomado por otro programa que se encargue de leer el archivo de texto e imprimirlo, enviarlo por algún medio de comunicación u otra idea) investigue el uso de vectores para poder imprimir información de una manera mas práctica por lo que instancié una variable llamada vectorRecibo donde se almacena el valor del vector que permite mostrar el contenido de la tabla que muestra los productos, precio y Cantidad a comprar.

Fuera de estos apartados realice algunas mejoras para una mejor experiencia de usuario, al elegir un producto este se mantiene en la misma elección así no se selecciona solo el producto “Milka” y otros arreglos menores, el botón que hace el llamado al método “jButton1ActionPerformed” tuve que dejarlo como esta aunque

prefería llamarlo “generarRecibo” aunque no entendí por que tenia tantos problemas con NetBeams, intelIJIDE y Eclipse ni funcionaba ademas las veces que quise editar el apartado visual se me generaban métodos nuevos al mover los botones y decidí no tocarlos.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) throws IOException {
    //utilizo el bloque try/catch/finally para excepciones
    //Realizo escritura de archivos
    FileWriter archivoEntrada = null;
    PrintWriter archivoSalida;
    try {
        archivoEntrada = new FileWriter( fileName: "recibo.txt");
        archivoSalida = new PrintWriter(archivoEntrada);
        archivoSalida.println("Usted compro: ");
        archivoSalida.println("");
        archivoSalida.println("Producto , Precio, Cantidad : ");
        archivoSalida.println("");
        for(int i=0; i<vectorRecibo.size();i++) {
            archivoSalida.println(vectorRecibo.get(i));
        }
        archivoSalida.println("");
        archivoSalida.println("Por un Total de:");
        archivoSalida.println(txtTotal.getText());
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    } finally {
        assert archivoEntrada != null;
        archivoEntrada.close();
    }
};
```