



Dolce Alba

Avance de Proyecto



Universidad
Tecmilenio.



Java

Diseñado por los programadores expertos:

- Luis Fernando Núñez Díaz
- Ricardo Baranda Cisneros
- Bryan David Mariñelarena Ponce



Introducción

Avance de Proyecto



La intención es desarrollar un programa en Java que simule un sistema de gestión de tareas para una pequeña empresa. Este sistema utilizará diferentes estructuras de datos (pilas, colas y listas) para manejar tareas asignadas a distintos departamentos. Así como el objetivo principal que pretende implementar y manipular diferentes estructuras de datos en Java, así como entender y analizar el uso de las aplicaciones prácticas de pilas, colas y listas en el manejo de información para desarrollar habilidades de programación y resolución de problemas en un contexto empresarial simulado.

En este planteamiento nuestro enfoque de empresa elegida fue el de un restaurante llamado “Dolce Alba” en italiano, traducido al español como “Dulce Amanecer”, para la implementación de las listas; se usó un menú para la administración de platillos que a su vez está dividido en categorías, mientras que para las colas con prioridad se realizó otro menú para organizar y atender a los clientes, en este caso se diferencia si un cliente posee una reservación o no , básicamente se les da más prioridad a atender un cliente que tiene su reservación, y por último y no menos importante están las pilas, que se usarán para gestionar el almacén de los insumos requeridos para la realización de los exquisitos platillos que se sirven en el restaurante, la pila en sí se usa para almacenar como primeros insumos aquellos que tienen la fecha más cercana a caducar y en el tope aquellos que están más próximos a caducar para usarlos lo más pronto posible y reducir pérdidas.





Menú Principal del Programa

Avance de Proyecto



```
★★ DOLCE ALBA ★★

1. Gestión de Menú █ (Listas)
2. Gestión de Clientes █ (Colas)
3. Gestión de Inventario █ (Pilas)
0. Salir █

Seleccione una opción:
```

En toda la ejecución del programa se estuvo utilizando una clase llamada Colores la cual hizo posible la implementación de diversos colores por medio de códigos **ANSI** almacenadas en cadenas de texto tipo **String** guardadas de manera pública y estática al igual que de forma **final** para que su valor sea inmutable.

```
// Método para limpiar la pantalla de la consola
public static void limpiarPantalla() {
    System.out.print("\u001B[2J");
    System.out.flush();
}
```

Se implementó un menú principal en la clase principal del programa para mostrar las distintas tareas que se pueden hacer en él y se utilizaron símbolos para mejorar el diseño visual del usuario.

```
// Clase para definir colores en la consola
class Colores {
    public static final String AZUL = "\u001B[34;1;5m";
    public static final String ROJO = "\u001B[31;1;3m";
    public static final String BLANCO = "\u001B[37;1;3m";
    public static final String VERDE = "\u001B[32;1;3m";
    public static final String MORADO = "\u001B[35;1;3m";
    public static final String CIAN = "\u001B[36;1;3m";
    public static final String AMARILLO = "\u001B[33;1;3m";
    public static final String RESET = "\u001B[0m";
}
```

Se creó un método de limpieza el cual tenía como función limpiar la pantalla de la consola para evitar tener una acumulación excesiva de operaciones que se realizaron anteriormente.

Durante toda la ejecución del programa, el menú principal se estuvo gestionando por medio de un método almacenado dentro de la clase principal del programa para tener un mejor control a la hora de estar desplegándolo

```
Mostrar en pantalla el menú principal del programa
private static void mostrarInterfaz() {
    int opcion;
    do {
        System.out.println(Colores.AZUL + " _____" + Colores.RESET);
        System.out.println(Colores.AZUL + "| ★ ★ DOLCE ALBA ★ ★ |" + Colores.RESET);
        System.out.println(Colores.AZUL + " _____" + Colores.RESET);
        System.out.println(Colores.BLANCO + "1. Gestión de Menú ■ (Listas) ■");
        System.out.println(Colores.BLANCO + "2. Gestión de Clientes ■ (Colas) ■");
        System.out.println(Colores.BLANCO + "3. Gestión de Inventario ■ (Pilas) ■");
        System.out.println(Colores.BLANCO + "4. Procesar Orden Completa ■");
        System.out.println(Colores.BLANCO + "5. Ver Estado del Restaurante ■");
        System.out.println(Colores.BLANCO + "0. Salir ■" + Colores.RESET);
        System.out.print(Colores.MORADO + "\nSelecione una opción: " + Colores.RESET + Colores.BLANCO);

        opcion = scanner.nextInt(); // Leer el buffer
    } while(opcion != 0);
    scanner.nextLine(); // Limpiar buffer

    switch (opcion) {
        case 1:
            limpiarPantalla();
            gestionarMenu();
            break;
        case 2:
            limpiarPantalla();
            Colas.gestionarClientes();
            break;
        case 3:
            limpiarPantalla();
            System.out.print(Colores.MORADO + "Inserta la capacidad del inventario (número de objetos almacenables): " + Colores.RESET + Colores.BLANCO);
            int capacity = scanner.nextInt();
            Pilas.pila = new Pilas(capacity);
            limpiarPantalla();
            pila.gestionarInventario();
            break;
        case 4:
            limpiarPantalla();
            //procesarOrdenCompleta();
            break;
        case 5:
            limpiarPantalla();
            //verEstadoRestaurante();
            break;
        case 0:
            System.out.println(Colores.VERDE + "\n¡Gracias por usar el sistema de Dolce Alba!" + Colores.RESET);
            break;
        default:
            System.out.println("Opción inválida. Intente nuevamente.");
    }
}
```



Implementación de Listas

Avance de Proyecto



★★ GESTIÓN DE MENÚ (LISTAS) ★★
95

```
1. Mostrar menú completo █
2. Mostrar por categoría █
3. Buscar platillo █
4. Agregar platillo +
5. Eliminar platillo █
0. Volver al menú principal █

Seleccione una opción: █
```

A lo largo del programa hubo casos en los que en algunos métodos tales como: **eliminar** y **buscar**, se tuvo que implementar la función `"replaceAll("\u001B\\[[;\\d]*m", "")"` al momento de estar evaluando o trabajando con variables, lo que hacia dicha función era eliminar toda referencia de los códigos **ANSI** que fueron utilizados para implementar los colores para así no tener errores de ejecución o evaluación de dichas variables

Se implementó un menú en el apartado de listas para gestionar las distintas tareas que el usuario puede realizar dentro de ella, cada opción está ligada con su respectivo método dentro de la clase menú

```
// Método para eliminar un platillo del menú
public boolean eliminar(String nombre) {
    return menu.removeIf(platillo -> platillo.getNombre().replaceAll("\u001B\\[[;\\d]*m", "").equalsIgnoreCase(nombre));
}

// Método para buscar un platillo en el menú
public Platillo buscar(String nombre) {
    for (Platillo platillo : menu) {
        String nombreLimpio = platillo.getNombre().replaceAll("\u001B\\[[;\\d]*m", "");
        if (nombreLimpio.equalsIgnoreCase(nombre)) {
            return platillo;
        }
    }
    return null;
}
```

```
// Método para gestionar el menú del restaurante
private static void gestionarMenu() {
    int opcion;

    do {
        System.out.println(Colores.AZUL + "\n★ *      GESTIÓN DE MENÚ (LISTAS) *");
        Colores.RESET);
        System.out.println(Colores.AZUL + "-----" + Colores.ROJO
+ "95" + Colores.AZUL + "-----" + Colores.RESET);
        System.out.println(Colores.BLANCO +
"█" + Colores.RESET);
        System.out.println(Colores.BLANCO + "█ 1. Mostrar menú completo █");
        System.out.println("█ 2. Mostrar por categoría █");
        System.out.println("█ 3. Buscar platillo █");
        System.out.println("█ 4. Agregar platillo +");
        System.out.println("█ 5. Eliminar platillo █");
        System.out.println("█ 0. Volver al menú principal █" +
Colores.RESET);
        System.out.println(Colores.BLANCO +
"█" + Colores.RESET);
        System.out.print(Colores.MORADO + "\nSeleccione una opción: " +
Colores.RESET + Colores.BLANCO);

        opcion = scanner.nextInt();
        scanner.nextLine();

        switch(opcion) {
```

El apartado para mostrar la interfaz de la gestión del menú fue instanciada dentro de la misma clase principal (**Main.java**), en forma de método, los demás métodos utilizados para las listas fueron instanciados dentro de una clase menú

Se creo una clase platillo la cual trabaja en colaboración con la clase menú a la hora de querer añadir un nuevo platillo al menú.

Se utilizo un constructor el cual recibe todos los datos del plátanos y unos **getters** y **setters** los cuales también se utilizan para la búsqueda de plátanos por nombre o categoría

```
// Clase para representar un platillo en el menú
class Platillo {
    private String nombre;
    private String descripcion;
    private double precio;
    private String categoria;

    // Constructor de la clase Platillo
    public Platillo(String nombre, String descripcion, double precio, String categoria) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.precio = precio;
        this.categoria = categoria;
    }

    // Getters y Setters
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```



Implementación de Colas

Avance de Proyecto



★★ GESTIÓN DE CLIENTES (COLAS) ★★

1. Agregar cliente
2. Mostrar cliente al frente de la cola
3. Atender cliente al frente de la cola
4. Eliminar cliente por nombre
5. Ver toda la cola
0. Salir

Selecciones una opción: ■

El menú de implementación de colas permite agregar clientes, mostrarlos y dar prioridad al cliente que este enfrente de la cola así como eliminar clientes y ver toda la cola como se puede apreciar en la imagen.

```
class Node {  
    String name; // nombre del cliente  
    int priority; // 1 - con reservación (mas urgente), 2 - sin reservación  
    Node next;  
  
    public Node(String name, int priority) {  
        this.name = name;  
        this.priority = priority;  
        this.next = null;  
    }  
}
```

Para poder definir y estructurar este menú primero se generó la clase “Node” para poder dar prioridad a clientes con reservación, así como agregar a los que no la tienen para agilizar el proceso y que todos los clientes que entran estén en la cola formando así la lista enlazada.

```
// Cola con prioridad para clientes  
class PriorityQueue {  
    private Node front; // frente de la cola (mayor prioridad al frente)  
  
    public PriorityQueue() {  
        front = null;  
    }  
  
    // Insertar cliente en función de su prioridad (1 mejor que 2)  
    public void enqueue(String name, int priority) {  
        Node newNode = new Node(name, priority);  
  
        if (front == null || priority < front.priority) {  
            newNode.next = front;  
            front = newNode;  
        } else {  
            Node temp = front;  
            while (temp.next != null && temp.next.priority <= priority) {  
                temp = temp.next;  
            }  
            newNode.next = temp.next;  
            temp.next = newNode;  
        }  
        System.out.println(Colores.CIAN + "Agregado: " + Colores.BLANCO + etiqueta(newNode));  
    }  
  
    // Atiende (elimina) al cliente del frente  
    public void dequeue() {  
        if (front == null) {  
            System.out.println("Cola vacía");  
        } else {  
            Node temp = front;  
            front = front.next;  
            System.out.println(Colores.CIAN + "Atendido: " + Colores.BLANCO + etiqueta(temp));  
            temp = null;  
        }  
    }  
}
```

También se implementaron métodos como enqueue (Inserta un cliente en el lugar correcto según la prioridad), dequeue (atiende al cliente enfrente eliminándolo de la cola) y removeByname (elimina a un cliente por nombre en cualquier posición que distinguen por prioridad por 1, siendo esta el primer cliente que entro a la cola o que cuenta con reservación y el 2 siendo el de menos prioridad, apoyándose de la clase “Node” establecida previamente

```

// Ver próximo a atender
public void peek() {
    if (front == null) {
        System.out.println(Colores.ROJO + "La cola está vacía" + Colores.RESET);
    } else {
        System.out.println(Colores.AMARILLO + "Siguiente: " + Colores.BLANCO + etiqueta(front));
    }
}

// Mostrar toda la cola
public void display() {
    if (front == null) {
        System.out.println(Colores.ROJO + "La cola está vacía" + Colores.RESET);
        return;
    }
    Node temp = front;
    System.out.println(Colores.CIAN + "-- Cola de clientes --" + Colores.RESET);
    while (temp != null) {
        System.out.println(" " + etiqueta(temp));
        temp = temp.next;
    }
}

public boolean isEmpty() { return front == null; }

private String etiqueta(Node n) {
    String tipo = (n.priority == 1) ? Colores.VERDE + "Con reservación" + Colores.RESET : Colores.ROJO + "Sin reservación" + Colores.RESET;
    return String.format(Colores.BLANCO + "%-15s | %s" + Colores.RESET, n.name, tipo);
}

```

En esta parte del código se estableció “peek” la cual muestra al próximo cliente que será atendido, “display” Recorre toda la lista y muestra la cola completa con un buen formato y “etiqueta” funciona como el método auxiliar para mostrar al cliente por su nombre y si tiene reservación o no aplicando colores dependiendo de la respuesta.

Y por último la clase colas que da a entender la estructura final de esta sección del código que a su vez es la clase encargada de la interfaz con el usuario, es capaz de ejecutarse en un bucle infinito hasta que el usuario elija “0. Salir”, se conforma por sus opciones que son:

- 1- Agregar cliente (pide nombre del cliente, pregunta si tiene reservación, inserta la respuesta en la cola enqueue)
- 2- Mostrar cliente al frente (Usa peek () para mostrar quien sigue
- 3- Atender cliente (Llama a dequeue () para quitar al cliente del frente)
- 4- Eliminar cliente por nombre (Pide un nombre y usa RemoveByName())
- 5- Ver toda la cola (Llama a display() y muestra la cola entera)
- 0- Salir (Regresa al menú principal del programa, limpia pantalla y termina con este submenú)

```

class Colas {
    public static void gestionarClientes() {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        PriorityQueue cola = new PriorityQueue();

        while (true) {
            System.out.println(Colores.AZUL + "\n★ ★      GESTIÓN DE CLIENTES (COLAS)      ★ ★" + Colores.RESET);
            System.out.println(Colores.AZUL + "-----" + Colores.ROJO + "98" + Colores.AZUL + "-----" + Colores.RESET);
            System.out.println(Colores.BLANCO + "-----" + Colores.RESET);
            System.out.println(Colores.BLANCO + " 1. Agregar cliente");
            System.out.println(" 2. Mostrar cliente al frente de la cola");
            System.out.println(" 3. Atender cliente al frente de la cola");
            System.out.println(" 4. Eliminar cliente por nombre");
            System.out.println(" 5. Ver toda la cola");
            System.out.println(" 0. Salir");
            System.out.println("-----" + Colores.RESET);
            System.out.println(Colores.BLANCO + "-----" + Colores.RESET);
            System.out.print(Colores.MORADO + "\nSelecciones una opción: " + Colores.RESET + Colores.BLANCO);

            String op = sc.nextLine().trim();

            switch (op) {

```



Pilas

Avance de Proyecto



Como se mencionó anteriormente, la implementación de las pilas fue para la administración del inventario de insumos del restaurante, a continuación se muestran partes del código para entenderlo más a detalle.

```

import java.util.Scanner;
class Pila1{
    private String[] data;
    private int top;
    private int capacity;
    public Pila1(int capacity){
        this.capacity = capacity;
        data = new String [capacity];
        top = -1;//empty stack
    }
    public boolean isEmpty(){
        return top == -1;
    }
    public boolean isFull(){
        return top == capacity - 1;
    }
}

```

A continuación, se muestra el método “push” para agregar los elementos primero comprobando si está lleno para lanzar un mensaje de error, de igual manera con los métodos “pop” para remover el último elemento agregado, “peek” para mostrar el último elemento agregado y “show” para mostrar todos los elementos agregados, usando a su vez “isFull” o “isEmpty” según sea el caso, y por último la inicialización del menú decorado con emojis sacados de una página de internet, además, con una opción para regresar al menú principal.

Para empezar se importó la respectiva librería para escanear las opciones introducidas por el usuario, después se agregaron las variables para el funcionamiento como el arreglo que usará “Strings”, también su capacidad y el “top” es decir el el último elemento agregado en la pila, después dos métodos tipo “boolean” para determinar si la pila está vacía o llena.

```

public String push(String x) {
    if(isFull()){
        System.out.println("Pila llena");
    }
    return data[++top] = x;
}

public void gestionarInventario() {
    Scanner sc = new Scanner(System.in);
    boolean condition = true;

    while (condition) {
        System.out.println(Colores.AZUL + "\n★ ★");
        System.out.println(Colores.AZUL + "_____");
        System.out.println(Colores.BLANCO + "████");
        System.out.println(Colores.BLANCO + "|| 1. Añadir");
        System.out.println("|| 2. Retirar insumo —");
        System.out.println("|| 3. Mostrar el último inventario");
        System.out.println("|| 4. Mostrar inventario");
        System.out.println("|| 0. Volver al menú principal");
        System.out.println(Colores.BLANCO + "████");
        System.out.print(Colores.MORADO + "\nSelección: ");
        int option = sc.nextInt();
    }
}

```

**Gracias por escoger el sistema de
Dolce Alba.**



© Derechos reservados Dolce Alba Systems.