Master Chef Colombia APP

Rojas Herreño Cristian Alexis
cristian.rojas.herreno@pi.edu.co
Politécnico Internacional

Resumen – Proyecto Inspirado en el programa de televisión de cocina llamado master chef Colombia, se trata de una aplicación en desarrollo en java haciendo uso de la Programación orientada de objetos y su 4 pilares que son Herencia,polimorfismo,abstracción y encapsulamiento ,en este programa se establece un registro e inicio de sesión de sesión ,aparte a sus 8 requerimientos donde el usuario puede visualizar y agregar información aparte realizar algunas funciones respecto al programa de cocina que hay dentro del programas con facilidad ,gracias a la elaboración intuitiva del programa e instrucciones claras para que el usuario no tenga perdida al hacer uso del mismo.

Palabras Clave: ABSTRACCION, POO, GUI.

Abstract -This Project is inspired by the televisión cooking show "Master Chef Colombia" and involves the development of a Java application using Object-Oriented Programming (OOP) and its four pillars: inheritance, polymorphism, abstraction, and encapsulation. The program incorporates user registration and login features along with eight distinct requirements. Users can easily view and input information, as well as perform various culinary-related functions within the program. The intuitive design and clear instructions ensure a user-friendly experience.

Keywords: ABSTRACTION, OOP, GUI.

I. INTRODUCCION

Este Proyecto de estudio Basado en el programa de televisión de cocina Master Chef ,se trata en el desarrollo de una aplicación en java sobre temas del programa, donde mediante del uso del paradigma de programación orientada de objetos, El usuario podrá registrarse en el programa y conocer información del programa desde cuantos participantes hay en el programa como los ganadores de sus temporadas, además de poder conocer algunas recetas típicas de Colombia y agregar recetas de su gusto y que quiera guardar en el programa , aparte de realizar y conocer el proceso de cocina de algunas recetas típicas de Colombia ,siendo esta aplicación de buen gusto para los amantes del programa de televisión Master Chef Colombia y amantes de la Gastronomía, también se conocerán los conceptos fundamentales de programación orientada a objetos y el usuario conozca como se realizó esta app .

II. Definiciones De Los Conceptos

A. ¿Qué es la Programación Orientada a Objetos?

Cuando hablamos de La programación orientada a objetos Nos referimos a un paradigma de programación establecido donde la base de esto es programar cosas de la vida real ósea objetos, utilizando su información (variables)y haciendo métodos (comportamientos)de tales objetos en forma de código, haciendo uso de sus 4 pilares que son Herencia, polimorfismo, abstracción y encapsulamiento los cuales van a ser explicados y como fueron usados en este programa, durante vayas leyendo este documento.

B. ¿Qué es la herencia y donde se encuentra en mi código?

La herencia es uno de los pilares de la programación orientada a objetos en la cual como en la vida real, una clase hija(objeto)podrá heredar Atributos de una clase padre, al heredar esto la clase hija podrá hacer uso de las variables(información) y métodos (Comportamientos) de su clase padre, aquí un ejemplo de herencia y cómo funciona en el código del programa Master chef Colombia APP:

```
public abstract class Persona {
    private String Nombre;
    private String Nacionalidad;
    private int Edad;
    private String Profesion;
```

FIGURA 1 HERENCIA 1.

Lo que vemos en la imagen es la creacion de una clase padre llamada Persona en la cual creamos 4 variables (Informacion)que son Nombre,Nacionalidad,Edad y Profesion las cuales mediante la herencia se podran usar en otra clase sin necesidad de crearlas de nuevo , a continuacion veremos como funciona en la otra clase:

```
public class Presentadora extends Persona {//aplicar

    public Presentadora(String Nombre, String Nacior
        super(Nombre,Nacionalidad,Edad,Profesion);
    }
```

FIGURA 2 HERENCIA 2.

aquí tenemos una clase hija llamada Presentadora Donde para heredar los atributos de la clase padre se usa la palabra extends y el nombre de la clase padre en este caso Persona haciendo un llamado a esta ,seguido a esto se crea un constructor para inicializar las variables de la clase persona ,se usa la palabra super para llamar el constructor de la clase padre y añadir los valores en orden a cada variable heredada, así aplicando una reutilización de código y haciendo mas optimo el proceso de programación.

C. ¿Qué es la Abstracción y donde se encuentra en mi código?

La abstracción es otro pilar fundamental de la programación orientada a objetos la cual consiste en que una clase o interfaz abstracta pueda interactuar sin tener conocimientos de su funcionalidad, en la cual la abstracción al uno querer hacer un objeto de la vida real se basara en las características mas importantes de tal objeto y que las pueda diferenciar de las demás, un ejemplo en nuestro código seria como la clase Persona que creamos y mostramos anterior mente siendo una clase abstracta definimos las variables mas importantes que tiene una persona y que las diferencia una de cada una en este caso fue ,nombre, Nacionalidad ,edad y profesión y que usamos en la otra clase presentadora que es una persona así mismo podríamos usarla en otra clase que se llame por ejemplo estudiante que también es una persona y tiene estas mismas características pero es una persona diferente a la presentadora ,siendo la abstracción una herramienta importante para gestionar la complejidad de los sistemas de software .

Aquí una imagen de referencia de una parte del código donde se crea una interfaz abstracta creando métodos sin saber el funcionamiento de estos mismos.

```
// Definición de la interfaz para la Lista Abstracta
interface ListaAbstractaJurados<E> {
    void agregar(E elemento);  // Método para agregar
    E obtener(int indice);     // Método para obtener
    int tamaño();              // Método para obtener
}
```

[1]FIGURA 3 ABSTRACCION.

D. ¿Qué es el polimorfismo y donde se encuentra en mi código?

El polimorfismo es otro pilar fundamental de la programación orientada a objetos el cual se refiere a una característica de la P.O.O que permite que una clase responda a diferentes métodos (comportamientos o acciones) ya establecidos de acuerdo a sus necesidades aquí un ejemplo de polimorfismo dentro de nuestro código:

```
recetascol.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        // Se muestran las recetas directamente en el JTextArea sin agregar a ListaRecetas

         recetasTextArea.setText("");

        VerRecetas(recetasTextArea);


    }
});

agregarrec.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        AgregarRecetas();


}
});

tienda.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        TiendaGUI.main(args);
}
});
```

FIGURA 4 POLIMORFISMO.

aquí tenemos un ejemplo de polimorfismo en la cual dentro de las acciones de los JButtons hacemos el llamado de metodos para que se ejecuten al clickear ese boton.

E. ¿Qué es el encapsulamiento y donde se encuentra en mi código ¿

El encapsulamiento es el ultimo pilar importante de la programación orientada a objetos en el cual podremos establecer el nivel de acceso a atributos o métodos establecidos en una clase, y establecer si es accesible o no desde otro elemento, mediante el uso de las palabras public, private, protected y package private, aquí un ejemplo de cómo se usa en nuestro código.

```
class Recetas{
private String NombreReceta;
private String OrigenReceta;
private String TiempoCocina;
private String Ingredientes;
```

FIGURA 5 ENCAPSULAMIENTO.

Aquí el ejemplo donde establecimos el nivel acceso de las variables de esta clase llamada recetas usando un private donde solo se puedan acceder a tales atributos mediante métodos públicos que son get y set.

F. ¿Qué es la modelo vista controlador y como lo aplico en mi proyecto?

La modelo vista controlador es un patron de arquitectura de software que se utiliza en aplicación de interfaz con el usuario como esta que estamos realizando y que esta en desarrollo y por el momento no tiene una interfaz gráfica, entonces este patron se divide en 3 lo primero el **Modelo**, el modelo es prácticamente todo lo que escribimos el código, la información y comportamientos que queremos que el programa realice cuando el usuario use este mismo, segundo, la **Vista ,**esta se encarga de interactuar con el usuario mediante una interfaz grafica mostrándole los datos y comportamientos establecidos en el modelo,tercero,el **Controlador** este es el intermediario entre el **Modelo** y la **Vista** ,recibe las interacciones del usuario a la información que brinda el

modelo mediante la vista y asi cada que el usuario seleccione una opción del programa el controlador le avise al modelo y este refleje lo que ha seleccionado el usuario. Aquí una imagen interactiva del funcionamiento de la modelo vista controlador:

Modelo:

```
public class MenuPrincipal extends JFrame {
    static JTabbedPane tabbedPane;//creamos un tabbedpane llamado awsi mismo

    public MenuPrincipal(){//constructor
        setTitle(title: "Master Chef Colombia App");//titulo del tabbedpane
        setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
        setSize(width: 800, height: 600);//tamaño  tabbed pane

        tabbedPane = new JTabbedPane();
        add(comp: tabbedPane);
        String[] args = null;

        Recetastipicas.main(args);
        CocinaGUI.main(args);


        setLocationRelativeTo(c: null); // Centrar la ventana en la pantalla
    }
```

FIGURA 6 MODELO.

Vista:

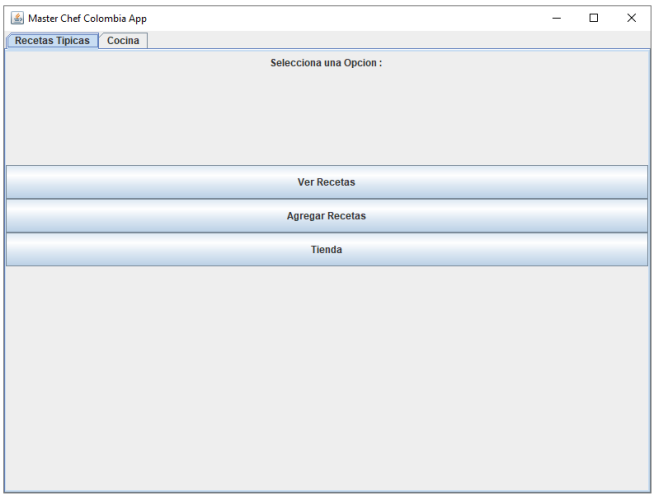

FIGURA 7 VISTA.

Controlador:

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        MenuPrincipal app = new MenuPrincipal();//instanciamos la clase como app
        app.setVisible(b: true);
    });
}
```

FIGURA 8 CONTROLADOR.

III. Diseño de Aplicación

A. Definición de Requerimientos Funcionales (Historias de Usuario).

Un requerimiento funcional o historias de usuario son los comportamiento o acciones que establecemos para nuestro programa mediante una planeación de funcionamiento de nuestra app o programa.

En este proyecto establecimos 8 requerimientos funcionales, los cuales están destinados para los usuarios amantes de la cocina y del programa master chef, mostrando información y permitiendo al usuario interactuar con el programa de manera eficientes, aparte de un filtro de seguridad con el registro e inicio de sesión en el programa, a continuación, se mostrará los 8 requisitos funcionales del programa:

```
System.out.println(x: "Menu Principal MasterCheff Colombia APP");
System.out.println(x: "1.Recetas Tipicas Colombianas(Agregar Receta,Tienda:(Precio de los Ingredientes)");
System.out.println(x: "2.cocina(Cocinar una Receta)");
System.out.println(x: "3.Ganadores MasterCheff Colombia");
System.out.println(x: "4.Jurados Actuales MasterCheff");
System.out.println(x: "5.Presentadora MasterCheff y cuantos participantes hay en master cheff");
System.out.println(x: "6.Otros Programas(mas programas de MasterCheff en Colombia)");
System.out.println(x: "7.salir");
```

FIGURA 9 REQUISITOS FUNCIONALES.

Estos 8 requerimientos están añadidos en orden, atados a un tabbedPane mostrando mediante JPanels y JFrame la información y opciones mediante Botones, JComboBox y otras más de cada requerimiento funcional.

Ya explicando como estan establecidos los requerimentos vamos a explicar uno a uno en la siguiente pagina.

Iniciaremos con el primer requerimiento establecido que es:
Recetas típicas colombianas – en este requerimiento nos encargamos de mostrar información de recetas típicas de Colombia con sus ingredientes tiempo de preparación y origen de la receta, aparte cuando el usuario seleccione esta opción del menú principal Tendrá tres opciones, la primera que es el **primer requerimiento funcional** que es mostrar recetas típicas de Colombia, en la manera que realizamos esta opción fue creando una clase principal llamada **Recetas típicas** y otra clase secundaria dentro de la misma página de código llamada **Recetas** , en la cual establecimos las variables(Información) que vamos a usar para la Clase **Recetas típicas** continuación una imagen de la creación de la clase **Recetas** y sus variables en el código **:**

```
class Recetas{
private String NombreReceta;
private String OrigenReceta;
private String TiempoCocina;
private String Ingredientes;
```

FIGURA 10 VARIABLES PRIMER REQUERIMENTO.

Aplicando el encapsulamiento a las variables de la clase, seguido a eso inicializar cada variable y el uso de métodos get y set (Mostrar y Editar Información de la variable) en estas mismas, también la creación de un método dentro de esta clase establecido para mostrar información de las variables en un Testarea, este método se llama **Mostrar Recetas** y a continuación su imagen dentro del código:

```
public void MostrarRecetas(JTextArea recetasTextArea) {


    recetasTextArea.append(str:"\n_____________________RECETA_____________________");
    recetasTextArea.append(str:"                                                  ");
    recetasTextArea.append("\nNombre de la receta: " + NombreReceta + "\nOrigen de la Receta: " + OrigenReceta +
            "\nTiempo de Preparacion: " + TiempoCocina + "\n Ingredientes: " + Ingredientes);

    recetasTextArea.append(str:"\n                                                ");
}
```

FIGURA 11 METODO MOSTRAR RECETAS.

Ya teniendo sus variables y métodos establecidos nos vamos A la clase principal **Recetas típicas,** donde para poder guardar información de las variables crearemos una lista llamada **ListaRecetas** que va a llamar a la clase **Receta** para que la lista agregue información a las variables de esta misma, y creación de un JTextArea para agregar las listas anteriores imagen de la creación de la lista y el JTextArea:

```
public class Recetastipicas{
private static List<Recetas> ListaRecetas = new ArrayList<>();
private static JTextArea recetasTextArea = new JTextArea();
```

FIGURA 12 CREACION LISTA Y JTEXTAREA.

por segunda opción y **segundo requerimiento funcional** del programa la opción de **Agregar Receta** donde el usuario podrá ingresar una receta usando las variables ya establecidas en la clase **Recetas** y agregándolas a la lista **Lista Recetas.**

Por tercera opción y **tercer requerimiento funcional** del programa la opción de **Tienda** donde el usuario podrá ver información de los precios de los ingredientes mas comunes y usados en las recetas agregadas en el programa, ya terminando las tres opciones una última de volver al menú principal**.**

Ya volviendo al menú principal el **Cuarto requerimiento funcional** y segunda opción del menú principal es **Cocina** donde se dará la opción de elegir recetas y de una manera interactiva el usuario podrá ser parte del paso a paso del proceso de preparación de la receta.

El **quinto requerimiento funcional** y tercera opción del menú principal es **ganadores MasterCheff Colombia** donde el usuario podrá conocer información principal de los ganadores de algunas de las temporadas del programa.

El **Sexto requerimiento funcional** y cuarta opción del menú principal es mostrar información de los **jurados actuales** del programa MasterCheff.

El **Séptimo requerimiento funcional** y quinta opción del menú principal **presentadora Master Chef y cuantos participantes hay en Master Chef** donde se mostrará información de la presentadora y la cantidad de personas que participan en el programa.

El **Octavo requerimiento funcional** y sexta opción del menú principal es Otros Programas Donde el usuario conocerá los otros realitys de master chef en Colombia, así conociendo sus años de transmisión y temporadas, y última opción de salir que terminará con la ejecución del programa.

ID. HISTORIA DE USUARIO	ROL	CARACTERISTICAS /FUNCIONALIDADES	RAZON/ RESUELTO	CRITERIO DE ACEPTACION
ID 01	CLIENTE	Quiero ver una lista recetas típicas colombianas.	Para poder conocer el listado de recetas típicas colombianas.	Muestra de información importante de las recetas típicas.
ID 02	CLIENTE	Quiero agregar mis recetas a la lista de recetas típicas colombianas.	Para Poder agregar mis conocimientos de otras recetas al listado de recetas típicas colombianas.	Agregar recetas y actualizar el listado el listado de recetas típicas colombianas.
ID 03	CLIENTE	Quiero conocer el precio de ingredientes importantes de las recetas	Para poder ver y conocer el precio de los ingredientes más usados.	Muestra una tabla con el nombre y precio del ingrediente.
ID 04	Cliente	Quiero conocer y participar del proceso de cocina de las recetas típicas colombianas.	Para realizar el proceso de cocina de cualquier receta.	Muestra de pasos uno por uno en el proceso de cocina de la receta.
ID 05	cliente	Quiero saber quienes han sido los ganadores de master Chef Colombia.	Para Realizar consulta de los ganadores del programa master chef Colombia.	Desplegué de información de los ganadores del programa master chef Colombia.
ID 06	cliente	Quiero saber información sobre los jurados de master chef Colombia.	Para Realizar consulta de los jurados de master chef Colombia.	Despliegue de información sobre los jurados de master chef Colombia.
ID 07	cliente	Quiero saber quien es la presentadora de master chef y cuantas personas participan en el programa.	Para Realizar consulta de información de la presentadora y numero de participantes del programa.	Despliegue de información de la presentadora y numero de participantes del programa.
ID 08	cliente	Quiero saber si hay mas programas relacionados a este, y su número de temporadas y años de transmisión.	Para conocer información sobre otros programas de master chef.	Muestra una ventana con información de otros programas de master chef.

TABLA N 1.

B. Definición de requerimientos No Funcionales.

Los requerimientos no funcionales son aquellas características y propiedades que se relacionan con el desempeño del sistema y su calidad del mismo.

PROGRAMA DESARROLLADO EN UN SISTEMA OPERATIVO WINDOWS 10 X64 BITS	
Requisitos recomendados Windows:	
Procesador	procesador con capacidad de velocidad de al menos 1GHz o más.
Memoria RAM	2 GB RAM o Mas.
Sistema Operativo	Windows 10 x64 bits
Espacio Disco Duro	10 GB como mínimo.
Java Platform	JDK 20.
IDE de Desarrollo	Apache NetBeans IDE 18.

TABLA N2.

C. Definición del diagrama de arquitectura de mi proyecto.

Inicialmente el usuario ejecutara el programa ,seguido a esto el programa le envia una respuesta de registro ,usuario recibe tal respuesta en pantalla ,completara el registro y le enviara al programa los datos de registro,ya estando el usuario registrado ,el programa enviara al usuario un inisio de sesion de verificacion de registro,usuario se registra y le eenvia esta verificacion al programa ,ya teniendo esta verificacion el programa le da acceso al menu principal donde estan los requerimentos funcionales y entre esas las de finalizar informacion,cuando el usuario seleccione alguna opcion del menu principal estara recibiendo y enviando informacion al programa.

FIGURA 13 DIAGRAMA DE ARQUITECTURA.

D. Diagrama de flujo de mi proyecto.

FIGURA 14 DIAGRAMA DE FLUJO.

E. Diccionario de datos

A continuación, conoceremos la definición de las variables que hemos creado en nuestro programa.

Boolean – es un tipo de dato lógico que solo puede tener los valores true o false.

String- es un tipo de dato que construye una cadena de caracteres, y en java se utiliza para almacenar textos y se define en comillas.

Int – es un tipo de dato entero que almacena números enteros y tiene la capacidad de almacenar números negativos.

Nombre Variable	Tipo de Dato	Funcionamiento
resultado	int	Mostrar cuadro de dialogo.
UserRegistrado	boolean	Sirve para identificar el registro del usuario, mediante true o false.
Usuario	String	Recibir Y guardar el nombre de Usuario que elija el Usuario en el proceso de registro.
Password	String	Recibir Y guardar la contraseña que elija el Usuario en el proceso de registro.
ConfirmarUser	String	Utilizada en el proceso de inicio de sesión y verificar el Nombre de usuario sea el mismo que se registró.
ConfirmarPass	String	Utilizada en el proceso de inicio de sesión y verificar la contraseña sea la misma que se registró.
NombreReceta	String	Guardar y recibir información del nombre de una receta.
OrigenReceta	String	Guardar y recibir información del origen de una receta.
TiempoCocina	String	Guardar y recibir información del tiempo de preparación de una receta.
Ingredientes	String	Guardar y recibir información de los Ingredientes de una receta.
nombreReceta	String	Sirve para guardar información que se recibe del campo Nombrerec.
origenReceta	String	Sirve para guardar información que se recibe del campo Origenrec.
tiempoCocina	String	Sirve para guardar información que se recibe del campo Tiemporec.

ingredientes	String	Sirve para guardar información que se recibe del campo Ingredientesrec.
Plato	String	Guardar el nombre del plato típico.
Nombre	String	Guardar información del nombre de los ganadores de master chef.
Profesion	String	Guardar información de la profesión de los ganadores de master chef.
Edad	int	Guardar información de la edad de los ganadores de master chef.
Nacionalidad	String	Guardar información de la nacionalidad de los ganadores de master chef.
Año	int	Guardar información del año en que ganaron los ganadores de master chef.
EstadoCivil	String	Guardar información del estado civil de los ganadores de master chef.
Nombre	String	Guardar y recibir información de la nacionalidad de una persona.
Nacionalidad	String	Guardar y recibir información de la nacionalidad de una persona.
Edad	int	Guardar y recibir información de la edad de una persona.
Profesión	String	Guardar y recibir información de la profesión de una persona.
tamaño	int	Sirve para rastrear el tamaño actual de la lista abstracta.
índice	int	Sirve para saber la posición exacta de un objeto en la lista.
juradoSeleccionado	String	Sirve para almacenar el nombre del jurado seleccionado.
informacionJurado	String	Sirve para mostrar la información del método obtener información.
NombreProg	String	Guardar y recibir información de el nombre de un programa.
Temporadas	int	Guardar y recibir información del número de temporadas de un programa.
añosemision	String	Guardar y recibir información de todos los años de transmisión de un programa.

TABLA N3.

F. Diagrama de secuencia.

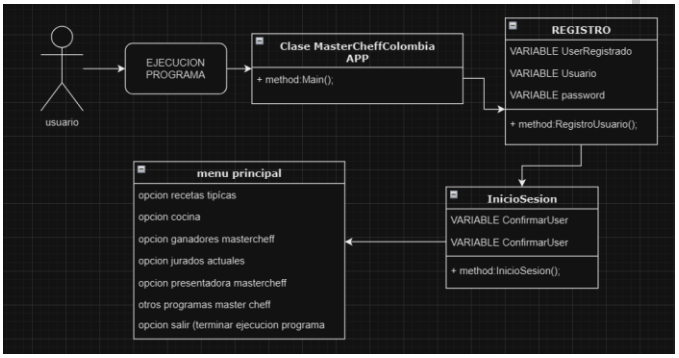

FIGURA 15 DIAGRAMA DE SECUENCIA.

G. Pseudocodigo y prueba de escritorio.

FIGURA 16 PSEUDOCODIGO.

Entrada	Salida	Funciona
Mensaje De Bienvenida Al Aplicativo.	Muestra De Mensaje De Bienvenida Al Programa Y Muestra De Paso Siguiente A Registrar.	Ok.
Ventana De Registro	Muestra Ventana De Registro Con Campos Para Ingresar Usuario Y Contraseña Y Boton Registrarse Y Filtro Para Que No Continue Hasta Que Los Dos Campos Esten Llenos.	Ok
Ventana Inicio Sesion	Muestra Ventana Inicon Sesion ,Usuario Llenara Campos Con Los Mismos Datos De Registro ,Si No Son Iguales Mensaje Que Nos Son Iguales Y Una Soluccion Por Si Usuario Olvida Datos.	Ok

Ventana Menu Principal	Mostrara Frame De Menu Principal El Cual Tiene 7 Apartados En Los Cuales Estan Los Requerimentos Funcionales.	Ok
Primer Apartado Menu Principal	Saldran Los Tres Primeros Reuisitos Funcionales Que Son Ver Recetas,Agregar Recetas Y Tienda.	Ok
Segundo Apartado Menu Principal	Mostrara El Cuarto Requisito Funcional Llamado Cocina , Muestra Paso A Paso De La Receta Que El Usuario Eliga.	Ok
Tercer Apartado Menu Principal	Mostrara El Quinto Requisito Funcional Llamado Ganadores ,Usuario Podra Conocer Quienes Fueron Los Ganadores Del Programa.	Ok
Tercer Apartado Menu Principal	Mostrara El Quinto Requisito Funcional Llamado Ganadores ,Usuario Podra Conocer Quienes Fueron Los Ganadores Del Programa.	Ok
Cuarto Apartado Menu Principal	Mostrara El Sexto Requerimento Funciopnal Llamado Jurados ,Usuario Podra Conocer Informacion De Cada Uno De Los Jurados Del Programa .	Ok
Quinto Apartado Menu Principal	Mostrara El Septimo Requerimento Funcional Llamado Presentadora ,Donde El Usuario Conocera Informacion De La Presentadora Y Numeroi De Participantes Del Programa .	Ok
Sexto Partado Menu Principal	Mostrar El Octavo Requeriimento Funcionnal Llamado Otros Programas ,Usuario Conocera Informacionm De Otros Porgramas De Mastercheff En Colombia .	Ok
Septimo Apartado Menu Principal	Mopstrar Creditos Del Creador Del Programa Y Boton Para Cerrar El Programa.	Ok

TABLA N4.

H. Definición de listas abstractas

Una lista abstracta es un concepto de estructura de datos en el cual se mantiene una colección de datos ordenadamente, definiendo unas acciones a realizar como lo es agregar eliminar o mostrar elemento por su índice (aquí una imagen del código del proyecto).

```
// Definición de la interfaz para la Lista Abstracta
interface ListaAbstractaJurados<E> {
    void agregar(E elemento);  // Método para agregar un elemento a la lista
    E obtener(int indice);     // Método para obtener un elemento en un índice específico
    int tamaño();              // Método para obtener el tamaño de la lista
}
```

```
public class JuradosMasterCheff{
    public static void main(String[] args) {
        // Crear una lista abstracta implementada con arreglo
        ListaAbstractaJurados<String> lista = new ListaArreglo<>(capacidadInicial:3);

        // Agregar elementos a la lista
        lista.agregar(elemento: " Cheff Jorge Rausch,colombiano");
        lista.agregar(elemento: " Cheff Nicolás de Zubiria,colombiano");
        lista.agregar(elemento: " Cheff Christopher Carpentier,chileno");

        // Obtener y mostrar elementos
        System.out.println(x: "------------------JURADOS MASTERCHEFF COLOMBIA--------------------");
        System.out.println("JURADO NUMERO 1: " + lista.obtener(indice:0));
        System.out.println("JURADO NUMERO 2: " + lista.obtener(indice:1));
        System.out.println("JURADO NUMERO 3: " + lista.obtener(indice:2));
        System.out.println("En Total Son " + lista.tamaño()+" Jurados.");
        System.out.println(x: "                          ");
    }
}
```

[1]FIGURA 17 LISTA ABSTRACTA.

I. Definición diagrama listas sencillas

Un diagrama de lista sencilla es una estructura de datos representada gráficamente y que los datos de los elementos se pueden almacenar en nodos, teniendo cada uno un valor diferente y el ultimo nodo apunta a null.

```
public class EjemploLista {

    public static void main(String[] args) {
        // Crear una lista de enteros
        List<Integer> listaDeEnteros = new ArrayList<>();

        // Agregar elementos a la lista
        listaDeEnteros.add(e: 5);
        listaDeEnteros.add(e: 10);
        listaDeEnteros.add(e: 15);

        // Acceder a elementos de la lista e imprimirlos
        System.out.println(x: "Elementos de la lista:");
        for (Integer numero : listaDeEnteros) {
            System.out.println(x: numero);
        }
    }
}
```

[2]FIGURA 18 LISTA SENCILLA.

J. Definición lista enlazada

Una lista enlazada organiza los objetos de manera secuencial mediante nodos que contienen valores y punteros al siguiente elemento.

```
// Crear una lista enlazada
ListaEnlazada lista = new ListaEnlazada();

// Agregar valores al principio de la lista
lista.agregarAlPrincipio(valor: 3);
lista.agregarAlPrincipio(valor: 2);
lista.agregarAlPrincipio(valor: 1);
```

[3]FIGURA 19 LISTA ENLAZADA.

K. Definición lista tipada

Una lista tipada se refiere a cuando se crea una lista limitada a un solo tipo de datos, ósea que si se creo con un tipo de dato String solo se podrá añadir información de este tipo de dato, lo mismo si se hace de tipo de dato Integer.

```java
public class Tienda {
    public static void main(String[] args) {
        //lista tipada para organizar los productos mas comunes y sus precios
        List<String> listaPrecios = new ArrayList<>();

        // Agregar elementos a la lista

        listaPrecios.add(e: "Harina de Maiz 1 Kg \nPrecio : 4.000 \n                ");
        listaPrecios.add(e: "Pechuga de Pollo \nPrecio : 15.000 \n               ");
        listaPrecios.add(e: "Papa Pastusa 1 Kg \nPrecio : 5.400 \n               ");
        listaPrecios.add(e: "Arroz Blanco Diana 500 Gr \nPrecio : 4.000 \n                ");
        listaPrecios.add(e: "Salsa De Tomate Doy Pack FRUCO 600 gr \nPrecio : 13.100 \n                ");

        System.out.println(x: "----------------------PRODUCTOS-----------------------------");
        for(String Precios : listaPrecios){
            System.out.println(x: Precios);
        }
    }
}
```

[3]FIGURA 20 LISTA TIPADA.

L. Definición Lista Genérica.

Una lista genérica es una estructura de datos la cual permite almacenar información de cualquier tipo de dato, dando flexibilidad a la hora de programar con diferentes tipos de datos y no solo un tipo de dato.

```java
public class EjemploListaGenerica {
    public static void main(String[] args) {
        // Crear una lista genérica para almacenar enteros
        List<Integer> listaDeEnteros = new ArrayList<>();

        // Agregar elementos a la lista
        listaDeEnteros.add(e: 10);
        listaDeEnteros.add(e: 20);
        listaDeEnteros.add(e: 30);

        // Imprimir los elementos de la lista
        System.out.println(x: "|Politecnico Internacional | Ejemplo Lista Generica|");
        for (int elemento : listaDeEnteros) {
            System.out.println(x: elemento);
        }
    }
}
```

[3]FIGURA 21 LISTA GENERICA.

M. Definición posición ordinal.

La posición ordinal se refiere a la posición relativa de un elemento en una secuencia o colección como una lista o array el cual inicia siempre desde 0 y así sucesivamente va aumentando la posición en cada elemento agregado.

```java
public class ListaPosicionOrdinal {
    public static void main(String[] args) {
        // Crear una lista de nombres en posición ordinal
        List<String> listaNombres = new ArrayList<>();

        // Agregar nombres a la lista
        listaNombres.add(e: "Goku");
        listaNombres.add(e: "Vegeta");
        listaNombres.add(e: "Trunks");

        // Imprimir la lista de nombres junto con su posición ordinal
        System.out.println(x: "|Politecnico Internacional | Ejemplo Lista Posicion Ordinal|");
        for (int i = 0; i < listaNombres.size(); i++) {
            String nombre = listaNombres.get(index: i);
            int posicion = i + 1; // Sumar 1 para que las posiciones comiencen desde 1
            System.out.println("Posición " + posicion + ": " + nombre);
        }
    }
}
```

[3]FIGURA 22 POSICION ORDINAL.

N. Definición Métodos de iteración.

Un método de iteración es una técnica muy utilizada que sirve para hacer recorridos sobre colecciones y ver la información de los elementos que hay en estas, va mostrando información como este definida su condición.

```java
public class EjemploMetodosIteracion {
    public static void main(String[] args) {
        // Crear una lista de números
        List<Integer> numeros = new ArrayList<>();
        numeros.add(e: 1);
        numeros.add(e: 2);
        numeros.add(e: 3);
        numeros.add(e: 4);
        numeros.add(e: 5);

        // Utilizar el método forEach para imprimir el doble de cada número
        System.out.println(x: "|Politecnico Internacional | Ejemplo Metodos de Iteraccion ForEach|");
        System.out.println(x: "Números originales y su doble:");
        numeros.forEach(numero -> {
            int doble = numero * 2;
            System.out.println(numero + " -> " + doble);
        });
    }
}
```

[3]FIGURA 23 METODOS DE INTERACCION.

O. Definición pila.

Una pila es un stack el cual consiste que, si un elemento ingresa a la pila, el siguiente va a estar por encima por lo cual cuando se vaya a sacar un elemento de la pila el que va a salir va a hacer el ultimo que se ingreso y así sucesivamente en orden del último al primer elemento que ingreso a la pila.

```java
public class EjemploPila {
    public static void main(String[] args) {
        // Crear una pila para números enteros
        Stack<Integer> pila = new Stack<>();

        // Agregar elementos a la pila (operación "push")
        pila.push(item: 10);
        pila.push(item: 20);
        pila.push(item: 30);
        System.out.println(x: "|Politecnico Internacional | Ejemplo Pila Push Pop|");
        System.out.println("Pila después de agregar elementos: " + pila);

        // Retirar elementos de la pila (operación "pop")
        int elemento1 = pila.pop();
        int elemento2 = pila.pop();

        System.out.println("Elemento retirado: " + elemento1);
        System.out.println("Elemento retirado: " + elemento2);

        System.out.println("Pila después de retirar elementos: " + pila);
    }
}
```

[3]FIGURA 24 PILA.

P. Definición de Apuntadores.

Un apuntador es una variable que guarda la dirección de memoria de otra variable de cierto tipo.

```java
public class ReferenciasExample {

    public static void main(String[] args) {
        // Crear un objeto de tipo String
        String cadena = "Politecnico Internacional, Implementacion de una Cadena";

        // Asignar una referencia al objeto a otra variable
        String otraCadena = cadena;

        // Modificar el contenido del objeto a través de la referencia
        otraCadena = otraCadena + "Politecnico Internacional, Implementacion de una cadena modificada";

        // Imprimir ambas variables
        System.out.println(x: "| Politecnico Internacional | Ejemplo Referencia |");
        System.out.println("Cadena original: " + cadena);
        System.out.println("Otra cadena: " + otraCadena);
    }
}
```

[3]FIGURA 25 APUNTADORES.

Q. Definición de Lifo.

Un Lifo es un tipo de estructura de datos donde el ultimo elemento que se añade es el primero en ser eliminado, así como en las pilas.

```java
import java.util.Stack;

public class LIFOExample {

    public static void main(String[] args) {
        // Crear una instancia de Stack para implementar LIFO
        Stack<String> lifoStack = new Stack<>();

        // Agregar elementos a la pila
        lifoStack.push(item: "Elemento 1");
        lifoStack.push(item: "Elemento 2");
        lifoStack.push(item: "Elemento 3");

        System.out.println(x: "| Politecnico Internacional | LIFO|");

        // Imprimir la pila actual
        System.out.println("Pila después de agregar elementos: " + lifoStack);

        // Retirar y mostrar el elemento superior de la pila
        String topElement = lifoStack.pop();
        System.out.println("Elemento retirado de la pila: " + topElement);

        // Imprimir la pila después de retirar un elemento
        System.out.println("Pila después de retirar un elemento: " + lifoStack);
    }
}
```

[3]FIGURA 26 LIFO

R. Definición de Colas.

Colas es una estructura de datos que sigue el principio de first in, first out, que significa que el primer elemento que se agrega es el primero en ser eliminado.

```java
public class QueueExample {

    public static void main(String[] args) {
        // Crear una instancia de Queue utilizando LinkedList
        Queue<String> cola = new LinkedList<>();

        // Agregar elementos a la cola
        cola.offer(e: "Elemento 1");
        cola.offer(e: "Elemento 2");
        cola.offer(e: "Elemento 3");

        // Mostrar la cola actual
        System.out.println("Cola después de agregar elementos: " + cola);

        // Retirar y mostrar el primer elemento de la cola
        String primerElemento = cola.poll();
        System.out.println("Primer elemento retirado de la cola: " + primerElemento);

        // Mostrar la cola después de retirar un elemento
        System.out.println("Cola después de retirar un elemento: " + cola);
    }
}
```

[3]FIGURA 27 COLAS.

S. Definición de Método de Ordenamiento.

Un método de ordenamiento es el procedimiento que se utiliza para organizar los elementos de una estructura de datos, como un array o una lista, en un orden especifico.

```java
public class EjemploOrdenamientoBurbuja {

    public static void main(String[] args) {
        int[] arreglo = {5, 2, 8, 1, 3};

        System.out.println(x: "Arreglo original:");
        imprimirArreglo(arr:arreglo);

        ordenarBurbuja(arr:arreglo);

        System.out.println(x: "\nArreglo ordenado:");
        imprimirArreglo(arr:arreglo);
    }

    // Implementación del método de ordenamiento Burbuja
    public static void ordenarBurbuja(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Intercambiar elementos si están en el orden incorrecto
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    // Método auxiliar para imprimir un arreglo
    public static void imprimirArreglo(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

[3]FIGURA 28 METODO DE ORDENAMIENTO.

T. Definición de JFRAME.

JFrame es un componente de la biblioteca swing y se usa para crear interfaces gráficas, es decir JFrame es una ventana en la que podrás agregar componentes gráficos en un programa desarrollado en java.

```java
JFrame frameagregar=new JFrame(title: "Agregar Receta");
frameagregar.setSize(width: 600, height: 600);
  frameagregar.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
  frameagregar.setLocationRelativeTo(c: null); // Centrar la ventana en la pantalla
```

FIGURA 29 JFRAME.

U. Definición JTable.

JTable es otro componente perteneciente a la biblioteca swing, que su funcionamiento es mostrar y editar información gráficamente en tablas.

```java
DefaultTableModel tableModeltienda = new DefaultTableModel();

// Agregar columnas al modelo de datos
tableModeltienda.addColumn(columnName: "Producto");
tableModeltienda.addColumn(columnName: "Precio");


// Agregar filas al modelo de datos
tableModeltienda.addRow(new Object[]{"Harina de Maiz 1 Kg", 4000});
tableModeltienda.addRow(new Object[]{"Pechuga de Pollo", 15000});
tableModeltienda.addRow(new Object[]{"Papa Pastusa 1 Kg", 5400});
tableModeltienda.addRow(new Object[]{"Arroz Blanco Diana 500 Gr ", 4000});
tableModeltienda.addRow(new Object[]{"Salsa De Tomate Doy Pack FRUCO 600 gr",13100});

JTable productos=new JTable(dm: tableModeltienda);
```

[3]FIGURA 30 JTABLE.

V. Definición JComboBox.

JComboBox es otro componente perteneciente a la biblioteca swing, sirve para establecer una lista de opciones definidas de manera gráfica y de una forma despegable.

```java
String[] options = {"Arepas Caseras", "Ajiaco", "Arroz con pollo"};
JComboBox<String> comboBox = new JComboBox<>(items: options);
```

[3]FIGURA 31 JCOMBOBOX.

W. Definición JList.

JList es otro componente perteneciente a la biblioteca swing, esta permite mostrar al usuario de manera grafica una lista de objetos con sus datos.

```java
// Crear un JList con los datos
JList<String> list = new JList<>(listData);
```

[3]FIGURA 32 JLIST.

X. Definición JTree.

JTree es otro componente perteneciente a la biblioteca swing, que muestra al usuario de manera grafica una estructura de datos jerárquica y en forma de árbol, se utiliza cuando se necesita mostrar datos con una organización jerárquica.

```java
public class JTreeExample {
    public static void main(String[] args) {
        // Crear el nodo raíz
        DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode(userObject: "Raíz");

        // Crear algunos nodos secundarios
        DefaultMutableTreeNode child1 = new DefaultMutableTreeNode(userObject: "Hijo 1");
        DefaultMutableTreeNode child2 = new DefaultMutableTreeNode(userObject: "Hijo 2");

        // Agregar los nodos secundarios al nodo raíz
        rootNode.add(newChild: child1);
        rootNode.add(newChild: child2);

        // Crear un JTree con el nodo raíz
        JTree tree = new JTree(root: rootNode);

        // Crear un JFrame para mostrar el JTree
        JFrame frame = new JFrame(title: "Ejemplo de JTree");
        frame.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);

        // Agregar el JTree al JFrame
        frame.getContentPane().add(new JScrollPane(view: tree));

        // Establecer el tamaño y hacer visible el JFrame
        frame.setSize(width: 300, height: 300);
        frame.setVisible(b: true);
    }
}
```

[3]FIGURA 33 JTREE.

Y. Definición JProgressBar.

JProgressBar es otro componente perteneciente a la biblioteca swing, esta es una barra de progreso que se representa gráficamente y sirve para representar el avance de una ejecución.

```java
progressBar = new JProgressBar(min:0, max:100);
progressBar.setValue(n: 0);
progressBar.setStringPainted(b: true);
```

[3]FIGURA 34 JPROGRESSBAR.

Z. Definición JTable + JList.

JTable + JList estos son dos componentes pertenecientes a la biblioteca swing, el primero JTable sirve para mostrar datos gráficamente en forma de tabla, y JList permite almacenar datos de una manera lineal las cuales se representan gráficamente en orden como se van agregando los elementos a la lista.

```java
public static void main(String[] args) {
    // Crear una instancia de JFrame
    JFrame frame = new JFrame(title: "Ejemplo de JTable y JList");

    // Crear datos para el JTable
    Vector<String> columnNames = new Vector<>();
    columnNames.add(e: "Nombre");
    columnNames.add(e: "Edad");
    columnNames.add(e: "Género");

    Vector<Vector<Object>> data = new Vector<>();
    Vector<Object> row1 = new Vector<>();
    row1.add(e: "Goku");
    row1.add(e: 35);
    row1.add(e: "Masculino");
    Vector<Object> row2 = new Vector<>();
    row2.add(e: "Vegeta");
    row2.add(e: 40);
    row2.add(e: "Masculino");
    data.add(e: row1);
    data.add(e: row2);

    // Crear un modelo de datos para el JTable
    DefaultTableModel tableModel = new DefaultTableModel(data, columnNames);

    // Crear un JTable con el modelo de datos
    JTable table = new JTable(dm: tableModel);

    // Crear un JScrollPane para el JTable
    JScrollPane tableScrollPane = new JScrollPane(view: table);

    // Crear datos para el JList
    String[] listData = {"Goku", "Vegeta", "Gohan", "Piccolo", "Trunks"};

    // Crear un JList con los datos
    JList<String> list = new JList<>(listData);

    // Crear un JScrollPane para el JList
    JScrollPane listScrollPane = new JScrollPane(view: list);

    // Crear un panel para organizar los componentes
    JPanel panel = new JPanel(new GridLayout(rows: 1, cols: 2));
    panel.add(comp: tableScrollPane);
```

[3]FIGURA 35 JTABLE+JLIST.

AA. Definición JButton.

JButton es otro componente perteneciente a la biblioteca swing, el cual permite mostrar gráficamente botones los cuales se pueden configurar como el nombre, tamaño, color, posición y añadirle acciones para cuando se pulse dicho botón tenga un comportamiento como abrir otra ventana.

```java
JButton salir = new JButton(text: "Salir menu principal");

salir.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        recetasTextArea.setText(t: "");
        framerecetas.dispose();
    }
});
```

[3]FIGURA 36 JBUTTON.

BB. Definición de AWT Eventos.

AWT Eventos son las acciones que ocurren durante la ejecución del programa, como clics de ratón, pulsaciones de teclas y cambios en el estado de los componentes gráficos.

```java
cerrar.addActionListener(new ActionListener(){
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(parentComponent:null,"!Hasta Luego "+Usuario+" un Gu
        System.exit(status: 0);

    }

});
```

[3]FIGURA 37 AWT EVENTOS.

CC. Definición de AWT Button.

AWT Button proporciona una clase llamada Button que se utiliza para crear un botón en la interfaz grafica de usuario el cual es un componente interactivo que puede ser presionado por el usuario para activar alguna acción.

```java
public static void VerRecetas(JTextArea recetasTextArea) {
    JFrame framerecetas = new JFrame(title: "Recetas Tipicas Colombianas");
    framerecetas.setLayout(new BorderLayout());
    JButton salir = new JButton(text: "Salir menu principal");

    salir.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            recetasTextArea.setText(t: "");
            framerecetas.dispose();
        }
    });
```

[3]FIGURA 38 AWT BUTTON.

DD. Definición de AWT Label.

AWT Label proporciona una clase llamada Label que se utiliza para mostrar texto no editable en una interfaz gráfica, la cual funciona como etiquetas para mostrar información en la interfaz gráfica.

```java
import java.awt.Frame;
import java.awt.Label;

public class EjemploLabelAWT {

    public static void main(String[] args) {
        // Crear un marco (Frame) que actuará como la ventana principal
        Frame frame = new Frame(title: "Ejemplo de AWT Label");

        // Crear una etiqueta (Label) con el texto especificado
        Label label = new Label(text: "Hola, esto es un Label de AWT.");

        // Agregar la etiqueta al marco
        frame.add(comp: label);

        // Establecer el tamaño del marco
        frame.setSize(width: 300, height: 200);

        // Establecer el diseño del marco (null significa que no se utili
        frame.setLayout(mgr:null);

        // Hacer visible el marco
        frame.setVisible(b: true);
    }
}
```

FIGURA 39 AWT LABEL.

EE. Definición de AWT TextField.

AWT TextField proporciona una clase llamada TextField que se utiliza para crear un campo de texto en una interfaz gráfica, este es un componente interactivo que permite a los usuarios ingresar y editar texto.

```java
import java.awt.Frame;
import java.awt.TextField;

public class EjemploTextFieldAWT {

    public static void main(String[] args) {
        // Crear un marco (Frame) que actuará como la ventana principal
        Frame frame = new Frame(title: "Ejemplo de AWT TextField");

        // Crear un campo de texto (TextField)
        TextField textField = new TextField(text: "Texto inicial");

        // Agregar el campo de texto al marco
        frame.add(comp: textField);

        // Establecer el tamaño del marco
        frame.setSize(width: 300, height: 200);

        // Establecer el diseño del marco (null significa que no se utiliz
        frame.setLayout(mgr:null);

        // Hacer visible el marco
        frame.setVisible(b: true);
    }
}
```

FIGURA 40 AWT TEXTFIELD.

IV. Conclusiones y Referencias.

Este emocionante proyecto inspirado en el programa de televisión Master Chef Colombia, fue una aplicación desarrollada en Java, el cual utiliza la Programación Orientada a Objetos y sus 4 pilares esenciales, donde se llegó a conocer más a fondo la funcionalidad de estos mismos en este campo, y así haciendo uso de ellos en el programa centrándose en la facilidad de uso y la experiencia del usuario.

Conociendo Los requisitos funcionales y no funcionales que se establecen en un programa, así como el diseño de la aplicación, demuestran un enfoque integral para satisfacer las necesidades de los usuarios interesados en la cocina y el programa Master Chef Colombia.

El proyecto ofrece una amplia gama de funcionalidades, las cuales se representan gráficamente gracias a las bibliotecas Swing y AWT para que nuestro programa tenga capacidad de entorno gráfico y así el usuario acceda a la visualización de información e interacción con el programa de cocina.

En resumen, este proyecto representa un esfuerzo significativo para combinar la pasión por la cocina y haciendo uso de los aspectos principales y más importantes de la programación brindando a los usuarios una aplicación atractiva y funcional relacionada con el popular programa de televisión Master Chef Colombia.

I. REFERENCIAS

[1] H. Torres. [En línea]. Available: https://github.com/Harol003/Java_Es

[2] H. Torres. [En línea]. Available: https://github.com/Harol003/Java_Estructura_Datos/blob/main/Ejempl

[3] H. Torres. [En línea]. Available: https://github.com/Harol003.

V. Lista de figuras.

VI. Lista de tablas.