Recipe Management System

1. Project Overview

The **Recipe Management System** is a Java-based application that helps manage different functions regarding the recipes. This system allows librarians to add a recipe, view recipes, delete recipes, search recipe by ingredient and then save the recipes. This project was developed as part of an OOP course to demonstrate understanding and applicationof core OOP principles.

2. Objectives

- To design and implement a simple recipe management system using Object-OrientedProgramming.
- To incorporate key OOP concepts like inheritance, polymorphism, encapsulation, andabstraction.
- To demonstrate the use of packages, exception handling, file handling, and console interactionin Java.

3. Key Features

1. Add a Recipe: Users can add new recipes with ingredients and instructions.
2. View Recipes: Users can view a list of all recipes or a specific recipe.
3. Edit a Recipe: Users can modify existing recipes.
4. Delete a Recipe: Users can remove recipes from the system.
5. Search for Recipes: Users can search recipes by ingredient or name.

4. Project Structure

1. Packages: Organize the code into packages.
2. Abstract Class: Create an abstract class for shared recipe functionality.
3. Interface: Define an interface for managing recipes.
4. Inheritance: Use inheritance to extend classes.
5. Exception Handling: Handle exceptions gracefully.
6. File I/O: Save and load recipes from a file.

- The Directory of classes:

```
C:\Users\Samiksha\Downloads\OOPSASS\innovativeoops\src\
    recipe\
        Recipe.java
        SimpleRecipe.java
        RecipeManager.java
        FileManager.java
        RecipeInterface.java
        RecipeApp.java
```

4. Core OOP Concepts Applied

- **Inheritance**: Use inheritance to create subclasses.
- **Abstraction and Interfaces**: Create an abstract class Recipe that includes common methods like getIngredients() and getInstructions(). This serves as a base for specific recipe types. Define an interface RecipeManager with methods like addRecipe(), removeRecipe(), and listRecipes(), ensuring consistent recipe management across implementations.
- **Encapsulation**: Data within each class is protected, ensuring that fields are accessible onlythrough class methods.
- **Polymorphism**: Method overloading is used for managing multiple constructors and adding flexibility in function handling.

5. Exception Handling

- Implement try-catch blocks to handle exceptions gracefully, such as file not found errors or invalid inputs, enhancing user experience.
- Example: FileNotFoundException and IOException are caught when loading files to handle caseswhere the file does not exist or is unreadable.

6. Console Interaction

The application is menu-driven, allowing users to interact with the system through the console. Options are provided for adding recipes, viewing recipes, deleting recipes, searching recipe by ingredient, and saving the recipes.

7. File Handling

The FileHandler class is responsible for saving and loading the data of recipes, ensuringthat data is preserved between application sessions. This involves serializing and deserializingobjects to/from files.

8. Sample Code Snippets:

Recipe.java

```java
package recipe;

import java.util.List;

public abstract class Recipe {
    protected String name;
    protected List<String> ingredients;
    protected String instructions;

    public Recipe(String name, List<String> ingredients, String instructions) {
        this.name = name;
        this.ingredients = ingredients;
        this.instructions = instructions;
    }

    public abstract void displayRecipe();

    public String getName() {
        return name;
    }

    public List<String> getIngredients() {
        return ingredients;
    }

    public String getInstructions() {
        return instructions;
    }
}
```

SimpleRecipe.java

```java
package recipe;

import java.util.List;

public class SimpleRecipe extends Recipe {
    public SimpleRecipe(String name, List<String> ingredients, String instructions) {
        super(name, ingredients, instructions);
    }

    @Override
    public void displayRecipe() {
    System.out.println("Name: " + getName());
    System.out.println("Ingredients: " + String.join(delimiter:", ", getIngredients()));
    System.out.println("Instructions: " + getInstructions());
    System.out.println();
}
    @Override
    public String toString() {
        return name + "\nIngredients: " + String.join(delimiter:", ", ingredients) + "\nInstructions: " +
    }


}
```

RecipeManager.java

```java
package recipe;

import java.util.ArrayList;
import java.util.List;

public class RecipeManager implements RecipeInterface {
    private List<Recipe> recipes = new ArrayList<>();

    @Override
    public void addRecipe(Recipe recipe) {
        recipes.add(recipe);
    }

    @Override
    public void removeRecipe(String recipeName) {
        recipes.removeIf(r -> r.getName().equalsIgnoreCase(recipeName));
    }

    @Override
    public Recipe findRecipe(String recipeName) {
        return recipes.stream()
                      .filter(r -> r.getName().equalsIgnoreCase(recipeName))
                      .findFirst()
                      .orElse(other:null);
    }

    @Override
    public List<Recipe> listRecipes() {
        return new ArrayList<>(recipes);
    }

    @Override
    public List<Recipe> searchByIngredient(String ingredient) {
        List<Recipe> results = new ArrayList<>();
        for (Recipe recipe : recipes) {
            for (String recipeIngredient : recipe.getIngredients()) {
                if (recipeIngredient.equalsIgnoreCase(ingredient)) {
                    results.add(recipe);
                    break;  // Move to the next recipe after finding a match
                }
            }
        }
        return results; // Fixed the formatting issue here
    }
}
```

FileManger.java

```java
package recipe;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class FileManager {
    public void saveRecipes(List<Recipe> recipes, String filename) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
            for (Recipe recipe : recipes) {
                writer.write(recipe.getName() + ";" + String.join(delimiter:",", recipe.getIngredients())
                writer.newLine();
            }
        }
    }

    public List<Recipe> loadRecipes(String filename) throws IOException {
        List<Recipe> recipes = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(regex:";");
                String name = parts[0];
                List<String> ingredients = List.of(parts[1].split(regex:","));
                String instructions = parts[2];
                recipes.add(new SimpleRecipe(name, ingredients, instructions));
            }
        }
        return recipes;
    }
}
```

RecipeInterface.java

```java
package recipe;

import java.util.List;

public interface RecipeInterface {
    void addRecipe(Recipe recipe);
    void removeRecipe(String recipeName);
    Recipe findRecipe(String recipeName);
    List<Recipe> listRecipes();
    List<Recipe> searchByIngredient(String ingredient);
}
```

RecipeApp.java

```java
package recipe;

import java.io.IOException;
import java.util.List;
import java.util.Scanner;

public class RecipeApp {
    Run | Debug
    public static void main(String[] args) {
        RecipeManager recipeManager = new RecipeManager();
        FileManager fileManager = new FileManager();
        Scanner scanner = new Scanner(System.in);

        // Load existing recipes from file
        try {
            List<Recipe> loadedRecipes = fileManager.loadRecipes(filename:"recipes.txt");
            loadedRecipes.forEach(recipeManager::addRecipe);
        } catch (IOException e) {
            System.out.println("Error loading recipes: " + e.getMessage());
        }

        while (true) {
            System.out.println(x:"1. Add Recipe\n2. View Recipes\n3. Delete Recipe\n4. Search Recipe by In
            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    System.out.print(s:"Enter recipe name: ");
                    String name = scanner.nextLine();
                    System.out.print(s:"Enter ingredients (comma-separated): ");
                    List<String> ingredients = List.of(scanner.nextLine().split(regex:","));
                    System.out.print(s:"Enter instructions: ");
                    String instructions = scanner.nextLine();
                    recipeManager.addRecipe(new SimpleRecipe(name, ingredients, instructions));
                    break;

                case "2":
                    recipeManager.listRecipes().forEach(Recipe::displayRecipe);
                    break;

                case "3":
                    System.out.print(s:"Enter recipe name to delete: ");
                    recipeManager.removeRecipe(scanner.nextLine());
                    break;
```

```
                case "4":
                    System.out.print(s:"Enter ingredient to search: ");
                    List<Recipe> foundRecipes = recipeManager.searchByIngredient(scanner.nextLine());
                    if (foundRecipes.isEmpty()) {
                        System.out.println(x:"No recipes found with that ingredient.");
                    } else {
                        foundRecipes.forEach(Recipe::displayRecipe);
                    }
                    break;


                case "5":
                    try {
                        fileManager.saveRecipes(recipeManager.listRecipes(), filename:"recipes.txt");
                        System.out.println(x:"Recipes saved successfully.");
                    } catch (IOException e) {
                        System.out.println("Error saving recipes: " + e.getMessage());
                    }
                    break;

                case "6":
                    scanner.close();
                    return;

                default:
                    System.out.println(x:"Invalid choice. Please try again.");
            }
        }
    }
}
```

OUTPUTS

Each and every function showed below.

1)Add a recipe

```
C:\Users\Samiksha>cd C:\Users\Samiksha\Desktop\OOPSASS\innovativeoops\src

C:\Users\Samiksha\Desktop\OOPSASS\innovativeoops\src>javac recipe/*.java

C:\Users\Samiksha\Desktop\OOPSASS\innovativeoops\src>java recipe.RecipeApp
1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
1
Enter recipe name: Tea
Enter ingredients (comma-separated): Tea leaves, sugar, water, milk, cardamom
Enter instructions: Boil the water first. Then add the tea leaves and sugar and let it simmer. Then add milk and cardamom and let it
boil.
1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
```

2) View the recipes

```
1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
2
Name: Vegetable Stir Fry
Ingredients: broccoli,  bell pepper,  carrot,  soy sauce,  sesame oil
Instructions: Heat sesame oil in a wok. Add chopped vegetables and stir-fry until tender. Add soy sauce and cook for another minute.
Serve hot.

Name: Chocolate Cake
Ingredients: flour,  sugar,  cocoa powder,  eggs,  butter,  baking powder
Instructions: Mix dry ingredients in one bowl and wet ingredients in another. Combine, pour into a cake pan, and bake at 350°F (175°C
) for 30-35 minutes.

Name: Spaghetti Bolognese
Ingredients: spaghetti,  ground beef,  onion,  garlic,  tomato sauce
Instructions: Cook spaghetti according to package instructions. In a pan, saut? onion and garlic, add ground beef, and cook until bro
wned. Add tomato sauce and simmer. Serve over spaghetti.

Name: Tea
Ingredients: Tea leaves,  sugar,  water,  milk,  cardamom
Instructions: Boil the water first. Then add the tea leaves and sugar and let it simmer. Then add milk and cardamom and let it boil.
```

3) Delete a recipe

```
Enter recipe name to delete: Tea
1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
2
Name: Vegetable Stir Fry
Ingredients: broccoli,  bell pepper,  carrot,  soy sauce,  sesame oil
Instructions: Heat sesame oil in a wok. Add chopped vegetables and stir-fry until tender. Add soy sauce and cook for another minute.
Serve hot.

Name: Chocolate Cake
Ingredients: flour,  sugar,  cocoa powder,  eggs,  butter,  baking powder
Instructions: Mix dry ingredients in one bowl and wet ingredients in another. Combine, pour into a cake pan, and bake at 350°F (175°C
) for 30-35 minutes.

Name: Spaghetti Bolognese
Ingredients: spaghetti,  ground beef,  onion,  garlic,  tomato sauce
Instructions: Cook spaghetti according to package instructions. In a pan, saut? onion and garlic, add ground beef, and cook until bro
wned. Add tomato sauce and simmer. Serve over spaghetti.

1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
```

4) Search a recipe by ingredient

```
1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
4
Enter ingredient to search: flour
Name: Chocolate Cake
Ingredients: flour,  sugar,  cocoa powder,  eggs,  butter,  baking powder
Instructions: Mix dry ingredients in one bowl and wet ingredients in another. Combine, pour into a cake pan, and bake at 350°F (175°C
) for 30-35 minutes.
```

5) Save the recipes

```
1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
5
Recipes saved successfully.
```

6) Exit

```
1. Add Recipe
2. View Recipes
3. Delete Recipe
4. Search Recipe by Ingredient
5. Save Recipes
6. Exit
6

C:\Users\Samiksha\Desktop\OOPSASS\innovativeoops\src>
```

Conclusion

The Recipe Management System project successfully demonstrates the application of OOP principles in Java. By following a modular structure and utilizing OOP concepts, we were able to create a functional, maintainable, and scalable application. This project not only enhanced our understanding of OOP but also taught us practical aspects of Java programming, such as file handling and exception management.

THANK YOU

-x-