Mariam Shalikashvili

# Recipe Management System

This Photo by

**Description**

Create Recipe Management System (RMS) in Java. RMS is a versatile software widely used to organize and manage recipes. Our example provides a basic framework with the following features:

Storage for recipes: The system stores a collection of recipes, allowing users to easily manage and access their culinary creations.

Ability to add recipes: Users can seamlessly add new recipes to the system, specifying details such as title, author, and ingredients.

Ability to remove recipes: Users have the flexibility to remove recipes from their collection, ensuring their recipe library remains up-to-date.

Ability to print recipe information: The system offers the functionality to print detailed information about recipes, including title, author, and ingredients, directly to the console.

**RMS structure**

We will need the following classes for the software:

1. Recipe - Represents a recipe itself.
2. RMS - Recipe Management System, the core system for managing recipes.
3. RMSTester - Tester class for validating and testing the functional System.

**Class Recipe**

String title

String author

String ingredients

**Class RMS**

RMS: Utilizes a list of books (List<Book> storage) with methods to add (void addBook(Book)), remove (boolean removeBook(Book)), and print (void printStorage()) book information.

**Class Recipe:**

The Recipe class should contain various attributes, including title and author, to represent a recipe. This class can be implemented as follows:

```java
package task2;

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

9 usages
public class Recipe {
    2 usages
    private String title;
    2 usages
    private String author;

    2 usages
    private String ingredients;

    3 usages
    public String getTitle() { return title; }

    2 usages
    public void setTitle(String title) { this.title = title; }

    3 usages
    public String getAuthor() { return author; }

    2 usages
    public void setAuthor(String author) { this.author = author; }

    1 usage
    public String getIngredients(){ return ingredients; }
    2 usages
    public void setIngredients(String ingredients) {this.ingredients = ingredients;
    }

}
```

Ensure proper implementation of setters and getters for all fields. Typically, fields are private and accessed using getters and setters. Consider implementing the toString() method for the Recipe class to provide a string representation of the object.

**Class GRMS**

The recipe management system should possess an internal structure for storing recipes. It must include functionalities to add new recipes and remove existing ones. Additionally, it should offer the capability to print the entire recipe collection as required. The class can be structured and implemented accordingly. The code for this class looks like following:

```java
package task2;

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

2 usages
public class GRMS {
    6 usages
    private List<Recipe> storage = new ArrayList<>();

    2 usages
    public void addRecipe(Recipe recipe) {
        storage.add(recipe);
    }

    1 usage
    public boolean removeRecipe(Recipe recipe) {
        boolean removed = false;
        for (int i = 0; i < storage.size(); i++) {
            Recipe r = storage.get(i);
            if (r.getTitle().equals(recipe.getTitle()) && r.getAuthor().equals(recipe.getAuthor())) {
                storage.remove(i);
                removed = true;
                break;
            }
        }
        return removed;
    }

    1 usage
    public void printStorage() {
        if (storage.isEmpty()) {
            System.out.println("The recipe storage is empty");
        } else {
            for (Recipe recipe : storage) {
                System.out.println(recipe.getTitle() + " by " + recipe.getAuthor()+ " ingredients: "+ recipe.getIngredients());
            }
        }
    }
}
```

Pay close attention to the implementation of the List interface, utilization of for loops for iterating through lists and proper object comparison. Understanding the implementation of interfaces is crucial in this context. Additionally, observe the usage of Boolean variables in the methods of this example for effective control flow.

**GRMSTester Class:**

Let's initiate testing for our recipe management system. Start by creating a few recipes. Then, establish the GRMS and add these recipes to the system. Finally, attempt to remove some of the recipes to validate the functionality of our management system. The code for this class looks like following:

```java
package task2;

public class GRMSTester {
    public static void main(String[] args) {
        Recipe r1= new Recipe();
        r1.setTitle("Khachapuri");
        r1.setAuthor("Nana Arakishvili");
        r1.setIngredients("flour and cheese");

        Recipe r2 = new Recipe();
        r2.setTitle("Khinkali");
        r2.setAuthor("Gela Melikishvili");
        r2.setIngredients("flour and meat");

        GRMS grms = new GRMS();
        grms.addRecipe(r1);
        grms.addRecipe(r2);
        grms.removeRecipe(r1); // Removing r1
        grms.printStorage();
    }
}
```

And finally, we print the state of the management system to check if all the methods are working properly.