**GROUP MEMBERS**

**168657 - Lisa Mukami**

**166534 - Stephanie Naliaka**

**168827 - Sasha Ndirangu**

**PYTHON CODE**

from abc import ABC, abstractmethod

# Ingredient class to represent individual ingredients in a recipe

class Ingredient:

def \_\_init\_\_(self, name, quantity, unit):

self.name = name

self.quantity = quantity

self.unit = unit

def \_\_str\_\_(self):

return f"{self.quantity} {self.unit} of {self.name}"

# Abstract Recipe class

class Recipe(ABC):

def \_\_init\_\_(self, title):

self.title = title

self.ingredients = [] # List of Ingredient objects

def add\_ingredient(self, ingredient):

self.ingredients.append(ingredient)

def remove\_ingredient(self, ingredient\_name):

self.ingredients = [ing for ing in self.ingredients if ing.name != ingredient\_name]

def display\_ingredients(self):

for ingredient in self.ingredients:

print(ingredient)

@abstractmethod

def get\_type(self):

pass

# Breakfast subclass inheriting from Recipe

class BreakfastRecipe(Recipe):

def get\_type(self):

return "Breakfast"

# Lunch subclass inheriting from Recipe

class LunchRecipe(Recipe):

def get\_type(self):

return "Lunch"

# Dinner subclass inheriting from Recipe

class DinnerRecipe(Recipe):

def get\_type(self):

return "Dinner"

# MealPlan class to manage a collection of recipes for each day

class MealPlan:

def \_\_init\_\_(self):

self.meals = {"Breakfast": None, "Lunch": None, "Dinner": None}

def add\_meal(self, meal\_type, recipe):

if meal\_type in self.meals:

self.meals[meal\_type] = recipe

else:

print("Invalid meal type. Choose from Breakfast, Lunch, or Dinner.")

def view\_meal\_plan(self):

for meal\_type, recipe in self.meals.items():

if recipe:

print(f"{meal\_type}: {recipe.title} - Ingredients:")

recipe.display\_ingredients()

else:

print(f"{meal\_type}: No meal planned")

# Main Functionality

def main():

# Create a few recipes

pancakes = BreakfastRecipe("Pancakes")

pancakes.add\_ingredient(Ingredient("Flour", 2, "cups"))

pancakes.add\_ingredient(Ingredient("Milk", 1, "cup"))

pancakes.add\_ingredient(Ingredient("Egg", 1, "piece"))

salad = LunchRecipe("Salad")

salad.add\_ingredient(Ingredient("Lettuce", 1, "bunch"))

salad.add\_ingredient(Ingredient("Tomato", 2, "pieces"))

salad.add\_ingredient(Ingredient("Cucumber", 1, "piece"))

pasta = DinnerRecipe("Pasta")

pasta.add\_ingredient(Ingredient("Pasta", 200, "grams"))

pasta.add\_ingredient(Ingredient("Tomato Sauce", 100, "ml"))

pasta.add\_ingredient(Ingredient("Cheese", 50, "grams"))

# Create a meal plan and add the recipes

meal\_plan = MealPlan()

meal\_plan.add\_meal("Breakfast", pancakes)

meal\_plan.add\_meal("Lunch", salad)

meal\_plan.add\_meal("Dinner", pasta)

# Display the meal plan

print("\nMeal Plan:")

meal\_plan.view\_meal\_plan()

# Run the program

if \_\_name\_\_ == "\_\_main\_\_":

main()

**DOCUMENTATION - RECIPE MANAGEMENT SYSTEM**

This system allows users to manage recipes and create a daily meal plan. Users can add ingredients to recipes, assign recipes to different meal slots (breakfast, lunch, dinner), and view the meal plan. This project demonstrates OOP principles including inheritance, encapsulation, polymorphism, and abstraction.

**Classes**

1**. Ingredient**

**- Attributes**: `name` (ingredient name), `quantity`, `unit`.

- **Methods:**

- `\_\_str\_\_()`: Returns the ingredient as a formatted string.

2. **Recipe (Abstract)**

- **Attributes:** `title` (recipe name), `ingredients` (list of Ingredient objects).

**- Methods:**

- `add\_ingredient(ingredient)`: Adds an ingredient.

- `remove\_ingredient(name)`: Removes an ingredient by name.

- `display\_ingredients()`: Displays ingredients in the recipe.

- `get\_type()`: Abstract method for meal type, implemented by subclasses.

3. **Breakfast Recipe, Lunch Recipe, Dinner Recipe**

- Inheritance: Subclasses of `Recipe`.

***-* Methods:**

- `get\_type()`: Returns the specific meal type (e.g., "Breakfast").

4**.Meal Plan**

- Attributes: `meals` (dictionary with meal slots like "Breakfast", "Lunch", etc.).

***Methods:***

- `add\_meal (type, recipe)`: Adds a recipe to a meal slot.

- `view\_meal\_plan()`: Displays all meals and their ingredients.

**Main Functionality**

- ***Process***:

- Creates instances of recipes, adds ingredients, assigns them to a daily `MealPlan`, and displays the full meal plan.

**OOP Principles**

- Inheritance: Meal-specific classes inherit from `Recipe`.

-Polymorphism: `get\_type()` method differs for each meal type.

- Encapsulation: `ingredients` managed privately within `Recipe`.

- Abstraction: `Recipe` is an abstract base class.

This brief documentation covers the essentials of the system, making it easy to understand each class’s purpose and the main functionality. Let me know if this is concise enough or if further reduction is needed!