The Recipe Cook Book Program

By: Zeus Noah G. Castillo
Ralph Ian N. Comendador

Designed to serve as a digital repository for cooking instructions. Its primary purpose is to leverage OOP to create a structured, scalable, and maintainable system for managing recipes. This structure allows for easy addition of new recipe types, clear separation of concerns, and consistent handling of common recipe attributes and actions. Instead of a flat list, the program utilizes a hierarchy to categorize recipes, ensuring that specific dish types (such as a cake versus a steak) have appropriate, specialized properties and behaviors while sharing core recipe characteristics.

Encapsulation Usage:

Recipe details such as name, ingredients list, cooking time, and instructions will be declared as private within the base Recipe class and its subclasses. Purpose: This hides the internal state of a recipe object from direct, external manipulation. It ensures data integrity by forcing all interactions to go through public getter and setter methods (accessors and mutators). For instance, a user cannot accidentally delete the ingredients list; they must call a method like getIngredients().

Abstraction Usage:

The base class, Recipe, will be defined as an abstract class that implements the Cookable interface. The Cookable interface defines the mandatory contract for the cook() method. The Recipe class will contain other abstract methods like displayDetailedInstructions(). This means you cannot create a generic Recipe object directly. Purpose: Abstraction focuses on the essential idea of a recipe (it has a name, it can be cooked) without providing the concrete implementation details. By implementing the interface, it enforces a contract (cook() must exist) while serving as a blueprint. It ensures that all derived classes (e.g., Dessert) must provide their own specific implementation for all abstract methods and the interface's contract, guaranteeing completeness and consistency across all recipe types.

Inheritance Usage:

The specialized dish classes Dessert, MainDish, and SideDish will inherit from the abstract base class Recipe. Purpose: Inheritance promotes code reusability. The subclasses automatically receive all common recipe attributes (like name, ingredients, time) and methods (like getName()) defined in the Recipe base class. It also establishes a clear "is-a" relationship (e.g., a Dessert is-a Recipe), forming a type hierarchy that makes the program structure logical and easy to navigate.

Polymorphism Usage:

Methods like displayRecipe() and cook() will be implemented in the base class and overridden in the subclasses (Dessert, MainDish, SideDish). For example, Dessert.cook() might include steps for cooling and frosting, while MainDish.cook() might focus on searing and resting meat. Purpose: Polymorphism ("many forms") allows a single interface (a method call like

recipe.cook()) to execute different logic depending on the actual type of the object it's called on. This allows the program to treat a list of recipes uniformly, calling cook() on every item, while each item executes its specialized set of instructions.

Class / Interface	Type	Attributes (Fields)	Methods	Description / Role
Cookable	Interface		cook()	Defines a contract that all recipe types must follow. Any class implementing this interface must provide its own version of cook().
Recipe	Abstract Class	- private String name - private List<String> ingredients - private int cookingTime - private String instructions	- getName() - setName(String name) - getIngredients() - setIngredients(List<String> ingredients) - getCookingTime() - setCookingTime(int time) - getInstructions() - setInstructions(String instructions) - displayRecipe() (may have default or abstract version) - abstract void displayDetailedInstructions() - abstract void cook() (from Cookable interface)	Serves as the **base class** for all recipe types. Implements shared attributes and behaviors, but leaves abstract methods for subclasses to define specific details.
Dessert	Subclass of Recipe	Inherits all from Recipe; may add attributes like sweetnessLevel or requiresChilling	- cook() (overridden) - displayDetailedInstructions() (overridden)	Represents dessert recipes (e.g., cake, pudding).

				Overrides methods to include dessert-specific steps like cooling or frosting.
MainDish	Subclass of Recipe	Inherits from Recipe; may add proteinType or servingTemperature	- cook() (overridden) - displayDetailedInstructions() (overridden)	Represents main dish recipes (e.g., steak, pasta). Customizes cooking logic such as searing, boiling, or resting meat.
SideDish	Subclass of Recipe	Inherits from Recipe; may add pairingSuggestion	- cook() (overridden) - displayDetailedInstructions() (overridden)	Represents side dishes. Defines specific preparation steps that complement main dishes.
CookBook	Regular Class	- List<Recipe> recipes	- addRecipe(Recipe recipe) - removeRecipe(String name) - displayAllRecipes()	Manages a collection of recipes. Demonstrates polymorphism by handling Recipe objects of different subclasses uniformly.

AI Help Source: https://chatgpt.com/share/68e5028f-0d94-800b-a953-c18165657590