Scenario: **The Magical Vegetarian Recipe Generator**

You're a wizard chef in a fantasy world, and you run a magical kitchen specializing in vegetarian dishes. In this kitchen, recipes are living beings that can interact with each other and possess various magical properties. Your task is to develop a system that allows you to create, modify, and manage these vegetarian recipes using Object-Oriented Programming (OOP) principles.

OOP Concepts to Apply:

1. **Classes & Objects**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**

Task Breakdown:

Step 1: Create the Base Class - `Ingredient`

- **Objective:** Define a class called `Ingredient` that has attributes like `name`, `quantity`, `unit`, and a unique magical property (such as `spiciness`, `sweetness`, or `bitterness`).
- **Twist:** Since this is a vegetarian dish, make sure that the `Ingredient` class includes an attribute indicating if the ingredient is a vegetable or spice.

Step 2: Create Derived Classes - `Vegetable` and `Spice`

- **Objective:** Using inheritance, create classes `Vegetable` and `Spice` that inherit from the `Ingredient` class.
- **Twist:** Each derived class should have an additional attribute unique to it, such as `color` for `Vegetable` and `heatLevel` for `Spice`. These classes should override a method called `describe()` to provide a detailed description of the ingredient, including its magical properties.

Step 3: Encapsulate Recipe Logic in a `Recipe` Class

- **Objective:** Create a `Recipe` class that holds a list of `Ingredient` objects. Include methods to add ingredients, remove ingredients, and adjust quantities.
- **Twist:** The `Recipe` class should also have a method called `castSpell()` that magically transforms the dish based on the combination of ingredients (e.g., a mix of sweet and spicy vegetables could result in a dish with a "Fiery Sweet" flavor).

Step 4: Introduce Polymorphism - Multiple Types of Vegetarian Recipes

- **Objective:** Use polymorphism to create different types of vegetarian recipes like `SaladRecipe`, `SoupRecipe`, and `StirFryRecipe`, all inheriting from `Recipe`.
- **Twist:** Each type of recipe should implement its own version of the `castSpell()` method. For instance, `SaladRecipe` might enhance the freshness and crunchiness of the vegetables, while `SoupRecipe` might merge the flavors into a harmonious broth.

Step 5: Abstract the Cooking Process

- **Objective:** Introduce an abstract class `CookingProcess` that outlines the steps of a recipe without specifying the details.
- **Twist:** Implement this abstract class in concrete classes like `BoilingProcess`, `SauteingProcess`, and `BlendingProcess`. The `Recipe` class should accept a `CookingProcess` object and execute the process accordingly.

Final Challenge:

After completing the above steps, create a full program that simulates creating and managing a `SoupRecipe` using these concepts. Demonstrate how the addition, removal, or adjustment of ingredients affects the final magical properties of the vegetarian dish.
