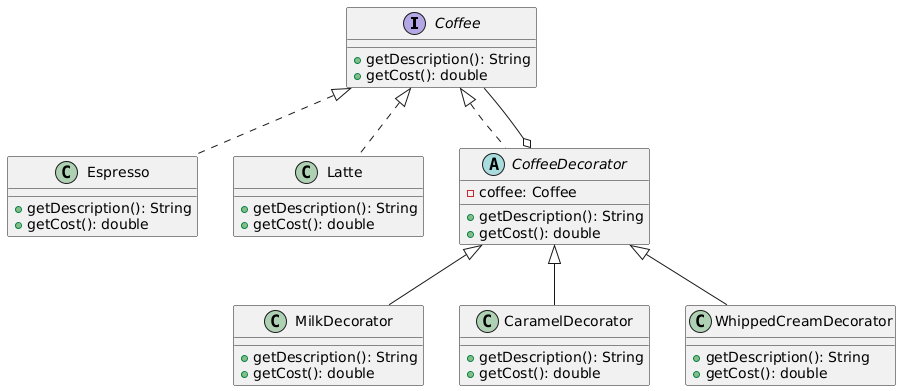
## **Assignment: Implement the Decorator Pattern for a Coffee Shop**

### **Problem Statement**

A coffee shop wants to sell different types of coffee. Each coffee has a base description and price. Customers can add extra ingredients (like Milk, Whipped Cream, Caramel, etc.).

Instead of creating separate classes for every possible combination (e.g., *EspressoWithMilkAndCaramel*), use the **Decorator Pattern** to dynamically add toppings.

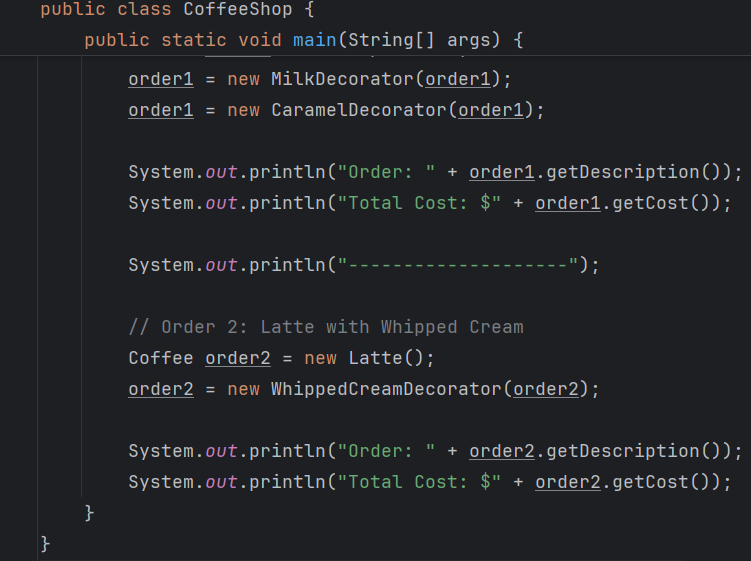
### **Class Diagram**



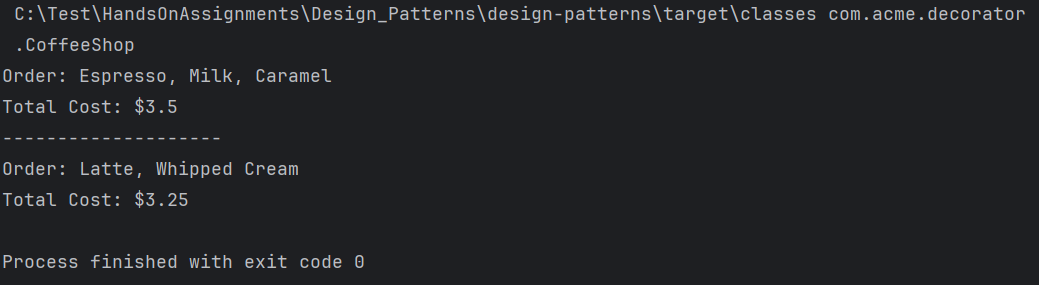
* **Component (Coffee)**
  + getDescription()
  + getCost()
* **Concrete Component**
  + Espresso
  + Latte
* **Decorator (CoffeeDecorator)**
  + coffee : Coffee
  + getDescription()
  + getCost()
* **Concrete Decorators**
  + MilkDecorator
  + CaramelDecorator
  + WhippedCreamDecorator

### **Tasks**

1. Create the Coffee interface with methods getDescription() and getCost().
2. Implement Espresso and Latte as base coffee types.
3. Create an abstract class CoffeeDecorator that implements Coffee and wraps another Coffee object.
4. Implement at least **three decorators** (MilkDecorator, CaramelDecorator, WhippedCreamDecorator) that enhance the description and increase the cost.
5. In the Main class:
   1. Create a simple Espresso.
   2. Add **Milk** and **Caramel** using decorators.
   3. Print out the final description and cost.



### **Expected Output**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*