Barrista

[Concepts](#_v343i2vrhx6a)

[Use Story](#_l75uyqw6212x)

[Design](#_9a7r6qixpjmv)

[GitHub repository](#_h8i6o0s2al7k)

[Video Link](#_f7yl6196iz0t)

## Concepts

* OOPS
* Interface
* Abstract Class
* Inheritance
* Constructor
* Decorator Design Pattern
* Junit

## User Story

You are the owner of a coffee shop.

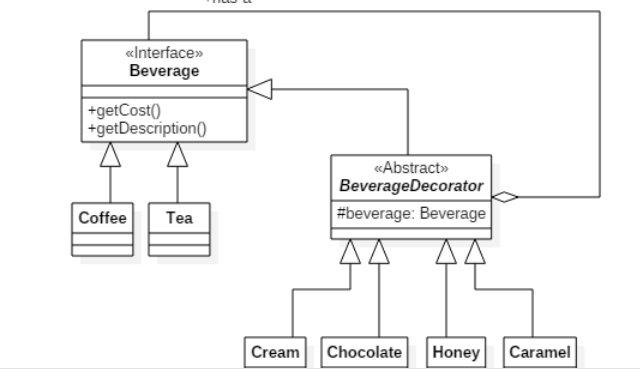
* As a coffee shop owner, I should be able to automate orders of beverages such as cream, chocolate, honey, or caramel.
* As a coffee shop owner, I should be able able to blend the items/addon items
* As a coffee shop owner, I want to find the cost of the coffee with caramel.
* As a coffee shop owner, I am interested to find the cost of tea/coffee/blended items.
* As a user, I want to get a description of the order that I place
* As a user, I want to get the cost of the order that I place

While developing an application, applications of design patterns help to avoid such problems and allow you to build scalable and efficient applications. To solve this issue, you must use the decorator pattern. This definition of this pattern is:

*In object-oriented programming, the decorator pattern is a design pattern that allows the behavior to be added to an individual object, either statically or dynamically, without affecting the behavior of other objects from the same class.*

## Design

The application of the mentioned use case is represented as follows:



The items that are available on the menu are represented as classes. These classes are as follows:

* Coffee
* Tea
* CoffeeWithCream
* TeaWithHoney
* CoffeeWithChocolate
* CoffeeWithCaramel

**Beverage** (Beverage.java): Itis an interface that all beverages must implement. It contains two operations,

* **getCost()**: Returns an integer denoting the cost of the beverage
* **getDescription()**: Returns a string denoting the description of a beverage and all the add-ons that the beverage contains

The **Coffee** (Coffee.java)and **Tea** (Tea.java) classes implement the **Beverage** interface. These classes accept a single parameter of the **integer** type in their constructor that defines the cost. The **getDescription()** operations return "Coffee" and "Tea" respectively.

**BeverageDecorator** (BeverageDecorator.java): It is an abstract class that all the add-ons must extend. It is also a **Beverage** (i.e implements it) and has a **Beverage** as an attribute denoting the beverage to this add-on is added to.

**Cream** (Cream.java), **Chocolate** (Chocolate.java), **Honey** (Honey.java), and **Caramel** (Caramel.java)are classes that extend **BeverageDecorator** and they represent the add-on. Their constructor accepts two parameters, one of type **Beverage** denoting the beverage this add-on is added to and other **integer** denoting the cost of this add-on.

* **getCost():** Returns an integer denoting the cost of the beverage inclusive of all the add-on
* **getDescription(): R**eturns a string denoting a description of the beverage in the following format:  
  <Description of base beverage> topped with <name of add-on>
* For example, coffee topped with caramel, coffee topped with cream topped with chocolate, tea topped with honey, etc.

**Note**: Your implementation is run against pre-defined JUnit test cases

## GitHub repository of the code :

* <https://github.com/MinatoN25/Barrista>

## Video Link

* <https://drive.google.com/file/d/1nU8i6iyVBfGAh0uUzold0vSs_pwhWH4b/view>

------------------------------------------------End of Document —--------------------------------------------------