SWE-4501: Design Pattern

Decorator Design Pattern

Md. Nazmul Haque
Lecturer, IUT

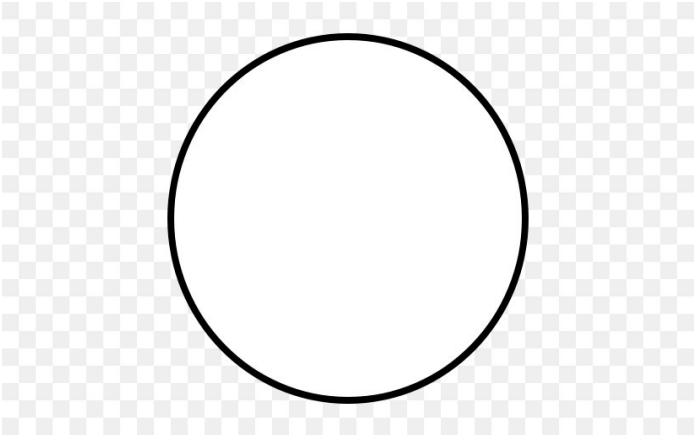
Department of Computer Science and Engineering
Islamic University of Technology

July 12, 2021

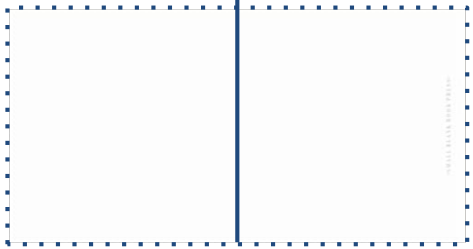
Contents

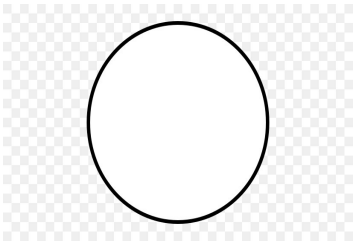
Circle decoration

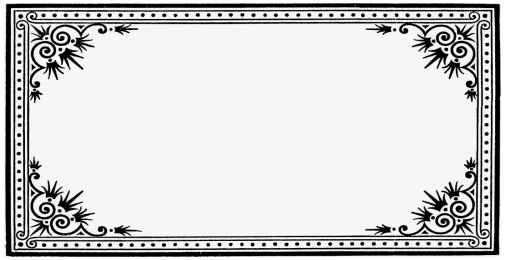
decorate

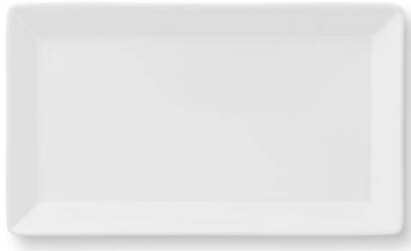


Shape decoration





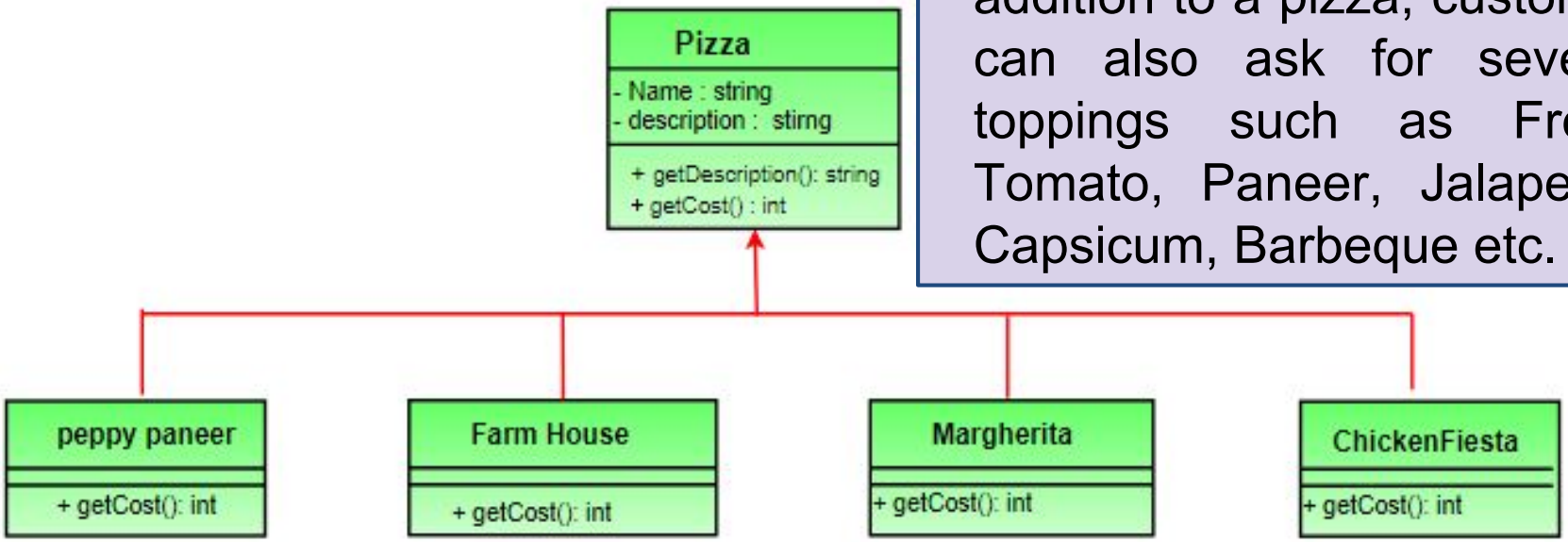




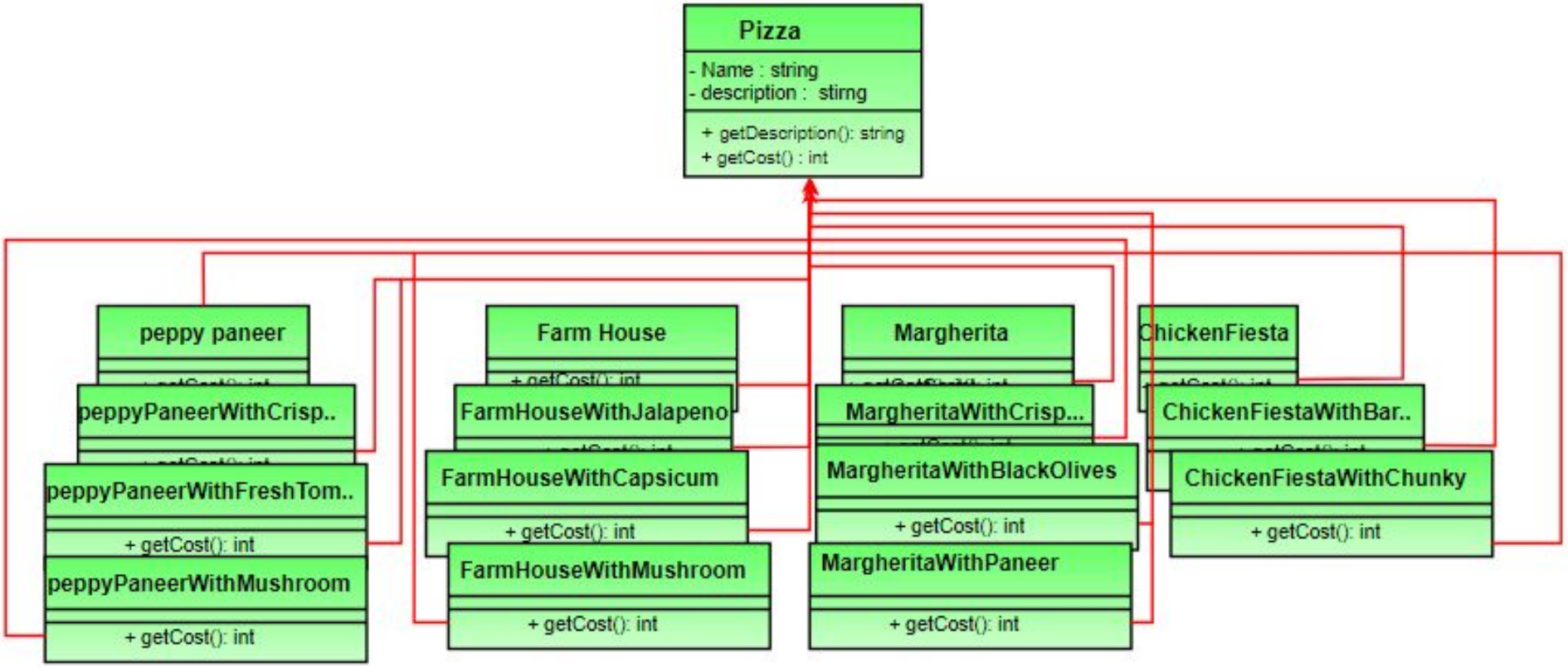
Problem scenario

- Suppose we are building an application for a pizza store that offers four types of pizzas such as Peppy Paneer, Farmhouse, Margherita and Chicken Fiesta. Each pizza has different cost.

New requirement: In addition to a pizza, customer can also ask for several toppings such as Fresh Tomato, Paneer, Jalapeno, Capsicum, Barbeque etc.

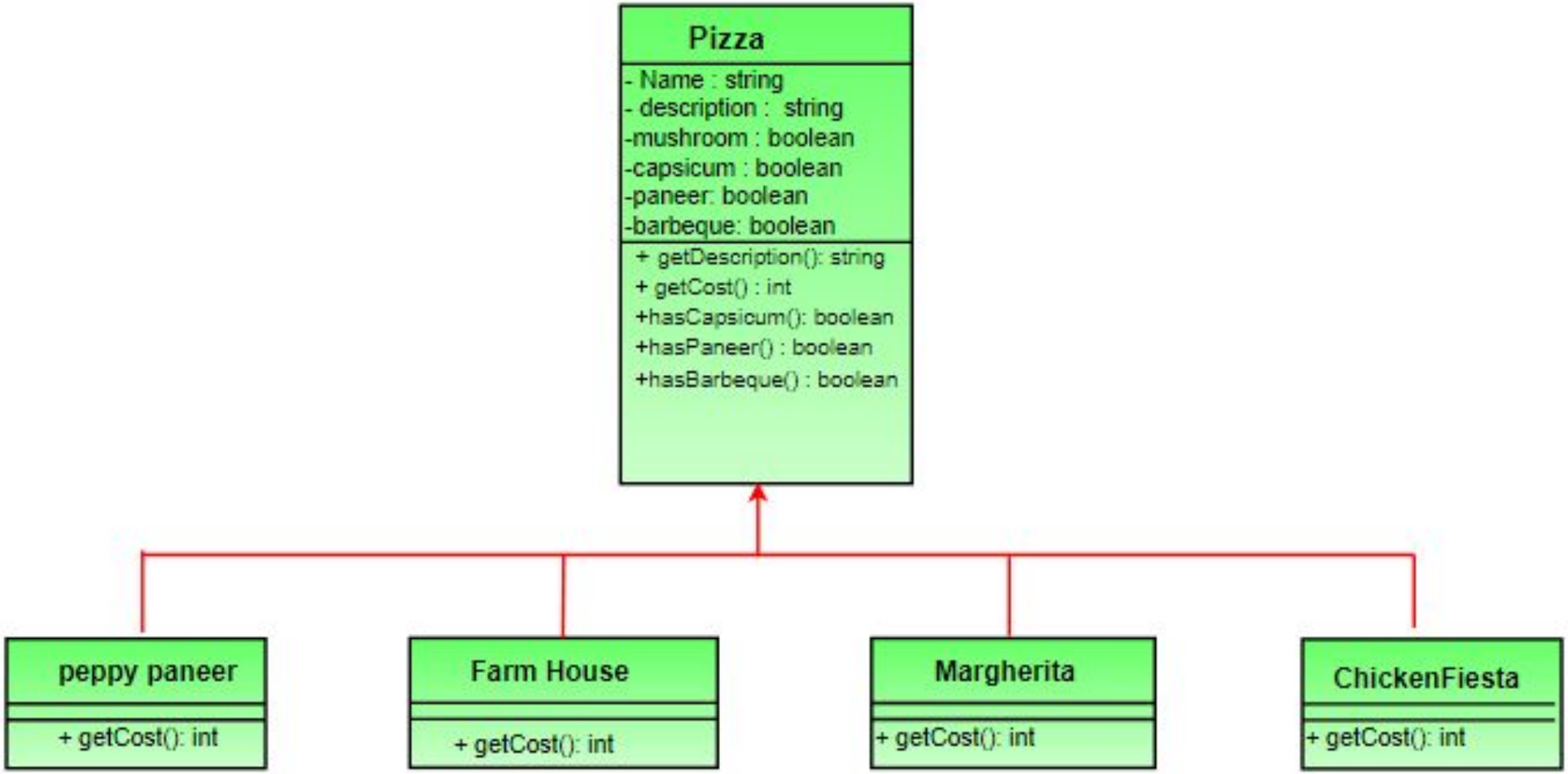


Problem scenario



Class EXPLOSION !!!

Solution option-2



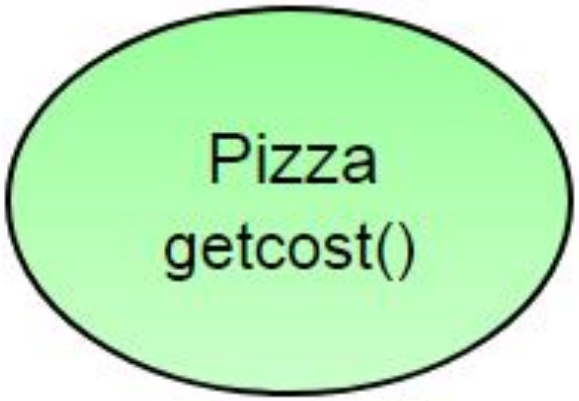
Problems of the solution option-2

- Price changes in toppings will lead to alteration in the existing code.
- New toppings will force us to add new methods and alter getCost() method in superclass.
- For some pizzas, some toppings may not be appropriate yet the subclass inherits them.
- What if customer wants double capsicum or double cheeseburst?

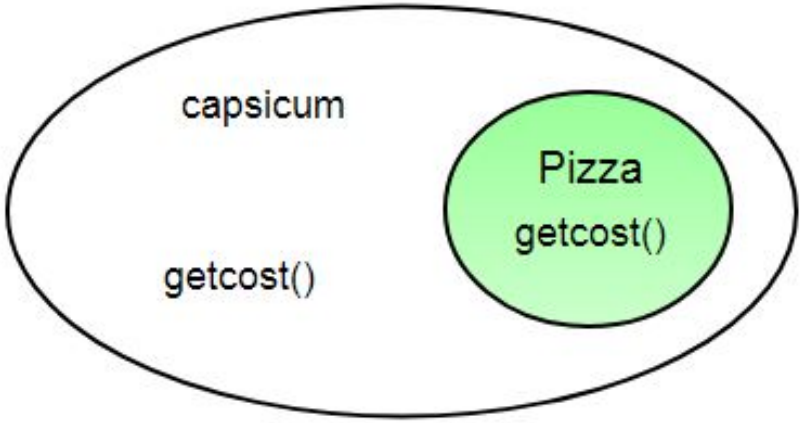
VIOLATE

OPEN CLOSED DESIGN PRINCIPLE

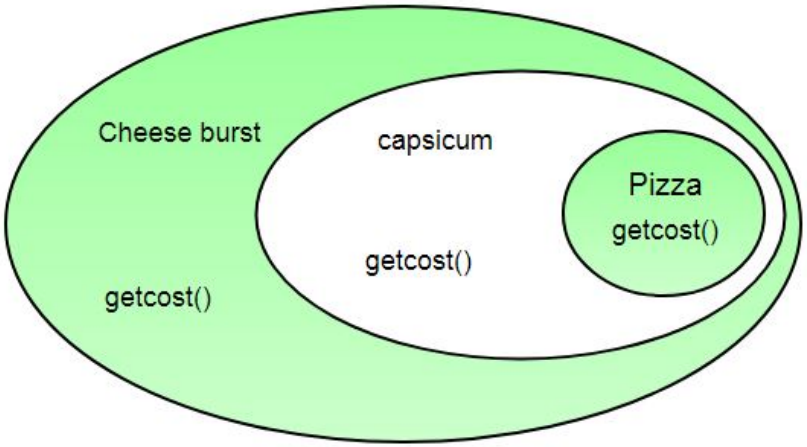
Solution



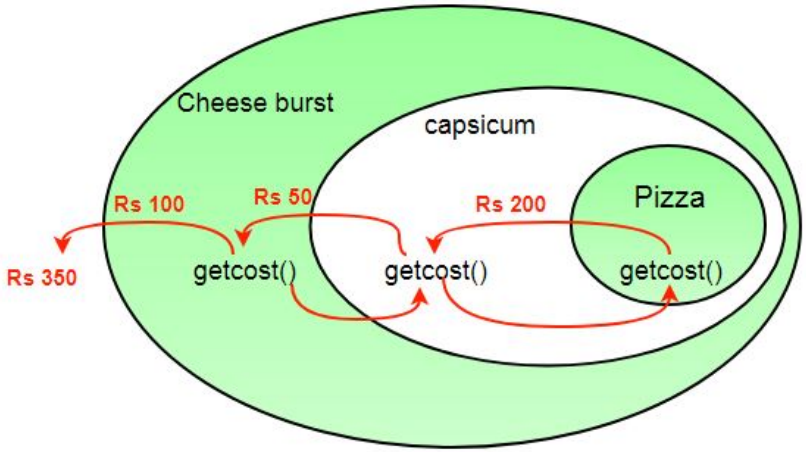
1



2

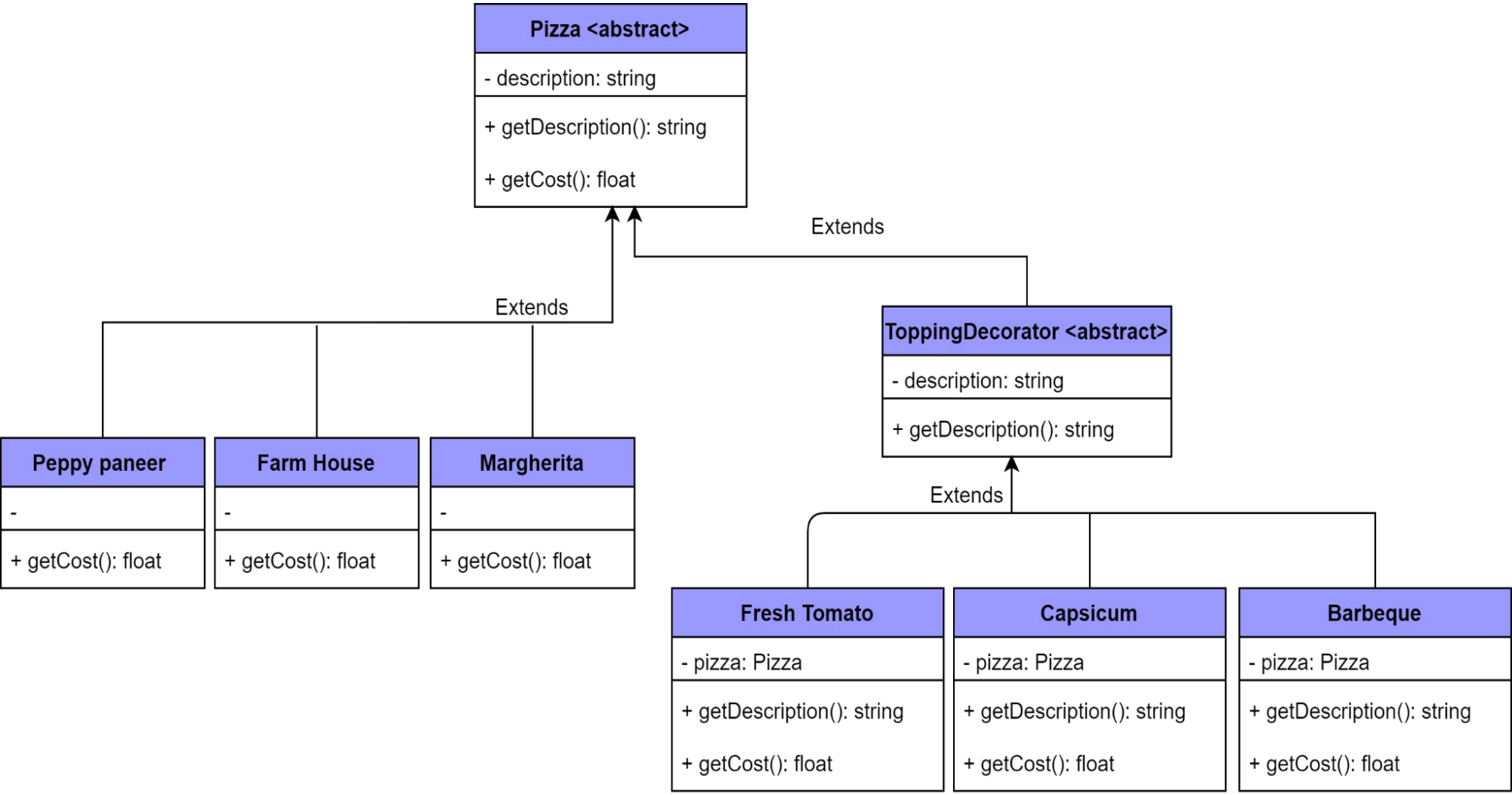


3



4

Solution


Pizza <abstract>
- description: string
+ getDescription(): string
+ getCost(): float
Extends
Extends
Peppy paneer
-
+ getCost(): float
Farm House
-
+ getCost(): float
Margherita
-
+ getCost(): float
ToppingDecorator <abstract>
- description: string
+ getDescription(): string
Extends
Fresh Tomato
- pizza: Pizza
+ getDescription(): string
+ getCost(): float
Capsicum
- pizza: Pizza
+ getDescription(): string
+ getCost(): float
Barbeque
- pizza: Pizza
+ getDescription(): string
+ getCost(): float

Implementation

```
public abstract class Pizza {
    String description = "Unknown Pizza";

    public String getDescription() {
        return description;
    }

    public abstract float getCost();
}
```

```
public class PeppyPaneer extends Pizza {

    public PeppyPaneer() {
        description = "PeppyPaneer";
    }

    public double cost() {
        return 1.99;
    }
}
```

```
public class FarmHouse extends Pizza {

    public FarmHouse() {
        description = "FarmHouse";
    }

    public double cost() {
        return 0.79;
    }
}
```

```
public abstract class ToppingDecorator extends Pizza {

    public abstract String getDescription();
}
```

```
public class FreshTomato extends ToppingDecorator {
    Pizza pizza;

    public FreshTomato(Pizza pizza) {
        this.pizza = pizza;
    }

    public String getDescription() {
        return pizza.getDescription() + ", FreshTomato";
    }

    public double cost() {
        return .20 + pizza.cost();
    }
}
```

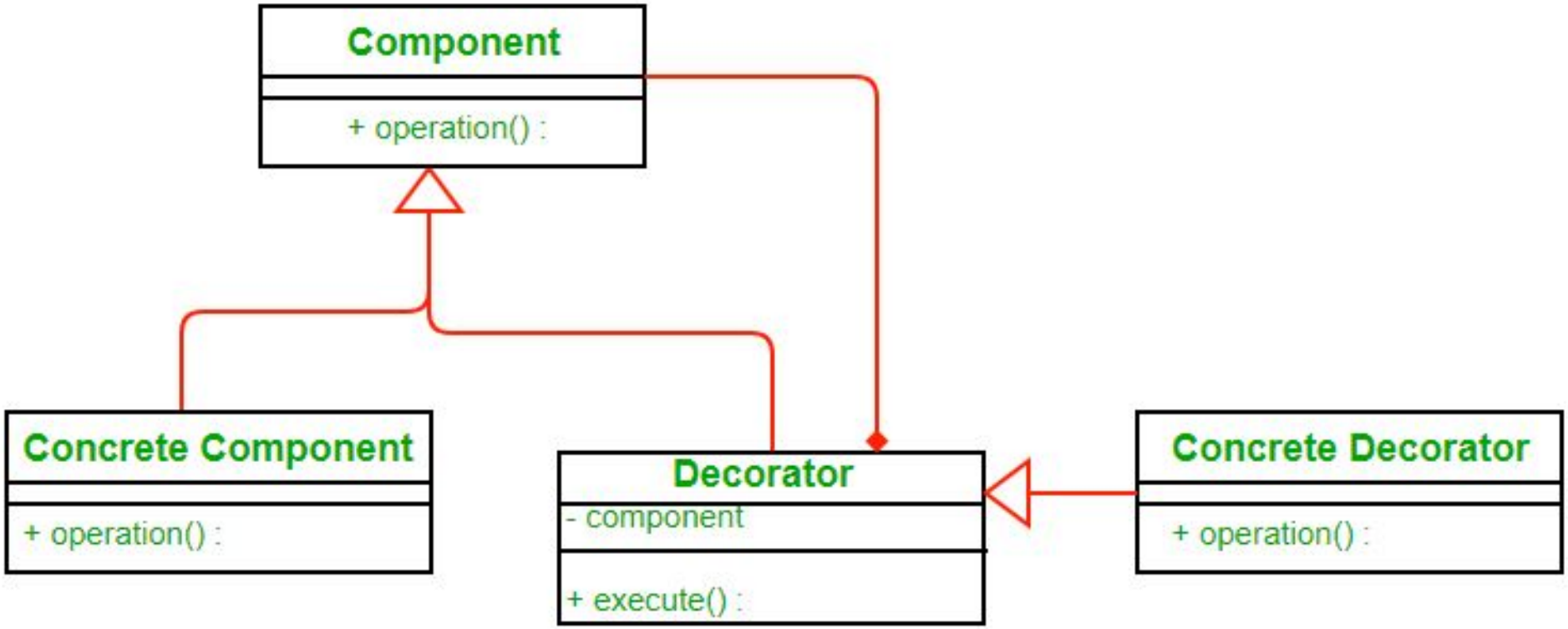
```
public class Capsicum extends ToppingDecorator {
    Pizza pizza;

    public Capsicum(Pizza pizza) {
        this.pizza = pizza;
    }

    public String getDescription() {
        return pizza.getDescription() + ", Capsicum";
    }

    public double cost() {
        return .20 + pizza.cost();
    }
}
```

Decorator Design Pattern



ANY QUESTION ?
THANK YOU !

Acknowledgements

[1] Gamma, Erich. Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional, 1 edition, 1994.

[2] Freeman, Eric, et al. Head first design patterns. " O'Reilly Media, Inc.", 2008.

[3] TutorialsPoint

[4] GeeksforGeeks