SENG 350
- Software Architecture & Design

Shuja Mughal

Design Patterns

Fall 2024

University of Victoria

Design Patterns (Hands-on)

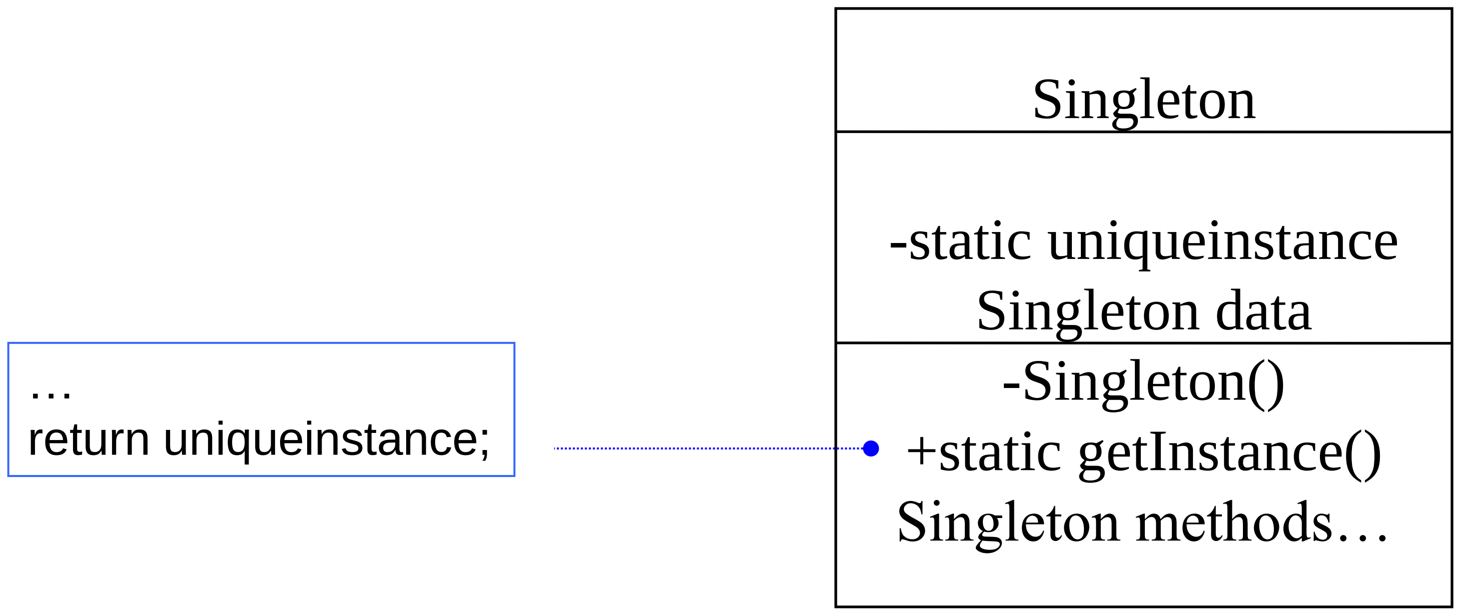
University of Victoria

Singleton Pattern

Design Solution

Defines a getInstance() operation that lets clients access its unique instance

May be responsible for creating its own unique instance



Singleton Pattern Implementation

```
public class Singleton {
    private static volatile Singleton instance;
    private String data;
    private Singleton(String data) {
        this.data = data;
    }
    public static Singleton getInstance(String data) {
        Singleton result = instance;
        if (result == null) {
            synchronized (Singleton.class) {
                result = instance;
                if (result == null) {
                    instance = result = new Singleton (data);
                }
            }
        }
        return result;
    }
}
```

Decorator Pattern

University of Victoria

Decorator Pattern in its general form

Each Component can be used on its own or wrapped by a decorator.

Each decorator HAS-A (wraps) a component, which means the decorator has an instance variable that holds a reference to the component.

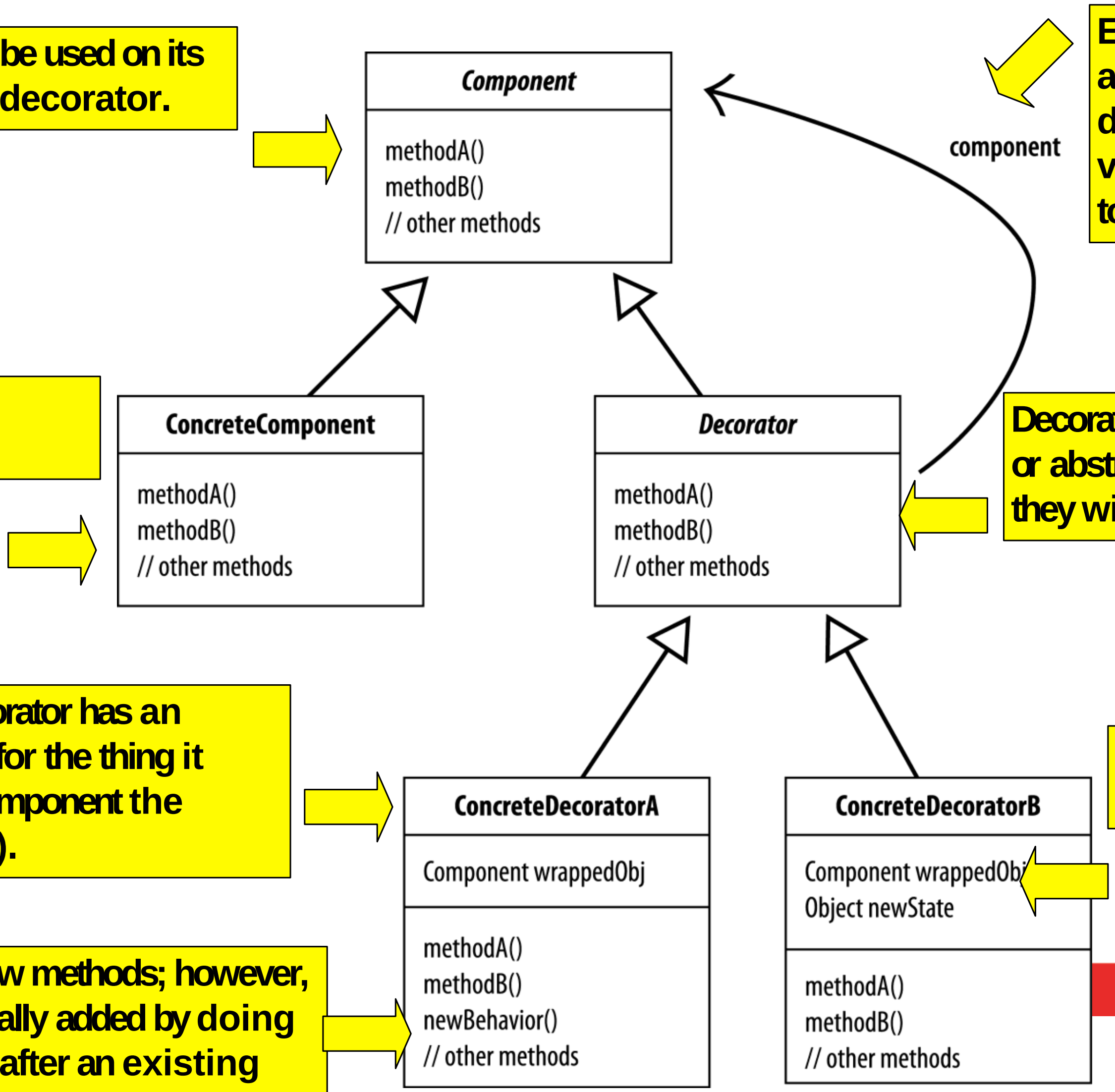
We intend to add new behaviour to this.

Decorators implement the interface or abstract class as the component they will decorate.

The ConcreteDecorator has an instance variable for the thing it decorates (the Component the Decorator wraps).

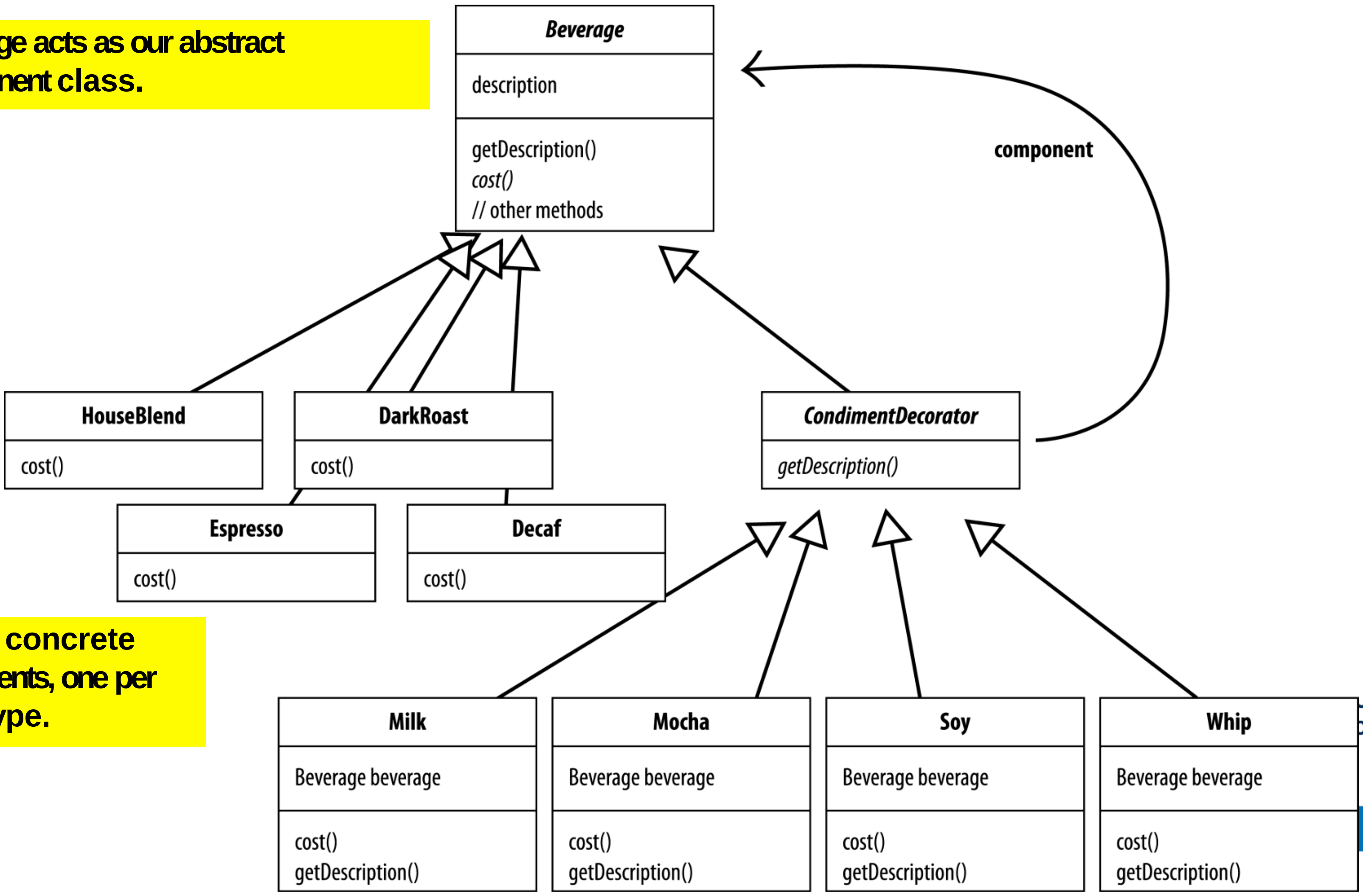
Decorators can extend the state of the component.

Decorators can add new methods; however, new behaviour is typically added by doing computation before or after an existing method in the component.



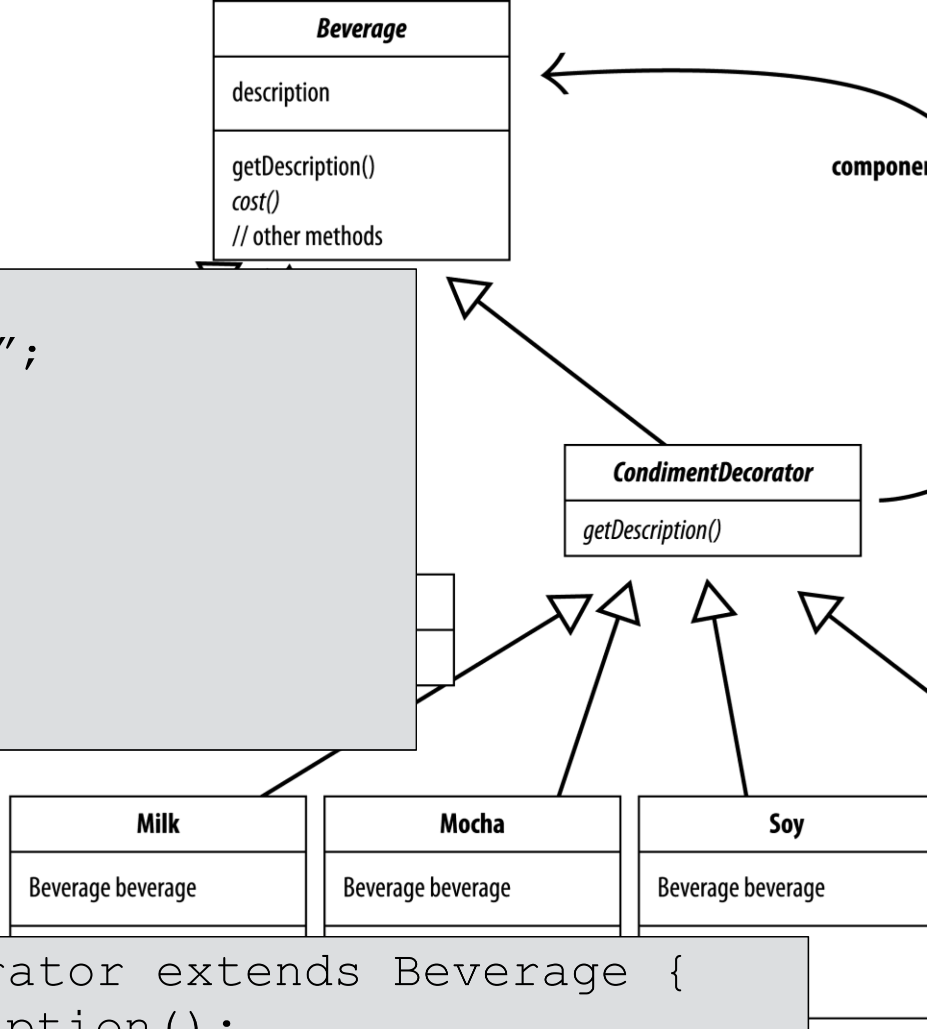
Now applied to Starbuzz (v.3)

Beverage acts as our abstract component class.



The four concrete components, one per coffee type.

Condiment decorators; note which methods they need to implement – both cost() and getDescription().

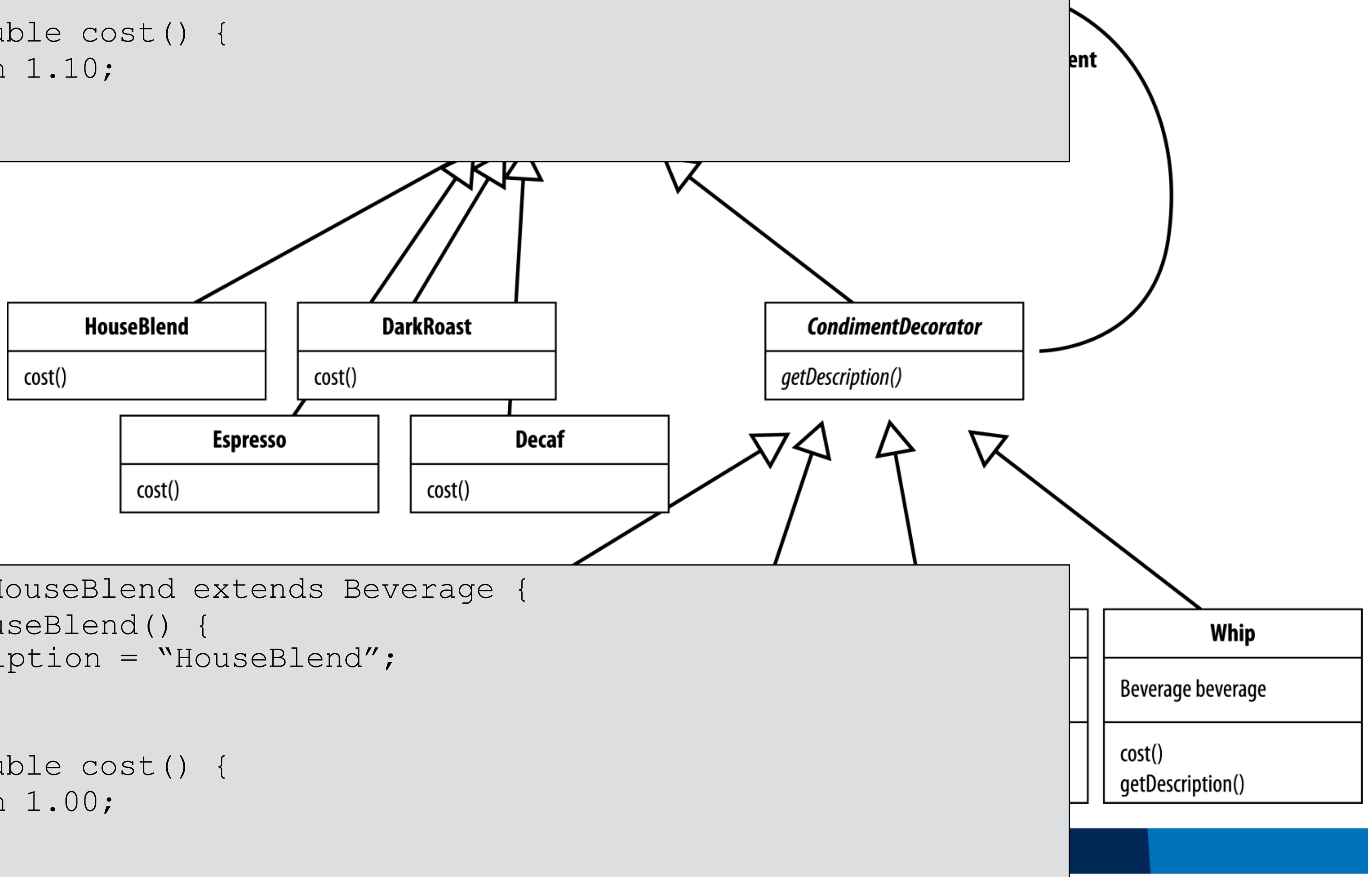


```
public abstract class Beverage {
    String description = "Unknown Beverage";

    public String getDescription() {
        return description;
    }

    public abstract double cost();
}
```

```
public abstract class CondimentDecorator extends Beverage {
    public abstract String getDescription();
}
```

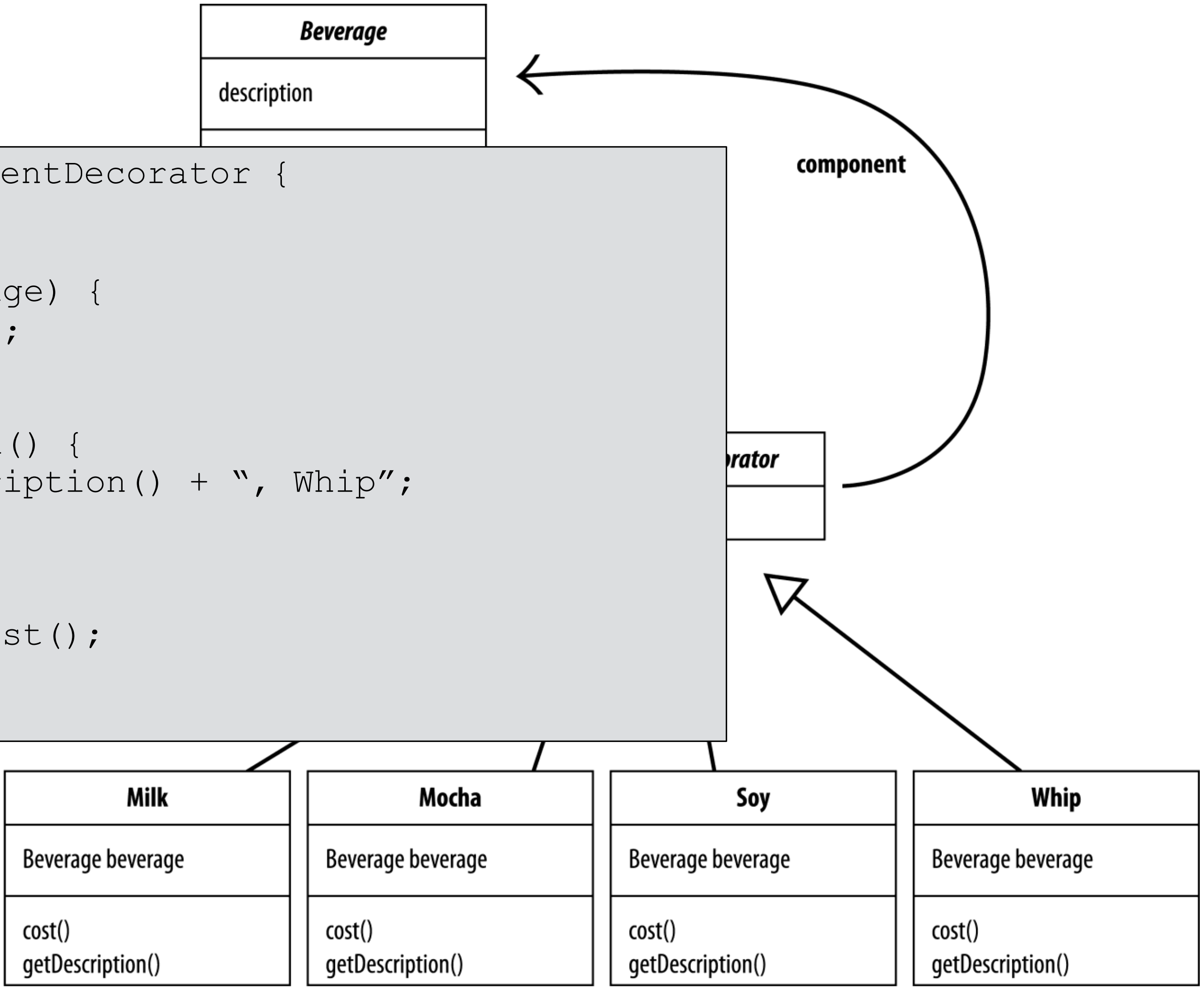


```
public class Decaf extends Beverage {
    public Decaf() {
        description = “Decaf relaxant”;
    }

    public double cost() {
        return 1.10;
    }
}
```

```
public class HouseBlend extends Beverage {
    public HouseBlend() {
        description = “HouseBlend”;
    }

    public double cost() {
        return 1.00;
    }
}
```



```
public class Whip extends CondimentDecorator {
    Beverage beverage;

    public Whip (Beverage beverage) {
        this.beverage = beverage;
    }

    public String getDescription() {
        return beverage.getDescription() + ", Whip";
    }

    public double cost() {
       return 0.40 + beverage.cost();
    }
}
```

```
public class Starbuzz  {
    public static void main (String args[]) {
        Beverage beverage = new Espresso();
        System.out.println (beverage.getDescription() + " $" + beverage.cost());

        Beverage beverage2 = new Decaf();
        beverage2 = new Mocha (beverage2);
        beverage2 = new Mocha (beverage2);
        beverage2 = new Whip (beverage2);
        System.out.println (beverage2.getDescription() + " $" + beverage2.cost());

        Beverage beverage3 = new HouseBlend();
        beverage3 = new Soy (beverage3);
        beverage3 = new Mocha (beverage3);
        beverage3 = new Whip (beverage3);
        System.out.println (beverage3.getDescription() + " $" + beverage3.cost());
    }
}
```

Espresso

Decaf double Mocha with whip

HouseBlend with Soy, Mocha and Whip

Decorator Pattern

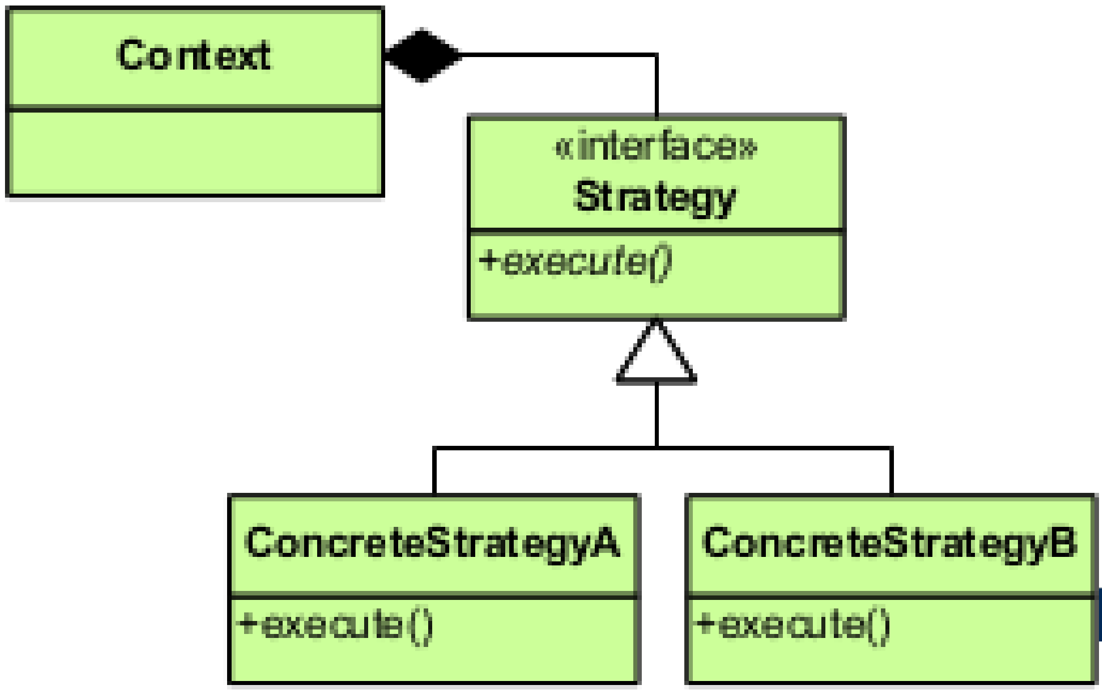
- Implement the decorator pattern for the ice cream example we did in class.

The Strategy Pattern

University of Victoria

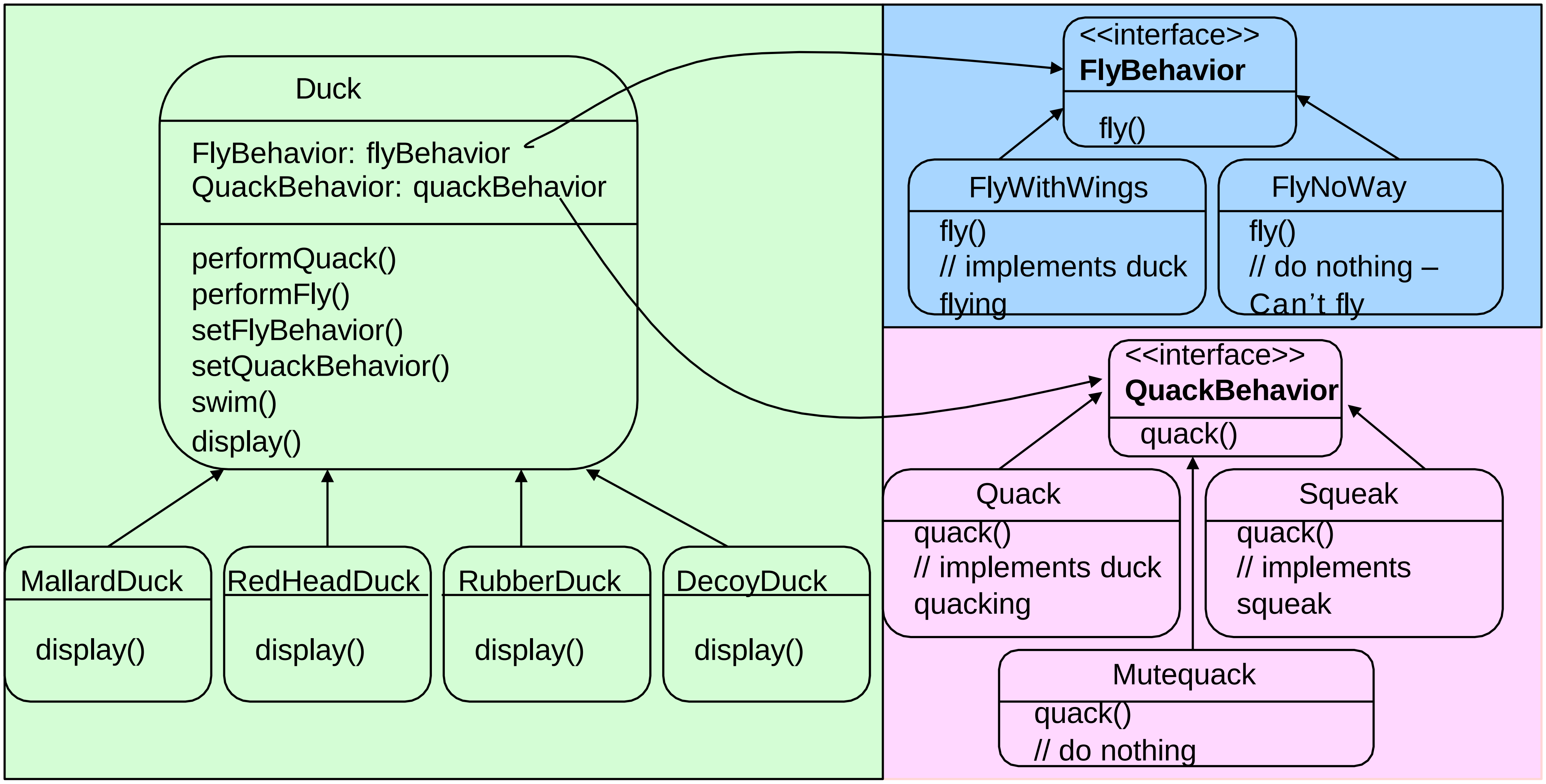
Strategy Pattern Defined

The Strategy Pattern defines a family of algorithms, Encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.



University of Victoria

Using UML on Strategy Pattern



Specific behaviours by implementing interface QuackBehavior

```
public class Quack implements QuackBehavior { public void quack() {
            System.out.println("Quack");
    }
}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public class Squeak implements QuackBehavior { public void quack() {
            System.out.println("Squeak");
    }
}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
public class MuteQuack implements QuackBehavior { public void quack() {
            System.out.println("<< Silence >>");
    }
}
```

2. Implement performQuack()

```
public abstract class Duck {
    // Declare two reference variables for the behavior interface types
    FlyBehavior flyBehavior;
    QuackBehavior quackBehavior; // All duck subclasses inherit these
    // etc

    public Duck() {
    }


    public void performQuack() {
        quackBehavior.quack();   // Delegate to the behavior class
    }
```

3. How to set the quackBehavior variable & flyBehavior variable

```
public class MallardDuck extends Duck {


    public MallardDuck() {


        quackBehavior = new Quack();
                        // A MallardDuck uses the Quack class to handle its quack,
                        // so when performQuack is called, the responsibility for the quack
                        // is delegated to the Quack object and we get a real quack


        flyBehavior = new FlyWithWings();
                        // And it uses flyWithWings as its flyBehavior type

    }

    public void display() {
        System.out.println("I'm a real Mallard duck");
    }
```

quackBehavior = new Quack(); and **flyBehavior = new FlyWithWings();** are printed in bold.

How to set behaviour dynamically?

Add new methods to the Duck class

```
public void setFlyBehavior (FlyBehavior fb) {
    flyBehavior = fb;
}

public void setQuackBehavior(QuackBehavior qb) {
    quackBehavior = qb;
}
```

University of Victoria

Strategy Pattern

- Implement the strategy pattern for the coupon discount example we did in class.