**Decorator Pattern**

Overview and benefits

* Adds new functionality of objects created without changing method signature of a class in new class
* Reduces having to duplicate code
* Makes code more readable as method signature will be same in all classes which uses inheritance to extend and overrides existing behavior

Scenario

A coffee machine can produce coffees in different flavors/types such as Espresso, LongBlack, Cappuccino, FlatWhite. They have things in common but slightly different ingredients. Also, different customers could ask for different amount of sugar, skim milk/regular milk etc. So, without pattern code would look like this.

BEFORE REFACTOR

**Espresso Object**

Class Espresso() {

addRegularMilk() { // add regular milk

print ‘milkAmount’

}

addSugar() { // add 2 s

print ‘2 teaspoon sugarAmount’

}

}

**Cappucino Object**

Class Cappucino {

addSkimMilk() { //add skim milk instead

print ‘milkAmount’

}

addSugar() {

print ‘sugarAmount’ // add 4 teaspoon sugar amount instead

}}

AFTER REFACTOR

A new interface called CoffeeMachine will have method signatures that will be carried to objects which implement interface and can override or reuse method.

Interface CoffeeMachine {

void addMilk() {}

void addSugar() { print ‘1 tablespoon of sugar’ }

void addMilkAmount() {}

}

class Espresso implements CoffeeMachne {

void addMilk() {

print ‘half cup of milk’

}

//no milk/sugar required so addSugar and addMilk method not used

}

Class Cappucino implements CoffeeMachine {

@Override

addSugar {

print `3 tablespoon of sugar’;

}

addMilk {

print `half cup of milk`;

}}

**UML Diagram**

check CoffeeMachineUML.png in DecoratorPattern folder

**or**

[**https://app.creately.com/diagram/kdCWnNZh9v1/edit**](https://app.creately.com/diagram/kdCWnNZh9v1/edit)