Step 10 – Polymorphism

Think about a coffee maker.

You will create AbstractCoffeeMaker, BasicCoffeeMaker, and PremiumCoffeeMaker.

BasicCoffeeMaker supports Coffee.Filter.

PremiumCoffeeMaker supports Coffee.Filter, Coffee.Espresso, Coffee.Americano.

The CoffeeMaker needs to grind and brew in order to make coffee.

Task 1:

Create enum CoffeeType, which has at least three enum constants: Filter, Espresso, Americano

Create the Coffee Class, and it has two attributes: CoffeeType and quantity. Create Getters/Setters and Constructor(s). We can add an exception handler which ensures that the volume needs to be larger than zero, otherwise a CoffeeException will be thrown.

Create a CoffeeConfiguration class, and it has two attributes: quantityCoffee, and quantityWater. Make Getters/Setters and A Constructor.

Create CoffeeException Class, and implement several Constructors.

Task 2: Create the AbstractCoffeeMaker Class. Make it abstract.

It has one attribute called HashMap<CoffeeType, CoffeeConfiguration > configMap. You will need to get CoffeeConfiguration based on CoffeeType before making a coffee. However, in this abstract class, you don't need to initiate the HashMap.

It has one abstract methods called brewCoffee. It accepts one parameter which is CoffeeType, and return Coffee.

Task 3: Create a BasicCoffeeMaker Class, which extends the abstractCoffeeMaker Class. It is composed of a Grinder, a BrewingUnit. Think about aggregation we have learned.

A BasicCoffeeMaker only supports CoffeeType.Filter. For any CoffeeType other than FILTER, a CoffeeException will be thrown.

You will need to initiate configMap so that it supports CoffeeType.Filter. for example,

```
this.configMap.put(CoffeeType.Filter, new CoffeeConfiguration(30, 180));
```

Create a method called Coffee brewFilterCoffee(). The return type is Coffee, which has been created in Task 1. You leave brewFilterCoffee() method body as empty at this point.

Task 4: Compile-time polymorphism comes into play!

In BasicCoffeeMaker class, you will need to implement two methods with the same name brewCoffee. Make it as compile-time polymorphism. The first one accepts one parameter which is CoffeeType, the second one accepts two parameters, CoffeeType and number. Both brewCoffee methods will call brewFilterCoffee() method. The difference is that brewCoffee(CoffeeType coffeeType) returns Coffee, while brewCoffee(CoffeeType coffeeType, int number) returns ArrayList<Coffee>.

Task 5:

Create the Grinder Class. It has one method grind(). The method signature is GroundCoffee grind(CoffeeType).

Create GroundCoffee Class. Leave the GroundCoffee Class body empty.

Create BrewingUnit class. This class has one method brew(). The method signature is Coffee brew(CoffeeType, GroundCoffee).

Task 6: Runtime polymorphism comes into play!

Create a PremiumCoffeeMaker Class, which is a child class of BasicCoffeeMaker.

A PremiumCoffeeMaker supports Filter, Espresso, Americano. Other coffee types are not supported and a CoffeeException will be thrown.

You will need to initiate configMap so that it supports CoffeeType.Filter, CoffeeType.Espresso, and CoffeeType.Americano. for example,

```
this.configMap.put(CoffeeType.Filter, new CoffeeConfiguration(30, 180));
this.configMap.put(CoffeeType.Espresso, new CoffeeConfiguration(30, 30));
this.configMap.put(CoffeeType.Americano, new CoffeeConfiguration(30, 100));
```

You will also need to override brewCoffee(CoffeeType coffeeType) method, so that in case of CoffeeType.Espresso, it calls brewEspresso(); in case of CoffeeType.Americano, it calls brewAmericano(). By default, it calls brewCoffee from its parent class. Try to use switch statement.

In addition, you will implement brewEspresso() and brewAmericano().

Think: do we need to override brewCoffee(CoffeeType coffeeType, int number) method?

Task 7: for all tasks, try to rephase it using polymorphism, for example upcast, downcasting, this, super, constructor overloading, etc.

Good luck and having fun!