Builder Pattern

Lecture-4

Problem

Consider a business case of fast-food restaurant where a typical meal could be a burger and a cold drink. Burger could be either a Veg Burger or Chicken Burger and will be packed by a wrapper. Cold drink could be either a coke or pepsi and will be packed in a bottle.

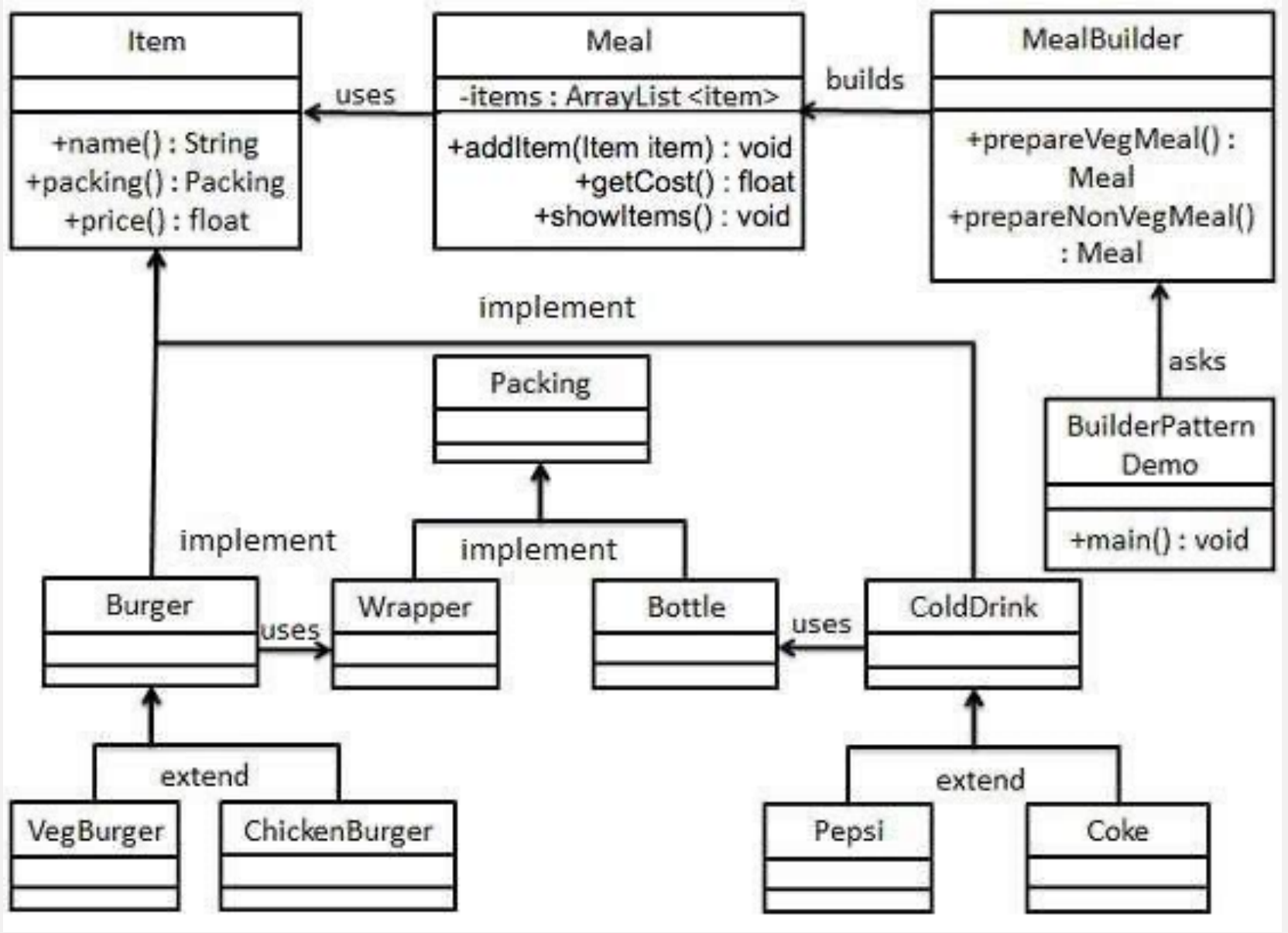
Builder Pattern

- Builder Pattern says that **"construct a complex object from simple objects using a step-by-step approach"**
- It is mostly used when an object can’t be created in a single step
- Builder pattern aims to “Separate the construction of a complex object from its representation so that the same construction process can create different representations.”

Advantages

- It provides a clear separation between the construction and representation of an object.
- It provides better control over the construction process.
- It supports changing the internal representation of objects.

Builder Pattern Diagram



STEP: 1

```java
// Item.java
public interface Item {
	public String name();
	public Packing packing();
	public float price();
}

// Packaging.java
public interface Packing {
	public String pack();
}
```

STEP: 2

```java
// Wrapper.java
public class Wrapper implements Packing {
	@Override public String pack() {
		return "Wrapper";
	}
}

// Bottle.java
public class Bottle implements Packing {
	@Override public String pack() {
		return "Bottle";
	}
}
```

STEP: 3

```java
// Burger.java
public abstract class Burger implements Item {
	@Override public Packing packing() {
		return new Wrapper();
	}
	@Override public abstract float price();
	@Override public abstract String name();
}

// ColdDrink.java
public abstract class ColdDrink implements Item {
	@Override public Packing packing() {
		return new Bottle();
	}
	@Override public abstract float price();
	@Override public abstract String name();
}
```

Step: 4

```
// VegBurger.java

public class VegBurger extends Burger {
        @Override public float price() { return 25.0f; }
        @Override public String name() { return "Veg Burger"; }
}

// ChickenBurger.java

public class ChickenBurger extends Burger {
        @Override public float price() { return 50.5f; }
        @Override public String name() { return "Chicken Burger"; }
}


// Coke.java

public class Coke extends ColdDrink {
        @Override public float price() { return 30.0f; }
        @Override public String name() { return "Coke"; }
}

// Pepsi.java

public class Pepsi extends ColdDrink {
        @Override public float price() { return 35.0f; }
        @Override public String name() { return "Pepsi"; }
}
```

STEP: 5

```
// Meal.java

public class Meal {
        private List<Item> items = new ArrayList<Item>();
        public void addItem(Item item) { items.add(item); }
        public float getCost() {
                float cost = 0.0f;
                for (Item item : items) {
                        cost += item.price();
                }
                return cost;
        }
        public void showItems() {
                for (Item item : items) {
                        System.out.print("Item : " + item.name());
                        System.out.print(", Packing : " + item.packing().pack());
                        System.out.println(", Price : " + item.price());
                }
        }
}
```

Step: 6

```
// MealBuilder.java

public class MealBuilder {
        public Meal prepareVegMeal () {
                Meal meal = new Meal();
                meal.addItem(new VegBurger());
                meal.addItem(new Coke());
                return meal;
        }

        public Meal prepareNonVegMeal () {
                Meal meal = new Meal();
                meal.addItem(new ChickenBurger());
                meal.addItem(new Pepsi());
                return meal;
        }
}
```

STEP: 7

```
// Demo.java
public class BuilderPatternDemo {
        public static void main(String[] args) {
                MealBuilder mealBuilder = new MealBuilder();
                Meal vegMeal = mealBuilder.prepareVegMeal();

                System.out.println("Veg Meal"); vegMeal.showItems();
                System.out.println("Total Cost: " + vegMeal.getCost());

                Meal nonVegMeal = mealBuilder.prepareNonVegMeal();
                System.out.println("\n\nNon-Veg Meal");

                nonVegMeal.showItems();
                System.out.println("Total Cost: " + nonVegMeal.getCost());
        }
}
```