Builder Pattern

The Builder Pattern is a design pattern used in object-oriented programming that allows for the creation of complex objects step-by-step by separating the construction of an object from its representation.

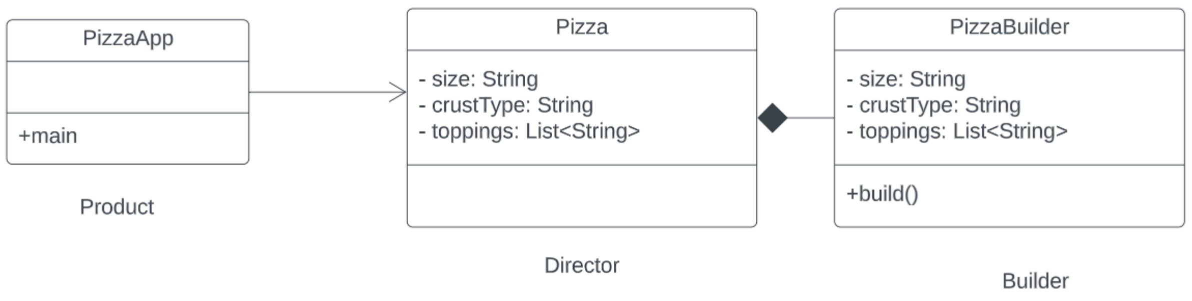
The main idea behind this pattern is to provide a flexible and easy-to-use interface to build different types of objects without having to expose their internal details or complexities. It is particularly useful when dealing with complex objects that require multiple steps to construct or when there are different ways to create an object.

In the Builder pattern, a director class is responsible for managing the construction process and coordinating the steps involved in creating the object. A builder class is responsible for defining the individual steps involved in creating the object, while a product class represents the final object being constructed.

The key benefit of using the Builder pattern is that it allows for the creation of objects with a flexible and intuitive API, while also providing a way to hide the implementation details from the client code. This can make code easier to maintain, extend, and test, and can also improve the overall readability and usability of the codebase.

Example of Builder Pattern

A pizza restaurant offers a wide variety of pizza sizes, crust types, and toppings. Customers should be able to order pizzas with different combinations of these attributes, but the order process should be as simple and user-friendly as possible. The restaurant needs a way to build pizza objects with the correct combination of attributes, while ensuring that the order is accurate and consistent.



The Builder pattern can be used to provide a clear and easy-to-use interface for building pizza objects. The Pizza class can contain a static inner class PizzaBuilder, which allows customers to specify the pizza size, crust type, and toppings in a fluent and easy-to-read syntax.

The PizzaBuilder class can also include validation logic to ensure that the pizza order is complete and accurate before the pizza is built.

```
package builder;
import java.util.*;

public class Pizza {
   protected final String size;
   protected final String crustType;

   private Pizza(PizzaBuilder builder){
      this.size = builder.size;
      this.crustType = builder.crustType;

   }

   protected static class PizzaBuilder{
      private String size;
      private String crustType;

      public PizzaBuilder size(String size){
         this.size = size;
         return this;
      }

      public PizzaBuilder crustType (String crustType){
         this.crustType = crustType;
         return this;
      }

      public Pizza build(){
         return new Pizza(this);
      }
   }
}
```

```
public class PizzaApp {
   public static void main(String[] args){
        Pizza myPizza = new Pizza.PizzaBuilder()
                .crustType("Thick")
                .size("18")
                .build();

         System.out.println(myPizza.size);
         System.out.println(myPizza.crustType);

}
```