Xideral

Java Academy

Week 3-Day 2

Builder pattern

Presented by:

Edgar Itzak Sánchez Rogers

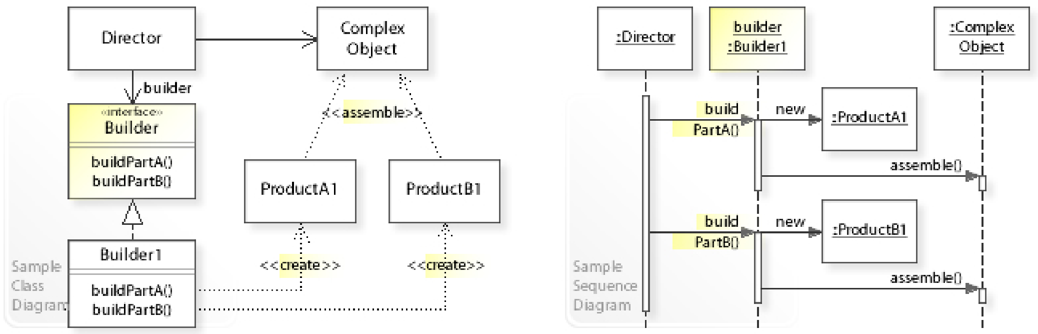
Monterrey Nuevo León México at 28 august 2024

Introduction:

Builder pattern is a creational design pattern that can create complex objects step by step. The builder patterns separate the construction of a complex object from its representation.

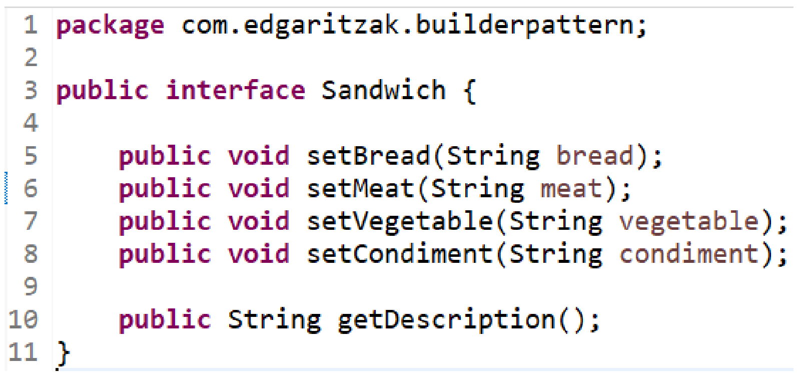
Parts of the builder pattern:

- Builder Interface: Defines the methods that builders must define
- Builder: Builder class creates the instance of the object and has the methods to set the attributes
- Director: The director class defines the order in which to execute the building steps, while the builder provides the implementation for those steps.



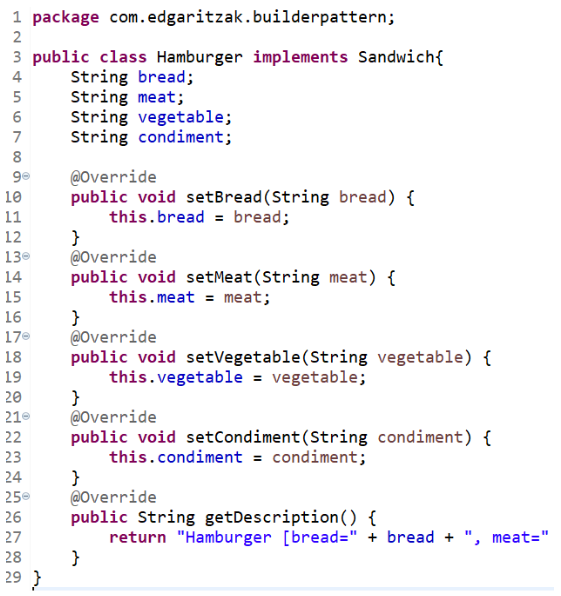
Builder pattern demonstration:

Interface Object class (Sandwich):

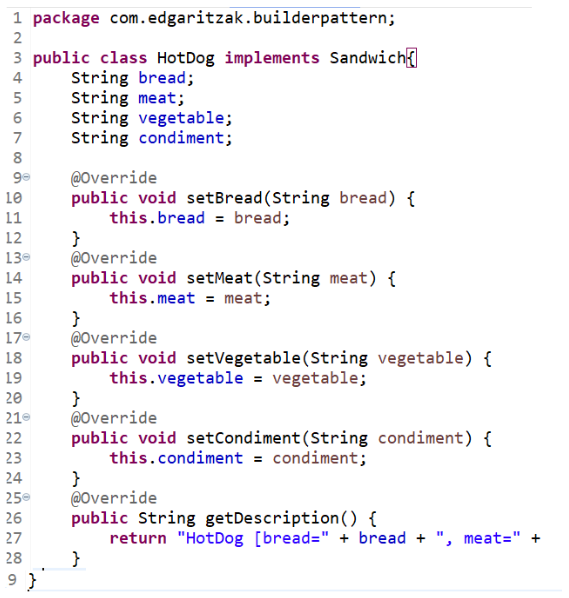


```java
package com.edgaritzak.builderpattern;

public interface Sandwich {

    public void setBread(String bread);
    public void setMeat(String meat);
    public void setVegetable(String vegetable);
    public void setCondiment(String condiment);

    public String getDescription();
}
```

Object (Sandwich) HotDog & Hamburger classes:

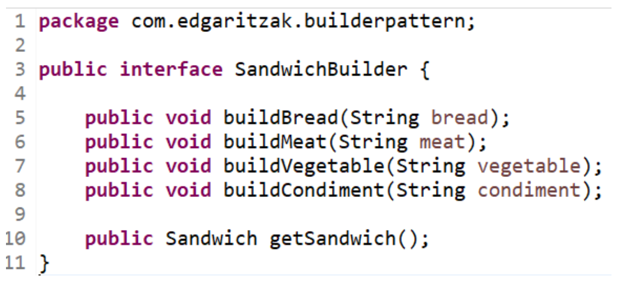


```java
package com.edgaritzak.builderpattern;

public class Hamburger implements Sandwich{
    String bread;
    String meat;
    String vegetable;
    String condiment;

    @Override
    public void setBread(String bread) {
        this.bread = bread;
    }
    @Override
    public void setMeat(String meat) {
        this.meat = meat;
    }
    @Override
    public void setVegetable(String vegetable) {
        this.vegetable = vegetable;
    }
    @Override
    public void setCondiment(String condiment) {
        this.condiment = condiment;
    }
    @Override
    public String getDescription() {
        return "Hamburger [bread=" + bread + ", meat="
    }
}
```



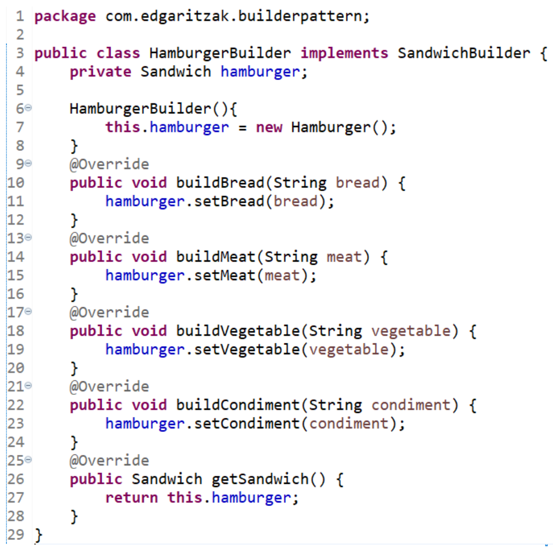
```java
package com.edgaritzak.builderpattern;

public class HotDog implements Sandwich{
    String bread;
    String meat;
    String vegetable;
    String condiment;

    @Override
    public void setBread(String bread) {
        this.bread = bread;
    }
    @Override
    public void setMeat(String meat) {
        this.meat = meat;
    }
    @Override
    public void setVegetable(String vegetable) {
        this.vegetable = vegetable;
    }
    @Override
    public void setCondiment(String condiment) {
        this.condiment = condiment;
    }
    @Override
    public String getDescription() {
        return "HotDog [bread=" + bread + ", meat=" +
    }
}
```

Interface Builder class:

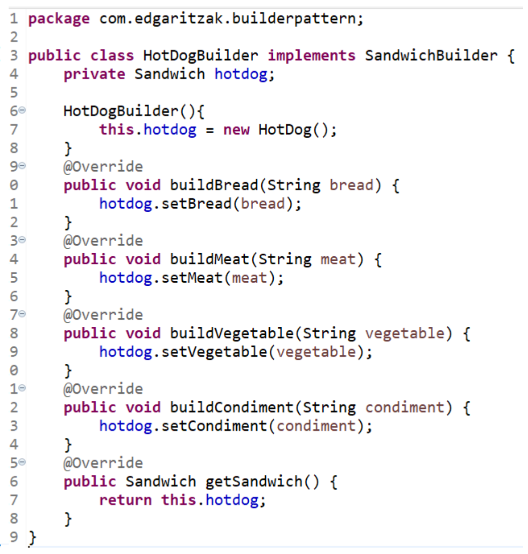


```java
package com.edgaritzak.builderpattern;

public interface SandwichBuilder {

    public void buildBread(String bread);
    public void buildMeat(String meat);
    public void buildVegetable(String vegetable);
    public void buildCondiment(String condiment);

    public Sandwich getSandwich();
}
```

Builder classes:

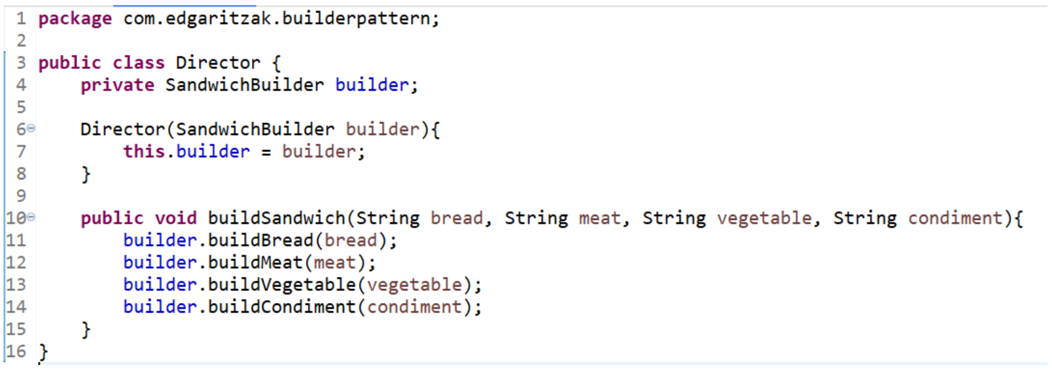


```java
package com.edgaritzak.builderpattern;

public class HamburgerBuilder implements SandwichBuilder {
    private Sandwich hamburger;

    HamburgerBuilder(){
        this.hamburger = new Hamburger();
    }
    @Override
    public void buildBread(String bread) {
        hamburger.setBread(bread);
    }
    @Override
    public void buildMeat(String meat) {
        hamburger.setMeat(meat);
    }
    @Override
    public void buildVegetable(String vegetable) {
        hamburger.setVegetable(vegetable);
    }
    @Override
    public void buildCondiment(String condiment) {
        hamburger.setCondiment(condiment);
    }
    @Override
    public Sandwich getSandwich() {
        return this.hamburger;
    }
}
```



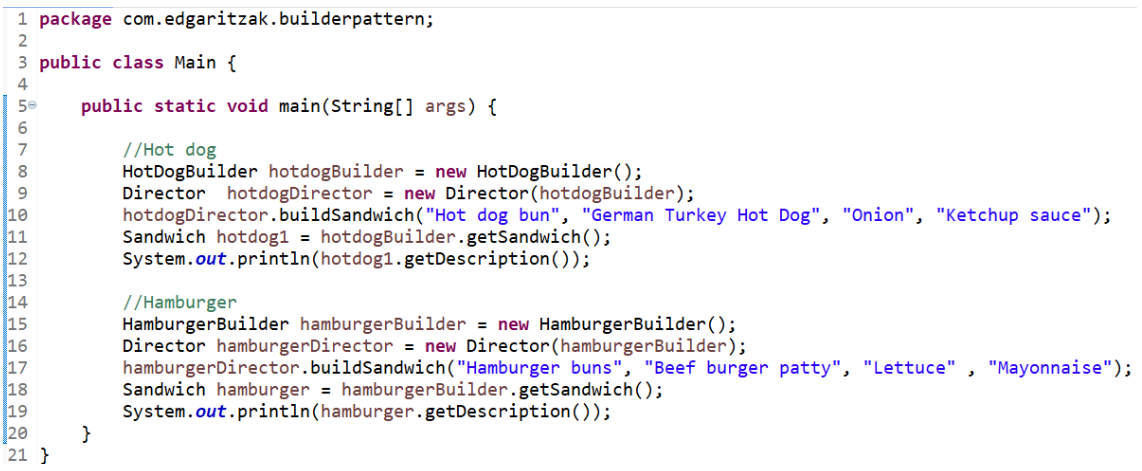
```java
package com.edgaritzak.builderpattern;

public class HotDogBuilder implements SandwichBuilder {
    private Sandwich hotdog;

    HotDogBuilder(){
        this.hotdog = new HotDog();
    }
    @Override
    public void buildBread(String bread) {
        hotdog.setBread(bread);
    }
    @Override
    public void buildMeat(String meat) {
        hotdog.setMeat(meat);
    }
    @Override
    public void buildVegetable(String vegetable) {
        hotdog.setVegetable(vegetable);
    }
    @Override
    public void buildCondiment(String condiment) {
        hotdog.setCondiment(condiment);
    }
    @Override
    public Sandwich getSandwich() {
        return this.hotdog;
    }
}
```

Director class:



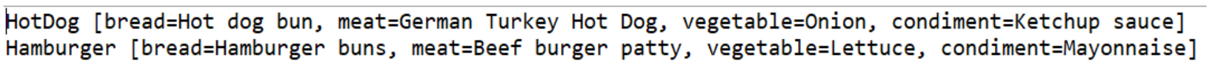
```java
package com.edgaritzak.builderpattern;

public class Director {
    private SandwichBuilder builder;

    Director(SandwichBuilder builder){
        this.builder = builder;
    }

    public void buildSandwich(String bread, String meat, String vegetable, String condiment){
        builder.buildBread(bread);
        builder.buildMeat(meat);
        builder.buildVegetable(vegetable);
        builder.buildCondiment(condiment);
    }
}
```

Main class:



```java
package com.edgaritzak.builderpattern;

public class Main {

    public static void main(String[] args) {

        //Hot dog
        HotDogBuilder hotdogBuilder = new HotDogBuilder();
        Director  hotdogDirector = new Director(hotdogBuilder);
        hotdogDirector.buildSandwich("Hot dog bun", "German Turkey Hot Dog", "Onion", "Ketchup sauce");
        Sandwich hotdog1 = hotdogBuilder.getSandwich();
        System.out.println(hotdog1.getDescription());

        //Hamburger
        HamburgerBuilder hamburgerBuilder = new HamburgerBuilder();
        Director hamburgerDirector = new Director(hamburgerBuilder);
        hamburgerDirector.buildSandwich("Hamburger buns", "Beef burger patty", "Lettuce" , "Mayonnaise");
        Sandwich hamburger = hamburgerBuilder.getSandwich();
        System.out.println(hamburger.getDescription());
    }
}
```

Output:



```
HotDog [bread=Hot dog bun, meat=German Turkey Hot Dog, vegetable=Onion, condiment=Ketchup sauce]
Hamburger [bread=Hamburger buns, meat=Beef burger patty, vegetable=Lettuce, condiment=Mayonnaise]
```

Explanation:

In this example is created a sandwich object interface, 2 objects that implement the sandwich interface, hotdog and hamburger. In addition, a builder interface is created and for each of the sandwiches a builder is created. Finally, the Director class is created, which will use the methods of the builders to create the objects.

To test the operation of the builder design pattern, builders and a director are defined. The functions .buildSandwich(), .getSandwich() and .getDescription() are executed.

Conclusion:

The Builder design pattern in Java provides an efficient solution for the creation of complex objects. This pattern allows the step-by-step construction of objects and facilitates the creation of different representations of the same type of object. Its strengths include the clear separation between the construction of an object and its representation.

References:

[1] Builder pattern - Wikipedia

[2] Builder (refactoring.guru)

[3] Builder Design Pattern - GeeksforGeeks