Karen Giselle Valdez Muñoz

Builder Pattern

Introduction

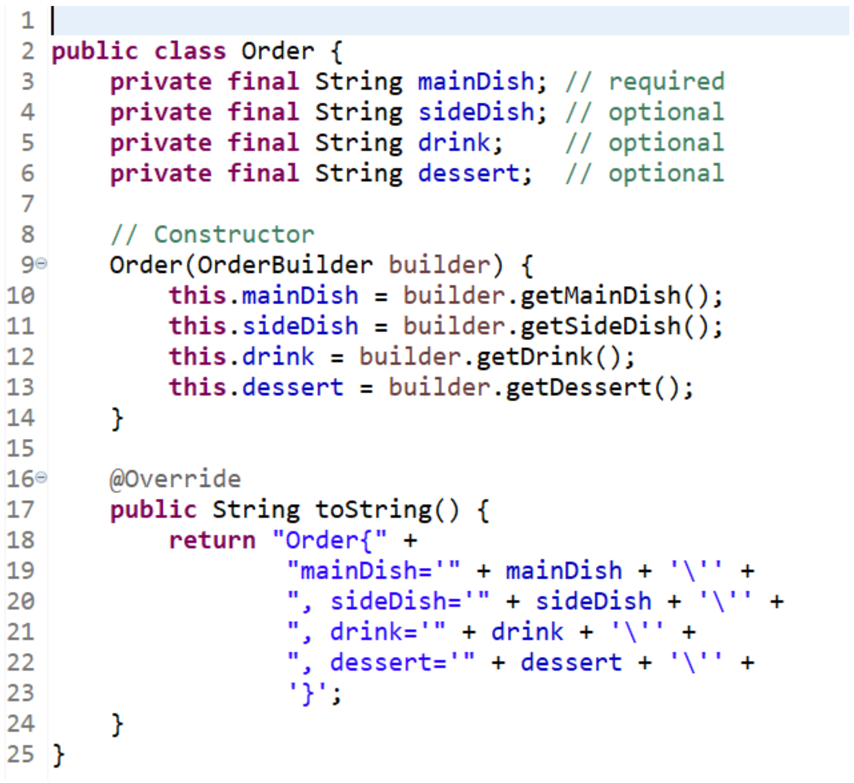
The Builder pattern is a creational design pattern that allows for the step-by-step creation of complex objects. This pattern is particularly useful when an object contains multiple optional attributes. The goal of this document is to explain the implementation of the Builder pattern in a restaurant order system where each part of the order (main dish, side dish, drink, dessert) can be optionally specified.

Class Breakdown

1. Order Class

File: Order.java

Purpose: The Order class represents a final, immutable object that contains the details of a customer's order. It has fields for the main dish, side dish, drink, and dessert, all of which are set via the OrderBuilder.



```java

public class Order {
    private final String mainDish; // required
    private final String sideDish; // optional
    private final String drink;    // optional
    private final String dessert;  // optional

    // Constructor
    Order(OrderBuilder builder) {
        this.mainDish = builder.getMainDish();
        this.sideDish = builder.getSideDish();
        this.drink = builder.getDrink();
        this.dessert = builder.getDessert();
    }

    @Override
    public String toString() {
        return "Order{" +
                "mainDish='" + mainDish + '\'' +
                ", sideDish='" + sideDish + '\'' +
                ", drink='" + drink + '\'' +
                ", dessert='" + dessert + '\'' +
                '}';
    }
}
```

Fields:

- private final String mainDish: The main dish is a required attribute.
- private final String sideDish: The side dish is optional.
- private final String drink: The drink is optional.
- private final String dessert: The dessert is optional.

Constructor:

The constructor for the Order class takes an OrderBuilder object as a parameter and extracts the values from it to set its own fields. The constructor is package-private, allowing only the OrderBuilder class to instantiate Order.

Methods:

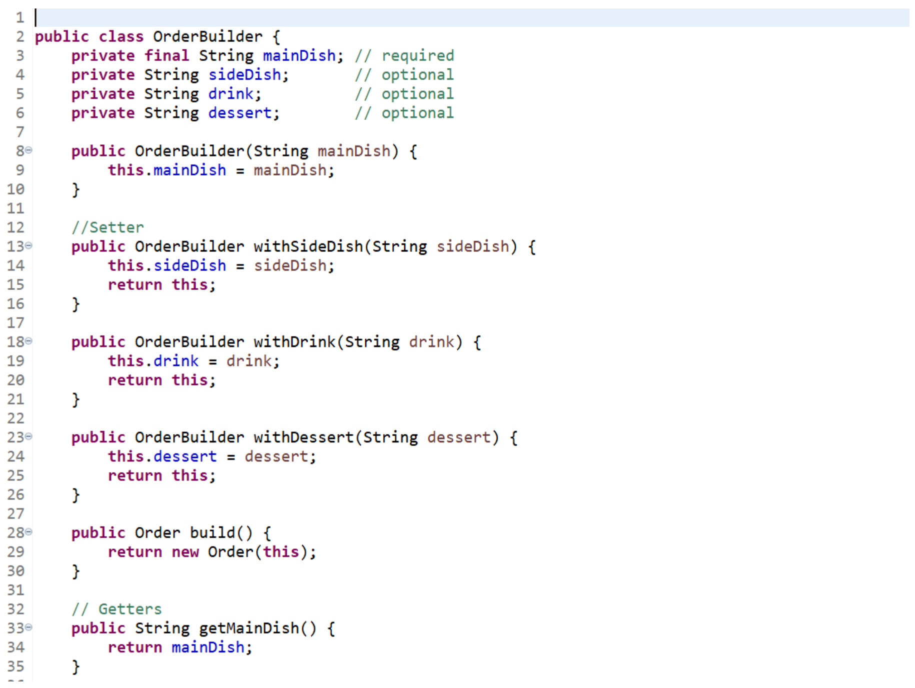
- toString(): This method is overridden to provide a string representation of the Order object, displaying all the components of the order.

Key Point:

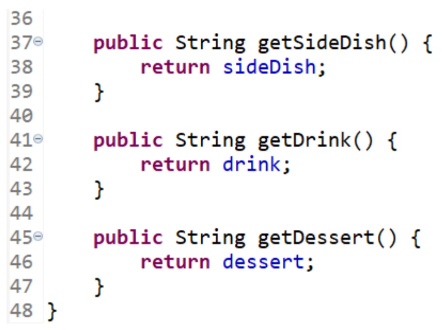
The Order class is immutable, meaning once it is created, its fields cannot be changed. This immutability is crucial for maintaining the integrity of the order.

2. OrderBuilder Class

File: OrderBuilder.java



```java

public class OrderBuilder {
    private final String mainDish; // required
    private String sideDish;       // optional
    private String drink;          // optional
    private String dessert;        // optional

    public OrderBuilder(String mainDish) {
        this.mainDish = mainDish;
    }

    //Setter
    public OrderBuilder withSideDish(String sideDish) {
        this.sideDish = sideDish;
        return this;
    }

    public OrderBuilder withDrink(String drink) {
        this.drink = drink;
        return this;
    }

    public OrderBuilder withDessert(String dessert) {
        this.dessert = dessert;
        return this;
    }

    public Order build() {
        return new Order(this);
    }

    // Getters
    public String getMainDish() {
        return mainDish;
    }
```



```
    public String getSideDish() {
        return sideDish;
    }

    public String getDrink() {
        return drink;
    }

    public String getDessert() {
        return dessert;
    }
}
```

Purpose: The OrderBuilder class provides a flexible way to construct an Order object. It allows setting optional components (side dish, drink, dessert) and ensures that the required component (main dish) is always set.

Fields:

- private final String mainDish: This is a required field and must be provided during the creation of an OrderBuilder.
- private String sideDish: Optional field for the side dish.
- private String drink: Optional field for the drink.
- private String dessert: Optional field for the dessert.

Constructor:

- OrderBuilder(String mainDish): This constructor initializes the OrderBuilder with the required main dish.

Methods:

- withSideDish(String sideDish): Sets the side dish and returns the OrderBuilder object to allow method chaining.
- withDrink(String drink): Sets the drink and returns the OrderBuilder object to allow method chaining.
- withDessert(String dessert): Sets the dessert and returns the OrderBuilder object to allow method chaining.
- build(): Creates and returns an Order object using the current state of the OrderBuilder.

Getters:

getMainDish(), getSideDish(), getDrink(), getDessert(): These methods provide access to the fields of the OrderBuilder for use by the Order class.

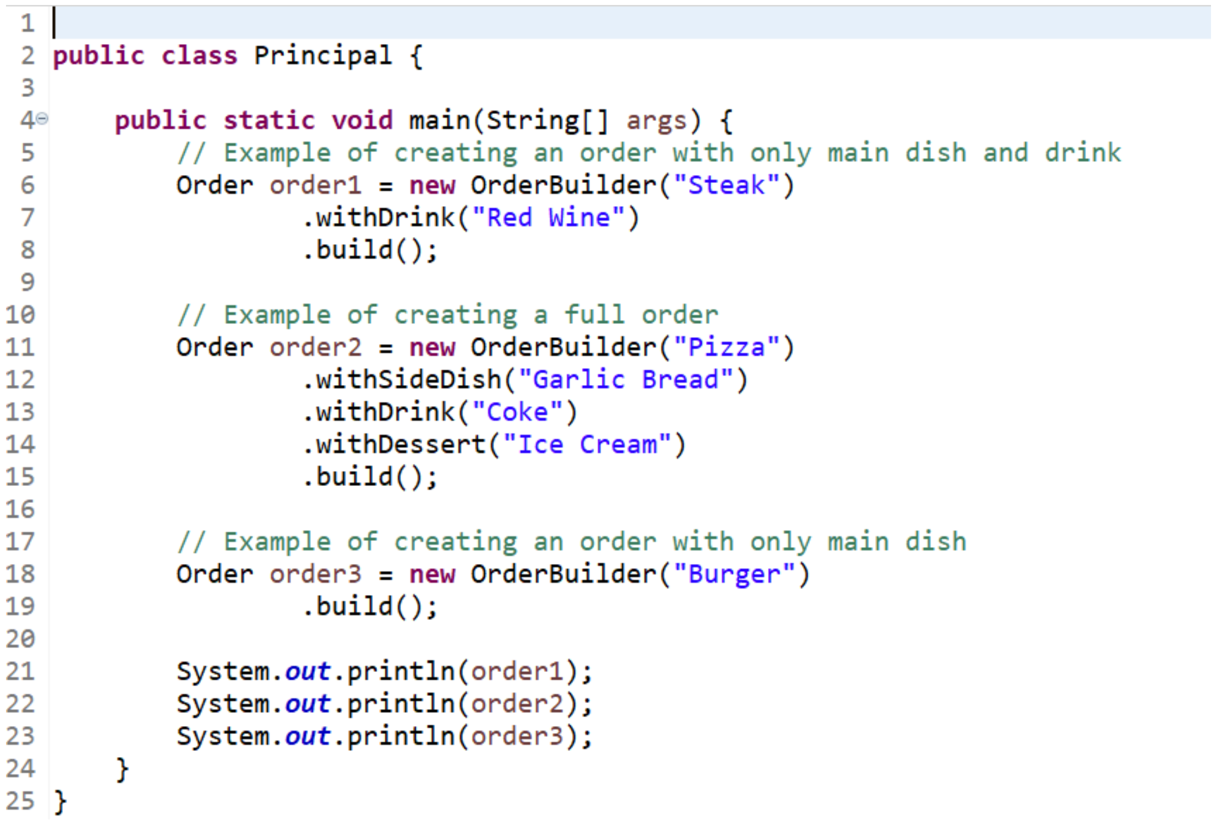
Key Point:

The OrderBuilder class enables a fluent interface, allowing for concise and readable code when constructing an Order.

3. Main Class

File: Main.java

Purpose: The Main class contains the main() method, which demonstrates how to use the OrderBuilder to create various Order objects with different combinations of components.



```java

public class Principal {

    public static void main(String[] args) {
        // Example of creating an order with only main dish and drink
        Order order1 = new OrderBuilder("Steak")
                .withDrink("Red Wine")
                .build();

        // Example of creating a full order
        Order order2 = new OrderBuilder("Pizza")
                .withSideDish("Garlic Bread")
                .withDrink("Coke")
                .withDessert("Ice Cream")
                .build();

        // Example of creating an order with only main dish
        Order order3 = new OrderBuilder("Burger")
                .build();

        System.out.println(order1);
        System.out.println(order2);
        System.out.println(order3);
    }
}
```

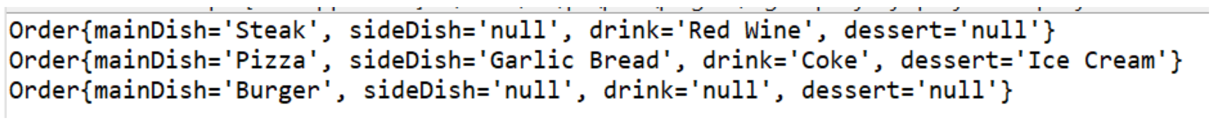
Methods:

- main(String[] args): This method showcases three examples of creating orders:
- An order with only a main dish and a drink.
- A full order with all components.
- An order with only a main dish.

Key Point:

The Main class demonstrates the practical application of the Builder pattern and the flexibility it offers in creating objects.

Output:



```
Order{mainDish='Steak', sideDish='null', drink='Red Wine', dessert='null'}
Order{mainDish='Pizza', sideDish='Garlic Bread', drink='Coke', dessert='Ice Cream'}
Order{mainDish='Burger', sideDish='null', drink='null', dessert='null'}
```

Conclusion

The Builder pattern is a powerful tool for constructing complex objects, especially when some attributes are optional. By separating the object creation logic into a builder class, we gain flexibility and maintainability in our code. This pattern ensures that the final object is always in a valid state and allows for a clean and readable way to build complex objects.