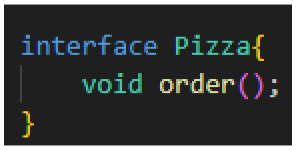
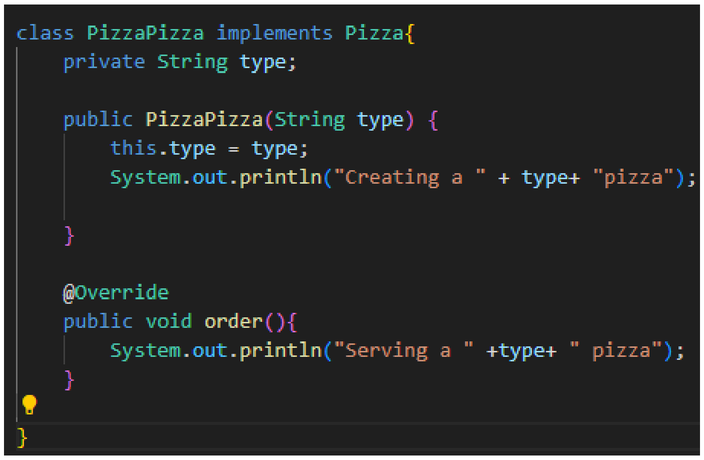
The Proxy pattern is similar to the lazy initialization which means that the proxy pattern will ask for conditions before creating a new subject.



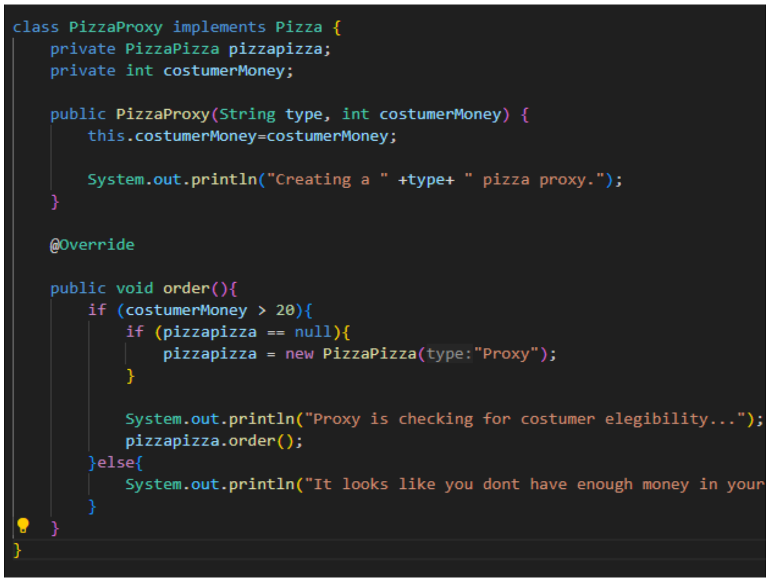
```java
interface Pizza{
    void order();
}
```

This code defines the Pizza interface which has declared a method order().



```java
class PizzaPizza implements Pizza{
    private String type;

    public PizzaPizza(String type) {
        this.type = type;
        System.out.println("Creating a " + type+ "pizza");

    }

    @Override
    public void order(){
        System.out.println("Serving a " +type+ " pizza");
    }

}
```

Here PizzaPizza is a real subject. It implements the Pizza interface and represents the actual pizza object that can be ordered. The order() method prints a message indicating that the pizza of a specific type is being served.



```java
class PizzaProxy implements Pizza {
    private PizzaPizza pizzapizza;
    private int costumerMoney;

    public PizzaProxy(String type, int costumerMoney) {
        this.costumerMoney=costumerMoney;

        System.out.println("Creating a " +type+ " pizza proxy.");
    }

    @Override

    public void order(){
        if (costumerMoney > 20){
            if (pizzapizza == null){
                pizzapizza = new PizzaPizza(type:"Proxy");
            }

            System.out.println("Proxy is checking for costumer elegibility...");
            pizzapizza.order();
        }else{
            System.out.println("It looks like you dont have enough money in your
        }
    }
}
```

This is the PizzaProxy class which implements the Pizza interface. Just like the subject(PizzaPizza). I have put a reference to the subject (PizzaPizza) called pizzapizza.
The constructor will take the pizza type and the costumerMoney as Parameters then it prints a message about creating a pizza proxy and sets the costumer's money.

The order() method will check if the customer has enough money. If this is the case then it will check if the real subject (PizzaPizza) is instantiated. If not then it will create a new PizzaPizza object then it will print a message about customer eligibility and call the order() method on the real subject. If the customer does not have enough money then it will print a message according to it.

There are a lot of benefits of the proxy pattern like Controlled Access, which means that the proxy pattern allows for controlled access to the real object(PizzaPizza). This means that it checks the condition in my case costumerMoney before allowing the order to be placed. Without Proxy, direct access to real subjects might lead to orders being placed without checking the conditions. For my case the customer could order pizzas without having enough funds in the bank account. The Proxy pattern also supports lazy Initialization, meaning that the real subject(PizzaPizza) is only created when needed. This is beneficial for resource efficiency, especially when creating way more complex subjects. Without Proxy in this case, we would also not have lazy initialization which would lead to the subject being created upon starting the application. This could result in unnecessary resource usage, even if the real subject does not need it.

In summary I would say that Proxy is an update of the lazy initialization, which means with lazy initialization an instance is ONLY created when needed, but with Proxy pattern you need to fulfill conditions in order for the instance to be created.