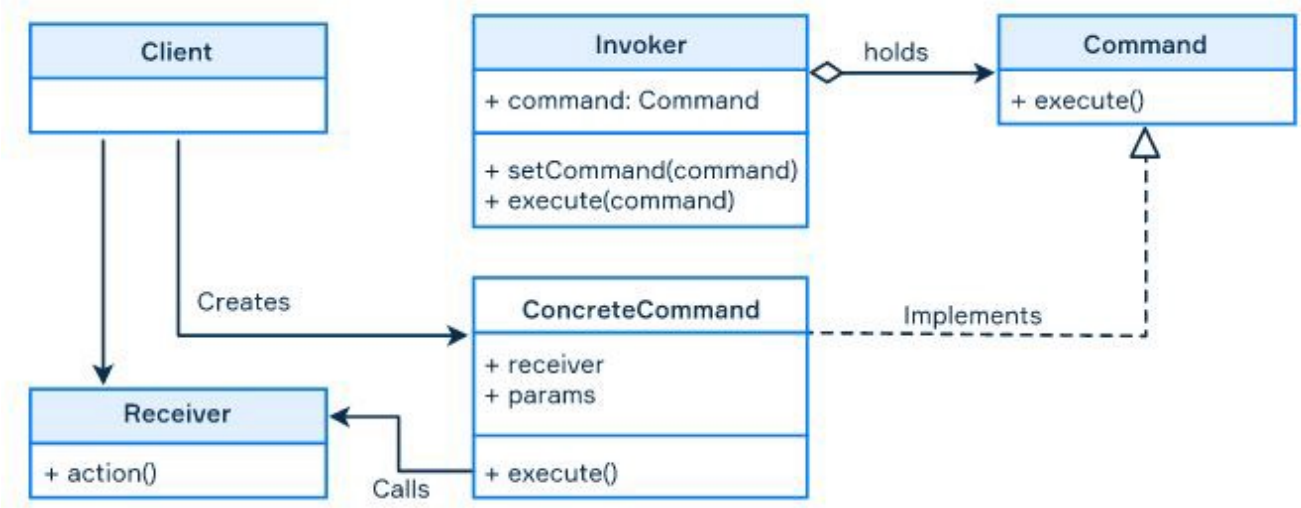
Deelvraag: Hoe implementeer ik het command pattern?

Het standaard ontwerp van een command pattern ziet er als volgt uit:



Hoe?

In een klassiek voorbeeld, kan het geïmplementeerd worden in de volgende 5 stappen:

- De Command interface: deze bevat meestal slechts een enkele methode voor het uitvoeren van de opdracht
- De ConcreteCommand: dit is een operatie met parameters die de oproep doorgeeft aan de ontvanger. Bij benadering roept een opdracht alleen 1 of meerdere methoden aan in plaats van bedrijfslogica uit te voeren
- De Receiver: deze weet hoe de actie uitgevoerd dient te worden
- De Invoker: deze vraagt aan command om het verzoek uit te voeren
- De Client: deze maakt een ConcreteCommand object aan en zet de ontvanger

Voorbeeld: Home automation system, turn on/of heating system

Maak een interface aan, Command

```
public interface Command {
    void execute();
}
```

Nu maak ik 2 klassen aan om de verwarming aan of uit te zetten.
Deze klassen implementeren de interface Command. Hierdoor wordt de data die nogid is om het commando uit te voeren(Heating On, Heating Off) ge-encapsuleerd.

```java
public class HeatingOnCommand implements Command {

    private HeatingSystem heatingSystem;

    public HeatingOnCommand(HeatingSystem heatingSystem) {
        this.heatingSystem = heatingSystem;
    }

    @Override
    public void execute() {
        heatingSystem.heatOn();
    }
}
```

```java
public class HeatingOffCommand implements Command {

    private HeatingSystem heatingSystem;

    public HeatingOffCommand(HeatingSystem heatingSystem) {
        this.heatingSystem = heatingSystem;
    }

    @Override
    public void execute() {
        heatingSystem.heatOff();
    }
}
```

De ontvanger: HeatingSystem

```java
public class HeatingSystem  {

    public void heatOn() {
        System.out.println("Turn on heat");
    }

    public void heatOff() {
        System.out.println("Turn off heat");
    }
}
```

Nu hebben we de Invoker klasse nodig. Deze bepaald hoe de commando's uitgevoerd gaan worden. De Invoker klasse kan een lijst bevatten van commando's welke in een specifieke volgorde uitgevoerd worden.
In dit voorbeeld heet de klasse Controller

```
public class Controller {

    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void executeCommand() {
        command.execute();
    }
}
```

Als laatste, de Client

```
public class HomeHeatingSystemAutomation {

    public static void main(String\[\] args) {

        Controller controller = new Controller();
        HeatingSystem heatingSystem = new HeatingSystem();

        Command heatOn = new HeatingOnCommand(light);
        Command heatOff = new HeatingOffCommand(light);

        controller.setCommand(heatOn);
        controller.executeCommand();

        controller.setCommand(heatOff);
        controller.executeCommand();
    }
}
```

De 3 belangrijkste stappen die in de Client afspelen zijn:

1. Het object van de Invoker klasse wordt aangemaakt. In dit voorbeeld is het Controller

2. Aanmaken van objecten van de commandos die uitgevoerd gaan worden
3. De commando's uitvoeren door de aanroeper te gebruiken.

Voorbeeld 2: Een bestelling in een restaurant

Wat is er nodig:

1. Een interface Command
2. Een klasse Order dat de interface Command implementeerd
3. Een klasse Ober(invoker)
4. Een klasse Kok(ontvanger)

De klasse Chef, de ontvanger

```
public class Chef {
    public void cookPasta() {
        System.out.println(“Chef is cooking Chicken Alfredo…”);
    }

    public void bakeCake() {
        System.out.println(“Chef is baking Chocolate Fudge Cake…”);
    }
}
```

De interface Command

```
public interface Command {
    public abstract void execute();
}
```

De klasse Order

```
public class Order implements Command {
    private Chef chef;
    private String food;

    public Order(Chef chef, String food) {
        this.chef = chef;
        this.food = food;
    }

    @Override
    public void execute() {
```

```
        if (this.food.equals(“Pasta”)) {
            this.chef.cookPasta();
        } else {
            this.chef.bakeCake();
        }
    }
}
```

De Ober, de invoker

```
public class Waiter {
    private Order order;

    public Waiter(Order ord) {
        this.order = ord;
    }

    public void execute() {
        this.order.execute();
    }
}
```

De klant, de client

```
public class Client {
    public static void main(String[] args) {
        Chef chef = new Chef();

        Order order = new Order(chef, “Pasta”);
        Waiter waiter = new Waiter(order);
        waiter.execute();

        order = new Order(chef, “Cake”);
        waiter = new Waiter(order);
        waiter.execute();
    }
}
```

Zoals je hierboven ziet, de Client maakt een bestelling(Order) en zet de ontvanger op kok(Chef). De bestelling wordt verzonden naar de ober. Deze weet wanneer de bestelling uitgevoerd moet worden(wanneer deze aan de kok gegeven moet worden

zodat deze kan beginnen met koken). Wanneer de ober de bestelling heeft uitgevoerd, de methode excute van de bestelling wordt uitgevoerd door de ontvanger(de kok, deze krijgt het commando bereid paste of een cake).

Bronnen:
Prototyping