Deelvraag: Hoe wordt inkapsulatie bereikt?

Het command patroon is een gedrags pattern, het is gericht op de communicatie tussen de zender en ontvanger.
Het inkapsuleerd alle informatie die nodig is om een actie of bij een speciale gebeurtenis op een later tijdstip een actie uit te voeren. De informatie bevat de naam van de methode, het object dat de methode bevat en de waardes voor deze methode.

Een verzoek wordt geinkapsuleerd als een object, hierdoor is het mogelijk om clients met verschillende verzoeken, wachtrij- of logverzoeken te parametriseren en onuitvoerbare bewerkingen te ondersteunen.

Waarom?

Het command patroon ontkoppelt het object dat de bewerking aanroept van het object dat weet hoe het uitgevoerd moet worden. Om deze scheiding te bereiken, maakt de ontwerper een abstracte basisklasse die een ontvanger (een object) in kaart brengt met een actie. De basisklasse bevat een methode execute() die de actie eenvoudig op de ontvanger aanroept.

Dit geeft de volgende voordelen:

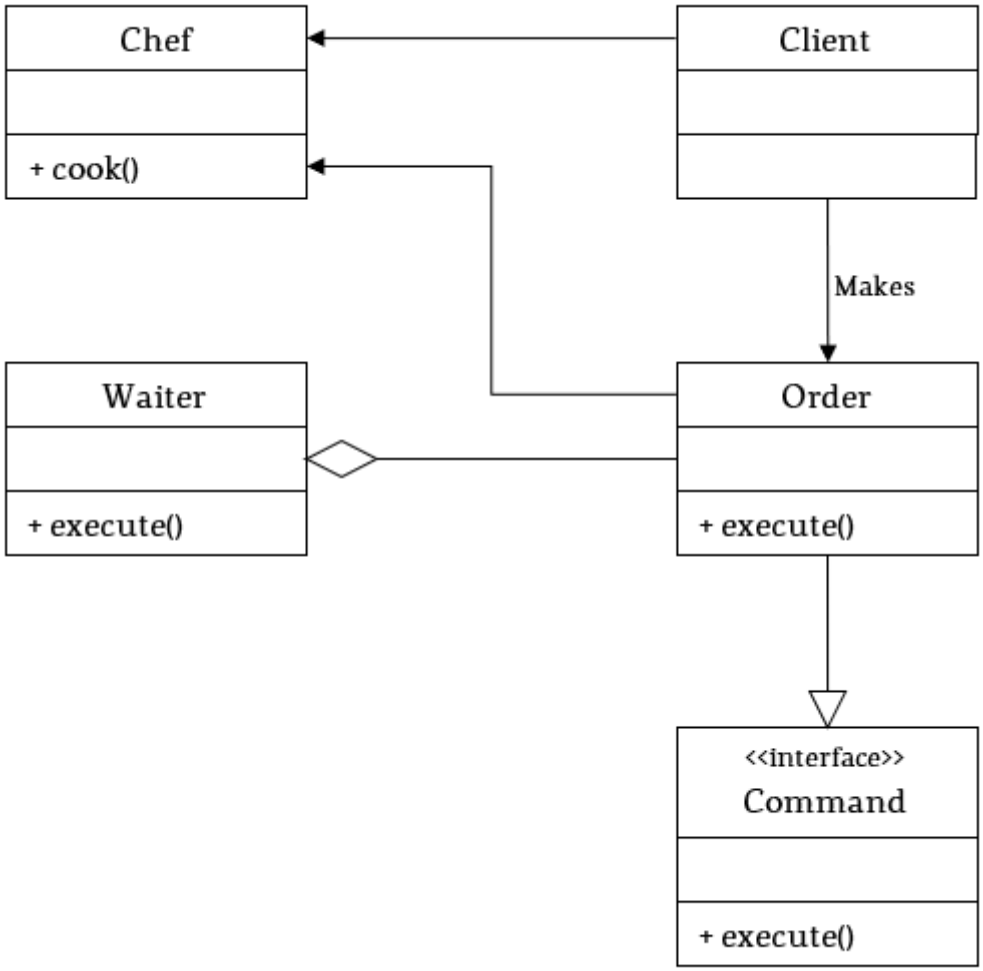
- Een verzoek wordt ingekapseld als een object
- Clients kunnen geparametriseerd verzoeken ontvangen
- Object georiënteerde callback

Het doel hiervan is om in hogere mate een losse koppeling tussen betrokken klassen te cree:eren. Hierdoor zijn de klassen niet meer afhankelijk van elkaar.

Hoe?

De aanroepende klasse(caller) is ontkoppeld van de klasse die de daadwerkelijke actie uitvoerd. De aanroepende klasse heeft alleen een methode *execute*, deze voert het benodigde commando uit wanneer de client dit verzoekt.

Voorbeeld, een bestelling in een restaurant. Je zit in een restaurant en je geeft(command) de ober(zender) je bestelling(commando), de ober geeft vervolgens de bestelling door aan de kok(ontvanger), hierdoor krijg jij je eten. Het lijkt simpel maar er zit een speciaal sausje over de code om dit te bewerktstelligen.



Klassendiagram van het voorbeeld volgens het command design pattern

Bronnen

Soure Making - Command Design Pattern
Wikipedia - Command Pattern
GeeksforGeeks - Command Pattern
Info World - Java Tip 68: Learn how to implement the Command pattern in Java
Design pattern research