Gago Khachatryan, Stan Wiekstra

Pizza-case

Software en Netwerkarchitectuur

Inhoud

[1. Wat is er ingeleverd 2](#_Toc149645913)

[2. Design patterns 3](#_Toc149645914)

[3. Netwerkcommunicatie 5](#_Toc149645915)

# Wat is er ingeleverd

Ingeleverd is een programma waarmee je via een client-code een pizza kan bestellen. Bij de bestelling moet je je persoonsgegevens invullen (NAW) en daarna kan je kiezen wat voor pizza je wilt. Er is een lijst gemaakt van pizza’s, deze lijst is makkelijk om aan te vullen indien er meerdere opties moeten zijn. Na het kiezen van de gewenste pizza kan je er voor kiezen om extra toppings op je pizza te doen. Elke pizza heeft een standaard prijs en per topping komt er een prijs verhoging. Als je klaar bent met bestellen kan je de order doorsturen via de server-code. De order komt dan binnen op de server en zal zichtbaar zijn op een website (indien gekozen is voor HTTP-communicatie) of in de terminal van de server-code. Er is gebruik gemaakt van verschillende design patterns, hierover later meer. Ook zijn er verschillende netwerkcommunicatie mogelijkheden. Het programma is zo gemaakt dat het niet uit maakt welk netwerkcommunicatie je kiest, het resultaat zou hetzelfde zijn.

# Design patterns

De gegeven code maakt gebruik van verschillende designpatronen om de code georganiseerd, herbruikbaar en onderhoudbaar te maken. Hier zijn enkele van de designpatronen die worden gebruikt en een beschrijving van hoe ze zijn geïmplementeerd:

1. **Singleton Pattern**:
   * Het Singleton-patroon wordt gebruikt in de CommunicationStrategiesSingleton-klasse om ervoor te zorgen dat er slechts één instantie van de klasse is, die verantwoordelijk is voor het beheren van de communicatiestrategieën (TCP, UDP, HTTP). Dit zorgt ervoor dat er maar één keer verbinding wordt gemaakt met een specifieke communicatiemethode, wat efficiënter is.
   * Waarom: Het gebruik van Singleton zorgt voor een enkele bron van waarheid voor communicatiestrategieën, waardoor inconsistente toestanden worden vermeden en de middelen effectief worden beheerd.
2. **Visitor Pattern**:
   * Het Visitor-patroon wordt gebruikt in de klassen OrderVisitor, Pizza, en Order om gedrag (bezoek) toe te voegen aan verschillende soorten objecten (pizza's en bestellingen) zonder de klassen zelf te wijzigen.
   * Waarom: Het Visitor-patroon helpt om nieuwe operaties toe te voegen zonder de bestaande klassen te wijzigen, waardoor de open-gesloten principe (OCP) wordt nageleefd en de codebase flexibeler wordt.
3. **Iterator Pattern**:
   * Het Iterator-patroon wordt gebruikt in de Order-klasse om iteratie over de bestelitems mogelijk te maken zonder dat de interne implementatie van de lijst met items wordt blootgesteld.
   * Waarom: Het Iterator-patroon verbetert de modulariteit en maakt het gemakkelijk om door de lijst van bestelitems te gaan zonder te hoeven weten hoe deze intern is georganiseerd.
4. **Strategy Pattern:**
   * Het Strategy Pattern wordt gebruikt om verschillende communicatiestrategieën (TCP, UDP, HTTP) te definiëren en te koppelen aan de geselecteerde communicatiemethode. De concrete strategieën zijn geïmplementeerd als afzonderlijke klassen (TcpCommunication, UdpCommunication, HttpCommunication), en ze implementeren een gemeenschappelijke interface (CommunicationStrategy).
   * Waarom: Dit patroon maakt het gemakkelijk om nieuwe communicatiestrategieën toe te voegen zonder de bestaande code te wijzigen. Het maakt de code flexibel en aanpasbaar aan verschillende behoeften.

Ook wordt er gebruik gemaakt van verschillende securitypatronen om de code veilig te maken. Hier zijn enkele van de securitypatronen die worden gebruikt:

1. **Input Security-patroon:**
   * Het Input Security-patroon is toegepast in de clientcode om de gebruikersinvoer te valideren en te beveiligen tegen mogelijke aanvallen. Door de invoer te controleren op geldigheid voordat deze wordt verwerkt, wordt de robuustheid en veiligheid van het systeem verbeterd.
2. **Data-encryptie:**
   * De data die wordt verzonden over het netwerk wordt eerst encrypt waardoor deze data beschermt is tegen mensen van buiten af.

Over het algemeen worden deze designpatronen gebruikt om de code te structureren, de leesbaarheid te verbeteren, herbruikbaarheid te bevorderen en wijzigingen in de toekomst gemakkelijker te maken. Ze dragen bij aan de flexibiliteit en onderhoudbaarheid van het programma.

# Netwerkcommunicatie

In de implementatie van de pizza-case wordt netwerkcommunicatie toegepast met behulp van het Strategy-ontwerppatroon. Hier volgen de stappen waarin netwerkcommunicatie plaatsvindt:

1. **Communicatiestrategieën**: In de applicatie zijn verschillende communicatiestrategieën gedefinieerd, waaronder 'TCP', 'UDP' en 'HTTP'. Elke strategie heeft specifieke implementaties voor het verzenden van berichten naar een externe server.
2. **Instantiatie van Communicatiestrategie**: Binnen de **init\_strategies**-methode van de **CommunicationStrategiesSingleton** wordt de gekozen communicatiemethode geïnstantieerd met de juiste configuratie, zoals de URL van de server of de host en poort, afhankelijk van de gekozen strategie.
3. **Verzenden van Bestelling**: Nadat een bestelling is gemaakt en samengesteld, wordt de **construct\_and\_send\_order**-functie aangeroepen. In deze functie wordt de bestelling omgezet in een tekstreeks en vervolgens verzonden naar de server via de gekozen communicatiestrategie.
4. **De gekozen strategie gebruiken**: De **construct\_and\_send\_order**-functie maakt gebruik van de geselecteerde communicatiestrategie door deze op te halen via de **CommunicationStrategiesSingleton** en vervolgens de **send\_message**-methode van de strategie aan te roepen. De daadwerkelijke communicatie met de server, zoals het verzenden van een bericht, vindt plaats binnen deze strategie.
5. **HTTP-communicatiestrategie**: Als voorbeeld van de implementatie, is de 'HTTP'-communicatiestrategie geconfigureerd met een URL (bijvoorbeeld **'http://localhost:8080'**). Wanneer de **send\_message**-methode van deze strategie wordt aangeroepen, wordt een HTTP POST-verzoek naar die URL verzonden met de bestelling als gegevens.

Dit ontwerp biedt flexibiliteit en modulariteit. Het stelt de applicatie in staat om eenvoudig van communicatiemethode te wisselen door de configuratie aan te passen, zoals het overschakelen van 'HTTP' naar 'TCP', zonder de rest van de code te wijzigen. Bovendien maakt dit ontwerp het mogelijk om in de toekomst nieuwe communicatiestrategieën toe te voegen, bijvoorbeeld ter ondersteuning van andere protocollen, zonder verstoring van de kernlogica van de applicatie.