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. **Composite**:
   * Het composite design pattern wordt gebruikt in de klasse composite om de pizza aan de bestelling toe te voegen. Daarnaast wordt het ook gebruikt om de pizzas uit de bestelling te verwijderen indien dit nodig is.
   * Waarom: Het composite design pattern kan objecten toevoegen in een boom structuur dit is erg handig een bestel app omdat een bestelling bestaat uit meerdere onderdelen. Daarnaast kan hiermee de bestelling makkelijk worden mee aangepast.
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.

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.