# Pizza Restaurant Design Documentation

## 1. Design Patterns

This document explains the design patterns used in the Pizza Restaurant application.

### 1.1 Factory Pattern

The Factory Pattern is used in the PizzaFactory class to create different types of pizzas such as Margherita and Pepperoni. This centralizes the creation logic, making it easier to manage and extend.

### 1.2 Decorator Pattern

The Decorator Pattern is used to dynamically add toppings to a pizza. The ToppingDecorator class serves as the base decorator, and specific toppings like Cheese, Olives, and Mushrooms extend it. This approach avoids creating multiple subclasses for every combination of pizza and toppings.

## 2. SOLID Principles

The Pizza Restaurant application adheres to the SOLID principles as follows:

### 2.1 Single Responsibility Principle (SRP)

Each class in the application is responsible for a single functionality:  
- The InventoryManager class handles inventory operations.  
- The Pizza class represents the base pizza interface.  
- The ToppingDecorator class manages toppings.

### 2.2 Open/Closed Principle (OCP)

The application is open for extension but closed for modification. For example, new pizza types or toppings can be added by creating new classes without modifying the existing code.

### 2.3 Liskov Substitution Principle (LSP)

Subtypes can replace their base types without breaking the application. For instance, both Margherita and Pepperoni classes can replace the Pizza interface.

### 2.4 Interface Segregation Principle (ISP)

The Pizza interface defines only the methods that are relevant to pizzas, avoiding unnecessary dependencies.

### 2.5 Dependency Inversion Principle (DIP)

High-level modules depend on abstractions, not concrete implementations. For example, the ToppingDecorator class depends on the Pizza abstraction.

## 3. Overengineering

Overengineering refers to adding unnecessary complexity to a system. This often leads to harder-to-maintain code and wasted development effort.

### 3.1 Example of Overengineering

If every possible combination of pizza and toppings were implemented as a separate class, it would lead to an exponential growth in the number of classes, making the system difficult to maintain. For example:  
- MargheritaWithCheese  
- MargheritaWithOlives  
- PepperoniWithMushrooms  
Instead, the application uses the Decorator Pattern to handle toppings dynamically.

### 3.2 Avoiding Overengineering

The current implementation strikes a balance by using the Factory Pattern and Decorator Pattern, avoiding unnecessary complexity while remaining flexible and extensible.