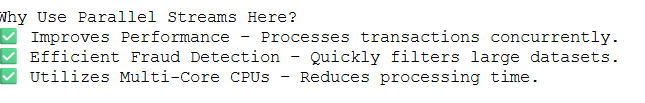
1. **Explain parallel streams with an example?**

Parallel streams in Java are useful for processing large datasets efficiently by leveraging multiple CPU cores. Here’s a real-time example where we process a list of transactions in parallel:

**Scenario**: Fraud Detection in Banking Transactions

Imagine a banking system that needs to analyse transactions for potential fraud. Using parallel streams, we can speed up the detection process.

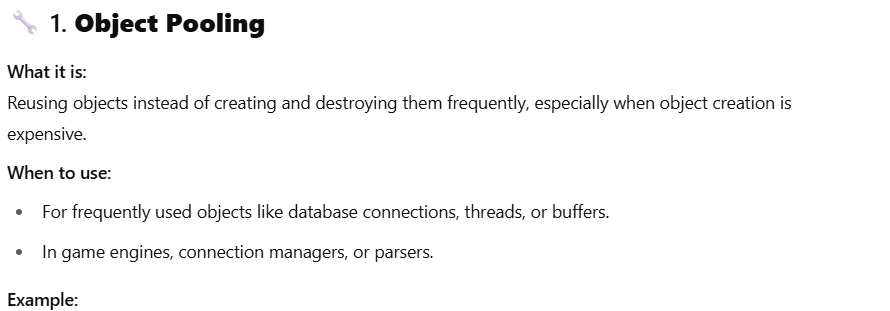
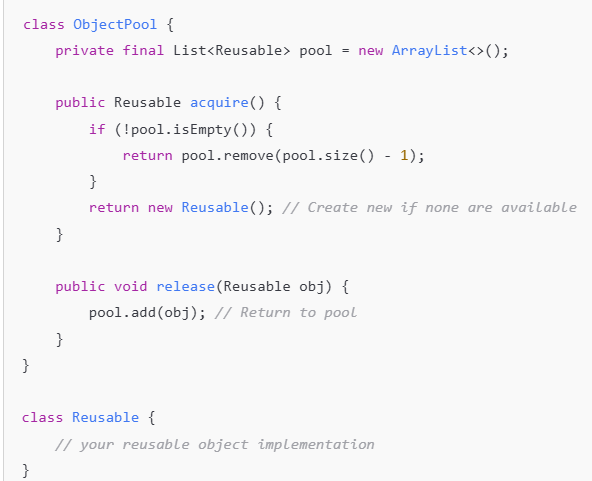


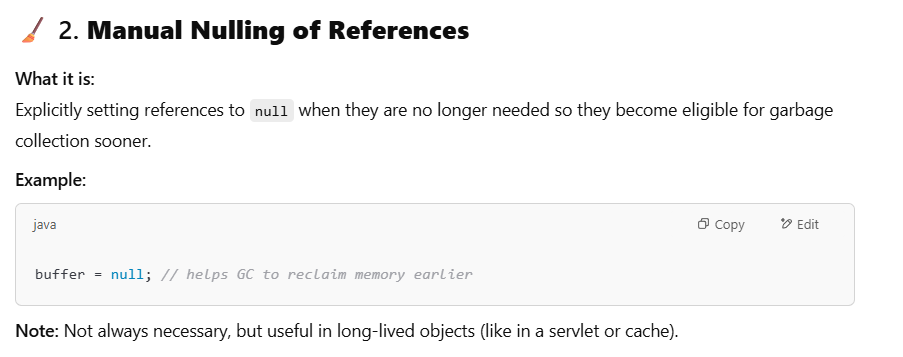


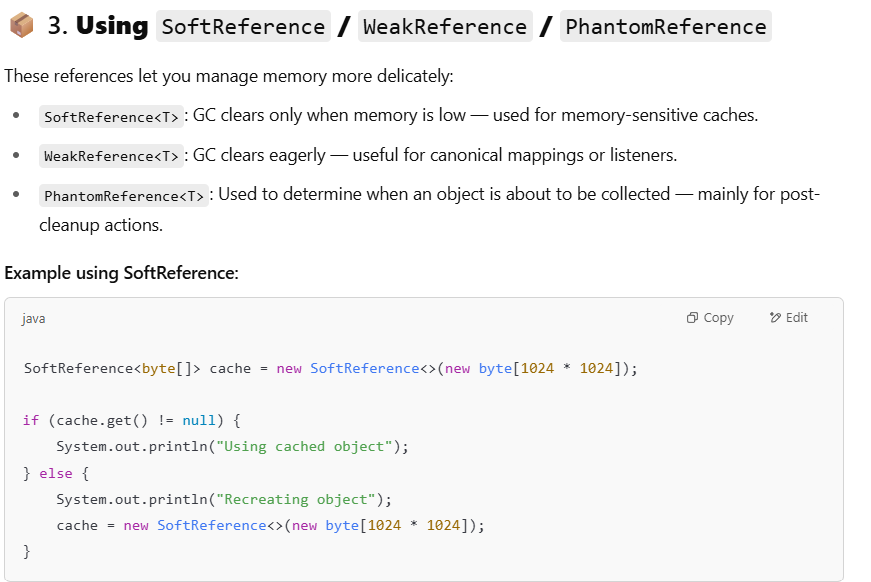
1. **How to do the custom memory management techniques in the Java?**

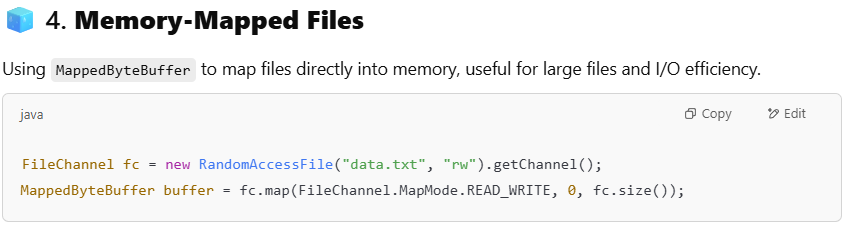
In Java, memory management is largely handled by the **JVM's garbage collector**, but sometimes custom memory management techniques are needed to optimize performance, especially in **high-performance** or **real-time systems**.

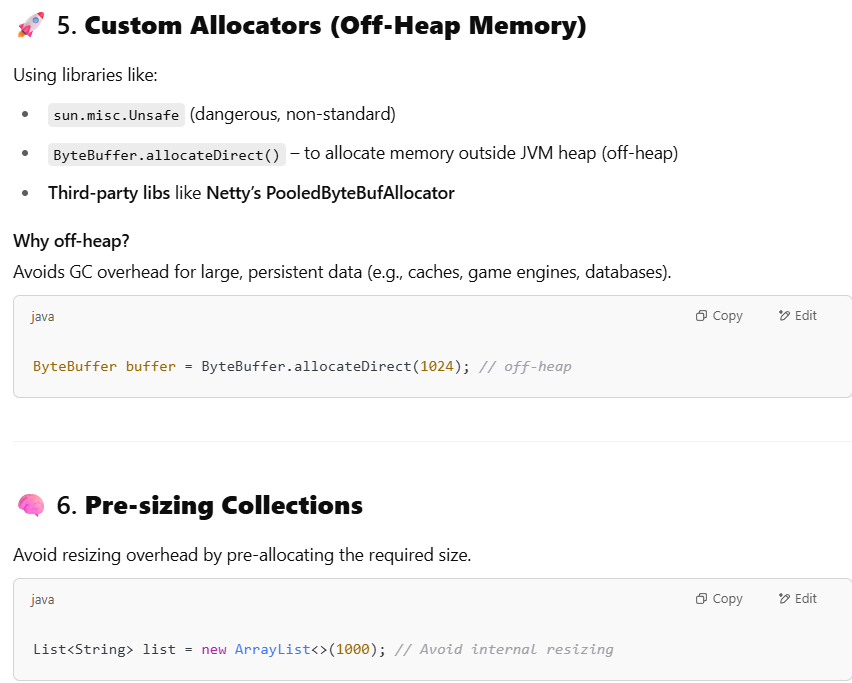
Here’s a breakdown of **custom memory management techniques** in Java:

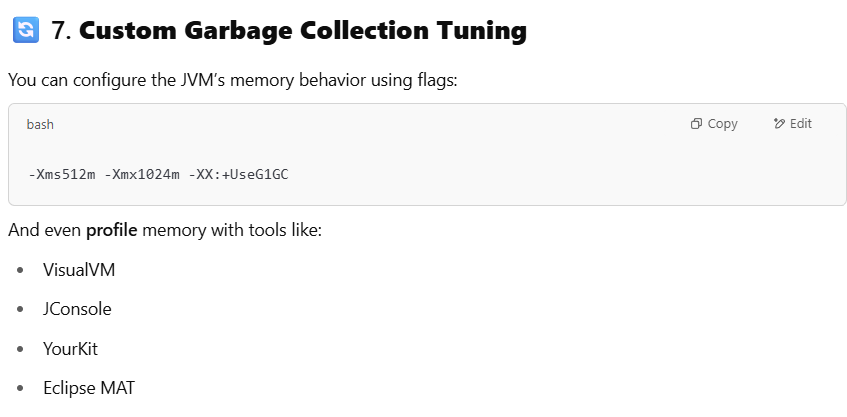
 











1. **How to restrict the access of immutable classes from reflection?**

Preventing reflection from modifying immutable classes in Java requires additional security measures beyond standard encapsulation. Here are some strategies:

**🔹 1. Use a Security Manager (Deprecated in Java 17+)**

- The SecurityManager can restrict reflective access to private fields.

- Example:

@Override

public void checkPackageAccess(String pkg) {

if (pkg.equals("java.lang.reflect")) {

throw new SecurityException("Reflection is not allowed!");

}

}

- Note: SecurityManager is deprecated in Java 17+, so alternative approaches are recommended.

**🔹 2. Use Sealed Classes (Java 17+)**

- Sealed classes restrict inheritance, preventing unauthorized modifications.

- Example:

public sealed class ImmutablePerson permits AllowedSubclass {

private final String name;

private final int age;

public ImmutablePerson(String name, int age) {

this.name = name;

this.age = age;

}

}

1. **Explain solid principles with examples?**

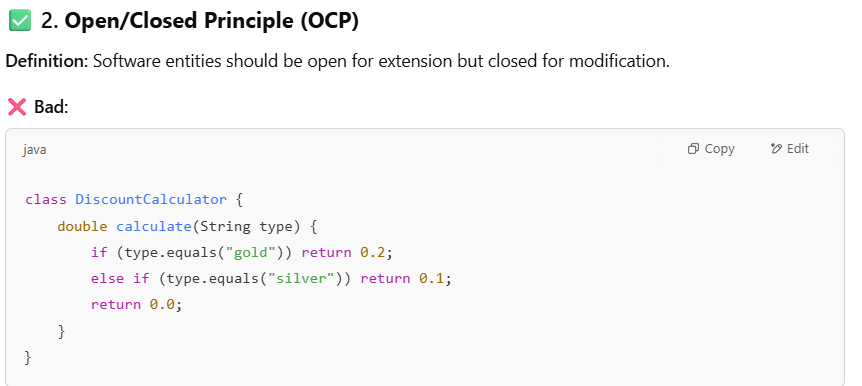
The SOLID principles are a set of five design principles in object-oriented programming that help developers create more maintainable, understandable, and flexible software. These principles were introduced by Robert C. Martin, also known as Uncle Bob. SOLID stands for:

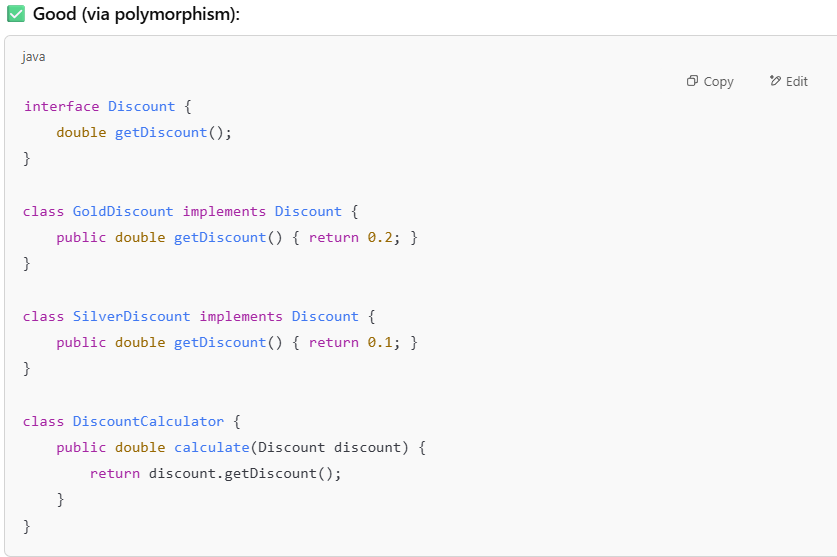
* **Single Responsibility Principle (SRP)**
* **Open/Closed Principle (OCP)**
* **Liskov Substitution Principle (LSP)**
* **Interface Segregation Principle (ISP)**
* **Dependency Inversion Principle (DIP)**

**Single Responsibility Principle (SRP)**

The Single Responsibility Principle states that a class should have only one reason to change, meaning it should have only one responsibility.





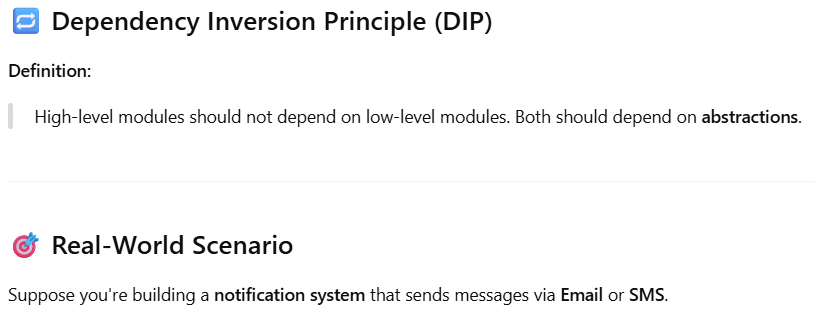


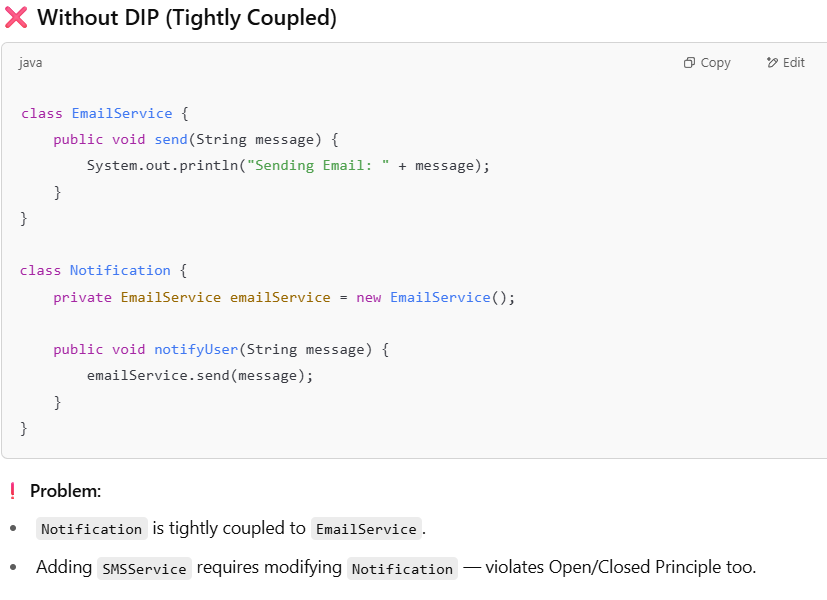


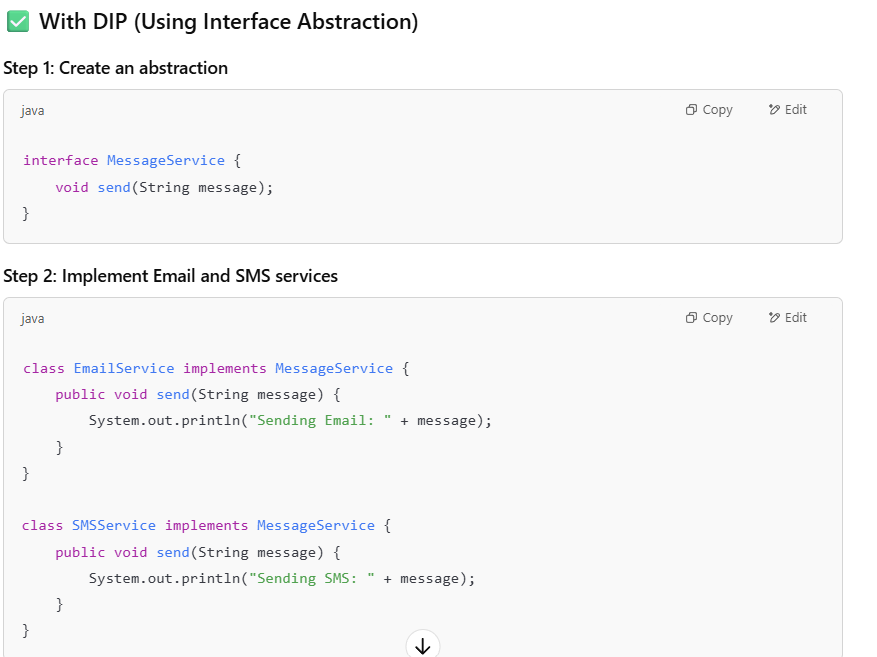


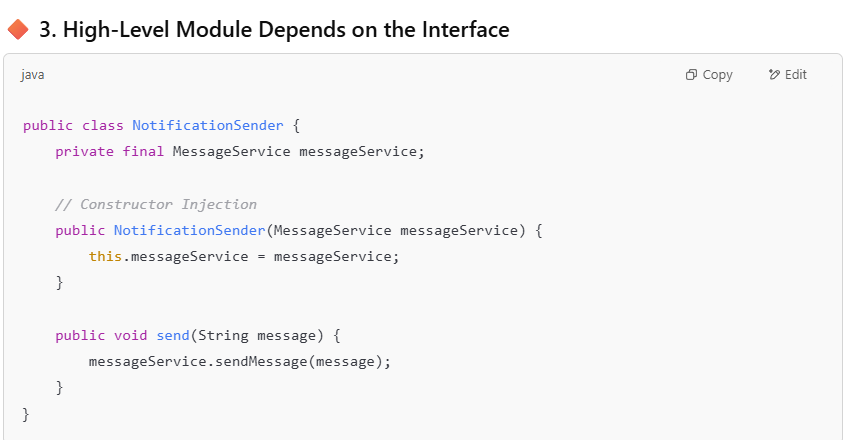




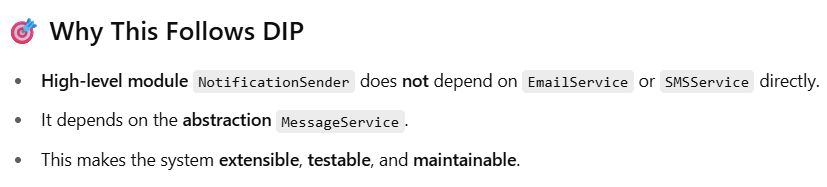






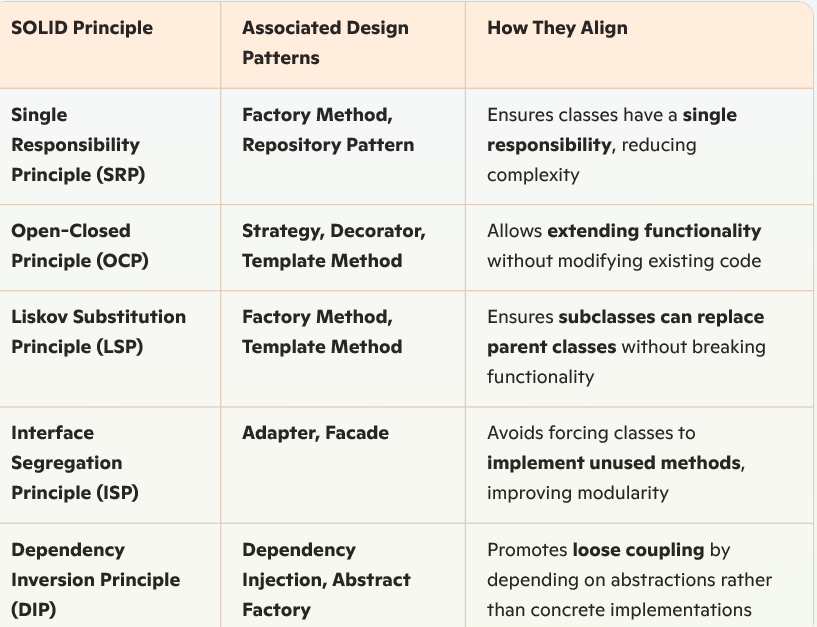






1. **Explain the contract between design patterns and solid principles?**

**The** SOLID principles **provide guidelines for writing** maintainable, scalable, and flexible **object-oriented software, while** design patterns **offer** reusable solutions **to common software design problems. The** contract between them **lies in how** design patterns embody SOLID principles **to create robust software architectures.**

****

1. **How to handle if consumer is unable to match the speed of producer in Kafka?**

**Increase Consumer Parallelism:** Scale horizontally by adding more consumer instances within the same consumer group. Kafka will automatically balance the partitions across them, helping to speed up consumption.

**Optimize Message Processing:** Use asynchronous processing with worker threads to parallelize handling. If possible, process messages in batches instead of one by one.

**Enable Backpressure Mechanisms:** Introduce flow control mechanisms in the producer to slow down production when consumers lag. Utilize rate limiting in producers or introduce throttling mechanisms.

**Use Compact Topic Retention & Consumer Lag Monitoring:** Implement a retention policy that ensures old messages are cleaned up. Monitor consumer lag using Kafka metrics and alerting systems to detect bottlenecks early.

**Improve Consumer Hardware & Configuration:** Increase memory and CPU allocation to ensure consumers have enough power. Optimize disk I/O performance, especially if consumers persist data.

1. **In spring boot when post construct load data?**

