Berikut adalah **rangkuman lengkap mengenai prinsip SOLID** dalam OOP yang bisa kamu gunakan untuk menghadapi tes:

Apa itu SOLID?

SOLID adalah kumpulan lima prinsip desain dalam *Object-Oriented Programming* (OOP) yang membantu mengembangkan perangkat lunak dengan struktur yang rapi, fleksibel, dan mudah untuk diubah atau dikembangkan. Prinsip ini diperkenalkan oleh Robert C. Martin (*Uncle Bob*) untuk membantu developer menghasilkan kode yang lebih baik.

Penjelasan Setiap Prinsip SOLID

1. **Single Responsibility Principle (SRP)**
    - **Inti:** Setiap kelas hanya memiliki satu tanggung jawab atau alasan untuk berubah.
    - **Tujuan:** Mempermudah pemeliharaan kode karena setiap kelas memiliki fungsi spesifik.
    - **Contoh:** Jika ada kelas Invoice yang menangani pembuatan faktur sekaligus mencetaknya, pisahkan fungsi ini menjadi dua kelas, misalnya Invoice untuk logika pembuatan faktur dan InvoicePrinter untuk logika pencetakan.
2. **Open/Closed Principle (OCP)**
    - **Inti:** Kelas harus bisa diperluas (*open for extension*) tanpa harus diubah (*closed for modification*).
    - **Tujuan:** Memungkinkan penambahan fitur tanpa mengganggu kode yang sudah berjalan.
    - **Contoh:** Daripada mengubah kode di kelas Shape untuk menambahkan bentuk baru seperti Circle atau Square, gunakan pola desain seperti *inheritance* atau *strategy pattern* untuk menambah bentuk baru tanpa memodifikasi kelas asli.
3. **Liskov Substitution Principle (LSP)**
    - **Inti:** Objek dari kelas turunan harus dapat menggantikan objek dari kelas induknya tanpa mengubah perilaku program.
    - **Tujuan:** Memastikan bahwa subclass dapat digunakan di mana pun superclass-nya digunakan.
    - **Contoh:** Jika ada superclass Bird dengan metode fly(), maka subclass Penguin (yang tidak bisa terbang) melanggar prinsip ini. Solusi: Gunakan abstraksi yang lebih spesifik, misalnya FlyingBird dan NonFlyingBird.
4. **Interface Segregation Principle (ISP)**
    - **Inti:** Jangan membuat antarmuka (*interface*) besar yang mengharuskan klien menggunakan metode yang tidak mereka butuhkan. Sebaiknya, buat antarmuka kecil yang spesifik sesuai kebutuhan.

- **Tujuan:** Mengurangi ketergantungan pada metode yang tidak diperlukan, sehingga kode lebih modular.
- **Contoh:** Daripada memiliki satu antarmuka besar Animal dengan metode fly(), swim(), dan run(), pisahkan menjadi beberapa antarmuka seperti Flyable, Swimmable, dan Runnable.

5. **Dependency Inversion Principle (DIP)**
   - **Inti:** Modul tingkat tinggi (logika bisnis) tidak boleh bergantung pada modul tingkat rendah (detail implementasi). Keduanya harus bergantung pada abstraksi.
   - **Tujuan:** Memisahkan logika utama dari detail implementasi agar mudah diubah dan diuji.
   - **Contoh:** Jika sebuah kelas PaymentProcessor menggunakan kelas CreditCardPayment, gantilah dengan abstraksi seperti PaymentMethod. Ini memungkinkan kelas PaymentProcessor bekerja dengan metode pembayaran lain seperti PayPalPayment tanpa perubahan besar.

Mengapa SOLID Penting?

- **Mengurangi kerumitan kode:** Setiap komponen memiliki tugas spesifik, sehingga lebih mudah dipahami.
- **Mempermudah pengujian:** Modularitas kode membuat pengujian unit lebih efisien.
- **Mempercepat pengembangan:** Perubahan atau penambahan fitur tidak akan merusak kode yang ada.
- **Memastikan fleksibilitas:** Kode dapat diperluas tanpa mengganggu struktur utama.
- **Meningkatkan kolaborasi:** Dengan kode yang terstruktur, tim developer dapat bekerja lebih efisien.

Tips untuk Tes dengan Dosen

1. **Pahami definisi setiap prinsip dan fokus pada tujuan utamanya.**
2. **Gunakan contoh konkret untuk menjelaskan, karena dosen sering menilai pemahaman melalui aplikasi nyata.**
3. **Pelajari hubungan antar prinsip:** Misalnya, DIP mendukung OCP karena keduanya berbasis abstraksi.
4. **Siapkan analogi sederhana:** Misalnya, SRP bisa diibaratkan seperti pekerja spesialis dalam sebuah tim.

Semoga sukses di tesmu! 😊