NAMA : NURIANA SIPAHUTAR
NIM : 424125000 7
KELAS: PSIK 24A

" JAWABAN SOAL ESSAY"

1) ➢ Encapsulation (pengkapsulan)
- Prinsip: Encapsulation adalah membungkus data (atribut/properti) dan metode (Fungsi/perilaku) yang bekerja pada data tersebut menjadi satu unit yang disebut kelas. Tujuannya adalah untuk menyembunyikan detail implementasi internal objek dari dunia luar dan hanya menyediakan antarmuka Publik yang terdefinisi dengan baik untuk berinteraksi dengan objek tersebut.
- Contoh: ATM, kita tidak tahu bagaimana mesin bekerja di dalamnya. kita hanya tau menekan tombol, memasukkan kartu dan pin, dan uang keluar.

➢ Inheritance (pewarisan)
- Prinsip: Inheritance adalah mekanisme di mana sebuah kelas (kelas anak/subkelas) dapat mewarisi properti dan metode dari kelas lain (kelas induk/superkelas). Ini memungkinkan kita membuat hirearki kelas di mana kelas anak dapat mengkhususkan atau memperluas perilaku dari kelas induk tanpa harus menulis ulang kode yang sama.
- Contoh: keluarga biologis, seorang anak mewarisi karakteristik (Fisik, sifat) dari orang tuanya. Meskipun anak memiliki karakteristik uniknya sendiri, ia tetap memiliki dasar genetik yang sama dari orang tuanya. Misalnya, anak bisa berjalan (Fitur dari orang tua), tapi anak juga bisa berbain bola (Fitur spesifiknya)

➢ Polymorphism (polimorFisme)
- Prinsip : Polymorphism merupakan bahasa yunani yang berarti "bentuk banyak" memungkinkan objek dari kelas yang berbeda untuk merespon panggilan metode yang sama dengan cara yang berbeda berdasarkan tipe objeknya.
- Contoh : Tombol "play" diberbagai aplikasi, pada pemutar musik, pemutar video, dan aplikasi streaming memiliki Fungsi yang sama secara konseptual (memulai pemutaran media). Namun, implementasi dibaliknya berbeda untuk setiap jenis perangkat.

➢ Abstraction (Abstraksi)
- Prinsip : Abraction adalah proses menyembunyikan detail implementasi yang kompleks dan hanya menampilkan informasi penting atau antarmuka tingkat

tinggi kepada pengguna. Ini membantu dalam mengelola kompleksitas dengan fokus pada "apa" yang dilakukan suatu objek, bukan "bagaimana" ia melakukannya.
-Contoh: Mengemudi mobil, kita hanya perlu tau cara menggunakan stir, gas, dan rem. kita tidak perlu tau bagaimana mesin internal bekerja.

2) Kelebihan utama menggunakan Java versi terbaru (Java 21) dibandingkan versi-versi sebelumnya dalam konteks pengembangan berbasis OOP terletak pada peningkatan sintaksis, fitur bahasa baru, dan peningkatan pada API yang secara signifikan menyederhanakan dan mempercepat pengembangan aplikasi yang berorientasi pada objek.

• Record Patterns (Pola Record)
Fitur ini menyempurnakan pattern matching untuk instance dari record classes (diperkenalkan di Java 16). Record adalah cara ringkas untuk membuat kelas data yang immutable. Sebelum Java 21, untuk mengakses komponen (field) dari sebuah record dalam blok "instanceof" atau switch, kita perlu melakukan casting secara eksplisit. Dengan Record Patterns, kita dapat secara langsung mendekonstruksi instance record ke dalam variabel-variabel komponennya dalam pola itu sendiri.

• Pattern Matching For swich (Pola pencocokan untuk switch)
Sebelum Java 21, konstruksi switch terbatas pada tipe primitif integral, enum, dan String. Java 21 memperluas kemampuan switch untuk menerima patterns, termasuk type patterns dan record patterns. Ini memungkinkan kita untuk melakukan pencabangan berdasarkan tipe objek dan bahkan mendekonstruksi objek secara langsung dalam case label.

3) Perbedaan mendasar antara Class dan Object :

Fitur	Class (kelas)	Object (objek)
Definisi	cetak biru (blueprint), template, atau rancangan untuk membuat objek.	Instansiasi konkret (perwujudan nyata) dari sebuah kelas
Sifat	Abstrak (tidak berwujud fisik). Mendefinisikan struktur dan perilaku	konkret (berwujud fisik dalam memori komputer). Memiliki data dan dapat melakukan aksi
Jumlah	Suatu kelas dapat digunakan untuk membuat banyak objek.	Banyak objek dapat dibuat dari satu kelas. Setiap objek memiliki identitas dan status yang unik.
Memori	Tidak menempati ruang memori secara langsung (hanya definisinya).	Menempati ruang memori untuk menyimpan data (nilai atribut).

Waktu Ada	Ada selama waktu definisi kelas ada dalam kode program	Dibuat (diinstansiasi) saat program berjalan dan dihancurkan saat tidak lagi digunakan.
Analogi	Cetak biru rumah, resep kue, kategori mobil (misalnya, "Mobil Sedan").	Rumah yang dibangun berdasarkan cetak biru, kue yang dibuat berdasarkan resep, mobil sedan tertentu (misalnya, "Toyota Camry").

→ Contoh penggunaan class
Mahasiswa adalah class. Ia mendefinisikan struktur (atribut nama, nim, jurusan, angkatan) dan perilaku (tampilkanInfo(), ubahJurusan()) dari entitas "Mahasiswa" Class Mahasiswa sendiri tidak menyimpan data mahasiswa tertentu.

→ Contoh penggunaan object (dalam kode main)
mahasiswa1 = new Mahasiswa ("Nuriana Siregar", "4241250007", "Ilmu komputer", 2024); membuat sebuah object bernama mahasiswa1 dari class Mahasiswa. Objek ini memiliki nilai atribut yang spesifik atau menyimpan data mahasiswa untuk Nuriana Siregar

4) Dalam class BankAccount, enkapsulasi diterapkan melalui penggunaan access modifier private pada atribut balance.
- Penggunaan private, kata kunci private memastikan bahwa atribut balance hanya dapat diakses dan dimodifikasi dari dalam class BankAccount itu sendiri. Tidak ada kode di luar class BankAccount yang dapat langsung mengubah nilai balance.
- Menyediakan Metode Publik Terkontrol, untuk berinteraksi dengan balance, class menyediakan metode publik yang terkontrol:
  - getBalance(), metode ini memungkinkan pihak luar untuk melihat nilai balance tetapi hanya untuk membaca (read-only).
  - deposit(double amount), metode ini memungkinkan untuk penambahan dana ke balance, tetapi dengan validasi.
  - withdraw(double amount), metode ini memungkinkan pengurangan dana dari balance, tetapi dengan validasi.
- Metode internal private, metode logTransaction adalah contoh metode internal yang bersifat private. Metode ini hanya dapat dipanggil dari dalam class BankAccount dan tidak dapat diakses langsung dari luar. Ini membantu menyembunyikan detail implementasi internal dan menjaga konsistensi data.

→ Mengapa enkapsulasi penting untuk keamanan sistem?

1. Mencegah akses dan modifikasi data yang tidak sah, dan tidak terkontrol ke data sensitif (seperti saldo)
2. Mengontrol bagaimana data dimodifikasi, menjamin validasi sebelum data diubah, sehingga tidak terjadi data corrupt atau kondisi tidak valid (misal saldo negatif).
3. Menyembunyikan detail implementasi, sehingga internal class bisa diubah tanpa memengaruhi kode luar.
4. Memudahkan debugging dan pemeliharaan, karena perubahan data hanya lewat fungsi terkontrol.
5. Meningkatkan keandalan sistem, mengurangi resiko bug dan celah keamanan yang bisa dimanfaatkan oleh pihak tidak bertanggung jawab

5) → Mekanisme Constructor Chaining Pada Pewarisan di Java:

Dalam pewarisan (inheritance) di Java, ketika sebuah objek dari subclass dibuat, konstruktor dari superclass juga akan dieksekusi sebagai bagian dari proses inisialisasi objek. Mekanisme ini disebut constructor chaining. Secara default, Java secara otomatis memanggil konstruktor tanpa argumen (no-arg constructor) dari superclass sebagai pernyataan pertama dalam konstruktor subclass. Proses constructor chaining terjadi secara berantai ke atas hirearki pewarisan. Artinya, jika suatu subclass memiliki superclass lain, konstruktor tanpa argumen dari superclass tersebut akan dipanggil terlebih dahulu, dan seterusnya hingga mencapai class Object (yang merupakan superclass dari semua class di Java secara implisit).

→ Apa yang terjadi jika constructor superclass tidak dipanggil secara eksplisit?

- Jika constructor superclass punya constructor default (tanpa parameter), maka Java akan memanggilnya otomatis.
- Namun, jika superclass hanya punya constructor dengan parameter dan tidak ada constructor default, maka subclass harus memanggil super(...) secara eksplisit, kalau tidak program akan error kompilasi.

→ Contoh jika konstruktor superclass tidak dipanggil Eksplisit (dan superclass tidak memiliki No-Arg constructor) dengan ilustrasi class karyawan dan subclass Manager.

- Jika kita modifikasi class Manager tanpa memanggil super() secara eksplisit, maka compiler Java akan memberikan kesalahan kompilasi karena superclass karyawan hanya memiliki konstruktor dengan argumen. Java secara otomatis akan mencoba memvisipkan super() (pemanggilan konstruktor tanpa argumen dari

superclass) sebagai baris pertama dalam konstruktor Manager, tetapi konstruktor tanpa argumen tidak ada di karyawan.
- Dalam kasus ini, jika kita tidak secara eksplisit memanggil super(nama,id); di konstruktor Manager, Java akan secara otomatis memanggil super(), yang akan menjalankan konstruktor tanpa argumen dari karyawan,
- Jika tidak ada pemanggilan super() eksplisit dan superclass memiliki konstruktor tanpa argumen, konstruktor tanpa argumen tersebut akan dipanggil secara otomatis. Namun, dalam skenario di mana superclass memerlukan argumen untuk inisialisasi yang benar, memanggil konstruktor superclass secara eksplisit dengan argumen yang sesuai adalah praktik yang sangat penting.

6) → Bagaimana Interface mendukung Polymorphism
- Dengan interface, kita bisa menulis kode berdasarkan kontak perilaku, bukan tipe konkret class-nya.
- Ini memungkinkan kita memproses berbagai objek yang berperilaku sama, tanpa peduli bagaimana cara kerjanya di dalam.
- Artinya, satu variable bertipe interface bisa menyimpan objek dari berbagai class yang mengimplementasikan interface itu.

• Dukungan Polymorphism melalui interface terjadi karena, Tipe referensi interface, Pemanggilan polimorfik, dan Decoupling (pelonggaran ketergantungan).

→ Contoh penggunaannya
Bayangkan kita sedang membangun sistem pemesanan makanan online. kita memiliki berbagai jenis metode pembayaran yang dapat digunakan oleh pengguna, seperti kartu kredit, dompet digital, dan transfer bank. Bagaimana Interface mendukung Polymorphism dalam contoh ini:
- Tipe referensi interface, dalam class OrderProcessor, metode ProcessOrder menerima parameter bertipe PaymentGateway.
- Pemanggilan metode polimorfik, di dalam metode processOrder, ketika paymentMethod, processPayment(totalAmount) dipanggil, implementasi processPayment() yang sebenarnya dieksekusi akan bergantung pada apakah PaymentMethod adalah objek CreditCardPayment atau DigitalWalletPayment.
- Fleksibilitas dan kemudahan pemeliharaan, menambahkan metode

Pembatasan baru dan mengganti implementasi

7) -> Abstract Class
- bisa punya implementasi method lengkap maupun method abstract (tanpa implementasi)
- bisa memiliki state (variabel instance) dan konstruktor
- Subclass harus meng-override method abstract
- mendukung pewarisan tunggal (satu class hanya bisa extends satu abstract class).

• kapan digunakan?
- saat ada relasi "is-a" yang kuat
- Saat ingin memberikan default implementation yang bisa di pakai ulang oleh subclass.
- Jika ingin menyembunyikan detail implementasi kompleks yang terkait state bersama.

-> Interface
- semua method secara default abstract (kecuali default method sejak Java 8)
- tidak bisa menyimpan state (kecuali static final constants)
- mendukung multiple inheritance (class bisa implement banyak interface)
- Fokus pada kontak (behavior), bukan implementasi.

• kapan digunakan:
- Saat hanya ingin mendefinisikan kontak perilaku yang harus di ikuti class tanpa memaksakan pewarisan tunggal.
- Saat ingin memberikan kemampuan tambahan (misal Berenang, Terbang) ke berbagai class yang tidak terkait secara hirarki

-> Sealed class
- membatasi class mana saja yang boleh menjadi subclass-nya
- memberikan kontrol kemanan dan eksplusit atas hirarki pewarisan
- bisa memiliki abstrak method atau concrete method
- Introduced sejak Java 15, jadi Fitur relatif baru

• kapan digunakan:
- Saat ingin mengontrol dan membatasi extensibility (perluasan kelas)
- cocok untuk hirearki domain tertutup dimana hanya subclass tertentu yang boleh ada.

- membantu compiler memberikan pemeriksaan exhaustiveness (misalnya di switch statement)
- Contoh: Bentuk (geometri) yang hanya boleh punya subclass segitiga dan lingkaran.