Nama = MOHD. FAHMI ALGHANI
NIM = 4293260036
Kelas = PSIK-21A
Pemograman Berorientasi Objek

1.) 1. Encapsulation
Dengan membungkus data dan fungsi dalam satu kesatuan (kelas), kita bisa membatasi akses hanya pada bagian-bagian yang memang perlu diketahui. Ini mencegah perubahan tak sengaja dan membuat kami punya sistem dapat diubah, tanpa memengaruhi bagian lain.
Contoh analogi = Mesin mobil disembunyikan di balik kap. Pengguna cukup menggunakan pedal dan setir tanpa perlu tahu cara kerja internal mesin

2. Inheritance
Saat sistem butuh banyak entitas dengan kesamaan perilaku atau data, inheritance memungkinkan kita membuat turunan dari satu kelas dasar. Ini mempercepat pengembangan dan konsistensi fungsi dalam sistem.
Contoh analogi = Anak mewarisi sifat fisik dari orang tuanya, seperti warna mata atau cara bicara, tapi bisa memiliki keunikan sendiri

3 Polymorphism
Sistem bisa memperlakukan objek berbeda dengan cara yang sama melalui antarmuka umum, meskipun setiap objek punya cara kerjanya sendiri. Ini membuat kode lebih fleksibel dan mudah diperluas.
Contoh analogi = Printah "Berjalan" bisa diberikan ke manusia, robot, atau hewan. Mereka semua akan berjalan, tapi dengan cara masing-masing

4 Abstraction
Dengan menyembunyikan detail teknis dan hanya menampilkan fitur penting, abstraction membantu developer fokus pada struktur dan alur sistem, bukan pada rincian teknis dari komponen.
Contoh analogi = Saat menggunakan ATM, pengguna cukup memencet tombol dan mengambil uang. Ia tak perlu tahu proses teknik di balik transaksi berlangsung.

2) kelebihan Java 21 dalam konteks OOP adalah kemampuannya menyederhanakan struktur kode, meningkatkan keamanan data, dan mempercepat pengembangan aplikasi melalui fitur-fitur mesin yang lebih ringkas dan ekspresif. Contoh dua fitur modern Java 21:
1. Record Patterns
Fitur ini memungkinkan kita mengekstrak data dari objek record secara langsung dan lebih efisien dalam pencocokan atau pemrosesan logika

2. Sealed Classes
Fitur ini memungkinkan pembatasan kelas mana saja yang boleh mewarisi kelas tertentu

3.) Secara sederhana, class adalah cetak biru (template), sedangkan object adalah hasil nyata dari cetak biru tersebut. Class adalah struktur atau rancangan yang mendefinisikan Properti (atribut) dan Perilaku (method) dari suatu entitas. Ia belum memiliki bentuk konkret dalam memori sampai kita membuat Objek dari Class tersebut. Sementara itu, Object adalah instansi nyata dari class, yaitu representasi konkret yang memiliki data dan dapat melakukan aksi sesuai dengan class-nya.
Contoh dalam konteks Program manajemen data mahasiswa

```
Public class Mahasiswa {
  // atribut (Property)
  String nama;
  String nim;
  int umur;
  // constructor
  Public Mahasiswa (String nama, String nim, int umur) {
    this.nama = nama;
    this.nim = nim;
    this.umur = umur;
  // method (Perilaku)
  Public void tampilkanInfo() {
    System.out.println("Nama : " + nama);
    System.out.println("NIM : " + nim);
    System.out.println("Umur : " + umur);

Public class main {
  Public Static void main (String[] args) {
  // membuat objek mahasiswa dari class mahasiswa
  Mahasiswa mhs1 = new Mahasiswa ("Andi", "123456", 20);
  Mahasiswa mhs2 = new Mahasiswa ("Budi", "654321", 22);
  // memanggil method
  mhs1.tampilkanInfo();
  mhs2.tampilkanInfo();
  }
}
```

4.) Penerapan Encapsulation dilakukan dengan cara menjadikan atribut balance bersifat Private, lalu menyediakan metode akses yang bersifat terkontrol, seperti getBalance() untuk melihat saldo dan deposit() atau withdraw() untuk memodifikasi saldo. Dengan begitu, data balance tidak bisa diakses atau diubah langsung dari luar class. Misalnya Programmer tidak bisa menulis akun.balance = 1000000 secara

langsung. Encapsulation penting untuk keamanan sistem karena mencegah data sensitif seperti saldo diubah secara sembarangan atau bisa salah. Ini menjaga integritas data dan memastikan bahwa setiap perubahan pada data terjadi melalui jalur yang aman dan terverifikasi.

5.) Constructor Chaining dalam pewarisan di Java adalah mekanisme di mana konstruktor subclass secara otomatis atau eksplisit memanggil konstruktor dari superclass terlebih dahulu sebelum menjalankan isi konstruktor subclass itu sendiri. Jika konstruktor superclass tidak dipanggil secara eksplisit menggunakan super(...), maka Java akan secara otomatis menyisipkan pemanggilan konstruktor tanpa parameter (default constructor) dari superclass di baris pertama konstruktor subclass.
Ilustrasi class karyawan dan subclass manager :

```
class karyawan {
 String nama;

 karyawan (String nama){
   this.nama = nama;
   System.out.println("konstruktor karyawan dijalankan");
 }
}

class manager extend karyawan {
 String divisi
 Manager(String nama, String divisi){
  super(nama); // constructor chaining secara eksplisit
  this.divisi = divisi;
  System.out.println("konstruktor manager dijalankan");
 }
}
```

6.) Dalam Java, Interface sangat mendukung konsep polymorphism karena memungkinkan berbagai class berbeda untuk mengimplementasikan metode yang sama dengan cara masing-masing. Dengan begitu, kita bisa menulis kode yang fleksibel, dapat diperluas tanpa mengubah struktur lama, dan mudah dipelihara. Dalam konteks sistem pemesanan makanan online, misalkan ada berbagai jenis layanan pengiriman seperti GoFood dan GrabFood. Kita bisa mendefinisikan sebuah interface bernama LayananPengiriman yang memiliki method prosesPesanan(). Masing-masing layanan kemudian mengimplementasikan interface tersebut sesuai dengan mekanisme internal mereka. Misalnya, GoFood bisa mencetak "Pesanan diproses oleh GoFood", sementara

OKEY

Date

GrabFood bisa merepresentasikan "Pesanan dikirim oleh GrabFood driver", dan seterusnya.

7. A. Abstract Class :
- Bisa punya state
- bisa implementasi method
- hanya bisa 1 superclass
- tidak membatasi subclass
- cocok untuk template kelas dengan kode

kasus yang tepat digunakan : ketika subclass harus memiliki state/properti sama

B. Interface
- tidak bisa punya state (kecuali konstanta)
- bisa implementasi method default / static
- bisa memiliki banyak interface
- tidak membatasi subclass
- cocok untuk membuat kontrak/perilaku

kasus yang tepat digunakan = ketika ingin membentuk sistem kontrak atau kemampuan tertentu

C. Sealed Class
- Bisa punya state
- Bisa implementasi method
- hanya bisa 1 superclass
- subclass dibatasi
- cocok untuk kontrol hierarki kelas terbatas

kasus yang tepat digunakan = ketika ingin mengontrol keamanan tipe dan menghindari subclass yang tidak diinginkan

OKEY