

UTS

Nama : Gus Rosario Pandiangan

Nim : 4293250055

Kelas : PSIK - 29 A

Mata kuliahan : Pemrograman Berorientasi Objek

Dosen Pengampu : Insan Taufik, M. Kom.

Soal Essay

1 Jelaskan bagaimana prinsip encapsulation, Inheritance, Polymorphism, dan abstraction saling mendukung dalam membangun sistem perangkat lunak yang mudah dikembangkan dan dipelihara. Sertakan contoh analogi dalam kehidupan nyata untuk masing-masing konsep.

Jawab:

•> Encapsulation (Enkapsulasi)

↳ Encapsulation adalah prinsip menggabungkan atau membungkus data dan metode yang beroperasi pada data tersebut menjadi satu unit, yaitu objek. Tujuannya adalah Mengembungkan detail informasi internal dan Melindungi data dari akses atau modifikasi yang tidak sah dari luar. Data hanya dapat diakses atau diubah melalui metode-metode publik yang disediakan oleh objek itu sendiri.

Contoh analogi:

Bayangkan sebuah Mobil. kita bisa mengendarainya (Mengaktifkan fungsi) dengan Menggerakkan setir, gas atau rem. kita tidak perlu tahu bagaimana detail bagaimana bekerja, bagaimana transmisi menghubungkan gigi, atau bagaimana sistem keluarnya terhubung. Semua detail internal ini dienkapsulasi di dalam mobil. Kita hanya berinteraksi melalui antarmuka (kontrol) yang disediakan. ini melindungi komponen internal dan memudahkan kita sebagai pengguna.

•> Inheritance (Pewarisan)

↳ Inheritance adalah mekanisme dimana kelas baru (disebut kelas anak, ataupun subclass) dapat mewarisi atribut dan metode dari kelas yang sudah ada (di sebut kelas induk atau superclass). ini mempromosikan penggunaan kembali kode (code reusability) dan memungkinkan pembangunan kelas-kelas yang mempergunakan hubungan. kelas anak dapat memanfaatkan fungsi-fungsi kelas induk dan menambahkan fungsi-fungsi baru atau mengubah perniku yang diwarisi.

Contoh analogi:

Konsep spesies hewan. kita bisa membuat kelas induk Hewan dengan atribut umum seperti: Nama, berat dan metode bernafas() makan(). kita juga bisa membuat kelas anak seperti: kucing, anjing, atau burung. kelas kucing akan mewarisi semua atribut atau metode dari Hewan. tapi



Juga dapat memiliki atribut spesifik seperti Jenis Bulu dan merde Mengeong(). ini menghindari pengulangan depan atau dasar yang sama untuk semua jenis hewan.

> Polymorphism

↳ Polymorphism yang artinya "banyak bentuk" adalah kemampuan objek suatu kelas yang berbeda untuk merespons panggilan dengan metode yang sama dengan cara yang berbeda. ini dicapai melalui metoda overriding (kelas anak menafsirkan ulang metode yang sudah ada di kelas induk) atau metoda overloading (beberapa metode dengan nama yang sama tetapi nilai parameter berbeda).

Contoh Analogi: ~~istilah teknis~~ Menggambar garis, hasilnya adalah gambar penit. jika anda memilih alat "pensil" dan menggambar garis yang sama, hasilnya adalah garis lurus yang lebih tebal dan bertekstur. jika anda memilih alat "pen" dan Menggambar, hasilnya adalah garis Vektor yang tajam. Satu perintah (gambar()). namun implementasinya berbeda tergantung pada Objek (alat) yang digunakan.

> Abstraction (Abstraksi)

↳ Abstraction adalah proses menyembunyikan detail informasi yang tidak relevan dan hanya menampilkan fungsionalitas esensial para pengguna atau programmet tan tujuannya adalah untuk menyederhanakan komplikasi sistem, memungkinkan tetapi pada "Apa" yang dilakukan objek, bukan "Bagaimana" objek Melakukannya. ini sering diimplementasikan melalui Interface atau abstract class

Contoh Analogi:

Pilot kontrol di pesawat terbang. seorang pilot berinteraksi dengan ribuan tombol dan tuas untuk Mengendalikan pesawat. Namun setiap kontrol (Misalnya: tuas throttle untuk mengatur kekuatan) hanya menunggu fungsionalitas esensial. pilot tidak perlu tahu detail rumit bagaimana mesin pesawat bekerja secara internal setiap kali ia Menggerakkan tuas detail - detail tersebut di abstrahi dari pandangan pilot. Memungkinkan pilot fokus pada tugas utamanya Mengendalikan pesawat.

"Keterkaitan dan Saling Mengukung"

Keempat Prinsip ISO ini saling Melengkapi untuk membangun sistem Perangkat Lunak yang efisien

- ↳ Encapsulation memastikan bagian objek mandiri dan datanya terlindungi
- ↳ Membentuk unit modular yang terorganisir
- ↳ Inheritance Menyulihfaatkan modularitas ini memungkinkan fungsiionalitas



yang ada untuk digunakan, kembali dan dipertahui, mempercepat pengembangan dan mengurangi duplikasi kode.

- ⇒ Polymorphism Memungkinkan Fleksibilitas. Memungkinkan objek-objek dalam hierarki inheritance (yang sudah di-encapsulate) untuk di-olah secara seragam melalui antarmuka umum. Meskipun penulisan spesifiknya berbeda.
 - ⇒ Abstraction Melengkapi dengan menyembunyikan kompleksitas sehingga pengembang dapat fokus pada fungsionalitas inti tanpa terganggu oleh detail implementasi yang rumit.
- Q. Apa kelebihan menggunakan Java terbaru (Java 21) dibanding versi-versi sebelumnya dalam konteks pengembangan berbasis OOP? berikan minimal 3 fitur modern Java di bawah ini yang bagaimana fitur tersebut menyederhanakan pengembangan aplikasi OOP!

Jawab: Menggunakan Java versi terbaru seperti Java 21 menawarkan berbagai keunggulan signifikan dibandingkan versi sebelumnya terutama dalam OOP karena peningkatan efisiensi, keamanan, performa, terutama penyederhanaan sintaksis serta pola pengembangan yang lebih modern.

Kebutuhan Umum:

1. Stabilitas dan dukungan fungka panjang(LTS): Sebagian versi LTS, Java 21 menjamin pembaruan dan dukungan keamanan selama bertahun-tahun yang sangat krusial untuk aplikasi skala produksi yang membutuhkan stabilitas tinggi.
2. Peningkatan performa: fitur versi Java baru umumnya membantu optimasi pada JVM (Java Virtual Machine) dan garbage collection. Mengoptimalkan performa aplikasi yang lebih cepat dan efisien.
3. Fitur Bahasa yang lebih Modern: Java 21 tenus mengembangkan sintaksis agar lebih tingkat, elegan, dan aman, mengurangi boilerplate code yang sulit ditemukan di versi lama.
4. Keamanan yang ditingkatkan: pembaruan rutin pada Java selalu menyertakan perbaikan keamanan, melindungi aplikasi dari keamanan terbaru.

Fitur Modern Java 21 adalah menyederhanakan pengembangan aplikasi OOP:

1. Virtual Threads (§ 17.4.9)

Virtual Threads adalah thread ringan yang diciptakan JVM. Metode memungkinkan aplikasi Java menangani lebih banyak tugas konkuren (sekuar bersamaan) dengan efisien sumber daya yang sangat lebih baik dibanding platform thread tradisional.

2. Penyederhanaan OOP:

- Kode konkuren lebih simple: Pengembang dapat menulis kode konkuren seolah-olah itu statis (langkah demi langkah.) tanpa perlu



Pola pemrograman abstrak yang rumit ini membuat logika klien berbasis objek jadi lebih mudah di tulis dan dipahami, karena tidak perlu Mengubah cara berpikir OOP yang sudah ada.

- ⇒ Peningkatan Skalabilitas : Memungkinkan aplikasi OOP menangani tiba-tiba banyak jumlah permintaan secara efisien. Meningkatkan kemampuan aplikasi untuk melayani banyak pengguna atau proses secara bersamaan.

2. Record Patterns (SEP 940) dan pattern Matching switch (SEP 941)

↳ Fitur ini memungkinkan ekstrak data dari objek record (tipe data ringkas untuk objek data) secara langsung dalam ekspresi pattern matching.

Bersamaan dengan itu, pernyataan switch kita bisa mencocokkan pola objek (bukan hanya nilai) untuk mengetahui type dan mengambil datanya sekaligus.

⇒ Penyederhanaan OOP

Kode lebih ringkas, Mengurangi boiler plate code (kode berulang) yang sebenarnya diperlukan untuk memerlukan tipe objek (instanceof) dan Melakukan Casting sebelum mengakses data. ini membuat kode untuk memproses objek data (yang sering di temukan dalam OOP) jadi jauh lebih bersih dan mudah dibaca.

- ⇒ Peningkatan Keamanan Kode : Membantu Mengurangi bug dan potensi NullPointerException karena kompiler dapat memastikan semua kemungkinan kasus penanganan objek sudah terjangkau dan fitur ini bisa meningkatkan keandalan aplikasi OOP anda.

3. Mahasiswa Seringkali salah Memahami Perbedaan antara class dan object. jelaskan secara detail perbedaan keduanya dan berikan contoh penggunaan class dan object dalam konteks program Manajemen data mahasiswa.

Jawab :

Perbedaan Class dan Objek dalam PBO

1. Class

⇒ Definisi Konseptual Class

- Class dapat diibaratkan sebagai cetak biru (blueprint), rangungan, atau prototipe abstrak untuk menciptakan objek. ia adalah suatu template yang mendefinisikan struktur (atribut / data) dan perilaku (metode / fungsi) yang akan dimiliki oleh semua objek yang dibuat dari class tersebut.
- Class adalah konsep logis semata : ia hanya ada sebagai definisi di dalam kode sumber program. secara inheren, class itu sendiri tidak menempati Memori fisik (RAM) saat program dijalankan (runtime). Untuk menyimpan data spesifik, melainkan hanya definisi strukturnya.
- Fungsinya untuk Mengkategorikan atau mendefinisikan jenis identitas.



⇒ Sifat dan Karakteristik Utama,

- Abstrak dan Statis : Class tidak bisa "Melakukan" apapun secara langsung. ia hanya mendefinisikan potensi atau kemampuan dari objek yang akan diexpresikan.
- Tidak Memiliki State (Status) konkret : Class itu sendiri tidak memiliki nilai data tertentu. Contohnya : Class Mahasiswa tidak memiliki nama "Budi" atau Nomor Induk mahasiswa : secara spesifik ia menyatakan bahwa semua objek Mahasiswa akan memiliki atribut Nama dan Nim.
- Definisi struktur data dan perilaku : iiii adalah tempat di mana Variabel - Variabel (atributes atau fields) yang akan menyimpan data dan fungsi-fungsi (Method) yang akan melakukan operasi pada data tersebut, di definisikan.
- Didefinisikan sekali : Dalam sebuah program, sebuah class biasanya hanya didefinisikan satu kali sebagai representasi dari suatu tipe dan kelas.
- Tujuan utama : Sebagai "pabrik" atau "Cetakar" untuk membuat banyak object yang konkret.

(Contoh analogi kehidupan nyata)

Resep kue : Resep adalah kumpulan instruksi (Metode) dan sifat-sifat (atribut) untuk membuat kue. Resep itu sendiri tidak bisa dimakan.

2. Object

⇒ Definisi konseptual Object

- Object adalah instansiasi (wujud nyata) dari sebuah class. ia identitas konkret, manusia, dan unit yang dibuat, berdasarkan definisi class.
- Setiap Object Memiliki Identitas, state (nilai atribut spesifik) yang berbeda dari object lain dari class yang sama, dan perilaku yang ditentukan oleh metode - metode di class-nya.
- Object adalah konsep fiktif, ia menempati ruang di memori (RAM) saat program di jalankan (runtime). menyimpan nilai-nilai khususnya yang spesifik.
- ⇒ Sifat Karakteristik Utama Object.
- konkret dan Dinamis : Object adalah entitas nyata yang dapat berinteraksi, memproses data, dan melakukan aksi. statusnya dapat berubah selama dijalankan program.
- Memiliki State (Status) Spesifik : setiap Object Memiliki nilai-nilai spesifik untuk atribut-atribut yang diidentifikasi dalam class-nya. Misalnya, Object Mahasiswa bernama mhs1 akan memiliki nilai Nama: "Gus Rosauli", Nim : "9213150055" dan Prodi = "Ilmu Komputer".
- Dapat Melakukan aksi : Object dapat Mengjalankan metode-metode yang di definisikan dalam class-nya. Dan aksi tersebut akan mempengaruhi state dan object itu sendiri.
- Bisa dibuat Banyak : Dari satu class, kita dapat membuat atau membuat



banyak objek yang berbeda. Setiap Objek akan diidentifikasi satu sama lain dalam hal state dan identitasnya. meskipun mereka berbagai definiti penamaan

- Interaksi : Objek adalah unit dasar yang berinteraksi dalam suatu penggunaan PBO. Mengirim dan menerima pesan melalui pemanggilan metode.

Analogi kehidupan nyata:

- Kue yang sudah jadi. kue yang sudah dipanggang dari resep. kue itu bisa dimakan, rasakan, dan sentuh. sedangkan kue bisa punya tepung.
- Atau roda yang sekitar berbeda meskipun dibuat dari resip yang sama

Tabel Perbandingan Dari Class

Aspek Penting	Class (Kelas)	Objek (objek)
Sifat Konseptual	Abstrak, Logis	Konkret, Fisik
Keteradaan	Ada difase komplikasi dan definisi kode	ada difase runtime (saat program dijalankan)
Alokasi Memori	tidak mengalokasikan memori secara spesifik	Mengalokasikan memori untuk menyimpan nilai aktivitas
Peran	Cetak baru / template / paralel	listensi, bukti nyata / produk dari cetak baru
Fisi	Mendefinisikan struktur (atribut) & penelaku (Metode)	Memiliki nilai spesifik Untuk setiap aktivitas
State (Status)	tidak memiliki state (hanya definisi)	Memiliki state (misalnya yang spesifik).
Jumlah	Hanya satu Definisi class	Dapat membuat banyak object di satu class yg sama
Kata kunci Java	Class	New (untuk instansiasi)

Contoh Penggunaan Class dan Objek dalam program Manajemen

Data Mahasiswa:

→ Class → Mendefinisikan | Class Mahasiswa | → Mahasiswa.java

```
Java
// Definisi Class Mahasiswa
public class Mahasiswa {
    // Atribut (Data) - Mewakili karakteristik setiap mahasiswa
    String Nama;
    String Nim;
```

String Jurusan;
Double IPK ;

// Konstruktor (Metode khusus untuk membuat object)

Digunakan untuk menginisiasi atribut saat object baru di buat
Public Mahasiswa (String nama, String nim, String jurusan, Double IPK){
this.nama = nama; // 'this.nama' menyatakan atribut object 'nama' pada parameter
this.nim = nim;
this.jurusan = jurusan;
this.ipk = ipk;

// Metode (perilaku) - mempresentasikan data yg bisa dituliskan mahasiswa

Public void tampilkanInfo(){

System.out.println("Nama : " + this.nama);

System.out.println("NIM : " + this.nim);

System.out.println("Jurusan : " + this.jurusan);

System.out.println("IPK : " + this.ipk);

Public void updateIPK(Double newIPK){

this.ipk = newIPK;

System.out.print(this.nama + "(NIM: " + this.nim + ")") IPK dibeharui
diperbarui dari newIPK

→ Program Object → ManajemenMahasiswa.java

Java

// Program utama untuk membuat dan menggunakan object Mahasiswa

Public class ManajemenMahasiswa {

Public static void main (String [] args) {

// Membuat object Mahasiswa

// Nama diubah menjadi "Gus Rosauji"

Mahasiswa mhs1 = new Mahasiswa("Gus Rosauji", "4293250055", "Ilmu Komputer", 3.85);

Mahasiswa mhs2 = new Mahasiswa("Fitri Amilia", "20230021", "Teknik Elektro", 3.50);

Mahasiswa mhs3 = new Mahasiswa("Joko Sujono", "2023003", "Pendidikan Grafisi", 3.70);

System.out.println("\n-- Informasi Mahasiswa --");

mhs1.tampilkanInfo();



System.out.println ("In --> Informasi Mahasiswa");
 mhs1.tampilkanInfo();

// Memperbarui dan menampilkan informasi Mahasiswa

System.out.println ("In --> Perkuliahan IPK Mahasiswa ---");

mhs1.UpdateIPK(3,39);

System.out.println ("In --> Informasi Mahasiswa telah update ---");

mhs1.tampilkanInfo();

4. Anda diminta membuat class BankAccount. Jelaskan bagaimana anda akan menerapkan encapsulation agar data balance tidak bisa diubah sembarangan.

Mengapa encapsulation penting untuk keamanan sistem?

Jawab:

Penerapan Encapsulation pada class BankAccount

C. Menerapkan encapsulation pada class Bank Account bertujuan untuk melindungi data sensitif seperti balance agar tidak diubah sembarangan, serta mengontrol bagaimana data tersebut diakses.

Berikut cara penerapan yang saya ralatkan:

⇒ Implementasi Encapsulation

1. Atribut Private: Deklarasi atribut balance dengan access Modifier private. ini memastikan balance hanya bisa diakses dan diubah dalam class Bank Account. itu sendiri, mencegah modifikasi langsung dari luar.

Java

```
public class BankAccount {
    private double balance; // Hanya diakses dalam class ini
    ...
}
```

2. Metode public (Getter & Setter Terkontrol): Sedekian metode public untuk berinteraksi dengan balance secara terkontrol.

⇒ getBalance() (Getter): Untuk membaca nilai balance

Java

```
public double getBalance() {
    return this.balance;
}
```

⇒ Deposit() & Withdraw() (Setter terkontrol): Untuk mengubah balance

Metode ini dapat menyertakan logika validasi (misalnya, cek jumlah

deposit posisi atau saldo cukup untuk withdraw) sebelum perubahan data terjadi. ini dapat mengjamin perubahan sesuai aturan bisnis.

Java

```
public void deposit (double amount) {
    if (amount > 0) { // Validasi
        this.balance += amount;
    }
}
```

```
public void withdraw (double amount) {
    if (amount > 0 && this.balance >= amount) { // Validasi
        this.balance -= amount;
    }
}
```

Encapsulation penting untuk keamanan sistem karena:

- ⇒ Proteksi data: Encapsulation menegakkan kewajiban langsung dan mandiratif terhadap data sensitif (balance). ini menjaga Integritas dan Validitas data, yang merupakan fondasi utama keamanan.
- ⇒ Kontrol akses: Data hanya bisa diubah melalui metode yang telah disahkan (Deposit, Withdraw) ini memastikan setiap penurunan balance mematuhi aturan bisnis dan sistem tetap aman dari operasi yang tidak sah.
- ⇒ Modularitas & Pemeliharaan: Membuat kode lebih modular dan mudah di periksa. Penyalinan internal pada data tidak akan memengaruhi bug lain dalam program sehingga menguntungkan nenko bug dan mempermudah debugging, yang secara keseluruhan meningkatkan stabilitas dan keamanan sistem.

5. Jelaskan Bagaimana Mekanisme Constructor Chaining bekerja pada bahasan Java apa yang terjadi jika Constructor pada Superclass tidak dipanggil secara eksplisit? Senakan ilustrasi Class Petyawan dan Subclass Manager Jawab:

Constructor chaining → Mekanisme PBO untuk memastikan initialisasi objek dari kelas induk (superclass) selalu terjadi sebelum initialisasi kelas anak (subclass).

Cara constructor chaining bekerja:

1. Panggilan implicit (super()):

⇒ Setiap konstruktur subclass secara otomatis akan memanggil konstruktur no-argument dari superclassnya melalui super ()

⇒ Panggilan super () ini akan ditambahkan secara otomatis oleh kompiler sebagai basis pertama jika ada panggilan eksplisit lain

2. Panggilan Eksplisit (super (arguments);):

⇒ jika superclass hanya punya konstruktur berparameter (tidak ada konstruktur



No-argument) maka subclass harus secara eksplisit memanggil konstruktur superclass tersebut menggunakan super (Argument).

⇒ Panggilan ini wajib menjadi basis pertama dalam konstruktur subclass.

Jika subclass tidak memanggil konstruktur secara eksplisit (misal: Super (nama, id karyawan);) dan superclass hanya memiliki konstruktur berparameter.

- Maka:
1. kompiler akan mencoba melanjutkan Super () ; tetapi Impunit
 2. karena tidak ada Super () di superclass, kompiler akan menghasilkan error kompilasi
 3. Hal ini menegaskan bahwa setiap objek subclass harus ikut memastikan bagian superclass nya terimplementasi dengan benar terlebih dahulu

Ilustrasi (Class Karyawan & Manager)

Java

// Karyawan.java (superclass)

public class Karyawan

String nama,

String idKaryawan

public Karyawan (String nama, String idKaryawan) { // hanya constructor

System.out.print ("Constructor Karyawan dipanggil .");

this.nama = nama;

this.idKaryawan = idKaryawan;

}

// Manager.java (subclass)

public class Manager extends Karyawan {

String Departemen,

public Manager (String nama, String idKaryawan, String Departemen) {

// Wajib panggil constructor superclass secara eksplisit karena karyawan tidak punya constructor no-argument

super (nama, idKaryawan),

System.out.println ("Constructor Manager dipanggil .");

this.Departemen = Departemen

}

// MainApp.java (pengujian)

public class MainApp {

public static void main (String [] args) {

```
Manager mgr1 = new Manager ("Budi Samoso", "kool", "pemasaran");
```

```
// output:
```

```
/constructor karyawan dipanggil
```

```
/constructor Manager dipanggil.
```

```
}
```

If Superclass tidak memiliki konstruktor no-argument akan menyebabkan error.

- 6 Polymorphism Memungkinkan kita menulis kode yang fleksibel dan mudah di-maintain. Jelaskan bagaimana penggunaan interface mendukung konsep ini, dan berikan contoh penggunaannya dalam sistem pemesanan makanan online Jawab.

Peran Interface dalam mendukung polymorfisme

• Interface Mendefinisikan kontrak perilaku dan blueprint metode tanpa implementasi kelas yang Mengimplementasikan interface wajib menyediakan implementasi konkret Metode tersebut :

i. Mendefinisikan tipe umum : Interface menciptakan tipe spesifikasi abstrak, berbagai objek dari kelas berbeda yang bisa dilakukan oleh tipe interface yang sama,

Menyederhanakan interaksi seragam tanpa detail implementasi konkret

d. Pemisahan perilaku dan implementasi : Ini akan memisahkan "Apa yang bisa dilakukan" (kontrak interface) dan "Bagaimana melakukannya" (implementasi kelas). Ini mendorong loose coupling

3. Meningkatkan Fleksibilitas dan Estensibilitas : karena kode berinteraksi dengan interface, perambahkan objek baru (yang mengimplementasikan interface sumbu) tidak memerlukan modifikasi kode yang sudah ada.

4. Memungkinkan penggantian (Injeksi dependensi / Substitusi) : Memungkinkan penggantian implementasi konkret tanpa mengubah kode pengguna interface, memfasilitasi dependency injection dan mengisolasi komponen secara terpisah.

Contoh : Pemesanan makanan online

Didefinisikan Interface PesananItem sebagai kontrak untuk setiap item yang bisa di pesan :

Java

```
// kontrak umum untuk item yang bisa di pesan.
```

```
public interface PesananItem {
```

```
double hitungharga();
```

```
// kelas-kelas konkret yang mengimplementasikan PesananItem
public class Pizza implements PesananItem /* ... implementasi hitung harga ... */
public class Burger implements PesananItem /* ... implementasi hitung harga ... */
public class Salad implements PesananItem /* ... implementasi hitung harga ... */
```

Penggunaan Polimorfisme

Java

```
import java.util.ArrayList;
import java.util.List;

public class MainApp {
    public static void main (String [] args) {
        List < PesananItem > keranjang = new ArrayList <> (); // ini berupa interface
        keranjang.add (new Pizza ("pepperoni", 85000, "Normal"));
        keranjang.add (new Burger ("cheekburger", 45000, true));
        keranjang.add (new Salad ("caesar", 55000, "Caesar Dressing"));

        double total_harga = 0;
        for (PesananItem item : keranjang) { // ikuti polimorfik!
            item.tampilkan_detail(); // otomatis panggil detail pizza / burger / salad benar
            total_harga += item.hitung_harga(); // otomatis panggil hitung_harga yang benar
        }
        System.out.println ("Total Harga Pesanan: Rp " + total_harga);
    }
}
```

⇒ Polimorfisme: List < PesananItem > dapat menampung Pizza, Burger dan Salad.
 Dalam loop, item-tampilkan_detail() dan item-hitung_harga () secara otomatis
 Memanggil metode yang tepat berdasarkan tipe objek aktual saat runtime.
 Tidak perlu cek instanceof atau if-else yang rumit.

⇒ Interface secara fundamental mendukung polimorfisme dengan menyediakan
 Kontrak firugam. Menciptakan sistem yang sangat fleksibel, mudah
 diubah, mudah di uji, dan skalabel. Ciri eksternal dalam peralihan turun
 Modern yang baik.



7 Abstraction Memberi menyembunyikan kompleksitas internal. Bandingkan penggunaan Abstract Class, Interface, dan Sealed Class di Java. Dalam temu kpa masing-masing lebih tepat digunakan?

Jawab:

Perbandingan Abstrak Class - Interface, Sealed Class

Fitur/Aspek	Abstrak Class (kelas abstrak)	Interface (antarmuka)	Sealed Class (kelas tertutup)
Tujuan Utama	Nendepnisikan kerangka atau "adalah tipe" (isa relationship) dengan implementasi/pemisian.	Mendefinisikan kontrak atau "bisa melakukan" (has a capability)	Membatasi dan Mengontrol "bisa melakukan" trik secara eksplisit kelas mana yang diwinkan untuk jadi subclass.
Atribut	Dua (Interface + static final)	Hanya konstanta (public static final)	Bisa (mengantarkan dipakai oleh class/interface).
Konsumidor	Bisa (di panggil oleh subclass)	Tidak bisa	Bisa (jika class)
Focus Derain	Fokus umum untuk kelas hierarki yang sangat ber variasi	Kontrak perlaku	Kontrol tetapi pada hierarki untuk exhaustive
Multiple Inheritance	Tidak di dukung untuk kelas	Disediakan untuk Interface (Melalui Implementasi)	aturannya sama seperti kelas/interface biasa, tapi ditarik permas
Evolusi (Menambah Method)	Menambah metode tidak menarik subclass. Haruskan metode abstrak	Metode abstrak	Membatasi evolusi hierarki
	Menarik subclass	Menarik kelas Mengimplemenkan (fokus default Method J&T).	Strada eksplisit

Kapan masing-masing Tepat digunakan?

1. Abstrak Class:

Cp saat memiliki kelas-kelas terdiri yang berbagai kode dan state dasar yang sama, tetapi memiliki beberapa perlaku yang harus diimplementasikan secara berbeda oleh subclass.

Contoh: Hierarki Reptilia dengan Mammalia, Reptilia, dll. Mereka berbagai limur, naga, tapi cara berular (1) berbeda.

2 Interface

Cp saat ingin mendefinisikan sebuah kontrak perlaku ("bisa Melakukan") yang bisa diimplementasikan oleh-kelas-kelas yang tidak saling terkait



Secara hierarki penawaran cocok untuk callback plugin, atau mendefinisikan kemampuan untuk hierarki. Contoh : Flyable (interface) untuk burung dan pesawat. Mereka berbeda jenis, tapi sama-sama bisa terbang.

3 Sealed Class :

Jika saat ingin membatasi dan mengontrol secara eksplisit bahwa saja yang bolehjadi subclass dari suatu kelas atau interface. Berguna untuk membangun hierarki kelas ketika dah memastikan semua implementasi sub class sudah di ketahui (misal untuk pattern matching atau exhaustive checking).

Contoh Bentuk Geometri, kroms Lingkaran, Segitiga, Persegi.

• Memastikan hanya bentuk-bentuk itu saja yang diizinkan menjadi subclass langsung.

