

# UTS Soal Essay

Nama : Angelica Barus

NIM : 4243250029

Kelas : Prik 24 A

Mata Kuliah : Pemrograman Berorientasi Objek

Dosen Pengampu : Insan Taufik, S.Kom., M.Kom.

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 :

Sistem perangkat lunak sendiri merupakan suatu kesatuan yang terdiri dari program, data, dan dokumentasi yang saling berkaitan dan bekerja sama untuk mencapai tujuan tertentu. Sementara itu, Pemrograman berorientasi Objek merupakan suatu paradigma pemrograman yang sangat cocok untuk membangun sistem perangkat lunak yang kompleks dan besar. Dimain ada 4 prinsip dalam Pemrograman berorientasi Objek, yaitu sebagai berikut :

## 1. Encapsulation

Encapsulation atau enkapsulasi adalah prinsip dalam pemrograman berorientasi objek yang mengacu pada pengemasan data (atribut) dan metode (fungsi) yang beroperasi pada data tersebut ke dalam satu unit, yaitu kelas (class). Dengan encapsulation, data dilindungi dari akses langsung oleh bagian lain dalam program. Encapsulation bekerja dengan mendeklarasikan atribut sebagai private, sehingga hanya dapat diakses melalui metode publik (getter dan setter). Dimana hal ini menjaga integritas data dan mencegah akses data yang tidak sah.

Analoginya : Kita anggap dalam sebuah kotak penyimpanan tersebut tersimpan sebuah barang-barang di dalamnya (data), yang dimana kotak penyimpanan tersebut memiliki kunci, maka barang-barang (data) yang ada dalam kotak tersebut hanya dapat diakses oleh orang yang memiliki kunci (metode) yang dapat membuka kotak penyimpanan tersebut.

## 2. Inheritance

Inheritance atau pewarisan adalah mekanisme yang memungkinkan sebuah class (subclass) untuk mewarisi atribut dan metode dari class lain (superclass), dengan kata lain inheritance memungkinkan kita untuk menciptakan kelas baru (kelas turunan) berdasarkan kelas yang sudah ada (kelas induk). Ini memungkinkan pengembangan untuk menciptakan hierarki class yang terstruktur. Dengan inheritance, kelas turunan

SIDU

dapat menggunakan kembali kode yang sudah ada dalam kelas induk, sehingga mengurangi adanya duplikasi dan memudahkan pemeliharaan. Kelas turunan juga dapat menambahkan atau mengubah penulisan yang diwarisi.

Analoginya : Inheritance diibaratkan seperti hierarki keluarga. Dimana setiap anak (subclass atau kelas turunan), mewarisi sifat-sifat (atribut dan metode) dari orang tuanya (superclass atau kelas Induk), namun anak tersebut juga dapat memiliki sifat uniknya sendiri.

### 3. Polymorphism

Polymorphism atau polimorfisme adalah kemampuan untuk menggunakan satu antarmuka untuk merepresentasikan berbagai jenis objek, dengan kata lain polymorphism memungkinkan objek dari kelas yang berbeda untuk merespon metode yang sama dengan cara yang berbeda. Ini berarti bahwa metode yang sama dapat berpilkulai berbeda tergantung pada objek yang memanggilnya. Pada polymorphism, kita dapat menulis kode yang lebih fleksibel dan dapat digunakan kembali.

Analoginya : Ibaratkan seorang seniman yang dapat beradaptasi dengan berbagai medium seni, seperti seorang seniman dapat melukis, memahat, ataupun bermain musik. Meskipun semua aktivitas tersebut berbeda, mereka dapat dilakukan dengan cara yang sama, yaitu melalui kreativitas seniman.

### 4. Abstraction

Abstraction atau abstraksi adalah proses menyembunyikan detail implementasi yang kompleks dan hanya menampilkan antarmuka yang diperlukan kepada pengguna.

Dengan kata lain, abstraction membantu menyederhanakan interaksi dengan objek. Dengan menggunakan abstraction, pengembang dapat fokus pada apa yang dilakukan objek, bukan bagaimana objek tersebut melakukannya. Ini memungkinkan pengguna untuk berinteraksi dengan objek tanpa perlu memahami semua detail teknis dibaliknya.

Analoginya : Abstraction diibaratkan dengan sebuah mobil, dimana pengemudi tidak perlu mengetahui semua detail teknis di dalam mesin mobil untuk dapat mengemukannya. Pengemudi hanya perlu memahami cara mengendalikan mobil tersebut seperti cara mengoperasikan pedal gas, rem, dan kemudi.

Jadi dapat disimpulkan bahwa kumpulan prinsip pemrograman berorientasi objek ini saling berkait dan saling mendukung dalam membangun sistem perangkat lunak yang mudah dikembangkan dan dipelihara. Dimana encapsulation melindungi data, inheritance meningkatkan reusability, polymorphism meningkatkan fleksibilitas dan abstraction menyederhanakan interaksi.

2. Apa kelebihan menggunakan Java versi terbaru (Java 21) dibanding versi-versi sebelumnya dalam konteks pengembangan berbasis OOP? Berikan minimal 2 fitur modern Java 21 dan jelaskan bagaimana fitur tersebut menyederhanakan pengembangan aplikasi OOP.
- Jawab :

Java adalah bahasa pemrograman yang dikembangkan oleh Sun Microsystems dan sekarang bagian dari Oracle Corporation dan juga pertama kali dirilis pada tahun 1995. Java dirancang dengan prinsip Write Once, Run Anywhere (WORA), yang berarti bahwa kode yang ditulis dalam Java dapat dijalankan di berbagai platform tanpa perlu adanya modifikasi. Setiap versi Java membawa pembaharuan fitur, peningkatan performa, dan peningkatan kemanan. Java 21 dirilis pada September 2023 dan menambahkan versi LTS (Long Term Support) terbaru yang menyempurnakan fitur-fitur modern serta menghadirkan stabilitas, efisiensi, dan sintaks yang lebih ringkas. Java 21 fokus pada peningkatan produktivitas pengembang, performa aplikasi, serta penyederhanaan pengembangan berbasis OOP.

Jika dibandingkan dengan versi-versi sebelumnya seperti versi Java 8, Java 11, dan Java 17, Java 21 ini memiliki sejumlah keunggulan. Dalam versi-versi sebelumnya, pengembang sering kali dihadapkan pada penulisan kode yang panjang dan berulang, terutama saat membuat kelas-kelas data sederhana yang membutuhkan banyak metode seperti konstruktur, getter, equals(), hashCode(), dan toString(). Selain itu, dalam menangani berbagai tipe objek dalam alur logika program, pengembang harus menulis banyak pernyataan if-else atau instanceof yang membuat kode menjadi sulit dibaca dan dipelihara.

Sementara itu, Java 21 hadir dengan fitur-fitur yang menyederhanakan tantangan tersebut. Misalnya, dengan adanya record classes, pengembang dapat mendefinisikan class data dengan hanya satu baris kode, tanpa harus menulis metode-metode dasar secara manual. Kemudian, dengan pattern matching for switch, proses pengecekan tiap objek dan pengambilan tindakan berdasarkan tipe tersebut bisa dilakukan dengan cara yang lebih cepat dan struktural. Maka, kelebihan menggunakan Java 21 adalah sintak yang lebih sederhana dan deklaratif sehingga membuat kode lebih bersih dan mudah dipahami, struktur data dan hierarki objek lebih baik dan struktural, dan juga produktivitas pengembang meningkat karena kode lebih ringkas dan ekspressif.

Berikut fitur-fitur modern Java 21, sebagai berikut :

### 1. Record classes

Record classes adalah fitur Java yang memungkinkan pengembang untuk mendefinisikan kelas data dengan sintak yang lebih ringkas. Saat menggunakan record, maka Java secara otomatis menghasilkan constructor, method equals(), hashCode(), dan toString() berdasarkan field yang dideklarasikan. Dalam paradigma OOP, kita sering membuat kelas untuk menyimpan data. Sebelum adanya record, pengembang harus menulis kode panjang dan berulang hanya untuk mendefinisikan kelas sederhana. Dengan record, semua itu cukup dituliskan dalam satu baris. Ini mengurangi boilerplate code,

meningkatkan keterbacaan, serta mempercepat pengembangan dan pemeliharaan kode.

Contoh: Jika ingin mendefinisikan kelas 'Dosen', kita dapat menggunakan record untuk mendeklarasikan dengan cara efisien, seperti berikut contohnya:

```
public record Dosen (String nama, String Nip, String unit) {}
```

## 2. Pattern Matching for Switch

Fitur ini menyederhanakan penggunaan pernyataan switch dengan memungkinkan pengembang untuk melakukan pattern matching. Pattern matching menyederhanakan penanganan objek yang termasuk dalam hierarki class (polymorphism). Alih-alih menulis banyak kode if-else dalam instanceof, kita cukup menuliskannya dalam satu blok switch yang bersih dan mudah dibaca. Hal ini mempercepat proses pengambilan keputusan berdasarkan tipe objek, serta menjaga agar kode tetap ringkas dan terstruktur.

Contoh: Kita dapat menggunakan pattern matching untuk memproses berbagai jenis objek, berikut contoh kodennya:

```
switch (objek) {  
    case Mahasiswa m -> System.out.println("Mahasiswa : " + m.nama());  
    case Dosen d -> System.out.println("Dosen : " + d.getNama());  
    default -> System.out.println("Objek tidak diketahui");  
}
```

## 3. Sealed Classes

Sealed classes adalah class yang membatasi siapa saja yang boleh mewarainya. Dengan keyword sealed, kita bisa menentukan subclass manu sajai yang diizinkan untuk menjadi turunan dari sebuah class tertentu. Subclass harus ditandai dengan final, sealed atau non-sealed. Dengan sealed classes, pengembang pun secara langsung mendefinisikan hierarki turunan, sehingga mencegah penyalahgunaan inheritance dan menjaga desain arsitektur tetap bersih. Ini sangat membantu dalam sistem besar dan kompleks, di mana pengendalian pewarisan sangat penting untuk konsistensi dan keintiman kodai.

Berikut contoh kode: 

```
public sealed class Kendaraan permits Pesawat, Kapal {}  
final class Pesawat extends Kendaraan {}  
final class Kapal extends Kendaraan {}
```

3 Mahasiswa sering kali 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.

Jawabannya:

Dalam pemrograman berorientasi objek, dua konsep inti yang selalu muncul adalah class dan object. Class adalah struktur, cetakan atau blueprint yang digunakan untuk

SiDU

membuat objek. Class mendefinisikan atribut (variabel) dan perilaku (method/fungsi) yang akan dimiliki oleh objek. Class bersifat abstrak karena tidak mewujudkan data nyata, melainkan hanya kerangka bagi manusia objek akan dibentuk di sana berpindah. Class adalah definisi umum yang menggambarkan struktur suatu entitas dalam sistem. Misalnya, class Mahasiswa mendefinisikan bahwa setiap mahasiswa memiliki nama, nim, dan program studi, serta dapat melakukan sesuatu seperti cetakData() atau tampilkanInfo().

Sebaliknya, object adalah instansiasi dari class. Object memuat data spesifik sesuai dengan struktur yang ditetapkan oleh class. Object diciptakan dari class, dan masing-masing object bisa memiliki nilai atribut yang berbeda satu sama lain, meskipun berasal dari class yang sama. Object memiliki nilai konkret dan bisa menjalankan fungsi yang telah ditentukan dalam class. Object tidak dapat ada tanpa class, karena object bergantung pada definisi class untuk eksistensinya. Namun, satu class bisa digunakan untuk membuat banyak object yang berbeda, yang masing-masing menyimpan data spesifik. Misalnya dari class Mahasiswa, kita bisa membuat object mahasiswa yang mewakili "Angel" dan mahasiswa2 yang mewakili "Rosi". Keduanya adalah mahasiswa, tapi data dalam identitas mereka berbeda.

Pembeda lainnya adalah class hanya dideklarasikan dekali dalam kode, sedangkan object bisa diciptakan berkali-kali dalam satu class. Class juga tidak membutuhkan memori runtime secara langsung, tetapi object dibuat saat program berjalan (runtime) dan memori untuk menyimpan datanya, sehingga class bersifat statis, sedangkan object bersifat dinamis. Class biasanya dideklarasikan dalam file atau modul program sebagai definisi tipe data, sedangkan object diciptakan di dalam method seperti main() untuk digunakan dan dijalankan. Class juga tidak bisa digunakan secara langsung untuk memproses data, class harus diinstansiasi terlebih dahulu sedangkan object bisa digunakan untuk mengakses atribut, memodifikasi data, dan menjalankan method. Makin perbedaan dari object dan class dapat dilihat mulai dari berbagai aspek seperti Definisi, Data, Eksekusi, keberadaan, ketergantungan, memori dan jumlah.

Berikut adalah contoh penggunaan class dan object dalam konteks program manajemen data siswa.

### (1) class

```
public class Mahasiswa {  
    String nama;  
    String nim;  
    String programStudi;  
  
    public void cetakData() {  
        System.out.println("Nama : " + nama);  
        System.out.println("NIM : " + nim);  
        System.out.println("Program studi : " + programStudi);  
    }  
}
```

## (2) Object

```
public class Main {  
    public static void main(String[] args) {  
        Mahasiswa mahasiswa1 = new Mahasiswa("Angel", "4213250029", "Ilmu Komputer");  
        Mahasiswa mahasiswa2 = new Mahasiswa("Rosa", "4213250055", "Ilmu Komputer");  
  
        mahasiswa1.tampilkanInfo();  
        mahasiswa2.tampilkanInfo();  
    }  
}
```

Kesalahan umum mahasiswa dalam memahami perbedaan antara class dan object adalah menganggap keduanya adalah sama, padahal class disini adalah konsepnya sedangkan object adalah implementasinya.

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?

Jawaban :

Berikut adalah class BankAccount :

```
public class BankAccount {  
    private double balance;  
  
    public BankAccount(double initialBalance) {  
        if (initialBalance >= 0) {  
            this.balance = initialBalance;  
        } else {  
            this.balance = 0;  
            System.out.println("Saldo awal tidak valid");  
        }  
    }  
}
```

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

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

SiDU

// lanjutkan kode

```
System.out.println("Deposit berhasil : " + amount);
} else {
    System.out.println("Jumlah deposit harus lebih dari 0.");
}
}

public boolean withdraw(double amount) {
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        System.out.println("Penarikan berhasil : " + amount);
        return true;
    } else {
        System.out.println("Penarikan gagal : saldo tidak mencukupi atau jumlah tidak valid");
        return false;
    }
}
```

Encapsulation adalah prinsip dalam pemrograman berorientasi objek yang mengacu pada pengemasan data (atribut) dan metode (fungsi) yang beroperasi pada data tersebut ke dalam satu unit, yaitu kelas (class). Dalam class BankAccount diatas, atribut balance dibuat sebagai private, artinya hanya dapat diakses oleh class BankAccount itu sendiri. Dengan cara ini, data penting seperti saldo tetap tidak dapat dimain-pakai secara langsung dari luar class.

Agar tetap bisa mengakses nilai saldo, disediakan method getter (getBalance()) yang bersifat public, sehingga objek-luar dapat membaca nilai saldo tanpa bisa mengubahnya. Untuk mengubah saldo, hanya diperbolehkan melalui method deposit() dan withdraw(), yang keduanya memiliki logika validasi untuk memastikan hanya nilai-nilai yang salah yang bisa memodifikasi saldo. Hal ini mencegah kondisi berbahaya seperti saldo menjadi negatif karena penarikan berlebih, atau saldo dimasukkan langsung menjadi angka tidak masuk akal seperti -1.000.000. Dengan kata lain, encapsulation memastikan bahwa semua perubahan data melalui mekanisme resmi yang terkontrol, bukan dengan akses bebas.

Encapsulation penting untuk keamanan sistem karena Encapsulation memungkinkan untuk melindungi data dari akses ilegal atau tidak sah. Dalam sistem seperti tetapan bank, data seperti saldo sangat sensitif. Tanpa encapsulation, data tersebut bisa saja langsung diubah oleh sistem luar tanpa validasi. Dengan encapsulation, pengembang dapat mengatur logika internal dalam method setter seperti deposit() atau withdraw() untuk mencegah perubahan yang tidak sah. Encapsulation juga membuat kode lebih mudah ditulis dan dikembangkan. Apabila suatu saat logika withdraw() ingin

SiDU



Dipindai dengan CamScanner

diubah misalnya untuk menambahkan biaya administrasi, dapat dengan cukup diubah di dalam method tersebut tanpa mengubah bagian lain dari sistem. Terakhir Encapsulation juga membantu membuat sistem menjadi modular, dimana class hanya memperlakukan bagian yang memang perlu diketahui pengguna (public API) dan menyembunyikan rincian internalnya (private data). Hal ini membuat sistem terjaga keamanannya dan terkontrol dengan baik.

5. Jelaskan bagaimana mekanisme constructor chaining berjalan pada pewarisan di Java. Apa yang terjadi jika constructor pada superclass tidak dipanggil secara eksplisit? Sertakan ilustrasi class Karyawan dari subclass Manager.

Jawaban :

Constructor chaining adalah mekanisme dalam pemrograman berorientasi objek khususnya di Java, di mana satu constructor memanggil constructor lain dalam hierarki kelas. Constructor chaining menjalankan praktik pemanggilan satu constructor dari constructor lainnya, baik dalam kelas yang sama maupun antar kelas yang memiliki hubungan pewarisan (inheritance). Tujuan utama dari constructor chaining adalah untuk memastikan bahwa semua prosesinisialisasi objek berjalan secara benar dan sistematis. Dalam Java, pemanggilan constructor dalam kelas yang sama dilakukan menggunakan keyword `this()`, sedangkan pemanggilan constructor dari superclass / kelas induk dilakukan menggunakan `super()`.

Saat bekerja pada pewarisan, objek dari subclass tidak dapat dibuat secara lengkap tanpa terlebih dahulu memanggil constructor dari superclassnya. Ini karena subclass mewarisi atribut dan perilaku dari superclass, sehingga constructor superclass harus dipanggil terlebih dahulu untuk menginisialisasi bagian-bagian tersebut. Jika constructor pada superclass tidak dipanggil secara eksplisit menggunakan `super()`, maka Java secara otomatis akan menyediakan pemanggilan `super()`; pada basis pertama constructor pada subclass. Hal ini hanya akan berhasil jika superclass memiliki constructor default (tanpa parameter). Namun jika superclass hanya menyediakan constructor dengan parameter dan subclass tidak memanggilnya secara eksplisit, maka Java tidak dapat mengkompilasi program tersebut dan akan menghasilkan error.

Sebagai ilustrasi, kita gunakan kelas induk bernama `Karyawan` yang memiliki constructor dengan dua parameter, yaitu `nama` dan `gaji`. Kemudian kita gunakan subclass bernama `Manager` yang mewarisi dari `Karyawan` dan menambahkan atribut `departemen`. Agar objek `Manager` dapat dibuat dengan benar, constructor di kelas `Manager` harus secara eksplisit memanggil constructor di kelas `Karyawan`, dengan memberikan parameter yang sesuai seperti `super(nama, gaji)`. Ini menjamin bahwa atribut `nama` dan `gaji` yang diwarisi dari `Karyawan` akan terisi dengan benar, sebelum atribut tambahan seperti `departemen` diinisialisasi dalam constructor.

SiDU

Manager. Berikut adalah contoh codonya

```
public class Karyawan {
```

```
    String nama;
```

```
    double gaji;
```

```
    public Karyawan(String nama, double gaji) {
```

```
        this.nama = nama;
```

```
        this.gaji = gaji;
```

```
        System.out.println("Constructor Karyawan dijalankan");
```

```
}
```

```
}
```

```
public class Manager extends Karyawan {
```

```
    String department;
```

```
    public Manager(String nama, double gaji, String department) {
```

```
        super(nama, gaji);
```

```
        this.department = department;
```

```
        System.out.println("Constructor Manager dijalankan");
```

```
}
```

```
    public void tampilkanInfo() {
```

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

```
        System.out.println("Gaji : " + gaji);
```

```
        System.out.println("Departemen : " + department);
```

```
}
```

```
}
```

Jika objek dari kelas Manager dibuat, maka akan terlebih dahulu dijalankan constructor Karyawan, kemudian baru constructor Manager, dan akhirnya informasi lengkap dapat ditampilkan. Apabila dalam kode tersebut kita tidak menuliskan code super(nama, gaji); dan constructor Karyawan tidak menyediakan constructor default tanpa parameter maka program tidak dapat dikompilasi.

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.

Jawaban:

Interface adalah sebuah kontrak yang mendefinisikan sekumpulan metode yang harus

SiDU

dilakukan oleh kelas yang mengimplementasikan interface tersebut. Interface memungkinkan pengembang untuk mendefinisikan perilaku yang dapat diwariskan oleh berbagai kelas yang tidak memiliki hubungan hierarkis, sehingga mendukung prinsip pemrograman berorientasi objek seperti polymorphism. Polymorphism memungkinkan satu referensi dari tipe interface digunakan untuk memanggil metode pada berbagai jenis objek yang berbeda, selama objek tersebut mengimplementasikan interface yang sama.

Penggunaan interface mendukung konsep polymorphism dengan menyediakan struktur umum bagi kelas-kelas yang beragam. Melalui interface, kita dapat mendefinisikan perilaku umum, lalu memberikan setiap kelas yang berbeda implementasinya sesuai dengan kebutuhannya masing-masing. Ketika kita memanggil metode dari interface, Java akan secara otomatis memilih implementasi metode yang sesuai dengan tipe objek aktual yang digunakan.

Meskipun kelas-kelas tersebut memiliki implementasi berbeda, selama semuanya mengacu pada interface yang sama, kita dapat memanggil metode yang sama.

Berikut merupakan contoh penggunaannya dalam sistem pemesanan makanan online:

```
interface LayananPengiriman {  
    void kirimPesanan(String pesanan);  
}
```

```
class GoFood implements LayananPengiriman {  
    public void kirimPesanan(String pesanan) {  
        System.out.println("Mengirim pesanan melalui GoFood: " + pesanan);  
    }  
}
```

```
class GrabFood implements LayananPengiriman {  
    public void kirimPesanan(String pesanan) {  
        System.out.println("Mengantarkan pesanan melalui GrabFood: " + pesanan);  
    }  
}
```

```
class ShopeeFood implements LayananPengiriman {  
    public void kirimPesanan(String pesanan) {  
        System.out.println("Pesanan dikirim via ShopeeFood: " + pesanan);  
    }  
}
```

```

public class AplikasiPemesanan {
    public static void main(String[] args) {
        LayananPengiriman[] layanan = {
            new GoFood(),
            new GrabFood(),
            new ShopeeFood(),
        };
        String namaPesanan = "Nasi Goreng Seafood";
        for (LayananPengiriman : layanan) {
            layananPengiriman.kirimPesanan(namaPesanan);
        }
    }
}

```

Kode diatas menunjukkan penggunaan interface LayananPengiriman untuk mengekspresikan konsep polymorphism dalam sistem pemesanan makanan online. Interface ini mendefinisikan metode kirimPesanan() yang wajib diimplementasikan oleh setiap kelas yang mewakili layanan pengiriman, seperti GoFood, GrabFood dan ShopeeFood. Masing-masing kelas memberikan implementasi berbeda sesuai cara pengiriman masing-masing platform. Di dalam kelas AplikasiPemesanan, array berisi LayananPengiriman digunakan untuk mendampingi semua jenis layanan, lalu metode kirimPesanan() dipanggil secara variasi.

7 Abstraction membantu menyembunyikan kompleksitas internal. Bandingkan penggunaan abstract class, interface, dan sealed class di Java. Dalam kasus apa masing-masing lebih tepat digunakan?

Jawaban:

(i) Abstract class

Abstract class adalah kelas dasar yang tidak dapat dibuat objek langsung dari nya. Biasanya digunakan saat ada perilaku umum yang ingin diwariskan ke subclasses. Abstract class bisa memiliki method dengan atau tanpa isi. Abstract class lebih tepat digunakan ketika ada kebutuhan untuk berbagi kode diantara beberapa kelas yang memiliki hubungan hierarkis. Misalnya jika kita memiliki beberapa jenis kendaraan seperti mobil, motor, sepeda. Kita dapat membuat abstract class Kendaraan yang memiliki metode umum seperti bergerak() dan atribut seperti Kecepatan.

Contoh kelas : ketika kita ingin mendefinisikan perilaku umum dan juga menyediakan implementasi dasar yang dapat digunakan oleh subclass, seperti dalam hierarki kelas kendaraan

Contoh kode :

```
public abstract class Kendaraan {  
    protected String merk;  
  
    Kendaraan(String merk) {  
        this.merk = merk;  
    }  
  
    public abstract void bergerak();  
  
    public void info() {  
        System.out.println("Merk kendaraan : " + merk)  
    }  
  
    class Mobil extends Kendaraan {  
  
        public Mobil(String merk) {  
            super(merk);  
        }  
    }  
}
```

@Override

```
public void bergerak() {  
    System.out.println("Mobil bergerak dengan kecepatan normal.");  
}  
}
```

Penjelasan : Kendaraan adalah abstract class karena memiliki metode abstract 'bergerak()' yang tidak memiliki implementasi di kelas induk, tetapi harus diimplementasikan oleh kelas anak seperti Mobil.

(1) Interface

Interface adalah sebuah kontrak yang mendefinisikan sekumpulan metode yang harus diimplementasikan oleh kelas yang mengimplementasikan interface tersebut. Interface lebih tepat digunakan jika ingin mendefinisikan perilaku yang dapat diturunkan oleh berbagai kelas yang tidak memiliki hubungan hierarkis. Misalnya

SiDU

Jika kita memiliki berbagai jenis hewan yang dapat bersuara, kita dapat membuat interface 'Bersuara' dengan metode suarai).

Contoh kasus : ketika ingin mendefinisikan perilaku yang dapat ditampilkan oleh berbagai kelas yang tidak terkait, seperti dalam kasus hewan yang dapat bernyanyi

Contoh kode :

```
public interface Bersuara {  
    void suarai();  
}
```

```
class Kucing implements Bersuara {
```

@Override

```
public void suarai() {  
    System.out.println("Meong Meong");  
}
```

```
}
```

```
class Kambing implements Bersuara {
```

@Override

```
public void suarai() {  
    System.out.println("mbek mbek");  
}
```

```
}
```

Penjelasan : Interface 'Bersuara' mendefinisikan kontrak bahwa kelas apapun yang mengimplementasikannya harus memiliki metode 'suarai'. Baik kucing maupun kambing mengimplementasikan metode tersebut dengan cara masing-masing.

### (3) Sealed class

Sealed class adalah class yang membatasi siapa saja yang boleh mewariskanya.

Sealed class lebih tepat digunakan ketika ingin membatasi subclass yang dapat dibuat dari kelas tertentu, sehingga memberikan kontrol lebih besar atas hierarki kelas. Dengan kata lain, sealed class digunakan ketika kita ingin membatasi pewarisan untuk menjaga kontrol atas hierarki kelas.

Contoh kasus : Misalnya jika kita memiliki kelas Transportasi dan hanya ingin mengizinkan subclass tertentu seperti motor, mobil, sepeda, kita dapat mendeklarasikan Transportasi sebagai sealed class

Contoh kode :

```
public sealed class Transportasi permits Motor, Mobil, Sepeda {  
    public void info() {  
    }
```

SiDU

```
System.out.println("ini adalah transportasi");
}

final class Motor extends Transportasi {
    @Override
    public void info() {
        System.out.println("ini adalah motor");
    }
}

final class Mobil extends Transportasi {
    @Override
    public void info() {
        System.out.println("ini adalah Mobil");
    }
}
```

```
final class Sepeda extends Transportasi {
    @Override
    public void info() {
        System.out.println("ini adalah sepeda");
    }
}
```

Penjelasan: Transportasi dideklarasikan sebagai sealed class yang membatasi hanya kelas Motor, Mobil dan Sepeda yang boleh menjadi subclassnya