Nama : M. Ainur Ramadhan

NRP : 3122500047

Kelas : 2 D3 Teknik Informatika B (Semester 3)

UTS : Mata Kuliah Konsep Pemrograman Berbasis Objek

1. Sebutkan minimal 3 perbedaan overriding dan overloading?
   Jawab:

   Terdapat beberapa perbedaan:

   1. **Perbedaan Definisi:**
      - Overriding: Overriding terjadi ketika sebuah subclass (turunan) mengganti implementasi dari sebuah metode yang sudah ada di superclass (induk). Dalam overriding, nama metode, parameter, dan tipe kembalian harus sama antara subclass dan superclass.
      - Overloading: Overloading terjadi ketika terdapat dua atau lebih metode dalam sebuah kelas dengan nama yang sama, tetapi jumlah, urutan, atau tipe parameter yang berbeda. Overloading terkait dengan metode yang memiliki nama yang sama tetapi dengan parameter yang berbeda.
   2. **Perbedaan Pewarisan:**
      - Overriding: Overriding terkait dengan konsep pewarisan (inheritance) dan digunakan untuk mengubah perilaku metode yang diwarisi dari superclass dalam subclass.
      - Overloading: Overloading tidak terkait dengan pewarisan; itu hanya mengizinkan kelas yang sama memiliki beberapa implementasi metode dengan nama yang sama.
   3. **Perbedaan Parameter:**
      - Overriding: Metode overriding harus memiliki parameter yang sama persis (nama, tipe, dan urutan) dengan metode yang diwarisi dari superclass.
      - Overloading: Metode overloading harus memiliki parameter yang berbeda dalam hal jumlah, urutan, atau tipe data.
   4. **Perbedaan Penggunaan:**
      - Overriding: Overriding digunakan untuk mengubah metode yang sudah ada dalam superclass sehingga sesuai dengan kebutuhan subclass.
      - Overloading: Overloading digunakan ketika kita ingin memiliki beberapa versi dari metode yang sama dalam sebuah kelas untuk menangani berbagai kasus atau tipe data.

2. Mengapa sebuah kelas perlu di enkapsulasi?
   Jawab:

   Terdapat beberapa alasan penting mengapa sebuah kelas perlu dienkapsulasi:
   1. **Pengendalian Akses:** Enkapsulasi memungkinkan kita untuk mengontrol akses ke data kelas dengan mendefinisikan aksesibilitasnya (public, private, protected, dsb.). Ini memungkinkan kita untuk melindungi data yang sensitif atau kritis dalam kelas sehingga hanya metode-metode dalam kelas itu sendiri yang dapat mengakses atau

memodifikasinya. Ini membantu mencegah perubahan yang tidak diinginkan atau kesalahan yang disebabkan oleh akses yang tidak terkontrol.

2. **Abstraksi:** Dengan enkapsulasi, kita dapat menyembunyikan rincian implementasi internal kelas dari pengguna eksternal. Pengguna eksternal hanya perlu tahu cara menggunakan antarmuka publik kelas tersebut tanpa perlu peduli tentang bagaimana data atau operasi diimplementasikan. Ini memungkinkan pemisahan antara apa yang dilakukan oleh sebuah objek dan bagaimana objek tersebut melakukannya.
3. **Maintenance dan Perubahan:** Enkapsulasi membuat perubahan internal dalam kelas menjadi lebih aman. Jika kita ingin mengubah implementasi internal suatu kelas (misalnya, mengganti tipe data variabel atau metode yang digunakan), kita dapat melakukannya tanpa mempengaruhi kode yang menggunakan kelas tersebut, asalkan antarmuka publiknya tetap tidak berubah. Ini memfasilitasi pemeliharaan dan pengembangan berkelanjutan.
4. **Reusable dan Modular:** Enkapsulasi memungkinkan kita untuk membuat kelas-kelas yang dapat digunakan kembali (reusable) dengan mudah. Kita dapat menginstansiasi objek dari kelas tersebut di berbagai bagian program kita tanpa harus menulis ulang kode yang sama. Ini mempromosikan konsep modularitas dalam pemrograman, yang merupakan salah satu prinsip penting dalam pengembangan perangkat lunak.
5. **Keamanan:** Dengan membatasi akses langsung ke data kelas, enkapsulasi membantu dalam menjaga keamanan data. kita dapat menerapkan validasi dan kontrol akses di dalam metode-metode yang berinteraksi dengan data, sehingga mencegah perubahan yang tidak sah atau merusak.

3. Mengapa ada konsep inheritance pada sebuah kelas?
   Jawab:

   Karena konsep inheritance dalam pemrograman berorientasi objek memungkinkan kelas turunan (subclass) untuk mewarisi sifat dan perilaku dari kelas induk (superclass), yang membantu dalam penggunaan kembali kode, abstraksi, perluasan, dan struktur hierarki yang terorganisir. Ini membuat kode lebih efisien dan mudah dimengerti.

4. Apa kegunaan inheritance pada saat kita membuat aplikasi?
   Jawab:

   Kegunaan inheritance (pewarisan) dalam pembuatan aplikasi adalah:

   1. **Penggunaan Kembali Kode (Code Reusability):** Inheritance memungkinkan kita untuk menggunakan kembali kode yang sudah ada dalam kelas-kelas yang telah teruji dan terbukti. Ini menghemat waktu dan usaha dalam pengembangan aplikasi.
   2. **Abstraksi dan Generalisasi:** Kita dapat membuat kelas-kelas abstrak yang menggambarkan sifat-sifat umum atau kerangka kerja yang dapat digunakan oleh banyak bagian dalam aplikasi. Ini memungkinkan untuk mengidentifikasi kesamaan dalam aplikasi dan membuat kelas-kelas umum yang berfungsi sebagai dasar.
   3. **Perpanjangan (Extension):** Inheritance memungkinkan kita untuk memperluas atau mengubah perilaku yang sudah ada sesuai dengan kebutuhan aplikasi. Kita dapat menambahkan fungsi baru atau mengubah fungsi yang ada tanpa harus mengganggu kode yang sudah ada.
   4. **Hierarki dan Struktur yang Terorganisir:** Dengan inheritance, Kita dapat membentuk hierarki kelas yang terstruktur dan terorganisir. Ini memudahkan

pemahaman dan pemeliharaan aplikasi karena hubungan antara kelas-kelas dijelaskan dengan jelas.

5. **Polimorfisme:** Inheritance mendukung konsep polimorfisme, yang memungkinkan objek dari kelas-kelas yang berbeda untuk digunakan dengan cara yang serupa. Ini memfasilitasi penanganan objek secara fleksibel dalam aplikasi.
6. **Memisahkan Tanggung Jawab:** Inheritance membantu dalam memisahkan tanggung jawab dan peran berbagai kelas dalam aplikasi. Ini membuat desain aplikasi lebih terstruktur dan memudahkan pemeliharaan.
7. **Pengembangan yang Efisien:** Jika kita perlu memodifikasi atau menambahkan fitur baru ke aplikasi, inheritance memungkinkan kita untuk melakukannya dengan lebih efisien. Kita dapat melakukan perubahan di kelas induk dan semua kelas turunan akan mendapatkan manfaatnya.
8. **Pengelolaan Kompleksitas:** Inheritance membantu dalam mengelola kompleksitas aplikasi dengan memecahnya menjadi bagian-bagian yang lebih kecil, terfokus, dan mudah dikelola.

5. Studi kasus misalkan saya mau membuat kelas PrinterAIO, printer all in one yang bisa printer, scan, dan fax dengan cara menurunkan dari kelas Printer, kelas Scanner, dan kelas Fax, bagaimana menurut Anda?
   Jawab:

   Pada Studi kasus tersebut kita dapat membuat hierarki kelas dengan sebuah kelas **PrinterAIO** (Printer All-in-One) yang menurunkan sifat dan perilaku dari kelas-kelas **Printer**, **Scanner**, dan **Fax**. Berikut untuk implementasinya:

   1. **Kelas Printer:** Kelas **Printer** akan memiliki metode-metode yang terkait dengan mencetak, seperti **printDocument()**, **checkInkLevel()**, dll.

   2. **Kelas Scanner:** Kelas **Scanner** akan memiliki metode-metode yang terkait dengan pemindaian, seperti **scanDocument()**, **adjustResolution()**, dll.

   3. **Kelas Fax:** Kelas **Fax** akan memiliki metode-metode yang terkait dengan pengiriman faks, seperti **sendFax()**, **receiveFax()**, dll.

   4. **Kelas PrinterAIO (Printer All-in-One):** Kelas **PrinterAIO** akan menjadi kelas turunan yang menggabungkan sifat-sifat dari kelas **Printer**, **Scanner**, dan **Fax**. Ini dapat dilakukan dengan menyatakan **PrinterAIO** sebagai subclass dari ketiga kelas tersebut.

   Berikut Implementasi dalam kode program:

```
// Kelas Printer
class Printer {
    private int inkLevel;

    public Printer() {
        inkLevel = 100;
    }

    public void printDocument(String document) {
        System.out.println("Printing: " + document);
```

```
    }

    public int checkInkLevel() {
        return inkLevel;
    }
}

// Kelas Scanner
class Scanner {
    private int resolution;

    public Scanner() {
        resolution = 300;
    }

    public void scanDocument(String document) {
        System.out.println("Scanning: " + document);
    }

    public void adjustResolution(int resolution) {
        this.resolution = resolution;
    }
}

// Kelas Fax
class Fax {
    private String faxNumber;

    public Fax() {
        faxNumber = "";
    }

    public void sendFax(String document) {
        System.out.println("Sending Fax to: " + faxNumber);
    }

    public void receiveFax() {
        System.out.println("Receiving Fax");
    }
}

// Kelas PrinterAIO (Printer All-in-One)
class PrinterAIO extends Printer, Scanner, Fax {
    public PrinterAIO() {
        super();
    }

    @Override
```

```
    public void printDocument(String document) {
        System.out.println("Printing (All-in-One): " + document);
    }

    @Override
    public void scanDocument(String document) {
        System.out.println("Scanning (All-in-One): " + document);
    }

    @Override
    public void sendFax(String document) {
        System.out.println("Sending Fax (All-in-One): " + document);
    }
}

public class Main {
    public static void main(String[] args) {
        PrinterAIO aioPrinter = new PrinterAIO();
        aioPrinter.printDocument("Document1");
        aioPrinter.scanDocument("Document2");
        aioPrinter.sendFax("Document3");
    }
}
```