Modul 12 Encapsulation

Tugas Pendahuluan :

1. Apa yang dimaksud dengan encapsulation dalam pemrograman berorientasi objek (OOP)? Jelaskan prinsip-prinsip utama dari encapsulation.
2. Mengapa penting untuk mengimplementasikan encapsulation dalam pengembangan perangkat lunak?
3. Apa yang dimaksud dengan variabel privat dalam sebuah class Java? Bagaimana Kita dapat mengakses variabel privat tersebut dari luar class?
4. Mengapa kita menggunakan metode getter dan setter dalam encapsulation? Berikan contoh situasi di mana penggunaan metode getter dan setter bermanfaat.
5. Bagaimana cara mengimplementasikan encapsulation dalam Java? Berikan contoh sederhana penggunaan variabel privat dan metode getter-setter.
6. Mengapa variabel harus dideklarasikan sebagai privat, dan metode akses didefinisikan sebagai public dalam prinsip encapsulation?

Materi :

Encapsulation adalah salah satu dari empat konsep dasar dalam pemrograman berorientasi objek (OOP), bersama dengan abstraksi, inheritance, dan polymorphism. Konsep ini mengacu pada pembungkusan data (variabel) dan metode yang beroperasi pada data dalam sebuah unit tunggal yang disebut class. Dalam Java, encapsulation diimplementasikan menggunakan modifier akses (seperti private, protected, dan public) untuk mengendalikan akses ke variabel dan metode dalam class.

Prinsip-prinsip Encapsulation:

1. Variabel Privat: Variabel yang terdapat dalam class sebaiknya dideklarasikan sebagai private, yang berarti mereka hanya dapat diakses secara langsung oleh metode yang ada dalam class tersebut. Ini membatasi akses langsung dari luar class dan mencegah perubahan yang tidak terkontrol.
2. Metode Publik: Metode yang digunakan untuk mengakses atau mengubah nilai variabel yang bersifat privat sebaiknya di-deklarasikan sebagai public. Ini memberikan cara terkontrol dan aman untuk mengakses dan memanipulasi data yang terdapat dalam class.

3. Getter dan Setter: Untuk mengakses dan mengubah nilai variabel privat, class sebaiknya menyediakan metode getter (untuk mendapatkan nilai) dan setter (untuk mengatur nilai). Ini memungkinkan validasi dan kontrol akses ke data.

Manfaat Encapsulation:

1. Kontrol Akses: Membatasi akses langsung ke variabel privat sehingga menghindari perubahan yang tidak diinginkan atau tidak terkontrol dari luar class.
2. Keamanan: Data yang bersifat privat tidak dapat diubah secara sembarangan, sehingga meningkatkan keamanan program.
3. Perubahan Internal: Mengizinkan class untuk mengubah implementasinya tanpa memengaruhi kode yang menggunakan class tersebut. Ini memungkinkan perbaikan atau perubahan internal tanpa mempengaruhi client.
4. Validasi Data: Dengan menggunakan metode setter, Kita dapat melakukan validasi data sebelum mengatur nilainya. Misalnya, memeriksa apakah nilai dalam rentang yang benar.
5. Dokumentasi: Dapat memberikan dokumentasi yang jelas tentang bagaimana cara mengakses dan menggunakan class, karena metode yang publik berfungsi sebagai antarmuka.
6. Pengelolaan Kompleksitas: Mengelompokkan data dan metode yang terkait dalam satu unit class membantu mengelola kompleksitas program.

Dalam implementasinya, encapsulation merupakan salah satu konsep yang fundamental dalam OOP dan berperan penting dalam membangun perangkat lunak yang lebih aman, terstruktur, dan mudah dipelihara.

Encapsulation harus digunakan dalam pemrograman berorientasi objek (OOP) ketika Kita ingin mencapai beberapa tujuan tertentu, termasuk:

1. Kontrol Akses Data: Ketika Kita ingin mengontrol siapa yang dapat mengakses dan mengubah data dalam sebuah class. Dengan menggunakan variabel privat dan metode publik, Kita dapat membatasi akses langsung ke data dan memerlukan pengguna untuk melalui metode yang ditentukan untuk mengakses atau mengubah data.
2. Keamanan: Untuk menjaga keamanan data dan mencegah perubahan yang tidak sah atau tidak terkontrol. Data yang terbuka dapat dengan mudah dimanipulasi, sementara encapsulation memungkinkan Kita menambahkan validasi dan kontrol akses.

3. Perubahan Internal: Ketika Kita ingin dapat mengubah implementasi internal dari sebuah class tanpa memengaruhi kode yang menggunakannya. Ini penting ketika Kita ingin melakukan perbaikan atau perubahan dalam class tanpa mempengaruhi fungsionalitas eksternal.
4. Validasi Data: Untuk melakukan validasi data sebelum mengizinkan perubahan. Dengan menggunakan metode setter, Kita dapat memeriksa apakah data yang dimasukkan sesuai dengan aturan bisnis atau batasan tertentu sebelum mengizinkan perubahan.
5. Dokumentasi yang Jelas: Untuk memberikan dokumentasi yang jelas tentang cara mengakses dan menggunakan class. Dengan metode publik sebagai antarmuka utama, pengguna class memiliki panduan yang jelas tentang cara berinteraksi dengan objek tersebut.
6. Pengelolaan Kompleksitas: Saat Kita ingin mengelompokkan data dan metode yang berhubungan dalam satu unit class. Ini membantu mengelola kompleksitas program dengan mengorganisasi dan mengemas konsep yang berbeda ke dalam unit yang terpisah.
7. Reusabilitas: Untuk meningkatkan reusabilitas kode. Dengan class yang terenkapsulasi dengan baik, Kita dapat dengan mudah menggunakan class yang sama dalam berbagai proyek tanpa perubahan besar.
8. Pengujian dan Debugging: Ketika Kita ingin memudahkan pengujian dan debugging. Dengan encapsulation, Kita dapat fokus pada unit individual (class) tanpa perlu mempertimbangkan implikasi dari unit lain.

Jadi, umumnya, Kita harus menggunakan encapsulation ketika Kita ingin meningkatkan kontrol, keamanan, dan keterbacaan kode Kita, serta ketika Kita ingin merancang class yang dapat digunakan kembali dan mudah dikelola dalam berbagai proyek.

Berikut adalah beberapa contoh situasi di mana Kita harus menggunakan encapsulation dalam Java:

1. Data Pribadi Pengguna:

Kita mengembangkan sistem manajemen pengguna, dan Kita memiliki data pengguna seperti nama, alamat email, dan kata sandi yang harus dilindungi. Kita ingin membatasi akses langsung ke data ini dan memastikan hanya metode yang tepat (seperti metode untuk login atau perubahan kata sandi) yang dapat mengaksesnya.

```
public class Pengguna {
    private String nama;
```

```
    private String email;
    private String kataSandi;

    // Metode getter dan setter untuk mengakses/mengubah data pengguna.
}
```

2. Keuangan dan Transaksi:

Kita mengembangkan aplikasi keuangan yang menangani transaksi bank. Data seperti saldo akun dan riwayat transaksi harus dilindungi dengan baik untuk menghindari kebocoran data atau perubahan yang tidak sah.

```
public class Rekening {
    private double saldo;
    private List<Transaksi> riwayatTransaksi;

    // Metode getter dan setter untuk mengakses/mengubah saldo dan riwayat transaksi.
}
```

3. Pengaturan Aplikasi:

Dalam aplikasi Kita, Kita memiliki pengaturan seperti suara, bahasa, dan tema yang harus tetap aman dan hanya dapat diakses melalui pengaturan yang disediakan.

```
public class PengaturanAplikasi {
    private String suara;
    private String bahasa;
    private String tema;

    // Metode getter dan setter untuk mengakses/mengubah pengaturan aplikasi.
}
```

4. Kelas Olahraga:

Kita memiliki class TimOlahraga yang berisi data seperti nama tim, skor, dan statistik. Data ini hanya harus diakses dan diubah melalui metode yang telah ditentukan.

```
public class TimOlahraga {
    private String namaTim;
    private int skor;
```

```
    private StatistikTim statistik;

    // Metode getter dan setter untuk mengakses/mengubah data tim olahraga.
}
```

Dalam semua contoh di atas, encapsulation membantu melindungi data sensitif, memastikan bahwa hanya metode yang tepat yang dapat mengakses data tersebut, dan memungkinkan pengguna class untuk berinteraksi dengan objek secara terkontrol melalui metode getter dan setter yang telah ditentukan.

```
 1. public class Pengguna {
 2.     private String nama;
 3.     private String email;
 4.     private String kataSandi;
 5.
 6.     // Constructor untuk inisialisasi objek Pengguna
 7.     public Pengguna(String nama, String email, String kataSandi) {
 8.         this.nama = nama;
 9.         this.email = email;
10.         this.kataSandi = kataSandi;
11.     }
12.
13.     // Metode getter untuk mendapatkan nama pengguna
14.     public String getNama() {
15.         return nama;
16.     }
17.
18.     // Metode setter untuk mengubah email pengguna
19.     public void setEmail(String email) {
20.         this.email = email;
21.     }
22.
23.     // Metode untuk mencetak informasi pengguna
24.     public void cetakInfoPengguna() {
25.         System.out.println("Nama: " + nama);
26.         System.out.println("Email: " + email);
27.         // Kata sandi tidak dicetak untuk alasan keamanan.
28.     }
29.
30.     public static void main(String[] args) {
31.         // Membuat objek Pengguna
```

```
        Pengguna pengguna1 = new Pengguna("John Doe", "john@example.com", "password123");

        // Mengakses dan mencetak informasi pengguna
        pengguna1.cetakInfoPengguna();

        // Mengubah email pengguna
        pengguna1.setEmail("john.doe@example.com");
        System.out.println("Email setelah diubah: " + pengguna1.getEmail());
    }
}

```

Dalam contoh ini, kita memiliki class Pengguna yang menggunakan encapsulation. Data-data pengguna seperti nama, email, dan kataSandi di-deklarasikan sebagai variabel privat. Kita kemudian menyediakan metode getter dan setter untuk mengakses dan mengubah data ini dengan cara yang terkontrol.

Pada main method, kita membuat objek Pengguna baru, mengakses dan mencetak informasi pengguna, dan juga mengubah email pengguna menggunakan metode setter yang telah ditentukan.

Dengan menggunakan encapsulation, kita dapat melindungi data sensitif seperti kata sandi dan memberikan cara yang terstruktur untuk berinteraksi dengan objek Pengguna.

```
public class Main {
    public static void main(String[] args) {
        // Membuat objek Pengguna
        Pengguna user1 = new Pengguna("John Doe", "john@example.com", "s3cr3t");

        // Mengakses data pengguna menggunakan metode getter
        System.out.println("Nama Pengguna: " + user1.getNama());
        System.out.println("Email Pengguna: " + user1.getEmail());

        // Perubahan kata sandi menggunakan metode setter
        user1.setKataSandi("new-password");

        // Mencetak informasi pengguna setelah perubahan
        System.out.println("Kata Sandi Pengguna: " + user1.getKataSandi());
    }
}

```

```
18. class Pengguna {
19.     private String nama;
20.     private String email;
21.     private String kataSandi;
22. 
23.     // Konstruktor
24.     public Pengguna(String nama, String email, String kataSandi) {
25.         this.nama = nama;
26.         this.email = email;
27.         this.kataSandi = kataSandi;
28.     }
29. 
30.     // Metode getter untuk nama
31.     public String getNama() {
32.         return nama;
33.     }
34. 
35.     // Metode getter untuk email
36.     public String getEmail() {
37.         return email;
38.     }
39. 
40.     // Metode getter untuk kata sandi
41.     public String getKataSandi() {
42.         return kataSandi;
43.     }
44. 
45.     // Metode setter untuk kata sandi
46.     public void setKataSandi(String kataSandi) {
47.         this.kataSandi = kataSandi;
48.     }
49. }
50. 
```

Dalam contoh di atas, kita memiliki class Pengguna yang menerapkan encapsulation. Data anggota nama, email, dan kataSandi adalah privat, yang berarti mereka hanya dapat diakses melalui metode getter dan setter yang telah ditentukan.

Ketika kita membuat objek Pengguna, kita menginisialisasinya dengan data pengguna. Kemudian, kita menggunakan metode getter untuk mengakses data pengguna, dan metode setter untuk mengubah kata sandi. Hal ini memastikan bahwa akses ke data pengguna dikontrol dan aman.

Hasil keluaran dari program di atas akan mencetak informasi pengguna sebelum dan setelah perubahan kata sandi.

```
 1. import java.util.ArrayList;
 2. import java.util.List;
 3.
 4. public class Main {
 5.     public static void main(String[] args) {
 6.         // Membuat objek Rekening
 7.         Rekening rekening = new Rekening("123456", 1000.0);
 8.
 9.         // Menambahkan beberapa transaksi
10.         rekening.tambahTransaksi("Deposit", 500.0);
11.         rekening.tambahTransaksi("Penarikan", -200.0);
12.
13.         // Mengakses saldo menggunakan metode getter
14.         System.out.println("Saldo saat ini: $" + rekening.getSaldo());
15.
16.         // Menampilkan riwayat transaksi
17.         List<Transaksi> riwayat = rekening.getRiwayatTransaksi();
18.         System.out.println("Riwayat Transaksi:");
19.         for (Transaksi transaksi : riwayat) {
20.             System.out.println(transaksi.getJenis() + ": $" + transaksi.getJumlah());
21.         }
22.     }
23. }
24.
25. class Rekening {
26.     private String nomorRekening;
27.     private double saldo;
28.     private List<Transaksi> riwayatTransaksi;
29.
30.     public Rekening(String nomorRekening, double saldoAwal) {
31.         this.nomorRekening = nomorRekening;
32.         this.saldo = saldoAwal;
33.         this.riwayatTransaksi = new ArrayList<>();
34.     }
35.
36.     public double getSaldo() {
37.         return saldo;
38.     }
39.
40.     public List<Transaksi> getRiwayatTransaksi() {
```

```
41.         return riwayatTransaksi;
42.     }
43.
44.     public void tambahTransaksi(String jenis, double jumlah) {
45.         Transaksi transaksi = new Transaksi(jenis, jumlah);
46.         riwayatTransaksi.add(transaksi);
47.         saldo += jumlah;
48.     }
49. }
50.
51. class Transaksi {
52.     private String jenis;
53.     private double jumlah;
54.
55.     public Transaksi(String jenis, double jumlah) {
56.         this.jenis = jenis;
57.         this.jumlah = jumlah;
58.     }
59.
60.     public String getJenis() {
61.         return jenis;
62.     }
63.
64.     public double getJumlah() {
65.         return jumlah;
66.     }
67. }
68.
```

Dalam contoh ini, class Rekening memiliki variabel privat nomorRekening, saldo, dan riwayatTransaksi, yang hanya dapat diakses melalui metode getter dan setter yang telah ditentukan. Metode tambahTransaksi digunakan untuk menambahkan transaksi ke riwayat dan mengupdate saldo.

Hasil keluaran program ini akan mencetak saldo saat ini dan riwayat transaksi yang telah ditambahkan. Encapsulation memungkinkan kita untuk memastikan bahwa data rekening hanya dapat diakses dan dimodifikasi melalui metode yang telah ditentukan, sehingga memastikan keamanan data dan kontrol akses yang lebih baik.

Latihan :

Soal 1: Penyembunyian Data

Kita diberikan program sederhana yang mencoba menerapkan encapsulation, tetapi ada kesalahan. Program ini mencoba merepresentasikan sebuah buku dengan judul dan harga. Identifikasi kesalahan dalam program berikut dan perbaiki:

```
1. public class Buku {
2.     private String judul;
3.     public double harga;
4.
5.     public Buku(String judul, double harga) {
6.         this.judul = judul;
7.         harga = harga;
8.     }
9.
10.    public String getJudul() {
11.        return judul;
12.    }
13.
14.    public double getHarga() {
15.        return harga;
16.    }
17. }
18.
19. public class Main {
20.    public static void main(String[] args) {
21.        Buku buku = new Buku("Java Programming", 45.0);
22.        System.out.println("Judul Buku: " + buku.getJudul());
23.        System.out.println("Harga Buku: " + buku.getHarga());
24.    }
25. }
26.
```

Soal 2: Manajemen Akun Pengguna

Kita memiliki class AkunPengguna yang digunakan untuk menyimpan informasi akun pengguna, termasuk nama pengguna dan kata sandi. Namun, ada masalah dalam program ini yang memungkinkan

kata sandi diakses secara langsung. Perbaiki program ini agar kata sandi hanya dapat diakses melalui metode getter dan setter.

```
1. public class AkunPengguna {
2.     private String namaPengguna;
3.     public String kataSandi;
4.
5.     public AkunPengguna(String namaPengguna, String kataSandi) {
6.         this.namaPengguna = namaPengguna;
7.         this.kataSandi = kataSandi;
8.     }
9.
10.    public String getNamaPengguna() {
11.        return namaPengguna;
12.    }
13.
14.    // Tambahkan metode getter dan setter untuk kataSandi
15. }
16.
17. public class Main {
18.    public static void main(String[] args) {
19.        AkunPengguna akun = new AkunPengguna("user123", "password123");
20.        System.out.println("Nama Pengguna: " + akun.getNamaPengguna());
21.        System.out.println("Kata Sandi: " + akun.kataSandi); // Perbaiki ini
22.    }
23. }
24.
```

Soal 3: Data Mahasiswa

Kita memiliki class Mahasiswa yang digunakan untuk menyimpan data mahasiswa, termasuk nama dan nim. Namun, program ini mengizinkan perubahan langsung pada nim. Perbaiki program ini agar nim hanya dapat diakses melalui metode getter dan setter.

```
1. public class Mahasiswa {
2.     private String nama;
3.     public int nim;
4.
5.     public Mahasiswa(String nama, int nim) {
6.         this.nama = nama;
7.         this.nim = nim;
8.     }
```

```
 9.
10.     public String getNama() {
11.         return nama;
12.     }
13.
14.     // Tambahkan metode getter dan setter untuk nim
15. }
16.
17. public class Main {
18.     public static void main(String[] args) {
19.         Mahasiswa mahasiswa = new Mahasiswa("John Doe", 123456);
20.         System.out.println("Nama Mahasiswa: " + mahasiswa.getNama());
21.         System.out.println("NIM: " + mahasiswa.nim); // Perbaiki ini
22.     }
23. }
24.
```

Setiap soal di atas memiliki kesalahan terkait dengan encapsulation yang harus diperbaiki oleh siswa agar program berfungsi dengan benar sesuai dengan prinsip-prinsip encapsulation.

Tugas Rumah :

Pada soal dibawah ini implementasikan encapsulation dengan benar dalam kelas yang diberikan, menggunakan metode getter dan setter untuk mengakses dan mengubah atribut privat, dan melakukan operasi yang sesuai pada objek-objek yang dibuat.

1. **Kelas Produk**

   Kita diberikan kelas Produk yang digunakan untuk merepresentasikan produk dalam sebuah toko. Kelas ini memiliki atribut privat namaProduk dan hargaProduk. Buatlah kelas ini dengan metode getter dan setter yang sesuai, lalu gunakan kelas tersebut dalam program utama untuk membuat dan menampilkan informasi produk.

2. **Pengaturan Lampu**

   Kita sedang mengembangkan sistem pengaturan lampu yang dapat dihidupkan dan dimatikan. Buatlah sebuah kelas Lampu yang memiliki atribut privat nyala (boolean). Tambahkan metode getter dan setter untuk mengakses dan mengubah status lampu ini. Kemudian, dalam program utama, buat objek lampu dan uji kemampuan metode getter dan setter.

3. Simpanan Nasabah

Kita bekerja untuk sebuah bank dan perlu mengelola informasi simpanan nasabah. Buatlah kelas SimpananNasabah dengan atribut privat saldo dan metode getter dan setter yang sesuai. Kemudian, dalam program utama, buat beberapa objek simpanan nasabah, lakukan beberapa transaksi, dan tampilkan saldo akhir mereka.

4. Data Produk

Kita sedang mengembangkan sistem untuk mengelola data produk di sebuah perusahaan. Buatlah kelas DataProduk yang memiliki atribut privat namaProduk (String), hargaProduk (double), dan stok (int). Tambahkan metode getter dan setter untuk mengakses dan mengubah data produk ini. Kemudian, dalam program utama, buat objek-objek produk dan lakukan operasi-operasi yang sesuai pada mereka.

5. Informasi Mahasiswa

Kita ingin mengelola informasi mahasiswa di sebuah universitas. Buatlah kelas InformasiMahasiswa dengan atribut privat nama (String), nim (String), dan ipk (double). Tambahkan metode getter dan setter untuk mengakses dan mengubah data mahasiswa ini. Dalam program utama, buat beberapa objek mahasiswa dan tampilkan informasi mereka.