**Catetan**

import java.util.Scanner;

class Registrasi {

private static String username, password;

public static void registrasiAkun() {

Scanner input = new Scanner(System.in);

System.out.println("== DAFTAR AKUN ==");

System.out.print("Username: ");

username = input.nextLine();

System.out.print("Password: ");

password = input.nextLine();

System.out.println("Akun Anda berhasil didaftarkan.");

}

public static String getUsername() {

return username;

}

public static String getPassword() {

return password;

}

}

class Login {

private static String username, password;

public static boolean loginAkun() {

Scanner input = new Scanner(System.in);

System.out.println("== LOGIN ==");

System.out.print("Username: ");

username = input.nextLine();

System.out.print("Password: ");

password = input.nextLine();

if (username.equals(Registrasi.getUsername()) && password.equals(Registrasi.getPassword())) {

System.out.println("Login berhasil.");

return true;

} else {

System.out.println("Username atau password salah.");

return false;

}

}

public static String getUsername() {

return username;

}

}

class Logout {

public static boolean logoutAkun() {

System.out.println("Anda berhasil logout.");

return false;

}

}

abstract class Email {

public abstract void tampilkan();

}

class Kotak\_Masuk extends Email {

@Override

public void tampilkan() {

System.out.println("Anda masuk ke Kotak Masuk.");

}

}

class Terkirim extends Email {

@Override

public void tampilkan() {

System.out.println("Anda masuk ke Kotak Terkirim.");

}

}

class Draf extends Email {

@Override

public void tampilkan() {

System.out.println("Anda masuk ke Draf.");

}

}

class Buat extends Email {

@Override

public void tampilkan() {

System.out.println("Anda membuat email baru.");

}

}

class Main\_Menu {

public static void tampilkan(String username) {

Scanner input = new Scanner(System.in);

int Menupilihan;

do {

System.out.println("\nSelamat Datang di Gmail, " + username);

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

System.out.println("== Silahkan Memilih Menu ==");

System.out.println("1. Kotak Masuk");

System.out.println("2. Terkirim");

System.out.println("3. Draf");

System.out.println("4. Buat");

System.out.println("5. Logout");

System.out.println();

System.out.print("Pilihan: ");

Menupilihan = input.nextInt();

switch (Menupilihan) {

case 1:

Email kotakmasuk = new Kotak\_Masuk();

kotakmasuk.tampilkan();

break;

case 2:

Email terkirim = new Terkirim();

terkirim.tampilkan();

break;

case 3:

Email draf = new Draf();

draf.tampilkan();

break;

case 4:

Email buat = new Buat();

buat.tampilkan();

break;

case 5:

Logout.logoutAkun();

break;

default:

System.out.println("Pilihan tidak valid.");

break;

}

} while (Menupilihan != 5);

}

}

public class MainGmail {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

int pilihanMenu;

do {

System.out.println("\nSelamat Datang Di Gmail");

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

System.out.println("== Silahkan Memilih Menu ==");

System.out.println("1. Daftar Akun");

System.out.println("2. Login");

System.out.println("3. Keluar");

System.out.println();

System.out.print("Pilihan: ");

pilihanMenu = input.nextInt();

switch (pilihanMenu) {

case 1:

Registrasi.registrasiAkun();

break;

case 2:

if (Login.loginAkun()) {

Main\_Menu.tampilkan(Login.getUsername());

}

break;

case 3:

System.out.println("Terima kasih telah menggunakan Gmail.");

break;

default:

System.out.println("Pilihan tidak valid.");

break;

}

} while (pilihanMenu != 3);

}

}

Ya, program di atas sudah menerapkan prinsip OOP (Object-Oriented Programming). Beberapa prinsip OOP yang terlihat pada program tersebut antara lain:

1. Encapsulation:
   * Kelas-kelas seperti Registrasi, Login, dan Logout mempunyai atribut dan method yang diakses hanya oleh kelas itu sendiri atau oleh kelas lain yang mempunyai hak akses tertentu (private).
   * Kelas-kelas seperti Inbox, Sent, Drafts, dan Compose merupakan kelas turunan dari kelas Email, yang menunjukkan bahwa kelas-kelas tersebut mempunyai atribut dan method yang sama dengan kelas Email.
   * Dalam kelas Registrasi dan Login, terdapat variabel private yang diakses melalui method publik. Ini menerapkan konsep encapsulation, yaitu mengelompokkan data bersama-sama dan melindungi mereka dari akses dari luar.
2. Inheritance:
   * Kelas Inbox, Sent, Drafts, dan Compose mewarisi sifat-sifat (atribut dan method) dari kelas Email.
   * Kelas Email, Kotak\_Masuk, Terkirim, Draf, dan Buat menerapkan konsep inheritance, di mana kelas-kelas tersebut mewarisi sifat-sifat kelas Email.
3. Polymorphism:
   * Kelas Inbox, Sent, Drafts, dan Compose mengimplementasikan method abstrak yang didefinisikan di kelas Email dengan cara yang berbeda-beda (polymorphism).
   * Pada kelas MainScreen, objek dari kelas-kelas Inbox, Sent, Drafts, dan Compose diinisialisasi dengan tipe Email, sehingga objek-objek tersebut dapat dipanggil melalui variabel yang sama (polymorphism).
   * Kelas Email, Kotak\_Masuk, Terkirim, Draf, dan Buat menerapkan konsep polymorphism, di mana kelas-kelas tersebut memiliki method yang sama namun menghasilkan output yang berbeda-beda.
4. Abstraction:
   * Kelas Email merupakan kelas abstrak yang tidak bisa diinstansiasi secara langsung, hanya bisa diwarisi oleh kelas turunannya.
   * Kelas MainScreen merupakan kelas yang menggunakan abstraksi pada objek-objek dari kelas Inbox, Sent, Drafts, dan Compose.
   * Kelas Email menggunakan konsep abstraction, di mana kelas Email tidak memiliki implementasi method tampilkan() yang merupakan method abstrak. Kelas-kelas yang mewarisi sifat Email, seperti Kotak\_Masuk, Terkirim, dan Draf, harus mengimplementasikan method tersebut.
5. Static :

* Kelas Registrasi, Login, dan Logout menggunakan keyword static pada beberapa variabel dan method. Ini mengindikasikan bahwa variabel dan method tersebut terkait dengan kelas dan bukan dengan objek, sehingga dapat diakses dari kelas lain secara langsung tanpa membuat objek terlebih dahulu.

1. Modularity:
   * Program di atas terdiri dari beberapa kelas yang memiliki fungsi masing-masing, sehingga dapat memudahkan dalam pengembangan dan perawatan kode.

Kelas-kelas yang ada pada program ini adalah:

* Registrasi: Kelas ini berfungsi untuk melakukan registrasi akun baru. Pada kelas ini terdapat dua atribut, yaitu username dan password, dan terdapat tiga metode, yaitu registrasiAkun(), getUsername(), dan getPassword().
* Login: Kelas ini berfungsi untuk melakukan login. Pada kelas ini terdapat dua atribut, yaitu username dan password, dan terdapat satu metode, yaitu loginAkun().
* Logout: Kelas ini berfungsi untuk melakukan logout. Pada kelas ini terdapat satu metode, yaitu logoutAkun().
* Email: Kelas ini adalah kelas abstrak yang menjadi parent class dari kelas Kotak\_Masuk, Terkirim, Draf, dan Buat. Kelas ini memiliki satu metode abstrak, yaitu tampilkan().
* Kotak\_Masuk, Terkirim, Draf, dan Buat: Kelas-kelas ini merupakan child class dari kelas Email. Setiap kelas memiliki satu metode tampilkan() yang mengoverride metode tampilkan() pada kelas Email.
* MainScreen: Kelas ini berfungsi untuk menampilkan menu utama setelah user berhasil login. Pada kelas ini terdapat satu metode tampilkan() yang menerima parameter username.
* MainGmail: Kelas ini adalah kelas utama yang akan dieksekusi ketika program dijalankan. Pada kelas ini terdapat tiga metode, yaitu main(), yang merupakan metode utama, dan terdapat dua metode lainnya, yaitu registrasiAkun() dan loginAkun(), yang dipanggil dari kelas Registrasi dan Login.

Dalam program ini, konsep OOP diimplementasikan dengan menggunakan inheritance (pewarisan), encapsulation (pembungkusan), dan polymorphism (polimorfisme). Pewarisan digunakan untuk membuat kelas-kelas child yang memiliki atribut dan metode yang sama dengan kelas parent-nya. Pembungkusan digunakan untuk menyembunyikan detail implementasi dari sebuah kelas dan hanya menampilkan metode yang penting saja. Polimorfisme digunakan untuk membuat beberapa kelas child yang berbeda dapat menggunakan metode dengan nama yang sama, tetapi dengan implementasi yang berbeda.

* **fungsi override dalam program di atas**

Pada program di atas, terdapat beberapa penggunaan fungsi override pada beberapa kelas turunan dari kelas abstrak Email. Fungsi override digunakan untuk menimpa implementasi metode yang didefinisikan di kelas induk dengan implementasi baru yang didefinisikan di kelas turunan.

Contohnya, pada kelas Kotak\_Masuk, fungsi tampilkan() ditimpa dengan implementasi baru yang mencetak pesan "Anda masuk ke Kotak Masuk", pada kelas Terkirim, fungsi tampilkan() ditimpa dengan implementasi baru yang mencetak pesan "Anda masuk ke Kotak Terkirim", dan pada kelas Draf, fungsi tampilkan() ditimpa dengan implementasi baru yang mencetak pesan "Anda masuk ke Draf". Hal ini memungkinkan penggunaan metode tampilkan() yang sama pada semua kelas turunan, namun dapat menghasilkan keluaran yang berbeda-beda tergantung dari implementasi yang ditimpa.

Selain itu, pada kelas Buat, juga terdapat fungsi override, tetapi kali ini tidak menimpa metode yang ada di kelas induk. Sebagai gantinya, kelas Buat mengimplementasikan metode tampilkan() sendiri yang berbeda dengan metode tampilkan() di kelas-kelas turunan lainnya. Dalam hal ini, metode tampilkan() pada kelas Buat mencetak pesan "Anda membuat email baru".

* **Void**

Fungsi void digunakan pada program tersebut untuk menunjukkan bahwa suatu method tidak mengembalikan nilai apapun. Contohnya, pada class Registrasi, method registrasiAkun() dan pada class Logout, method logoutAkun() menggunakan tipe data void pada deklarasi methodnya.

Pada method **registrasiAkun()**, setelah proses pendaftaran selesai, tidak ada nilai kembalian apapun yang diberikan. Sedangkan pada method **logoutAkun()**, setelah proses logout selesai, dikembalikan nilai boolean **false**, namun hal ini tidak berpengaruh pada tipe data method karena nilai tersebut hanya digunakan untuk operasi tertentu dalam program

* **Kontructor .**

menginisialisasi atau memberikan nilai (melalui parameter) kepada sebuah objek pada saat objek pertama kali diciptakan

* **List:** List adalah antarmuka (interface) dalam banyak bahasa pemrograman yang menyediakan operasi dasar untuk bekerja dengan kumpulan elemen. List memungkinkan kita untuk menambah, menghapus, dan mengakses elemen-elemen secara berurutan. Implementasi konkret dari antarmuka List dapat berbeda-beda, seperti ArrayList, LinkedList, Vector, dan sebagainya. List umumnya memiliki sifat kapasitas dinamis, artinya ukuran list dapat berubah saat elemen-elemen ditambah atau dihapus.
* **ArrayList:** ArrayList adalah salah satu implementasi konkret dari antarmuka List. ArrayList menggunakan array sebagai dasar penyimpanan elemen-elemennya. Dalam ArrayList, elemen-elemen ditempatkan dalam urutan kontigu di dalam memori. ArrayList memiliki kapasitas dinamis, yang berarti ukuran ArrayList dapat berubah secara otomatis saat elemen-elemen ditambah atau dihapus. ArrayList juga menyediakan metode untuk memanipulasi elemen-elemen, seperti menambahkan, menghapus, dan mengakses elemen berdasarkan indeksnya.

Dalam banyak bahasa pemrograman, seperti Java, Python, dan C#, ArrayList merupakan struktur data yang sering digunakan karena kemudahannya dalam penggunaan dan kinerja yang baik untuk akses elemen secara acak. Namun, tergantung pada kebutuhan dan konteks aplikasi, pilihan antara List dan ArrayList dapat berbeda.

Top of Form