## Soal Praktikum

1. Jelaskan apa bedanya Abstraction dan Interfaces! [wajib] kamu ketahui!

## Jawaban

Abstract : bisa berisi abstract dan non abstract method, kita harus menuliskan sendiri modifiernya, bisa mendeklarasikan constant dan instance variable

Interface : hanya boleh berisi abstract method, hanya bisa mendeklarsikan constant, method tidak boleh bersifat statis

## Soal Praktikum

1. Jelaskan apa yang dimaksud Exception Handling! [wajib]

## Jawaban

## Exception merupakan permasalahan yang terjadi Ketika kode program. Pada saat terjadi sebuah kesalahan pada kode program kita, alur jalannya program akan terganggu danbisa terjadi salah satunya user telah memasukkan nilai yang salah.

## Soal Praktikum

1. Budi Arya membuat sedang belajar tentang abstraction dan exception handling

Ia membuat program sebagai berikut :

## Source Code

*abstract class* Account {  
 String nomorakun;  
 *double* balance;  
  
 Account(String nomorakun, *double* balance) {  
 *this*.nomorakun = nomorakun;  
 *this*.balance = balance;  
 }  
  
 *abstract void* deposit(*int* amount);  
 *abstract void* withdraw(*double* amount) *throws* InsufficientFundsException;  
  
 *void* displayBalance() {  
 System.out.println("Account Number: " + nomorakun + ", Balance: " + balance);  
 }}  
*class* SavingsAccount *extends* Account {  
 *double* interestRate;  
  
 SavingsAccount(String nomorakun, *double* balance, *double* interestRate) {  
 *super*(nomorakun, balance);  
 *this*.interestRate = interestRate;  
 }  
  
 *@Override  
 void* deposit(*int* amount) {  
 *if* (amount <= 0) {  
 *throw new* IllegalArgumentException("mana ada orang deposit -.");  
 }  
 balance += amount;  
 System.out.println("Deposited: " + amount + ", New Balance: " + balance);  
 }  
  
 *@Override  
 void* withdraw(*double* amount) *throws* InsufficientFundsException { *// Menggunakan exception sesuai kebutuhan  
 if* (amount > balance) {  
 *throw new* InsufficientFundsException("Saldo anda tidak cukup!!!!!. Insufficient funds for withdrawal.!!!!!");  
 }  
 balance -= amount;  
 System.out.println("Withdrew: " + amount + ", New Balance: " + balance);  
 }  
  
 *void* addInterest() {  
 balance += balance \* interestRate;  
 System.out.println("Interest Added: " + (balance \* interestRate) + ", New Balance: " + balance);  
 }  
}  
  
*class* InsufficientFundsException *extends* Exception {  
 InsufficientFundsException (String message) {  
 *super*(message);}  
}  
*public class* Main {  
 *public static void* main(String[] args) {  
 *try* {  
 System.out.println("Selamat Datang Nasabah BTK");  
 SavingsAccount savings = *new* SavingsAccount("123456789", 1000.0, 0.05);  
 savings.displayBalance();  
 savings.deposit(500);  
 savings.addInterest();  
 savings.withdraw(2000);  
 } *catch* (IllegalArgumentException e) {  
 System.out.println("Error: " + e.getMessage());  
 } *catch* (InsufficientFundsException e) {  
 System.out.println("Error: " + e.getMessage());  
 System.out.println("Maaf saldo gasesuai silahkan bekerja keras lagi");  
 } *finally* {  
 System.out.println("Terima Kasih telah menjadi nasabah dari bank BTK.");  
 System.out.println("Thank you for being a customer of Bank BTK.");  
 }  
 }  
}

Tulis Penjelasan disini …

Program ini merupakan implementasi dari konsep \*\*Object-Oriented Programming (OOP)\*\* di Java yang mencakup penggunaan kelas abstrak, pewarisan, dan penanganan \*\*exception\*\*. Di dalam program ini, ada kelas abstrak `Account` yang berfungsi sebagai template untuk kelas-kelas lain. Kelas ini memiliki dua atribut, yaitu `nomorakun` (untuk menyimpan nomor akun) dan `balance` (untuk menyimpan saldo akun). Selain itu, terdapat dua metode abstrak: `deposit(int amount)` untuk melakukan setoran dan `withdraw(double amount)` untuk melakukan penarikan uang, yang masing-masing harus diimplementasikan oleh kelas turunan. Salah satu kelas turunan dari `Account` adalah `SavingsAccount`, yang mewarisi semua sifat dari `Account` dan menambahkan fungsionalitas untuk menambahkan bunga (`addInterest`) serta menangani setoran dan penarikan. Kelas `SavingsAccount` juga menangani pengecekan saldo yang cukup sebelum melakukan penarikan dan memberikan error jika saldo tidak mencukupi dengan menggunakan \*\*exception\*\* `InsufficientFundsException`. Pada bagian utama program (`Main`), terdapat contoh penggunaan objek `SavingsAccount` untuk mendemonstrasikan proses setoran, penarikan, dan penambahan bunga, serta penanganan exception yang bisa terjadi seperti kesalahan saat melakukan deposit dengan jumlah negatif atau penarikan dengan saldo tidak mencukupi. Program ini menggunakan blok `try-catch` untuk menangani error dan memberikan pesan yang sesuai kepada pengguna, serta diakhiri dengan blok `finally` yang memastikan pesan ucapan terima kasih selalu muncul meskipun terjadi error.

**Output**

|  |
| --- |
|  |

## Soal Praktikum

1. Dari soal praktikum sebelumnya kalian telah membuat class User yang di

extend atau diturunkan ke kelas Customer dan Driver.Buatlah interface

Pembayaran dengan metode prosesPembayaran(double jumlah).

Kemudian buatlah class PembayaranTunai dan PembayaranKartu yang

mengimplementasikan interface Pembayaran,dengan syarat sebagai

berikut:

• Pada class PembayaranKartu, tambahkan atribut nomorkartu dan

namapemilik.

• Tambahkan metode bayar pada class Customer yang menerima

objek Pembayaran sebagai parameter dan memanggil metode

prosesPembayaran.

• Buatlah objek PembayaranTunai dan PembayaranKartu, lalu

gunakan metode bayar pada objek Customer untuk melakukan

pembayaran.

## Source Code

*package* soal4;  
  
*import* java.util.Scanner;  
  
*class* user {  
 String nama;  
 *int* no\_id;  
  
 user(String nama, *int* no\_id) {  
 *this*.nama = nama;  
 *this*.no\_id = no\_id;  
 }  
  
 *public void* tampilan() {  
 System.out.println("Nama : " + nama + "\nID : " + no\_id);  
 }  
}  
  
*class* customer *extends* user {  
 String email;  
  
 customer(String nama, *int* no\_id, String email) {  
 *super*(nama, no\_id);  
 *this*.email = email;  
 }  
  
 *@Override  
 public void* tampilan() {  
 System.out.println("Customer: ");  
 *super*.tampilan();  
 System.out.println("Email : " + email);  
 }  
  
 *public void* bayar(Pembayaran pembayaran, *double* jumlah) {  
 System.out.println(nama + " sedang melakukan pembayaran");  
 pembayaran.prosesPembayaran(jumlah);  
 }  
}  
  
*interface* Pembayaran {  
 *void* prosesPembayaran(*double* jumlah);  
}  
  
*class* PembayaranTunai *implements* Pembayaran {  
 *@Override  
 public void* prosesPembayaran(*double* jumlah) {  
 System.out.println("Pembayaran sebesar Rp" + jumlah + " dilakukan menggunakan tunai.");  
 }  
}  
  
*class* PembayaranKartu *implements* Pembayaran {  
 String nomorKartu;  
 String namaPemilik;  
  
 PembayaranKartu(String nomorKartu, String namaPemilik) {  
 *this*.nomorKartu = nomorKartu;  
 *this*.namaPemilik = namaPemilik;  
 }  
  
 *@Override  
 public void* prosesPembayaran(*double* jumlah) {  
 System.out.println("Pembayaran sebesar Rp" + jumlah + " dilakukan menggunakan kartu.");  
 System.out.println("Nomor Kartu : " + nomorKartu + ", Nama Pemilik : " + namaPemilik);  
 }  
}  
  
*// Main class  
public class* main {  
 *public static void* main(String[] args) {  
 Scanner scanner = *new* Scanner(System.in);  
  
  
 customer cs = *new* customer("Adam", 1231, "Adaminsaan24@gmail.com");  
  
  
 cs.tampilan();  
 System.out.println();  
  
  
 System.out.print("Masukkan jumlah pembayaran: ");  
 *double* jumlah = scanner.nextDouble();  
 scanner.nextLine();  
  
  
 System.out.println("Pilih metode pembayaran:");  
 System.out.println("1. Tunai");  
 System.out.println("2. Kartu");  
 System.out.print("Pilihan Anda: ");  
 *int* pilihan = scanner.nextInt();  
 scanner.nextLine();  
  
 Pembayaran pembayaran = *null*;  
  
 *if* (pilihan == 1) {  
 pembayaran = *new* PembayaranTunai();  
 } *else if* (pilihan == 2) {  
  
 System.out.print("Masukkan nomor kartu: ");  
 String nomorKartu = scanner.nextLine();  
 System.out.print("Masukkan nama pemilik kartu: ");  
 String namaPemilik = scanner.nextLine();  
 pembayaran = *new* PembayaranKartu(nomorKartu, namaPemilik);  
 } *else* {  
 System.out.println("Pilihan tidak valid.");  
 *return*;  
 }  
  
  
 cs.bayar(pembayaran, jumlah);  
 }  
}

**Penjelasan**

Program ini menggambarkan implementasi konsep \*\*Object-Oriented Programming (OOP)\*\* dalam Java dengan menggunakan kelas, pewarisan, dan antarmuka (interface). Dalam program ini, terdapat sebuah kelas dasar bernama `user` yang menyimpan data dasar pengguna, yaitu `nama` dan `no\_id`, dengan metode `tampilan()` untuk menampilkan informasi tersebut. Kelas `customer` merupakan turunan dari kelas `user` dan menambahkan atribut `email` serta mengoverride metode `tampilan()` untuk menampilkan informasi yang lebih lengkap, termasuk email. Selain itu, kelas `customer` juga memiliki metode `bayar()` yang memungkinkan objek customer untuk melakukan pembayaran menggunakan salah satu metode pembayaran yang ditentukan.

Pembayaran dapat dilakukan dengan dua cara: menggunakan uang tunai atau kartu. Untuk ini, terdapat sebuah \*\*interface\*\* bernama `Pembayaran` yang memiliki metode `prosesPembayaran()`. Ada dua implementasi dari interface ini: `PembayaranTunai`, yang melakukan pembayaran secara tunai, dan `PembayaranKartu`, yang memproses pembayaran menggunakan kartu kredit atau debit dan mencetak detail kartu serta pemiliknya. Dalam metode `main()`, pertama-tama program meminta input informasi pelanggan dan menampilkan data tersebut, lalu meminta pengguna untuk memasukkan jumlah pembayaran dan memilih metode pembayaran. Berdasarkan pilihan yang diberikan, objek `Pembayaran` yang sesuai (baik `PembayaranTunai` atau `PembayaranKartu`) akan dibuat, dan metode `bayar()` milik objek `customer` akan dipanggil untuk memproses pembayaran sesuai dengan jumlah yang dimasukkan. Program ini mengilustrasikan bagaimana objek dapat berinteraksi melalui pewarisan dan interface untuk mencapai fungsionalitas yang diinginkan.

**Output**

|  |
| --- |
|  |

## Soal Praktikum

1. Berdasarkan soal No.4 Buatlah interface ValidasiPembayaran dengan

metode validasi (double jumlah). Buatlah class ValidasiSaldo dan

ValidasiKartu

yang

mengimplementasikan

interface

ValidasiPembayaran, dengan ketentuan sebagai berikut:

• Pada class ValidasiSaldo, tambahkan atribut saldo dan metode

validasi untuk memeriksa apakah saldo mencukupi.

• Pada class ValidasiKartu, tambahkan atribut nomorkartu dan

metode validasi untuk memeriksa validnya nomor kartu.

• Tambahkan metode validasiPembayaran pada class Customer

yang menerima objek ValidasiPembayaran sebagai parameter dan

memanggil metode validasi.

• Buat objek ValidasiSaldo dan ValidasiKartu, lalu gunakan metode

validasiPembayaran pada objek Customer untuk melakukan validasi

sebelum pembayaran.

## Jawaban

Ketik jawaban disini …

## Source Code

*package* soal5;  
  
*import* java.util.Scanner;  
  
*class* user {  
 String nama;  
 *int* no\_id;  
  
 user(String nama, *int* no\_id) {  
 *this*.nama = nama;  
 *this*.no\_id = no\_id;  
 }  
  
 *public void* tampilan() {  
 System.out.println("Nama : " + nama + "\nID : " + no\_id);  
 }  
}  
  
*class* customer *extends* user {  
 String email;  
  
 customer(String nama, *int* no\_id, String email) {  
 *super*(nama, no\_id);  
 *this*.email = email;  
 }  
  
 *@Override  
 public void* tampilan() {  
 System.out.println("Customer: ");  
 *super*.tampilan();  
 System.out.println("Email : " + email);  
 }  
  
 *public void* bayar(Pembayaran pembayaran, *double* jumlah) {  
 System.out.println(nama + " sedang melakukan pembayaran...");  
 pembayaran.prosesPembayaran(jumlah);  
 }  
  
 *public void* validasiPembayaran(ValidasiPembayaran validasi, *double* jumlah) {  
 *if* (validasi.validasi(jumlah)) {  
 System.out.println("Validasi berhasil, pembayaran dapat dilakukan.");  
 } *else* {  
 System.out.println("Validasi gagal, pembayaran tidak dapat dilakukan.");  
 }  
 }  
}  
  
*interface* Pembayaran {  
 *void* prosesPembayaran(*double* jumlah);  
}  
  
*class* PembayaranTunai *implements* Pembayaran {  
 *@Override  
 public void* prosesPembayaran(*double* jumlah) {  
 System.out.println("Pembayaran sebesar Rp" + jumlah + " dilakukan menggunakan tunai.");  
 }  
}  
  
*class* PembayaranKartu *implements* Pembayaran {  
 String nomorKartu;  
 String namaPemilik;  
  
 PembayaranKartu(String nomorKartu, String namaPemilik) {  
 *this*.nomorKartu = nomorKartu;  
 *this*.namaPemilik = namaPemilik;  
 }  
  
 *@Override  
 public void* prosesPembayaran(*double* jumlah) {  
 System.out.println("Pembayaran sebesar Rp" + jumlah + " dilakukan menggunakan kartu.");  
 System.out.println("Nomor Kartu : " + nomorKartu + ", Nama Pemilik : " + namaPemilik);  
 }  
}  
  
*interface* ValidasiPembayaran {  
 *boolean* validasi(*double* jumlah);  
}  
  
*class* ValidasiSaldo *implements* ValidasiPembayaran {  
 *double* saldo;  
  
 ValidasiSaldo(*double* saldo) {  
 *this*.saldo = saldo;  
 }  
  
 *@Override  
 public boolean* validasi(*double* jumlah) {  
 *if* (jumlah <= saldo) {  
 *return true*;  
 } *else* {  
 System.out.println("Saldo tidak mencukupi. Saldo saat ini: Rp" + saldo);  
 *return false*;  
 }  
 }  
}  
  
*class* ValidasiKartu *implements* ValidasiPembayaran {  
 String nomorKartu;  
  
 ValidasiKartu(String nomorKartu) {  
 *this*.nomorKartu = nomorKartu;  
 }  
  
 *@Override  
 public boolean* validasi(*double* jumlah) {  
 *if* (nomorKartu.matches("\\d{4}-\\d{4}")) {  
 *return true*;  
 } *else* {  
 System.out.println("Nomor kartu tidak valid: " + nomorKartu);  
 *return false*;  
 }  
 }  
}  
  
*// Main class  
public class* main {  
 *public static void* main(String[] args) {  
 Scanner scanner = *new* Scanner(System.in);  
  
 customer cs = *new* customer("Adam", 1231, "Adaminsaan24@gmail.com");  
  
 cs.tampilan();  
 System.out.println();  
  
  
 System.out.print("Masukkan jumlah pembayaran: ");  
 *double* jumlah = scanner.nextDouble();  
 scanner.nextLine();  
  
 System.out.println("Pilih metode pembayaran:");  
 System.out.println("1. Tunai");  
 System.out.println("2. Kartu");  
 System.out.print("Pilihan Anda: ");  
 *int* pilihan = scanner.nextInt();  
 scanner.nextLine();  
  
 Pembayaran pembayaran = *null*;  
 ValidasiPembayaran validasi = *null*;  
  
 *if* (pilihan == 1) {  
 pembayaran = *new* PembayaranTunai();  
  
 System.out.print("Masukkan saldo Anda: ");  
 *double* saldo = scanner.nextDouble();  
 validasi = *new* ValidasiSaldo(saldo);  
  
 } *else if* (pilihan == 2) {  
 System.out.print("Masukkan nomor kartu, contoh: xxxx-xxxx : ");  
 String nomorKartu = scanner.nextLine();  
 System.out.print("Masukkan nama pemilik kartu: ");  
 String namaPemilik = scanner.nextLine();  
 pembayaran = *new* PembayaranKartu(nomorKartu, namaPemilik);  
  
 validasi = *new* ValidasiKartu(nomorKartu);  
  
 } *else* {  
 System.out.println("Pilihan tidak valid.");  
 *return*;  
 }  
  
 cs.validasiPembayaran(validasi, jumlah);  
  
 *if* (validasi.validasi(jumlah)) {  
 cs.bayar(pembayaran, jumlah);  
 }  
 }  
}

**Penjelasan**

Program ini mengimplementasikan konsep \*\*Object-Oriented Programming (OOP)\*\* di Java dengan menggunakan kelas abstrak, pewarisan, dan penanganan exception. Di dalam program ini, terdapat kelas abstrak `Account` yang mendefinisikan struktur dasar untuk akun bank, dengan dua atribut: `nomorakun` untuk nomor akun dan `balance` untuk saldo akun. Kelas ini juga memiliki dua metode abstrak, yaitu `deposit(int amount)` untuk melakukan setoran dan `withdraw(double amount)` untuk melakukan penarikan, yang akan diimplementasikan oleh kelas turunan. Kelas `SavingsAccount` merupakan turunan dari `Account` yang menambahkan atribut `interestRate` (tingkat bunga) serta mengimplementasikan metode `deposit` dan `withdraw`. Metode `deposit` memeriksa apakah jumlah yang disetorkan positif, sedangkan metode `withdraw` memeriksa apakah saldo cukup untuk melakukan penarikan. Jika saldo tidak mencukupi, sebuah exception `InsufficientFundsException` akan dilemparkan. Selain itu, kelas `SavingsAccount` juga memiliki metode `addInterest()` yang menambahkan bunga berdasarkan saldo dan tingkat bunga yang ditentukan.

Di dalam kelas `Main`, program dimulai dengan membuat objek `SavingsAccount` dan memanipulasi saldo akun dengan cara melakukan setoran, menambahkan bunga, dan mencoba melakukan penarikan. Penanganan exception dilakukan dengan blok `try-catch` untuk menangani kesalahan seperti setoran dengan jumlah negatif (menggunakan `IllegalArgumentException`) dan penarikan yang melebihi saldo (menggunakan `InsufficientFundsException`). Program ini juga menggunakan blok `finally` untuk memastikan bahwa pesan ucapan terima kasih tetap ditampilkan, terlepas dari apakah terjadi error atau tidak. Program ini mengilustrasikan penggunaan konsep OOP seperti inheritance, exception handling, dan encapsulation untuk menyelesaikan kasus perbankan sederhana.

**Output**

|  |
| --- |
|  |

## Soal Praktikum

Ketik soal disini …

## Jawaban

Ketik jawaban disini …

## Source Code

Tulis kode program dikotak ini...

1 kotak dan 1 Penjelasan untuk 1 Class

**Penjelasan**

Tulis Penjelasan disini …

**Output**

|  |
| --- |
| Masukan screenshot output disini |