LAPORAN PRAKTIKUM

PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
PERIODE X

Nama	:	Michael Agung
NPM	:	06.2023.1.07651
Pertemuan	:	4

SOAL PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
PERIODE X

Laboratorium Rekayasa Perangkat Lunak, ITATS

PERTEMUAN 4 RPL-MF8T9-A

MEKANISME PRAKTIKUM

1) Buat Sebuah **Project Java Baru** pada IntelliJ IDEA dengan nama Project:
   "*PertemuanX_NPM AKHIR*"
   Ganti "X" menjadi Pertemuan yang sedang berlangsung.
2) Pada saat Praktikum, Jawabanlah Soal Pertanyaan yang memiliki Label **WAJIB** terlebih dahulu Pada Lembar **"Laporan Praktikum".**
3) Segala Bentuk **Soal yang memiliki Jawaban** berupa **Kode Program**, maka kode program tersebut harus disimpan pada **File java Project** yang telah dibuat.
4) Setiap **File Java** yang dibuat harus mencantumkan Pertanyaan pada bagian atas (baris pertama)
5) Simpan **File Laporan Praktikum** yang berupa **DOCX** menjadi **FILE PDF** kemudian ubah nama file PDF menjadi:
   "PertemuanX_NPM AKHIR.pdf"
6) Upload File **Laporan Praktikum [PDF]** pada form yang sudah disediakan.

TUGAS PRAKTIKUM

1. Jelaskan apa bedanya **Abstraction** dan **Interfaces**! [wajib]
2. Jelaskan apa yang dimaksud **Exception Handling**! [wajib]
3. Budi Arye membuat sedang belajar tentang abstraction dan exception handling
   Ia membuat program sebagai berikut :

```
abstract class Animal {
    string nama;

    Animal(String naMa) {
        this.nama = nama;
    }

    abstract void bersuara();

    void sleep() {
        System.out.println(nama + " lagi tidur.");
    }
}
```

```
class Wolf implements Animal {
    Wolf(String name) {
        super(String name);
    }

    @Override
    void bersuara() {
        System.out.println(nama + " says: Howl Howl");
    }
}
```

```
class Dog extends Wolf {

    Dog(String name) {

        super(name);

    }

    @Override

    void bersuara() {

        System.out.println(nama + " says: Woof Woof");}}

public class Main {
    public static void main(String[] args) {
        try {
            Wolf seringila = new Wolf("Budyeah");
            seringila.bersuara();
            seringila.sleep();
            Dog anjing = new Dog("Aryeah");
            anjing.bersuara();
            anjing.sleep();
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[5]);
        } catch (ArrayBoundsException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            System.out.println("end.");
        }
    }
}
```

Bantu Budi menyelesaikan kodingannya agar mendapatkan output kurang lebih sebagai berikut: [wajib]

```
Budyeah says: Woof Woof
Budyeah mending tidur.
Aryeah says: Howl Howl
Aryeah mending tidur.
Error: Index 5 out of bounds for length 3
End.
```

4. Dari soal praktikum sebelumnya kalian telah membuat class **Paket** yang di extend atau diturunkan ke kelas **PaketDomestik** dan **PaketInternasional**.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 **PaketDomestik** yang menerima objek Pembayaran sebagai parameter dan memanggil metode prosesPembayaran.
    - Buatlah objek **PembayaranTunai** dan **PembayaranKartu**, lalu gunakan metode bayar pada objek PaketDomestik untuk melakukan pembayaran.

5. 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 PaketDomestik yang menerima objek **ValidasiPembayaran** sebagai parameter dan memanggil metode validasi.
- Buat objek ValidasiSaldo dan ValidasiKartu, lalu gunakan metode validasiPembayaran pada objek PaketDomestik untuk melakukan validasi sebelum pembayaran.

TUGAS PRAKTIKUM

Soal Praktikum

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

Jawaban

pengertian
- Abstraction adalah proses menyembunyikan detail implementasi suatu fungsi atau objek dan hanya menampilkan fitur atau perilaku yang relevan bagi pengguna.
- Interface adalah kontrak atau perjanjian dalam pemrograman, yang mendefinisikan metode tanpa implementasi (hanya tanda tangan metode).
- Abstraction
  - bisa berisi abstract dan non abstract method
- kita harus menuliskan sendiri modifiernya
- Interfaces
- hanya boleh berisi abstract method
- kita tidak perlu menuliskan public abtrack di depan nama method

Source Code

```
Tulis kode program dikotak ini...


1 kotak dan 1 Penjelasan untuk 1 Class
```

Penjelasan

Tulis Penjelasan disini …

Output

Masukan screenshot output disini

TUGAS PRAKTIKUM

Soal Praktikum

Jelaskan apa yang dimaksud Exception Handling!

Jawaban

Exception Handling adalah masalah yang terjadi Ketika kita mengeksekusi program. Pada saat masalah ini terjadi alur jalanya program yang terganggu Bisa terjadi karena user salah memasukan Nilai, file tidak dapat di temukan atau terputusnya koneksi jaringan di saat melakukan Komunikasi

Source Code

```
Tulis kode program dikotak ini...

1 kotak dan 1 Penjelasan untuk 1 Class
```

Penjelasan

Tulis Penjelasan disini …

Output

Masukan screenshot output disini

TUGAS PRAKTIKUM

Soal Praktikum

Budi Arye membuat sedang belajar tentang abstraction dan exception handling Ia membuat program sebagai berikut :

Bantu Budi menyelesaikan kodingannya agar mendapatkan output kurang

lebih sebagai berikut: [wajib]

```
Budyeah says: Woof Woof
Budyeah mending tidur.
Aryeah says: Howl Howl
Aryeah mending tidur.
Error: Index 5 out of bounds for length 3
End.
```

Jawaban

Ketik jawaban disini …

Source Code

```
abstract class Animal {
    String nama;

    // Konstruktor untuk menginisialisasi nama
    Animal(String nama) {
        this.nama = nama;
    }


    abstract void bersuara();

    void sleep() {
        System.out.println(nama + " mending tidur.");
    }
}
```

```
class Wolf extends Animal {

    Wolf(String name) {
        super(name);
    }

    @Override
    void bersuara() {
        System.out.println(nama + " says: Howl Howl");
    }
}

class Dog extends Wolf {

    Dog(String name) {
        super(name);
    }

    @Override
    void bersuara() {
        System.out.println(nama + " says: Woof Woof");
    }
}

public class Main {
    public static void main(String[] args) {
        try {
            Wolf seringila = new Wolf("Budyeah");
            seringila.bersuara();
            seringila.sleep();

            Dog anjing = new Dog("Aryeah");
            anjing.bersuara();
            anjing.sleep();

            int[] numbers = {1, 2, 3, 4, 5};
             System.out.println(numbers[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
```

```
            System.out.println("Error: " + e.getMessage());
        } finally {
            System.out.println("end.");
        }
    }
}
```

Penjelasan

Kelas abstrak Animal mendefinisikan metode bersuara() yang harus diimplementasikan oleh kelas turunan, yaitu Wolf dan Dog. Metode sleep() pada kelas Animal memberikan pesan bahwa hewan tersebut sedang tidur. Di dalam main, objek Wolf dan Dog dibuat untuk menunjukkan implementasi metode bersuara() dan sleep(). Exception handling digunakan untuk menangani kesalahan ketika mencoba mengakses elemen array di luar batas, dengan menangkap ArrayIndexOutOfBoundsException. Program ini juga menampilkan pesan kesalahan dan memastikan bahwa blok finally selalu dieksekusi setelah pengecualian.

Output

```
aceStorage\d17c6d556253b5608ce04372ebfd128d2\redhat.java\jdt_ws\Praktikum 4_5fc0
Budyeah says: Howl Howl
Budyeah mending tidur.
Aryeah says: Woof Woof
Aryeah mending tidur.
Error: Index 5 out of bounds for length 5
end.
PS C:\Users\agung\Pictures\Kuliah\Semester 3\Coding Java\Praktikum 4>
```

TUGAS PRAKTIKUM

Soal Praktikum

Dari soal praktikum sebelumnya kalian telah membuat class Paket yang di extend atau diturunkan ke kelas PaketDomestik dan PaketInternasional.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 PaketDomestik yang menerima objek Pembayaran sebagai parameter dan memanggil metode prosesPembayaran.

• Buatlah objek PembayaranTunai dan PembayaranKartu, lalu gunakan metode bayar pada objek PaketDomestik untuk melakukan pembayaran.

Jawaban

Ketik jawaban disini …

Source Code

```
interface Pembayaran {
    void prosesPembayaran(double jumlah);
}

class PembayaranTunai implements Pembayaran {
    @Override
    public void prosesPembayaran(double jumlah) {
```

```
        System.out.println("Pembayaran Tunai sebesar: " +
jumlah);
    }
}

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 Kartu sebesar: " +
jumlah);
        System.out.println("Nomor Kartu: " + nomorkartu);
        System.out.println("Nama Pemilik: " + namapemilik);
    }
}

abstract class Paket {
    String namaPaket;

    Paket(String namaPaket) {
        this.namaPaket = namaPaket;
    }

    abstract void bayar(Pembayaran pembayaran, double jumlah);
}

class PaketDomestik extends Paket {
    PaketDomestik(String namaPaket) {
        super(namaPaket);
    }

    @Override
```

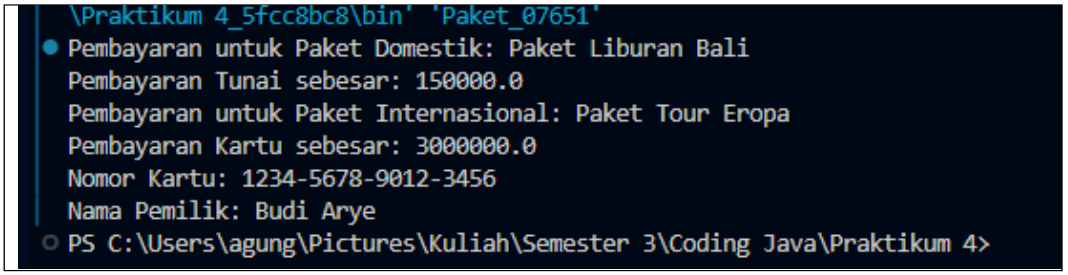
```
    void bayar(Pembayaran pembayaran, double jumlah) {
        System.out.println("Pembayaran untuk Paket Domestik: " +
namaPaket);
        pembayaran.prosesPembayaran(jumlah);
    }
}

class PaketInternasional extends Paket {
    PaketInternasional(String namaPaket) {
        super(namaPaket);
    }

    @Override
    void bayar(Pembayaran pembayaran, double jumlah) {
        System.out.println("Pembayaran untuk Paket
Internasional: " + namaPaket);
        pembayaran.prosesPembayaran(jumlah);
    }
}

public class Paket_07651 {
    public static void main(String[] args) {
        Pembayaran pembayaranTunai = new PembayaranTunai();
        Pembayaran pembayaranKartu = new PembayaranKartu("1234-
5678-9012-3456", "Budi Arye");

        Paket paketDomestik = new PaketDomestik("Paket Liburan
Bali");
        Paket paketInternasional = new PaketInternasional("Paket
Tour Eropa");

        paketDomestik.bayar(pembayaranTunai, 150000);
        paketInternasional.bayar(pembayaranKartu, 3000000);
    }
}
```

Penjelasan

Kode di atas mendemonstrasikan penggunaan interface di Java untuk mendefinisikan metode umum prosesPembayaran(double jumlah), yang kemudian

diimplementasikan oleh dua kelas pembayaran: PembayaranTunai dan PembayaranKartu. Kelas PaketDomestik dan PaketInternasional mewarisi kelas abstrak Paket dan mengimplementasikan metode bayar, yang menerima objek Pembayaran untuk memproses pembayaran paket. PembayaranTunai hanya mencetak jumlah pembayaran tunai, sedangkan PembayaranKartu juga mencetak informasi kartu yang digunakan. Kode ini menekankan prinsip polymorphism dengan menggunakan objek Pembayaran yang dapat berupa jenis pembayaran apa pun, baik tunai maupun kartu.

Output



TUGAS PRAKTIKUM

Soal Praktikum

5. Berdasarkan soal No.4 Buatlah interface ValidasiPembayaran dengan metode validasi (double jumlah). Buatlah class ValidasiSaldo dan ValidasiKartu yang mengimplementasikan 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 PaketDomestik yang menerima objek ValidasiPembayaran sebagai parameter dan memanggil metode validasi.
• Buat objek ValidasiSaldo dan ValidasiKartu, lalu gunakan metode validasiPembayaran pada objek PaketDomestik untuk melakukan validasi sebelum pembayaran.

Jawaban

Ketik jawaban disini …

Source Code

```
interface Pembayaran {
    void prosesPembayaran(double jumlah);
}

interface ValidasiPembayaran {
    boolean validasi(double jumlah);
}

class ValidasiSaldo implements ValidasiPembayaran {
    private double saldo;

    public ValidasiSaldo(double saldo) {
        this.saldo = saldo;
    }

    @Override
    public boolean validasi(double jumlah) {
        System.out.println("Memeriksa saldo...");
        return saldo >= jumlah;
    }
```

```
}

class ValidasiKartu implements ValidasiPembayaran {
    private String nomorKartu;

    public ValidasiKartu(String nomorKartu) {
        this.nomorKartu = nomorKartu.replaceAll("[^0-9]", ""); // Hapus
semua karakter non-angka
    }

    @Override
    public boolean validasi(double jumlah) {
        System.out.println("Memeriksa nomor kartu...");
        return nomorKartu.length() == 16;
    }
}

class PembayaranTunai implements Pembayaran {
    @Override
    public void prosesPembayaran(double jumlah) {
        System.out.println("Pembayaran Tunai sebesar: " + jumlah);
    }
}

class PembayaranKartu implements Pembayaran {
    String nomorkartu;
    String namapemilik;

    PembayaranKartu(String nomorkartu, String namapemilik) {
        this.nomorkartu = nomorkartu.replaceAll("[^0-9]", ""); // Hapus
karakter non-angka
        this.namapemilik = namapemilik;
    }

    @Override
    public void prosesPembayaran(double jumlah) {
        System.out.println("Pembayaran Kartu sebesar: " + jumlah);
        System.out.println("Nomor Kartu: " + nomorkartu);
        System.out.println("Nama Pemilik: " + namapemilik);
    }
}
```

```
abstract class Paket {
    String namaPaket;

    Paket(String namaPaket) {
        this.namaPaket = namaPaket;
    }

    abstract void bayar(Pembayaran pembayaran, double jumlah);

    public void validasiPembayaran(ValidasiPembayaran validasi,
Pembayaran pembayaran, double jumlah) {
        if (validasi.validasi(jumlah)) {
            System.out.println("Validasi berhasil.");
            bayar(pembayaran, jumlah);
        } else {
            System.out.println("Validasi gagal. Pembayaran
dibatalkan.");
        }
    }
}

class PaketDomestik extends Paket {
    PaketDomestik(String namaPaket) {
        super(namaPaket);
    }

    @Override
    void bayar(Pembayaran pembayaran, double jumlah) {
        System.out.println("Pembayaran untuk Paket Domestik: " +
namaPaket);
        pembayaran.prosesPembayaran(jumlah);
    }
}

class PaketInternasional extends Paket {
    PaketInternasional(String namaPaket) {
        super(namaPaket);
    }

    @Override
    void bayar(Pembayaran pembayaran, double jumlah) {
```

```
        System.out.println("Pembayaran untuk Paket Internasional: " +
namaPaket);
        pembayaran.prosesPembayaran(jumlah);
    }
}

public class ValidasiPembayaran_07651 {
    public static void main(String[] args) {
        ValidasiPembayaran validasiSaldo = new ValidasiSaldo(200000);
        ValidasiKartu validasiKartu = new ValidasiKartu("1234-5678-
9012-3456");

        Pembayaran pembayaranTunai = new PembayaranTunai();
        Pembayaran pembayaranKartu = new PembayaranKartu("1234-5678-
9012-3456", "Budi Arye");

        Paket paketDomestik = new PaketDomestik("Paket Liburan Bali");
        Paket paketInternasional = new PaketInternasional("Paket Tour
Eropa");

        System.out.println("Proses untuk Paket Domestik:");
        paketDomestik.validasiPembayaran(validasiSaldo,
pembayaranTunai, 150000);

        System.out.println("\nProses untuk Paket Internasional:");
        paketInternasional.validasiPembayaran(validasiKartu,
pembayaranKartu, 3000000);
    }
}
```

Penjelasan

Program ini mengimplementasikan proses pembayaran dengan validasi menggunakan Java, melibatkan antarmuka Pembayaran untuk mendefinisikan metode prosesPembayaran dan ValidasiPembayaran untuk validasi. Dua kelas validasi, yaitu ValidasiSaldo memeriksa cukup atau tidaknya saldo, dan ValidasiKartu memastikan nomor kartu valid dengan panjang 16 digit. Pembayaran dapat dilakukan tunai melalui kelas PembayaranTunai atau dengan kartu melalui PembayaranKartu. Ada dua jenis paket, yaitu PaketDomestik dan PaketInternasional, yang memanfaatkan metode bayar untuk memproses pembayaran setelah validasi berhasil. Program utama memvalidasi dan

memproses pembayaran untuk dua paket liburan dengan metode pembayaran dan validasi berbeda.

Output

```
[Running] cd "d:\Kuliah\Semester 3\Coding Java\Praktikum 4\" && j
Proses untuk Paket Domestik:
Memeriksa saldo...
Validasi berhasil.
Pembayaran untuk Paket Domestik: Paket Liburan Bali
Pembayaran Tunai sebesar: 150000.0

Proses untuk Paket Internasional:
Memeriksa nomor kartu...
Validasi berhasil.
Pembayaran untuk Paket Internasional: Paket Tour Eropa
Pembayaran Kartu sebesar: 3000000.0
Nomor Kartu: 1234567890123456
Nama Pemilik: Budi Arye
```

TUGAS PRAKTIKUM

Soal Praktikum

Ketik soal disini …

Jawaban

Ketik jawaban disini …

Source Code

```
Tulis kode program dikotak ini...

1 kotak dan 1 Penjelasan untuk 1 Class
```

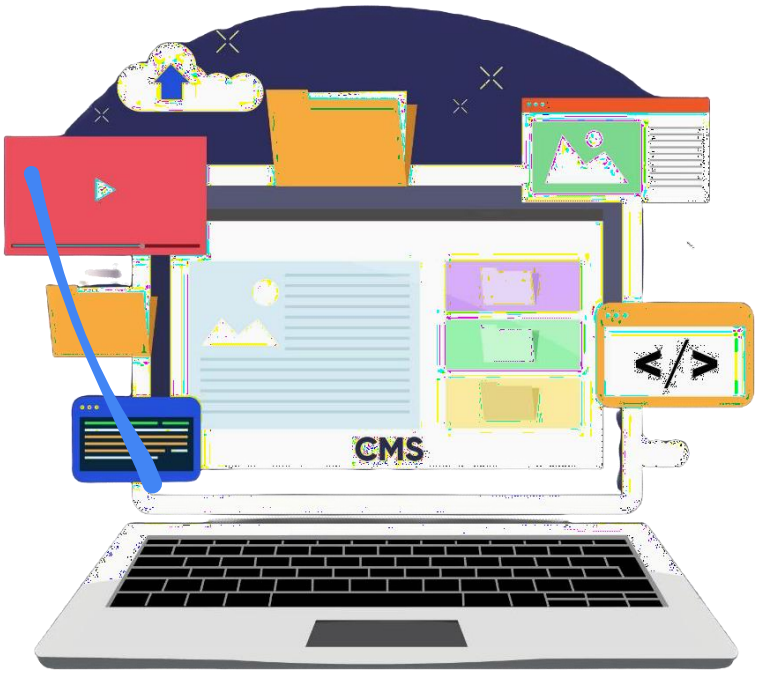
Penjelasan

Tulis Penjelasan disini …

Output

Masukan screenshot output disini

TUGAS DAN EVALUASI



PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
PERIODE X

Nama	:	Michael Agung Y.P
NPM	:	06.2023.1.07657
Modul	:	8

TUGAS & EVALUASI

Soal Tugas & Evaluasi

Jelaskan secara singkat tentang Abstraction dan Interfaces, serta perbedaannya!

Jawaban

Abstraction adalah konsep dalam OOP yang digunakan untuk menyembunyikan detail implementasi dan hanya menampilkan fungsional penting kepada pengguna.

Interface adalah kontrak yang mendefinisikan sekumpulan metode yang harus diimplementasikan oleh kelas yang menggunakan

Perbedaan Abstract dan interface:

- Abstract menggunakan keyword abstract, sedangkan interface menggunakan keyword interface
- Abstract class dapat memiliki modifier private, protected dan public sedangkan semua method di interface bersifat public secara default
- Abstract tidak mendukung multiple inheritance langsung, tetapi interface mendukung multiple inheritance dangan implementasi

Source Code

```
Tulis kode program dikotak ini...

1 kotak dan 1 Penjelasan untuk 1 Class
```

Penjelasan

Tulis Penjelasan disini …

Output

Masukan screenshot output disini

TUGAS & EVALUASI

Soal Tugas & Evaluasi

Buatlah class abstract dengan nama Mahasiswa, kemudian tambahkan method nama(), npm(), isMhsAktif(). Kemudian buatlah method non-abstract untuk menampilkan data mahasiswa tersebut. Pada method non-abstract gunakan operator ternary. Setelahnya, buatlah 2 sub-class dari Mahasiswa tersebut dengan nama Pagi dan Malam!

Jawaban

Ketik jawaban disini …

Source Code

```
abstract class Mahasiswa {
    abstract String nama();
    abstract String NPM();
    abstract boolean isMhsAktif();

    void tampilkanMahasiswa() {
        System.out.println("Nama: " + nama());
        System.out.println("NPM: " + NPM());
        System.out.println("Status: " + (isMhsAktif() ? "Aktif"
: "Tidak Aktif"));
    }
}

class Pagi extends Mahasiswa {
    private String nama;
    private String NPM;
    private boolean aktif;


    Pagi(String nama, String NPM, boolean aktif){
        this.nama = nama;
```

```
        this.NPM = NPM;
        this.aktif = aktif;
    }

    @Override
    String nama() {
        return nama;
    }

    @Override
    String NPM() {
        return NPM;
    }

    @Override
    boolean isMhsAktif() {
        return aktif;
    }
}

class Malam extends Mahasiswa {
    private String nama;
    private String NPM;
    private boolean aktif;


    Malam(String nama, String NPM, boolean aktif){
        this.nama = nama;
        this.NPM = NPM;
        this.aktif = aktif;
    }

    @Override
    String nama() {
        return nama;
    }

    @Override
    String NPM() {
```

```
        return NPM;
    }

    @Override
    boolean isMhsAktif() {
        return aktif;
    }
}


public class Mahasiswa_07651{
    public static void main(String[] args) {
        Mahasiswa mhsPagi = new Pagi("Budi", "12345", true);
        Mahasiswa mhsMalam = new Malam("Anggi", "12345", false);



        System.out.println("Mahasiswa pagi");
        mhsPagi.tampilkanMahasiswa();

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

        System.out.println("Mahasiswa malam");
        mhsMalam.tampilkanMahasiswa();
    }
}
```

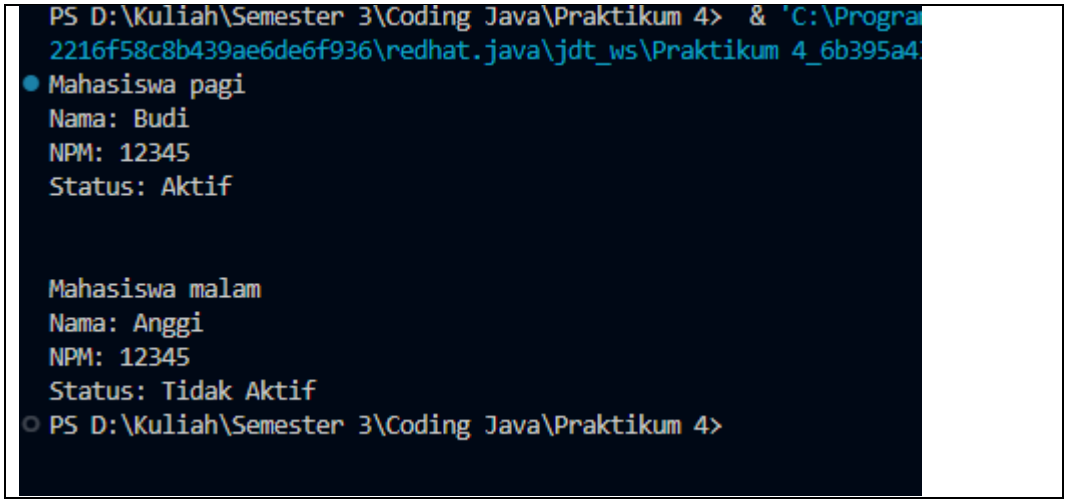
Penjelasan

Kode ini menggunakan *class abstrak* untuk mendefinisikan template data mahasiswa, termasuk nama, NPM, dan status aktif mahasiswa. Class abstrak **Mahasiswa** memiliki metode abstrak seperti nama(), NPM(), dan isMhsAktif() yang harus diimplementasikan oleh subkelas seperti **Pagi** dan **Malam**. Kedua subkelas ini mengimplementasikan perilaku spesifik sesuai data mahasiswa pagi dan malam. Metode tampilkanMahasiswa() di kelas abstrak digunakan untuk mencetak informasi mahasiswa dengan format yang seragam. Kode ini

TUGAS & EVALUASI

memanfaatkan *inheritance dan polymorphism* untuk mempermudah pengelolaan berbagai tipe mahasiswa.

Output



TUGAS & EVALUASI

Soal Tugas & Evaluasi

Buatlah class abstract dengan nama Transportasi, yang di dalamnya terdapat atribut nama dan kapasitas serta method abstract untuk menampilkan cara memesan dan juga untuk menghitung tarif transportasinya. Setelahnya, buat sub-class Taksi, Bus, dan KeretaApi yang meng-implementasikan method serta atribut dari superclass. Note: Masing-masing Sub-Class harus memiliki perilaku pemesanan dan penghitungan tarif yang berbeda sesuai jenisnya.

Jawaban

Ketik jawaban disini …

Source Code

```
abstract class Transportasi {
    String nama;
    int kapasitas;

    abstract String caraMemesan();

    abstract String hitungTarif(int jarak);

    Transportasi(String nama, int kapasitas) {
        this.nama = nama;
        this.kapasitas = kapasitas;
    }
}

class Taksi extends Transportasi {
```

```
    double tarifPerKm;

    Taksi(String nama, int kapasitas, double tarifPerKm) {
        super(nama, kapasitas);
        this.tarifPerKm = tarifPerKm;
    }

    @Override
    String caraMemesan() {
        return "Pesan melalui aplikasi";
    }

    @Override
    String hitungTarif(int jarak) {
        return tarifPerKm * jarak + " Rupiah";
    }
}

class Bus extends Transportasi {
    double tarifPerPenumpang;

    Bus(String nama, int kapasitas, double tarifPerPenumpang) {
        super(nama, kapasitas);
        this.tarifPerPenumpang = tarifPerPenumpang;
    }

    @Override
    String caraMemesan() {
        return "Pesan melalui terminal atau halte";
    }

    @Override
    String hitungTarif(int jarak) {
        return tarifPerPenumpang * kapasitas + " Rupiah";
    }
}

class KeretaApi extends Transportasi {
    double tarifDasar;
```

```
    KeretaApi(String nama, int kapasitas, double tarifDasar) {
        super(nama, kapasitas);
        this.tarifDasar = tarifDasar;
    }

    @Override
    String caraMemesan() {
        return "Pesan melalui aplikasi atau datang langsung ke
stasiun";
    }

    @Override
    String hitungTarif(int jarak) {
        return tarifDasar + (jarak * 500) + " Rupiah";
    }
}

public class Transport_07651 {
    public static void main(String[] args) {
        Transportasi taksi = new Taksi("Taksi", 4, 5000);
        Transportasi bus = new Bus("Bus", 40, 2000);
        Transportasi keretaApi = new KeretaApi("Kereta Api",
200, 5000);

        System.out.println("Cara memesan Taksi: " +
taksi.caraMemesan());
        System.out.println("Tarif Taksi untuk 10 km: " +
taksi.hitungTarif(10) + "\n");

        System.out.println("Cara memesan Bus: " +
bus.caraMemesan());
        System.out.println("Tarif Bus untuk 10 km: " +
bus.hitungTarif(10) + "\n");

        System.out.println("Cara memesan Kereta Api: " +
keretaApi.caraMemesan());
        System.out.println("Tarif Kereta Api untuk 10 km: " +
keretaApi.hitungTarif(10) + "\n");
    }
```

```
}
```

Penjelasan

Kode di atas merupakan penerapan abstraksi dalam Java, di mana class abstract Transportasi digunakan sebagai kerangka kerja untuk berbagai jenis transportasi seperti Taksi, Bus, dan KeretaApi. Kelas abstrak ini mendefinisikan metode caraMemesan() dan hitungTarif(int jarak) yang wajib diimplementasikan oleh subkelasnya. Setiap jenis transportasi memiliki logika perhitungan tarif dan cara pemesanan yang berbeda sesuai karakteristiknya. Kode ini juga memanfaatkan polymorphism, di mana semua subkelas diperlakukan sebagai objek Transportasi di dalam metode main() untuk mengakses metode yang diimplementasikan masing-masing kelas.

Output

```
PS D:\Kuliah\Semester 3\Coding Java\Praktikum 4>  & 'C:\Program Files\Java\jdk-23\bi
2216f58c8b439ae6de6f936\redhat.java\jdt_ws\Praktikum 4_6b395a43\bin' 'Transport_0765
Cara memesan Taksi: Pesan melalui aplikasi
Tarif Taksi untuk 10 km: 50000.0

Cara memesan Bus: Pesan melalui terminal atau halte
Tarif Bus untuk 10 km: 80000.0

Cara memesan Kereta Api: Pesan melalui aplikasi atau datang langsung ke stasiun
Tarif Kereta Api untuk 10 km: 1000000.0

PS D:\Kuliah\Semester 3\Coding Java\Praktikum 4>
```

TUGAS & EVALUASI

Soal Tugas & Evaluasi

Buatlah class interface Pembayaran dan Transportasi berdasarkan studi kasus dari soal no.3, kemudian implementasikan kedua class tersebut pada class Taksi, Bus, dan KeretaApi yang baru!

PETUNJUK:

- Interface Transportasi berisi method void bergerak() dan double hitungTarif().

- Interface Pembayaran berisi method void bayar().

Jawaban

Ketik jawaban disini …

Source Code

```
abstract class Transportasi {
    String nama;
    int kapasitas;

    abstract String  caraMemesan();

    abstract double hitungTarif(int jarak);

    abstract void bergerak();

    Transportasi(String nama, int kapasitas){
        this.nama = nama;
        this.kapasitas = kapasitas;

    }
}

interface Pembayaran{
    void bayar(double jumlah);
}
```

```
class Taksi extends Transportasi implements Pembayaran
{
    double tarifPerKm;


    Taksi(String nama, int kapasitas, double tarifPerKm){
        super(nama, kapasitas);
        this.tarifPerKm = tarifPerKm;
    }

    @Override
    String caraMemesan() {
        return "Pesan melalui aplikasi";
    }

    @Override
    double hitungTarif(int jarak) {
        return tarifPerKm * jarak;
    }

    @Override
    public void bayar(double jumlah) {
        System.out.println("Bayar dengan uang tunai atau kartu "
+ jumlah + " Rupiah");
    }

    @Override
    void bergerak() {
        System.out.println(nama + " Taksi bergerak");
    }
}

class Bus extends Transportasi implements Pembayaran{
    double tarifPerPenumpang;


    Bus(String nama, int kapasitas, double tarifPerPenumpang){
        super(nama, kapasitas);
```

```
        this.tarifPerPenumpang = 2000;
    }

    @Override
    String caraMemesan() {
        return "Pesan melalui terminal atau halte";
    }

    @Override
    double hitungTarif(int jarak) {
        return tarifPerPenumpang * kapasitas;
    }

    @Override
    public void bayar(double jumlah) {
        System.out.println("Bayar dengan uang tunai " + jumlah +
" Rupiah");
    }

    @Override
    void bergerak() {
        System.out.println(nama + " Bus bergerak");
    }
}

class KeretaApi extends Transportasi{
    double tarifDasar;

    KeretaApi(String nama, int kapasitas, double tarifDasar){
        super(nama, kapasitas);
        this.tarifDasar = 5000;
    }

    @Override
    String caraMemesan() {
        return "Pesan melalui aplikasi atau datang langsung ke
stasiun";
    }
```

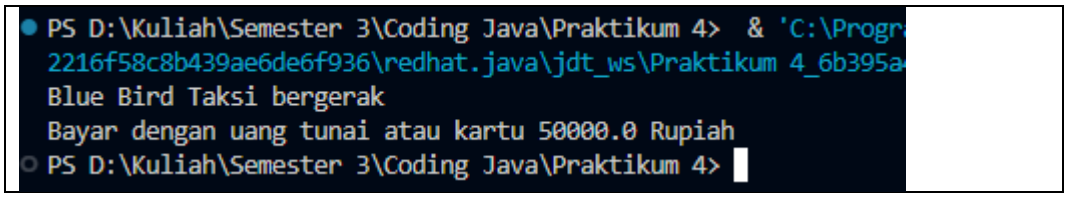
```
    @Override
    double hitungTarif(int jarak) {
        return tarifDasar * kapasitas;
    }

    @Override
    void bergerak() {
        System.out.println("Kereta Api bergerak");
    }
}


public class Tranport2_07651 {
    public static void main(String[] args) {
        Taksi taksi = new Taksi("Blue Bird", 4, 5000);

        taksi.bergerak();
        double tarifTaksi = taksi.hitungTarif(10);
        taksi.bayar(tarifTaksi);

    }
}
```

Penjelasan

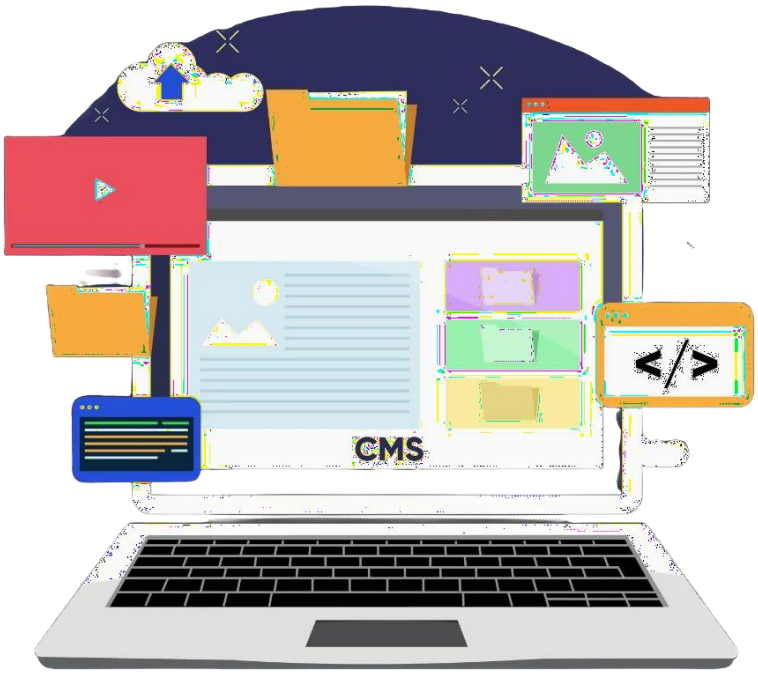
Kode di atas menunjukkan penerapan **abstract class** dan **interface** dalam Java untuk mendefinisikan kerangka dasar transportasi. Kelas abstrak **Transportasi** digunakan untuk memberikan atribut dan metode umum seperti caraMemesan, hitungTarif, dan bergerak, sementara **interface Pembayaran** menyediakan kontrak untuk metode pembayaran. Kelas **Taksi** dan **Bus** mewarisi Transportasi dan mengimplementasikan Pembayaran, sehingga mereka memiliki perilaku khusus seperti cara memesan, tarif, dan metode pembayaran yang spesifik. **KeretaApi**, meskipun tidak mengimplementasikan Pembayaran, tetap mengimplementasikan metode Transportasi.

TUGAS & EVALUASI

Output



TUGAS DAN EVALUASI



PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
PERIODE X

Nama	:	Michael Agung Y.P
NPM	:	06.2023.1.07657
Modul	:	9

TUGAS & EVALUASI

Soal Tugas & Evaluasi

Jelaskan apa yang dimaksud dengan **Exception Handling** dan sebutkan tujuannya!

Jawaban

Exception handling adalah mekanisme untuk menangani salah satu permasalahan yang terjadi Ketika mengeksekusi Program. Dimana pada saat terjadi sebuah kesalahan pada kode program kita, alur jalanya program akan terganggu dan bisa jadi berakhir

Tujuan Utama

- Mengidentifikasi situasi tidak normal
- Memberikan informasi yang berguna
- Menghindari penghetian mendadak
- Menangani kodisi tidak terduga
- Menigkatkan keamanan system

Source Code

```
Tulis kode program dikotak ini...

1 kotak dan 1 Penjelasan untuk 1 Class
```

Penjelasan

Tulis Penjelasan disini …

Output

Masukan screenshot output disini

TUGAS & EVALUASI

Soal Tugas & Evaluasi

Sebutkan kategori **Exception Handling** beserta penjelasannya!

Jawaban

- Checked Exception adalah masalah yang terjadi di saat proses compile program sedang berlangsung. Kategori exception ini tidak dapat dibiarkan begitu saja, kita harus menangani kesalahan ini jika tidak ingin program kita berhenti di Tengah jalan

- unChecked Exception adalah masalah yang tidak harus di tangani seperti halnya pada Checked Exception. Sederhanannya Exception ini terjadi pada saat program menemukan sebuah Bug seperti kesalahan Logika pemrograman

- Error Bukan termasuk kedalam Exception karena masalah ini bukan di sebabkan oleh user maupun programmer. Melaikan biasanya terjadi karena kondisi yang serius dan tidak dapat di pulihkan

-

Source Code

```
Tulis kode program dikotak ini...

1 kotak dan 1 Penjelasan untuk 1 Class
```

Penjelasan

Tulis Penjelasan disini …

Output

Masukan screenshot output disini

TUGAS & EVALUASI

Soal Tugas & Evaluasi

Buatlah kode program yang menunjukkan implementasi kategori Exception Handling berikut:

-*Checked Exception*: Buat program yang menangani *FileNotFoundException* dan *IOException*.

-*Unchecked Exception*: Buat program yang menunjukkan penanganan *ArithmeticException* dan *NullPointerException*.

Jawaban

Ketik jawaban disini …

Source Code

```
import java.io.*;

public class CheckedException {
    public static void main(String[] args) {
        try (FileReader fileReader = new
FileReader("nonexistent_file.txt");
             BufferedReader bufferedReader = new
BufferedReader(fileReader)) {

            String line;
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (FileNotFoundException e) {
            System.out.println("Error: File tidak ditemukan.");
        } catch (IOException e) {
            System.out.println("Error: Terjadi kesalahan saat
membaca file.");
        }
    }
}
```

Penjelasan

Kode ini membaca file menggunakan FileReader dan BufferedReader dalam blok try-with-resources, memastikan sumber daya ditutup otomatis. Jika file tidak ditemukan, FileNotFoundException ditangani dengan pesan "File tidak ditemukan." Jika terjadi kesalahan lain saat membaca file, IOException ditangani dengan pesan "Terjadi kesalahan saat membaca file."

Output

```
PS D:\Kuliah\Semester 3\Coding Java\Praktikum 4>  & 'C:\Program Fi
2216f58c8b439ae6de6f936\redhat.java\jdt_ws\Praktikum 4_6b395a43\bi
Error: File tidak ditemukan.
PS D:\Kuliah\Semester 3\Coding Java\Praktikum 4>
```

Source Code

```
public class UncheckedExceptionExample {
    public static void main(String[] args) {
        try {
            int a = 10;
            int b = 0;
            int result = a / b;
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        // Contoh NullPointerException
```

```
        try {
            String str = null;
            System.out.println(str.length());
        } catch (NullPointerException e) {
            System.out.println("Error: Objek bernilai null.");
        }
    }
}
```

Penjelasan

Kode ini menunjukkan contoh penanganan **Unchecked Exception** di Java, seperti ArithmeticException dan NullPointerException. Pada blok pertama, pembagian angka dengan nol menyebabkan ArithmeticException, yang ditangani dengan mencetak pesan error. Pada blok kedua, mencoba mengakses metode pada objek null menghasilkan NullPointerException, dan ditangani dengan pesan "Error: Objek bernilai null."

Output

```
PS D:\Kuliah\Semester 3\Coding Java\Praktikum 4>  d:; cd 'd:\Kuliah
ng\AppData\Roaming\Code\User\workspaceStorage\8b00e56d92216f58c8b43
/ by zero
Error: Objek bernilai null.
PS D:\Kuliah\Semester 3\Coding Java\Praktikum 4>
```

TUGAS & EVALUASI

Soal Tugas & Evaluasi

Buat sebuah program yang melakukan operasi pembagian angka dengan penanganan menggunakan blok *try-catchfinally*. Program ini harus dapat menangani input yang tidak valid, termasuk pembagian dengan nol.

Jawaban

Ketik jawaban disini …

Source Code

```
import java.util.Scanner;

public class DivisionWithExceptionHandling {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Masukkan angka pertama: ");
            int angka1 = Integer.parseInt(scanner.nextLine());

            System.out.print("Masukkan angka kedua: ");
            int angka2 = Integer.parseInt(scanner.nextLine());

            int hasil = angka1 / angka2;
            System.out.println("Hasil pembagian: " + hasil);

        } catch (NumberFormatException e) {
            System.out.println("Error: Input harus berupa
angka.");
        } catch (ArithmeticException e) {
            System.out.println("Error: Tidak dapat membagi
dengan nol.");
        } finally {
            System.out.println("Operasi pembagian selesai.");
            scanner.close();
```

```
        }
    }
}
```

Penjelasan

Kode ini adalah contoh penanganan exception saat membagi dua angka yang dimasukkan oleh pengguna. Input dibaca menggunakan Scanner dan dikonversi menjadi integer. Jika input bukan angka, NumberFormatException ditangani dengan pesan "Input harus berupa angka." Jika terjadi pembagian dengan nol, ArithmeticException ditangani dengan pesan "Tidak dapat membagi dengan nol." Blok finally memastikan pesan "Operasi pembagian selesai." dicetak dan Scanner ditutup untuk mencegah kebocoran sumber daya.

Output