UMSIDA PRESS

BUKU AJAR PEMROGRAMAN BERORIENTASI OBJEK (PBO)

2025

Penulis:
Uce Indahyanti, M.Kom.
Ika Ratna Indra Astutik, S.Kom., MT.

Universitas Muhammadiyah Sidoarjo

Buku Ajar Pemrograman Berorientasi Objek (PBO)

Penulis:

Uce Indahyanti

Ika Ratna Indra Astutik

Anggota APPTI Nomor : 002.018.1.09.2017
Anggota IKAPI Nomor : 218/Anggota Luar Biasa/JTI/2019

Diterbitkan oleh
UMSIDA PRESS
Jl. Mojopahit 666 B Sidoarjo
ISBN: 978-623-464-130-1

Buku Ajar Pemrograman Berorientasi Objek (PBO)
Penulis: Uce Indahyanti;Ika Ratna Indra Astutik
ISBN: 978-623-464-130-1
Editor: M. Tanzil Multazam & Mahardika Darmawan K.W.
Copy Editor: Wiwit Wahyu Wijayanti
Design Sampul dan Tata Letak: Wiwit Wahyu Wijayanti
Penerbit: UMSIDA Press
Redaksi: Universitas Muhammadiyah Sidoarjo Jl. Mojopahit No.666B Sidoarjo, Jawa Timur

Cetakan Pertama, Oktober 2025

KATA PENGANTAR

Alhamdulillah, shalawat dan salam semoga selalu tercurahkan kepada Nabi Muhammad shallallahu ‘alaihi wa sallam. Puji syukur kami panjatkan ke hadirat Allah Ta’ala yang telah memberikan taufik dan kemudahan sehingga kami dapat menyelesaikan penyusunan buku ajar Pemrograman Berorientasi Objek (PBO) ini.

Buku ajar ini disusun berdasarkan Rencana Pembelajaran Semester (RPS) mata kuliah PBO pada Program Studi Informatika. Pembahasan di dalamnya meliputi konsep dasar dan lanjutan dalam pemrograman berorientasi objek menggunakan bahasa pemrograman Java, seperti enkapsulasi, pewarisan, polimorfisme, abstraksi, serta penerapan teknologi Java pada pengembangan aplikasi berbasis desktop, web, dan mobile. Setiap bab dilengkapi dengan refleksi dan latihan soal guna memperdalam pemahaman materi bagi mahasiswa.

Sebagian konten teknis dalam buku ini disusun dengan bantuan teknologi AI untuk menyunting naskah dan verifikasi sintaks. Kendati demikian, seluruh isi telah dikaji dan disesuaikan dengan kebutuhan pembelajaran agar tetap relevan dan akurat.

Ucapan terima kasih kami sampaikan kepada Dekan Fakultas Sains dan Teknologi, Kaprodi Informatika, serta Sekprodi Informatika, atas arahan dan dukungan yang telah diberikan. Kami juga menyampaikan terima kasih kepada semua pihak yang turut membantu dalam penyusunan buku ini.

Kami menyadari bahwa buku ajar ini masih memiliki keterbatasan. Oleh karena itu, kami sangat mengharapkan kritik dan saran yang membangun untuk perbaikan pada edisi berikutnya. Semoga buku ajar ini dapat memberikan manfaat bagi mahasiswa, dosen, serta semua pihak yang membutuhkan.

Sidoarjo, Juni 2025

Tim Penulis

DAFTAR ISI

DAFTAR TABEL

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

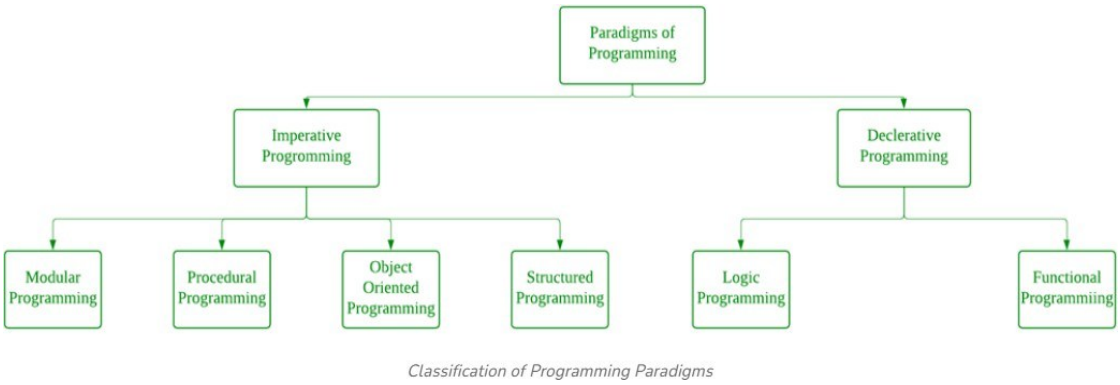
DAFTAR GAMBAR

Bab 1 Konsep Pemrograman Berorientasi Objek

Pemrograman Berorientasi Objek (PBO) atau Object Oriented Programming (OOP) merupakan salah satu paradigma pemrograman yang banyak digunakan dalam pengembangan perangkat lunak modern. Berbeda dengan pemrograman terstruktur yang lebih menitikberatkan pada urutan eksekusi instruksi, PBO berfokus pada pengolahan objek yang berisi data dan operasi yang dapat dilakukan terhadapnya (Hadiprakoso, 2021). Dalam PBO, setiap objek memiliki atribut (properti) dan perilaku (metode) yang merepresentasikan karakteristik objek pada dunia nyata. Paradigma PBO dirancang untuk meningkatkan modularitas, fleksibilitas, dan kemudahan dalam pengelolaan kode program. PBO memungkinkan pengembang untuk membangun sistem yang lebih terstruktur, mudah dipelihara, dan dapat digunakan kembali. Prinsip utama dalam PBO mencakup enkapsulasi, abstraksi, pewarisan, dan polimorfisme, yang akan dibahas lebih lanjut dalam bab ini.

1.1 Paradigma Pemrograman

Paradigma adalah suatu cara pandang atau cara berpikir. Paradigma pemrograman merupakan pendekatan-pendekatan yang digunakan dalam menyelesaikan persoalan pemrograman. Paradigma pemrograman secara luas diklasifikasikan menjadi dua yaitu: imperatif dan deklaratif yang ditunjukkan gambar 1.



Gambar 1.1 Paradigma Pemrograman

(Sumber: https://www.geeksforgeeks.org/what-is-imperative-programming/?ref=ml_lbp)

Paradigma pemrograman yang didasarkan pada langkah-langkah atau rangkaian pernyataan dan memberikan variasi pada keadaan program di setiap langkah disebut pemrograman imperatif. Pemrograman imperatif berfokus pada kinerja program dengan menentukan urutan perintah, di mana perintah dijalankan secara berurutan dan mengubah keadaan program hingga hasil akhir tercapai. Dalam pemrograman imperatif, komputer menerima serangkaian perintah secara bertahap dari program untuk mencapai hasil yang diinginkan.

Selain itu, pemrograman imperatif mengikuti gaya imperatif dalam menjalankan seluruh fungsionalitas secara bertahap. Seluruh subdomain dari pemrograman imperatif bersifat saling eksklusif dan menangani blok-blok kode kecil. Subdomain dari pemrograman imperatif tersebut antara lain sebagai berikut:

- Pemrograman Modular berurusan dengan blok-blok kode yang disebut sebagai modul yang dapat digunakan kembali. Modul-modul ini dikompilasi bersama untuk memperoleh hasil akhir. Baik pemrograman terstruktur maupun pemrograman modular menangani pembagian kode ke dalam bagian-bagian yang lebih kecil, namun pemrograman terstruktur melakukannya melalui struktur kendali, sedangkan pemrograman modular melalui penggunaan modul. Contohnya Java, C++, Python, dan C#.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

- Pemrograman prosedural membagi seluruh tugas ke dalam fungsi-fungsi yang lebih kecil. Kode yang terstruktur dan berbasis fungsi, namun bukan berorientasi objek, termasuk dalam domain pemrograman prosedural. Dalam paradigma pemrograman ini, prosedur yang didefinisikan melalui fungsi lebih dominan dibandingkan data, sehingga membedakannya dari pemrograman berorientasi objek (OOP). Contohnya Bahasa C, Pascal, Fortran, Cobol, dan BASIC.
- Pemrograman Berorientasi Objek (PBO/OOP): paradigma ini berfokus pada mengolah objek-objek berisi data dan operasi yang dibutuhkan melalui Class dan Method. Seluruh objek memiliki sekumpulan atribut dan perilaku (properti dan fungsi yang merepresentasikan perilaku tertentu). Pemrograman berorientasi objek dapat disebut sebagai pemrograman terstruktur di mana kode program mendefinisikan prosedur dan objek mendefinisikan atribut. Pemrograman berorientasi objek merupakan subkategori dari pemrograman imperatif dengan fitur tambahan berupa abstraksi, enkapsulasi, pewarisan, dan polimorfisme. Contohnya Java, Python, C++, dan PHP.
- Pemrograman Terstruktur: berfokus pada sekumpulan instruksi yang berurutan untuk menyelesaikan suatu masalah. Pemrograman terstruktur merupakan perluasan dari pemrograman imperatif yang menggunakan struktur kendali melalui iterasi dan urutan untuk mengorganisasi kode dengan menggunakan struktur blok seperti for loop, do-while, atau while loop, dan sebagainya.Contohnya Pascal, C++.

Sedangkan Pemrograman Deklaratif merupakan paradigma yang berfokus pada mendeskripsikan apa yang perlu dilakukan oleh program untuk menyelesaikan tugas tertentu. Programmer tidak perlu mengetahui secara eksplisit metode atau konsep apa yang digunakan oleh mesin, yang penting hasilnya sesuai dengan apa yang diperintahkan. Contohnya SQL dan HTML.

Sebuah bahasa pemrograman dapat mengadopsi satu atau lebih paradigma di atas, seperti bahasa Java, Delphi, Python, dan C++. Pembahasan di dalam buku ajar ini akan menyandingkan paradigma PBO menggunakan bahasa pemrograman Java.

Untuk memperoleh gambaran, di bawah ini terdapat perbandingan kedua paradigma pemrograman. Pemrograman prosedural (seperti C dan Pascal) merupakan pendekatan tradisional yang memfokuskan pada urutan eksekusi instruksi dan penggunaan fungsi. Sementara itu, Pemrograman Berorientasi Objek (PBO) menekankan pada objek, yaitu gabungan antara data dan operasi. Perbandingan antara kedua paradigma tersebut dapat dilihat pada tabel berikut:

Tabel 1.1 Perbandingan PBO dengan Pemrograman Prosedural

Aspek	**Pemrograman Prosedural**	**Pemrograman Berorientasi Objek**
Pendekatan utama	Berbasis fungsi/instruksi	Berbasis objek dan class
Organisasi program	Fungsi dan prosedur	Objek dan metode
Fokus	Algoritma dan struktur data	Entitas dunia nyata (objek)
Data	Dapat diakses bebas oleh fungsi	Tersembunyi melalui enkapsulasi
Reusabilitas	Sulit, kode sering diulang	Mudah, melalui inheritance
Skalabilitas program besar	Kurang fleksibel	Lebih mudah dikembangkan dan modular
Organisasi program	Fungsi dan prosedur	Objek dan metode

Contoh kasus sistem parkir dengan pendekatan Prosedural (C/Pascal-style pseudocode):

```
int totalTarif(int durasiJam) {
   return 5000 + (durasiJam - 1) * 3000; }
```

Sedangkan dengan pendekatan OOP (Java-style pseudocode), logika terkait parkir dapat dikemas dalam class Parkir, sehingga dapat dikembangkan lagi misalnya untuk motor, VIP, atau valet service melalui prinsip-prinsip PBO seperti pewarisan (inheritance) atau banyak bentuk (polymorphism).

```
class Parkir {
   int tarifAwal = 5000;
   int tarifPerJam = 3000;
   int hitungTarif(int durasi) {
      return tarifAwal + (durasi - 1) * tarifPerJam;
   }
}
```

Contoh lain perbandingan program tanpa OOP dengan versi OOP (class dan object) pada studi kasus menyimpan dan menampilkan data mahasiswa disajikan di bawah ini:

Versi Prosedural (Tanpa OOP)

```
public class MahasiswaProsedural {
   public static void main(String[] args) {
      String[] nama = {"Andi", "Budi", "Citra"};
      int[] umur = {20, 21, 22};
      for (int i = 0; i < nama.length; i++)
      {
         System.out.println("Nama: " + nama[i] + ", Umur: " + umur[i]);
      }
   }
}
```

Versi OOP (Menggunakan Class dan Objek)

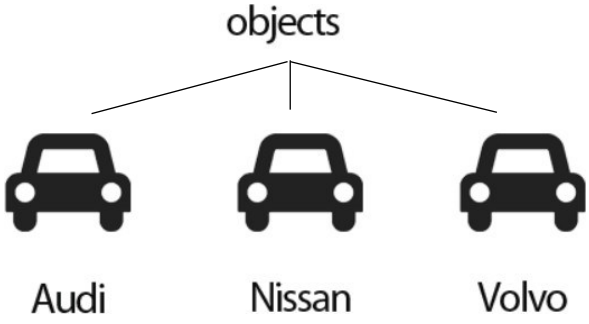
```
class Mahasiswa {
   String nama;
   int umur;
   Mahasiswa(String nama, int umur) {
      this.nama = nama;
      this.umur = umur;
   }
   void tampilkanData() {
      System.out.println("Nama: " + nama + ", Umur: " + umur);
   }
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
}
public class Main {
   public static void main(String[] args) {
      Mahasiswa[] mhs = {
         new Mahasiswa("Andi", 20),
         new Mahasiswa("Budi", 21),
         new Mahasiswa("Citra", 22)
      };
      for (Mahasiswa m : mhs)
      {
         m.tampilkanData();
      }
   }
}
```

1.2 Konsep Objek

Paradigma objek adalah cara berpikir yang memandang segala sesuatu sebagai objek. Obyek merupakan komponen yang mempunyai bentuk fisik dan dapat dilihat (visual). Obyek biasanya dipakai untuk melakukan tugas tertentu dan mempunyai batasan tertentu. Berbagai benda di sekitar kita adalah objek nyata yang dapat dilihat, seperti sebuah mobil dapat dianggap sebagai objek dengan properti seperti warna, model, dan kecepatan, serta aksi seperti mempercepat dan mengerem.



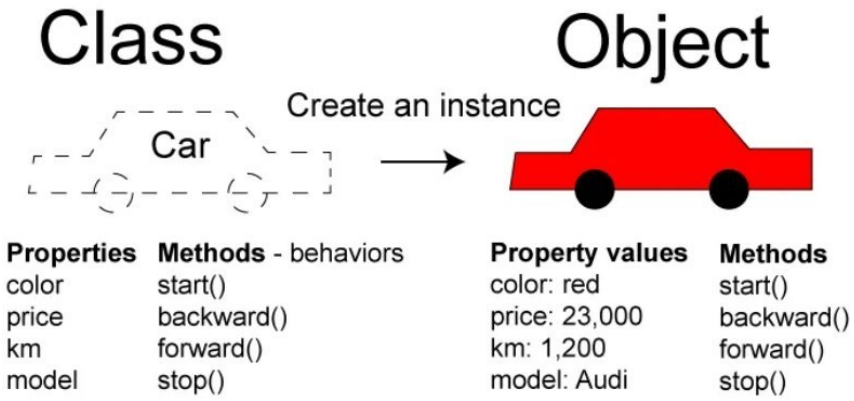
Gambar 1.2 Objek Mobil

(sumber: https://www.almabetter.com/bytes/articles/what-is-object-oriented-programming)

Gabungan obyek dan bahasa pemrograman atau sering disebut Pemrograman Berorientasi Objek atau Object Oriented Programming (OOP). Semua aspek dalam bahasa pemrograman Java dapat dianggap sebagai objek, kecuali tipe data primitif, karena semua library dan objek dalam Java memiliki akar awal class java.lang. Setiap bagian yang ada pada program dip ng sebagai suatu obyek yang dapat diubah dan diatur.

Pada paradigma PBO dikenal class dan objek yang dapat diubah dan diatur untuk memudahkan pekerjaan. Class dan objek memiliki dua karakteristik utama yaitu dari properti/atribut dan method/perilaku. Dalam PBO, obyek akan menyimpan atributnya dalam variabel dan informasi perilaku dalam method/fungsi. Misal mobil memiliki properti/atribut seperti pintu, roda, lampu, warna, kapasitas, dan harga. Mobil juga memiliki method/perilaku. Method merepresentasikan sebagai hal-hal yang dapat dilakukan oleh objek, contoh mobil dapat menyalakan mesin, mematikan mesin, berjalan maju/mundur, membunyikan klakson, menyalakan lampu, dll. Dalam OOP,

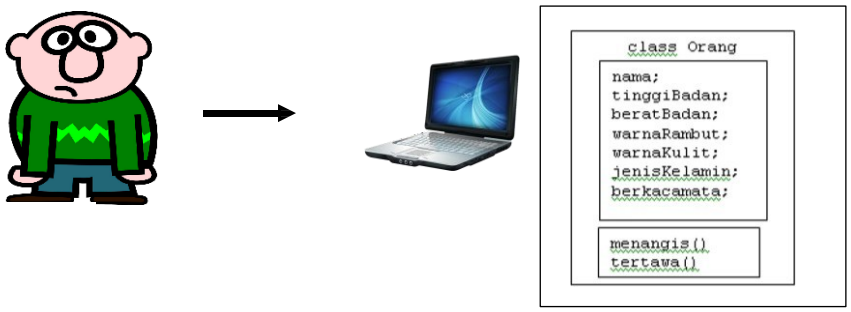
properti dan aksi tersebut dibungkus ke dalam sebuah entitas bernama kelas. Kelas berfungsi sebagai cetak biru (blueprint) untuk membuat objek.



Gambar 1.3 Class Mobil

(Sumber: https://blog.aigents.co/understanding-object-oriented-programming-concepts-4ddcc0eb3c96)

Perhatikan gambar di bawah ini untuk lebih memahami bagaimana cara memindahkan orang dari dunia nyata ke dalam bentuk class bernama Orang.



Gambar 1.4 Class Orang

Pada PBO method dan variabel dibungkus dalam sebuah objek atau class yang dapat saling berinteraksi, sehingga membentuk sebuah program. Variabel dalam objek akan menyimpan data dari objek, sedangkan method akan menentukan operasinya. Contoh berikut adalah program untuk membuat objek berupa dua buah mobil yang dipesan dua orang customer.

```
// Tanpa OOP: data mobil dikelola manual
public class DemoCarWithoutOOP {
    public static void main(String[] args) {
        String cust1Name = "Pembeli 1";
        int cust1Price = 100000000;
        String cust1Color = "Red";
        String cust2Name = "Pembeli 2";
        int cust2Price = 80000000;
        String cust2Color = "White";
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    System.out.println(cust1Name + " membeli mobil warna " + cust1Color);
    System.out.println(cust2Name + " membeli mobil warna " + cust2Color);
}
```

Pada contoh program di atas, class mobil memiliki properti/atribut seperti nama pembeli, warna, dan harga. Data tiap mobil disimpan terpisah dalam variabel (cust1Name, cust1Price, dll.). Untuk mobil kedua, diulangi lagi variabel baru sehingga terjadi duplikasi kode. Jika ada 100 mobil, maka harus ada 100 set variabel, sehingga tidak efisien, sulit dikelola, data tercecer. Tanpa OOP maka data dan fungsi terpisah sehingga banyak duplikasi.

Tentu hal itu sangat menyulitkan. Maka disinilah peran class dan objek pada konsep PBO. Class dapat dianalogikan seperti cetakan / blueprint, dimana sebuah objek yang sama dengan jumlah banyak dapat dibentuk berdasarkan cetakan yang didefinisikan. Objek dari sebuah class disebut juga instance dari class, sehingga sebuah cetakan yang didefinisikan dapat membuat banyak mobil sekaligus (pada contoh di atas). Sehingga program dapat disusun menggunakan cetakan (class Car) sebagai berikut:

```
class Car
{
  name;
  price;
  color;
  capacity;
  isEngineStarted = false;
   constructor({ name, price, color, capacity = 4, policeNumber })
{
    this.name = name;
    this.price = price;
    this.color = color;
    this.capacity = capacity;
  }
 start()
{
    this.isEngineStarted = true;
    console.log('Start the engine');
  }
stop()
{
    this.isEngineStarted = false;
    console.log('Stop the engine');
  }
moveForward()
```

```
{
   console.log('Move forward');
 }
moveBackward()
{
   console.log('Move backward');
 }
brake() {
   console.log('Brake');
 }
}
```

Pada kode OOP, data mobil dikelompokkan dalam class Car dengan atribut (owner, price, color) dan method (info()). Untuk membuat mobil baru, cukup buat objek dengan constructor (new Car(...)). Tidak ada duplikasi variabel: semua sudah diorganisir dalam class. Sehingga kode lebih rapi, mudah diperluas (misalnya bisa ditambahkan method start(), brake()), dan mendukung reuse. Dengan OOP, data dan fungsi terintegrasi dalam class sehingga lebih rapi, efisien, dan terukur.

1.3 Prinsip PBO

Prinsip dasar dari PBO meliputi abstraksi, enkapsulasi, pewarisan, dan polimorfisme sebagai berikut:

Enkapsulasi

Konsep ini mengacu pada pembungkusan data (variabel dan fungsi/metode) dalam satu unit, yang disebut class (kelas). Konsep ini dilakukan dengan mendeklarasikan semua variabel di kelas sebagai privat dan fungsi di kelas sebagai public.

Contoh enkapsulasi pada Java (variabel nama dan umur dibuat private, fungsi/method getName dan fungsi lainnya dibuat public):

```
class Person
{
   private String name;
   private int age;
    public String getName()
{
   return name;
}
    public void setName(String name)
{
  this.name = name;
 }
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
   public int getAge()
{
  return age;
}
   public void setAge(int age) { this.age = age;
 }
```

Abstraksi

Konsep ini memilah beberapa atribut dan operasi pada suatu obyek untuk yang diperlukan saja. Konsep ini hanya menampilkan fungsi yang penting kepada pengguna, dan menyembunyikan detil implementasi yang tidak penting.
Contoh abstraksi pada Java berikut ini mendefinisikan template untuk sebuah kelas (misal Animal), dan implementasi fungsi/metode tertentu ditentukan oleh subkelas - misal makeSound().

```
abstract class Animal
{
    private String name;
     public Animal(String name)
  {
  this.name = name;
  }
     public abstract void makeSound();
     public String getName()
  {
  return name;
  }
  }
  class Dog extends Animal
  {
    public Dog(String name) { super(name);
  }
     public void makeSound()
    {
     System.out.println(getName() + " barks");
   }
  }
   class Cat extends Animal
  {
    public Cat(String name) { super(name);
```

```
}
    public void makeSound()
  {
    System.out.println(getName() + " meows");
}
}
 public class AbstractionExample
{
  // Main Function
  public static void main(String[] args)
  {
     Animal myDog = new Dog("Buddy");
    Animal myCat = new Cat("Fluffy");
    myDog.makeSound();
    myCat.makeSound();
  }
}
```

Inheritance

Konsep ini memungkinkan suatu kelas dapat mewarisi fitur (atribut/variabel dan fungsi/method) kelas lainnya, atau membuat kelas baru berdasarkan kelas yang sudah ada. Sebuah kelas yang mewarisi dari kelas lain dapat menggunakan kembali fitur kelas tersebut. Tetapi perlu diingat, implementasi pewarisan hanya objek subkelas yang dibuat, bukan superkelas. Beberapa jenis pewarisan pada Java antara lain pewarisan tunggal, bertingkat, hirarki, dan multiple. Contoh: kelas bernama Mahasiswa mewariskan atribut dan perilaku/operasi/fungsinya pada obyek mahasiswa bernama Budi (atribut: nim, nama, prodi; fungsi: isi KRS, perwalian).

Berikut adalah contoh sederhana yang menggunakan pewarisan untuk mendefinisikan kelas dasar Vehicle dan turunannya Car.

```
// Definisi kelas induk Vehicle
class Vehicle
 {
   private String brand;
   private int year;
   public Vehicle(String brand, int year)
   {
      this.brand = brand;
      this.year = year;
   }
   public void honk()
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
        {
            System.out.println("Tuut, tuut!");
        }  public String getBrand()
         {
            return brand;
        }
        public int getYear()
        {
            return year;
        }
    }
// Definisi kelas turunan Car yang mewarisi dari Vehicle
class Car extends Vehicle
{
    private int doors;
    public Car(String brand, int year, int doors)
    {
        super(brand, year); // Memanggil konstruktor dari kelas induk
        this.doors = doors;
    }
    public void displayInfo()
    {
     System.out.println("Brand: " + getBrand() + ", Year: " + getYear() + ", Doors: " +
doors);
    }
}
//Menggunakan Objek di Main Class
public class Main
{
    public static void main(String[] args)
    {
        Car myCar = new Car("Toyota", 2021, 4);
        myCar.honk();        // Memanggil metode dari kelas induk
        myCar.displayInfo(); // Output: Brand: Toyota, Year: 2021, Doors: 4
    }
}
```

Kelas Induk (**Vehicle**) mendefinisikan sifat umum dan metode yang berhubungan dengan kendaraan, seperti brand, year, dan metode honk(). Kelas Turunan **(Car)** mewarisi dari kelas Vehicle dan menambahkan sifat khusus seperti jumlah pintu (doors). Kelas ini juga menggunakan konstruktor dari Vehicle untuk menginisialisasi atribut yang diwarisi. Di dalam fungsi main, objek Car dibuat dan metode yang diwarisi serta yang baru didefinisikan dapat dipanggil. Contoh ini menunjukkan bagaimana pewarisan memungkinkan penggunaan ulang kode dan memudahkan pengelolaan dan pemeliharaan kode dalam aplikasi yang lebih besar, dengan memisahkan aspek-aspek umum ke dalam kelas induk dan aspek spesifik ke dalam kelas turunan.

Polimorfisme

Konsep ini memungkinkan untuk mendefinisikan satu antarmuka dan memiliki banyak implementasi. Polimorfisme atau banyak bentuk dapat dianalogikan ibarat perempuan sekaligus sebagai istri, ibu, dan karyawati; yaitu orang yang sama mempunyai peran/perilaku yang berbeda dalam situasi yang berbeda. Dapat juga diibaratkan sebuah operasi dengan nama yang sama, tetapi jika diberikan pada obyek atau kelas yang berbeda akan mengakibatkan operasi yang berbeda. contoh : nama operasi “membuka” - membuka berkas, membuka jendela, membuka koran, membuka percakapan.
Berikut adalah contoh polimorfisme dalam mencari luas beberapa macam bangun datar, menggunakan kelas abstrak Shape dengan dua kelas turunan, Rectangle dan Circle, yang masing-masing mengimplementasikan metode area():

```
//Definisi Kelas Abstrak dan Kelas Turunan
abstract class Shape
{
   abstract double area();
}
class Rectangle extends Shape
{
   private double width;
   private double height;
   public Rectangle(double width, double height)
   {
      this.width = width;
      this.height = height;
   }
   @Override
   double area()
  {
      return width * height;
  }
}
class Circle extends Shape
{
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    private double radius;
    public Circle(double radius)
{
        this.radius = radius;
}
    @Override
    double area()
{
        return Math.PI * radius * radius;
    }
}
//Menggunakan Objek dalam Main Class
public class PolymorphismExample
{
    public static void main(String[] args)
{
        Shape rect = new Rectangle(10, 5);
        Shape circle = new Circle(7);
        printArea(rect);   // Output: Area: 50.0
        printArea(circle); // Output: Area: 153.93804002589985
   }
    public static void printArea(Shape shape)
{
        System.out.println("Area: " + shape.area());
    }
}
```

Kelas Abstrak **(**Shape**)** m**e**ndefinisikan metode abstrak area() yang harus diimplementasikan oleh semua kelas yang mewarisi Shape. Kelas Turunan **(**Rectangle**,** Circle**)** merupakan implementasi khusus dari metode area() sesuai dengan properti geometri masing-masing bentuk. Polimorfisme terdapat dalam main, objek Rectangle dan Circle disimpan sebagai referensi Shape. Ketika metode printArea dipanggil, versi area() yang tepat dipanggil tergantung pada tipe objek aktual. Hal ini memungkinkan kode lebih modular dan dapat diperluas untuk bangun datar lainnya.

Implementasi ini menunjukkan bagaimana polimorfisme dalam Java memudahkan penanganan berbagai bentuk dengan cara yang seragam sambil mempertahankan detail implementasi masing-masing kelas secara internal.

PBO dalam AI: Menyatukan Konsep Objek dan Kecerdasan

PBO juga dapat diimplementasikan dalam pengembangan aplikasi berbasis AI dan NLP (Ross, 2014). Dalam pengembangan sistem berbasis kecerdasan artifisial (AI), khususnya pada domain Natural Language Processing (NLP), prinsip-prinsip PBO sangat penting untuk menjaga struktur kode yang modular, fleksibel, dan mudah dikembangkan. Dua prinsip utama yang sangat relevan dalam konteks ini adalah enkapsulasi dan polimorfisme. Banyak sistem AI modern dikembangkan dengan pendekatan berorientasi objek karena:

- AI memiliki struktur modular (preprocessing, model, evaluator) yang cocok direpresentasikan sebagai objek
- Library seperti TensorFlow, PyTorch, spaCy, OpenNLP, dan juga Java-based tools memanfaatkan class dan inheritance
- Memudahkan pengujian ulang, integrasi, dan pemeliharaan sistem

Perlu dipahami bahwa PBO bukan hanya penting untuk membuat aplikasi, tetapi juga untuk membangun dan memahami sistem AI yang digunakan secara luas saat ini.

Penerapan konsep enkapsulasi dalam library NLP: Enkapsulasi adalah prinsip menyembunyikan detail implementasi suatu objek dan hanya memperlihatkan bagian yang penting melalui antarmuka publik (public interface). Dalam sistem NLP, proses-proses seperti tokenisasi, lemmatisasi, dan parsing dapat dikemas dalam class-class yang tidak mengekspos kompleksitas internalnya.

Contoh program:

```
public class TextProcessor {
   private Tokenizer tokenizer;
   private Lemmatizer lemmatizer;
   public TextProcessor() {
      tokenizer = new Tokenizer();
      lemmatizer = new Lemmatizer();
   }
   public List<String> process(String sentence) {
      List<String> tokens = tokenizer.tokenize(sentence);
      return lemmatizer.lemmatize(tokens);
   }
}
```

Dengan pendekatan ini, pengguna cukup memanggil *process()*, tanpa harus tahu bagaimana tokenisasi atau lemmatisasi dilakukan. Tokenisasi merupakan proses memecah teks menjadi unit-unit kecil yang disebut token, seperti kata, frasa, atau simbol. Lematisasi adalah proses mengubah kata ke bentuk dasarnya (lema) berdasarkan konteks dan aturan linguistik. Tujuannya adalah untuk menyederhanakan dan menstandardisasi teks agar lebih mudah diproses secara komputasional. Tokenisasi dan lematisasi merupakan dua proses penting dalam NLP.

Penerapan konsep polimorfisme dalam Library NLP: Polimorfisme memungkinkan objek-objek yang berasal dari class turunan yang berbeda untuk merespons metode yang sama dengan cara berbeda. Hal ini memudahkan penggantian komponen tanpa harus mengubah struktur program utama.

Contoh program:

```
public abstract class Tokenizer {
   public abstract List<String> tokenize(String sentence);
}
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
public class EnglishTokenizer extends Tokenizer {
   public List<String> tokenize(String sentence) {
      return Arrays.asList(sentence.split(" "));
   }
}
public class IndonesiaTokenizer extends Tokenizer {
  public List<String> tokenize(String sentence) {
    return Arrays.asList(sentence.split("\\s+"));
   }
}
```

Program di atas menjelaskan secara praktis dan sederhana, bagaimana class-class turunan dalam sistem AI seperti Tokenizer bahasa Inggris dan Indonesia dapat berbagi method yang sama (tokenize()), namun dengan implementasi berbeda. Dapat dilihat bahwa prinsip polimorfisme tidak hanya ada dalam pengembangan aplikasi, tetapi juga dalam AI.

Refleksi dan Latihan Soal:

1. Tuliskan apa saja yang menjadi bagian tersulit dalam memahami konsep PBO (misalnya: inheritance, polymorphism, interface, dll). Unggah tanggapannya pada forum diskusi di platform e-learning. Jika masih ada hal yang belum dipahami, sertakan pertanyaan terbuka untuk didiskusikan bersama.
2. Berikan 1 analogi nyata dari prinsip PBO, misalnya: pewarisan sifat dari orang tua ke anak (pewarisan), atau berbagai kendaraan (motor, mobil, sepeda) punya cara "bergerak" yang berbeda (polymorphism). Jelaskan hubungan antara analogi dan konsep PBO tersebut.
3. Lanjutkan contoh polimorfisme pada program di atas. Buatlah algoritma atau pseudocode untuk menerima input jenis bangun datar (lingkaran, persegi, segitiga), memanggil method, mengitungLuas() yang sesuai secara polimorfik, dan menampilkan hasil luasnya ke pengguna.
4. Apakah konsep pewarisan dan polimorfisme dalam PBO bisa membantu membuat program AI (seperti chatbot atau pengolah bahasa) lebih fleksibel dan mudah dikembangkan? Jelaskan pendapat secara singkat!

Mini Project:

Buat class `Person` dengan atribut `nama`, `alamat`, dan `telepon`. Tambahkan method `printCard()` untuk menampilkan informasi dalam format kartu nama. Buat 2 objek Person dan panggil `printCard()`.

Rubrik penilaian:

- Fungsionalitas dasar (40%)
- Struktur OOP (30%)
- Kerapian kode (20%)
- Dokumentasi singkat (10%)

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 2 Komponen PBO

Pembahasan komponen PBO di sini berfokus pada komponen-komponen yang terdapat di dalam bahasa Java. Komponen/elemen yang dimaksud antara lain class, object, dan method. Dalam dunia kecerdasan buatan (AI) juga dapat ditemui komponen PBO, dimana object juga dibentuk dari class dan method, seperti pada class ModelAI, TextClassifier, atau Tokenizer.

2.1 Class

Class merupakan sekumpulan objek yang memiliki atribut dan perilaku yang sama. Class adalah cetakan / blueprint / prototipe yang ditentukan pengguna dari mana objek dibuat. Misalnya Mahasiswa adalah class sedangkan mahasiswa tertentu bernama Budi adalah objek (Sianipar, 2018)

Class bukan entitas nyata, tapi merupakan template atau cetak biru atau prototipe dari mana objek dibuat. Class merupakan sekelompok variabel dengan tipe data berbeda dan sekelompok fungsi/method. Class tidak menempati memori. Class di Java dapat terdiri dari data member, method, constructor, nested class, dan interface.

Untuk membentuk sebuah class, diawali dengan menggunakan kata kunci (*keyword)* class kemudian diikuti nama kelasnya, dibuka dengan tanda kurung kurawal {, isidariclass, ditutup dengan};. Berikut adalah sintaks class:

```
Class NamaKelas
{
Data Members;
Methods;
};
```

Contoh penerapan class:

```
class Student
{
  // data member (instance variable)
  int id;
  string name;
   public static void main(String args[])
  {
    // creating an object of Student
    Student s1 = new Student();
    System.out.println(s1.id);
    System.out.println(s1.name);
  }
}
```

Deklarasi class secara umum dapat mencakup beberapa hal sebagai berikut:

- Modifiers: class dapat bersifat publik atau memiliki akses default

- Keyword ***Class***: perintah yang digunakan untuk membuat kelas/class.
- Class name: nama class harus diawali dengan huruf (bukan angka atau karakter lainnya)
- Superclass (jika ada): nama kelas induk yang diawali dengan perintah/keyword ***extends***.
- Interfaces (jika ada): daftar antarmuka yang diimplementasikan oleh class dipisahkan koma, dan diawali dengan keyword ***implements***. Sebuah class dapat mengimplementasikan lebih dari satu antarmuka.
- Body: badan/isi class diawali dan diakhiri oleh kurung kurawal {}.

2.2 Object

Objek merupakan unit dasar PBO yang mewakili entitas nyata. Objek adalah turunan dari suatu class yang dibuat menggunakan atribut dan fungsi/metode class-nya. Suatu objek terdiri dari:

- Atribut/state : diwakili oleh atribut suatu objek, yang juga mencerminkan properti suatu objek.
- Perilaku : diwakili oleh fungsi /metode suatu objek, yang juga mencerminkan respon suatu objek dengan objek lainnya.
- Identitas : pemberian nama yang unik pada suatu objek dan memungkinkan satu objek berinteraksi dengan objek lainnya.

Pada saat membuat atau mendeklarasikan suatu objek maka dapat diartikan membuat instance kelas. Semua instance berbagi atribut dan perilaku kelas. Namun nilai atribut tersebut bersifat unik untuk setiap objek. Sebuah kelas tunggal mungkin mempunyai sejumlah instance. Misal sebuah kelas mobil dapat memiliki beberapa objek yaitu beberapa jenis mobil dengan warna, kapasitas dan merk yang berbeda-beda.

Objek merupakan entitas sesuai dengan hal-hal yang ditemukan di dunia nyata. Misalnya, program grafis mungkin memiliki objek seperti “player”, “castle”, dan “menu”. Sistem belanja online mungkin memiliki objek seperti “keranjang belanja”, “pelanggan”, dan “produk”. Berikut Sintaks untuk membuat sebuah objek :

```
NamaKelas namaVariable;
```

Sedangkan untuk mengakses data dan method dalam sebuah class dapat menggunakan sintaks berikut :

```
namaObjek.namaVariable;
namaObjek.namaMethod();
```

Contoh inisialisasi objek pada Java:

```
public class Aplikasi
{
  // sw=software
  static String sw_name;
  static float sw_price;
   static void set(String n, float p)
  {  sw_name = n;
     sw_price = p;
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
  }
   static void get()
   {
      System.out.println("Nama Aplikasi: " + sw_name);
      System.out.println("Harganya ($): "   + sw_price);
  }
   public static void main(String args[])
   {  Aplikasi.set("SmartPLS ", 20.0f);
      Aplikasi.get();
   }
}
```

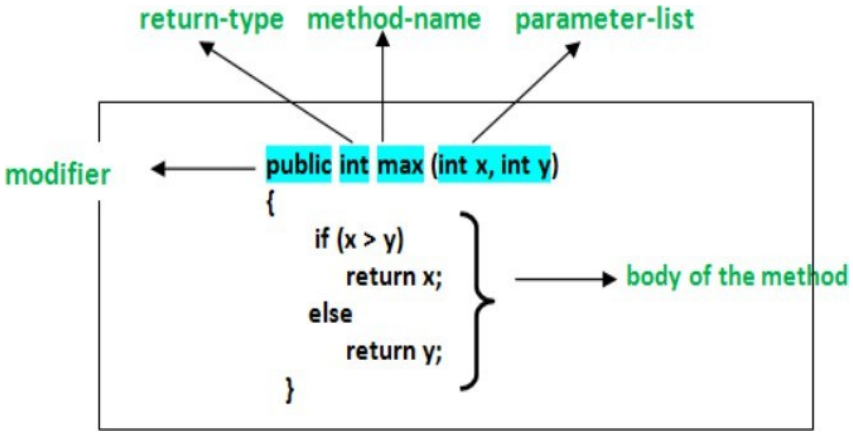
Output

Nama Aplikasi: SmartPLS

Harganya ($): 20.0

2.3 Methods

Methods atau fungsi merupakan sekumpulan pernyataan yang melakukan beberapa tugas tertentu dan mengembalikan hasilnya ke pemanggil. Methods juga dapat melakukan beberapa tugas tertentu tanpa mengembalikan nilai apa pun. Methods memungkinkan kita menggunakan kembali kode programnya tanpa mengetik ulang. Contoh method pada Java:



Gambar 2.1 Method Pada Java

Sumber gambar: www.geeksforgeeks.org

Terdapat 2 jenis method pada Java yaitu predefined dan user-defined. Predefined method merupakan method bawaan atau sudah tersedia di library Java, sedangkan user-defined method merupakan method yang ditentukan

dan dimodifikasi sesuai kebutuhan oleh pengguna. Selain itu terdapat 2 cara untuk membuat method yaitu instance dan static. Berikut penjelasannya:

- Instance: Akses data instance menggunakan nama objek yang dideklarasikan di dalam kelas.

```
void method_name()
{
 body // instance area
}
```

- Static: Akses data statis menggunakan nama kelas yang dideklarasikan di dalam kelas dengan perintah / keyword ***static*** .

```
static void method_name()
{
  body // static area
}
```

Kode lengkap instance dan static dapat dilihat pada program berikut ini:

```
public class VariabelContoh
{
   static int statis = 0; // Variabel statis
   int instance; // Variabel instance
public void hitungLokal()
   {
      int lokal = 10; // Variabel lokal
      System.out.println("Lokal: " + lokal);
      System.out.println("Instance: " + instance);
      System.out.println("Statis: " + statis);
   }
 public static void main(String[] args)
   {
      VariabelContoh obj1 = new VariabelContoh();
      VariabelContoh obj2 = new VariabelContoh();
      obj1.instance = 5;
      obj2.instance = 15;
      VariabelContoh.statis = 100; // Mengubah variabel statis
      obj1.hitungLokal();
      obj2.hitungLokal();
   }
}
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Output:

Lokal: 10

Instance: 5

Statis: 100

Lokal: 10

Instance: 15

Statis: 100

Untuk lebih memahami peran class dan objek melalui studi kasus simulasi perpustakaan digital, berikut disajikan class diagram dan kode programnya. Sistem Peminjaman Buku. Sistem ini menyimpan daftar buku dan memungkinkan pengguna untuk meminjam buku jika tersedia.

Gambar 2.2 Class Diagram Perpustakaan Sederhana

Kode simulasi perpustakaan digital dimulai dengan pembuatan class Buku dengan atribut judul, penulis, tersedia, dan method infoBuku(). Kemudian diikuti dengan class Perpustakaan dan class Main sebagai berikut:

```
class Buku {
   String judul, penulis;
   boolean tersedia;
   public Buku(String judul, String penulis) {
      this.judul = judul;
      this.penulis = penulis;
      this.tersedia = true;
   }
   void infoBuku() {
      System.out.println(judul + " oleh " + penulis + " - " + (tersedia ? "Tersedia" : "Dipinjam"));
```

```
    }
}
class Perpustakaan {
    ArrayList<Buku> daftarBuku = new ArrayList<>();
    void tambahBuku(Buku b) {
        daftarBuku.add(b);
    }
    void pinjamBuku(String judul) {
        for (Buku b : daftarBuku) {
            if (b.judul.equalsIgnoreCase(judul) && b.tersedia) {
                b.tersedia = false;
                System.out.println("Buku '" + judul + "' berhasil dipinjam.");
                return;
            }
        }
        System.out.println("Buku tidak tersedia.");
    }
}
public class Main {
    public static void main(String[] args) {
        Perpustakaan p = new Perpustakaan();
        p.tambahBuku(new Buku("Laskar Pelangi", "Andrea Hirata"));
        p.tambahBuku(new Buku("Bumi", "Tere Liye"));
        p.pinjamBuku("Bumi");
        p.daftarBuku.forEach(b -> b.infoBuku());
    }
}
```

Kode program di atas digunakan untuk membangun simulasi sederhana sistem perpustakaan, yang terdiri atas class Buku, Perpustakaan, dan Main. Masing-masing class memiliki peran yang menunjukkan penggunaan class, objek, dan method secara terpadu.

Class Buku berfungsi sebagai cetak biru (blueprint) dari sebuah entitas buku. Di dalamnya terdapat tiga atribut, yaitu judul, penulis, dan tersedia. Konstruktor Buku digunakan untuk menginisialisasi objek dengan nilai awal, di mana status ketersediaan buku disetel ke nilai true secara default. Hal ini mencerminkan prinsip enkapsulasi, di mana data internal objek diatur melalui mekanisme konstruktor dan tidak diakses secara langsung dari luar. Selain itu, class ini juga memiliki method infoBuku(), yang menampilkan informasi terkait buku, termasuk status apakah buku tersedia atau sudah dipinjam.

Class Perpustakaan berperan sebagai pengelola kumpulan objek buku. Objek-objek Buku disimpan dalam struktur data ArrayList. Class ini menyediakan method tambahBuku() untuk menambahkan buku ke dalam daftar, dan pinjamBuku() untuk mencari serta memproses peminjaman berdasarkan judul. Pencarian dilakukan secara

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

iteratif dan hanya akan berhasil jika buku ditemukan dan masih tersedia. Jika kondisi tersebut terpenuhi, status tersedia pada objek buku akan diubah menjadi false. Apabila tidak ditemukan atau buku sudah dipinjam, maka sistem memberikan notifikasi bahwa buku tidak tersedia.

Class Main merupakan titik awal eksekusi program. Pada bagian ini, objek Perpustakaan dibuat dan beberapa buku ditambahkan ke dalam daftar. Setelah itu, salah satu buku disimulasikan untuk dipinjam. Untuk menampilkan seluruh data buku yang tersimpan, digunakan pendekatan ekspresi lambda *(`b -> b.infoBuku()`)* yang menunjukkan pemanfaatan sintaksis modern dalam Java guna melakukan iterasi terhadap koleksi objek. Pendekatan tersebut merujuk pada cara modern dalam Java untuk menulis kode yang lebih ringkas dan ekspresif**,** khususnya saat melakukan operasi terhadap koleksi data seperti ArrayList.

Struktur kode tersebut memberikan gambaran yang utuh tentang keterkaitan antara konsep class sebagai definisi entitas, objek sebagai instansiasi data nyata, dan method sebagai perilaku dari objek. Melalui program tersebut, dapat diamati bagaimana sebuah sistem dibangun secara modular dan bagaimana interaksi antar objek menjadi dasar dari logika program secara keseluruhan.

Refleksi dan Latihan Soal:

Sistem Informasi Akademik (SIAKAD) merupakan salah satu contoh aplikasi menggunakan konsep PBO, yang digunakan untuk mengelola berbagai data akademik di institusi pendidikan. SIAKAD mencakup fitur utama seperti manajemen data mahasiswa, dosen, mata kuliah, jadwal kuliah, dan nilai.

Analisis konsep PBO dalam SIAKAD:

1. Identifikasi minimal tiga class yang dapat digunakan untuk merepresentasikan komponen dalam sistem tersebut.
2. Tentukan atribut dan metode yang sesuai untuk masing-masing class yang telah diidentifikasi.
3. Jelaskan bagaimana hubungan antar class yang dibuat dapat mencerminkan konsep pewarisan (inheritance) dan enkapsulasi (encapsulation) dalam PBO.
4. Buat algoritma atau pseudocode sederhana yang mencerminkan hubungan antara class-class yang telah diidentifikasi, serta bagaimana class tersebut dapat digunakan dalam program utama untuk mengelola data akademik.

Mini-Project: Perpustakaan v1 (CRUD sederhana)

Buat class `Buku` dengan atribut `judul`, `penulis`, dan `tahunTerbit`. Buat class `Perpustakaan` yang menyimpan daftar buku (ArrayList). Tambahkan method untuk menambah, menghapus, dan menampilkan daftar buku. Implementasikan menu sederhana (tambah, hapus, tampilkan). Mahasiswa membuat minimal 3 buku dan menampilkannya.

Rubrik penilaian:

- Fungsi CRUD dasar berjalan (40%)
- Struktur OOP benar (30%)
- Kerapian kode (20%)
- Dokumentasi singkat (10%)

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 3 Mengenal Java

Bahasa pemrograman Java merupakan salah satu bahasa yang mendukung paradigma PBO, selain bahasa Python, C++, Delphi, dan PHP. Bahasa pemrograman Java awalnya dibuat oleh James Arthur Gosling atau yang biasa dikenal dengan nama James Gosling saat masih bergabung di Sun Microsystems, yang saat ini merupakan bagian dari Oracle dan dirilis tahun 1995. Nama JAVA berasal dari kopi yang digiling langsung atau yang dinamakan sebagai kopi tubruk yang merupakan kopi kesukaan Gosling. Ada yang berpendapat bahwa kopi yang satu ini berasal dari Pulau Jawa sehingga tidak heran jika nama bahasa pemograman ini yakni 'Java' yang dalam bahasa Indonesia berarti 'Jawa'.

Bahasa pemrograman java memiliki keunggulan sebagai berikut :

Penggunaan bahasa pemograman yang sederhana: untuk membantu pengguna di dalam menggunakan suatu bahasa pemograman yang sederhana. jika membandingkan antara java dengan bahasa pemograman yang lain. Java tentu memiliki tingkat kesederhanaan yang lebih baik.

Berorientasi pada Objek (Object Oriented): Ketika suatu program hanya fokus pada suatu objek. Maka program komputer yang berjalan bisa saling berkomunikasi satu sama lainnya di dalam suatu kelompok objek.

Bersifat Multiplatform (dapat dijalankan di berbagai jenis sistem operasi): Java dikenal memiliki moto "Write Once, Run Anywhere". Bahasa pemograman yang dapat digunakan untuk semua jenis sistem operasi yang ada. karena java termasuk ke dalam Platform Independence. Pada saat membuat sebuah file, maka file tersebut sudah bisa digunakan untuk sistem operasi apapun.

Bagi pemula, proses mengunduh paket bahasa pemrogramman Java mungkin masih membingungkan, karena banyak versi Java SDK (Software Development Kit) yang tersedia, seperti Java SE (St rd Edition), Java ME (Micro Edition, kadang disebut Mobile Edition) atau Java EE (Enterprise Edition).

Untuk pemula disarankan menggunakan Java SE, bukan berarti Java SE punya kemampuan terbatas. SE disini lebih kepada paket fundamental Java, yang mencakup kelas-kelas yang mendukung pengembangan Java Web Services dan untuk Java Platform Enterprise Edition (Java EE).

Online Java compiler merupakan alternatif yang dapat digunakan untuk membuat dan menjalankan program Java tanpa harus menginstal paket Java. Online Java compiler menyiapkan teks editor atau IDE-nya terlebih dahulu. Salah satu online Java compiler dapat diakses di sini: https://www.jdoodle.com/online-java-compiler

3.1 Dasar-dasar Java

Seperti halnya bahasa pemrograman, elemen-elemen bahasa Java juga meliputi identifier, keyword, tipe data, operator, dan ekspresi. Java merupakan bahasa yang peka terhadap huruf besar-kecil (case sensitivity), contohnya identifikasi AB, Ab, aB , dan ab dianggap hal yang berbeda di Java. Penamaan sebuah class juga harus diawali dengan huruf (tidak boleh angka atau karakter selain huruf).

Beberapa istilah seputar perangkat lunak Java yang perlu diketahui:

- JVM (Java Virtual Machine): JVM merupakan jantung dari platform Java. JVM adalah pihak yang bertanggung jawab untuk mengeksekusi program Java menjadi bahasa mesin untuk diproses oleh processor. JVM mampu menerjemahkan code-code Java ke hampir semua platform. JVM membuat Java menjadi "*write once, run everywhere*" alias multi-platform.
- JRE (Java Runtime Environment): JRE inilah yang memungkinkan sebuah program Java dapat berjalan di mesin . JRE ini mengeksekusi binary-binary dari class-class dan mengirimnya ke JVM untuk diproses lagi ke prosesor. Setiap JRE pasti memiliki sebuah JVM di dalamnya untuk melakukan pemrosesan selanjutnya
- JDK (Java Development Kit): Seperti namanya, JDK adalah semacam kotak peralatan (kit) yang digunakan untuk men-develop program. JDK ini berguna saat menulis kode program. Seperti halnya JRE, JDK juga memiliki JVM di dalamnya.

Sebelum memulai pemrograman Java secara langsung, penting untuk menginstal dua komponen utama:

- JDK (Java Development Kit) – untuk menjalankan dan meng-compile kode Java.
- IDE (Integrated Development Environment) – alat bantu seperti NetBeans, IntelliJ IDEA, atau VS Code untuk menulis dan menjalankan program Java.

Langkah 1: Instalasi Java JDK

- Buka situs resmi Oracle JDK: https://www.oracle.com/java/technologies/javase-downloads.html
- Pilih versi terbaru dari Java SE Development Kit (JDK).
- Pilih sistem operasi yang sesuai: Windows, macOS, atau Linux.
- Unduh file instalasi, lalu jalankan dan ikuti petunjuknya.
- Setelah instalasi selesai, pastikan path JDK ditambahkan ke Environment Variables:
- Windows: Masuk ke System Properties → Environment Variables → Path → New → tambahkan lokasi bin dari JDK (misalnya: C:\Program Files\Java\jdk-XX\bin)
- Verifikasi instalasi via Terminal atau Command Prompt: java -version javac -version

Langkah 2: Instalasi NetBeans IDE

- Buka: https://netbeans.apache.org/download/
- Unduh installer NetBeans terbaru (biasanya sudah bundling JDK).
- Jalankan installer → pilih Custom Install jika ingin mengatur fitur tambahan.
- Setelah terpasang, buka NetBeans, dan buat proyek baru: File → New Project → Java → Java Application → Next

Online Compiler Java juga bisa digunakan jika tidak bisa menginstal secara lokal. Salah satu online compiler Java dapat akses di sini: https://www.jdoodle.com/ tetapi online compiler tidak cocok untuk program GUI atau penggunaan file eksternal.

Sintaks Java

Pada Java, struktur dasar perintahnya terdiri dari class dan main method, seperti contoh berikut:

```
class PerintahJava  // nama class sama dengan nama file
{   public static void main(String[] args) //main method atau fungsi utama Java
   {     System.out.println("Memulai  Java");
   }
}
```

Fungsi utama / main method pada contoh di atas:

- ***public*** : bersifat publik agar JVM dapat mengeksekusi metode tersebut dari mana saja.
- ***static*** : main method dipanggil tanpa objek. pengubah (modifier) bersifat publik dan statis dapat ditulis dalam urutan apa pun.
- ***void*** : main method tidak mengembalikan nilai apa pun.
- ***main()*** : nama yang dikonfigurasi di JVM. main method harus berada di dalam definisi *class*. Kompiler mengeksekusi kode yang selalu dimulai dari fungsi utama.
- ***String[]*** : main method menerima argumen tunggal, yaitu array bertipe String.
- **Argument args**: adalah array objek string (argument) dari baris-baris perintah yang berfungsi menyimpan nilai argumen yang diberikan dari user melalui CMD atau terminal, sehingga bisa diolah dalam program.

Java Output

Di Java untuk menampilkan hasil eksekusi program dalam tampilan layar menggunakan method print() atau println() pada kelas System dalam struktur paket java.lang:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

- System.out.print(); → Hasil dalam satu baris
- System.out.println(); → hasil ganti baris (dibawahnya)

Penulisan ouput untuk teks:

Untuk menampilkan nilai atau mencetak teks di Java menggunakan tanda kutip ganda (“)

System.out.println("Hello World!");

Penulisan Output untuk angka:

Sedangkan untuk menampilkan nilai atau mencetak angka tidak menggunakan tanda kutip ganda

System.out.println(311);

Komentar pada Java

Bagian dari program yang bersifat sisipan; hanya untuk menjelaskan, dan bukan merupakan bagian operasi.Komentar dibutuhkan agar source code menjadi lebih jelas (well documented). Ada 3 penulisan komentar

- // komentar sebanyak satu baris
- /* komentar lebih dari satu baris */
- /** komentar lebih dari satu baris dan bersifat sebagai official document yang menjelaskan bagaimana kelas dan method public bekerja */

Selain istilah-istilah di atas, beberapa hal lain seputar Java yang perlu diketahui yaitu API (library standar), Java console, Java GUI, Java Applet, dan Javascript.

API (Application Programming Interface) adalah sekumpulan peraturan dan spesifikasi yang memungkinkan satu perangkat lunak untuk berinteraksi dengan perangkat lunak lain. Dalam konteks bahasa pemrograman Java, API sering digunakan untuk menyediakan kumpulan fungsi dan metode yang memungkinkan pengembang untuk melakukan tugas-tugas umum tanpa perlu menulis kode dari awal.

Java Standard Library adalah contoh API yang luas yang digunakan dalam pengembangan Java. Library ini mencakup berbagai fungsi dari manipulasi string dan matematika, hingga manajemen file dan operasi jaringan. Ini adalah bagian dari Java Development Kit (JDK) dan langsung tersedia untuk digunakan dalam aplikasi Java, selengkapnya lihat di sini: https://www.w3schools.com/java/java_packages.asp

Contoh pemanfaatan library, untuk mengelola daftar item dalam aplikasi Java, bisa menggunakan kelas ArrayList dari paket java.util, yang menyediakan metode untuk menambahkan, menghapus, dan mengakses elemen dalam daftar.

```
import java.util.ArrayList; // Import API ArrayList
public class Main {
  public static void main(String[] args)


{
ArrayList<String> cars = new ArrayList<String>(); // Membuat objek ArrayList
    // Menambahkan item ke dalam ArrayList
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
```

```
        cars.add("Mazda");
    // Mengakses elemen
        System.out.println(cars.get(0)); // Output: Volvo
        // Menghapus elemen
        cars.remove(0); // Menghapus elemen pertama (Volvo)
        // Menampilkan semua mobil
        System.out.println("Available cars:");
        for (String car : cars)
        {
            System.out.println(car);
        }
    }
}
```

Pada kode di atas, kelas ArrayList diimpor dari java.util, yang merupakan bagian dari Java API. Membuat ArrayList: Membuat instance ArrayList yang bisa menyimpan string. Operasi ArrayList: Menambahkan, menghapus, dan mengakses item dalam daftar menggunakan metode yang disediakan oleh ArrayList.

Program Java dapat dibuat melalui beberapa cara antara lain melalui konsol, berbasis grafis, web, dan mobile, berikut penjelasannya:

Java Console Application

adalah program Java yang berjalan di terminal atau command line interface dan tidak memiliki antarmuka grafis. Input dan output dilakukan melalui teks di konsol.

Contoh**:**

```
public class HelloWorld
{
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Java GUI Application

adalah program Java yang menggunakan elemen grafis untuk berinteraksi dengan pengguna, seperti windows, buttons, text fields, dan lain-lain. Java menyediakan beberapa pustaka untuk membuat GUI, seperti Swing dan JavaFX.

Contoh dengan Swing**:**

```
import javax.swing.JFrame;
import javax.swing.JLabel;
public class SimpleGUI {
    public static void main(String[] args)
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    {
        JFrame frame = new JFrame("Simple GUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 100);
            JLabel label = new JLabel("Hello, World!", JLabel.CENTER);
        frame.getContentPane().add(label);
        frame.setVisible(true);
    }
}
```

Berbasis web

Program Java yang dirancang untuk dijalankan dalam konteks browser web. Applet memungkinkan program Java diembed dalam halaman web. Namun, penggunaan applet telah berkurang dan tidak lagi didukung di banyak browser modern karena masalah keamanan dan kompatibilitas.

Contoh**:**

```
import java.applet.Applet;
import java.awt.Graphics;
public class SimpleApplet extends Applet
{
    public void paint(Graphics g) {
        g.drawString("Hello, Applet!", 20, 20);
    }
}
```

Menurunnya penggunaan dan dukungan untuk applets Java dalam browser modern, memunculkan berbagai alternatif untuk menggantikan teknologi ini dalam pengembangan web dan aplikasi interaktif. Beberapa pengganti populer untuk applets Java antara lain JavaScript dan HTML5 serta Java Web Start**.**

Java Mobile

Pemrograman Java berbasis mobile dapat dilakukan melalui J2ME atau melalui platform Android. J2ME (Java 2 Platform, Micro Edition) adalah teknologi yang dirancang oleh Sun Microsystems (sekarang bagian dari Oracle) yang memungkinkan aplikasi Java berjalan pada perangkat mobile. Penggunaan J2ME saat ini telah menurun seiring berkembangnya smartphone modern yang mendukung platform lebih canggih seperti Android dan iOS. Meskipun J2ME tidak lagi populer untuk pengembangan aplikasi baru, memahami cara kerjanya bisa memberikan wawasan tentang pengembangan untuk perangkat mobile dengan keterbatasan sumber daya atau dalam konteks pendidikan dan penelitian.

Pengembangan aplikasi mobile berbasis Java saat ini lebih berfokus pada platform Android, karena Android SDK dan platformnya sendiri dibangun menggunakan Java (selain Kotlin yang juga menjadi bahasa utama).

Jadi dapat disimpulkan Java Console cocok untuk tool ringan seperti kalkulator, parser, CLI admin tools. Java GUI dengan JavaFX atau Swing, cocok untuk aplikasi desktop modern. Java Web menggunakan framework seperti Spring Boot, cocok untuk aplikasi enterprise dan layanan berbasis REST API.

3.2 Elemen-elemen Java

Elemen-elemen pada bahasa Java sama dengan elemen-elemen bahasa pemrograman lain seperti C++ atau Python, yang terdiri dari identifier/pengidentifikasi, tipe data, dan operator (Indahyanti, 2020).

1. Identifier

Semua variabel dalam Java harus diidentifikasi dengan nama yang unik. Nama-nama unik tersebut disebut sebagai identifier (pengidentifikasi). Identifier merupakan nama yang diberikan pada method, variable, atau item-item yang didefinisikan oleh user. Dapat berupa satu hingga beberapa karakter yang dimulai dari huruf, underscore, atau tanda dollar, dan bisa diikuti oleh karakter lainnya. Indetifier bersifat case sensitive dan tidak boleh menggunakan Java Keywords (perintah dasar atau bawaan Java).

```
public class Test
{   public static void main(String[] args)
    {    int a = 20;
    }
}
```

Perintah di atas mempunyai 5 identifier, yaitu : Test = class name, main = method name, String = predefined class name, args = variable name, dan a = variable name.

Variabel

Variabel adalah satuan yang dipakai oleh program sebagai basis atau tempat untuk penyimpanan data sementara (memory) yang nantinya dapat di manipulasi oleh user. Ketika membuat sebuah variabel, maka satu 'slot' memory akan disiapkan untuk menampung nilai tersebut. Setiap variabel memiliki nama yang dipakai sebagai identitas dari variabel itu. Setiap variabel di Java memiliki tipe data (Data Type) tertentu, yang menentukan ukuran dan tata letak memori. Saat akan mendeklarasikan sebuah variabel, harus menentukan tipe data serta menggunakan tipe data yang tepat. karena tipe data ini akan mempengaruhi memori yang akan digunakan.

Aturan penamaan variabel di Java ebagai berikut:

- Variabel bisa terdiri dari huruf, angka dan karakter underscore / garis bawah (_).
- Karakter pertama dari variabel hanya boleh berupa huruf , underscore (_) atau dollar sign ($), tidak bisa berupa angka.
- Variabel harus selain dari keyword. Misal kata int tidak bisa dipakai sebagai nama variabel, karena int merupakan keyword untuk menandakan tipe data integer.
- Nama variabel sebaiknya ditulis menggunakan gaya penulisan camelCase, dimana setiap kata juga diawali dengan huruf besar, kecuali kata pertama dan tanpa spasi.
- Variabel bersifat Case sensitive dan tidak memiliki panjang maximal.
  Contoh : Username , username, User_name, _username_var1, $username, UserName.

Penulisan Variabel di Java

- Format penulisan variabel sbb :

```
<tipe data> namaVariabel;
```

- Contoh Membuat variabel kosong bertipe integer:

```
int namaVariabel;
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

- Membuat sekumpulan variabel yang tipe datanya sama:

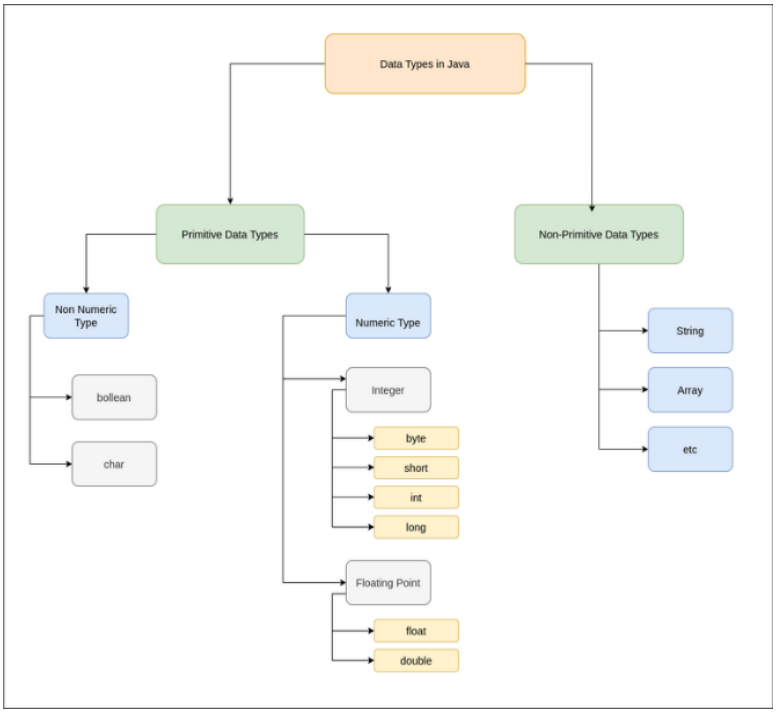
```
int a, b, c;
```

2. Tipe Data

Terdapat 2 tipe data di Java, yaitu:

- Tipe Data Primitif: boolean, char, int, short, byte, long, float, dll
- Tipe Data Non-Primitif atau Tipe Data Objek: String, Array, dll.

Gambar berikut menunjukkan pembagian tipe data di Java, dimana tipe primitif terbagi lagi menjadi tipe numerik dan non numerik (boolean dan char), dan tipe non primitif terbagi menjadi string, array, dll.



Gambar 3.1 Pembagian Tipe Data

Sumber gambar: www.geeksforgeeks.org

Contoh program yang memuat beberapa tipe data primitif dan non-primitif:

```
import java.lang.*;
import java.util.*;
 class CONTOHTIPEDATA {
   public static void main(String ar[])
   {  System.out.println("Tipe Data Primitif\n");
      int x = 10;
      int y = x;
```

```
        System.out.print("Initially: ");
        System.out.println("x = " + x + ", y = " + y);
            y = 30;
         System.out.print("After changing y to 30: ");
        System.out.println("x = " + x + ", y = " + y);
        System.out.println( "Only value of y is affected here "+"because of Primitive Data Type\n");
         System.out.println("Tipe Data Non-Primitif\n");
                int[] c = { 10, 20, 30, 40 };
               int[] d = c;
         System.out.println("Initially");
        System.out.println("Array c: " + Arrays.toString(c));
        System.out.println("Array d: " + Arrays.toString(d));
        System.out.println("\nModifying the value at " + "index 1 to 50 in array d\n");
        d[1] = 50;
         System.out.println("After modification");
        System.out.println("Array c: " + Arrays.toString(c));
        System.out.println("Array d: "+ Arrays.toString(d));
        System.out.println("Here value of c[1] is also affected "+ "because of Non-Primitive Data
Type\n");
    }
}
```

Output:

Tipe Data Primitif

Initially: x = 10, y = 10

After changing y to 30: x = 10, y = 30

Only value of y is affected here because of Primitive Data Type

Tipe Data Non-Primitif

Initially

Array c: [10, 20, 30, 40]

Array d: [10, 20, 30, 40]

Modifying the value at index 1 to 50 in array d

After modification

Array c: [10, 50, 30, 40]

Array d: [10, 50, 30, 40]

Here value of c[1] is also affected because of Non-Primitive Data Type

3. Operator

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Terdapat beberapa jenis operator di Java, antara lain:

- Arithmetic Operators :
  operator yang digunakan untuk operasi sederhana pada tipe data primitif, seperti perkalian, penambahan dll (* , / , % , + , –)
- Assignment Operator :
  digunakan untuk memberikan nilai pada variabel apa pun. operator penugasan ini dapat digabungkan dengan operator lain untuk membuat versi pernyataan yang lebih pendek, misal a += 5 untuk menggantikan pernyataan a = a+5
- Relational Operators:
  Operator ini digunakan untuk memeriksa hubungan seperti sama dengan, lebih kecil, lebih besar, dll (== , != , >= , <=). Operator ini mengembalikan hasil boolean setelah perbandingan, dan banyak digunakan dalam pernyataan perulangan serta pernyataan bersyarat (if-else)
- Logical Operators:
  Operator ini digunakan untuk melakukan operasi logis seperti AND (&&), OR (||), NOT (!). Fungsi tersebut mirip dengan gerbang AND, OR, dan NOT dalam sistem digital.
  Contoh program menggunakan operator logika:

```
import java.io.*;
class LOGIKA {
    // Main Function/Method
  public static void main (String[] args) {
    // Logical operators
    boolean x = true;
    boolean y = false;
        System.out.println("x && y: " + (x && y));
    System.out.println("x || y: " + (x || y));
    System.out.println("!x: " + (!x));
  }
}
```

Output:

x && y : false

x || y : true

!x : false

Kasus sistem rekam medis sederhana di bawah ini, menggabungkan pemakaian operator logika dengan method sesuai proses bisnis rekam medis. Dalam sistem informasi rumah sakit, data pasien harus dikelola secara aman. Tidak semua data bisa diakses sembarangan, dan setiap perubahan data harus dikontrol melalui method yang sesuai. Class diagram konseptual terdiri dari class Pasien dengan atribut nama, umur, diagnosa; dan method getNama(), getUmur(), setDiagnosa(). Berikut kode selengkapnya:

```
class Pasien {
  private String nama;
  private int umur;
  private String diagnosa;
```

```
    public Pasien(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
        this.diagnosa = "Belum diperiksa";
    }
    public String getNama() {
        return nama;
    }
    public int getUmur() {
        return umur;
    }
    public void setDiagnosa(String diagnosaBaru) {
        if (diagnosaBaru != null && !diagnosaBaru.isEmpty()) {
            this.diagnosa = diagnosaBaru;
        }
    }
    public String getDiagnosa() {
        return diagnosa;
    }
}
```

Pada program di atas, enkapsulasi diterapkan dengan menjadikan atribut bersifat private. Akses data dilakukan hanya melalui method getter dan setter, serta terdapat validasi pada setDiagnosa untuk mencegah perubahan yang tidak diinginkan.

4. Java Keyword (Kata Kunci)

Program Java mempunya beberapa kata kunci yang tidak boleh digunakan sebagai nama variabel, nama class, dan yang lainnya. Kata kunci tersebut sebagai berikut:

abstract	finally	public
boolean	float	return
break	for	short
byte	goto	static
case	if	super
catch	implements	switch
char	import	synchronized
class	instanceof	this
const	int	throw
continue	interface	throws
default	long	transient
do	native	try
double	new	void
else	package	volatile
extends	private	while
final	protected	

Selain menggunakan Java, konsep class dan object juga ada di berbagai bahasa pemrograman lainnya. Berikut perbandingan singkat untuk melihat kesamaan konsep meski sintaks berbeda.

Java:

```
class Person {
   private String name;
   public Person(String name) { this.name = name; }
   public void greet() { System.out.println("Hi, " + name); }
}
```

Python:

```
class Person:
   def __init__(self, name):
      self.name = name
   def greet(self):
      print(f"Hi, {self.name}")
```

C#:

```
class Person {
   private string name;
   public Person(string name) { this.name = name; }
   public void Greet() { Console.WriteLine($"Hi, {name}"); }
}
```

Refleksi dan Latihan Soal:

1. Pada saat kondisi apa dibutuhkan JRE dan atau JDK ?
2. Pada perintah: *public static void main(String[] args),* apa yang terjadi jika *static* dihapus?
3. Buat sebuah program kalkulator sederhana yang di dalamnya terdapat operasi menggunakan semua jenis operator!
4. Buat program sederhana untuk mendeteksi aktifitas komputer dan menampilkan status hasil deteksi, berikut ketentuannya:
    - aktivitasTidakBiasa = true
    - gagalLoginBerturut = true
    - Jika kedua kondisi true, cetak "Ancaman Tinggi".
    - Jika salah satu saja true, cetak "Ancaman Sedang".
    - Jika semua false, cetak "Komputer Aman".

Mini-Project: Manajemen Kontak

Buat class `Kontak` dengan atribut `nama`, `telepon`, dan `email`. Buat class `BukuAlamat` yang menyimpan daftar kontak. Tambahkan method untuk menambah dan menampilkan daftar kontak. Buat minimal 3 kontak, lalu tampilkan semua kontak.

Rubrik penilaian:

- Fungsionalitas dasar (40%)
- Struktur OOP (30%)
- Kerapian kode (20%)
- Dokumentasi singkat (10%)

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 4 Struktur Program

4.1 Flow control

Struktur program bersyarat atau flow control mirip dengan pengambilan keputusan dalam dunia nyata. Dalam pemrograman juga menghadapi beberapa situasi di mana sebuah blok kode tertentu dieksekusi ketika kondisi terpenuhi.

Pernyataan seleksi di Java meliputi perintah:

- if
  pernyataan seleksi yang paling sederhana, digunakan untuk memutuskan apakah suatu blok statement akan dieksekusi (hanya jika kondisi terpenuhi)
- if-else
  terdapat blok statement untuk kondisi terpenuhi (benar) maupun tidak (salah), berikut pernyataannya:

```
if (kondisi)
{
   // Jalankan blok ini jika
   // kondisi benar
}
else
{
   // Jalankan blok ini jika
   // kondisi salah
}
```

- nested-if

  terdapat pernyataan if di dalam if (if bersarang), perhatikan struktur program berikut:

```
if (kondisi utama)
   {
     // Jalankan blok ini jika
     // kondisi utama benar
} if (kondisi cabang)
          {   // Jalankan blok ini jika
             // kondisi cabang benar
           }
          else {
              // Jalankan blok ini jika
              // kondisi cabang salah  }
else {

    // Jalankan blok ini jika
   // kondisi utama salah
     }
```

- if-else-if

  adalah struktur kontrol yang digunakan untuk menangani lebih dari dua kondisi secara berurutan. Jika kondisi pertama tidak terpenuhi, program akan memeriksa kondisi berikutnya, dan seterusnya. *if-else-if* umum digunakan dalam penilaian skala (grading sistem) atau klasifikasi data.

  Struktur:

```
if (kondisi1) {
   // Blok kode jika kondisi1 benar
```

} else if (kondisi2) {

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    // Blok kode jika kondisi2 benar
} else if (kondisi3) {
    // Blok kode jika kondisi3 benar
} else {
    // Blok kode jika semua kondisi di atas salah
}
```

Contoh:

```
if (nilai >= 90) {
    System.out.println("A (Sangat Baik)");
} else if (nilai >= 80) {
    System.out.println("B (Baik)");
} else if (nilai >= 70) {
    System.out.println("C (Cukup)");
} else {
    System.out.println("D  (Kurang)");
}
```

• switch-case

switch-case adalah struktur percabangan yang digunakan untuk memeriksa nilai variabel terhadap beberapa kemungkinan kasus secara langsung. Berbeda dengan if-else-if, switch-case lebih efisien dalam kasus yang melibatkan banyak perbandingan nilai tetap.

Struktur:

```
switch (ekspresi) {
    case nilai1:
        // Blok kode jika ekspresi == nilai1
        break;
    case nilai2:
        // Blok kode jika ekspresi == nilai2
        break;
    case nilai3:
        // Blok kode jika ekspresi == nilai3
        break;
    default:
        // Blok kode jika tidak ada nilai yang cocok }
```

Contoh:

```
switch (hari) {
    case 1:
        System.out.println("Senin");
        break;
    case 2:
        System.out.println("Selasa");
        break;
    case 3:
        System.out.println("Rabu");
        break;
    default:
        System.out.println("Hari tidak valid");
}
```

Gunakan break untuk menghentikan eksekusi setelah menemukan case yang cocok, default digunakan jika tidak ada case yang cocok.

• jump: break, continue, return

Jump statement digunakan untuk mengontrol alur eksekusi dalam perulangan atau fungsi.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

break → Menghentikan loop atau switch-case secara langsung.

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        break; // Loop berhenti ketika i == 3
    }
    System.out.println("Nilai i: " + i);
}
```

Output:

```
Nilai i: 1
Nilai i: 2
```

continue → Melewati iterasi saat kondisi tertentu terpenuhi, tetapi tetap melanjutkan loop berikutnya.

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue; // Melewati iterasi saat i == 3
    }
    System.out.println("Nilai i: " + i);
}
```

Output:

```
Nilai i: 1
Nilai i: 2
Nilai i: 4
Nilai i: 5
```

return → Mengembalikan nilai dalam sebuah fungsi dan menghentikan eksekusi fungsi tersebut.

```
public class Contoh {
    public static void main(String[] args) {
        System.out.println(tambah(5, 3));
    }

    public static int tambah(int a, int b) {
        return a + b; // Mengembalikan hasil penjumlahan
    }
}
```

Output:

```
8
```

Perintah break digunakan untuk keluar dari loop atau switch-case, continue digunakan untuk melewati satu iterasi dalam loop, dan return digunakan untuk mengembalikan nilai dari fungsi dan menghentikan eksekusi fungsi.

4.2 Looping

Looping digunakan untuk menjalankan blok kode secara berulang selama kondisi tertentu masih terpenuhi. Java memiliki beberapa jenis perulangan, yaitu:

• **for loop** : digunakan ketika jumlah iterasi sudah diketahui.

contoh

```
for (int i = 1; i <= 5; i++)
{
   System.out.println("Iterasi ke-" + i);
}
```

Output:

```
Iterasi ke-1
Iterasi ke-2
Iterasi ke-3
Iterasi ke-4
Iterasi ke-5
```

Enhanced for loop (juga disebut **for-each**) : khusus digunakan untuk mengakses elemen koleksi (array, list, dsb) tanpa harus pakai indeks. contoh:

```
int[] angka = {1, 2, 3, 4, 5};
for (int num : angka) {
   System.out.println("Angka: " + num);
}
```

• **while loop:** digunakan ketika jumlah iterasi tidak pasti dan bergantung pada kondisi.
contoh

```
int i = 1;
while (i <= 5)
{
   System.out.println("Iterasi ke-" + i);
   i++;
}
```

Output:

```
Iterasi ke-1
Iterasi ke-2
Iterasi ke-3
Iterasi ke-4
Iterasi ke-5
```

• **do-while loop : s**erupa dengan while, tetapi selalu menjalankan satu iterasi pertama, meskipun kondisi false.
contoh

```
int i = 1;
do
{
   System.out.println("Iterasi ke-" + i);
   i++;
}
while (i <= 5);
```

Output:

```
Iterasi ke-1
Iterasi ke-2
Iterasi ke-3
Iterasi ke-4
Iterasi ke-5
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Selain teknik di atas, loop pada Java juga mengenal **nested loop**, yaitu teknik dimana sebuah loop berada di dalam loop lainnya, yaitu membuat sebuah perulangan di dalam perulangan lain. Ini sangat berguna untuk memproses data dua dimensi seperti: matriks (baris × kolom), pola piramida angka atau bintang, game papan (board games), akses elemen dalam array n dimensi, dan kasus lainnya.

Struktur umum nested loop (**for dalam for**):

```
for (int i = 0; i < jumlahBaris; i++)
{     // loop luar: baris
   for (int j = 0; j < jumlahKolom; j++)
 {   // loop dalam: kolom
      // aksi untuk setiap baris dan kolom
   }
}
```

Loop pertama (luar) digunakan untuk mengontrol baris, dan loop kedua (dalam) untuk mengontrol kolom pada setiap baris. Berikut beberapa contoh penerapan struktur nested loop, perhatikan indentasi (spasi) di dalam kode sangat penting untuk membaca nested loop. Pastikan variabel loop (i dan j) tidak saling tumpang tindih, serta jumlah kurung kurawal buka dan tutup yang harus sama (berlaku untuk semua struktur program).

Contoh Matriks Angka

1 2 3

1 2 3

1 2 3

```
//program matriks
public class MatriksAngka
{
   public static void main(String[] args)
  {
     for (int i = 1; i <= 3; i++) { // loop baris
        for (int j = 1; j <= 3; j++) { // loop kolom
           System.out.print(j + " ");
    }
        System.out.println(); // pindah baris setelah selesai satu baris
     }
   }
 }
```

Contoh Pola Piramida Bintang

```
*
**
***
****
*****
```

Loop luar (i) menentukan berapa banyak baris. Loop dalam (j) mencetak bintang sebanyak nomor barisnya.

```
//program piramida
public class PiramidaBintang
{
   public static void main(String[] args)
{
      for (int i = 1; i <= 5; i++) { // loop baris
         for (int j = 1; j <= i; j++) { // loop kolom (jumlah bintang = nomor baris)
            System.out.print("*"); }
      }
   }
}
```

Dari penjelasan di atas, dapat disimpulkan for digunakan jika jumlah perulangan diketahui. Gunakan while jika jumlah perulangan tidak pasti. Gunakan do-while jika menginginkan minimal satu kali eksekusi sebelum pengecekan kondisi. Serta gunakan nested loop untuk membaca perulangan dua dimensi seperti matrik, piramida dan sejenisnya. Untuk melengkapi pembahasan teknik perulangan, dapat akses materi lain di sini: https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/?ref=lbp

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Refeksi dan Latihan Soal:

1. Sebuah pusat perbelanjaan memiliki sistem tarif parkir progresif sebagai berikut:
   - Tarif mobil pada 1 jam pertama sebesar Rp 5.000. Setiap jam berikutnya: Rp 3.000 per jam.
   - Tarif motor pada 1 jam pertama sebesar Rp 3.000. Setiap jam berikutnya: Rp 2.000 per jam.

   Buat program if-else-if untuk menghitung total tarif parkir berdasarkan jenis kendaraan dan durasi parkir. Buat program yang meminta input jenis kendaraan dan lama parkir, lalu tampilkan tarif yang harus dibayar.
2. Terdapat kebijakan pajak penghasilan sebagai berikut:
   - Pendapatan ≤ Rp 5.000.000: Bebas pajak.
   - Rp 5.000.001 – Rp 10.000.000: Pajak 5%.
   - Rp 10.000.001 – Rp 20.000.000: Pajak 10%.
   - Rp 20.000.001 – Rp 50.000.000: Pajak 15%.
   - Rp 50.000.001 ke atas: Pajak 20%.

   Buat program switch-case untuk menghitung pajak berdasarkan kategori pendapatan. Tambahkan validasi input: jika pengguna memasukkan angka negatif, tampilkan pesan error.
3. Sebuah ATM hanya memperbolehkan pengguna melakukan maks 3x percobaan memasukkan PIN. Jika pengguna salah PIN sebanyak 3x, akun akan terblokir. Buat program untuk membatasi jumlah percobaan memasukkan PIN maksimal 3 kali. Jika pengguna berhasil memasukkan PIN yang benar, tampilkan pesan "Akses Diterima". Jika salah 3 kali, tampilkan pesan "Akun Terblokir", dan hentikan program menggunakan break. Gunakan continue untuk memberikan kesempatan memasukkan PIN jika masih ada percobaan tersisa.
4. Bagaimana jika ingin menampilkan piramida angka terbalik dengan jumlah baris dimasukkan oleh pengguna? Syarat dan kondisi apa yang harus diperhatikan?

Mini-Project: Sistem Kasir Sederhana

Buat class `Produk` dengan atribut `nama`, `harga`, dan `jumlah`. Buat class `Kasir` dengan method untuk menghitung total belanja. Tambahkan method untuk menambah produk ke daftar belanja. Simulasikan transaksi dengan minimal 3 produk, lalu tampilkan total harga.

Rubrik penilaian:

- Perhitungan total belanja benar (40%)
- Struktur OOP (30%)
- Kerapian kode (20%)
- Dokumentasi singkat (10%)

Bab 5 Struktur Data Berbasis Objek

Pada bab ini dibahas bagaimana konsep struktur data dipadukan dengan prinsip pemrograman berorientasi objek. Struktur data berfungsi sebagai wadah untuk menyimpan dan mengelola kumpulan data agar dapat diakses dan diproses dengan lebih efisien. Dalam konteks Java, struktur data tidak hanya berupa tipe dasar seperti array, tetapi juga dapat dikembangkan dalam bentuk objek yang merepresentasikan kumpulan data tertentu.

5.1 Array

Array adalah struktur data yang digunakan untuk menyimpan sekumpulan nilai dengan tipe data yang sama. Array dapat memiliki ukuran tetap dan tidak dapat diubah setelah deklarasinya (array statis), atau ukuran dinamis yang dapat berubah selama runtime (array dinamis). Array merupakan salah satu struktur data dasar yang sangat penting dalam pemrograman Java, dan sering digunakan dalam berbagai konteks untuk menyimpan dan mengelola data.

Java menyediakan berbagai struktur data seperti array dan ArrayList untuk menyimpan sekumpulan data (Kreher and Stinson, 2020). Array merupakan salah satu struktur data dasar yang sangat penting dalam pemrograman Java, dan sering digunakan dalam berbagai konteks untuk menyimpan dan mengelola data.

Berikut adalah cara untuk mendeklarasikan, menginisialisasi, dan menggunakan array dalam Java:

```
// Deklarasi array dengan ukuran tertentu
TipeData[] namaArray = new TipeData[ukuran];
// Contoh: Deklarasi array integer dengan ukuran 5
int[] angka = new int[5];
// Inisialisasi array dengan nilai awal
TipeData[] namaArray = {nilai1, nilai2, nilai3, ...};
// Contoh: Inisialisasi array integer
int[] angka = {10, 20, 30, 40, 50};
// Mengakses elemen array berdasarkan indeks
TipeData nilai = namaArray[indeks];
// Contoh: Mengakses elemen array integer
int nilaiPertama = angka[0];
// Mengubah nilai elemen array berdasarkan indeks
namaArray[indeks] = nilaiBaru;
// Contoh: Mengubah nilai elemen array integer
angka[1] = 25;
// Mendapatkan panjang array (jumlah elemen)
int panjangArray = namaArray.length;
// Contoh: Mendapatkan panjang array angka
int panjangAngka = angka.length;
// Menggunakan loop for
for (int i = 0; i < namaArray.length; i++)
{
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    // Lakukan sesuatu dengan setiap elemen array
}
// Contoh: Iterasi melalui array angka
for (int i = 0; i < angka.length; i++)
{
    System.out.println(angka[i]);
}
// Menggunakan loop for-each (enhanced for loop)
for (TipeData elemen : namaArray)
{
    // Lakukan sesuatu dengan setiap elemen array
}
// Contoh: Iterasi melalui array angka menggunakan for-each
for (int nilai : angka)
{
    System.out.println(nilai);
}
```

Program di atas memberikan contoh perintah untuk deklarasi dan inisialisasi array, akses dan manipulasi array, serta perulangan menggunakan array. Dalam Java, array memiliki indeks yang dimulai dari angka 0, yang dapat dideklarasikan dengan dua cara sebagai berikut:

```
int[] angka = new int[5];
```

Pernyataan di atas adalah membuat array angka dengan 5 elemen bertipe int. Nilai awal dari semua elemen akan mengikuti default tipe data, yaitu 0 untuk int. Deklarasi di bawah ini berbeda, yaitu membuat sekaligus inisialisasi nilai awal. Array angka langsung berisi 5 nilai pada saat dibuat. Ini lebih ringkas dan sering digunakan ketika data sudah diketahui sejak awal.

```
int[] angka = {10, 20, 30, 40, 50};
```

Setiap elemen array dapat diakses menggunakan indeks. Indeks pertama dimulai dari 0, berikut perintak aksesnya:

```
int nilaiPertama = angka[0]; // mengambil nilai 10
```

Untuk mengubah elemen array, diberikan perintah seperti contoh berikut ini:

```
angka[1] = 25; // mengubah nilai di indeks ke-1 (sebelumnya 20) menjadi 25
```

Untuk mengetahui panjang array digunakan perintah length. Properti .length digunakan untuk mengetahui jumlah elemen dalam array. Ini penting ketika menggunakan perulangan untuk menghindari error indeks di luar batas.

Sedangkan untuk membaca atau memproses seluruh elemen array, dapat digunakan dua jenis perulangan yaitu indexed loop dan enhanced loop seperti contoh berikut.

Perulangan for biasa (indexed loop): memberikan kontrol penuh terhadap indeks, cocok digunakan jika perlu mengetahui posisi elemen.

```
for (int i = 0; i < angka.length; i++)
{
```

```
    System.out.println(angka[i]);
}
```

Perulangan for-each (enhanced for loop): versi ini lebih sederhana dan digunakan ketika hanya perlu mengakses nilai elemen tanpa memperhatikan indeks.

```
for (int nilai : angka)
{
    System.out.println(nilai);
}
```

Array dalam Java memberikan cara yang efisien untuk menyimpan dan mengelola data dalam jumlah banyak dengan tipe yang sama. Dengan memahami cara deklarasi, inisialisasi, pengaksesan, serta penggunaan perulangan di dalamnya, maka pengolahan data menjadi lebih terstruktur dan mudah dipelihara.

5.2 ArrayList

ArrayList adalah struktur data yang sangat fleksibel dan sering digunakan dalam pemrograman Java untuk menyimpan dan mengelola sekumpulan data dengan ukuran yang dapat berubah-ubah. ArrayList merupakan salah satu implementasi dari antarmuka List dalam bahasa pemrograman Java. ArrayList memungkinkan penyimpanan elemen-elemen dengan tipe data tertentu secara dinamis, yang berarti ukuran ArrayList dapat berubah seiring waktu.

Berikut ini adalah cara menggunakan ArrayList dalam Java:

```
import java.util.ArrayList;
// Deklarasi ArrayList tanpa inisialisasi
ArrayList<TipeData> namaArrayList = new ArrayList<>();
// Contoh: Deklarasi ArrayList untuk menyimpan angka integer
ArrayList<Integer> angka = new ArrayList<>();
// Menambahkan elemen ke ArrayList
namaArrayList.add(nilai);
// Contoh: Menambahkan angka ke ArrayList
angka.add(10);
angka.add(20);
angka.add(30);
// Mengakses elemen ArrayList berdasarkan indeks
TipeData nilai = namaArrayList.get(indeks);
// Contoh: Mengakses elemen ArrayList angka
int nilaiPertama = angka.get(0);
// Mengubah nilai elemen ArrayList berdasarkan indeks
namaArrayList.set(indeks, nilaiBaru);
// Contoh: Mengubah nilai elemen ArrayList angka
angka.set(1, 25);
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
// Menghapus elemen ArrayList berdasarkan indeks
namaArrayList.remove(indeks);
// Contoh: Menghapus elemen ArrayList angka berdasarkan indeks
angka.remove(1);
// Mendapatkan panjang ArrayList (jumlah elemen)
int panjangArrayList = namaArrayList.size();
// Contoh: Mendapatkan panjang ArrayList angka
int panjangAngka = angka.size();
// Menggunakan loop for
for (int i = 0; i < namaArrayList.size(); i++)
{
   // Lakukan sesuatu dengan setiap elemen ArrayList
}
// Contoh: Iterasi melalui ArrayList angka
for (int i = 0; i < angka.size(); i++)
{
   System.out.println(angka.get(i));
}
// Menggunakan loop for-each (enhanced for loop)
for (TipeData elemen : namaArrayList)
{
   // Lakukan sesuatu dengan setiap elemen ArrayList
}
// Contoh: Iterasi melalui ArrayList angka menggunakan for-each
for (int nilai : angka)
{
   System.out.println(nilai);
}
```

Program di atas, memperkenalkan penggunaan ArrayList dalam Java sebagai bentuk struktur data yang lebih fleksibel dibandingkan array biasa. ArrayList digunakan untuk menyimpan kumpulan data dengan ukuran yang bisa bertambah atau berkurang secara otomatis. Pada contoh di atas, ArrayList<Integer> dibuat untuk menyimpan nilai-nilai bertipe integer. Elemen dapat ditambahkan menggunakan method add(), diakses dengan get(), diperbarui menggunakan set(), dan dihapus melalui remove(). Untuk mengetahui jumlah elemen yang tersimpan, method size() digunakan. Proses pembacaan seluruh elemen dapat dilakukan melalui perulangan for biasa atau for-each, tergantung kebutuhan. Dibandingkan array biasa yang memiliki ukuran tetap, ArrayList lebih praktis karena tidak perlu mengatur ukuran secara manual dan menyediakan banyak method untuk mempermudah pengolahan data.

Pada Java, Array dan ArrayList adalah struktur data yang digunakan untuk menyimpan elemen-elemen, tetapi keduanya memiliki perbedaan signifikan dalam hal sifat dan fungsionalitas. Berikut penjelasan tentang perbedaan Array dan ArrayList serta contohnya:

1. Ukuran

Array: Memiliki ukuran tetap setelah dideklarasikan. Jika ingin menambahkan elemen lebih dari ukuran awal, tidak bisa melakukannya tanpa membuat array baru (Jusuf, 2017)

ArrayList: Ukurannya dinamis, artinya bisa menambahkan atau menghapus elemen tanpa perlu khawatir tentang ukuran awal. ArrayList secara otomatis menyesuaikan ukuran sesuai kebutuhan (Adiputra, 2022)

2. Tipe Data

Array**:** Dapat menyimpan tipe data primitif (seperti int, char, double, dll) dan objek.

ArrayList**:** Hanya dapat menyimpan objek (tidak dapat menyimpan tipe data primitif langsung). Namun, Java menyediakan autoboxing yang secara otomatis mengonversi tipe data primitif menjadi objek (misalnya, int menjadi Integer).

3. Performa

Array**:** Lebih cepat dalam performa karena elemen-elemen disimpan secara langsung dan tidak memerlukan pengaturan ulang ukuran.

ArrayList: Sedikit lebih lambat karena saat ukuran bertambah, ArrayList perlu mengalokasikan ulang memori, menyalin elemen-elemen yang sudah ada, dan menyesuaikan ukuran.

4. Fitur-fitur

Array: Tidak memiliki metode bawaan untuk menambah atau menghapus elemen, mencari elemen, atau memanipulasi elemen lainnya.

ArrayList: Memiliki metode bawaan seperti add(), remove(), get(), contains(), dan lainnya untuk memudahkan pengelolaan data.

Implementasi Array:

```
public class ArrayExample
{
   public static void main(String[] args)
  {
      // Deklarasi dan inisialisasi array
      int[] angka = new int[5];
      // Mengisi elemen array
      angka[0] = 10;
      angka[1] = 20;
      angka[2] = 30;
      angka[3] = 40;
      angka[4] = 50;
      // Mencetak elemen array
      for (int i = 0; i < angka.length; i++)
       {
```

```
            System.out.println("Angka pada indeks " + i + " : " + angka[i]);
        }
    }
}
```

Pada contoh di atas, dibuat array angka dengan ukuran tetap 5, tidak bisa menambahkan elemen tambahan setelah lima elemen.

Contoh ArrayList

```
import java.util.ArrayList;
public class ArrayListExample
{
    public static void main(String[] args)
    {
        // Deklarasi dan inisialisasi ArrayList
        ArrayList<Integer> angka = new ArrayList<>();
        // Menambah elemen ke ArrayList
        angka.add(10);
        angka.add(20);
        angka.add(30);
        angka.add(40);
        angka.add(50);
        // Mencetak elemen ArrayList
        for (int i = 0; i < angka.size(); i++) {
            System.out.println("Angka pada indeks " + i + " : " + angka.get(i));
    }
        // Menambah elemen lain
        angka.add(60); // Ukuran bertambah secara otomatis
        // Menghapus elemen tertentu
        angka.remove(Integer.valueOf(20)); // Menghapus angka 20
        System.out.println("Setelah menambah dan menghapus elemen:");
        for (int num : angka)
          {
            System.out.println(num);
          }
    }
}
```

Pada contoh program di atas, dibuat ArrayList bernama **angka** yang dapat menambah elemen baru tanpa perlu menentukan ukuran awal, dan juga bisa menghapus elemen tertentu dengan menggunakan metode **remove().**

Untuk menguatkan pemahaman tentang ArrayList melalui simulasi aplikasi yang realistis dan aplikatif, berikut diberikan studi kasus manajemen buku perpustakaan. Dalam dunia nyata, perpustakaan digital membutuhkan sistem untuk mencatat daftar buku, menambah buku baru, memperbarui data, dan menampilkan isi koleksi. Semua proses tersebut bisa disimulasikan menggunakan ArrayList. Berikut gambaran struktur class dan kode utamanya:

```
//Struktur class:
class Buku
{
   String judul;
   String pengarang;
   int tahun;
   public Buku(String judul, String pengarang, int tahun)
{
      this.judul = judul;
      this.pengarang = pengarang;
      this.tahun = tahun;
   }
   public void tampilkanInfo()
   {
      System.out.println("Judul: " + judul + ", Pengarang: " + pengarang + ", Tahun: " + tahun);
   }
}
// Kode utama:
import java.util.ArrayList;
import java.util.Scanner;
public class Perpustakaan
{
   public static void main(String[] args)
{
      ArrayList<Buku> daftarBuku = new ArrayList<>();
      Scanner input = new Scanner(System.in);
      int pilihan;
      do
 {
         System.out.println("\nMenu:");
         System.out.println("1. Tambah Buku");
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
System.out.println("2. Lihat Daftar Buku");
System.out.println("3. Hapus Buku");
System.out.println("4. Cari Buku");
System.out.println("0. Keluar");
System.out.print("Pilihan Anda: ");
pilihan = input.nextInt();
input.nextLine(); // membersihkan buffer
switch (pilihan)
{
   case 1:
      System.out.print("Judul: ");
      String judul = input.nextLine();
      System.out.print("Pengarang: ");
      String pengarang = input.nextLine();
      System.out.print("Tahun Terbit: ");
      int tahun = input.nextInt();
      daftarBuku.add(new Buku(judul, pengarang, tahun));
      System.out.println("Buku berhasil ditambahkan.");
      break;
   case 2:
      for (Buku b : daftarBuku)
      {
          b.tampilkanInfo();
      }
      break;
    case 3:
      System.out.print("Masukkan judul buku yang ingin dihapus: ");
      String hapus = input.nextLine();
      daftarBuku.removeIf(b -> b.judul.equalsIgnoreCase(hapus));
      System.out.println("Jika ditemukan, buku sudah dihapus.");
      break;
   case 4:
      System.out.print("Masukkan kata kunci pencarian: ");
      String kunci = input.nextLine();
      for (Buku b : daftarBuku)
      {
          if (b.judul.toLowerCase().contains(kunci.toLowerCase()))
```

```
                {
                    b.tampilkanInfo();
                }
            }
            break;
        case 0:
            System.out.println("Keluar dari sistem.");
            break;
        default:
            System.out.println("Pilihan tidak valid.");
      }
    } while (pilihan != 0);
    input.close();
  }
}
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Refleksi dan Latihan Soal:

1. Bagaimana penggunaan array dan ArrayList dapat membantu dalam mengelola data yang jumlahnya tidak tetap di dunia nyata?
2. Tuliskan satu situasi nyata (selain yang telah dicontohkan dalam latihan) di mana penggunaan ArrayList akan lebih efektif dibanding array biasa.
3. Apakah ada tantangan yang ditemui saat mencoba menyelesaikan latihan soal ini? Jika ya, bagian mana yang paling membingungkan dan mengapa?
4. Buat program untuk menghitung rata-rata nilai ujian dari sejumlah siswa. Nilai setiap siswa akan disimpan dalam sebuah array. Program harus dapat menerima input berupa nilai-nilai siswa, kemudian menampilkan nilai tertinggi, nilai terendah, dan rata-rata dari nilai-nilai tersebut.

Mini-Project: Sistem Inventori Gudang

Buat class `Barang` dengan atribut `nama`, `stok`, dan `harga`. Buat class `Inventori` dengan array untuk menyimpan daftar barang. Tambahkan method untuk menambah barang, mengurangi stok, dan menampilkan daftar barang. Simulasikan inventori dengan minimal 3 barang, lakukan update stok, lalu tampilkan semua barang.

Rubrik penilaian:

- Fungsi update stok berjalan (40%)
- Struktur OOP (30%)
- Kerapian kode (20%)
- Dokumentasi singkat (10%)

Bab 6 Input - Output

Dalam dunia pemrograman, pengambilan dan pengolahan input dari pengguna merupakan aspek penting, baik dalam aplikasi berbasis konsol (teks) maupun antarmuka grafis (GUI). Java menyediakan berbagai cara dan kelas untuk menangani input-output (I/O) sesuai kebutuhan aplikasi, seperti Scanner, BufferedReader, dan JOptionPane. Masing-masing memiliki kelebihan dan digunakan dalam konteks yang berbeda. Berikut penjelasan **Scanner**, **Console**, dan **Buffer** pada Java yang digunakan untuk proses **input output.**

6.1 Jenis Kelas I/O

Scanner

Scanner adalah kelas yang paling umum digunakan untuk membaca input dari pengguna di Java. Kelas ini berada dalam paket java.util dan menyediakan metode untuk membaca berbagai tipe data seperti int, double, String, dll. Scanner mudah digunakan dan mendukung berbagai tipe data. Beberapa contoh di bawah ini merupakan penerapan

Contoh (UMSIDA, 2017):

```
import java.util.Scanner;
public class ScannerExample
{
   public static void main(String[] args)
{
      Scanner scanner = new Scanner(System.in);
      System.out.print("Masukkan nama : ");
      String nama = scanner.nextLine();
      System.out.print("Masukkan usia : ");
      int usia = scanner.nextInt();
      System.out.println("Halo, " + nama + ".  berusia " + usia + " tahun.");
      scanner.close();
   }
}
```

Pada proses I/O dibutuhkan sebuah proses yang disebut parsing (khususnya pada pemakaian Scanner). Parsing dapat berupa parsing otomatis atau manual, berikut perbedaannya:

1. Parsing Otomatis

Parsing otomatis adalah proses di mana input yang diterima dari pengguna akan langsung diubah ke tipe data yang sesuai oleh metode bawaan Java, tanpa memerlukan langkah tambahan dari programmer.

Contoh pada Scanner: nextInt() atau nextDouble() pada kelas Scanner, input pengguna (biasanya berupa teks dari keyboard) secara otomatis diubah menjadi tipe data numerik (int, double, dll.).

Contoh perintah:

```
int usia = scanner.nextInt();
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

2. Parsing Manual

Parsing manual adalah proses di mana programmer melakukan konversi input dari tipe data default (biasanya String) ke tipe data lain yang diinginkan. Contoh pada BufferedReader: hanya membaca input sebagai String. Untuk mengubahnya ke tipe data lain (seperti int atau double), harus menggunakan metode parsing, misalnya: Integer.parseInt() atau Double.parseDouble()

Contoh perintah:

```
String input = reader.readLine(); // Membaca sebagai String
int usia = Integer.parseInt(input); // Parsing manual dari String ke int
```

Console

Console adalah kelas dalam Java yang memungkinkan kita membaca input dari pengguna dengan cara yang lebih aman dibandingkan Scanner. Kelas ini berada dalam paket java.io dan biasanya digunakan untuk aplikasi terminal. Console aman untuk input sensitif seperti readPassword(), dan lebih cepat karena tidak melakukan **parsing*** seperti Scanner.

Contoh:

```
import java.io.Console;
public class ConsoleExample
{
   public static void main(String[] args)
   {
      Console console = System.console();
      if (console != null)
      {
         String nama = console.readLine("Masukkan nama  : ");
         char[] password = console.readPassword("Masukkan password: ");
         System.out.println("Halo, " + nama + ". Password  telah diterima.");
      } else
      {
         System.out.println("Console tidak tersedia.");
      }
   }
}
```

BufferedReader

BufferedReader adalah kelas yang digunakan untuk membaca teks secara efisien. Biasanya digunakan bersama dengan InputStreamReader. Ini bekerja dengan baik untuk membaca data dalam jumlah besar atau dari sumber file. BufferedReader sangat efisien untuk input/output dalam jumlah besar, serta lebih fleksibel untuk memproses data baris demi baris.

Contoh:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
public class BufferedReaderExample
{
   public static void main(String[] args) throws IOException
{
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Masukkan nama  : ");
      String nama = reader.readLine();
      System.out.print("Masukkan usia  : ");
      int usia = Integer.parseInt(reader.readLine());
      System.out.println("Halo, " + nama + ".  berusia " + usia + " tahun.");
   }
}
```

Berikut adalah contoh program sederhana yang menggunakan **BufferedReader** untuk membaca file teks:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class Main {
   public static void main(String[] args)
{
      try
        { // Membuat objek FileReader untuk membaca file
         FileReader fileReader = new FileReader("input.txt");
                // Membuat objek BufferedReader untuk membungkus FileReader
         BufferedReader bufferedReader = new BufferedReader(fileReader);
                // Membaca baris pertama dari file
         String line = bufferedReader.readLine();
                // Menampilkan baris yang dibaca
         System.out.println("Isi file: " + line);
                // Menutup BufferedReader
         bufferedReader.close();
      } catch (IOException e)
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
        {
            System.out.println("Terjadi kesalahan: " + e.getMessage());
        }
    }
}
```

Pada program di atas, untuk membuka file "input.txt" menggunakan FileReader. Objek FileReader dibungkus oleh objek BufferedReader untuk meningkatkan efisiensi membaca teks dengan menyediakan buffering. Program membaca baris pertama dari file menggunakan **readLine()** method dari BufferedReader. Baris yang dibaca kemudian ditampilkan di layar. BufferedReader ditutup dengan menggunakan method **close()** untuk menghindari kebocoran sumber daya. Pastikan file "**input.txt**" berada dalam direktori yang sama dengan file Java atau berikan path absolut jika file tersebut berada di lokasi yang berbeda. Jika file tidak ditemukan atau ada kesalahan saat membacanya, program akan menangkap dan menampilkan pesan kesalahan.

BufferedReader pada umumnya digunakan untuk membaca data masukan, sedangkan untuk menulis output digunakan **BufferedWriter**. Namun, jika ingin menggunakan BufferedReader untuk menulis output, dapat menggunakan PrintWriter yang juga dapat dibungkus oleh BufferedWriter untuk meningkatkan efisiensi. Berikut adalah contoh penggunaan BufferedReader untuk menulis output:

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
public class Main
  {
   public static void main(String[] args)
    {
      try {
         // Membuat objek FileWriter untuk menulis ke file
         FileWriter fileWriter = new FileWriter("output.txt");
          // Membungkus FileWriter BufferedWriter untuk efisiensi penulisan
         BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
        // Membungkus BufferedWriter PrintWriter untuk menulis teks ke file
         PrintWriter printWriter = new PrintWriter(bufferedWriter);
        // Menulis teks ke file
         printWriter.println("Ini adalah baris pertama.");
         printWriter.println("Ini adalah baris kedua.");
       // Menutup PrintWriter secara otomatis menutup BufferedWriter dan FileWriter)
         printWriter.close();
                System.out.println("File 'output.txt' telah berhasil ditulis.");
      }
         catch (IOException e)
```

```
            {
             System.out.println("Terjadi kesalahan: " + e.getMessage());
            }
        }
    }
```

Program di atas membuat objek FileWriter untuk menulis ke file "output.txt". Objek FileWriter dibungkus oleh objek BufferedWriter untuk meningkatkan efisiensi penulisan. Objek BufferedWriter kemudian dibungkus oleh objek PrintWriter untuk menulis teks ke file. Program menggunakan method **println()** dari PrintWriter untuk menulis teks ke file. Setelah selesai menulis, program menutup PrintWriter, dan secara otomatis menutup BufferedWriter dan FileWriter. Jika ada kesalahan saat menulis ke file, program akan menangkap dan menampilkan pesan kesalahan.

6.2 JOption

JOptionPane merupakan bagian dari Java Swing untuk menampilkan dialog kepada pengguna. JOptionPane memungkinkan pengguna untuk membuat dialog sederhana untuk meminta input atau menampilkan pesan. Java Swing adalah bagian dari Java Foundation Classes (JFC) yang digunakan untuk membangun aplikasi desktop berbasis GUI (Graphical User Interface) dalam bahasa pemrograman Java. Swing menyediakan serangkaian komponen GUI yang kaya dan fleksibel yang memungkinkan pengembang untuk membuat antarmuka pengguna yang menarik dan interaktif.

Berikut adalah contoh penggunaan JOptionPane:

```
import javax.swing.JOptionPane;
public class Main
{
   public static void main(String[] args)
{
      // Menampilkan dialog dengan pesan
      JOptionPane.showMessageDialog(null, "Ini adalah pesan dialog.");
      // Meminta input dari pengguna
      String nama = JOptionPane.showInputDialog(null, "Masukkan nama  :");
      // Menampilkan pesan dengan input dari pengguna
      JOptionPane.showMessageDialog(null, "Halo, " + nama + "!");
   }
}
```

Method **showMessageDialog()** pada program di atas digunakan untuk menampilkan pesan kepada pengguna dalam dialog. Method **showInputDialog()** digunakan untuk meminta input dari pengguna dalam dialog. Parameter pertama adalah komponen induk (parent component) di mana dialog akan muncul. Jika null digunakan, maka dialog akan muncul di tengah layar. Parameter kedua adalah pesan atau teks yang akan ditampilkan dalam dialog. Pastikan telah mengimpor paket **javax.swing.JOptionPane** untuk menggunakan JOptionPane. Jika

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

menjalankan program ini dalam lingkungan pengembangan yang mendukung GUI, seperti Java Swing, maka dialog akan muncul saat program dijalankan.

Dalam pengembangan perangkat lunak menggunakan bahasa Java, terdapat berbagai jenis aplikasi yang dapat dibuat. Setiap jenis aplikasi memiliki karakteristik, keunggulan, dan batasan tersendiri, yang disesuaikan dengan kebutuhan sistem atau tujuan dari pengembangannya. Tiga jenis aplikasi utama yang umum dijumpai adalah: aplikasi konsol, aplikasi GUI (Graphical User Interface), dan applet Java.

Aplikasi konsol adalah jenis aplikasi yang berjalan dalam mode teks, biasanya menggunakan standard input dan standard output (via Scanner, System.out.print(), dan sebagainya). Aplikasi jenis ini sangat cocok digunakan untuk tujuan-tujuan seperti skrip otomatisasi, perhitungan batch, tes logika dasar, dan sebagai fondasi awal belajar pemrograman Java karena tidak memerlukan pengaturan tampilan visual yang kompleks.

Sementara itu, aplikasi GUI memungkinkan pengguna untuk berinteraksi melalui elemen visual seperti tombol, kotak teks, label, dan jendela. GUI dibangun menggunakan pustaka Java seperti Swing atau JavaFX. Aplikasi GUI biasanya digunakan untuk aplikasi desktop interaktif, seperti kalkulator, manajemen data, hingga sistem POS (Point of Sale). Desain antarmuka yang intuitif sangat penting dalam aplikasi jenis ini, karena pengguna tidak lagi berinteraksi lewat baris perintah.

Jenis aplikasi ketiga adalah applet Java, yaitu program kecil berbasis Java yang dapat dijalankan di dalam halaman web menggunakan browser atau viewer khusus. Applet ditujukan untuk menambahkan konten interaktif ke halaman web, dan merupakan bagian dari sejarah Java yang mendukung web-based interactivity sebelum era JavaScript modern. Berikut ini adalah contoh program applet Java paling sederhana:

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello, World!", 20, 20);
   }
}
```

Dalam kode tersebut, class HelloWorldApplet mewarisi dari class Applet (yang berada dalam paket java.applet). Method paint(Graphics g) merupakan method penting dalam applet, yang dipanggil secara otomatis oleh sistem saat applet perlu digambar ulang pada layar. Di dalam method ini, digunakan method drawString() dari objek Graphics untuk menggambar teks "Hello, World!" di posisi koordinat (20, 20).

Ketika applet ini dijalankan dalam lingkungan yang mendukung applet viewer, seperti browser lama atau IDE seperti Eclipse/IntelliJ IDEA yang masih menyediakan fitur tersebut, maka teks akan muncul dalam jendela kecil di halaman web atau jendela editor. Untuk menampilkan applet dalam halaman HTML, biasanya digunakan tag <applet> (meskipun kini sudah tidak disarankan), seperti berikut:

```
<applet code="HelloWorldApplet.class" width="300" height="100"></applet>
```

Namun demikian, dukungan terhadap applet telah ditinggalkan oleh mayoritas browser modern, terutama karena alasan keamanan serta munculnya teknologi web yang lebih fleksibel dan aman seperti HTML5, CSS3, dan JavaScript. Saat ini, applet dianggap sebagai teknologi lawas dan sebaiknya hanya digunakan untuk keperluan belajar atau eksperimen lokal.

Di sisi lain, aplikasi GUI berbasis Java seperti yang menggunakan komponen dari Swing (JFrame, JOptionPane, dan lainnya) tidak selalu berjalan dengan baik di lingkungan online Java compiler. Beberapa platform kompilasi online seperti JDoodle, Replit, atau OnlineGDB menjalankan kode Java di lingkungan headless, yaitu sistem tanpa tampilan grafis (GUI). Oleh karena itu, ketika dijalankan di platform tersebut, program yang mencoba

menampilkan GUI akan mengalami java.awt.HeadlessException, yaitu error yang muncul karena sistem tidak mampu membuka window atau komponen visual.

Sebagai contoh, jika sebuah program menggunakan JOptionPane.showInputDialog() untuk mengambil input dari pengguna, maka di platform headless, program akan gagal berjalan meskipun tidak terdapat kesalahan sintaks. Solusinya adalah menjalankan aplikasi GUI tersebut secara lokal di komputer dengan IDE Java seperti NetBeans, Eclipse, atau IntelliJ IDEA, yang memang mendukung tampilan grafis secara penuh.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Refleksi dan Latihan Soal:

1. Buat sebuah program GUI sederhana untuk manajemen buku perpustakaan yang mencakup berbagai jenis kotak dialog input output menggunakan JOptionPane seperti :

   - Input dialog untuk memasukkan data buku (judul, penulis, kategori)
   - Message dialog untuk menampilkan informasi hasil input
   - Confirm dialog untuk konfirmasi sebelum menyimpan atau menghapus data
   - Option dialog untuk memilih kategori buku dari beberapa opsi yang tersediaBuat adaptasi program GUI di atas yang dapat berjalan di konsol.

2. Adaptasikan program GUI di atas ke dalam versi konsol, dengan memanfaatkan kelas Scanner untuk input dan System.out.println() untuk output. Struktur logika program tetap sama, hanya antarmuka yang diubah.
3. Pikirkan dan catat, apakah pembuatan antarmuka menggunakan GUI terasa lebih mudah atau justru lebih kompleks dibanding versi konsol? Dari sisi pengguna, mana yang lebih nyaman digunakan, dan dari sisi programmer, mana yang lebih mudah untuk diuji dan dikembangkan?

Mini-Project: Sistem Reservasi Tiket

Buat class `Tiket` dengan atribut `kode`, `tujuan`, dan `harga`. Buat class `Reservasi` dengan method untuk menambah tiket dan menampilkan daftar tiket. Simulasikan input – output proses reservasi tiket, minimal 3 tiket, lalu tampilkan semua tiket.

Rubrik penilaian:

- Fungsionalitas CRUD dasar (40%)
- Struktur OOP (30%)
- Kerapian kode (20%)
- Dokumentasi singkat (10%)

Bab 7 Method & Constructor

7.1 Method

Method merupakan blok kode yang digunakan untuk mengeksekusi tugas tertentu. Method itu layaknya perintah yang bisa dipanggil supaya objek melakukan sesuatu. Method bisa diartikan juga aksi / fungsi yang bisa dilakukan sebuah objek. Method adalah prosedur atau fungsi yag dimiliki oleh sebuah objek. Sehingga method akan mengolah atau mengubah data/variabel yang ada di dalam objek sesuai dengan operasi yang ditentukan. Dari proses tersebut akan terjadi 3 kemungkinan :

- Mengembalikan suatu nilai akhir
- Mengembalikan nilai yang bersifat sementara untuk kemudian diumpankan ke method lain.
- Tidak mengembalikan nilai sama sekali.

Method bisa diletakkan sebelum method main atau sesudah method main. Method dibagi menjadi beberapa jenis yaitu :

1. Method void.

Void adalah method yang tidak memiliki nilai kembali (return). Biasanya digunakan tidak untuk mencari nilai dalam suatu operasi. Untuk mendeklarasikannya method void harus menembahkan kata kunci void. method ini hanya mengolah nilai dari variabel dan menampilkan hasilnya pada command prompt. Method void Biasa disebut prosedur. Methon void di jalankan pada method main. Pendeklarasian method diletakkan pada blok dari suatu kelas. Deklarasi method void:

```
Void namaMethod()
{
   // statemen atau kode method
}
```

Dimana :

- Void: kosong, method yang tidak memiliki nilai kembali.
- namaMethod(): nama method yang dibuat. Untuk penamaan biasanya pada awal kata ditulis dengan huruf kecil, apabila terdapat lebih dari satu suku kata, huruf awal di kata kedua ditulis kapital.

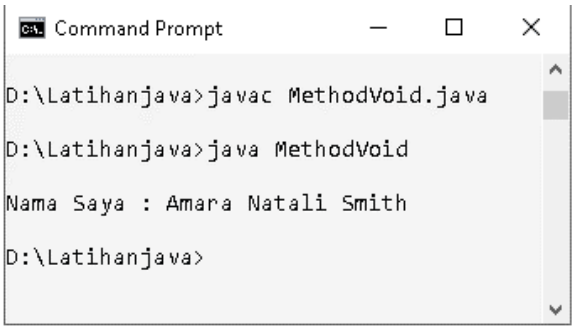
Cara memanggil method void sama dengan memanggil objek. Sintaks sebagai berikut :

```
NamaObjek.NamaVariabel();
```

Contoh Program:

```
public class MethodVoid {
//method void
void nama(String nama){
```

Tampilan Hasil:



2. Method return (non-void)

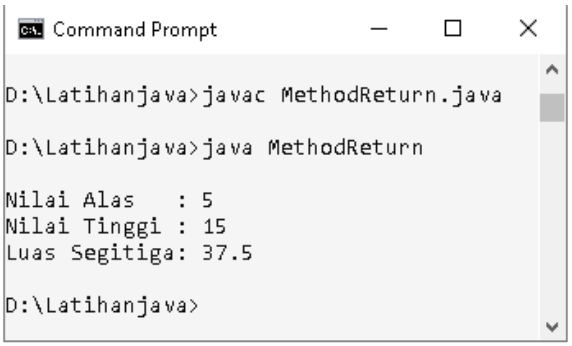
Return adalah method yang mengembalikan nilai secara langsung atau sebuah nilai dari variable. Biasanya method ini disebut fungsi. Pengembalian nilai pada method menggunakan keyword return. tipe data dari nilai yang akan dikembalikan oleh method tersebut harus di definisikan sebelum nama method. Tipe data pada method return harus sama dengan nilai yang ingin dikembalikan. Deklarasi method return:

```
TipeData NamaMethod(){
  //statement atau kode fungsi
  return //Nilai yang ingin
dikembalikan;
```

Contoh program:

```
public class MethodReturn {
    //deklarasi variable
    int alas=5, tinggi=15;
    double Luas;
    public static void main(String[] args){
        //      Membuat      Objek      dari      Class
        MethodReturn data = new MethodReturn();
        System.out.println("\nNilai Alas : "+data.alas);
        System.out.println("Nilai Tinggi : "+data.tinggi);
        System.out.println("Luas Segitiga: "+data.hasil());
    }
    double hasil(){
        Luas = 0.5*alas*tinggi;
        return Luas; //Mengembalikan Nilai dari Luas
    }
}
```

Tampilan hasil:



3. Method Static

Static merupakan modifier yang bisa digunakan pada variable atau method. Apabila menggunakan static pada sebuah variabel ataupun method, untuk memanggilnya tidak perlu membuat sebuah objek dari class tersebut. berbeda dengan method-method sebelumnya, pada method Void dan Return untuk memanggilnya harus membuat objek dari class terlebih dahulu. Deklarasi method :

```
static TypeDataKembalian
namaMethod()
{
   // statemen atau kode method
}
```

```
static TypeData namaMethod()
{
   // statemen atau kode method
}
```

Dimana :

- Keyword "static" akan membuat fungsi dapat dieksekusi langsung, tanpa harus membuat instansiasi objek dari class.
- TypeDataKembalian: merupakan tipe data dari nilai yang dikembalikan setelah fungsi dieksekusi.
- TypeData: Tipe data yang digunakan oleh method.
- namaMethod(): nama method yang dibuat. Untuk penamaan biasanya pada awal kata ditulis dengan huruf kecil, apabila terdapat lebih dari satu suku kata, huruf awal di kata kedua ditulis kapital.

Cara memanggil method :

Method dapat dipanggil dari method main atau dari method yang lainnya. Sintaks sebagai berikut :

```
namaMethod();
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Contoh:

```
public static void main(String[] args)
{
        Salam();//namamethod
}
```

Contoh program:

```
class FirstMethod {
// membuat method Salam
static void Salam()
{
System.out.println("\nHalo Java Apa Kabar");
}
//memanggil method di main
public static void main(String args[]){
//memanggil method Salam
Salam();
}
}
```

Tampilan hasil:

Method Static dengan Parameter

Parameter adalah variabel yang menampung nilai untuk diproses di dalam method. Parameter berperan sebagai input untuk method. Deklarasi method dengan parameter sebagai berikut :

```
static TipeDatapengembalian namaMethod(TipeData namaParameter, TipeData namaParameterLain)
{
   // Statement atau kode method
}
```

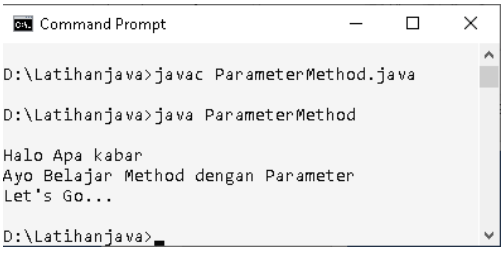
Dimana:

- Parameter ditulis di antara tanda kurung (...);
- Parameter harus diberikan tipe data;
- Bila terdapat lebih dari satu parameter, maka dipisah dengan tanda koma.

Contoh Program:

```
class ParameterMethod {
// membuat method dengan parameter
static void Salam(String helo)
{
System.out.println(helo);
}
//memanggil method di main
public static void main(String args[]){
//memanggil method yang memiliki parameter
Salam("\nHalo Apa kabar");
Salam("Ayo Belajar Method dengan Parameter");
Salam("Let's Go...");
}
}
```

Tampilan hasil:



Method Static dengan return value

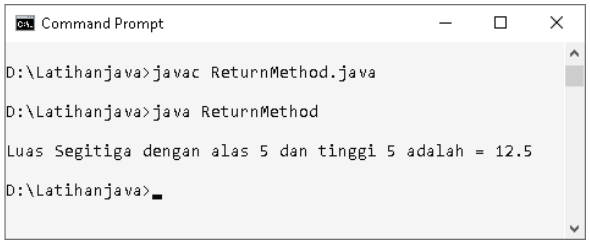
Setelah method memproses data yang diinputkan melalui parameter, untuk selanjutnya method harus mengembalikan nilai agar dapat diolah pada proses berikutnya. Pengembalian nilai pada method menggunakan keyword return.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Contoh program

```
class ReturnMethod {
// membuat method LuasSegitiga
static double LuasSegitiga(int alas, int tinggi)
{
double Luas = 0.5 * alas * tinggi;
return Luas;
}
//memanggil method di main
public static void main(String args[]){
//memanggil method
System.out.println("\nLuas Segitiga dengan alas 5 dan tinggi 5 adalah = " + LuasSegitiga(5,5));
}
}
```

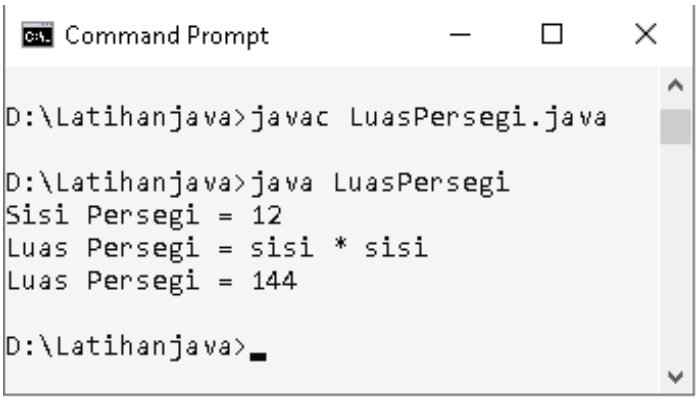
Tampian hasil:



Contoh 2:

```
public class LuasPersegi {
   public static void main(String[] args) {
      int sisi = 12;
                  int Luas=luasPersegi(sisi);
                  System.out.println("Sisi Persegi = " + sisi);
                  System.out.println("Luas Persegi = sisi * sisi");
                  System.out.println("Luas Persegi = " + Luas);
   }
   // membuat method luasPersegi()
               static int luasPersegi(int sisi)
               {
               return sisi * sisi;
               }
   }
```

Tampilan hasil:



Contoh 3:
Apabila objek itu robot, method itu seperti remote atau tombol untuk memberikan perintah pada robot seperti: jalan(), lompat(), bernyanyi()

Contoh method jalan() pada objek robot:

```
class Robot
{
   void jalan()
{
      System.out.println("Robot  jalan-jalan");
   }
}
```

Untuk membuat objeknya maka dibuat kode sebagai berikut:

```
Robot r = new Robot();
r.jalan(); // robot jalan-jalan
```

Contoh lain penggunaan method sekaligus constructor:

```
public class Lingkaran
{
   double jariJari;
   // Constructor untuk menginisialisasi jari-jari lingkaran
   public Lingkaran(double jariJari)
  {
      this.jariJari = jariJari;
   }
   // Method untuk menghitung luas lingkaran
   public double hitungLuas()
  {
      return Math.PI * jariJari * jariJari;
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    }
    public static void main(String[] args)
    {
        // Membuat objek lingkaran dengan jari-jari 5
        Lingkaran lingkaran = new Lingkaran(5);
        // Memanggil method hitungLuas() untuk menghitung luas lingkaran
        double luas = lingkaran.hitungLuas();
        // Menampilkan luas lingkaran
        System.out.println("Luas lingkaran: " + luas);
    }
}
```

Kelas Lingkaran memiliki satu variabel instance (jariJari) dan satu constructor. Constructor digunakan untuk menginisialisasi nilai jari-jari lingkaran. Method **hitungLuas()** digunakan untuk menghitung luas lingkaran berdasarkan nilai jari-jari yang telah diinisialisasi. Pada **main()** method, kita membuat objek Lingkaran, memanggil method hitungLuas() untuk menghitung luas lingkaran, dan menampilkan hasilnya.

7.2 Constructor

Sedangkan constructor merupakan method khusus yang digunakan untuk menginisialisasi objek. Constructor memiliki nama yang sama dengan nama kelasnya dan tidak memiliki tipe kembalian. Sederhananya, constructor seperti perintah khusus saat objek baru dibuat. Contoh program di atas juga telah menggunakan constructor, dimana nama method Lingkaran sama dengan nama class nya.

Contoh kode:

```
class Robot
{
    Robot()
{
        System.out.println("Robot baru dibuat!");
    }
}
Robot r = new Robot();
//Robot baru telah dibuat
```

Pada kasus pembuatan robot di atas, terdapat penggunaan constructor saat merakit yaitu dengan memberi nama dan warna pada robot (mengisi data awal). Sedangkan method saat memberikan aksi pada robot melalui tombol (fungsi) seperti berjalan, menyapa, berlari. Di dalam method & constructor juga sering digunakan perintah „this“. Pada contoh kasus pembuatan robot, this digunakan untuk menyimpan nama milik objek (robot ingat: “nama saya adalah...”, bukan nama dari orang lain), sedangkan nama tanpa this merupakan nama dari parameter yang dikirim.

Contoh:

```
class Robot
{
   String nama;
   Robot(String nama)
  {
      this.nama = nama; // "saya.nama" yang di objek = nama yang dikirim
   }
   void perkenalan()
  {
      System.out.println("Halo, saya " + this.nama);
   }
}
```

Contoh lain penggunaan this pada objek Mahasiswa yang mempunyai atribut Nama dan NIM, serta dapat memperkenalkan diri melalui method.

```
class Mahasiswa
{
   String nama;
   String nim;
   // Constructor
   Mahasiswa(String nama, String nim)
  {
      this.nama = nama; // "this" merujuk ke variabel milik objek
      this.nim = nim;
   }
   // Method
   void perkenalan()
  {
      System.out.println("Halo, nama saya " + this.nama + ", NIM: " + this.nim);
   }
}
Program Utama (Main Class):
public class TesMahasiswa
 {
   public static void main(String[] args)
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
{
 Mahasiswa m1 = new Mahasiswa("Andi", "123456");
    Mahasiswa m2 = new Mahasiswa("Budi", "654321");
    m1.perkenalan();
    m2.perkenalan();
  }
}
```

Output Program:

Halo, nama saya Andi, NIM: 123456

Halo, nama saya Budi, NIM: 654321

Constructor digunakan saat membuat objek new Mahasiswa(...). Sedangkan perintah this.nama dan this.nim merujuk ke variabel milik objek, bukan yang dikirim lewat parameter. Perintrah method perkenalan() digunakan untuk menampilkan data milik objek.

Contoh kasus lain penggunaan method, constructor, dan this pada class Mobil:

```
public class Mobil
{
  String merek;
  int tahunProduksi;
  // Constructor tanpa parameter
  public Mobil()
 {
    merek = "Tidak Diketahui";
    tahunProduksi = 0;
  }
  // Constructor dengan parameter
  public Mobil(String merek, int tahunProduksi)
 {
    this.merek = merek;
    this.tahunProduksi = tahunProduksi;
  }
  // Method untuk menampilkan informasi mobil
  public void infoMobil()
 {
    System.out.println("Merek: " + merek);
    System.out.println("Tahun Produksi: " + tahunProduksi);
  }
```

```
    public static void main(String[] args)
    {
      // Membuat objek mobil tanpa menggunakan constructor
      Mobil mobil1 = new Mobil();
      mobil1.infoMobil();
      // Membuat objek mobil dengan menggunakan constructor
      Mobil mobil2 = new Mobil("Toyota", 2020);
      mobil2.infoMobil();
    }
}
```

Kelas Mobil pada program di atas memiliki dua variabel instance (merek dan tahunProduksi) dan dua constructor. Constructor pertama adalah constructor tanpa parameter yang digunakan untuk menginisialisasi objek dengan nilai default. Constructor kedua adalah constructor dengan parameter yang digunakan untuk menginisialisasi objek dengan nilai yang diberikan. Atribut (Instance Variables) → String merek: menyimpan nama merek mobil. int tahunProduksi: menyimpan tahun produksi mobil. Atribut ini bersifat non-static, sehingga nilainya unik untuk setiap objek Mobil. Perintah **„this“** adalah referensi ke objek saat ini (objek yang sedang dibuat atau digunakan). dalam konteks constructor: this.merek mengacu pada atribut instance merek.

Sedangkan merek (tanpa this) adalah parameter lokal dari constructor. Karena nama parameter dan nama atribut sama, maka perlu this untuk membedakannya. Tanpa this, Java akan menganggap merek = merek; sebagai pengisian variabel lokal ke dirinya sendiri (tidak berguna). Perintah **this** sering digunakan dalam constructor dan method saat Nama parameter sama dengan atribut class, dan ingin merujuk ke objek saat ini dari dalam class. this juga bisa digunakan untuk constructor chaining: memanggil constructor lain di class yang sama misalnya: this(...) Method **infoMobil()** digunakan untuk menampilkan informasi mobil. Pada **main()** method, kita membuat dua objek Mobil, satu menggunakan constructor tanpa parameter dan yang lain menggunakan constructor dengan parameter.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Refleksi dan Latihan Soal:

1. Buatlah sebuah class bernama Mahasiswa dengan ketentuan sebagai berikut:
    - Atribut: nama, nim, nilai
    - Dua constructor: constructor tanpa parameter yang memberikan nilai default untuk setiap atribut, dan constructor dengan parameter untuk mengisi seluruh atribut saat objek dibuat
    - Method statusKelulusan() yang mengembalikan "Lulus" jika nilai ≥ 60, selain itu "Tidak Lulus"
2. Implementasikan kelas Mahasiswa tersebut. Buat 2 objek, masing-masing dengan nilai default, dan nilai dari parameter. Tampilkan status kelulusan untuk masing-masing objek. Tambahkan logika di method agar jika nilai > 90 maka tampilkan "Lulus dengan Pujian".
3. Cermati perbedaan antara dua cara membuat objek menggunakan constructor default dan constructor dengan parameter. Menurut pendapat pribadi, dalam situasi seperti apa constructor dengan parameter lebih sesuai digunakan? Jelaskan pula bagaimana logika tambahan dalam method dapat membuat program terasa lebih realistis atau "manusiawi".

Mini-Project: Rekening Bank (Method & Constructor)

Buat class `Account` dengan atribut `number`, `owner`, dan `balance`. Terapkan constructor overloading: Account(), Account(String number, String owner), Account(String number, String owner, double initialBalance). Tambahkan method: deposit(double amt), withdraw(double amt), printStatement(). Opsional: transfer(Account target, double amt). Buat minimal 2 objek Account dengan constructor berbeda. Lakukan deposit, withdraw, (opsional) transfer. Cetak mutasi singkat dengan printStatement().

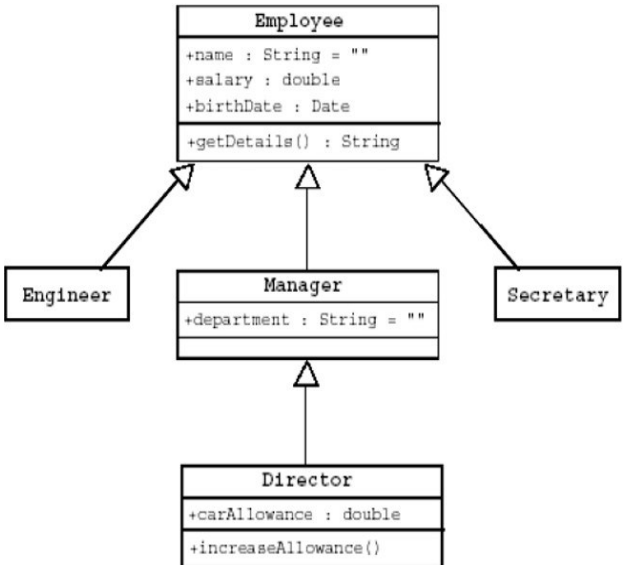
Rubrik penilaian:

- Fungsionalitas (deposit/withdraw/statement) berjalan (40%)
- Penerapan method & constructor (overloading) (30%)
- Kerapian kode & penamaan (20%)
- Dokumentasi singkat (10%)

Bab 8 Pewarisan

Inheritance atau pewarisan adalah salah satu prinsip / konsep dalam PBO yang paling banyak diterapkan. Konsep ini memungkinkan sebuah class (subclass) mewarisi properti dan method dari class lain (superclass). Dengan pewarisan, pengembangan program menjadi lebih efisien karena dapat menghindari pengulangan kode (code duplication). Pewarisan dalam Java dituliskan dengan kata kunci „extends“ (Sianipar, 2018).

Berikut ilustrasi pewarisan pada contoh data karyawan di sebuah perusahaan, dimana atribut-atributnya dapat diturunkan ke manager, secretary, engineer, dan director (semua level tersebut juga termasuk karyawan yang mempunyai atribut dasar yang sama). Atribut tambahan pada kelas manager diperlukan untuk membedakan dari departemen mana manager berasal. Begitu juga dengan tambahan atribut pada director, semua disesuaikan sesuai kebutuhan.



Gambar 8.1 Pewarisan Atribut (Jusuf, 2017)

Pada bab ini dibahas tiga jenis pewarisan yaitu tunggal, bertingkat, dan hirarki. Setiap jenis diberikan contoh program dan penjelasannya.

8.1 Pewarisan Tunggal

Pewarisan tunggal terjadi ketika sebuah subclass hanya mewarisi dari satu superclass. Ini adalah bentuk pewarisan yang paling dasar dan umum dalam Java.

Contoh Program:

```
class Kendaraan
{
   void bergerak()
  {
      System.out.println("Kendaraan sedang bergerak.");
   }
}
class Mobil extends Kendaraan
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
 {
  void klakson()
  {
     System.out.println("Mobil membunyikan klakson.");
  }
}
public class DemoTunggal
{
  public static void main(String[] args)
  {
     Mobil m = new Mobil();
     m.bergerak();
     m.klakson();
  }
}
```

Class Mobil pada program di atas, mewarisi method bergerak() dari superclass Kendaraan. Objek m dari class Mobil dapat mengakses method milik superclass dan method milik dirinya sendiri.

8.2 Pewarisan Bertingkat

Pewarisan bertingkat (multilevel inheritance) terjadi ketika sebuah class mewarisi dari class lain, dan class tersebut juga menjadi superclass bagi class berikutnya.

Contoh Program:

```
class MakhlukHidup
{
  void bernapas() {
     System.out.println("Makhluk hidup bernapas.");
  }
}
class Manusia extends MakhlukHidup {
  void berjalan() {
     System.out.println("Manusia berjalan.");
  }
}
class Mahasiswa extends Manusia {
  void belajar() {
     System.out.println("Mahasiswa belajar.");
  }
}
public class DemoBertingkat {
  public static void main(String[] args) {
     Mahasiswa mhs = new Mahasiswa();
     mhs.bernapas();
     mhs.berjalan();
     mhs.belajar();
  }
}
```

Class Mahasiswa mewarisi semua method dari Manusia dan MakhlukHidup, sehingga objek mhs dapat memanggil ketiga method tersebut.

8.3 Pewarisan Hirarki

Pewarisan hirarki terjadi ketika beberapa subclass mewarisi dari satu superclass yang sama.

Contoh Program:

```
class Hewan
{
   void makan()
  {
      System.out.println("Hewan sedang makan.");
   }
}
class Kucing extends Hewan
 {
   void suara()
  {
      System.out.println("Kucing mengeong.");
   }
}
class Anjing extends Hewan
 {
   void suara()
   {
      System.out.println("Anjing menggonggong.");
   }
}
public class DemoHirarki
  {
   public static void main(String[] args)
    {
      Kucing k = new Kucing();
      k.makan();
      k.suara();
      Anjing a = new Anjing();
      a.makan();
      a.suara();
   }
}
```

Class Kucing dan Anjing mewarisi method makan() dari superclass Hewan. Meskipun memiliki method suara() masing-masing, keduanya berbagi perilaku dasar dari class induk.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Program di bawah ini menunjukkan contoh lain pewarisan. Class Vehicle memiliki atribut brand dan method honk(), sedangkan class Car mewarisi class Vehicle dan menambahkan atribut sendiri yaitu modelName. Di dalam method main(), dibuat objek myCar dari class Car. Objek ini dapat memanggil method honk() dari class Vehicle karena mewarisi method tersebut. Program kemudian mencetak suara klakson dan informasi merek serta model mobil, menunjukkan bahwa objek dari subclass dapat mengakses anggota dari superclass jika visibilitasnya memungkinkan (seperti protected).

```
class Vehicle
{
  protected String brand = "Ford";  // Vehicle attribute
  public void honk() // Vehicle method
   {
    System.out.println("Tuut,  tuut!");
  }
}
class Car extends Vehicle
{
  private String modelName = "Mustang";  // Car attribute
  public static void main(String[] args)
{
    // Create a myCar object
    Car myCar = new Car();
    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();
       System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Dalam Java, modifier akses seperti protected menentukan sejauh mana anggota kelas (seperti variabel atau metode) dapat diakses dari kelas lain. Modifier protected memungkinkan anggota kelas diakses oleh kelas yang berada dalam paket yang sama, subclass (kelas turunan) yang berada di paket yang berbeda.

Sebagai contoh, jika sebuah variabel atau metode dideklarasikan dengan modifier protected, maka kelas lain dalam paket yang sama dapat mengaksesnya secara langsung, kelas turunan di paket lain dapat mengaksesnya melalui pewarisan, dan kelas di luar paket yang bukan subclass tidak dapat mengaksesnya.

Dalam konteks program di atas, variabel **brand** dideklarasikan sebagai protected di kelas Vehicle. Hal ini memungkinkan kelas Car, yang merupakan subclass dari Vehicle, untuk mengakses brand secara langsung. Namun, jika brand dideklarasikan sebagai private, maka hanya kelas Vehicle yang dapat mengaksesnya, dan kelas Car tidak akan memiliki akses langsung, meskipun merupakan subclass. Dengan demikian, penggunaan modifier protected memberikan fleksibilitas dalam desain kelas, memungkinkan akses yang lebih luas dibandingkan private, namun tetap menjaga enkapsulasi dibandingkan public.

Contoh lain program yang mengilustrasikan konsep pewarisan:

```
// Superclass atau kelas induk
class Hewan {
   // Method dalam superclass
   void bersuara()
   {
      System.out.println("Suara  hewan...");
   }
}
// Subclass atau kelas anak yang mewarisi dari kelas Hewan
class Kucing extends Hewan
{
   // Method khusus untuk subclass Kucing
   void miaw()
  {
      System.out.println("Miaw...");
   }
 }
// Subclass lain dari kelas Hewan
class Anjing extends Hewan
{
   // Method khusus untuk subclass Anjing
   void gonggong()
  {
      System.out.println("Gonggong...");
   }
}
// Program utama
public class Main
{
   public static void main(String[] args)
  {
      // Membuat objek dari kelas Kucing
      Kucing kucing = new Kucing();
      // Memanggil method yang diwarisi dari kelas Hewan
      kucing.bersuara();
      // Memanggil method khusus dari kelas Kucing
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
        kucing.miaw();
        // Membuat objek dari kelas Anjing
        Anjing anjing = new Anjing();
        // Memanggil method yang diwarisi dari kelas Hewan
        anjing.bersuara();
        // Memanggil method khusus dari kelas Anjing
        anjing.gonggong();
    }
}
```

Output:

Suara hewan...

Miaw...

Suara hewan...

Gonggong...

Kelas Hewan merupakan kelas induk atau superclass yang memiliki method bersuara(). Kelas Kucing dan Anjing adalah subclass yang mewarisi sifat dan perilaku dari kelas Hewan. Subclass Kucing memiliki method khusus miaw() dan subclass Anjing memiliki method khusus gonggong(). Dalam program utama, objek dari kelas Kucing dan Anjing dibuat dan method yang diwarisi dari kelas Hewan serta method khusus dari masing-masing subclass dipanggil.

Contoh di atas menunjukkan bahwa objek dari kelas Kucing dan Anjing mewarisi method bersuara() dari kelas Hewan, dan kelas Kucing memiliki method khusus miaw() sedangkan kelas Anjing memiliki method khusus gonggong().

8.4 Teknik Override

Overriding method adalah suatu rnekanisme pewarisan class di mana subclass mendefinisikan ulang satu atau lebih method (fungsi) yang ada di supper class. Dalam overriding, subclass bisa mendefinisikan method (fungsi) yang di-override dengan fungsi yang baru sama sekali, atau menambahi method / fungsi yang di-override yang sudah ada dengan fungsinal-fungsinal yang lain (UMSIDA, 2017). Override digunakan untuk memberikan implementasi spesifik pada subclass. Berikut adalah contoh program yang **membutuhkan override**, Contoh ini juga menunjukkan prinsip **polimorfisme** menjadi penting ketika metode di-override.

Contoh penerapan Override dalam sistem penggajian:

Sebuah perusahaan memiliki dua tipe pegawai: **Pegawai Tetap** yang menerima gaji pokok tetap, dan **Pegawai Harian** yang menerima gaji berdasarkan jumlah hari kerja. Superclass Pegawai memiliki metode hitungGaji() yang dioverride oleh subclass PegawaiTetap dan PegawaiHarian untuk memberikan implementasi spesifik.

```
// Kelas induk
class Pegawai
{
   String nama;
```

```
    public Pegawai(String nama)
  {
        this.nama = nama;
    }
    // Metode default untuk menghitung gaji
    public double hitungGaji()
    {
        return 0; // Default
    }
    // Metode untuk menampilkan informasi pegawai
    public void tampilkanInfo()
    {
        System.out.println("Nama: " + nama);
        System.out.println("Gaji: Rp " + hitungGaji());
    }
}
// Subclass PegawaiTetap
class PegawaiTetap extends Pegawai
{
    double gajiPokok;
    public PegawaiTetap(String nama, double gajiPokok)
{
        super(nama);
        this.gajiPokok = gajiPokok;
  }
    @Override
    public double hitungGaji()
{
        return gajiPokok;
    }
}
// Subclass PegawaiHarian
class PegawaiHarian extends Pegawai
{
    double upahPerHari;
    int jumlahHariKerja;
    public PegawaiHarian(String nama, double upahPerHari, int jumlahHariKerja)
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
{
    super(nama);
    this.upahPerHari = upahPerHari;
    this.jumlahHariKerja = jumlahHariKerja;
  }
  @Override
  public double hitungGaji()
  {
    return upahPerHari * jumlahHariKerja;
  }
}
// Kelas utama
public class SistemPenggajian
 {
  public static void main(String[] args)
  {
    // Membuat array pegawai
    Pegawai[] daftarPegawai =
    {
      new PegawaiTetap("Alif", 5000000),
      new PegawaiHarian("Bondan", 150000, 20),
      new PegawaiTetap("Hamid", 7000000),
      new PegawaiHarian("Mediana", 200000, 15)
    };
    // Menampilkan informasi semua pegawai
    for (Pegawai pegawai : daftarPegawai)
    {
      pegawai.tampilkanInfo();
      System.out.println();
    }
  }
}
```

Output:

Nama: Alif

Gaji: Rp 5000000.0

Nama: Bondan

Gaji: Rp 3000000.0

Nama: Hamid

Gaji: Rp 7000000.0

Nama: Mediana

Gaji: Rp 3000000.0

Superclass Pegawai merupakan kelas induk yang memiliki atribut dasar nama dan metode **hitungGaji().** Metode **hitungGaji()** pada superclass memberikan nilai default 0 karena implementasi spesifik diberikan di subclass. **Subclass PegawaiTetap** untuk menghitung gaji hanya berdasarkan atribut gajiPokok. Override metode **hitungGaji()** untuk mengembalikan gajiPokok. **Subclass PegawaiHarian** untuk menghitung gaji berdasarkan jumlah hari kerja (jumlahHariKerja) dan upah per hari (upahPerHari). Override metode hitungGaji() untuk mengembalikan nilai yang sesuai. **Polimorfisme** diterapkan pada referensi Pegawai yang digunakan untuk memanggil metode hitungGaji() di semua objek pegawai (baik tetap maupun harian). Implementasi metode dipilih berdasarkan tipe objek aktual (runtime binding). **Array dari Tipe Superclass**: semua pegawai dimasukkan ke dalam array tipe Pegawai, sehingga program dapat menangani berbagai tipe pegawai tanpa perlu memeriksa tipe objek secara manual.

Dapat disimpulkan bahwa teknik override diperlukan pada kasus di atas untuk beberapa hal berikut:

1. Kebutuhan Spesifik Subclass: setiap subclass memiliki cara yang berbeda untuk menghitung gaji. Override memungkinkan implementasi spesifik tanpa mengubah logika di superclass.
2. Dukungan Polimorfisme: dengan override, dapat menggunakan referensi Pegawai untuk emanggil metode hitungGaji() pada berbagai jenis pegawai tanpa peduli tipe spesifiknya.
3. Misalnya, pada for (Pegawai pegawai : daftarPegawai), metode yang dipanggil adalah hitungGaji() dari subclass yang sesuai.
4. Menghindari Kode Redundansi: tanpa override, maka perlu menulis logika yang terpisah untuk setiap tipe pegawai, yang membuat kode lebih panjang dan sulit dipelihara.
5. Ekspansi yang Mudah: jika perusahaan menambah tipe pegawai baru (misalnya, pegawai kontrak), cukup membuat subclass baru dengan mengoverride hitungGaji() tanpa mengubah kode yang ada.

8.5 Interface dan Abstract Class

Pewarisan dalam Java tidak terbatas pada hubungan antar kelas konkret. Java menyediakan dua fitur penting yang mendukung pewarisan secara lebih fleksibel, yaitu **abstract class** dan **interface**. Keduanya berperan sebagai sarana untuk mendefinisikan perilaku secara abstrak, dengan menetapkan apa yang harus dilakukan tanpa langsung menentukan bagaimana cara melakukannya. Abstract class digunakan untuk menyusun kerangka dasar dari suatu entitas, terutama ketika sebagian perilaku dapat langsung diimplementasikan, namun ada bagian lain yang masih bersifat umum dan perlu ditentukan lebih lanjut oleh kelas turunan. Interface berfungsi sebagai pernyataan bahwa suatu kelas memiliki perilaku atau kemampuan tertentu, tanpa bergantung pada struktur pewarisan yang hierarkis. Interface menekankan pada deskripsi perilaku, bukan pada pewarisan atribut atau implementasi.

Abstract class dan interface menunjukkan bagaimana keduanya mendukung prinsip pewarisan dalam Java. Perbedaan peran antara keduanya mencerminkan pendekatan yang berbeda dalam membangun sistem yang fleksibel, modular, dan mudah dikembangkan.

Contoh kasus sistem transportasi: berbagai kendaraan seperti mobil, motor, dan pesawat, semuanya bisa jalan, dan hanya pesawat yang bisa jalan dan juga terbang.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
// Interface: menyatakan kemampuan terbang
interface DapatTerbang
{
   void terbang();
}
// Abstract Class: kerangka umum untuk semua transportasi
abstract class Transportasi
{
   protected String nama;
   public Transportasi(String nama)
 {
      this.nama = nama;
   }
   // Metode abstrak: harus diisi oleh anaknya
   public abstract void jalankan();
   // Metode umum: bisa langsung dipakai
   public void info()
 {
      System.out.println("Jenis transportasi: " + nama);
   }
}
// Kelas Mobil: mewarisi Transportasi
class Mobil extends Transportasi
 {
   public Mobil() {
      super("Mobil");
 }
   @Override
   public void jalankan()
 {
      System.out.println("Mobil berjalan di jalan raya.");
   }
}
// Kelas Motor: mewarisi Transportasi
class Motor extends Transportasi
 {
```

```
    public Motor()
    {
        super("Motor");
    }
    @Override
    public void jalankan()
{
        System.out.println("Motor melaju di jalan umum.");
    }
}
// Kelas Pesawat: mewarisi Transportasi dan menyatakan bisa terbang
class Pesawat extends Transportasi implements DapatTerbang
{
    public Pesawat()
 {
        super("Pesawat");
    }
    @Override
    public void jalankan()
    {
        System.out.println("Pesawat bergerak di landasan.");
    }
    @Override
    public void terbang()
 {
        System.out.println("Pesawat mulai terbang di udara.");
    }
}
// Kelas utama (main program)
public class DemoTransportasi
 {
    public static void main(String[] args)
    {
        Transportasi mobil = new Mobil();
        Transportasi motor = new Motor();
        Pesawat pesawat = new Pesawat();
        mobil.info();
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
        mobil.jalankan();
        System.out.println();
        motor.info();
        motor.jalankan();
        System.out.println();
        pesawat.info();
        pesawat.jalankan();
        pesawat.terbang();
    }
}
```

Pada contoh di atas menunjukkan pewarisan umum dan pewarisan perilaku tambahan, dimana:

- Transportasi = abstract class, diwarisi oleh Mobil, Motor, Pesawat
- DapatTerbang = interface, menyatakan kemampuan terbang (perilaku tambahan)
- Pesawat = mewarisi dari Transportasi dan mengimplementasikan DapatTerbang (multiple inheritance via interface, karena Java tidak mendukung pewarisan multipel dari class).

Jadi gunakan abstract class saat butuh kerangka umum. Gunakan interface saat ingin menyatakan bahwa suatu objek memiliki kemampuan tertentu, walaupun jenisnya berbeda.Gabungkan keduanya untuk membangun sistem yang fleksibel dan modular

Penerapan konsep pewarisan pada studi kasus lainnya, dapat ditemui pada turunan pegawai pada sebuah kantor, misalnya pegawai tetap dan pegawai kontrak. Meskipun keduanya adalah pegawai, ada atribut dan perilaku khusus yang membedakan mereka. Hirarki Class terdiri dari Class Induk (Pegawai) dan Class Turunan (PegawaiTetap, PegawaiKontrak), berikut kode programnya:

```
class Pegawai {
    String nama;
    double gaji;
    public Pegawai(String nama) {
        this.nama = nama;
    }
    public void tampilkanInfo() {
        System.out.println("Nama: " + nama);
    }
}
class PegawaiTetap extends Pegawai {
    public PegawaiTetap(String nama) {
        super(nama);
        this.gaji = 5000000;
    }
    @Override
    public void tampilkanInfo() {
```

```
        super.tampilkanInfo();
        System.out.println("Status: Tetap\nGaji: " + gaji);
    }
}
class PegawaiKontrak extends Pegawai {
    public PegawaiKontrak(String nama) {
        super(nama);
        this.gaji = 3000000;
    }
    @Override
    public void tampilkanInfo() {
        super.tampilkanInfo();
        System.out.println("Status: Kontrak\nGaji: " + gaji);
    }
}
```

Class Pegawai berperan sebagai superclass, yaitu class induk yang mendefinisikan struktur umum dari semua jenis pegawai dalam sistem. Di dalamnya terdapat atribut dan method yang bersifat generik dan dapat diwariskan, seperti nama dan method tampilkanInfo(). Class ini menjadi dasar bagi class-class turunan, yaitu PegawaiTetap dan PegawaiKontrak, yang masing-masing disebut sebagai subclass karena mewarisi struktur dan perilaku dari Pegawai.

Pada masing-masing subclass, method tampilkanInfo() diimplementasikan ulang menggunakan konsep method overriding. Overriding adalah proses di mana subclass mendefinisikan ulang method yang sudah ada di superclass, dengan tujuan memberikan perilaku yang lebih spesifik sesuai kebutuhan class turunan. Misalnya, PegawaiTetap dan PegawaiKontrak memiliki cara berbeda dalam menampilkan status kepegawaiannya dan nilai gaji, meskipun keduanya berasal dari method yang sama di superclass.

Untuk tetap menjaga hubungan antara subclass dan superclass, digunakan keyword super dalam konstruktor dan pemanggilan method. Pada konstruktor subclass, super(nama) digunakan untuk memanggil konstruktor dari class Pegawai dan menginisialisasi atribut nama yang didefinisikan di superclass. Penggunaan super juga memungkinkan subclass mengakses method dari superclass, baik secara langsung maupun sebagai bagian dari method yang dioverride, seperti super.tampilkanInfo() yang dapat dipanggil terlebih dahulu sebelum menambahkan informasi tambahan spesifik di subclass.

Dengan struktur ini, sistem menjadi lebih modular dan mudah diperluas. Subclass hanya perlu menyesuaikan bagian-bagian yang berbeda, sementara bagian umum tetap dikelola oleh superclass. Ini mencerminkan prinsip OOP seperti inheritance (pewarisan), reusability (penggunaan kembali kode), dan polymorphism dalam bentuk overriding.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Refleksi dan Latihan Soal:

1. Perhatikan potongan kode berikut, kemudian jawab tiga pertanyaan ini: (a) Jika variabel **bahanBakar** diubah menjadi **private** dalam **Kendaraan**, apakah program masih bisa dijalankan? (b) Apa yang terjadi jika **bahanBakar** dijadikan **private**? Jelaskan alasannya. (c) Bagaimana cara terbaik agar **Motor** tetap bisa mengakses **bahanBakar** secara aman?

```
class Kendaraan
 {
  protected String bahanBakar = "Bensin";
 }
 class Motor extends Kendaraan
{
      private String jenis = "Skuter";
      public void info()
          {
            System.out.println(jenis + " menggunakan " + bahanBakar);
          }  }
```

2. Buat sistem klasifikasi hewan dengan ketentuan : Hewan adalah superclass dengan method bergerak(). Burung dan Ikan adalah subclass dari Hewan. Burung menambahkan method terbang(), dan Ikan menambahkan method berenang(). Implementasikan method agar setiap subclass mewarisi bergerak() dan menambahkan perilaku masing-masing. Buat method tampilkanAktivitas() yang mencetak semua aktivitas hewan berdasarkan jenisnya.
3. Setelah menyelesaikan latihan ini, coba renungkan kembali: bagaimana struktur pewarisan seperti ini membantu mengurangi duplikasi kode? Bandingkan juga penggunaan modifier protected, private, dan public dalam hubungan antarclass. Apakah ada perbedaan nyata dalam hasil program atau hanya sebatas pengaturan struktur ?

Mini-Project: Sistem Kepegawaian
Buat class `Pegawai` dengan atribut `nama` dan `gaji`. Buat subclass `Manajer` dengan atribut tambahan `departemen`. Tambahkan method untuk menghitung gaji dan menampilkan informasi pegawai. Buat minimal 2 objek Pegawai dan 1 Manajer, lalu tampilkan informasinya.

Rubrik penilaian:

- Pewarisan (inheritance) diterapkan (40%)
- Struktur OOP (30%)
- Kerapian kode (20%)
- Dokumentasi singkat (10%)

Bab 9 Import & Package

Dalam pemrograman Java, pengorganisasian kode yang baik sangat penting untuk mendukung pengembangan dan pemeliharaan aplikasi berskala besar. Dua konsep penting yang mendukung hal ini adalah import dan package. Package digunakan untuk mengelompokkan kelas-kelas yang memiliki fungsi atau tujuan yang serupa. Pendekatan ini membantu dalam menjaga keteraturan kode serta mencegah konflik penamaan antar kelas.

Import memungkinkan penggunaan kelas atau interface dari package lain tanpa perlu menyertakan nama lengkap package setiap kali kelas tersebut digunakan. Dengan menerapkan konsep package dan import, proyek Java dapat disusun secara modular, terstruktur, dan mudah dikembangkan lebih lanjut, termasuk dalam pemanfaatan pustaka bawaan Java maupun pustaka eksternal. Penggunaan import dan pembuatan package sangat penting dalam pengembangan program Java yang kompleks (Adiputra, 2022). Berikut adalah contoh penggunaan import dan pembuatan package di Java:

Contoh Penggunaan Import:

Kelas atau interface dari package lain dapat digunakan dengan menuliskan pernyataan import.

```
// Import kelas Scanner dari package java.util
import java.util.Scanner;
public class Main
{
   public static void main(String[] args)
{
      // Objek Scanner dibuat untuk membaca input dari pengguna
      Scanner scanner = new Scanner(System.in);
      System.out.print("Masukkan sebuah angka: ");
      int angka = scanner.nextInt();
      System.out.println("Angka yang dimasukkan: " + angka);
   }
}
```

Package dapat dibuat untuk mengelompokkan kelas-kelas tertentu ke dalam direktori yang terorganisasi. Sebagai contoh direktori baru dibuat dengan nama myPackage. Di dalam direktori tersebut, file MyClass.java dibuat dengan isi berikut:

```
// Package myPackage
package myPackage;
public class MyClass
{
   public void myMethod()
  {
      System.out.println("Ini adalah method dari kelas MyClass dalam package myPackage");
   }
}
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Di luar direktori myPackage, file Main.java dibuat:

```
// Import kelas MyClass dari package myPackage
import myPackage.MyClass;
public class Main
{
   public static void main(String[] args)
   {
      MyClass obj = new MyClass();
      obj.myMethod();
   }
}
```

Sebagai ilustrasi, pengembangan aplikasi kalkulator yang mendukung operasi dasar dapat dimodularisasi menggunakan package. Struktur direktori berikut ini dapat diterapkan:

```
project/
├── kalkulator/
│   ├── Penjumlahan.java
│   ├── Pengurangan.java
└── Main.java
```

File Penjumlahan.java:

```
package kalkulator;
public class Penjumlahan
{
   public int tambah(int a, int b)
   {
      return a + b;
   }
}
```

File Pengurangan.java:

```
package kalkulator;
public class Pengurangan
{
   public int kurang(int a, int b)
   {
      return a - b;
   }
```

```
    }
```

File Main.java:

```
    import  kalkulator.Penjumlahan;
    import kalkulator.Pengurangan;
    public class Main
    {
       public static void main(String[] args)
    {
        Penjumlahan p = new Penjumlahan();
        Pengurangan r = new Pengurangan();
        System.out.println("Hasil penjumlahan 5 + 3 = " + p.tambah(5, 3));
        System.out.println("Hasil pengurangan 5 - 3 = " + r.kurang(5, 3));
      }
    }
```

Contoh di bawah ini adalah membuat direktori baru dengan nama myPackage, dalam direktori myPackage, buat file MyClass.java dengan konten berikut:

```
// Package myPackage
package myPackage;
public class MyClass
{
   public void myMethod()
{
      System.out.println("Ini adalah method dari kelas MyClass dalam package myPackage");
   }
}
//Buat file Main.java di luar direktori myPackage dengan konten berikut:
// Import kelas MyClass dari package myPackage
import myPackage.MyClass;
public class Main
{
   public static void main(String[] args)
  {
      // Membuat objek MyClass
      MyClass obj = new MyClass();
      // Memanggil method myMethod dari objek MyClass
      obj.myMethod();
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    }
}
```

Pada program di atas. MyClass berada dalam package myPackage, dan diimpor ke dalam kelas Main untuk digunakan. Program tersebut menunjukkan bagaimana struktur package digunakan untuk mengorganisasi kode dalam proyek Java. File MyClass.java dideklarasikan di dalam direktori bernama myPackage. Baris package myPackage; di bagian atas file menunjukkan bahwa kelas tersebut menjadi bagian dari paket myPackage.

Dengan pendekatan ini, kelas MyClass tidak langsung dikenali oleh kelas lain yang berada di luar paket tersebut. Oleh karena itu, ketika kelas Main diletakkan di luar myPackage, diperlukan pernyataan import myPackage.MyClass; agar kelas Main dapat menggunakan MyClass.

Saat program dijalankan, objek dari MyClass dibuat dalam main() dan method myMethod() dipanggil untuk mencetak pesan ke layar. Proses ini menegaskan bahwa kelas dalam satu paket dapat digunakan di luar paket asalkan di-import dengan benar.

Struktur seperti ini mencerminkan prinsip modularitas dalam PBO / OOP, kode dipisahkan ke dalam unit-unit logis berdasarkan fungsinya. Selain membuat proyek lebih rapi, penggunaan package juga mendukung praktik enkapsulasi, karena memungkinkan pembatasan akses antar bagian program dengan mengatur visibilitas class atau method-nya. Untuk menjalankan program ini melalui terminal, pastikan struktur folder sudah sesuai, dan proses kompilasi dilakukan dengan memperhatikan struktur direktori, seperti:

```
javac myPackage/MyClass.java
javac -cp . Main.java
java Main
```

Penataan program ke dalam paket akan menjadi sangat penting dalam proyek berskala besar, karena membantu dalam menghindari konflik nama class dan mempermudah pemeliharaan kode dalam tim pengembang.

Refleksi dan Latihan Soal:

1. Sebuah sistem informasi akademik terdiri atas fitur utama seperti manajemen data Mahasiswa, Dosen, dan Nilai. Rancanglah struktur package yang sesuai untuk mengorganisasi kode program dari sistem tersebut. Contoh struktur yang mungkin digunakan:

   com.akademik.mahasiswa
   com.akademik.dosen
   com.akademik.nilai
   com.akademik.utils

   Jelaskan bagaimana struktur package yang dirancang agar dapat:
   - Membantu dalam pengelompokan class sesuai tanggung jawab masing-masing
   - Meningkatkan keterbacaan dan pemeliharaan kode
   - Mendukung prinsip encapsulation, yaitu dengan membatasi akses class tertentu hanya dalam lingkup paketnya bila perlu (misalnya dengan modifier default atau private)
2. Setelah menyusun struktur package dan menganalisis strategi impor, coba simpulkan: mengapa keteraturan struktur dan cara impor menjadi hal penting dalam tim pengembang perangkat lunak? Apa dampaknya jika struktur tidak dirancang sejak awal?

Mini-Project Bab 9 — Sistem Akademik dengan Package

Struktur Package:

com.akademik.mahasiswa
com.akademik.dosen
com.akademik.nilai

Spesifikasi:

- Mahasiswa (nim, nama) → info()
- Dosen (nidn, nama) → info()
- Nilai (mataKuliah, angka) → info()
- MainApp → import semua class, buat objek, panggil info().

Tugas:

- Buat semua package & class sesuai struktur.
- Buat minimal 1 objek tiap class.
- Jalankan MainApp untuk menampilkan data.

Rubrik:

- Struktur package & import benar (40%)
- Fungsionalitas method (30%)
- Kerapian kode (20%)
- Dokumentasi (10%)

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 10 Implementasi Prinsip PBO

PBO merupakan paradigma pemrograman yang berfokus pada penggunaan objek untuk merepresentasikan data dan perilaku dalam program. PBO membantu dalam membuat program yang modular, mudah dipelihara, dan dapat digunakan kembali (reusable). Berikut penjelasan dan contoh penerapan 3 prinsip PBO, yaitu pembungkusan, banyak bentuk, abstraksi (prinsip pewarisan telah dibahas pada bab sebelumnya).

10.1 Enkapsulasi

Prinsip enkapsulasi atau pembungkusan ini diperlukan untuk memastikan bahwa data "sensitif" disembunyikan dari pengguna. Untuk itu perlu mendeklarasikan variabel/atribut kelas sebagai privat dan menyediakan metode get dan set publik untuk mengakses dan memperbarui nilai variabel privat.

Get dan ***Set*** : variabel privat hanya dapat diakses dalam kelas yang sama (kelas luar tidak memiliki akses ke sana). Namun, variabel tersebut dapat diakses menggunakan metode get dan set publik. Metode get mengembalikan nilai variabel, dan metode set menetapkan nilainya. https://www.w3schools.com/java/java_encapsulation.asp

Contoh:

```
public class Person
{
 private String name; // private = restricted access
 // Getter
 public String getName()
{
   return name;
 }
 // Setter
 public void setName(String newName)
{
   this.name = newName;
 }
}
```

Metode get mengembalikan nilai dari variabel name. Metode set mengambil parameter (newName) dan menetapkannya ke variabel name. Kata kunci this digunakan untuk merujuk ke objek saat ini. Namun, karena variabel name dideklarasikan sebagai private, maka tidak dapat mengaksesnya dari luar kelas ini.

```
public class Main
 {
 public static void main(String[] args)
 {
   Person myObj = new Person();
```

```
    myObj.name = "Jasmine";  // error
    System.out.println(myObj.name); // error
  }
}
```

Jika variabel dalam suatu kelas dideklarasikan sebagai public, maka variabel tersebut bisa langsung diakses dan dimodifikasi dari luar kelas. Contohnya, jika variabel name di kelas Person bukan private, bisa langsung menulis seperti ini:

```
myObj.name = "Jasmine";
System.out.println(myObj.name);
```

Output : ”Jasmine”

Namun, karena dalam contoh variabel name dideklarasikan sebagai private, maka kita tidak bisa mengaksesnya secara langsung dari luar kelas. Kalau diakses langsung, maka akan muncul 2 error seperti ini:

```
MyClass.java:4: error: name has private access in Person
   myObj.name = "Jasmine";
      ^
MyClass.java:5: error: name has private access in Person
   System.out.println(myObj.name);
                 ^
```

Hal ini disebabkan aturan enkapsulasi melindungi data sensitif agar tidak bisa diubah atau dilihat sembarangan dari luar kelas. Sebagai gantinya, harus menggunakan metode get dan set yang sudah disediakan dalam kelas Person. Contohnya seperti ini:

```
public class Main
{
  public static void main(String[] args)
{
   Person myObj = new Person();
   myObj.setName("Jasmine"); // Mengatur nilai variabel name menjadi "Jasmine"
   System.out.println(myObj.getName()); // Mengambil nilai variabel name
  }
}
```

Output: “Jasmine”

Dengan cara ini, variabel name tetap terlindungi, tetapi masih bisa diakses dan diubah dengan cara yang terkontrol melalui method setName() dan getName().

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

10.2 Polimorfisme

Polimorfisme berarti "banyak bentuk", dan terjadi ketika terdapat banyak kelas yang saling terkait melalui pewarisan. Pewarisan memungkinkan untuk dapat mewarisi atribut dan metode dari kelas lain. Polimorfisme menggunakan metode tersebut untuk melakukan berbagai tugas. Hal ini memungkinkan untuk melakukan satu tindakan dengan cara yang berbeda (Hadiprakoso, 2021).

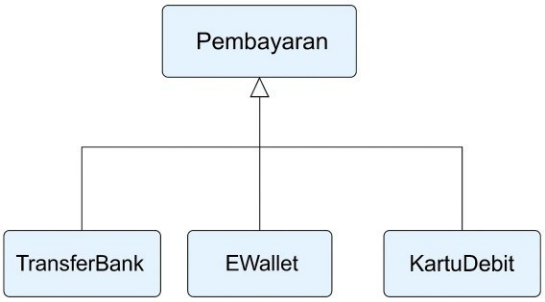
Misalnya, sebuah superkelas Hewan yang memiliki metode animalSound(). Subkelas Hewan dapat berupa Kucing, Anjing, Burung - Dan mereka juga memiliki implementasi suara hewan mereka sendiri (mengeong, menggonggong, dll.).

Selengkapnya dapat dilihat di link ini: https://www.w3schools.com/java/java_polymorphism.asp

```
class Animal
{
  public void animalSound()
{
    System.out.println("The animal makes a sound");
  }
}
class Pig extends Animal
{
  public void animalSound()
{
    System.out.println("The pig says: wee wee");
  }
 }
class Dog extends Animal
{
  public void animalSound()
{
    System.out.println("The dog says: bow wow");
  }
 }
class Main
{
  public static void main(String[] args)
{
    Animal myAnimal = new Animal();
    Animal myPig = new Pig();
    Animal myDog = new Dog();
    myAnimal.animalSound();
```

```
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

Studi kasus sistem pembayaran multi-metode (transfer, e-wallet, debit) di bawah ini menggambarkan penerapan konsep polimorfisme atau "banyak bentuk" sekaligus pewarisan dalam PBO. Dalam sistem e-commerce, pelanggan dapat memilih metode pembayaran berbeda seperti transfer bank, e-wallet, atau kartu debit. Meskipun semua melakukan "pembayaran", setiap metode memiliki logika berbeda. Struktur pewarisan ditampilkan pada diagram berikut ini:



Gambar 10. 1 Struktur Pewarisan Pembayaran

Program berikut menggambarkan polimorfisme dinamis dari sistem pembayaran dengan berbagai cara seperti penjelasan di atas:

```
abstract class Pembayaran {
    abstract void prosesBayar(double jumlah);
}
class TransferBank extends Pembayaran {
    public void prosesBayar(double jumlah) {
        System.out.println("Pembayaran Rp" + jumlah + " via Transfer Bank diproses.");
    }
}
class EWallet extends Pembayaran {
    public void prosesBayar(double jumlah) {
        System.out.println("Pembayaran Rp" + jumlah + " via E-Wallet diproses.");
    }
}
class KartuKredit extends Pembayaran {
    public void prosesBayar(double jumlah) {
```

```
        System.out.println("Pembayaran Rp" + jumlah + " via Kartu debit diproses.");
    }
}
public class Main {
    public static void main(String[] args) {
        Pembayaran p;
        p = new TransferBank();
        p.prosesBayar(50000);
        p = new EWallet();
        p.prosesBayar(75000);
        p = new KartuDebit ();
        p.prosesBayar(120000);
    }
}
```

Class Pembayaran dideklarasikan sebagai class abstrak, yang artinya class ini tidak dapat digunakan secara langsung untuk membuat objek. Class ini hanya berfungsi sebagai kerangka dasar bagi class-class turunan yang mewarisinya. Dalam class Pembayaran, terdapat method abstrak prosesBayar() yang tidak memiliki implementasi, sehingga wajib diimplementasikan secara spesifik oleh setiap subclass-nya, seperti TransferBank, EWallet, dan KartuDebit.

Setiap subclass tersebut mendefinisikan cara kerja prosesBayar() sesuai dengan jenis metode pembayaran masing-masing. Konsep ini mencerminkan prinsip polimorfisme dinamis, di mana satu variabel referensi bertipe Pembayaran (dalam hal ini p) dapat menunjuk ke objek dari subclass mana pun, dan saat method prosesBayar() dipanggil, perilaku yang dijalankan akan disesuaikan dengan tipe objek yang sebenarnya, bukan tipe referensinya. Dengan kata lain, satu referensi p dapat menghasilkan berbagai perilaku berbeda tergantung pada objek nyata yang diwakilinya. Hal tersebut merupakan inti atau esensi dari konsep “satu referensi, banyak perilaku” dalam polimorfisme.

10.3 Abstraksi

Abstraksi data adalah proses menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna. Abstraksi dapat dicapai dengan kelas abstrak atau antarmuka.

```
// Abstract class
abstract class Animal
{
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
```

```
  }
}
// Subclass (inherit from Animal)
class Pig extends Animal
{
  public void animalSound()
{
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
class Main
{
  public static void main(String[] args)
{
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Seperti telah disinggung pada Bab 1 dan Bab 2, bahwa dalam dunia AI juga berlaku prinsip-prinsip PBO seperti polimorfisme dan pembungkusan. Prinsip tersebut sangat berguna untuk membangun sistem yang fleksibel dan mudah dikembangkan. Salah satu penerapan nyata konsep polimorfisme adalah dalam pembuatan sistem klasifikasi teks. Misalnya, kita ingin membuat sistem sederhana untuk mengklasifikasikan teks ke dalam kategori "Olahraga", "Teknologi", dan "Kesehatan". Masing-masing kategori dapat memiliki aturan klasifikasi berbeda, tetapi semuanya tetap menggunakan metode yang sama, yaitu klasifikasi().

Struktur Kelas PBO:

```
// Kelas abstrak
public abstract class TextClassifier {
   public abstract String klasifikasi(String teks);
}
// Klasifikasi untuk kategori Olahraga
public class SportsClassifier extends TextClassifier {
   public String klasifikasi(String teks) {
      if (teks.contains("bola") || teks.contains("sepak")) {
         return "Olahraga";
      }
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
        return "Tidak diketahui";
    }
}
// Klasifikasi untuk kategori Teknologi
public class TechClassifier extends TextClassifier {
    public String klasifikasi(String teks) {
        if (teks.contains("komputer") || teks.contains("AI")) {
            return "Teknologi";
        }
        return "Tidak diketahui";
    }
}
// Klasifikasi untuk kategori Kesehatan
public class HealthClassifier extends TextClassifier {
    public String klasifikasi(String teks) {
        if (teks.contains("vaksin") || teks.contains("kesehatan")) {
            return "Kesehatan";
        }
        return "Tidak diketahui";
    }
}
```

Walaupun setiap class memiliki cara yang berbeda dalam mengklasifikasikan teks, semua class menggunakan method yang sama, yaitu klasifikasi(). Hal ini menunjukkan contoh polimorfisme: method yang sama, tapi perilaku berbeda tergantung objeknya.

```
public void tampilkanKategori(TextClassifier classifier, String teks)
{
    System.out.println("Kategori: " + classifier.klasifikasi(teks));
}
```

Contoh penggunaan:

```
tampilkanKategori(new SportsClassifier(), "Pemain sepak bola mencetak gol.");
tampilkanKategori(new TechClassifier(), "AI semakin canggih setiap tahun.");
tampilkanKategori(new HealthClassifier(), "Vaksinasi penting untuk kesehatan.");
```

Dengan pendekatan ini, kita dapat menambahkan jenis klasifikasi baru tanpa mengubah kode utama. Sistem menjadi modular dan mudah dikembangkan.

Refleksi:
Konteks: Sistem manajemen parkir modern dengan kendaraan Motor, Mobil, dan Bus. Data utama: Plat nomor, waktu masuk, waktu keluar, biaya parkir. Pertanyaan refleksi:

- Enkapsulasi: Bagaimana memastikan data kendaraan (plat, waktu masuk) aman, hanya bisa diakses via getter/setter?

- Polymorphism: Bagaimana satu method hitungBiaya() bisa berbeda perilaku untuk Motor, Mobil, dan Bus?

- Abstraksi: Mengapa class Kendaraan sebaiknya abstract, bukan untuk objek langsung, melainkan untuk diwarisi?

Mini-Project Sistem Parkir
Merancang sistem parkir sederhana dengan: Abstract class Kendaraan (atribut: plat, jamMasuk, jamKeluar, method abstrak hitungBiaya()). Subclass Motor, Mobil, Bus : masing-masing menghitung biaya parkir berbeda. Class utama ParkirApp untuk mencatat kendaraan masuk, keluar, dan menampilkan biaya. Buat minimal 1 objek Motor, 1 Mobil, 1 Bus. Atur jam masuk & keluar, lalu tampilkan biaya parkir. Tunjukkan penggunaan abstraksi, enkapsulasi, dan polymorphism.

Rubrik Penilaian:
- Abstraksi & inheritance diterapkan (30%)
- Polymorphism berjalan (30%)
- Enkapsulasi (getter/setter) benar (20%)
- Kerapian kode & dokumentasi singkat (20%)

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 11 Exception Handling

Dalam pemrograman, error (kesalahan) bisa terjadi kapan saja, terutama saat program dijalankan (runtime). Kesalahan ini bisa berupa kesalahan logika, kesalahan akses file, pembagian dengan nol, atau input yang tidak valid. Jika tidak ditangani, error ini dapat menyebabkan program crash atau menghasilkan hasil yang tidak diharapkan.

Java menyediakan mekanisme Exception Handling (penanganan pengecualian) yang dirancang untuk menangani situasi yang tidak normal ini dengan cara yang terkontrol dan profesional. Dengan menggunakan try-catch-finally, serta throw dan throws, maka dapat mencegah program berhenti secara tiba-tiba, menangani kesalahan secara elegan, dan memastikan proses cleanup atau penutupan tetap dilakukan (Somashekara, Guru and Manjunatha, 2017; Adiputra, 2022). Berikut komponen utama dari blok penanganan kesalahan di Java:

Ada banyak jenis kesalahan yang bisa muncul. Misalnya, ketika kita mencoba membagi angka dengan nol, program akan menghasilkan kesalahan aritmatika. Atau saat mencoba mengakses elemen array yang tidak ada, akan muncul kesalahan indeks. Bahkan ketika pengguna memasukkan teks padahal seharusnya angka, program bisa langsung gagal kalau tidak ditangani. Inilah mengapa penting bagi kita untuk mengenal berbagai jenis exception dan apa penyebabnya.

Dengan mengetahui jenis-jenis exception ini, kita bisa merancang program yang lebih tangguh. Artinya, program tetap bisa berjalan dengan baik walau terjadi kesalahan, dan pengguna pun bisa mendapatkan pesan yang jelas tentang apa yang salah. Jadi, bukan hanya soal membuat program berjalan, tapi juga membuatnya ramah pengguna dan siap menghadapi segala kemungkinan. Beberapa jenis penanganan kesalahan di Java ditampilkan dalam tabel di bawah ini.

Tabel 11. 1 Jenis Penanganan Kesalahan Pada Java
(https://www.w3schools.com/java/java_ref_errors.asp)

Error/Exception	Description
ArithmeticError	Occurs when a numeric calculation goes wrong
ArrayIndexOutOfBoundsException	Occurs when trying to access an index number that does not exist in an array
ClassFormatError	Occurs when a class file cannot be accessed
ClassNotFoundException	Occurs when trying to access a class that does not exist
ConcurrentModificationException	Occurs when an element is added or removed from iterables
FileNotFoundException	Occurs when a file cannot be accessed
IncompatibleClassChangeError	Occurs when there's been a change in a base class after a child class has already been initialized
InputMismatchException	Occurs when entering wrong input (e.g. text in a numerical input)
InterruptedException	Occurs when a Thread is interrupted while waiting/sleeping
InvalidClassException	Occurs when the Serialization runtime observes a problem with a class
IOException	Occurs when an input or output operation fails

NegativeArraySizeException	Occurs when trying to create an array with negative size
NoClassDefFoundError	Occurs when the class is not found at runtime
NoSuchFieldException	Occurs when trying to access a class field/variable that does not exist
NoSuchMethodException	Occurs when trying to access a class method that does not exist
NullPointerException	Occurs when trying to access an object referece that is null
NumberFormatException	Occurs when it is not possible to convert a specified string to a numeric type
RuntimeException	Occurs when an exception occurs at runtime
StringIndexOutOfBoundsException	Occurs when trying to access a character in a String that does not exist
TypeNotPresentException	Occurs when a type cannot be found
IllegalArgumentException	Occurs when when an illegal argument is passed to a method
IllegalStateException	Occurs when when a method is called at an illegal time

11.1 Try – Catch – Finally

Blok try digunakan untuk membungkus kode yang kemungkinan akan menimbulkan exception. Jika exception terjadi, blok catch akan menangani error tersebut secara spesifik. Blok finally akan selalu dijalankan, baik terjadi error atau tidak, dan biasanya digunakan untuk menutup koneksi atau membersihkan resource lain.

contoh:

```
try
{
   int[] data = {1, 2, 3};
   System.out.println(data[5]);  // akan menimbulkan ArrayIndexOutOfBoundsException
} catch (ArrayIndexOutOfBoundsException e)
{
   System.out.println("Index array di luar batas: " + e.getMessage());
}
finally
{
   System.out.println("Blok finally tetap dijalankan.");
}
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

11.2 Throw dan Throws

Keyword throw digunakan untuk secara eksplisit melempar sebuah exception, biasanya digunakan dalam logika validasi atau kondisi tertentu. Sedangkan throws digunakan dalam deklarasi method untuk menunjukkan bahwa method tersebut mungkin melempar exception ke pemanggilnya.

Contoh throw:

```
public class Validasi
{
   static void cekUmur(int umur)
 {
      if (umur < 18)
    {
        throw new IllegalArgumentException("Umur harus minimal 18 tahun.");
      }
  }
   public static void main(String[] args)
 {
      try
     {
        cekUmur(15);
      } catch (IllegalArgumentException e)
       {
        System.out.println("Validasi gagal: " + e.getMessage());
        }
    }
 }
```

Contoh throws:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class InputUser
{
   // Method ini bisa menimbulkan IOException, maka harus ditandai dengan 'throws'
   public static String bacaInput() throws IOException
{
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Masukkan nama Anda: ");
```

```
        return reader.readLine(); // readLine bisa melempar IOException
    }
    public static void main(String[] args)
    {
        try { String nama = bacaInput(); // panggil method yang bisa lempar exception
            System.out.println("Halo, " + nama);
        } catch (IOException e)
        {
            System.out.println("Terjadi kesalahan saat membaca input: " + e.getMessage());
        }
    }
}
```

Fungsi **readLine()** adalah metode yang bisa melempar IOException. Oleh karena itu, method bacaInput() harus mendeklarasikan bahwa dia bisa melempar exception tersebut menggunakan throws IOException. Di main(), menangkap error itu dengan try-catch yang merupakan penanganan exception yang sebenarnya.

Gunakan throws ketika tidak ingin langsung menangani exception di dalam method tersebut, tapi ingin meneruskannya ke pemanggil method. Cocok digunakan di level kode yang lebih dalam (seperti utility/helper) agar error ditangani di level aplikasi utama.

Berikut beberapa contoh lain penggunaan exception handling:

```
public class Main
{
    public static void main(String[] args)
    {
        try {  // Potensi kode yang dapat menimbulkan exception
            int hasil = bagi(10, 0);
            System.out.println("Hasil pembagian: " + hasil);
        }
        catch (ArithmeticException e)
        {
            // Tangani exception jika terjadi pembagian dengan nol
            System.out.println("Terjadi pembagian dengan nol: " + e.getMessage());
        }
        finally
        {
            // Blok finally akan dieksekusi selalu, baik terjadi exception atau tidak
            System.out.println("Program selesai.");
        }
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
    }
  // Method untuk melakukan pembagian
    static int bagi(int pembilang, int penyebut)
    {
      if (penyebut == 0)
       {
         // Buang exception jika terjadi pembagian dengan nol
         throw new ArithmeticException("Tidak bisa melakukan pembagian dengan nol");
      }    return pembilang / penyebut;
    }
}
```

Dalam blok try, ditempatkan kode yang memiliki potensi untuk menimbulkan exception. Jika exception terjadi, kontrol program akan beralih ke blok catch, yang menangkap dan menangani exception yang sesuai. Dalam contoh di atas, program menangkap exception ArithmeticException yang terjadi jika terjadi pembagian dengan nol. Blok finally merupakan bagian opsional yang akan dieksekusi setelah blok try atau catch, baik terjadi exception maupun tidak. Biasanya blok ini digunakan untuk melakukan pembersihan atau penutupan sumber daya. Program memanggil method bagi() yang memiliki potensi untuk menimbulkan exception ArithmeticException jika terjadi pembagian dengan nol.

Contoh kasus lain: Input Gaji Pegawai

```
import java.util.Scanner;
public class GajiPegawai
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);
      try
       {
         System.out.print("Masukkan gaji pokok: ");
         double gaji = Double.parseDouble(input.nextLine());
         if (gaji < 0)
          {
            throw new IllegalArgumentException("Gaji tidak boleh negatif.");
          }
         double total = gaji + (0.1 * gaji);
         System.out.println("Total gaji (dengan bonus 10%): " + total);
        }
         catch (NumberFormatException e)
```

```
{
        System.out.println("Input harus berupa angka!");
        }

catch (IllegalArgumentException e)
   {
      System.out.println("Kesalahan: " + e.getMessage());
    }
      finally
       {
        System.out.println("Program selesai dijalankan.");
       input.close();
    }
 }
}
```

Program di atas digunakan untuk menghitung total gaji seorang pegawai dengan menambahkan bonus sebesar 10% dari gaji pokok. Input gaji dibaca menggunakan Scanner dan dikonversi ke tipe double menggunakan Double.parseDouble(). Jika input bukan angka, program akan menangkap kesalahan melalui NumberFormatException. Selain itu, jika gaji yang dimasukkan bernilai negatif, program akan melempar dan menangani IllegalArgumentException dengan pesan khusus. Bagian finally memastikan bahwa objek Scanner ditutup dan pesan akhir ditampilkan, terlepas dari apakah terjadi kesalahan atau tidak. Program tersebut mencerminkan penerapan exception handling untuk menjaga agar input yang tidak valid tidak menyebabkan program berhenti secara tidak terduga.

Studi kasus lain yang menggambarkan penerapan teknik ini adalah validasi input pengguna dalam sistem tiket. Dalam sistem pemesanan tiket, pengguna seringkali memasukkan data yang tidak valid. Sistem harus dapat menanganinya tanpa crash, dan memberikan umpan balik yang jelas. Permasalahan terjadi ketika pengguna diminta memasukkan jumlah tiket yang ingin dibeli. Jika memasukkan teks atau angka negatif, program harus menangani kesalahan tersebut dengan baik. Berikut kode program untuk mengatasi masalah tersebut:

```
import  .util.Scanner;
public class TiketApp {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      try {
         System.out.print("Masukkan jumlah tiket: ");
         int jumlah = Integer.parseInt(scanner.nextLine());
         if (jumlah <= 0) {
            throw new IllegalArgumentException("Jumlah tiket harus lebih dari 0.");
         }
         System.out.println("Tiket berhasil dipesan sebanyak " + jumlah + " buah.");
      } catch (NumberFormatException e) {
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

```
            System.out.println("Input tidak valid. Harus berupa angka.");
        } catch (IllegalArgumentException e) {
            System.out.println("Kesalahan: " + e.getMessage());
        } finally {
            System.out.println("Terima kasih telah menggunakan layanan kami.");
            scanner.close();
        }
    }
```

Program TiketApp di atas merupakan contoh aplikasi Java sederhana yang menggunakan exception handling untuk menangani input pengguna secara aman. Program ini meminta pengguna memasukkan jumlah tiket yang ingin dipesan, lalu memvalidasi input tersebut sebelum memberikan respons. Input dibaca menggunakan objek Scanner dan dikonversi menjadi bilangan bulat melalui Integer.parseInt(), yang berpotensi menimbulkan kesalahan jika input tidak sesuai format angka.

Seluruh logika utama dibungkus dalam blok try untuk mencegah program berhenti mendadak saat terjadi kesalahan. Bila input tidak bisa dikonversi, maka NumberFormatException akan ditangkap dan ditangani oleh blok catch. Selain itu, validasi logika bisnis dilakukan dengan memeriksa apakah jumlah tiket lebih besar dari nol. Jika tidak, program melempar exception secara manual menggunakan throw new IllegalArgumentException, yang juga ditangani dalam blok catch terpisah dengan pesan kesalahan yang relevan.

Program ini juga menyertakan blok finally yang akan dijalankan dalam kondisi apa pun, baik terjadi exception maupun tidak. Blok ini digunakan untuk mencetak pesan penutup dan menutup objek Scanner, sehingga resource dapat dibebaskan dengan baik. Dengan struktur tersebut, program menunjukkan bagaimana exception handling dapat digunakan tidak hanya untuk kesalahan teknis, tetapi juga untuk mengelola alur logika secara efektif, menjadikan aplikasi lebih tangguh dan responsif terhadap input yang tidak valid.

Mini Project:

Rancang sistem parkir otomatis untuk mencatat data kendaraan (Motor, Mobil, Bus), menghitung biaya parkir sesuai tipe & durasi, mencetak struk parkir. Petugas dapat memasukkan beberapa kendaraan (menggunakan looping + array). Setiap kendaraan diproses berdasarkan tipe dan jam masuk/keluar. Input bisa dari konsol atau GUI (opsional).

Aturan Logika & Error Handling

- Validasi waktu: jika jamKeluar < jamMasuk → lempar IllegalArgumentException.
- Kapasitas array penuh (mis. 10 kendaraan): tampilkan pesan penolakan dengan try-catch.
- Blok finally: pastikan Scanner ditutup dengan benar, meskipun input tidak valid.

Struktur Paket & Kelas

```
parkir.model
 ├─ Kendaraan.java   (abstract: plat, jamMasuk, jamKeluar, abstract hitungBiaya())
 ├─ Motor.java       (override hitungBiaya)
 ├─ Mobil.java       (override hitungBiaya)
 └─ Bus.java         (override hitungBiaya)

parkir.app
 └─ ParkirSystem.java (main program: input, looping array, cetak struk)
```

Tugas Mahasiswa

- Buat struktur package & class sesuai diagram.
- Implementasikan method hitungBiaya() berbeda untuk Motor, Mobil, dan Bus.
- Simulasikan input ≥3 kendaraan, lalu cetak struk parkir masing-masing.
- Tunjukkan penggunaan enkapsulasi, inheritance, polymorphism, exception handling.

Rubrik Penilaian

- Struktur package & inheritance benar (30%)
- Polymorphism (hitungBiaya) berjalan (30%)
- Exception handling & finally diterapkan (20%)
- Kerapian kode & dokumentasi singkat (20%) └── ParkirSystem.java (main)

Refleksi:

Jelaskan bagaimana penerapan pewarisan, abstraksi, dan penanganan exception dalam latihan di atas membantu membangun sistem yang rapi, aman, dan mudah dikembangkan. Apakah ada bagian dari logika program yang paling menantang? Mengapa?

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 12 Penerapan Teknologi Java

Java bukan hanya digunakan untuk membangun program berbasis console seperti yang dipelajari dalam dasar PBO, tetapi juga mendukung pengembangan aplikasi Desktop, Web, dan Mobile. Di bab ini, dibahas mengenai beberapa teknologi Java yang umum digunakan, serta penerapannya pada proyek nyata. Pembahasan juga disertai contoh program dan refleksi terhadap pengembangannya. Berikut beberapa contoh teknologi Java berbasis desktop, web, dan mobile.

12.1 Java untuk Aplikasi Desktop

A. Java Swing

Java Swing merupakan toolkit GUI klasik di Java yang menyediakan komponen-komponen seperti JFrame, JPanel, JLabel, JTextField, dan JButton. Komponen ini digunakan untuk membangun antarmuka pengguna secara cepat berbasis event-driven.

Contoh: Aplikasi kalkulator sederhana

```
JButton tambah = new JButton("Tambah");
tambah.addActionListener(e ->
{
  int hasil = Integer.parseInt(tf1.getText()) + Integer.parseInt(tf2.getText());
  labelHasil.setText("Hasil: " + hasil);
};
```

Ketika tombol ditekan, program akan mengambil input dari dua text field, menjumlahkannya, dan menampilkan hasil pada label.

B. JavaFX

JavaFX adalah framework modern untuk membangun aplikasi desktop dengan antarmuka yang lebih kaya dan fleksibel. JavaFX menggunakan file FXML untuk pemisahan tampilan dan logika, serta mendukung penggunaan CSS dan animasi.

Contoh: Olah data mahasiswa (JavaFX)

```
@FXML private TextField tfNama;
@FXML private Label lblOutput;
@FXML
private void simpanData()
{
  lblOutput.setText("Halo, " + tfNama.getText());
}
```

Input dari pengguna diambil dari TextField dan ditampilkan melalui Label saat tombol diklik.

12.2 Java untuk Aplikasi Web

A. Spring Boot

Spring Boot merupakan framework backend berbasis Spring yang memudahkan pembuatan RESTful API. Banyak digunakan untuk membangun backend sistem informasi karena bersifat modular dan mudah dikonfigurasi.

Contoh: Endpoint Cek Mahasiswa

```
@RestController
@RequestMapping("/mahasiswa")
public class MahasiswaController
{
   @GetMapping("/{nim}")
   public Mahasiswa getByNim(@PathVariable String nim)
   {
      return service.findByNim(nim);
   }
}
```

Ketika URL dengan parameter NIM diakses, sistem akan mengembalikan objek Mahasiswa dari service layer.

B. Vaadin

Vaadin adalah framework Java full-stack untuk pengembangan antarmuka web tanpa menulis kode HTML, CSS, atau JavaScript secara langsung. Vaadin mengubah kode Java menjadi elemen UI berbasis web yang dinamis.

Contoh Form input mahasiswa:

```
TextField namaField = new TextField("Nama");
Button simpan = new Button("Simpan", e ->
{
   Notification.show("Nama: " + namaField.getValue());
});
add(namaField, simpan);
```

Form dibuat hanya dengan Java. Ketika tombol ditekan, akan muncul notifikasi dengan isi input pengguna.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

12.3 Java untuk Aplikasi Mobile

A. Android (Java/Kotlin)

Java digunakan dalam Android Studio untuk membuat aplikasi mobile native. Antarmuka pengguna ditentukan dalam file XML, sedangkan logika aplikasi menggunakan Java.

Contoh: Tombol Hitung Luas Persegi

```
Button hitungBtn = findViewById(R.id.btnHitung);
hitungBtn.setOnClickListener(v -> {
   int sisi = Integer.parseInt(input.getText().toString());
   hasil.setText("Luas: " + (sisi * sisi));
});
```

Input diambil dari user, dihitung luasnya, lalu hasilnya ditampilkan.

B. J2ME (Java 2 Micro Edition)

J2ME menjadi bagian dari sejarah perkembangan platform Java, terutama di konteks Java Mobile, walaupun saat ini sudah sangat jarang digunakan dalam praktik modern. J2ME merupakan platform Java versi ringan yang dirancang untuk perangkat dengan kemampuan terbatas seperti: Ponsel fitur (feature phones), PDA (Personal Digital Assistant), Smartcard, dan perangkat embedded. Tujuannya untuk memberikan kemampuan menjalankan aplikasi Java di perangkat mobile yang tidak mendukung Java SE atau Java EE. J2ME dirancang agar hemat memori dan prosesor.

J2ME saat ini jarang digunakan atau kurang populer karena terbatas dalam fitur dan tampilan UI serta tidak mendukung smartphone modern. Saat ini digantikan oleh Android, yang juga bisa memakai Java tapi berbasis Android SDK.

Contoh Aplikasi J2ME (MIDlet):

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HelloMidlet extends MIDlet
{
   protected void startApp()
 {
      System.out.println("Hello from J2ME!");
   }
   protected void pauseApp() {}
   protected void destroyApp(boolean unconditional) {}
}
```

Struktur aplikasi J2ME menggunakan lifecycle method seperti startApp(), dan tidak memiliki GUI modern seperti Android.

Untuk memperluas pemahaman mengenai penerapan prinsip-prinsip Pemrograman Berorientasi Objek (OOP) dalam pengembangan perangkat lunak, mahasiswa diarahkan untuk menelaah beberapa artikel ilmiah yang relevan.

Artikel pertama ditulis oleh (Indahyanti, 2023) yang mengkaji implementasi OOP pada sistem pembayaran digital. Fokus pembahasan terdapat pada integrasi modul payment gateway ke dalam sistem penerimaan siswa baru. Studi ini menunjukkan bahwa konsep OOP, seperti enkapsulasi, inheritance, dan polymorphism, dapat digunakan untuk menyusun modul perangkat lunak yang saling terhubung namun tetap terkelola secara mandiri. Penerapan konsep modular tersebut menjadikan sistem lebih mudah diintegrasikan, diperluas, dan dipelihara. Kasus ini dapat dipandang sebagai contoh nyata bagaimana OOP mendukung pembangunan aplikasi yang berskala institusional dengan kebutuhan layanan yang beragam.

Artikel kedua ditulis oleh (Indra Astutik and Wignya Radhitya, 2024).yang membahas pengembangan sistem informasi pemesanan jasa fotografi berbasis web dengan metode Waterfall. Tahapan yang dijelaskan meliputi analisis kebutuhan, perancangan sistem, implementasi, pengujian, hingga pemeliharaan. Walaupun aplikasi yang dibangun menggunakan bahasa PHP, konsep OOP yang digunakan dalam perancangannya memiliki relevansi yang sama dengan praktik menggunakan bahasa Java. Artikel ini memperlihatkan bagaimana proses bisnis dianalisis, bagaimana model data disusun dengan pendekatan objek, serta bagaimana dokumentasi pengujian sistem dilakukan untuk menjamin kualitas perangkat lunak.

Kedua artikel tersebut memberikan gambaran yang saling melengkapi. Studi oleh Indahyanti menekankan pada penerapan OOP untuk mendukung modularitas dan integrasi layanan, sedangkan studi oleh Astutik dan Radhitya menekankan pada pentingnya pendekatan sistematis dalam siklus pengembangan perangkat lunak. Dengan mempelajari kedua referensi ini, mahasiswa diharapkan mampu melihat keterkaitan antara konsep OOP yang dipelajari di kelas dengan penerapannya pada proyek perangkat lunak yang sesungguhnya.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Refleksi:

Berdasarkan pemahaman awal tentang berbagai teknologi Java (seperti JavaFX, Vaadin, dan Spring Boot), bagaimana perkiraan tantangan dan peluang yang mungkin muncul saat mengembangkan aplikasi menggunakan salah satu teknologi tersebut dalam proyek akhir nanti?

Proyek Kelompok:

Buat aplikasi Sistem Informasi Akademik sederhana dengan struktur class minimal:

- Person (superclass)
- Mahasiswa, Dosen (subclass dari Person)
- MataKuliah (class terpisah)
- Registrasi (interface)
- MainAkademik (class utama)

Mahasiswa memiliki daftar mata kuliah, Dosen membimbing mahasiswa, dan Mahasiswa mengimplementasikan interface Registrasi, dengan fitur utama:

- Mahasiswa ambil mata kuliah & hitung total SKS.
- Validasi input (misalnya batas SKS, exception handling).
- Relasi antar class sesuai OOP (inheritance, interface).

Teknologi Java yang digunakan masing-masing kelompok, antara lain (diundi di kelas):

- JavaFX (Desktop)
- Spring Boot (Web)
- Android (Mobile, Java/Kotlin)

Output yang dikumpulkan:

- Source code Java (.zip atau repo GitHub).
- Laporan ringkas (identitas kelompok, panduan run, evaluasi proyek).
- Mockup GUI (gambaran tampilan aplikasi).
- File .jar untuk menjalankan aplikasi.
- Video demo (opsional).

Rubrik penilaian:

- Fungsionalitas Program → 40%
- Laporan → 20%
- Demo/Presentasi → 20%
- Kontribusi Individu → 20%

DAFTAR ISTILAH

Istilah	Definisi
Class	Cetakan atau template untuk membuat objek.
Object	Instansi dari class yang memiliki atribut dan method.
Enkapsulasi	Penyembunyian data dengan mengatur hak akses.
Polymorphism	Kemampuan satu method memiliki banyak bentuk sesuai konteks objek.
Abstraksi	Menyembunyikan detail teknis dan hanya menampilkan fitur penting.
Inheritance	Pewarisan atribut dan method dari class induk ke class anak.
Interface	Kontrak perilaku yang harus diimplementasikan oleh class.
Constructor	Method khusus yang dijalankan saat objek dibuat.
Exception	Objek yang mewakili kesalahan saat runtime.
Method	Fungsi atau prosedur dalam class yang mendefinisikan perilaku objek.
This	Keyword yang merujuk pada objek itu sendiri (objek saat ini).
Super	Keyword untuk mengakses anggota class induk dari class anak.
Overriding	Proses mendefinisikan ulang method class induk di dalam class anak.
Overloading	Dua atau lebih method dengan nama sama tapi parameter berbeda.
Array	Struktur data untuk menyimpan kumpulan elemen dengan tipe yang sama.
ArrayList	Struktur data dinamis berbasis array dari library Java.
Thread	Unit eksekusi terpisah yang dapat berjalan bersamaan dalam program.
Synchronized	Keyword untuk mengamankan akses bersama ke data saat multithreading.
Main Method	Titik awal eksekusi program Java: public static void main(String[] args)
Package	Kumpulan class dalam namespace tertentu untuk pengorganisasian kode.
Jar File	Java Archive – file arsip yang memuat class dan resource program.
Class Diagram	Diagram UML yang menunjukkan struktur class dan relasi antar class.
Constructor Overloading	Lebih dari satu constructor dengan parameter berbeda dalam satu class.
UML (Unified Modeling Language)	Bahasa visual standar untuk memodelkan struktur dan perilaku sistem perangkat lunak.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

DAFTAR PUSTAKA

Adiputra, F. (2022) *Pemrograman Berorientasi Objek dengan Java*. Malang: Media Nusa Kreatif.

Hadiprakoso, R.B. (2021) *Pemrograman Berorientasi Objek Teori dan Implementasi dengan Java*. Jakarta: RBH.

Indahyanti, U. (2020) *Buku Ajar Algoritma Pemrograman*. Sidoarjo: UMSIDA Press.

Indahyanti, U. (2023) 'New Student Admission Information System Design With Payment Gateway', *Jurnal Mantik*, 7(1).

Indra Astutik, I.R. and Wignya Radhitya, Y. (2024) 'Sistem Informasi Pemesanan Jasa Fotografer Berbasis Web Pada Studio Fotograferku', *Journal of Technology and System Information*, 1(1), pp. 1–15. doi:10.47134/jtsi.v1i1.2142.

Jusuf, H. (2017) *Modul Konsep PBO*.

Kreher, D.L. and Stinson, D.R. (2020) *Structures and Algorithms*, *Combinatorial Algorithms*. doi:10.1201/9781003068006-1.

Ross, S.D. (2014) *Engineering Design: A Project-Based Introduction*, *Women's Human Rights*. doi:10.9783/9780812200027.toc.

Sianipar, R.H. (2018) *Dasar Analisis & Perancangan Berorientasi Objek Menggunakan Java*. Yogyakarta: ANDI.

Somashekara, M.T., Guru, D.S. and Manjunatha, K.S. (2017) *Object Oriented Programming with Java*. New Delhi: Prentice Hall India Pvt.

UMSIDA, I. (2017) *Modul Praktikum PBO*. Sidoarjo: Informatika UMSIDA.

https://www.w3schools.com/java

https://www.geeksforgeeks.org/java/

https://docs.oracle.com/javase/tutorial/essential/

BIODATA PENULIS

Uce Indahyanti, S.Kom, M.Kom lahir di Situbondo. Penulis menyelesaikan pendidikan S-1 di STIKOM Surabaya (sekarang UNDIKA), dan S-2 di Prodi Sistem Informasi ITS Surabaya dengan program beasiswa dari DIKTI. Saat ini sedang menempuh S-3 di Prodi Ilmu Komputer ITS Surabaya dengan program beasiswa internal UMSIDA. Penulis merupakan dosen tetap di Program Studi Informatika, Universitas Muhammadiyah Sidoarjo. Mata kuliah yang diampu antara lain Algoritma dan Pemrograman, Algoritma Struktur Data, Pemrograman Berorientasi Objek, dan Audit Sistem Informasi. Penulis juga aktif terlibat dalam kegiatan penelitian dan pengabdian kepada masyarakat (abdimas). Daftar publikasi hasil penelitian dan abdimas penulis bersama rekan seprofesi dan mahasiswa tersedia di https://tinyurl.com/publikasipenulis.

Ika Ratna Indra Astutik, S.Kom., M.T. lahir di Sidoarjo. Penulis menerima gelar Sarjana (S-1) Teknik Informatika dari Universitas Muhammadiyah Sidoarjo. Penulis melanjutkan magister (S-2) Jaringan Cerdas Multimedia di ITS dengan program beasiswa dari DIKTI. Penulis mengawali karirnya sebagai Dosen di prodi Informatika Fakultas Sains dan Teknologi Universitas Muhammadiyah Sidoarjo. Penulis juga aktif terlibat dalam kegiatan penelitian dan pengabdian kepada masyarakat. Penelitian yang pernah dilakukan oleh penulis adalah tentang Rekayasa Perangkat Lunak, Basis Data, Sistem Pakar dan Game. Daftar publikasi tersedia di https://tinyurl.com/publikasipenulis2.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

UMSIDA PRESS
Universitas Muhammadiyah Sidoarjo
Jl. Mojopahit No. 666B
Sidoarjo, Jawa Timur

ISBN 978-623-464-130-1 (PDF)