



PRODI S1 TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KH BAHAUDIN MUDHARY MADURA

FINAL REPORT

Object Oriented Programming



Java™

SOVI ALFI NAFILAH

2302310021

LEMBAR PENGESAHAN
PRAKTIKUM *OBJECT-ORIENTED PROGRAMING*

PROGRAM STUDI INFORMATIKA
UNIVERSITAS BAHAUDIN MUDHARY MADURA

Ditujukan untuk memenuhi persyaratan **LULUS** praktikum *Object-Oriented
Programing*



SOVI ALFI NAFILAH

NIM .2302310021

Laporan ini telah direvisi dan disetujui oleh Asisten Praktikum dan Dosen
Pengampu pada tanggal 24 Desember 2023

Mengetahui,
Assiten Pengampu Praktikum

Bagus Danu Raharjo

NIM. 2202310094

Menyetujui,
Dosen Pengampu Mata Kuliah

Akhmad Tajuddin Tholaby MS, S.Kom., M.Kom

NIP. 19890330.202203.1.079

LEMBAR ASISTENSI

KATA PENGANTAR

Puji syukur kami panjatkan ke hadirat Allah Swt., yang dengan rahmat dan petunjuk-Nya, laporan praktikum ini pada mata kuliah *Object Oriented Programing* dapat di selesaikan dengan baik. Laporan ini ditulis sebagai bentuk dokumentasi dari kegiatan praktikum yang telah dilaksanakan.

Penulis mengucapkan terimakasih yang sebesar-besarnya kepada dosen pengampu, Bapak Akhmad Tajuddin Tholaby MS, S.Kom., M.Kom dan Asiten Pengampu Praktikum yang telah memberikan bimbingan, ilmu, serta arahan yang sangat berharga serta memberikan dukungannpenuh kepada seluruh peserta praktikum.

Semoga laporan ini dapat memberikan manfaat dan pemahaman yang lebih mendalam. Kami menyadari bahwa masih banyak kekurangan dalam penyusunan laporan ini, oleh karena itu, kritik dan saran yang membangun sangat kami harapkan guna perbaikan di masa mendatang. Penulis berharap semoga laporan ini memberikan banyak manfaat dalam pengembangan ilmu pengetahuan dan teknologi.

Sumenep, 22 Desember 2024

SOVI ALFI NAFILAH

DAFTAR ISI

| | |
|--|------------|
| COVER | i |
| LEMBAR PENGESAHAN | ii |
| LEMBAR ASISTENSI | iii |
| KATA PENGANTAR..... | iv |
| DAFTAR ISI | v |
| DAFTAR GAMBAR | vii |
| | |
| MODUL I PEMROGRAMAN JAVA..... | 1 |
| BAB I PENDAHULUAN | 2 |
| BAB II DASAR TEORI..... | 3 |
| BAB III PERMASALAHAN..... | 8 |
| BAB IV IMPLEMENTASI..... | 9 |
| BAB V PENUTUP..... | 10 |
| | |
| MODUL II USER <i>INPUT</i>..... | 11 |
| BAB I PENDAHULUAN | 12 |
| BAB II DASAR TEORI..... | 13 |
| BAB III PERMASALAHAN..... | 17 |
| BAB IV IMPLEMENTASI..... | 19 |
| BAB V PENUTUP | 19 |
| | |
| MODUL III KONDISIONAL PERULANGAN DAN PERCABANGAN | 21 |
| BAB I PENDAHULUAN | 22 |
| BAB II DASAR TEORI..... | 23 |
| BAB III PERMASALAHAN..... | 28 |
| BAB IV IMPLEMENTASI..... | 29 |
| BAB V PENUTUP..... | 31 |
| | |
| MODUL IV JAVA <i>ARRAY</i> | 32 |

| | |
|--|-----------|
| BAB I PENDAHULUAN | 33 |
| BAB II DASAR TEORI..... | 34 |
| BAB III PERMASALAHAN..... | 39 |
| BAB IV IMPLEMENTASI..... | 40 |
| BAB V PENUTUP..... | 41 |
| MODUL V PEMROGRAMAN BERORIENTASI OBJEK | 42 |
| BAB I PENDAHULUAN | 43 |
| BAB II DASAR TEORI..... | 44 |
| BAB III PERMASALAHAN..... | 49 |
| BAB VI IMPLEMENTASI..... | 50 |
| BAB V PENUTUP..... | 52 |
| MODUL VI <i>CONSTRUCTOR</i> DAN <i>INHERITANCE</i>..... | 53 |
| BAB II DASAR TEORI..... | 55 |
| BAB III PERMASALAHAN..... | 60 |
| BAB IV IMPLEMENTASI..... | 61 |
| BAB V PENUTUP..... | 65 |
| A. KESIMPULAN | 66 |
| B. SARAN | 66 |
| DAFTAR PUSTAKA..... | 67 |

DAFTAR GAMBAR

| | | |
|-------------------|--|----|
| Gambar 1.1 | <i>Program Java Yang Menghitung Volume Bangun Ruang.....</i> | 9 |
| Gambar 1.2 | <i>Output Program Java Yang Menghitung Volume Bangun Ruang</i> | 9 |
| Gambar 2.1 | <i>Program Inputan Dengan Class Scanner</i> | 19 |
| Gambar 2.2 | <i>Output Program Inputan Dengan Class Scanner</i> | 19 |
| Gambar 3.1 | <i>Program Menentukan Bilang Ganjil dan Genap.....</i> | 29 |
| Gambar 3.2 | <i>Output Program Menentukan Ganjil dan Genap</i> | 29 |
| Gambar 3.3 | <i>Program Menentukan Nilai Beserta Outputnya</i> | 30 |
| Gambar 4.1 | <i>Soal Permasalahan</i> | 39 |
| Gambar 4.2 | <i>Program Data Karyawan</i> | 40 |
| Gambar 4.3 | <i>Output Program Data Karyawan</i> | 40 |
| Gambar 5.1 | <i>Class Fotocopy dan Class Perhitungan.....</i> | 50 |
| Gambar 5.2 | <i>Method Untuk Mengakses Class.....</i> | 50 |
| Gambar 5.3 | <i>Source Code Lanjutan</i> | 51 |
| Gambar 5.4 | <i>Output Program Fotokopi.....</i> | 51 |
| Gambar 6.1 | <i>Class dan Inheritance</i> | 61 |
| Gambar 6.2 | <i>Main Program Fotokopi</i> | 61 |
| Gambar 6.3 | <i>Kalkulasi Program.....</i> | 62 |
| Gambar 6.4 | <i>Output Program FotoKopi.....</i> | 62 |
| Gambar 6.5 | <i>Parent Class dan Child Class</i> | 63 |
| Gambar 6.6 | <i>Class Main Program Penjualan Kertas.....</i> | 63 |
| Gambar 6.7 | <i>Output Program Penjualan Kertas</i> | 64 |



PRODI S1 TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KH BAHAUDIN MUDHARY MADURA

MODUL 1

PEMROGRAMAN JAVA



JavaTM

SOVI ALFI NAFILAH

2302310021

BAB I

PENDAHULUAN

1.1 Latar Belakang

Salah satu bahasa pemrograman yang populer digunakan untuk mengembangkan aplikasi adalah bahasa pemrograman *Java*. Baik untuk membangun aplikasi *desktop*, *mobile* hingga *website*. Bahkan beberapa *website* besar di dunia seperti Spotify, LinkedIn dan Yahoo! Juga menggunakan *Java* untuk mengembangkan *website* nya.

Bahasa pemrograman *Java* ini pertama kali dikembangkan melalui sebuah proyek bernama “*The Green Project*” pada tahun 1991. Proyek ini dibentuk oleh Sun Microsystems. Proyek ini beranggotakan James Gosling, Mike Sheridan dan Patrick Naughton.

Popularitas *Java* didukung oleh fleksibilitasnya yang bisa digunakan di berbagai *platform*. Tidak heran jika *Java* sudah digunakan oleh sekitar 12 juta *developer* untuk mengembangkan aplikasi. Karena *Java* merupakan Bahasa pemrograman terkenal dengan slogannya, “*Write Once, Run Anywhere*” (WORA). Artinya, aplikasi yang ditulis di *Java* bisa berjalan di berbagai platform tanpa harus mengubah kode, berkat *Java Virtual Machine* (JVM). Ini memudahkan pengembang untuk membangun aplikasi lintas platform.

Java sudah ada selama lebih dari dua dekade, sehingga stabilitas dan performanya sangat teruji. Komunitas pengembang yang besar juga telah membantu menemukan dan memperbaiki *bug*, sehingga *Java* sangat handal untuk aplikasi skala besar. Karena fleksibilitasnya, *Java* sudah dijalankan di 13 miliar perangkat. Masih banyak kelebihan serta kekurangan Bahasa pemrograman *Java* yang penting untuk kita ketahui. (Larasati 2022)

1.2 Tujuan praktikum

- A. Mahasiswa mampu membuat program sederhana menggunakan *Java*.
- B. Mahasiswa mampu mengkompilasi dan menjalankan program *Java*.

BAB II DASAR TEORI

2.1 Sejarah dan Pengertian *Java*

Java adalah sebuah bahasa pemrograman yang dapat dijalankan di berbagai komputer, termasuk perangkat telepon genggam. Bahasa ini awalnya dikembangkan oleh James Gosling saat masih bekerja di Sun Microsystems, yang saat ini merupakan bagian dari Oracle, dan pertama kali dirilis pada tahun 1995. *Java* mengadopsi banyak *sintaksis* yang berasal dari bahasa *C* dan *C++*, tetapi dengan model *sintaksis* objek yang lebih sederhana serta dukungan yang lebih minim untuk rutin-rutin aras bawah.

Bahasa pemrograman *Java* lahir dari *The Green Project*, yang berjalan selama 18 bulan, dimulai pada awal tahun 1991 hingga musim panas 1992. Proyek ini awalnya belum menggunakan nama Oak. Inisiatif ini didorong oleh sejumlah individu utama, termasuk Patrick Naughton, Mike Sheridan, dan James Gosling, bersama dengan sembilan programmer lainnya dari Sun Microsystems. Sebagai hasil dari proyek ini, lahirlah maskot Duke yang diciptakan oleh Joe Palrang.

Java adalah bahasa pemrograman yang telah merajai pengodean aplikasi web selama lebih dari dua dekade. Dalam kurun waktu tersebut, *Java* telah menjadi pilihan populer di kalangan para pengembang, dengan jutaan aplikasi yang menggunakan bahasa ini saat ini. Keunggulan utama *Java* adalah sifatnya yang multiplatform, berorientasi pada objek, dan berfokus pada pengembangan aplikasi jaringan. Bahasa ini mampu berfungsi sebagai *platform* dalam berbagai konteks, dari pengembangan aplikasi seluler hingga perangkat lunak korporasi, serta aplikasi big data dan teknologi *server*.

Selain itu, *Java* juga dikenal karena kecepatan, keamanan, dan keandalannya dalam pengodean berbagai jenis perangkat lunak. Kemampuan bahasa ini untuk menjalankan aplikasi dengan performa tinggi, menjaga keamanan sistem, dan memberikan keandalan yang konsisten menjadikannya pilihan yang solid untuk berbagai kebutuhan pengembangan perangkat lunak di era digital ini. (Az-zahir 2023)

2.2 Komponen Bahasa Pemrograman *Java*

Java memiliki tiga komponen utama yang meliputi sebagai berikut :

A. *Java Development Kit (JDK)*

Java Development Kit (JDK) ang merupakan komponen inti dari *Java Environment* dan menyediakan semua alat, binari, dan *executable* untuk melakukan kompilasi, *debugging*, dan menjalankan program *Java*

B. *Java Virtual Machine (JVM)*

Java Virtual Machine (JVM) yang merupakan jantung dari bahasa pemrograman *Java*. JVM menjadi tumpuan dalam mengubah kode bit ke kode khusus. JVM sendiri bergantung pada *platform* dan menyediakan fungsi inti *Java* seperti pengelolaan memori, pengumpulan sampah, keamanan, dan sebagainya

C. *Java Runtime Environment (JRE)*

Java Runtime Environment (JRE) merupakan implementasi dari JVM yang menyediakan *platform* untuk menjalankan program *Java*. JRE terdiri dari JVM, binari *Java*, dan beberapa kelas untuk menjalankan program *Java* dengan baik. (Raharja 2022)

2.3 Tipe Data dan *Variable*

Secara garis besar, tipe data dalam bahasa *Java* terbagi menjadi dua kelompok, yaitu tipe data sederhana (*primitive data types*) dan tipe data kompleks atau objek (*non-primitive data types*). Tipe data sederhana meliputi tipe data *integer* untuk angka bulat seperti 5, 7, atau 48; tipe data *float/double* untuk angka pecahan seperti 3.14, 5.55, atau 0.00024; tipe data *boolean* yang berisi nilai *true* atau *false*; serta tipe data *char* untuk satu karakter, seperti 'a', 'Z', atau '%'.

Sementara itu, tipe data kompleks mencakup tipe data *string* untuk kumpulan karakter seperti "Andi", "Duniaikom", atau "Belajar *Java*"; tipe data *array* untuk kumpulan tipe data sejenis; dan tipe data objek yang dapat menampung beragam data serta memiliki fungsi atau metode sendiri. Bahasa pemrograman *Java* termasuk kelompok *strongly typed programming language*, yang artinya untuk setiap variabel harus ditulis akan berisi tipe data apa. Apakah itu angka bulat (*integer*), angka pecahan (*float*), huruf (*char*), atau yang lain. (Duniaikom 2019)

Dalam *Java*, pembuatan variabel harus mengikuti aturan tertentu. Nama variabel harus dimulai dengan huruf (a-z atau A-Z), simbol *underscore* (`_`) atau dollar sign (`$`), dan karakter berikutnya dapat berupa huruf, angka (0-9), *underscore*, atau dollar sign. Variabel tidak boleh menggunakan spasi, simbol lain seperti `@`, `#`, `%`, atau menggunakan kata kunci bawaan *Java* seperti `int`, `class`, atau `static`. *Java* bersifat *case-sensitive*, sehingga penulisan huruf besar dan kecil membedakan variabel, misalnya `age` berbeda dengan `Age`. Variabel harus dideklarasikan dengan tipe data, misalnya `int number;`, dan dapat diinisialisasi bersamaan atau di waktu berbeda. Variabel yang dideklarasikan dalam suatu blok hanya dapat diakses di dalam blok tersebut, sementara variabel global dideklarasikan di dalam kelas namun di luar metode. Untuk variabel konstan, gunakan kata kunci `final`, dan nama variabel ditulis dalam huruf kapital dengan pemisah *underscore*, seperti `final double PI = 3.14;`. Mematuhi aturan ini membuat kode *Java* lebih terstruktur dan mudah dipahami.

2.4 Operator Aritmatika

Java menyediakan lima operator aritmatika dasar yang ditampilkan dalam tabel berikut:

- A. Operator penjumlahan digunakan untuk menjumlahkan dua nilai. Dalam *Java*, operator ini dapat digunakan untuk bilangan bulat, bilangan desimal, dan bahkan untuk penggabungan *string*.
- B. Operator pengurangan digunakan untuk mengurangi satu nilai dari nilai lainnya. Sama seperti operator penjumlahan, pengurangan juga dapat digunakan pada berbagai tipe data numerik.
- C. Operator perkalian digunakan untuk mengalikan dua nilai. Ini sangat berguna dalam berbagai perhitungan matematika.
- D. Operator pembagian digunakan untuk membagi satu nilai dengan nilai lainnya. Hasil dari operasi pembagian ini dapat berupa bilangan bulat atau desimal tergantung pada tipe data yang digunakan.
- E. Operator modulus memberikan sisa dari pembagian dua bilangan. Ini sering digunakan dalam berbagai aplikasi, seperti menentukan apakah suatu bilangan genap atau ganjil. (Fahrizal 2024)

2.5 Kelebihan Java

Bahasa *Java* memiliki banyak kelebihan yang membuatnya populer di kalangan pengembang perangkat lunak. Beberapa kelebihannya antara lain:

A. Portabilitas (*Write Once, Run Anywhere*)

Java dirancang untuk menjadi *platform-independent*. Program yang ditulis dalam *Java* dapat dijalankan di berbagai *platform* tanpa perlu melakukan perubahan kode, selama *platform* tersebut mendukung *Java Runtime Environment* (JRE). Hal ini karena *Java* dikompilasi menjadi *bytecode*, yang dijalankan oleh *Java Virtual Machine* (JVM) pada berbagai sistem operasi.

B. Kemudahan Penggunaan

Java memiliki sintaks yang jelas dan mudah dipahami, yang menjadikannya pilihan yang baik untuk pemrogram pemula. Selain itu, *Java* memiliki banyak dokumentasi dan sumber daya pembelajaran yang tersedia.

C. Objek-Oriented Programming (OOP)

Java sepenuhnya mendukung paradigma pemrograman berorientasi objek, yang memungkinkan pengembang untuk membangun aplikasi modular dan mudah dipelihara. Konsep OOP seperti *Inheritance*, *polymorphism*, dan *encapsulation* sangat mendukung pengembangan perangkat lunak yang lebih fleksibel dan terstruktur.

D. Keamanan

Java menawarkan beberapa fitur keamanan, termasuk manajemen memori otomatis (*garbage collection*) dan pengelolaan akses melalui *sandboxing* yang membatasi akses program terhadap sistem. *Java* juga memiliki API keamanan yang kuat, mendukung enkripsi, autentikasi, dan kontrol akses.

E. Multithreading

Java memiliki dukungan bawaan untuk multithreading, yang memungkinkan pengembang untuk membuat aplikasi yang dapat menjalankan beberapa proses secara bersamaan (*concurrent processing*), meningkatkan performa aplikasi terutama dalam aplikasi yang membutuhkan pemrosesan *parallel*.

F. Kinerja yang Baik

Meskipun *Java* tidak secepat bahasa pemrograman seperti *C* atau *C++*, JVM terus ditingkatkan untuk meningkatkan kinerja. *Java* juga mendukung *Just-In-Time (JIT) compiler* yang membantu meningkatkan kecepatan eksekusi program.

G. Dukungan Komunitas yang Kuat

Java memiliki komunitas pengembang yang besar dan aktif. Ini berarti banyaknya pustaka (*library*) *open-source* dan *framework* yang bisa digunakan untuk mempercepat pengembangan aplikasi, seperti *Spring*, *Hibernate*, dan banyak lagi.

H. Ketersediaan Tools Pengembangan

Java dilengkapi dengan berbagai *Integrated Development Environment (IDE)* yang kuat seperti *Eclipse*, *IntelliJ IDEA*, dan *NetBeans* yang menawarkan *fitur debugging*, *autocompletion*, dan *refactoring* yang membantu pengembang menulis kode lebih efisien.

I. Ketersediaan di Berbagai Bidang

Java banyak digunakan dalam berbagai jenis aplikasi, mulai dari aplikasi web, aplikasi *mobile* (Android), aplikasi *desktop*, hingga aplikasi *enterprise* berskala besar. Ini menjadikan *Java* sangat fleksibel untuk berbagai jenis pengembangan perangkat lunak.

BAB III

PERMASALAHAN

3.1 Tugas Praktikum

A. Buatlah sebuah program *Java* yang menghitung *volume* dari beberapa bangun ruang secara langsung (tanpa *input* dari pengguna). Bangun ruang yang dihitung adalah:

- 1) Kubus
- 2) Balok
- 3) Tabung

Program harus menghitung dan menampilkan hasil *volume* dari masing-masing bangun ruang.

BAB IV IMPLEMENTASI

4.1 Pembahasan

A. Program *Java* yang menghitung *volume* dari beberapa bangun ruang:

```

1 public class modul_satu {
2     public static void main(String[] args) {
3
4         int sisi = 4;
5         int volume_kubus = sisi * sisi * sisi;
6         int r = 2;
7         int t = 6;
8         Double volume_tabung = Math.PI * r * r * t;
9         int panjang = 3;
10        int lebar = 6;
11        int tinggi = 7;
12        int volume_balok = panjang * lebar * tinggi;
13
14        System.out.println("volume kubus");
15        System.out.println("sisi : " + sisi);
16        System.out.println(sisi + "*" + sisi + "*" + sisi + " = " + volume_kubus + "\n");
17
18        System.out.println("volume balok");
19        System.out.println("panjang = " + panjang);
20        System.out.println("lebar = " + lebar);
21        System.out.println("tinggi = " + tinggi);
22        System.out.println(panjang + "*" + lebar + "*" + tinggi + " = " + volume_balok + "\n");
23
24        System.out.println("volume tabung");
25        System.out.println("phi = " + Math.PI);
26        System.out.println("r = " + r);
27        System.out.println("t = " + t);
28        System.out.println(Math.PI * r * r * t + " = " + volume_tabung + "\n");
29    }
30 }

```

Gambar 1.1 Program *Java* Yang Menghitung *Volume* Bangun Ruang

Gambar 1.1 diatas merupakan program *Java* yang menghitung *volume* bangun ruang yaitu *volume* kubus, *volume* balok dan *volume* tabung. Masing-masing bangun ruang memiliki inisialisai *variable* yang berbeda-beda serta rumus yang berbeda pula.

```

OUTPUT  TERMINAL  PORTS
▼ TERMINAL
PS C:\Users\pc\OneDrive\Dokumen\java\modul1_2.java> & 'C:\Program Files\Java\jdk-21\bin\jav
tionMessages' '-cp' 'C:\Users\pc\OneDrive\Dokumen\java\modul1_2.java\bin' 'modul_satu'
volume kubus
sisi : 4
4*4*4= 64

volume balok
panjang = 3
lebar = 6
tinggi = 7
3 * 6 * 7 = 126

volume tabung
phi = 3.141592653589793
r = 2
t = 6
3.141592653589793 * 2 * 6 = 75.39822368615503

PS C:\Users\pc\OneDrive\Dokumen\java\modul1_2.java>

```

Gambar 1.2 Output Program *Java* Yang Menghitung *Volume* Bangun Ruang

Gambar 1.2 diatas merupakan *output* dari Program *Java* yang menghitung *volume* bangun ruang . karena program tersebut merupakan program yang statis maka tidak perlu memasukkan data lagi.

BAB V

PENUTUP

5.1 Kesimpulan

Java adalah bahasa pemrograman yang telah merajai pengodean aplikasi web selama lebih dari dua dekade. Dalam kurun waktu tersebut, *Java* telah menjadi pilihan populer di kalangan para pengembang, dengan jutaan aplikasi yang menggunakan bahasa ini saat ini. Keunggulan utama *Java* adalah sifatnya yang *multiplatform*, berorientasi pada objek, dan berfokus pada pengembangan aplikasi jaringan.

Java memiliki tiga komponen utama yang meliputi *Java Development Kit* (JDK), *Java Virtual Machine* (JVM) dan *Java Runtime Environment* (JRE). Secara garis besar, tipe data dalam bahasa *Java* terbagi menjadi dua kelompok, yaitu tipe data sederhana (*primitive data types*) seperti *int*, *float* atau *double* dan *boolean*. Tipe data kompleks atau objek (*non-primitive data types*) seperti *string*, *Char*, *object* dan lainnya. Pengisialisasian *variable* pada *Java* memiliki aturan-aturan yang sebenarnya hamper sama dengan *variable* apa umunya seperti Bahasa pemrograman *C++*.

Java juga mendukung operator aritmatika seperti penjumlahan, pengurangan, pembagian perkalian dan modulus. Operator ini untuk memproses suatu data numerik untuk menghasilkan *output* yang diinginkan. Penggunaan operator aritmatika pada *Java* tidak memiliki aturan khusus atau kata lain sama saja dengan Bahasa pemrograman pada umumnya.

5.2 Saran

Saran yang dapat saya berikan dalam menggunakan tipe data, variabel, dan operator aritmatika pada *Java* ialah pilih tipe data yang sesuai dengan kebutuhan. Gunakan *int* untuk angka bulat, *double* untuk angka pecahan karena lebih presisi dibandingkan *float*, dan *boolean* untuk nilai benar atau salah. Jika hanya memerlukan satu karakter, gunakan *char*



PRODI S1 TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KH BAHAUDIN MUDHARY MADURA

MODUL 2

USER INPUT



Java™

SOVI ALFI NAFILAH

2302310021

BAB I

PENDAHULUAN

1.1 Latar Belakang

Seperti yang kita ketahui, program komputer terdiri dari tiga komponen utama, yaitu: *input*, proses, dan *output*. Semua bahasa pemrograman telah menyediakan fungsi-fungsi untuk melakukan *input* dan *output*. Java sendiri sudah menyediakan tiga *class* untuk mengambil *input* yaitu *class Scanner*, *class BufferedReader* dan *class Console*. Tiga *class* tersebut untuk mengambil *input* pada program berbasis teks (*console*). Sedangkan untuk GUI menggunakan *class* yang lain seperti *JOptionPane* dan *inputbox* pada *form*. *Input* dari pengguna (*user input*) adalah salah satu elemen kunci dalam pemrograman karena memungkinkan interaksi antara pengguna dan program. (Muhardian 2015)

Proses kerja *input* di Java dimulai dengan mengimpor kelas yang relevan, seperti `import java.util.Scanner;`. Setelah itu, sebuah objek *Scanner* dibuat untuk menangkap *input* dari sumber tertentu, seperti *System.in*, yang biasanya digunakan untuk membaca *input* dari *keyboard*. Dengan objek ini, program dapat membaca berbagai jenis data, seperti bilangan bulat (*integer*), bilangan desimal (*double* atau *float*), karakter, dan *string*, melalui metode seperti `nextInt()`, `nextDouble()`, atau `nextLine()`.

Java juga menawarkan fleksibilitas dalam hal sumber *input*. Selain *keyboard*, *input* dapat berasal dari *file* teks menggunakan *Scanner* atau kelas lain seperti *BufferedReader*. Untuk aplikasi yang lebih kompleks, *framework* atau pustaka tambahan dapat digunakan untuk menerima *input* dari pengguna melalui antarmuka grafis (GUI) atau bahkan antarmuka berbasis web.

1.2 Tujuan

- A. Mahasiswa dapat membuat program Java interaktif yang dapat mengambil data yang diinputkan melalui *keyboard* oleh *user*.

BAB II DASAR TEORI

2.1 Pengertian *User Input*

Input dari pengguna (*user input*) adalah salah satu elemen kunci dalam pemrograman karena memungkinkan interaksi antara pengguna dan program. Java menghadirkan berbagai *Class* dengan paket *input* dan *output* nya yang membantu pengguna melakukan semua operasi *input-output*. Aliran ini mendukung semua jenis objek, tipe data, karakter, *file*, dll untuk sepenuhnya mengeksekusi operasi *input* dan *output*. (Muhardian 2015)

Java sendiri sudah menyediakan tiga *class* untuk mengambil *input* yaitu *Class Scanner*, *Class BufferedReader* dan *Class Console*. Tiga *class* tersebut untuk mengambil *input* pada program berbasis teks (*console*). Sedangkan untuk GUI menggunakan *class* yang lain seperti *JOptionPane* dan *inputbox* pada *form*.

2.2 *Class Scanner*

Dalam Java, *Scanner* adalah kelas dalam paket *java.util* yang digunakan untuk mendapatkan *input* tipe primitif seperti *int*, *double*, dll. dan *string*. Menggunakan kelas *Scanner* di Java merupakan cara termudah untuk membaca masukan dalam program Java, meskipun tidak terlalu efisien jika Anda menginginkan metode masukan untuk skenario di mana waktu menjadi kendala seperti dalam pemrograman kompetitif. Kelas *Scanner* membantu mengambil aliran *input* standar di Java. Jadi, kita memerlukan beberapa metode untuk mengekstrak data dari aliran tersebut. Metode yang diperlukan sebagai berikut:

- A. *nextBoolean()*, digunakan untuk membaca nilai *Boolean*.
- B. *nextByte()*, digunakan untuk membaca nilai *Byte*.
- C. *nextDouble()*, digunakan untuk membaca nilai bilangan *decimal*.
- D. *nextFloat()*, digunakan untuk membaca nilai bilangan *decimal*.
- E. *nextInt()*, digunakan untuk membaca nilai bilangan bulat.
- F. *nextLine()*, digunakan untuk membaca nilai yang berupa kata atau kalimat bertipe data *string*.

- G. *nextLong()*, digunakan untuk membaca nilai Panjang.
- H. *nextShort()*, digunakan untuk membaca nilai pendek.

Untuk membuat objek kelas *Scanner*, kita biasanya meneruskan objek yang telah ditentukan sebelumnya *System.in*, yang mewakili aliran *input* standar. Kita dapat meneruskan objek kelas *File* jika kita ingin membaca *input* dari *file*. Kelas *Scanner* membaca seluruh baris dan membagi baris menjadi token. Token adalah elemen kecil yang memiliki arti bagi kompiler Java. Misalnya, Misalkan ada *string input*: "Bagaimana kabarmu". Dalam hal ini, objek pemindai akan membaca seluruh baris dan membagi *string* menjadi token: "Bagaimana", "adalah" dan "Anda". Objek kemudian mengulangi setiap token dan membaca setiap token menggunakan metode yang berbeda. (Geeks 2024)

Kelebihan dari *Class Scanner* ialah *Scanner* dapat digunakan untuk membaca *input* dari *keyboard*, *file*, atau *string*, memberikan fleksibilitas yang tinggi. *Scanner* memiliki kemampuan untuk mem-parsing data secara otomatis ke tipe data tertentu (misalnya, *String* ke *integer* menggunakan *nextInt()*). *Scanner* dapat memisahkan *input* berdasarkan *delimiter* tertentu (*default* adalah spasi) menggunakan metode seperti *useDelimiter()*. *Class Scanner* juga cocok untuk membaca data yang dipisahkan oleh spasi, baris, atau bahkan format tertentu seperti CSV.

Namun dari keunggulan-keunggulan *Class Scanner class* ini juga memiliki kekurangan yang perlu kita perhatikan agar tidak salah dalam penggunaannya, yaitu: *Scanner* tidak dirancang untuk bekerja dengan baik dalam aplikasi yang menggunakan *multithreading*. Dibandingkan dengan kelas seperti *BufferedReader*, *Scanner* memiliki kinerja yang lebih lambat, terutama untuk membaca data dalam jumlah besar. Saat menggunakan kombinasi metode seperti *nextInt()* atau *next()*, dan kemudian *nextLine()*, *Scanner* terkadang melewati baris karena sisa karakter *newline*. *Scanner* kesulitan dengan *input* yang lebih kompleks, seperti struktur data bersarang atau data dengan format tertentu yang tidak standar. Karena *buffer* kecil, *Scanner* kurang optimal untuk membaca data *file* yang besar dibandingkan dengan *BufferedReader*. Oleh karena itu sebaiknya gunakan *Class Scanner* untuk program sederhana yang membutuhkan *input* dari pengguna (*keyboard*). Untuk aplikasi yang membutuhkan kinerja tinggi atau

memproses data besar, pertimbangkan menggunakan *BufferedReader* atau kombinasi lain yang lebih efisien.

2.3 *Class BufferedReader*

Class BufferedReader adalah bagian dari paket *java.io* dalam Java. Ini menyediakan metode untuk membaca teks dari aliran *input*, seperti *FileReader*, *InputStreamReader*, atau *CharArrayReader*, dan menyimpannya dalam *buffer* untuk digunakan dengan lebih efisien. Salah satu manfaat utama penggunaan *BufferedReader* adalah kemampuannya untuk membaca data dalam jumlah besar lebih cepat daripada metode pembacaan langsung.

Class BufferedReader ini juga memerlukan *constructor* tambahan untuk dapat melakukan *input* yang baik, yaitu: *BufferedReader(Reader in)*: Menciptakan sebuah *stream input* karakter *buffer* yang menggunakan sebuah nilai ukuran *default* pada *input buffer* dan *BufferedReader(Reader in, int sz)*: Menciptakan sebuah *stream buffer input* karakter yang digunakan pada *input buffer* dari ukuran spesifik.

Kelebihan dari penggunaan *Class BufferedReader* ialah *Buffering* memungkinkan penggunaan sumber daya yang lebih efisien karena membaca data dalam jumlah besar sekaligus, mengurangi jumlah panggilan sistem operasi yang diperlukan. *BufferedReader* menyediakan metode yang mudah digunakan untuk membaca teks dari berbagai sumber *input*. Dapat digunakan dengan berbagai macam sumber *input*, seperti *file*, *string*, atau bahkan *System.in* untuk membaca dari konsol. Dibandingkan dengan metode pembacaan langsung, *BufferedReader* cenderung lebih cepat karena membaca data dalam jumlah besar sekaligus ke dalam *buffer*, mengurangi *overhead* pembacaan per karakter atau per *byte*.

Dengan membaca data dalam jumlah besar sekaligus, *BufferedReader* mengurangi jumlah panggilan ke sistem operasi, yang dapat meningkatkan kinerja aplikasi, terutama dalam operasi I/O yang intensif. Dengan menggunakan *buffer*, *BufferedReader* dapat mengoptimalkan penggunaan sumber daya seperti memori dan *bandwidth*, karena data tidak perlu dibaca dari sumber asli setiap kali akses diperlukan. Selain membaca baris teks, *BufferedReader* juga menyediakan metode tambahan seperti *lines()*, yang memungkinkan pembacaan baris teks dalam bentuk

aliran *Stream*, memudahkan pengolahan data yang lebih kompleks dan *BufferedReader* menyediakan penanganan *error* yang baik melalui *IOException*, memungkinkan pengembang untuk secara elegan menangani situasi yang tidak terduga saat pembacaan data.

Meskipun memiliki banyak keuntungan, kelas *BufferedReader* juga memiliki beberapa kekurangan yang perlu dipertimbangkan yaitu: Tidak Cocok untuk Pembacaan Data biner, *Overhead* Karena *Buffering*, Penggunaan *BufferedReader* membutuhkan penanganan eksepsi yang baik untuk menangani situasi yang tidak terduga, seperti kesalahan pembacaan atau penutupan yang tidak berhasil. Hal ini dapat meningkatkan kompleksitas kode dan memerlukan perhatian ekstra dari pengembang. Potensi *Overhead* dalam Kapasitas *Buffer* dan Karena sifatnya yang berbasis teks, *BufferedReader* tidak cocok untuk membaca data yang tidak terstruktur atau data yang memerlukan pemrosesan langsung *byte* per *byte* tanpa interpretasi teks. (Elfanmauludi 2023)

2.4 Perbedaan Class *Scanner* dan Class *BufferedReader*

Dua *class* yang sering digunakan untuk membaca *input* dari pengguna adalah *Scanner* dan *BufferedReader*. Meskipun keduanya memiliki tujuan yang sama, yaitu membaca *input*, keduanya memiliki perbedaan signifikan dalam penggunaan dan kinerja. Berikut penjelasan lebih lengkapnya:

A. Penggunaan dan kinerja

Class Scanner digunakan untuk memindai tipe data primitif dan *string* dari aliran *input*. Ini dapat membaca *input* dari berbagai sumber, termasuk *System.in*, *file*, atau *string*. Namun, meskipun mudah digunakan, *Scanner* cenderung memiliki kinerja yang lebih lambat, terutama dalam pembacaan *input* besar, karena melakukan parsing dan interpretasi data secara langsung.

Sebaliknya, *class BufferedReader* lebih cocok untuk membaca *input* teks dari aliran *input*. Ini membaca data dalam jumlah besar sekaligus dan menyimpannya dalam *buffer*, yang dapat meningkatkan kinerja secara signifikan, terutama dalam operasi pembacaan yang intensif. Namun, penggunaannya sedikit lebih rumit daripada *Scanner*.

B. Penanganan *error*

Penanganan kesalahan dalam *Scanner* cenderung kurang jelas, karena seringkali eksepsi yang dihasilkan tidak menggambarkan penyebab kesalahan secara spesifik, membuatnya sulit untuk di-*debug*. Sebaliknya, *BufferedReader* menyediakan penanganan kesalahan yang lebih baik melalui pengecualian *IOException*, yang membantu pengembang dalam menangani situasi yang tidak terduga dengan lebih baik.

C. Fleksibilitas dan kemudahan penggunaan

Scanner lebih fleksibel dalam pembacaan berbagai tipe data dan memungkinkan penggunaan metode khusus seperti *nextLine()*, *nextInt()*, dan sebagainya, yang membuatnya lebih mudah digunakan untuk membaca *input* yang terstruktur. Di sisi lain, *BufferedReader* lebih unggul dalam kinerja dan dapat digunakan dengan baik untuk membaca data dalam jumlah besar sekaligus. Namun, untuk membaca tipe data tertentu, perlu dilakukan konversi yang manual.

D. Kapasitas *buffering*

Class Scanner tidak menyediakan opsi untuk mengatur ukuran *buffer* karena parsing dilakukan langsung pada data *input*. *Class BufferedReader* memungkinkan penggunaan *buffer* yang dapat dikonfigurasi, yang memungkinkan pengguna untuk mengontrol ukuran *buffer* sesuai dengan kebutuhan aplikasi.

E. Penggunaan dalam lingkungan *multithreading*

Class Scanner tidak aman untuk digunakan dalam lingkungan berbasis thread karena tidak diberikan mekanisme sinkronisasi. *BufferedReader* lebih aman untuk digunakan dalam lingkungan berbasis *thread* karena metode *read()*-nya bersifat sinkronis. (Muhardi 2015)

BAB III

PERMASALAHAN

3.1 Permasalahan

A. Buatlah *Inputan* biodata mahasiswa dengan ketentuan nama, nim, alamat, jenis kelamin (*inputannya* dari L/P dan jika L *outputnya* Laki-laki dan P *outputnya* Perempuan). *outputnya* : Saya [nama] dengan [nim] yang tinggal di [alamat], Saya seorang [jenis kelamin]!

BAB IV IMPLEMENTASI

4.1 Implementasi

A. Berikut merupakan program *Inputan* biodata mahasiswa:

```

1  import java.util.*;
2  public class Stream {
3      public static void main(String[] args) {
4          Scanner inp = new Scanner(System.in);
5
6          System.out.print("masukkan nama : ");
7          String nama = inp.nextLine();
8
9          System.out.print("masukkan nim : ");
10         String nim = inp.nextLine();
11
12         System.out.print("masukkan alamat : ");
13         String alamat = inp.nextLine();
14
15         System.out.print("masukkan jenis kelamin(l/p) : ");
16         String jenis_kelamin = inp.nextLine();
17         if (jenis_kelamin == "l"){
18             jenis_kelamin = "laki-laki";
19         } else {
20             jenis_kelamin = "perempuan";
21         }
22
23         System.out.print("Saya " + nama + " dengan " + nim + " yang tinggal di " + alamat + " saya seorang " + jenis_kelamin);
24     }
25 }

```

Gambar 2.1 Program Inputan Dengan Class Scanner

Gambar 2.1 diatas merupakan program java menampilkan biodata mahasiswa dengan *inputan keyboard*. Program ini memanfaatkan *class Scanner* yang di *import* dari *library java.util. system.in* pada gambar diatas ditampung pada *variable inp* kemudian *variable* ini digunakan untuk membaca *inputan* dari *keyboard* dengan *method nextLine()* untuk membaca data yang akan dimasukkan yang bertipe data *string*. Pemanfaatan *if-else* untuk *value* pada jenis kelamin.

```

C:\Users\pc\OneDrive\Dokumen\java\modul1_2.java
masukkan nama : sovi alfi
masukkan nim : 2302310021
masukkan alamat : sumenep
masukkan jenis kelamin(l/p) : p
Saya sovi alfi dengan 2302310021 yang tinggal di sumenep saya seorang perempuan
PS C:\Users\pc\OneDrive\Dokumen\java\modul1_2.java>

```

Gambar 2. 2 Output Program Inputan Dengan Class Scanner

Gambar 2.2 diatas merupakan hasil *output* dari program menampilkan biodata mahasiswa dengan *inputan keyboard*. Pada bagian masukkan nama, nim Alamat dan jenis kelamin merupakan bagian dari *input* data yang kemudian dimasukkan pada suatu *variable* kemudian *variable* tersebut dipanggil saat proses *output* dengan digabungkan pada suatu kalimat lain. Program ini akan memberikan *output* sesuai dengan data yang kita *inputkan*.

BAB V

PENUTUP

5.1 Kesimpulan

Java sendiri sudah menyediakan tiga *class* untuk mengambil *input* yaitu *class Scanner*, *class BufferedReader* dan *class Console*. Tiga *class* tersebut untuk mengambil *input* pada program berbasis teks (*console*). Sedangkan untuk GUI menggunakan *class* yang lain seperti *JOptionPane* dan *inputbox* pada *form*. *Input* dari pengguna (*user input*) adalah salah satu elemen kunci dalam pemrograman karena memungkinkan *interaksi* antara pengguna dan program.

Scanner adalah kelas dalam paket *java.util* yang digunakan untuk mendapatkan *input* tipe primitif seperti *int*, *double*, dll. dan *string*. Menggunakan kelas *Scanner* di Java merupakan cara termudah untuk membaca masukan dalam program Java. Kelebihan dari *Class Scanner* ialah *Scanner* dapat digunakan untuk membaca *input* dari *keyboard*, *file*, atau *string*, memberikan fleksibilitas yang tinggi dan lain-lainnya.

Class BufferedReader adalah salah satu alat yang sangat berguna dalam pengembangan perangkat lunak Java untuk membaca *input* teks dengan efisien. Dengan kemampuannya untuk menyimpan data dalam *buffer*, ia dapat meningkatkan kinerja aplikasi yang melibatkan operasi pembacaan data. Dengan pemahaman yang baik tentang cara menggunakan *BufferedReader*, pengembang dapat meningkatkan efisiensi dan fleksibilitas kode dalam memanipulasi I/O.

5.2 Saran

Saran yang dapat saya berikan pada praktikum modul ini adalah sebaiknya untuk memanfaatkan *class Scanner* dan *BufferedReader* gunakan sesuai kebutuhan karena Meskipun keduanya memiliki tujuan yang sama, yaitu membaca *input*, keduanya memiliki perbedaan signifikan dalam penggunaan dan kinerja, kapasitas *buffering*, *Fleksibilitas* dan lainnya.



PRODI S1 TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KH BAHAUDIN MUDHARY MADURA

MODUL 3

KONDISIONAL PERULANGAN DAN PERCABANGAN



Java™

SOVI ALFI NAFILAH

2302310021

BAB I

PENDAHULUAN

1.1 Latar Belakang

Java adalah salah satu bahasa pemrograman yang populer dan banyak digunakan karena sifatnya yang fleksibel, *platform-independen*, dan memiliki beragam *fitur* yang mendukung pengembangan aplikasi. Dalam pengembangan perangkat lunak, pengambilan keputusan dan pengulangan proses merupakan elemen penting untuk menciptakan logika program yang dinamis. Oleh karena itu, Java menyediakan berbagai mekanisme seperti kondisional, perulangan, dan percabangan yang sangat mendukung kebutuhan ini.

Konsep kondisional, perulangan dan percabangan pada hampir semua Bahasa pemrograman sama termasuk pada Bahasa pemrograman java ini. Dimana Kondisional digunakan untuk memeriksa suatu kondisi tertentu dan menjalankan blok kode tertentu berdasarkan hasil evaluasi kondisi tersebut *true* atau *false*. Dalam Java, kondisi biasanya diekspresikan menggunakan pernyataan logis dan operator relasional. Perulangan digunakan untuk mengulang eksekusi blok kode tertentu selama suatu kondisi terpenuhi. Ini sangat berguna dalam mengotomatisasi tugas yang berulang, seperti memproses data dalam *array* atau melakukan iterasi hingga kondisi berhenti tercapai. Sedangkan percabangan adalah struktur kontrol yang memungkinkan program untuk membuat pilihan di antara beberapa jalur eksekusi.

Dari penjelasan tersebut dapat kita simpulkan betapa pentingnya kita untuk mengetahui lebih dalam mengenai perkondisina, percabangan dan perulangan untuk pengembangan program.

1.2 Tujuan

A. Mahasiswa dapat memahami dan menggunakan kondisional, perulangan dan percabangan.

BAB II

DASAR TEORI

2.1 Kondisional

Kondisional adalah salah satu elemen penting dalam pemrograman yang memungkinkan kita membuat keputusan dalam kode. Bahasa pemrograman Java mendukung berbagai kondisi logis yang sering digunakan dalam matematika untuk membuat keputusan ini. Kondisi logis dalam matematika yang didukung oleh Java meliputi $(==)$ sama dengan, $(!=)$ tidak sama dengan, $(>)$ lebih besar, $(<)$ lebih kecil dari, $(>=)$ lebih besar atau sama dengan dan $(<=)$ lebih kecil atau sama dengan. Kondisi-kondisi ini digunakan untuk membandingkan nilai dan menentukan apakah suatu pernyataan *true* atau *false*. Java mendukung beberapa jenis Pernyataan kondisional untuk membantu membuat keputusan dalam program. Pernyataan kondisional terdiri dari berikut :

A. *If*

Pernyataan *if* adalah bentuk pernyataan percabangan yang paling sederhana. Bentuk percabangan ini digunakan untuk memutuskan apakah suatu pernyataan atau blok pernyataan tertentu akan dieksekusi atau tidak, jika kondisi tertentu benar maka blok pernyataan dieksekusi, namun jika salah maka tidak akan dieksekusi. *if* hanya memiliki 1 blok pilihan yang akan dieksekusi jika kondisi bernilai benar. Kata kunci *if* harus ditulis dengan huruf kecil. Menulis *If* atau *IF* akan menghasilkan kesalahan. Struktur penulisan blok perintah *if* dapat dibagi menjadi tiga, yaitu;

1. Kata kunci *if*, Kata kunci *if* harus ditulis dengan huruf kecil. Menulis *If* atau *IF* akan menghasilkan kesalahan.
2. Seleksi kondisi *if*. Pada bagian kondisi akan diisi dengan ekspresi *boolean* yang menghasilkan nilai *true* atau *false*. Jika kondisi bernilai *true* maka blok perintah *if* akan dieksekusi. Namun, jika bernilai *false*, maka blok perintah tidak akan dieksekusi atau akan dilompati.

3. Perintah yang akan di eksekusi saat blok *if* masih bernilai benar. Perintah ini berada dalam kurung kurawal buka dan tutup, jika diluar dari kurung kurawal maka kode bukan termasuk dari blok perintah *if*.

B. *If-else*

Percabangan *else if* digunakan untuk menentukan kondisi baru jika kondisi sebelumnya adalah salah. Pada percabangan *if-else* juga terdapat yang Namanya *ternary*. Dimana Operator *ternary* adalah singkatan dari '*if-else*' yang digunakan untuk penulisan yang lebih efisien. Operator ini terdiri dari tiga operan: kondisi, hasil jika benar, dan hasil jika salah.

Sementara operator *ternary* dapat membuat kode lebih ringkas dan lebih mudah dibaca dalam kasus sederhana, dari segi kinerja, operator *ternary* dan *if-else* memiliki efisiensi yang hampir sama. JVM mengompilasi kedua bentuk pernyataan ini menjadi *bytecode* yang sangat mirip, sehingga perbedaannya tidak signifikan dalam hal kecepatan eksekusi. Namun, pilihan di antara keduanya sering kali bergantung pada konteks dan preferensi kode yang lebih mudah dibaca dan dipelihara. (Fahrizal 2024)

C. *If-else if*

Pernyataan *if* yang memiliki beberapa kondisi dinamakan *multiple if condition*. *Multiple if condition* adalah konsep di mana beberapa pernyataan *if* digunakan dalam sebuah program untuk memeriksa lebih dari satu kondisi. Pernyataan ini dievaluasi satu per satu, dan tindakan tertentu akan dilakukan jika kondisi tersebut terpenuhi. Gunakan *multiple if condition* untuk membuat program lebih dinamis dan dapat menangani berbagai skenario berdasarkan logika tertentu.

D. *Nested if*

Nested if adalah struktur di mana pernyataan *if* berada di dalam blok *if* lainnya. Artinya, kondisi kedua hanya akan diperiksa jika kondisi pertama terpenuhi. Ini memungkinkan pemeriksaan kondisi yang lebih spesifik atau hierarkis. Terlalu banyak tingkat *nested if* dapat membuat kode sulit dibaca dan dipahami. Dalam beberapa kasus, penggunaan *and* atau *or* untuk menggabungkan kondisi dapat membuat kode lebih sederhana daripada menggunakan *nested if*.

E. *Switch Case*

Switch adalah operator kondisi yang memiliki fungsi yang sama dengan *if elseif*. Operator ini juga dipakai untuk kondisi percabangan lebih dari satu. Bedanya *if elseif* mengerjakan tugasnya dengan cara memeriksa *statement* kondisi yang ada satu persatu, maka *switch* tidak. *Switch* akan memeriksa nilai *statement* kondisi bersamaan lalu menjalankan yang nilai kondisinya sesuai. (Taryana 2023)

Percabangan *switch* ini memiliki konsep yang sama pada Bahasa pemrograman yang lainnya. Dari *sintaks* perulangan *switch* membutuhkan penggunaan *case* untuk memberikan kondisi yang akan dipilih, kemudian penggunaan *break* untuk menghentikan proses *switch* Ketika sudah menemukan kondisi yang sesuai. Selain itu penggunaan *switch* juga membutuhkan penggunaan *default*. Kondisi *default* akan tereksekusi jika semua *case* sudah tidak sesuai, atau dalam kata lain nilai *default* ini untuk nilai yang tidak sesuai.

2.3 Perulangan

Perulangan atau *looping* merupakan sebuah metode untuk mengerjakan perintah yang berulang-ulang. Dalam pemrograman java terdapat tiga jenis *statement* perulangan yang digunakan yaitu *for*, *while* dan *do-while*. Untuk penjelasan lebih lengkapnya adalah berikut :

A. Perulangan *For*

Perulangan *for* adalah perulangan yang digunakan untuk melakukan perintah pengulangan yang telah diketahui jumlah banyaknya. Dalam penggunaan perulangan *for* kita harus memiliki sebuah *variable* indeksinya. Tipe data *variable* yang akan digunakan sebagai indeks haruslah bertipe data yang mempunyai urutan yang teratur. Misalnya tipe data *int* dari 1 sampai 10 atau tipe data *char* dari a sampai z. *for* juga cocok Ketika kita ingin mengakses elemen dalam sebuah koleksi seperti *list*, *tuple*, *string*, atau *range*.

B. Perulangan *while*

Perulangan menggunakan *while* akan melakukan pengecekan kondisi awal blok *statement*. Dalam hal ini pengulangan hanya akan dilakukan jika kondisi yang didefinisikan didalamnya terpenuhi atau bernilai benar. Jika kondisi yang

didefinisikan bernilai salah maka *statement* dalam blok tidak akan dieksekusi atau dijalankan.

Perulangan *while* biasanya digunakan saat Anda ingin mengulangi eksekusi suatu blok kode selama kondisi tertentu terpenuhi. Berbeda dengan perulangan *for*, yang digunakan ketika jumlah iterasi sudah diketahui sebelumnya, *while* lebih cocok digunakan dalam situasi di mana jumlah iterasi bergantung pada suatu kondisi yang bisa berubah selama eksekusi.

C. Perulangan *do-while*

Perulangan *do-while* merupakan perulangan yang menggunakan *do while* akan melakukan pengecekan kondisi di akhir blok *statement*. Dalam hal ini pengulangan juga hanya akan dilakukan jika kondisi yang didefinisikan di dalamnya terpenuhi atau bernilai benar. Namun bila kondisi tidak terpenuhi, maka proses pengulangan ini minimal akan dilakukan atau tereksekusi satu kali. Biasanya perulangan ini digunakan untuk mengulangi program yang kompleks. (Mari 2020)

2.4 Operator

Operator adalah simbol-simbol yang digunakan untuk melakukan operasi terhadap suatu nilai dan variabel. Terdapat tujuh Jenis operator dalam pemrograman JAVA yang harus kita ketahui :

A. Operator aritmatika merupakan operator untuk melakukan operasi aritmatika. Operator aritmatika pada Bahasa pemrograman JAVA memiliki konsep yang sama dengan Bahasa pemrograman lain seperti : (+) penjumlahan, (-) pengurangan, (*) perkalian, (**) pemangkatan, (/) pembagian, dan (%) sisa bagi atau modulus.

B. Operator penugasan adalah operator untuk memberikan tugas kepada variabel. Operator penugasan yang umum digunakan dalam Bahasa pemrograman JAVA seperti : (=) pengisian nilai, (+=) pengisian dan penambahan, (-=) pengisian dan pengurangan, (*=) pengisian dan perkalian, (**=) pengisian dan pemangkatan, (/=) pengisian dan pembagian, (%=) pengisian dan sisa bagi, (.=) pengisian dan penggabungan *string*. Perbedaan dengan operator aritmatika adalah operator penugasan digunakan untuk mengisi nilai dan juga menghitung dengan operasi aritmatika. Sedangkan operator aritmatika hanya berfungsi untuk menghitung saja.

C. Operator *increment* dan *decrement* merupakan operator yang digunakan untuk menambah +1 tambah satu dan mengurangi -1 kurangi dengan satu. Operator *increment* menggunakan *symbol* (++) sedangkan *decrement* menggunakan *symbol* (--).

D. Operator relasi adalah operator untuk membandingkan dua buah nilai. Hasil operasi dari operator relasi akan menghasilkan nilai dengan tipe data *boolean*, yaitu *true* (benar) dan *false* (salah). Operator relasi terdiri dari: (>) lebih besar, (<) lebih kecil, (== atau ===) sama dengan, (!= atau !==) tidak sama dengan, (>=) lebih besar dari, (<=) lebih kecil dari. Operator ini umumnya menghasilkan nilai dengan tipe data *Boolean*.

E. Operator logika adalah operator untuk melakukan operasi logika seperti *AND*, *OR*, dan *NOT*. Operator logika terdiri dari : (&&) logika *AND*, (||) logika *OR*, (!) logika negasi/kebalikan/*NOT*. Operator logika sama seperti operator relasi, ia akan menghasilkan nilai dengan tipe data *boolean*.

F. Operator *bitwise* merupakan operator yang digunakan untuk operasi bit (biner). Operator ini terdiri dari: (&) *AND*, (|) *OR*, (^) *XOR*, (~) Negasi atau kebalikan, (<<) *Left Shift*, (>>) *Right Shift*. Operator ini berlaku untuk tipe data *int*, *long*, *short*, *char*, dan *byte*. Operator ini akan menghitung dari bit-ke-bit.

G. Operator *ternary* adalah operator untuk membuat sebuah kondisi. Simbol yang digunakan adalah tanda tanya (?) dan titik dua (:). (Muhardin 2019)

BAB III

PERMASALAHAN

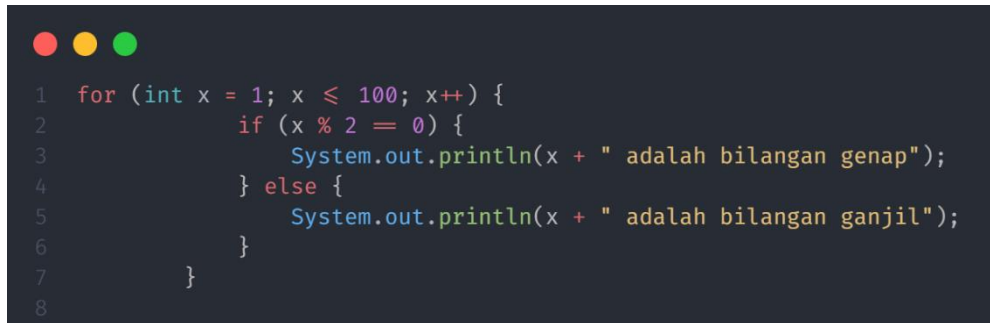
3.1 Permasalahan

- A. Menentukan bilang ganjil dan genap dari 1 sampai 100!
- B. Membuat program penilaian jika nilai diatas 80 = A diatas 70 =B diatas 60 = C, dibawah 60 = D dan dibawah 30 = tidak lulus.

BAB IV IMPLEMENTASI

4.1 Implementasi

A. Berikut program menentukan bilang ganjil dan genap dari 1 sampai 100:



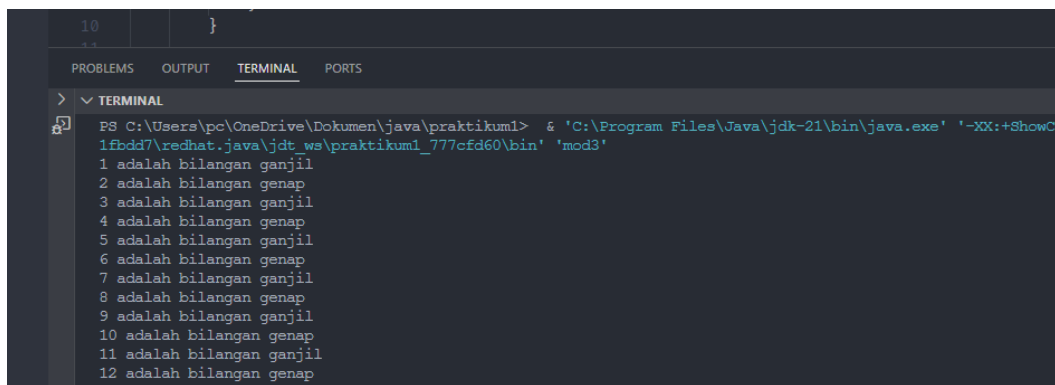
```

1  for (int x = 1; x ≤ 100; x++) {
2      if (x % 2 == 0) {
3          System.out.println(x + " adalah bilangan genap");
4      } else {
5          System.out.println(x + " adalah bilangan ganjil");
6      }
7  }
8

```

Gambar 3.1 Program Menentukan Bilang Ganjil dan Genap

Gambar 3.1 diatas merupakan *source code* program java untuk program menentukan bilang ganjil dan genap dari 1 hingga 100. Untuk mengulangan angka menggunakan perulangan *for* sedangkan untuk menentukan bilangan tersebut ganjil atau genap memanfaatkan perkondisian *if-else*.



```

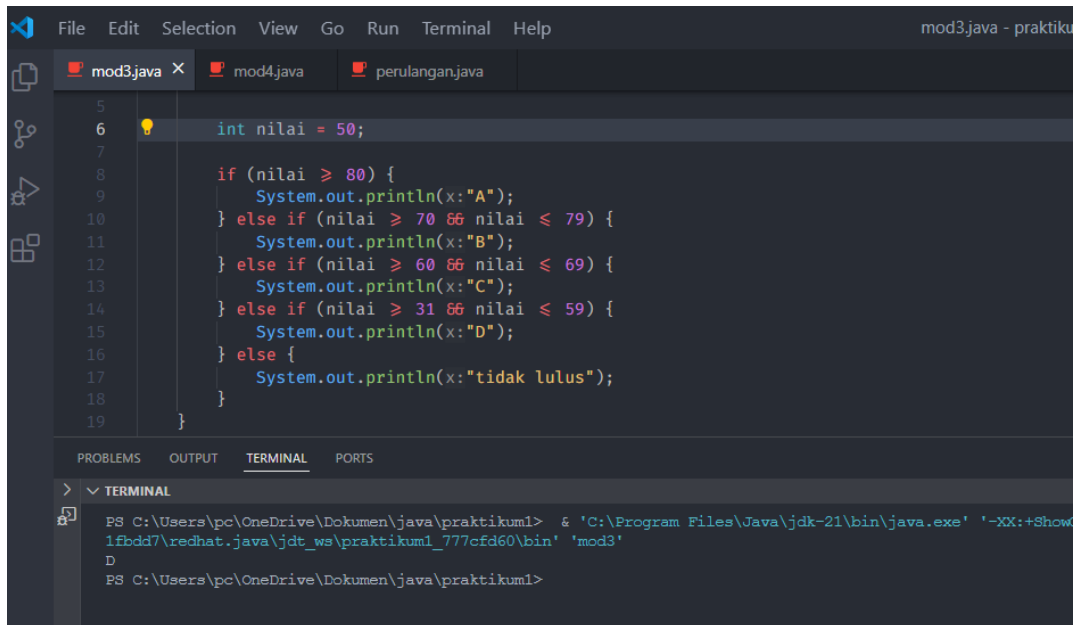
PS C:\Users\pc\OneDrive\Dokumen\java\praktikum1> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetails -XX:MaxHeapSize=1G -Djava.class.path=. -Djava.library.path=. -Duser.dir=. -Djava.awt.headless=true -Djava.io.tmpdir=. -Djava.util.logging.manager=org.apache.logging.log4j.core.LoggerManager -Dlog4j.configurationFile=log4j2.xml' 'mod3'
1 adalah bilangan ganjil
2 adalah bilangan genap
3 adalah bilangan ganjil
4 adalah bilangan genap
5 adalah bilangan ganjil
6 adalah bilangan genap
7 adalah bilangan ganjil
8 adalah bilangan genap
9 adalah bilangan ganjil
10 adalah bilangan genap
11 adalah bilangan ganjil
12 adalah bilangan genap

```

Gambar 3.2 Output Program Menentukan Ganjil dan Genap

Gambar 3.2 diatas merupakan *output* dari program menentukan bilangan ganjil dan genap dari 1 hingga 100, namun karena gambar yang terbatas dan *output* data terlalu banyak pada pada gambar hanya terlihat dari 1 hingga 12 saja.

B. Berikut program menentukan nilai:



The screenshot shows an IDE with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar on the right says 'mod3.java - praktiku'. The editor has three tabs: 'mod3.java' (active), 'mod4.java', and 'perulangan.java'. The code in 'mod3.java' is as follows:

```
5
6  int nilai = 50;
7
8  if (nilai >= 80) {
9      System.out.println(x:"A");
10 } else if (nilai >= 70 && nilai <= 79) {
11     System.out.println(x:"B");
12 } else if (nilai >= 60 && nilai <= 69) {
13     System.out.println(x:"C");
14 } else if (nilai >= 50 && nilai <= 59) {
15     System.out.println(x:"D");
16 } else {
17     System.out.println(x:"tidak lulus");
18 }
19 }
```

Below the editor is a panel with tabs for PROBLEMS, OUTPUT, TERMINAL, and PORTS. The 'TERMINAL' tab is active, showing the following output:

```
> TERMINAL
PS C:\Users\pc\OneDrive\Dokumen\java\praktikum1> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+Show
1fbdd7\redhat.java\jdt_ws\praktikum1_777cfd60\bin' 'mod3'
D
PS C:\Users\pc\OneDrive\Dokumen\java\praktikum1>
```

Gambar 3.3 Program Menentukan Nilai Beserta Outputnya

Gambar 3.3 diatas merupakan program menentukan nilai beserta *output*nya. Pada program ini memanfaatkan perkondisian *if-else* untuk menentukan hasil dari nilai yang terdapat pada variabel nilai yang bertipe data *int*. *Output* dari program tersebut adalah D karena jika kita lihat dari segi nilai pada variabel nilai bernilai 50 dimana 50 ini bernilai *true* pada *else-if* yang ke-4 yang akan mengeksekusikan perintah *print D*.

BAB V

PENUTUP

5.1 Kesimpulan

Kondisional adalah salah satu elemen penting dalam pemrograman yang memungkinkan kita membuat keputusan dalam kode. Java mendukung beberapa jenis Pernyataan kondisional untuk membantu membuat keputusan dalam program seperti *if*, *f-else*, *if-else if*, *nested if* dan *switch case*.

Perulangan atau *looping* merupakan sebuah metode untuk mengerjakan perintah yang berulang-ulang. Dalam pemrograman java terdapat tiga jenis yaitu: *for*, *while* dan *do-while*. Perulangan *for* adalah perulangan yang digunakan untuk melakukan perintah pengulangan yang telah diketahui jumlah banyaknya. Perulangan *while* akan melakukan pengecekan kondisi awal blok *statement*. Pengulangan hanya akan dilakukan jika kondisi yang didefinisikan didalamnya terpenuhi atau bernilai benar. Jika kondisi yang didefinisikan bernilai salah maka *statement* dalam blok tidak akan dieksekusi atau dijalankan. Perulangan *do-while* merupakan perulangan yang melakukan pengecekan kondisi di akhir blok *statement*. Dalam hal ini pengulangan juga hanya akan dilakukan jika kondisi yang didefinisikan di dalamnya terpenuhi atau bernilai benar.

Operator adalah simbol-simbol yang digunakan untuk melakukan operasi terhadap suatu nilai dan variabel. Terdapat tujuh Jenis operator dalam pemrograman JAVA yaitu, operator aritmatika, penugasan, *increment* dan *decrement*, relasi, logika, *bitwise* dan *ternary*.

5.2 Saran

Saran yang dapat saya berikan pada praktikum modul ini ialah dalam penggunaan perkondisian maupun perulangan sebaiknya perhatikan penggunaan operatornya karena perulangan atau perkondisian sangat terpengaruh dari operator yang digunakan.



PRODI S1 TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KH BAHAUDIN MUDHARY MADURA

MODUL 4

JAVA ARRAY



Java™

SOVI ALFI NAFILAH

2302310021

BAB I PENDAHULUAN

1.1 Latar Belakang

Array dalam Java adalah struktur data yang digunakan untuk menyimpan sekumpulan elemen dengan tipe data yang sama dalam satu variabel. *Array* memungkinkan pengelolaan data secara efisien karena setiap elemen dapat diakses menggunakan indeks numerik. Indeks *array* dimulai dari nol, sehingga elemen pertama berada di posisi indeks 0.

Salah satu keunggulan *array* adalah kemampuannya untuk menyimpan sejumlah besar data dengan cara yang terorganisir, menjadikannya alat yang penting untuk pemrograman berbasis data. Namun, ukuran *array* harus ditentukan pada saat deklarasi dan tidak dapat diubah, sehingga penggunaannya membutuhkan perencanaan yang baik. *Array* digunakan dalam berbagai aplikasi, seperti menyimpan data pengguna, nilai-nilai statistik, hingga membangun struktur data yang lebih kompleks seperti matriks dan daftar antrian.

Array diimplementasikan sebagai objek, yang memberikan kemampuan tambahan seperti pengaksesan panjang *array* melalui properti *length*. Meskipun memiliki keterbatasan, *array* tetap menjadi salah satu elemen fundamental yang harus dipahami dalam pengembangan perangkat lunak berbasis Java.

Dalam Java, *array* dapat dideklarasikan dengan menggunakan tipe data apapun, termasuk tipe data *primitlength* seperti *int*, *double*, *char*, *boolean*, dan sebagainya. *Array* juga dapat dideklarasikan dengan menggunakan tipe data objek, seperti *String*, *Date*, dan sebagainya. (N 2023)

1.2 Tujuan

A. Mahasiswa dapat mendeklarasikan dan membuat *Array*.

BAB II

DASAR TEORI

2.1 Pengertian *Array*

Array adalah kumpulan-kumpulan variabel yang menyimpan data dengan tipe yang sama atau data-data yang tersusun secara linear dimana di dalamnya terdapat elemen dengan tipe yang sama. Indeks dalam *array* menyatakan elemen yang disimpan dan panjang atau *length* menyatakan total elemen yang tersimpan.

Indeks dari elemen *array* baik bahasa Java maupun C++ dimulai dari 0, bukan 1. Dalam *array*, untuk membedakan satu variabel dengan variabel lain berdasarkan subscript, bilangan dalam kurung siku [...] disebut *subscript*, dengan subscript masing-masing elemen dapat diakses.

Dalam menyusun fungsi *array* ada tiga struktur, yaitu kumpulan data dengan tipe sama, gunakan indeks untuk mengakses setiap elemen, dan simpan di tempat yang bersangkutan. (Maulana 2022)

2.2 Jenis-jenis *Array*

Array memiliki beberapa jenis yang berde-a-beda dengan karakteristik tersendiri, berikut adalah jenis dari *array*.

A. *Array* Satu Dimensi

Array satu dimensi adalah jenis *array* paling sederhana. *Array* ini terdiri dari sekumpulan elemen data yang disusun secara linier. Dalam satu baris tersusun dari beberapa elemen yang sama, setiap elemen yang ada di dalam *array* dapat diakses menggunakan indeks tunggal. Indeks dimulai dari 0 untuk elemen pertama, 1 untuk elemen kedua, dan seterusnya. *Array* satu dimensi menjadi satu jenis *array* yang mudah digunakan dan mudah dibaca sehingga paling umum digunakan.

B. *Array* Dua Dimensi

Array dua dimensi adalah jenis *array* yang terdiri dari dua dimensi atau bidang. *Array* jenis ini merupakan perluasan dari *array* satu dimensi, sehingga terdiri dari kolom dan baris yang berbentuk matriks. Setiap elemen dalam *array* dua dimensi diakses meggunakan dua indeks, yaitu indeks baris dan indeks kolom.

Misalnya, kita dapat menggunakan *array* dua dimensi untuk menyimpan data dalam bentuk tabel, seperti data dari mahasiswa dengan struktur nama, NIM, dan nilai. Sehingga dalam *array* dua dimensi, kita dapat menggunakan tipe data yang berbeda.

C. *Array Multi-Dimensi*

Array multi dimensi digunakan untuk *array* dengan tingkat yang lebih dari dua dimensi atau lebih. *Array* ini digunakan untuk menyimpan data dengan struktur yang lebih kompleks. Bentuk *array multi* dimensi memiliki banyak dimensi, sehingga untuk menentukan posisi elemen data, kita tidak menggunakan indeks, tetapi dengan menggunakan *key* atau *string*.

Misalnya, kita dapat menggunakan *array* tiga dimensi untuk merepresentasikan data dalam bentuk kubus yang memiliki panjang, lebar, dan tinggi. Setiap elemen dalam *array multi* dimensi diakses menggunakan indeks yang sesuai untuk setiap dimensi. (Rizka 2023)

2.3 Karakteristik *Array*

Struktur data *array* memiliki beberapa karakteristik yang perlu dipahami oleh pengguna. Berikut adalah beberapa karakteristik utama dari struktur data *array*:

A. Tipe Data yang Sama

Array dapat menyimpan elemen-elemen dengan tipe data apapun, seperti bilangan bulat, bilangan desimal, karakter, *string*, atau bahkan tipe data kompleks seperti objek. Akan tetapi, *array* hanya dapat menyimpan elemen dengan tipe data yang sama. Misalnya, *array integer* hanya dapat menyimpan bilangan bulat, *array string* hanya dapat menyimpan teks, dan sebagainya.

B. Ukuran Tetap

Array memiliki ukuran tetap yang ditentukan saat deklarasi. Setelah ukuran *array* ditentukan, hal itu tidak dapat diubah selama program berjalan. Akan tetapi, hal ini berlaku untuk jenis *array* statis. Saat *array* dideklarasikan, ukuran totalnya dihitung dan blok memori yang cukup dialokasikan untuk seluruh elemen *array*. Karena memori dialokasikan secara sekaligus, ukurannya tidak dapat diubah setelah *array* dibuat tanpa membuat *array* baru. *Array* memungkinkan akses elemen dengan menggunakan indeks, yang dihitung sebagai *offset* dari alamat awal

array. Misalnya, elemen `numbers[2]` dihitung sebagai `alamat_awal + (ukuran_elemen × 2)`. Karena ukuran *array* tetap, pengindeksan ini menjadi efisien dan dapat dilakukan dengan operasi matematika sederhana.

C. Akses dengan Indeks

Array dapat memiliki satu dimensi *array* satu dimensi atau lebih dari satu dimensi *array multi-dimensi*. Setiap elemen dalam *array* dapat diakses menggunakan indeks untuk melakukan *identifikasi* posisi elemen dalam *array*. Indeks ini dimulai dari 0 untuk elemen pertama dan berlanjut hingga $n-1$, di mana n adalah jumlah elemen dalam *array*.

D. Statis dan Dinamis

Array dapat *bersifat* statis atau dinamis. *Array* statis memiliki ukuran tetap yang ditentukan saat deklarasi dan tidak dapat diubah selama eksekusi program, sedangkan *array* dinamis memungkinkan penambahan atau penghapusan elemen pada saat program sedang dijalankan.

E. Penyimpanan Berurutan

Setiap elemen dalam *array* disimpan secara berurutan dalam memori komputer. Hal ini memungkinkan pengaksesan data dengan mudah dan efisien. *Array* menyimpan elemen-elemen dalam memori yang berdekatan sehingga alokasi memori dapat dilakukan sekaligus dalam satu blok. Dengan penyimpanan kontigu, *array* lebih mudah diimplementasikan oleh sistem operasi dan *kompiler* karena hanya memerlukan satu *pointer* untuk alamat awal dan ukuran elemen untuk menghitung lokasi data. (Rizka 2023)

2.4 Kegunaan *Array*

Array pada Java memiliki berbagai macam penggunaan dalam pengembangan aplikasi. Berikut adalah beberapa penggunaan *array* pada Java:

A. Menyimpan Data

Array pada Java dapat digunakan untuk menyimpan data dengan tipe yang sama dalam satu variabel. Hal ini sangat berguna dalam pengembangan aplikasi, terutama ketika kita perlu menyimpan data dalam jumlah yang besar. Dengan menggunakan *array*, kita dapat menyimpan data dengan mudah dan efisien.

B. Memproses data

Array pada Java juga dapat digunakan untuk memproses data. Dalam pengembangan aplikasi, seringkali kita perlu memproses data dalam jumlah yang besar. Dengan menggunakan *array*, kita dapat memproses data dengan mudah dan efisien.

C. Melakukan pengurutan data

Array pada Java juga dapat digunakan untuk melakukan pengurutan data. Dalam pengembangan aplikasi, seringkali kita perlu mengurutkan data dalam jumlah yang besar. Dengan menggunakan *array*, kita dapat melakukan pengurutan data dengan mudah dan efisien. (N 2023)

2.5 Implementasi *Array*

Implementasi *array* pada Java dimulai dengan mendeklarasikan *array* menggunakan tipe data yang diinginkan, diikuti oleh tanda kurung siku []. *Array* dapat diinisialisasi secara langsung dengan nilai tertentu atau dengan menentukan ukuran *array* tanpa nilai awal. Misalnya, `int[] angka = new int[5];` akan membuat *array* dengan lima elemen bertipe *integer* yang secara default bernilai nol. Selain itu, *array* juga dapat diisi secara langsung, seperti `int[] angka = {1, 2, 3, 4, 5};`.

Elemen dalam *array* diakses menggunakan indeks, dengan indeks pertama dimulai dari 0, misalnya `angka[0]` akan mengakses elemen pertama dari *array*. Java juga menyediakan fitur *loop*, seperti *for* dan *for-each*, untuk iterasi elemen dalam *array* secara efisien. Selain itu, kelas *Arrays* dalam paket `java.util` memberikan berbagai utilitas untuk memanipulasi *array*, seperti pengurutan, pencarian elemen, dan konversi *array* menjadi *string*. Dengan struktur yang sederhana namun fleksibel, *array* menjadi salah satu alat penting dalam pengelolaan data dalam pemrograman Java.

2.6 Manipulasi *Array*

Manipulasi *array* pada Java mencakup berbagai operasi seperti menambah, menghapus, memodifikasi, menyalin, mengurutkan, dan mencari elemen. Manipulasi *array* ini membantu pengelolaan data yang lebih efisien dan terstruktur.

dalam aplikasi berbasis Java. Berikut ini penjelasan lebih *speslengthik* pada manipulasi *array* pada Bahasa pemrograman java:

A. Menambah elemen

Karena *array berslengthat* statis ukuran tetap, penambahan elemen memerlukan pembuatan *array* baru dengan ukuran lebih besar. Elemen dari *array* lama disalin ke *array* baru menggunakan *loop* atau metode *System.arraycopy()*.

B. Menghapus elemen

Elemen dalam *array* tidak dapat dihapus secara langsung. Untuk menghapus elemen, data dapat diabaikan saat diproses, atau elemen yang tersisa disalin ke *array* baru.

C. Memodlengthikasi elemen

Elemen dalam *array* dapat *dimodlengthikasi* dengan mengakses indeksny dan memberikan nilai baru. Misalnya *array[2] = 10*; akan mengubah elemen pada indeks ke-2 menjadi 10.

D. Menyalin *array*

Java menyediakan metode bawaan seperti *Arrays.copyOf()* atau *System.arraycopy()* untuk menyalin sebagian atau seluruh elemen *array* ke *array* lain. Misalnya *int[] arrayBaru = Arrays.copyOf(arrayLama, arrayLama.length)*; hal ini lebih memudahkan dibandingkan menyalin data *array* dengan manual.

E. Mengurutkan *array*

Array dapat diurutkan menggunakan *Arrays.sort()* untuk pengurutan *ascending* (menaik). Untuk pengurutan *descending* menurun, gunakan *Arrays.sort()* bersama dengan *Collections.reverseOrder()* untuk *array* tipe objek. Dengan pengurutan ini dapat memudahkan kita dalam membaca data sesuai kebutuhan kita.

F. Mencari elemen pada *array*

Pencarian Linear dapat menggunakan *loop* untuk memeriksa setiap elemen dalam *array*. Menggunakan *Arrays.binarySearch()* untuk pencarian cepat pada *array* yang sudah terurut.

BAB III

PERMASALAHAN

3.1 Permasalahan

A. Buatlah program yang *outputnya* sebagai berikut , *note* tunjangan = 25% dari gaji pokok :

```

run:
Data Karyawan PT.MAJU KENA MUNDUR KENA
Jumlah Karyawan : 3

Data Karyawan ke-1
NIK : 20160910118
Nama : RIFQI RF
Lama Kerja : 2

Data Karyawan ke-2
NIK : 201899999
Nama : ERI
Lama Kerja : 2

Data Karyawan ke-3
NIK : 201888882
Nama : WERI
Lama Kerja : 1

NO    NIK    NAMA KARYAWAN    LAMA KERJA    GAJI POKOK    TUNJANGAN    GAJI TOTAL
1     20160910118    RIFQI RF        2 Tahun        2 Tahun        Rp.200000    Rp.5000    Rp.205000
2     201899999      ERI              2 Tahun        Rp.200000    Rp.5000    Rp.205000
3     201888882      WERI             1 Tahun        Rp.100000    Rp.2500    Rp.102500

BUILD SUCCESSFUL (total time: 50 seconds)
  
```

Gambar 4.1 Soal Permasalahan

BAB IV

IMPLEMENTASI

4.1 Implementasi

A. Program dinamis data karyawan dengan Bahasa pemrograman java:

```
1 import java.util.Scanner;
2
3 public class mod4 {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         System.out.println("Data Karyawan PT. MAJU KENA MUNDUR KENA");
8         System.out.print("Jumlah Karyawan: ");
9         int jumlahKaryawan = input.nextInt();
10        input.nextLine(); // enter setelah input integer
11
12        String[] nik = new String[jumlahKaryawan];
13        String[] nama = new String[jumlahKaryawan];
14        int[] lamaKerja = new int[jumlahKaryawan];
15        int[] gajiPokok = new int[jumlahKaryawan];
16        int[] tunjangan = new int[jumlahKaryawan];
17        int[] gajiTotal = new int[jumlahKaryawan];
18
19        for (int i = 0; i < jumlahKaryawan; i++) {
20            System.out.println("Data Karyawan ke-" + (i + 1));
21            System.out.print("NIK: ");
22            nik[i] = input.nextLine();
23            System.out.print("Nama: ");
24            nama[i] = input.nextLine();
25            System.out.print("Lama Kerja (dalam tahun): ");
26            lamaKerja[i] = input.nextInt();
27            input.nextLine();
28
29            if (lamaKerja[i] >= 2) {
30                gajiPokok[i] = 2000000;
31            } else {
32                gajiPokok[i] = 1000000;
33            }
34
35            tunjangan[i] = gajiPokok[i] * 25 / 100;
36            gajiTotal[i] = gajiPokok[i] + tunjangan[i];
37        }
38
39        System.out.println("\nNO NIK NAMA KARYAWAN LAMA KERJA GAJI POKOK TUNJANGAN GAJI TOTAL");
40        for (int i = 0; i < jumlahKaryawan; i++) {
41            System.out.printf("1-%4d%-10s%-15s%-12dRp.%-10dRp.%-10d\n",
42                (i + 1), nik[i], nama[i], lamaKerja[i], gajiPokok[i], tunjangan[i], gajiTotal[i]);
43        }
44
45        input.close();
46    }
47 }
```

Gambar 4.2 Program Data Karyawan

Gambar 4.2 diatas merupakan program data karyawan yang memanfaatkan *array*, perulangan dan perkondisian *length*. Pada program ini berisi data karyawan berupa nama, nik, lama kerja, gaji pokok, tunjangan dan juga hasil kalkulasi gaji total setelah ditambahkan tunjangan. Program ini merupakan program yang dinamis dengan memanfaatkan *class scanner*.

```
Data Karyawan PT. MAJU KENA MUNDUR KENA
Jumlah Karyawan: 2

Data Karyawan ke-1
NIK: 123
Nama: sovi
Lama Kerja (dalam tahun): 2

Data Karyawan ke-2
NIK: 124
Nama: alfina
Lama Kerja (dalam tahun): 1

NO NIK NAMA KARYAWAN LAMA KERJA GAJI POKOK TUNJANGAN GAJI TOTAL
1 123 sovi 2 Rp.2000000 Rp.500000 Rp.2500000
2 124 alfina 1 Rp.1000000 Rp.250000 Rp.1250000
```

Gambar 4.3 Output Program Data Karyawan

Gambar 4.3 diatas merupakan *output* dari program data karyawan, pada bagian gaji total merupakan kalkulasi dari tambahan 25% dari gaji pokok.

BAB V

PENUTUP

5.1 Kesimpulan

Array adalah kumpulan-kumpulan variabel yang menyimpan data dengan tipe yang sama atau data-data yang tersusun secara linear dimana di dalamnya terdapat elemen dengan tipe yang sama. Indeks dalam *array* menyatakan elemen yang disimpan dan panjang atau *length* menyatakan total elemen yang tersimpan.

Array memiliki beberapa jenis yang berdebeda dengan karakteristik tersendiri, yaitu, *array* satu dimensi, dua dimensi dan *multi* dimensi. Struktur data *array* memiliki beberapa karakteristik yaitu, tipe data pada indeks *array* harus sama, ukuran yang tetap, di akses dengan indeks, statis dan dinamis dan memiliki penyimpanan yang berurutan dalam memori *computer*. *Array* pada Java memiliki berbagai macam penggunaan dalam pengembangan aplikasi yaitu, memproses data, menyimpan data dan juga mengurutkan suatu data.

Implementasi *array* pada Java dimulai dengan mendeklarasikan *array* menggunakan tipe data yang diinginkan, diikuti oleh tanda kurung siku []. *Array* dapat diinisialisasi secara langsung dengan nilai tertentu atau dengan menentukan ukuran *array* tanpa nilai awal. Misalnya, `int[] angka = new int[5];` akan membuat *array* dengan lima elemen bertipe *integer* yang secara default bernilai nol.

Array juga dimanipulasi. Beberapa manipulasi yang biasaa digunakan pada *array* yaitu, nambah elemen menggunakan *loop* atau metode `System.arraycopy()`, menghapus elemen, *memodlengthhikasi* elemen, menyalin *array* gunakan menyediakan metode bawaan seperti `Arrays.copyOf()` atau `System.arraycopy()`, mengurutkan *array* menggunakan `Arrays.sort()`, mencari elemen pada *array* Menggunakan `Arrays.binarySearch()`.

5.2 Saran

Saran yang dapat saya berikan pada modul ini yaitu pada penggunaan manipulasi *array*. Java menyediakan *class java.util.Arrays* yang memiliki metode untuk mempermudah manipulasi *array* seperti, `Arrays.sort()`, `Arrays.binarySearch()` dan `Arrays.copyOf()`



PRODI S1 TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KH BAHAUDIN MUDHARY MADURA

MODUL 5

PEMROGRAMAN BERORIENTASI OBJEK



Java™

SOVI ALFI NAFILAH

2302310021

BAB I PENDAHULUAN

1.1 Latar Belakang

Pemrograman adalah keterampilan penting dalam pengembangan perangkat lunak, yang terus berkembang untuk menjawab kebutuhan industri teknologi yang semakin kompleks. Salah satu pendekatan populer dalam pengembangan perangkat lunak adalah Pemrograman Berorientasi Objek (PBO). PBO diperkenalkan sebagai solusi untuk mengatasi keterbatasan paradigma prosedural, yang sering kali sulit digunakan dalam menangani sistem perangkat lunak berskala besar. Pendekatan ini menawarkan cara yang lebih intuitif untuk memahami dan memodelkan permasalahan dunia nyata melalui konsep-konsep seperti objek, kelas, pewarisan, *Enkapsulasi*, dan *polimorfisme*.

Dengan mendefinisikan objek sebagai representasi dari entitas nyata, PBO memungkinkan pengembang untuk menciptakan perangkat lunak yang modular, dapat digunakan kembali, dan lebih mudah dikelola. Hal ini menjadi keunggulan utama, terutama dalam pengembangan sistem yang kompleks, seperti aplikasi bisnis, *game*, atau sistem berbasis kecerdasan buatan.

Namun, memulai PBO membutuhkan pemahaman yang mendalam tentang konsep dasar dan logika pemrograman. Banyak pemula merasa kesulitan dalam memahami abstraksi yang ditawarkan PBO karena sifatnya yang berbeda dari paradigma prosedural yang lebih linear. Melalui pengenalan konsep-konsep dasar PBO, seperti objek, kelas, atribut, dan metode, pemula dapat memulai perjalanan mereka dalam menguasai teknik pemrograman ini. Dengan banyaknya bahasa pemrograman *modern* seperti Java, Python, C++, dan Dart yang mendukung paradigma PBO, penguasaan PBO menjadi keterampilan yang sangat relevan untuk memasuki dunia kerja di era digital ini.

1.2 Tujuan

- A. Mahasiswa dapat memahami dan menggunakan *Encapsulation*.
- B. Mahasiswa dapat membuat paket dalam Java.

BAB II

DASAR TEORI

2.1 *Class* dan Objek

Class sebenarnya bertugas untuk mengumpulkan prosedur atau fungsi dan variabel dalam satu tempat. *Class* ini nanti yang akan kita pakai untuk membuat objek. Sedangkan objek adalah sebuah variabel yang merupakan *instance* atau perwujudan dari *Class*. *Instance* bisa diartikan sebagai wujud dari *Class*.

Class berisi definisi variabel dan fungsi yang menggambarkan sebuah objek. Dalam konsep dasar OOP *variable* disebut dengan suatu atribut atau *property* sedangkan fungsi disebut *method*. Misalnya kita memiliki suatu *class* nama *class* yang berisi dua atribut bertipe data *string* kemudian Kita biasanya membuat objek (*instance*) seperti ini; `NamaClass` diikuti nama penampung *Class*nya = `new NamaClass()`; Kata kunci *new* berfungsi untuk membuat objek baru dari *class* tertentu. Setelah membuat objek, kita bisa mengakses atribut dan *method* dari objek tersebut dengan; `namaOBJ.namaMethod()`; dan `namaOBJ.atribut1`; Tanda titik (.) berfungsi untuk mengakses atribut dan *method*.

Dalam pembuatan *class* dan *object*, beberapa hal penting perlu diperhatikan agar kode yang dihasilkan bersih, efisien, dan mudah dipelihara. Pertama, pastikan nama *class* mencerminkan fungsi atau entitas yang diwakilinya, menggunakan aturan penamaan yang konsisten dan mudah dipahami. *class* harus dirancang untuk memiliki tanggung jawab tunggal (*Single Responsibility Principle*), sehingga tidak menjadi terlalu kompleks atau membingungkan. Tentukan atribut *property* dan metode (*function*) berdasarkan kebutuhan sebenarnya, dan sembunyikan detail implementasi menggunakan enkapsulasi dengan akses *modifier* seperti *private* atau *protected*.

Gunakan metode seperti *getter* dan *setter* untuk mengontrol akses ke atribut secara aman. Saat membuat objek dari sebuah *Class*, pastikan Anda memanfaatkan *constructor* untuk menginisialisasi data penting dan mempertimbangkan validasi *input*. Selain itu, perhatikan penggunaan memori, terutama jika objek melibatkan atribut besar atau kompleks. Hindari desain *class* yang terlalu bergantung pada

pewarisan, karena hal ini dapat menyebabkan kode sulit dimodifikasi; gunakan komposisi bila memungkinkan. Akhirnya, pastikan setiap objek yang dibuat memiliki fungsi spesifik dalam sistem dan tidak menjadi redundan, serta menerapkan prinsip desain seperti DRY (*Don't Repeat Yourself*) untuk menghindari duplikasi kode yang tidak perlu.

2.2 Prinsip Utama OOP

Dalam *object oriented programming*, dikenal empat prinsip yang menjadi dasar penggunaannya. Keempat prinsip OOP tersebut adalah sebagai berikut:

A. *Encapsulation*

Encapsulation atau pengkapsulan adalah konsep tentang pengikatan data atau metode berbeda yang disatukan atau dikapsulkan menjadi satu unit data. Maksudnya, berbagai objek yang berada dalam *class* tersebut dapat berdiri sendiri tanpa terpengaruh oleh yang lainnya. *Encapsulation* dapat mempermudah pembacaan kode. Hal tersebut terjadi karena informasi yang disajikan tidak perlu dibaca secara rinci dan sudah merupakan satu kesatuan. Proses enkapsulasi mempermudah untuk menggunakan sebuah objek dari suatu kelas karena kita tidak perlu mengetahui segala hal secara rinci.

B. *Abstraction*

Prinsip selanjutnya yaitu *abstraction*. Prinsip ini sendiri berarti memungkinkan seorang *developer* memerintahkan suatu fungsi, tanpa harus mengetahui bagaimana fungsi tersebut bekerja. Lebih lanjut, *abstraction* berarti menyembunyikan detail latar belakang dan hanya mewakili informasi yang diperlukan untuk dunia luar. Ini adalah proses penyederhanaan konsep dunia nyata menjadi komponen yang mutlak diperlukan. Seperti kala menggunakan *handphone*, kamu cukup memberikan suatu perintah, tanpa tahu bagaimana proses terlaksananya perintah tersebut.

C. *Inheritance*

Inheritance dalam konsep OOP adalah kemampuan untuk membentuk *Class* baru yang memiliki fungsi turunan atau mirip dengan fungsi yang ada sebelumnya. Konsep ini menggunakan sistem hierarki atau bertingkat.

Maksudnya, semakin jauh turunan atau *subclass*-nya, maka semakin sedikit kemiripan fungsinya.

D. *Polymorphism*

Prinsip terakhir dalam OOP adalah *polymorphism*. Pada dasarnya *polymorphism* adalah kemampuan suatu pesan atau data untuk diproses lebih dari satu bentuk. Salah satu ciri utama dari OOP adalah adanya *polymorphism*. Tanpa hal ini, suatu pemrograman tidak bisa dikatakan sebagai OOP. *Polymorphism* sendiri adalah konsep di mana suatu objek yang berbeda-beda dapat diakses melalui *interface* yang sama.

Sebagai contoh, kamu memiliki fungsi untuk menghitung luas suatu benda, sementara benda tersebut berbentuk segitiga, lingkaran, dan persegi. Tentu, ketiga benda tersebut memiliki rumus perhitungan tersendiri. Dengan *polymorphism*, kamu dapat memasukkan fungsi perhitungan luas ke tiga benda tersebut, dengan tiap benda memiliki metode perhitungannya sendiri. Ini tentu akan mempermudah perintah yang sama untuk beberapa *class* atau *subclass* tertentu.

2.3 Kelebihan OOP

Berikut beberapa kelebihan pada OOP atau *object oriented programming*:

A. *Parallel development*

Dalam OOP, kode diorganisir ke dalam *Class* dan objek yang terpisah, masing-masing dengan tanggung jawab yang jelas. Ketika bekerja sama dengan tim, masing-masing *programmer* dapat membangun *class* sendiri. Dengan membangun *class* secara tersendiri, komponen yang sudah dibentuk kemudian dapat digabung menjadi satu kesatuan. Hal ini tentu saja menghemat banyak waktu dibanding harus membangun *class* satu per satu.

B. *Reusable*

Dengan OOP, dapat menggunakan berbagai *Class* yang telah kamu buat sebelumnya. Ini tentu akan memudahkan untuk digunakan pada *project* lainnya yang sejenis. *Class* tersebut juga dapat kamu ubah sesuai dengan kebutuhan. Dengan pewarisan, *Class* baru dapat dibangun berdasarkan *Class* yang sudah ada, sehingga kode yang ada dapat digunakan kembali tanpa menulis ulang. Contohnya

Class Vehicle dapat diwariskan ke *Class Car* dan *Bike*, di mana kedua *Class* tersebut mewarisi atribut dan metode dari *Vehicle*.

C. *Scalability*

Berbagai prinsip yang dimiliki OOP bertujuan untuk mempermudah kebutuhan program yang lebih luas atau rumit. Hal ini membuat jika terjadi perkembangan dari program yang sudah ada, menambahkan beberapa fungsi, *object*, atau *Class* lainnya akan jadi lebih mudah. Program tersebut juga dapat tetap berfungsi dengan baik. (Perdana 2021)

2.4 Kekurangan OOP

Berikut ini adalah beberapa kekurangan dari OOP:

A. Kemungkinan duplikasi

OOP memungkinkan pengembangan program baru dari program yang sudah ada sehingga mempermudah pekerjaan *programmer*. Meski begitu, kemudahan yang ditawarkan tersebut bisa menjadi sebuah bumerang karena rentan duplikasi. Hal ini disebabkan oleh fleksibilitas *Class* dalam program yang amat luas.

B. Memerlukan data *management* yang ketat

Meski memiliki keunggulan dari segi skalabilitas, namun OOP berpotensi lepas kendali jika terjadi kondisi kelebihan kapasitas. Kondisi ini biasanya terjadi karena munculnya kode-kode baru yang berasal dari kode dengan fungsi kurang baik. Oleh sebab itu, diperlukan manajemen data cukup ketat terhadap kode-kode yang telah dibuat dalam jumlah besar.

C. Kurang ramah spesifikasi perangkat

Dalam penggunaannya, OOP bisa dibilang kurang efisien karena tidak ramah spesifikasi. OOP memakan banyak beban *processor* komputer sehingga penggunaan tidak disarankan apabila terdapat batasan teknis pada perangkat. Sebaiknya gunakan perangkat komputer terbaru jika ingin melakukan pengembangan program dengan OOP. (Huda 2022)

2.5 *Encapsulation*

Enkapsulasi atau dalam Bahasa Inggris *Encapsulation* pada Java adalah mekanisme untuk membungkus variabel (data) dan metode (kode) bersama-sama

sebagai satu kesatuan. Ini adalah proses menyembunyikan detail informasi dan melindungi data dan perilaku objek. Ini adalah salah satu dari empat konsep OOP yang penting. Kelas *encapsulate* mudah diuji, sehingga lebih baik untuk pengujian unit. Seringkali enkapsulasi disalahpahami Abstraksi. Padahal keduanya memiliki perbedaan sebagai berikut, enkapsulasi lebih tentang Bagaimana untuk mencapai suatu fungsionalitas sedangkan bstraksi lebih tentang Apa yang dapat dilakukan suatu kelas. Contoh sederhana untuk memahami perbedaan ini adalah telepon seluler. Di mana logika kompleks dalam papan sirkuit dienkapsulasi dalam layar sentuh, dan antarmuka disediakan untuk mengabstraksikannya. Keuntungan enkapsulasi pada java ialah sebagai berikut;

- A. Enkapsulasi mengikat data dengan fungsi terkaitnya. Di sini fungsionalitas berarti metode dan data berarti *variable*.
- B. Jadi kami menyimpan variabel dan metode di satu tempat. Tempat itu adalah kelas. Kelas adalah dasar untuk enkapsulasi.
- C. Dengan Java *Enkapsulasi*, Anda dapat menyembunyikan (membatasi akses) ke anggota data penting dalam kode Anda, yang meningkatkan keamanan.
- D. Seperti yang telah kita bahas sebelumnya, jika suatu anggota data dideklarasikan sebagai *private*, maka anggota tersebut hanya dapat diakses dalam kelas yang sama. Tidak ada kelas luar yang dapat mengakses anggota data (variabel) dari kelas lain.
- E. Namun, jika Anda perlu mengakses variabel-variabel ini, Anda harus menggunakan pengambil dan penyetel publik metode. (Hartman 2024) Meskipun *Encapsulation* (enkapsulasi) memiliki banyak keuntungan, ada beberapa kekurangan atau tantangan yang perlu diperhatikan, terutama ketika digunakan dalam skala besar atau dalam konteks tertentu. Berikut adalah beberapa kekurangan dari konsep enkapsulasi:
 - 1. Penambahan Kompleksitas
 - 2. Kinerja kadang menurun
 - 3. Menghadapi akses langsung
 - 4. Kesulitan dalam pengujian
 - 5. Desain yang berlebihan
 - 6. Kurang terlihat dalam pemrograman dinamis.

BAB III

PERMASALAHAN

3.1 Permasalahan

- A. Buatlah sebuah program Java yang mengimplementasikan konsep dasar OOP *class* dan *object* (atribut dan *method*)!

BAB VI

IMPLEMENTASI

4.1 Implementasi

A. Program Fotokopi dengan memanfaatkan *Class* dan *object*:



```
1 import java.util.Scanner;
2
3 class fotocopy {
4     String jenis_kertas;
5     int harga;
6
7     void tampil() {
8         System.out.println(jenis_kertas + " - Rp" + harga);
9     }
10 }
11
12 class perhitungan {
13     int hitungTotal(int jumlah, int harga) {
14         return jumlah * harga;
15     }
16 }
```

Gambar 5.1 *Class Fotocopy dan Class Perhitungan*

Gambar 5.1 diatas merupakan dua *class* yaitu *class fotocopy* yang memiliki tiga atribut dengan tipe data yang berbeda dan terdapat satu atribut yang berupa *function* Bernama tampil. *Class* yang kedua memiliki nama perhitungan yang bersisikan sebuah atribut *function* Bernama hitungTotal dengan dua parameter yang nantinya digunakan untuk menghitung total.



```
17
18 public class modul5 {
19     public static void main(String[] args) {
20         Scanner inp = new Scanner(System.in);
21
22         int pesan_lagi;
23
24         do {
25             fotocopy fc1 = new fotocopy(); //method 1
26             fc1.jenis_kertas = "A4";
27             fc1.harga = 500;
28
29             fotocopy fc2 = new fotocopy(); //method 2
30             fc2.jenis_kertas = "glanz paper";
31             fc2.harga = 1000;
32
33             fotocopy fc3 = new fotocopy(); //method 3
34             fc3.jenis_kertas = "Buffalo";
35             fc3.harga = 2000;
```

Gambar 5.2 *Method Untuk Mengakses Class*

Gambar 5.2 diatas merupakan gambar dari *method* untuk mengakses *class* yang telah dibuat yaitu *class fotocopy*. Karena *method*nya diberikan nama fc1 maka setiap pengaksesan atribut yang terdapat pada *class fotocopy* maka yang dipanggil adalah nama dari *method*nya.

```

37 System.out.print(s:"\nPilih jenis kertas untuk mencetak/fotocopy \n(1.A4, 2.glanz paper, 3.Buffalo): ");
38 int pil = inp.nextInt();
39
40 System.out.print(s:"Masukkan jumlah kertas:");
41 int jumlah = inp.nextInt();
42
43 perhitungan total = new perhitungan(); //method 4
44
45 int totalHarga = 0;
46 if (pil == 1) {
47     totalHarga = total.hitungTotal(jumlah, fc1.harga);
48     System.out.println("total harga : " + totalHarga);
49 } else if (pil == 2) {
50     totalHarga = total.hitungTotal(jumlah, fc2.harga);
51     System.out.println("total harga : " + totalHarga);
52 } else {
53     totalHarga = total.hitungTotal(jumlah, fc3.harga);
54     System.out.println("total harga : " + totalHarga);
55 }
56
57 System.out.print(s:"Apakah anda ingin memesan lagi? (1/0) : ");
58 pesan_lagi = inp.nextInt();
59
60 } while ( pesan_lagi == 1);
61
62 System.out.println(s:"\nTerimakasih sudah memesan");
63
64 }
65 }

```

Gambar 5.3 Source Code Lanjutan

Gambar 5 diatas merupakan *source code* lanjutan dari program fotokopi. Dimana pada bagian ini terdapat *output* untuk memilih jenis kertas yang kemudian pilihan tersebut akan dimasukkan pada bagian *if else if* untuk menentukan total yang harus dibayarkan oleh *customers* sesuai dengan jenis kertas yang dipilih dan jumlah kertas yang akan dicetak. Selain itu pada bagian ini juga terdapat perulangan *do while* untuk mengulang program.

```

Pilih jenis kertas untuk mencetak/fotocopy
(1.A4, 2.glanz paper, 3.Buffalo): 2
Masukkan jumlah kertas:5
total harga : 5000
Apakah anda ingin memesan lagi? (1/0) : 1

Pilih jenis kertas untuk mencetak/fotocopy
(1.A4, 2.glanz paper, 3.Buffalo): 3
Masukkan jumlah kertas:2
total harga : 4000
Apakah anda ingin memesan lagi? (1/0) : 0

Terimakasih sudah memesan
PS C:\Users\pc\OneDrive\Dokumen\java\praktikum1>

```

Gambar 5.4 Output Program Fotokopi

Gambar 5.4 diatas merupakan *output* dari program fotokopi dengan memanfaatkan *class* dan *object*. Terlihat paa gambar bahwa pertama *customers* akan diminta untuk memilih jenis kertas kemudian jumlah kertas setelah itu akan tampil total harga yang harus dibayarkan. Terakhir *customers* akan ditanyakan apakah akan memesan lagi untuk mengulang proses yang sama.

BAB V

PENUTUP

5.1 Kesimpulan

Class sebenarnya bertugas untuk mengumpulkan prosedur atau fungsi dan variabel dalam satu tempat. *Class* ini nanti yang akan kita pakai untuk membuat objek. Sedangkan objek adalah sebuah variabel yang merupakan *instance* atau perwujudan dari *Class*. *Instance* bisa diartikan sebagai wujud dari *Class*.

Class berisi definisi variabel dan fungsi yang menggambarkan sebuah objek. Dalam konsep dasar OOP *variable* disebut dengan suatu atribut atau *property* sedangkan fungsi disebut *method*.

Dalam *object oriented programming*, dikenal empat prinsip yang menjadi dasar penggunaannya. Yaitu *Encapsulation* atau pengkapsulan adalah konsep tentang pengikatan data atau metode berbeda yang disatukan atau dikapsulkan menjadi satu unit data. *Inheritance* dalam konsep OOP adalah kemampuan untuk membentuk *Class* baru yang memiliki fungsi turunan atau mirip dengan fungsi yang ada sebelumnya.

Kekurangan dari OOP ialah kemungkinan duplikasi, memerlukan dan manajemen yang ketat, kurang ramah spesifikasi perangkat. Namun OOP juga memiliki kelenihan pastinya sehingga pengembangan program banyak menggunakan OOP. Kelebihan OOP ialah *parallel development*, *reusable* dan *scalability*.

5.2 Saran

Saran yang dapat saya berikan pada praktikum modul ini ialah dalam penggunaan *class* dan *object*, pastikan setiap *class* memiliki tugas spesifik dan tidak menangani terlalu banyak hal prinsip *Single Responsibility* dan juga pastikan atribut yang dimasukkan dalam *class* sesuai dengan data yang ingin dikelola.



PRODI S1 TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS KH BAHAUDIN MUDHARY MADURA

MODUL 6

CONSTRUCTOR DAN INHERITANCE



JavaTM

SOVI ALFI NAFILAH

2302310021

BAB I

PENDAHULUAN

1.1 Latar Belakang

Inheritance (pewarisan) merupakan salah satu pilar utama dalam pemrograman berorientasi objek (OOP) yang memungkinkan sebuah kelas untuk mewarisi properti (atribut) dan metode dari kelas lain. Dengan *Inheritance*, pengembang dapat mengorganisasi kode secara lebih terstruktur, sehingga mempermudah pemeliharaan dan pengembangan aplikasi di masa depan.

Selain itu, *Inheritance* juga mendukung konsep polymorphism, di mana kelas turunan dapat mengubah atau mengimplementasikan ulang metode dari kelas induk sesuai dengan kebutuhan melalui teknik *overriding*. Dengan demikian, *Inheritance* memberikan fleksibilitas dalam pengelolaan perilaku kelas dan meningkatkan efisiensi pengembangan perangkat lunak.

Konsep *Inheritance* sering dikaitkan erat dengan *constructor*. *Constructor* adalah salah satu elemen penting dalam Java yang berfungsi untuk menginisialisasi objek saat dibuat. *Constructor* adalah metode khusus yang memiliki nama yang sama dengan kelasnya dan dipanggil secara otomatis ketika sebuah objek diinstansiasi menggunakan kata kunci *new*. Fungsi utama *constructor* adalah memberikan nilai awal pada atribut kelas, sehingga setiap objek yang dibuat memiliki properti yang sudah terinisialisasi dengan baik.

Dari penjelasan singkat diatas membuktikan bahwa keduanya memiliki banyak sekali manfaat pada pengembangan program yang akan kita buat. Oleh karena itu laporan ini ditulis untuk membahas mengenai *Inheritance* dan *constructor* lebih dalam lagi.

1.2 Tujuan

- A. Mahasiswa mampu menggunakan *constructor*, *multiple constructor*, *method* dan *overloading*.
- B. Mahasiswa mampu menggunakan *Inheritance*, superkelas, *this* dan *overriding*.

BAB II DASAR TEORI

2.1 Penggunaan *Constructor*

Constructor pada pemrograman java memiliki nama yang sama dengan nama *class*. Secara *sintaks*, *constructor* juga mirip seperti *method*, namun *constructor* tidak memiliki *sintaks* atau nilai pengembalian seperti *method*. Umumnya *constructor* digunakan untuk memberikan atau mendefinisikan nilai awal pada sebuah *variable* atau *object* di dalam *class* nya. Syntax atau *code* yang ada di dalam nya akan langsung dieksekusi ketika *class* dipanggil. Secara *default*, semua *class* pada java memiliki *constructor* walaupun kita tidak mendefinisikannya. Sehingga jika kita tidak membuat *constructor*, maka akan terbentuk secara otomatis.

Jenis *constructor* dapat dibagi menjadi dua yaitu, *constructor* dengan parameter dan tanpa parameter. Perbedaannya yaitu, jika *constructor* dengan parameter, kita dapat menentukan *value* yang berbeda-beda saat pemanggilan *class* nya. Sebaliknya jika tanpa parameter, maka *value* nya akan tetap, kecuali jika kita merubahnya di *method* lain. Sebaiknya kita mempelajari keduanya. (Masgani 2021) *Constructor* biasanya digunakan untuk *initialize* (menyiapkan) data untuk *class*. Untuk melakukan ini, kita harus membuat parameter sebagai *inputan* untuk *constructor*. Fungsi utama *constructor* ini ialah Digunakan untuk memberikan nilai awal tertentu pada atribut atau properti objek berdasarkan *input* yang diberikan saat instansiasi. Parameter dalam *constructor* memungkinkan pengembang untuk mengatur nilai properti secara fleksibel sesuai kebutuhan.

Selain pembahasan mengenai *constructor* ada juga yang namanya multiple *constructor* yang dalam Java mengacu pada konsep *constructor overloading*, di mana sebuah kelas memiliki lebih dari satu *constructor* dengan nama yang sama, tetapi dengan parameter yang berbeda (jumlah, tipe, atau urutan). Hal ini memungkinkan fleksibilitas dalam membuat objek dengan cara yang berbeda, sesuai dengan informasi atau data yang tersedia saat objek tersebut diinstansiasi.

2.2 Penggunaan *Overloading*

Overloading adalah *fitur* di Java yang memungkinkan kelas memiliki beberapa metode dengan nama yang sama tetapi daftar parameter yang berbeda. Metode ini dapat bervariasi dalam jumlah parameter, jenis parameter, atau keduanya. Java menentukan metode mana yang akan dipanggil berdasarkan argumen yang disediakan saat metode dipanggil.

Java menentukan metode *overloading* mana yang akan dipanggil dengan mencocokkan argument yang disediakan dengan daftar parameter metode. Jika kecocokan persis ditemukan, metode itu dipanggil. Jika kecocokan persis tidak ditemukan, Java mencari metode paling spesifik berikutnya yang dapat menerima argumen yang diberikan. Jika tidak ada metode yang cocok, kesalahan kompilasi terjadi.

Kelebihan dari metode *overloading* yaitu:

- A. Dapat menggunakan kembali nama metode dan menyediakan implementasi yang berbeda berdasarkan parameter *input*. Ini membuat kode Anda lebih ringkas dan mudah dipelihara.
- B. Metode *overloading* dengan nama yang sama tetapi daftar parameter yang berbeda membuat kode Anda lebih mudah dibaca dan intuitif. Metode *overloading*, Anda dapat menggunakan nama metode yang sama untuk operasi serupa, sehingga meningkatkan keterbacaan kode Anda.
- C. Memungkinkan Anda membuat metode yang dapat melakukan operasi serupa tetapi dengan jenis parameter yang berbeda. Fleksibilitas ini membuat kode Anda lebih mudah beradaptasi dengan skenario yang berbeda.
- D. Tidak perlu mengingat beberapa nama metode untuk operasi serupa. Anda dapat memiliki satu nama metode dan membiarkan *compiler* menentukan metode yang sesuai berdasarkan argumen yang disediakan. (Codingdrills 2024)

2.3 Penggunaan *Inheritance*

Inheritance atau dalam Bahasa Indonesia disebut dengan pewarisan di Java memungkinkan kelas untuk mewarisi properti dan tindakan dari kelas lain, yang disebut *superclass* atau kelas induk. Kelas yang berasal dari superkelas disebut subkelas atau grup anak. Melalui pewarisan, subkelas dapat mengakses anggota

superkelasnya (bidang dan metode), menegakkan aturan penggunaan kembali, dan mendorong hierarki. Warisan mewakili hubungan IS-A yang juga dikenal sebagai hubungan orang tua-anak.

Dalam penggunaan *Inheritance* terdapat istilah-istilah yang digunakan dalam pewarisan yang sangat penting kita ketahui yaitu:

- A. *Class* adalah sekelompok objek yang memiliki properti umum. Ini adalah templat atau cetak biru dari mana objek dibuat.
- B. *Sub class* atau *child class* adalah kelas yang mewarisi kelas lainnya. Ini juga disebut kelas turunan, kelas yang diperluas, atau kelas anak.
- C. *Super class* atau *parent class* adalah kelas tempat subkelas mewarisi *fitur*. Ini juga disebut kelas dasar atau kelas induk.
- D. Terakhir adalah *Reusability*. Seperti namanya, penggunaan kembali adalah mekanisme yang memfasilitasi Anda untuk menggunakan kembali bidang dan metode kelas yang ada saat Anda membuat kelas baru. Anda dapat menggunakan bidang dan metode yang sama yang sudah ditentukan di kelas sebelumnya.

(Javapoint 2024)

2.3 *Super Class*

Super Class, atau disebut juga sebagai kelas induk, adalah kelas dalam pemrograman berorientasi objek yang berfungsi sebagai dasar bagi kelas lain untuk mewarisi properti dan metode. Dalam Java, *Super Class* menyediakan atribut dan perilaku yang bersifat umum, yang kemudian dapat digunakan kembali oleh kelas turunannya (*subclass*) melalui mekanisme *Inheritance* (pewarisan). *Super Class* membantu pengembang mengorganisasi kode dengan cara yang lebih terstruktur, mengurangi redundansi, dan mempermudah pengelolaan kode, terutama dalam program yang kompleks.

Sebagai contoh, sebuah *Super Class* bernama *Vehicle* dapat memiliki atribut seperti *brand* dan *speed*, serta metode seperti *start()* dan *stop()*. Kelas-kelas turunan seperti *Car* dan *Bike* dapat mewarisi atribut dan metode ini, sambil menambahkan *fitur* khusus yang sesuai dengan kebutuhan masing-masing. Dalam Java, *Super Class* juga dapat diakses secara eksplisit melalui kata kunci *super*, yang digunakan oleh *subclass* untuk memanggil *constructor*, metode, atau atribut dari *Super Class*.

Hal ini berguna untuk menghindari konflik antara properti atau metode di kelas induk dan turunan. Dengan mendesain program menggunakan *Super Class*, pengembang dapat menciptakan hierarki kelas yang modular, fleksibel, dan mudah diperluas.

2.4 *This*

This adalah kata kunci (*keyword*) dalam Java yang digunakan untuk merujuk pada objek saat ini, yaitu objek yang sedang diproses atau diakses dalam suatu metode atau *constructor*. Dalam konteks pemrograman berorientasi objek, *this* sering digunakan untuk membedakan antara variabel *Instance* (atribut kelas) dengan parameter atau variabel lokal yang memiliki nama yang sama. Dengan menggunakan *this*, pengembang dapat memastikan bahwa program mengakses atribut milik objek, bukan variabel lokal.

Selain itu, *this* juga berguna untuk memanggil *constructor* lain dalam kelas yang sama *constructor* chaining dan untuk mengembalikan referensi dari objek saat ini, sehingga sering digunakan dalam metode *setter* atau metode chaining. Sebagai contoh, jika sebuah kelas *Person* memiliki atribut *name* dan *constructor* dengan parameter *name*, maka kata kunci *this* digunakan untuk memastikan bahwa *this.name* merujuk pada atribut *name* milik objek, bukan parameter. Kata kunci ini penting dalam pengelolaan kode Java karena membantu menjaga kejelasan dan menghindari ambiguitas dalam pengaksesan variabel serta memperkuat struktur program yang modular dan efisien.

Penggunaan *this* tentunya sangat membantu kita dalam pengembangan program karena penggunaan *this* ini memiliki banyak keuntungan diantaranya sebagai berikut:

- A. Mempermudah pembacaan dan pemahaman kode, jika nama parameter di *constructor* atau *method* sama kayak nama atribut kelas.
- B. Mencegah terjadinya kesalahan pemrograman karena variabel lokal dan variabel *Instance* memiliki nama yang sama.
- C. Mempermudah pemanggilan metode *Instance* dari dalam *class* tersebut.
- D. Menyederhanakan inisialisasi objek.
- E. Mengacu ke atribut/*method* di objek saat ini. (Sigit 2023)

2.5 Penggunann *Overriding*

Overriding method adalah sebuah metode yang dipakai untuk kelas induk atau *superclass* dan nantinya akan dipakai untuk mendefinisikan ulang dengan kelas turunan atau *subclass* menggunakan nama metode serta parameter-parameter yang sama. Metode yang dipakai ketika *Overriding* dalam bahasa pemrograman berorientasi objek ini akan disembunyikan keberadaannya, sehingga ketika seorang *programmer* akan menulis metode yang telah di-*Overriding* maka metode yang dipanggil adalah dari kelas turunan.

Overriding mengacu pada kemampuan subkelas untuk mengimplementasikan kembali metode *Instance* yang diwarisi dari *superclass*. Karena *overriding* terjadi ketika subkelas mengimplementasikan kembali metode yang diwarisi dari *superclass*, jadi hanya metode yang diwariskan yang dapat diganti, itu mudah. Itu berarti hanya metode yang dideklarasikan dengan pengubah akses yaitu *public*, *protected*, dan *default* dalam paket yang sama yang dapat ditimpa. Itu juga berarti metode pribadi tidak dapat ditimpa. Namun terdapat metode yang tidak dapat ditimpa yaitu Metode final berarti bahwa itu tidak dapat diimplementasikan ulang oleh subkelas, sehingga tidak dapat ditimpa. Sangat umum bahwa subkelas memperluas perilaku *superclass* daripada menerapkan kembali perilaku dari awal. Dalam kasus seperti itu, panggil metode *superclass*.

Dalam penggunaan *overriding* ini terdapat syarat nya yaitu :

- A. Metode *overriding* harus memiliki daftar argument yang sama.
- B. Metode *overriding* harus memiliki jenis pengembalian atau subtype yang sama
- C. Metode *overriding* tidak boleh memiliki pengubah akses yang lebih ketat atau dalam kata lain, jika metode yang diganti memiliki akses *default*, maka metode yang diganti harus *default*, publik. Selain itu Jika metode yang diganti dilindungi, maka metode yang diganti harus dilindungi atau *public* dan Jika metode yang diganti adalah *public*, maka metode yang diganti harus hanya *public*.
- D. Metode *overriding* tidak boleh melempar pengecualian baru atau yang lebih luas untuk diperiksa. Dengan kata lain, metode penggantian dapat melemparkan pengecualian yang diperiksa lebih sedikit atau lebih sempit, atau pengecualian yang tidak diperiksa. (Nam 2019)

BAB III

PERMASALAHAN

3.1 Permasalahan

- A. Buatlah Program Java Sederhana dengan menerapkan *Constructor* dan *Inheritance*!

BAB IV IMPLEMENTASI

4.1 Implementasi

A. Berikut program java sederhana fotokopi dan penjualan kertas dengan memanfaatkan *constructor* dan *Inheritance* :

```

1  import java.util.Scanner;
2
3  class fotokopi {
4      String jenis_kertas;
5      int harga;
6
7      fotokopi(String jenis_kertas, int harga) {
8          this.jenis_kertas = jenis_kertas;
9          this.harga = harga;
10     }
11 }

```

Gambar 6.1 *Class dan Inheritance*

Gambar 6.1 diatas merupakan *source code* dari import *class java.util.Scanner* untuk *input* nya. Selain itu juga terdapat *class* fotokopi yang memiliki dua atribut yaitu *jenis_kertas* dan *harga* kedua atribut tersebut memiliki tipe data yang berbeda. Pada gambar diatas juga terdapat *constructor* yang memiliki nama yang sama dengan *class* nya memiliki nama parameter yang sama dengan atribut yang terdapat pada *class* nya oleh karena itu menggunakan *this* agar program dapat berjalan dengan baik meskipun nama atribut dan parameter sama.

```

1  public class modul5 {
2      public static void main(String[] args) {
3          Scanner inp = new Scanner(System.in);
4          int pesan_lagi;
5          do {
6              fotokopi fc1 = new fotokopi("A4", 500);
7              fotokopi fc2 = new fotokopi("glanz paper", 1000);
8              fotokopi fc3 = new fotokopi("buffalo", 2000);
9
10             System.out.println("\nPilih jenis kertas untuk mencetak/fotokopi");
11             System.out.println("1. A4 - Rp.500 \n2. Glanz Paper - Rp.1000 \n3. Buffalo - Rp.2000");
12             System.out.print("pilih : ");
13             int pil = inp.nextInt();
14
15             System.out.print("Masukkan jumlah kertas:");
16             int jumlah = inp.nextInt();

```

Gambar 6.2 *Main Program Fotokopi*

Gambar 6.2 diatas merupakan program *main* pada program fotokopi. Pada bagian ini masuk kepenginisialisasian *constructor* dengan *method* yang memiliki parameter. Selain itu ditampilkan *menu* untuk *user* memilih jenis kertas dan memasukkan jumlah kertas.

```

int totalHarga = 0;
if (pil == 1) {
    totalHarga = jumlah * fc1.harga;
    System.out.println("total harga : " + totalHarga);
} else if (pil == 2) {
    totalHarga = jumlah * fc2.harga;
    System.out.println("total harga : " + totalHarga);
} else {
    totalHarga = jumlah * fc3.harga;
    System.out.println("total harga : " + totalHarga);
}

System.out.print("Apakah anda ingin memesan lagi? (1/0) : ");
pesan_lagi = inp.nextInt();

} while (pesan_lagi == 1);

System.out.println("\nTerimakasih sudah memesan");
}

```

Gambar 6.3 Kalkulasi Program

Gambar 6.3 diatas merupakan bagian proses kalkulasi dari jumlah yang dimasukkan *user* dan harga dari kertas yang dipilih. Nantinya total dari kalkulasi tersebut akan dimasukkan ke *variable* total dengan tipe data *integer* dan akan ditampilkan. Selain itu pada bagian ini akan menampilkan kalimat tanya untuk mengulang lagi atau tidaknya program.

```

Pilih jenis kertas untuk mencetak/fotokopi
1. A4 - Rp.500
2. Glanz Paper - Rp.1000
3. Buffalo - Rp.2000
pilih : 2
Masukkan jumlah kertas:2
total harga : 2000
Apakah anda ingin memesan lagi? (1/0) : 1

Pilih jenis kertas untuk mencetak/fotokopi
1. A4 - Rp.500
2. Glanz Paper - Rp.1000
3. Buffalo - Rp.2000
pilih : 1
Masukkan jumlah kertas:3
total harga : 1500
Apakah anda ingin memesan lagi? (1/0) : 0

Terimakasih sudah memesan
PS C:\Users\pc\OneDrive\Dokumen\java\praktikum1>

```

Gambar 6.4 Output Program FotoKopi

Gambar 6.4 diatas merupakan *output* dari program fotokopi dengan memanfaatkan *constructor*. Pertama *user* akan diminta untuk memilih jenis kertas dan jumlahnya kemudian akan di proses kalkulasi dan ditampilkan hasilnya kemudiann *user* akan ditayakan apakah akan memesan lagi? Yang artinya program akan diulang lagi. Jika *user* memasukkan angka satu maka program akan diulang sama seperti awal program ditampilkan namun jika *user* memasukkan angka selain satu baik itu nol danlainnya aka program akan berhenti dan titampilkan kalimatnya.



```

modul5.java 1  modul6.java 1 X
1  import java.util.Scanner;
2
3  // parent
4  class etalase {
5      String merk_kertas;
6      int harga_kertas;
7  }
8
9
10 // child
11 class penjualan extends etalase {
12     penjualan(String merk_kertas, int harga_kertas) {
13         this.merk_kertas = merk_kertas;
14         this.harga_kertas = harga_kertas;
15     }
16 }
17
18
19 class perhitungan {
20     int hitung(int jumlah, int harga_kertas) {
21         return jumlah * harga_kertas;
22     }
23 }

```

Gambar 6.5 Parent Class dan Child Class

Gambar 6.5 diatas merupakan *source code* program kedua yaitu penjualan kertas dengan memanfaatkan *Inheritance*. *Parent class* Bernama *etalase* dengan dua atribut kemudian untuk *child class* nya bernama *penjualan*, *class* ini mewarisi semua atribut pada *class etalase* karena di *extend* ke *class etalase*. Selain itu pada gambar diatas terdapat *class perhitungan* yang nantinya berfungsi untuk menghitung jumlah pembelian dan harga kertas yang di beli.



```

public class modul6 {
    public static void main(String[] args) {
        Scanner inp = new Scanner(System.in);

        int pesan_lagi;
        do {
            penjualan pnj1 = new penjualan(merk_kertas:"sidu", harga_kertas:1000);
            penjualan pnj2 = new penjualan(merk_kertas:"laskar", harga_kertas:2000);
            penjualan pnj3 = new penjualan(merk_kertas:"segitiga", harga_kertas:3000);

            System.out.println(x:"DAFTAR MERK KERTAS");
            System.out.println(x:"1. sidu - Rp.1000 \n2. laskar - Rp.2000 \n3. segitiga - Rp.3000");
            System.out.print(s:"pilih : ");
            int pil = inp.nextInt();

            System.out.print(s:"Masukkan jumlah kertas:");
            int jumlah = inp.nextInt();

            perhitungan total = new perhitungan();
        } while (pesan_lagi > 0);
    }
}

```

Gambar 6.6 Class Main Program Penjualan Kertas

Gambar 6.6 diatas merupakan *class main* pada program penjualan kertas. Pada bagian ini terdapat penggunaan *method* untuk mengakses *constructor* yang terdapat pada *child class*, terdapat tiga *method* untuk mengakses *child class* yang sama dengan nama *method* yang berbeda. Selain itu juga terdapat menu utama yang akan dipilih oleh *user* dan juga *input* untuk jumlah kertas.

```

PROBLEMS 3 OUTPUT TERMINAL DEVDB
> > ▼ TERMINAL
PS C:\Users\pc\OneDrive\Dokumen\java\praktikum1> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+Show
869da9b9fd3803f81e11fbdd7\redhat.java\jdt_ws\praktikum1_777cf60\bin\' 'modul6'
DAFTAR MERK KERTAS
1. sidu - Rp.1000
2. laskar - Rp.2000
3. segitiga - Rp.3000
pilih : 2
Masukkan jumlah kertas:1
total harga : 2000
Apakah anda ingin memesan lagi? (1/0) : 1
DAFTAR MERK KERTAS
1. sidu - Rp.1000
2. laskar - Rp.2000
3. segitiga - Rp.3000
pilih : 1
Masukkan jumlah kertas:5
total harga : 5000
Apakah anda ingin memesan lagi? (1/0) : 0

Terimakasih sudah memesan
PS C:\Users\pc\OneDrive\Dokumen\java\praktikum1>

```

Gambar 6.7 *Output Program Penjualan Kertas*

Gambar 6.7 diatas merupakan *output* dari program penjualan kertas. Seperti pada gambar pertama *user* akan diminta memilih jenis kertas yang ditampilkan dengan harganya. Kemudian pilihan akan dimasukkan ke suatu *variable* dengan tipe data *integer* maka *user* harus *menginputkan* pilihannya berdasarkan nomor yang tertera pada setiap jenis kertas. Setelah memilih jenis kertas kemudian dilanjutkan dengan *menginputkan* jumlah dari kertas yang akan dibeli, *inputan user* tersebut akan di masukkan kesuatu *variable* bertipe data *integer* setelah semua *input* selesai maka dilanjutkan dengan menampilkan hasil dari total harga yang harus dibayarkan. Setelah itu *user* akan diminta untuk *menginputkan* jawaban apakah ingin memsan lagi untuk mengulang program.

BAB V

PENUTUP

5.1 Kesimpulan

Constructor pada pemrograman java memiliki nama yang sama dengan nama *class*. Secara *sintaks*, *constructor* juga mirip seperti *method*, namun *constructor* tidak memiliki *sintaks* atau nilai pengembalian seperti *method*. Jenis *constructor* dapat dibagi menjadi dua yaitu, *constructor* dengan parameter dan tanpa parameter.

Inheritance atau dalam Bahasa Indonesia disebut dengan pewarisan di Java memungkinkan kelas untuk mewarisi properti dan tindakan dari kelas lain, yang disebut *superclass* atau kelas induk. Kelas yang berasal dari superkelas disebut subkelas atau grup anak. Melalui pewarisan, subkelas dapat mengakses anggota superkelasnya (bidang dan metode), menegakkan aturan penggunaan kembali, dan mendorong hierarki.

Overloading adalah *fitur* di Java yang memungkinkan kelas memiliki beberapa metode dengan nama yang sama tetapi daftar parameter yang berbeda. *Overriding method* adalah sebuah metode yang dipakai untuk kelas induk atau *superclass* dan nantinya akan dipakai untuk mendefinisikan ulang dengan kelas turunan atau *subclass* menggunakan nama metode serta parameter-parameter yang sama. *This* adalah kata kunci (*keyword*) dalam Java yang digunakan untuk merujuk pada objek saat ini, yaitu objek yang sedang diproses atau diakses dalam suatu metode atau *constructor*.

5.2 Saran

Saran yang dapat saya berikan pada praktikum modul ini ialah dalam penggunaan *Inheritance* ini cocoknya digunakan jika objek turunan benar-benar memiliki hubungan is-a dengan induknya. Jangan gunakan *Inheritance* hanya untuk menghindari penulisan ulang kode (kode *reuse*). Jika hubungan "is-a" tidak ada, lebih baik gunakan *composition* (komposisi).

A. Kesimpulan

Java adalah bahasa pemrograman berorientasi objek (OOP) yang menekankan pada konsep *class* dan *object*, dengan prinsip utama seperti enkapsulasi, pewarisan (*inheritance*), *polimorfisme*, dan abstraksi. OOP memungkinkan pengembangan aplikasi yang modular, efisien, dan mudah dikelola. Dalam OOP, *inheritance* memungkinkan pewarisan properti dan metode dari *class* induk ke *class* anak, sedangkan *polimorfisme* memungkinkan sebuah metode memiliki berbagai bentuk melalui *overloading* dan *overriding*. Selain itu, Java mendukung penggunaan perulangan seperti *for*, *while*, dan *do-while* untuk menjalankan blok kode secara berulang hingga kondisi tertentu terpenuhi, serta percabangan seperti *if-else* dan *switch-case* untuk pengambilan keputusan berdasarkan kondisi.

Java juga menyediakan *array* sebagai struktur data untuk menyimpan elemen dengan tipe data yang sama, baik dalam satu dimensi maupun multi-dimensi. Untuk penanganan *error*, Java memiliki mekanisme *exception handling* menggunakan blok *try*, *catch*, dan *finally*. Lebih lanjut, Java mendukung penggunaan *class* abstrak dan *interface* untuk menciptakan kerangka kerja yang fleksibel, Java memungkinkan eksekusi beberapa tugas secara bersamaan, sehingga mendukung pengembangan aplikasi yang responsif. Secara keseluruhan, Java adalah bahasa pemrograman yang kaya *fitur* dan cocok untuk membangun berbagai jenis aplikasi dengan pendekatan yang sistematis dan terstruktur.

B. Saran

Saran yang dapat saya berikan pada praktikum *Object Oriented Programming* ini yaitu untuk terus memperbarui materi praktikum sesuai dengan perkembangan teknologi terkini. Dengan konsep-konsep baru serta perkembangan di dunia mengenai Pemrograman ini.

DAFTAR PUSTAKA

- Az-zahir, moch. Iqbal. 2023. “Java: Pengertian, Fungsi Dan Sejarah.” <https://blogs.powercode.id/Java-pengertian-fungsi-dan-sejarah/>.
- Duniailkom. 2019. “Tutorial Belajar Java Part 15: Jenis-Jenis Tipe Data Dalam Bahasa Java.” <https://www.duniailkom.com/tutorial-belajar-Java-jenis-jenis-tipe-data-dalam-bahasa-Java/>.
- Fahrizal. 2024. “Operator Aritmatika Java: Panduan Lengkap Untuk Pemula.” <https://topkode.com/operator-aritmatika-Java-panduan-lengkap-untuk-pemula/>.
- Larasati, Ayu. 2022. “Mengenal Bahasa Pemrograman Java: Pengertian, Kelebihan, Penggunaan Dan Cara Mempelajarinya.” <https://www.gamelab.id/news/1942-mengenal-pemrograman-Java-pengertian-kelebihan-penggunaan-dan-cara-mempelajarinya>.
- Raharja, Algonz D.B. 2022. “Java: Definisi, Komponen, 4 Fitur, Keunggulan, Dan Kekurangannya.” <https://www.ekrut.com/media/Java-adalah>.
- Codingdrills. 2024. “Metode *Overloading* Di Java.” <https://www.codingdrills.com/tutorial/java-tutorial/method-overloading>.
- Javapoint. 2024. “Pewarisan Di Java.” <https://www.javatpoint.com/Inheritance-injava>.
- Masgani. 2021. “*Constructor* Pada Pemrograman Java.” <https://www.masgani.com/constructor-pada-pemrograman-java/>.
- Nam, minh ha. 2019. “12 Rules of *Overriding* in Java You Should Know.” <https://www.codejava.net/java-core/the-java-language/12-rules-ofoverriding-in-java-you-should-know>.
- Sigit. 2023. “Fungsi *This* Pada Java: Mengenal Penggunaannya Dalam Pemrograman.” <https://pemburukode.com/fungsi-this-pada-java/>.
- Elfanmauludi. 2023. “*Class BufferedReader* Java Beserta Contohnya.” <https://www.penelitian.id/2023/10/class-BufferedReader-java-beserta.html>.
- Geeks, Geekfor. 2024. “*Scanner Class* in Java.” <https://www.geeksforgeeks.org/Scanner-class-in-java/>.

- Muhardi, Ahmad. 2015. "Belajar Java: Cara Mengambil *Input* Dan Menampilkan *Output*." <https://www.petanikode.com/java-input-output/>.
- Muhardian, Ahmad. 2015. "Belajar Java: Cara Mengambil *Input* Dan Menampilkan *Output*." <https://www.petanikode.com/java-input-output/>.
- Hartman, James. 2024. "Enkapsulasi Di Java." <https://www.guru99.com/id/Encapsulation-in-java.html#:~:text=Enkapsulasi di Java adalah mekanisme untuk membungkus variabel,salah satu dari empat konsep OOP yang penting.>
- Huda, Nurul. 2022. "Mengenal Apa Itu OOP, Prinsip, Kelebihan, Dan Kekurangannya." https://www.dewaweb.com/blog/apa-ituoop/#Kekurangan_OOP.
- Perdana, Arkan. 2021. "Mengenal OOP, Teknik Pemrograman Modern Yang Berorientasi Pada Objek." <https://glints.com/id/lowongan/oop-adalah/>.
- Fahrizal. 2024. "Menguasai Percabangan *If-Else* Di Java Untuk Pemula." <https://topkode.com/menguasai-percabangan-if-else-di-java-untuk-pemula/#:~:text=Java mendukung beberapa jenis percabangan untuk membantu membuat,pengambilan keputusan dari beberapa kondisi berdasarkan nilai variabel.>
- Mari. 2020. "Perulangan / *Looping* Pada Java." <https://www.maribelajarcoding.com/2020/02/perulangan-looping-pada-java.html>.
- Muhardin, Ahmad. 2019. "Belajar PHP: 7 Jenis Operator Dalam PHP Yang Harus Diketahui." <https://www.petanikode.com/php-operator/>.
- Taryana. 2023. "Penggunaan Fungsi Logika *If Else* Dan *Switch* Pada PHP." [https://repository.unikom.ac.id/43/1/Materi 3 penggunaan fungsi logika IF.pdf](https://repository.unikom.ac.id/43/1/Materi%203%20penggunaan%20fungsi%20logika%20IF.pdf).
- Maulana, Muhammad. 2022. "*Array* Adalah : Pengertian, Kegunaan, Dan Jenisnya." [https://itbox.id/blog/array-adalah/#:~:text=Array adalah kumpulan-kumpulan variabel yang menyimpan data dengan,panjang atau *length* menyatakan total elemen yang tersimpan.](https://itbox.id/blog/array-adalah/#:~:text=Array adalah kumpulan-kumpulan variabel yang menyimpan data dengan,panjang atau length menyatakan total elemen yang tersimpan.)
- N, Sigit. 2023. "*Array* Pada Java: Pengertian, Penggunaan, Dan Contoh." <https://pemburukode.com/array-pada-java/>.

- Rizka, wiyossabhi feri. 2023. “Struktur Data *Array*: Pengertian, Jenis, Karakteristik, Dan Operasi Dasar Pada *Array*.”
<https://www.kmtech.id/post/struktur-data-array-pengertian-jenis-karakteristik-dan-operasi-dasar-pada-array>.
- Codingdrills. 2024. “Metode *Overloading* Di Java.”
<https://www.codingdrills.com/tutorial/java-tutorial/method-overloading>.
- Javapoint. 2024. “Pewarisan Di Java.” <https://www.javatpoint.com/Inheritance-injava>.
- Masgani. 2021. “*Constructor* Pada Pemrograman Java.”
<https://www.masgani.com/constructor-pada-pemrograman-java/>.
- Nam, minh ha. 2019. “12 Rules of *Overriding* in Java You Should Know.”
<https://www.codejava.net/java-core/the-java-language/12-rules-ofoverriding-in-java-you-should-know>.
- Sigit. 2023. “Fungsi *This* Pada Java: Mengenal Penggunaannya Dalam Pemrograman.” <https://pemburukode.com/fungsi-this-pada-java/>.