



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

FINAL REPORT

Object Oriented Programming



JavaTM

ZAIFUR ROHMAN

2302310175

LEMBAR PENGESAHAN
PRAKTIKUM *OBJECT-ORIENTED PROGRAMING*

PROGRAM STUDI INFORMATIKA
UNIVERSITAS K.H. BAHAUDIN MUDHARY MADURA

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



ZAIFUR ROHMAN

NIM .2302310175

Laporan ini telah direvisi dan disetujui oleh Asisten Praktikum dan Dosen
Pengampu pada tanggal 28 Desember 2024

Mengetahui,
Assiten Pengampu Praktikum

JEKI SERYODI

NIM. 2202310002

Menyetujui,
Dosen Pengampu Mata Kuliah

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

NIP. 19890330.202203.1.079

LEMBAR ASISTENSI

KATA PENGANTAR

Segala puji syukur mari kita panjatkan kehadiran Tuhan Yang Maha Esa karena berkat rahmat, hidayah dan karunia-Nya maka mahasiswa dapat menyelesaikan laporan praktikum ini. Laporan ini ditujukan untuk memenuhi tugas akhir dari praktikum mata kuliah *Object-Oriented Programming*.

Shalawat dan salam semoga senantiasa tercurah limpahkan kepada junjungan besar kita, yaitu Nabi Muhammad SAW. Mahasiswa menyadari bahwa tanpa bantuan dan bimbingan dari berbagai pihak, sangatlah sulit bagi mahasiswa untuk dapat menyelesaikan laporan praktikum ini. Oleh karena itu, mahasiswa mengucapkan terima kasih kepada asisten praktikum dan juga pihak yang telah membantu mahasiswa dengan sangat amat baiknya, tanpanya mahasiswa belum tentu bisa menyelesaikan semua ini.

Mahasiswa menyadari bahwa pembuatan dan penyusunan laporan praktikum ini masih banyak terdapat kekurangan dan jauh dari kata sempurna, hal ini dikarenakan keterbatasan dan kemampuan mahasiswa sebagai penulis ini. Atas segala kekurangan dan ketidak sempurnaan laporan praktikum ini, saya mengharapkan masukan, kritik maupun saran yang bisa membangun kearah perbaikan dan penyempurnaan laporan praktikum ini.

Akhir kata mahasiswa berharap Tuhan Yang Maha Esa berkenan membalas segala kebaikan semua pihak yang telah membantu khususnya pihak yang memberikan bantuan khusus kepada mahasiswa, semoga laporan praktikum ini memberi manfaat bagi kita semua.

Sumenep, 25 Desember 2024

Zaifur Rohman

DAFTAR ISI

COVER
LEMBAR PENGESAHAN	ii
LEMBAR ASISTENSI	iii
KATA PENGANTAR.....	iv
DAFTAR ISI.....	v
DAFTAR GAMBAR	vii
MODUL I MEMULAI 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 <i>USER INPUT</i>.....	11
BAB I PENDAHULUAN	12
BAB II DASAR TEORI.....	13
BAB III PERMASALAHAN.....	18
BAB IV IMPLEMENTASI.....	19
BAB V PENUTUP.....	20
MODUL I 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 MEMULAI PEMROGRAMAN BERORIENTASI OBJEK	42
BAB I PENDAHULUAN	43
BAB II DASAR TEORI.....	44
BAB III PERMASALAHAN.....	49
BAB IV IMPLEMENTASI.....	50
BAB V PENUTUP.....	51
 MODUL IV <i>CONSTRUCTOR</i> DAN <i>INHERITANCE</i>.....	52
BAB I PENDAHULUAN	53
BAB II DASAR TEORI.....	54
BAB III PERMASALAHAN.....	59
BAB IV IMPLEMENTASI.....	60
BAB V PENUTUP.....	61
 A. KESIMPULAN	62
B. SARAN.....	62
DAFTAR PUSTAKA.....	63

DAFTAR GAMBAR

Gambar 1. 1 Kode Program.....	9
Gambar 1. 2 <i>Output</i> Program	9
Gambar 2. 1 Program Biodata Mahasiswa	19
Gambar 2. 2 <i>Output</i> Program Biodata Diri.....	19
Gambar 3. 1 Program Bilangan Ganjil genap.....	29
Gambar 3. 2 <i>Output</i> Program	29
Gambar 3. 3 Program Penilaian.....	30
Gambar 3. 4 <i>Output</i> Program	30
Gambar 4. 1 Program Menghitung Gaji	40
Gambar 4. 2 <i>Output</i> Program	40
Gambar 5. 1 Program OOP.....	50
Gambar 5. 2 <i>Output</i> Program	50
Gambar 6. 1 Program <i>Constructor</i>	60
Gambar 6. 2 <i>Output</i> Program	60



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

MODUL 1

PEMROGRAMAN JAVA



JavaTM

ZAIFUR ROHMAN

2302310175

BAB I

PENDAHULUAN

1.1 Latar Belakang

Bahasa pemrograman Java merupakan salah satu bahasa yang paling banyak digunakan di dunia saat ini, dikenal karena fleksibilitas dan kemampuannya untuk dijalankan di berbagai platform tanpa perlu modifikasi kode. Diciptakan oleh tim yang dipimpin oleh James Gosling di Sun Microsystems pada awal 1990-an, Java awalnya dikenal dengan nama Oak, yang ditujukan untuk perangkat konsumen seperti televisi *interaktif*. Namun, karena nama Oak sudah digunakan oleh perusahaan lain, pada tahun 1995 nama tersebut diubah menjadi Java, terinspirasi dari kopi yang berasal dari Pulau Jawa.

Java dirilis secara resmi pada 23 Mei 1995 dan segera menjadi populer berkat kemampuannya untuk mendukung pengembangan aplikasi berbasis *web*. Konsep "*Write Once, Run Anywhere*" (WORA) menjadi motto utama Java, menandakan bahwa aplikasi yang ditulis dalam bahasa ini dapat dijalankan di berbagai sistem operasi tanpa perlu disusun ulang. Sejak saat itu, Java telah berkembang pesat dan menjadi fondasi bagi banyak teknologi modern, termasuk aplikasi *mobile* dan *enterprise*. Kelebihan utama dari Java meliputi pustaka yang lengkap, dukungan terhadap pemrograman berorientasi objek, serta komunitas pengguna yang besar, meskipun juga memiliki kekurangan seperti kebutuhan memori yang lebih besar dan kerentanan terhadap pencurian kode sumber.

1.2 Tujuan

- a. Mahasiswa mampu membuat program sederhana menggunakan Java.
- b. Mahasiswa mampu mengkompilasi dan menjalankan program Java.

BAB II

DASAR TEORI

2.1 Pengertian Java

Java adalah bahasa pemrograman berorientasi objek yang sangat populer dan banyak digunakan di seluruh dunia. Dikenal dengan moto "*Write Once, Run Anywhere*" (WORA), Java memungkinkan pengembang untuk menulis kode yang dapat dijalankan di berbagai platform tanpa perlu melakukan penyesuaian ulang. Ini dimungkinkan berkat penggunaan *Java Virtual Machine* (JVM), yang mengonversi kode sumber Java menjadi *bytecode* yang dapat dijalankan di berbagai sistem operasi, termasuk Windows, Linux, dan Android.

Java memiliki beberapa komponen penting, yaitu *Java Development Kit* (JDK), *Java Runtime Environment* (JRE), dan *Java Virtual Machine* (JVM). JDK menyediakan alat dan pustaka yang diperlukan untuk mengembangkan aplikasi Java, sementara JRE adalah lingkungan tempat aplikasi Java dijalankan. JVM bertugas mengeksekusi *bytecode* yang dihasilkan dari program Java.

Java memiliki *sintaks* yang mirip dengan bahasa pemrograman C dan C++, sehingga relatif mudah dipelajari bagi mereka yang sudah familiar dengan bahasa-bahasa tersebut. Kode Java dikompilasi menjadi *bytecode* yang dijalankan oleh *Java Virtual Machine* (JVM), yang berfungsi sebagai lapisan antara kode dan perangkat keras, memungkinkan aplikasi Java untuk beroperasi di berbagai platform. Saat ini, Java digunakan secara luas dalam berbagai aplikasi, mulai dari aplikasi *mobile* hingga perangkat lunak perusahaan, dan telah terpasang di lebih dari 13 miliar perangkat di seluruh dunia.

2.2 Struktur Java

Struktur dasar program Java terdiri dari beberapa komponen yang saling berinteraksi untuk membentuk keseluruhan aplikasi.

a. Deklarasi Paket

Poin pertama dalam struktur program Java adalah deklarasi paket, berfungsi untuk mengorganisir dan mengelompokkan kelas-kelas yang saling terkait dalam

suatu *namespace* tertentu. Dengan menggunakan paket, pengembang dapat menghindari konflik nama antara kelas yang berbeda dan memudahkan pengelolaan proyek, terutama ketika proyek tersebut berkembang menjadi lebih besar dan kompleks. Selain itu, paket juga memudahkan distribusi dan penggunaan kembali kode di berbagai proyek lainnya.

b. *Import Library*

Poin kedua adalah bagian *import*, di mana pengembang dapat mengakses berbagai pustaka atau *library* eksternal yang menyediakan fungsionalitas tambahan yang tidak tersedia secara langsung dalam bahasa Java. Melalui pernyataan *import*, pengembang dapat menggunakan kelas-kelas dari pustaka standar Java atau pustaka pihak ketiga tanpa harus menulis ulang kode dari awal. Ini sangat penting karena memungkinkan pengembang untuk mempercepat proses pengembangan dengan memanfaatkan solusi yang sudah ada, sehingga mereka dapat fokus pada logika bisnis aplikasi mereka.

c. *Bagian Kelas*

Poin ketiga dalam struktur program Java adalah bagian kelas, yang merupakan unit dasar dalam pemrograman berorientasi objek. Setiap program Java harus memiliki minimal satu kelas, dan di dalam kelas tersebut terdapat metode-metode yang mendefinisikan perilaku objek yang diwakili oleh kelas itu. Kelas memungkinkan pengembang untuk mengelompokkan data dan fungsi terkait dalam satu entitas, sehingga meningkatkan modularitas dan keterbacaan kode. Dengan menggunakan prinsip-prinsip pemrograman berorientasi objek, seperti enkapsulasi dan pewarisan, pengembang dapat menciptakan aplikasi yang lebih terstruktur dan mudah dipelihara.

d. *Metode Main*

Poin keempat adalah metode *main*, yang merupakan titik awal eksekusi program Java. Metode ini sangat penting karena di sinilah semua proses eksekusi dimulai ketika program dijalankan oleh Java *Virtual Machine* (JVM). Metode *main* menerima parameter berupa *Array string* yang memungkinkan pengguna untuk memberikan argumen saat menjalankan program dari *command line*. Dengan adanya metode *main*, pengembang dapat menentukan alur eksekusi program secara

jelas dan terstruktur, serta memanggil metode lain atau menjalankan logika bisnis sesuai kebutuhan.

e. Blok Program

Poin kelima adalah blok program, yang terdiri dari kumpulan pernyataan dan ekspresi yang dikelompokkan bersama dalam kurung kurawal. Blok kode ini memungkinkan pengembang untuk mengorganisir logika program dengan cara yang lebih terstruktur dan mudah dipahami. Setiap blok dapat berisi pernyataan variabel, kontrol aliran seperti percabangan dan perulangan, serta pemanggilan metode lainnya. Dengan menggunakan blok kode, pengembang dapat menciptakan struktur yang jelas dan logis dalam aplikasi mereka, membuatnya lebih mudah untuk dibaca dan dipelihara oleh programmer lain di masa depan.

2.3 Tipe Data

Dalam pemrograman Java, terdapat dua kategori utama tipe data, yaitu tipe data primitif dan tipe data objek. Tipe data primitif terdiri dari beberapa jenis yang digunakan untuk menyimpan nilai dasar, sedangkan tipe data objek digunakan untuk menyimpan kumpulan nilai atau objek yang lebih kompleks.

a. Tipe Data Primitif

Tipe data primitif di Java mencakup delapan jenis, masing-masing dengan karakteristik dan rentang nilai yang berbeda:

1. *byte*

Tipe data ini digunakan untuk menyimpan bilangan bulat kecil dengan ukuran 8 bit. Rentang nilai yang dapat disimpan adalah dari -128 hingga 127. *Byte* sering digunakan ketika memori terbatas dan nilai yang diperlukan dalam rentang tersebut.

2. *Short*

Tipe data ini menyimpan bilangan bulat dengan ukuran 16 bit. Rentang nilai yang dapat disimpan adalah dari -32,768 hingga 32,767. *Short* berguna untuk menghemat memori jika dibandingkan dengan tipe data *int* ketika nilai yang diperlukan tidak terlalu besar.

3. *Int*

Tipe data ini adalah bilangan bulat dengan ukuran 32 bit dan merupakan tipe yang paling umum digunakan dalam pemrograman Java. Rentang nilai untuk *int*

adalah dari -2,147,483,648 hingga 2,147,483,647. Tipe ini cocok untuk menyimpan angka yang lebih besar dibandingkan *byte* dan *short*.

4. *Long*

Tipe ini digunakan untuk menyimpan bilangan bulat yang sangat besar dengan ukuran 64 bit. Rentang nilainya adalah dari -9,223,372,036,854,775,808 hingga 9,223,372,036,854,775,807. *Long* diperlukan ketika nilai yang akan disimpan melebihi kapasitas tipe *int*.

5. *Float*

Tipe data ini digunakan untuk menyimpan bilangan desimal dengan presisi tunggal (32 bit). Rentang nilainya berkisar antara 1.4E-45 hingga 3.4E+38. *Float* sering digunakan untuk perhitungan yang membutuhkan desimal tetapi tidak memerlukan presisi tinggi.

6. *Double*

Tipe ini juga digunakan untuk menyimpan bilangan desimal tetapi dengan presisi ganda (64 bit). Rentang nilainya jauh lebih luas dibandingkan *float*, yaitu dari 4.9E-324 hingga 1.8E+308. *Double* lebih akurat dan sering digunakan dalam perhitungan matematis kompleks.

7. *Char*

Tipe data ini menyimpan satu karakter *unicode* dengan ukuran 16 bit. *Char* dapat menyimpan karakter apa pun dari berbagai bahasa dan simbol dengan rentang nilai dari '\u0000' (null *character*) hingga '\uffff' (karakter *Unicode* maksimum).

8. *Boolean*

Tipe ini hanya memiliki dua kemungkinan nilai yaitu *true* atau *false*. *Boolean* sering digunakan dalam logika pemrograman untuk menentukan kondisi atau status tertentu.

b. Tipe Data Objek

Selain tipe data primitif, Java juga memiliki beberapa tipe data objek yang lebih kompleks:

1. *String*

Tipe data ini digunakan untuk menyimpan teks atau kumpulan karakter. *String* sangat penting dalam pengembangan aplikasi karena sering digunakan untuk menampilkan informasi kepada pengguna.

2. *Array*

Array adalah struktur data yang memungkinkan penyimpanan sejumlah elemen dengan tipe yang sama dalam satu variabel. Ini memudahkan pengelolaan koleksi data seperti daftar angka atau *string*.

3. *Class*

Kelas di Java adalah *blueprint* untuk membuat objek dan dapat berisi atribut serta metode yang mendefinisikan perilaku objek tersebut.

2.4 Kelebihan dan Kekurangan Java

Java adalah bahasa pemrograman yang sangat populer dan banyak digunakan di berbagai platform. Berikut adalah beberapa kelebihan dan kekurangan dari Java yang perlu dipertimbangkan oleh para pengembang.

a. Kelebihan

Salah satu kelebihan utama dari Java adalah kemampuannya untuk dijalankan di berbagai platform, sesuai dengan prinsip "*Write Once, Run Anywhere*" (WORA), ia dapat dijalankan di berbagai sistem operasi seperti Windows, Linux, dan Mac OS tanpa perlu melakukan perubahan pada kode sumber. Fleksibilitas ini sangat menguntungkan bagi pengembang karena mereka tidak perlu menulis ulang kode untuk setiap platform yang berbeda, sehingga menghemat waktu dan usaha dalam pengembangan aplikasi. Dengan menggunakan Java *Virtual Machine* (JVM), Java dapat menerjemahkan *bytecode* yang dihasilkan ke dalam format yang dapat dipahami oleh masing-masing sistem operasi, menjadikannya pilihan yang ideal untuk aplikasi *lintas* platform.

b. Kekurangan

Di sisi lain, salah satu kekurangan Java adalah kebutuhan memori yang relatif tinggi. Meskipun Java menawarkan banyak fitur canggih dan kemudahan penggunaan, penggunaan sumber daya memori yang besar dapat menjadi masalah, terutama pada perangkat dengan spesifikasi rendah. Hal ini disebabkan oleh adanya *garbage collector* yang secara otomatis mengelola memori, serta berbagai modul dan pustaka tambahan yang digunakan dalam aplikasi. Penggunaan memori yang berlebihan dapat menyebabkan kinerja aplikasi menurun atau bahkan menyebabkan perangkat kehabisan RAM saat menjalankan aplikasi berbasis Java.

BAB III

PERMASALAHAN

3.1 Permasalahan

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 Implementasi

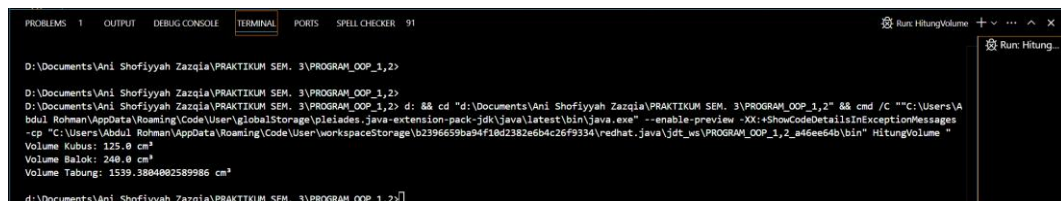
a. Berikut adalah program java menghitung bangun ruang secara langsung.



```
1 public class HitungVolume {
2
3     public static void main(String[] args) {
4         double sisiKubus = 5;
5         double panjangBalok = 10;
6         double lebarBalok = 5;
7         double tinggiBalok = 4;
8         double jariJariTabung = 7;
9         double tinggiTabung = 10;
10        double volumeKubus = hitungVolumeKubus(sisiKubus);
11        double volumeBalok = hitungVolumeBalok(panjangBalok, lebarBalok, tinggiBalok);
12
13        System.out.println("Volume Kubus: " + volumeKubus + " cm³");
14        System.out.println("Volume Balok: " + volumeBalok + " cm³");
15        System.out.println("Volume Tabung: " + (Math.PI * Math.pow(jariJariTabung, 2) * tinggiTabung) + " cm³");
16    }
17
18    public static double hitungVolumeKubus(double sisi) {
19        return Math.pow(sisi, 3);
20    }
21
22    public static double hitungVolumeBalok(double panjang, double lebar, double tinggi) {
23        return panjang * lebar * tinggi;
24    }
25
26    public static double hitungVolumeTabung(double jariJari, double tinggi) {
27        return Math.PI * Math.pow(jariJari, 2) * tinggi;
28    }
29 }
```

Gambar 1. 1 Kode Program

Gambar di atas merupakan program hitung *volume* menghitung dan menampilkan *volume* kubus, balok, dan tabung menggunakan rumus masing-masing. Nilai parameter diberikan langsung di metode *main*, dan perhitungan dilakukan melalui metode khusus atau langsung menggunakan rumus matematika.



```
D:\Documents\Ani Shofiyyah Zazqia\PRAKTIKUM SEM. 3\PROGRAM_OOP_1,2>
D:\Documents\Ani Shofiyyah Zazqia\PRAKTIKUM SEM. 3\PROGRAM_OOP_1,2>
D:\Documents\Ani Shofiyyah Zazqia\PRAKTIKUM SEM. 3\PROGRAM_OOP_1,2> d: && cd "d:\Documents\Ani Shofiyyah Zazqia\PRAKTIKUM SEM. 3\PROGRAM_OOP_1,2" && cmd /c ""C:\Users\Abdul Rohman\AppData\Roaming\Code\User\globalStorage\oleiades.java-extension-pack-jdk\java\latest\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Abdul Rohman\AppData\Roaming\Code\User\workspaceStorage\b2396659ba94f18d2382e6b4c26f9334\redhat.java\jdk_wa\PROGRAM_OOP_1,2_e46ee64b\bin" HitungVolume
Volume Kubus: 125.0 cm³
Volume Balok: 240.0 cm³
Volume Tabung: 1539.3804002589986 cm³
d:\Documents\Ani Shofiyyah Zazqia\PRAKTIKUM SEM. 3\PROGRAM_OOP_1,2>
```

Gambar 1. 2 Output Program

Gambar di atas adalah *Output* program menampilkan *volume* masing-masing bangun ruang dalam cm^3 , yaitu: *volume* kubus, *volume* balok, dan *volume* tabung, berdasarkan nilai parameter yang telah diberikan.

BAB V

PENUTUP

5.1 Kesimpulan

Java adalah bahasa pemrograman yang berorientasi objek dan multiplatform, memungkinkan pengembang untuk menulis kode yang dapat dijalankan di berbagai sistem operasi tanpa modifikasi. Dengan struktur dasar yang terdiri dari kelas dan metode, serta dukungan terhadap berbagai tipe data, Java menawarkan fleksibilitas dan kemudahan dalam pengembangan aplikasi.

Kelebihan utama Java termasuk prinsip "*Write Once, Run Anywhere*" yang mendukung aplikasi lintas platform, serta ketersediaan pustaka yang lengkap yang mempercepat proses pengembangan. Namun, Java juga memiliki kekurangan, seperti kebutuhan memori yang tinggi dan risiko dekompilasi yang dapat mengancam keamanan kode. Secara keseluruhan, Java adalah pilihan yang kuat untuk pengembangan perangkat lunak modern, tetapi pengembang harus mempertimbangkan kelebihan dan kekurangan ini dalam konteks proyek mereka untuk memaksimalkan efisiensi dan keamanan aplikasi yang dikembangkan.

5.2 Saran

Untuk terus memperdalam pemahaman tentang konsep-konsep dasar dan lanjutan dalam Java, termasuk pemrograman berorientasi objek, manajemen memori, serta penggunaan pustaka dan framework yang tersedia. Selain itu, penting untuk selalu mengikuti perkembangan terbaru dalam ekosistem Java, seperti pembaruan versi dan alat bantu pengembangan, agar dapat memanfaatkan fitur-fitur baru yang dapat meningkatkan efisiensi dan keamanan aplikasi. Pengembang juga disarankan untuk menerapkan praktik terbaik dalam pengkodean, seperti pengujian unit dan dokumentasi yang baik, serta mempertimbangkan penggunaan teknik pengamanan untuk melindungi kode dari dekompilasi. Dengan pendekatan yang tepat dan pemahaman yang mendalam, Java dapat menjadi alat yang sangat efektif dalam menciptakan solusi perangkat lunak yang inovatif dan berkualitas tinggi.



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

MODUL 2

USER INPUT



Java™

ZAIFUR ROHMAN

2302310175

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam era digital saat ini, interaksi antara pengguna dan sistem komputer menjadi semakin penting, terutama dalam pengembangan aplikasi perangkat lunak. Salah satu aspek fundamental dari interaksi ini adalah *user input*, yaitu data atau informasi yang dimasukkan oleh pengguna ke dalam program untuk diproses lebih lanjut. *User input* memungkinkan aplikasi untuk berfungsi secara dinamis dan responsif terhadap kebutuhan pengguna, menjadikannya elemen kunci dalam menciptakan pengalaman pengguna yang baik.

Dalam pemrograman, pengambilan *input* dapat dilakukan melalui berbagai metode, seperti menggunakan *class Scanner*, *BufferedReader*, atau *console* dalam bahasa pemrograman Java. Masing-masing metode memiliki keunggulan dan kekurangan yang berbeda, tergantung pada konteks penggunaan dan jenis data yang diperlukan. Selain itu, pentingnya validasi *input* tidak dapat diabaikan, karena hal ini memastikan bahwa data yang diterima sesuai dengan format yang diharapkan dan dapat diproses tanpa menghasilkan kesalahan. Dengan memahami konsep *user input* secara mendalam, para pengembang dapat menciptakan aplikasi yang lebih interaktif, efisien, dan aman, sehingga meningkatkan kepuasan pengguna dan efektivitas sistem secara keseluruhan.

1.2 Tujuan Praktikum

- 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*

User input adalah data atau informasi yang dimasukkan oleh pengguna ke dalam sistem komputer untuk diproses lebih lanjut. Dalam konteks pemrograman, *user input* memungkinkan interaksi antara pengguna dan program, sehingga program dapat berfungsi secara dinamis berdasarkan masukan yang diterima. *Input* ini dapat berupa teks, angka, atau data lainnya yang diperlukan untuk menjalankan fungsi tertentu dalam program.

Pada umumnya, *user input* diperoleh melalui perangkat keras seperti *keyboard*, mouse, atau alat *input* lainnya. Dalam pemrograman, *input* sering kali diambil menggunakan fungsi atau metode tertentu yang dirancang untuk membaca data dari pengguna. Misalnya, dalam bahasa pemrograman Java, fungsi *input()* digunakan untuk meminta pengguna memasukkan data melalui *keyboard*. Data yang dimasukkan akan disimpan dalam variabel dan dapat digunakan dalam operasi lebih lanjut. Penting untuk dicatat bahwa data yang diterima melalui fungsi ini biasanya berupa *string*, sehingga jika diperlukan tipe data lain (seperti *integer*), *programmer* harus melakukan konversi secara eksplisit.

Validasi *input* juga merupakan aspek penting dari pengolahan *user input*. Hal ini dilakukan untuk memastikan bahwa data yang dimasukkan sesuai dengan format yang diharapkan dan dapat diproses tanpa menghasilkan error. Misalnya, jika program meminta angka dan pengguna memasukkan teks, maka perlu ada penanganan kesalahan agar program tidak mengalami crash. Dengan demikian, *user input* memainkan peranan krusial dalam membuat aplikasi yang interaktif dan responsif terhadap kebutuhan pengguna.

User input berperan krusial dalam menciptakan pengalaman pengguna yang baik. Dengan memberikan cara bagi pengguna untuk berinteraksi dengan aplikasi baik melalui formulir, dialog, atau antarmuka grafis pengembang dapat mengumpulkan informasi penting yang dibutuhkan untuk memproses permintaan atau menyelesaikan tugas tertentu.

2.2 Metode Pengambilan *User Input*

User input dalam pemrograman Java dapat dilakukan melalui beberapa metode, yang masing-masing memiliki cara penggunaan dan karakteristik tersendiri. Berikut adalah penjelasan rinci mengenai beberapa metode utama untuk mengambil *input* dari pengguna.

a. *Class Scanner*

Class Scanner adalah salah satu cara paling umum dan mudah digunakan untuk mengambil *input* dari pengguna di Java. *Class* ini berada dalam package `java.util`, dan untuk menggunakannya, *programmer* perlu mengimpornya terlebih dahulu. Setelah itu, objek *Scanner* harus dibuat dengan mengaitkannya pada `System.in`, yang memungkinkan program untuk membaca *input* dari *keyboard*.

Dengan menggunakan berbagai metode yang disediakan oleh *class* ini, seperti `nextLine()`, `nextInt()`, dan `nextDouble()`, *programmer* dapat membaca berbagai jenis data. Misalnya, `nextLine()` digunakan untuk membaca *input* berupa *string*, sementara `nextInt()` digunakan untuk membaca *input* berupa *integer*. Namun, penting untuk diperhatikan bahwa jika pengguna memasukkan tipe data yang tidak sesuai (misalnya, teks saat diminta angka), program akan menghasilkan *InputMismatchException*, sehingga penanganan kesalahan perlu diterapkan untuk meningkatkan pengalaman pengguna.

b. *Class BufferedReader*

Alternatif lain untuk mengambil *input* di Java adalah menggunakan *class* *BufferedReader*. *Class* ini lebih efisien dalam membaca data dalam jumlah besar dan sering digunakan ketika kecepatan pembacaan menjadi prioritas. Untuk menggunakan *BufferedReader*, *programmer* harus mengimpor *class* tersebut dan membuat objek dengan mengaitkannya pada `InputStreamReader`.

Metode utama yang digunakan adalah `readLine()`, yang membaca seluruh baris *input* sebagai *string*. Meskipun lebih cepat dibandingkan dengan *Scanner*, penggunaan *BufferedReader* memerlukan penanganan *exception* (`IOException`) dan konversi tipe data secara manual, misalnya menggunakan `Integer.parseInt()` untuk mengubah *string* menjadi *integer*. Ini bisa membuat kode menjadi sedikit lebih rumit, tetapi sangat berguna dalam aplikasi yang memerlukan pembacaan data besar dengan lebih cepat.

c. *Class Console*

Class Console juga dapat digunakan untuk mengambil *input* dari pengguna, tetapi memiliki keterbatasan tertentu. *Class* ini hanya tersedia saat program dijalankan di terminal atau command line dan tidak berfungsi di beberapa ide seperti *Eclipse* atau *NetBeans*. Dengan menggunakan objek dari *class* ini, *programmer* dapat menggunakan metode seperti *readLine()* untuk mengambil *input string* dan *readPassword()* untuk mengambil *password* tanpa menampilkan karakter yang diketikkan di layar. Keuntungan utama dari *class* ini adalah kemampuannya untuk menangani *input* yang bersifat sensitif dengan tingkat keamanan yang lebih tinggi dibandingkan *class* lain yang serupa. Hal ini membuatnya menjadi pilihan yang sangat baik untuk situasi di mana data yang diproses memerlukan perlindungan ekstra terhadap potensi risiko atau pelanggaran keamanan.

d. *Class DataInputStream*

Class DataInputStream juga dapat digunakan untuk mengambil *input* dari pengguna, meskipun kurang umum dibandingkan dengan metode lainnya. *Class* ini memungkinkan pembacaan tipe data primitif dari aliran *input* (*input stream*). Untuk menggunakan *DataInputStream*, *programmer* perlu membuat objeknya dengan mengaitkannya pada *System.in*. Metode seperti *readInt()*, *readDouble()*, dan lain-lain dapat digunakan untuk membaca tipe data tertentu. Meskipun memberikan fleksibilitas dalam membaca berbagai tipe data, penggunaan *class* ini biasanya lebih kompleks dan kurang intuitif dibandingkan dengan *class Scanner* atau *BufferedReader*.

2.3 Jenis-Jenis *Input* Pengguna

User input dapat dibedakan berdasarkan tipe data yang dimasukkan. Berikut adalah beberapa jenis *input* pengguna yang umum dalam pemrograman:

a. *Input Teks (String)*

Input teks adalah jenis *input* yang paling umum, di mana pengguna memasukkan data berupa karakter atau *string*. Ini bisa mencakup nama, alamat, atau informasi lainnya yang tidak memerlukan format khusus. Dalam pemrograman Java, *input* ini biasanya diambil menggunakan metode seperti *nextLine()* dari *class*

Scanner. Misalnya, pengguna dapat diminta untuk memasukkan nama mereka, dan data tersebut akan disimpan sebagai *string*.

b. *Input Angka (Integer, Float, Double)*

Input angka mencakup berbagai tipe data numerik, seperti *integer* (bilangan bulat), *float* (bilangan pecahan dengan presisi tunggal), dan *double* (bilangan pecahan dengan presisi ganda). Pengguna dapat diminta untuk memasukkan nilai numerik untuk keperluan perhitungan atau analisis. Dalam Java, metode seperti *nextInt()*, *nextFloat()*, dan *nextDouble()* digunakan untuk membaca *input* angka dari pengguna. Contohnya, pengguna mungkin diminta untuk memasukkan umur atau gaji mereka.

c. *Input Boolean*

Input boolean adalah jenis *input* yang hanya memiliki dua kemungkinan nilai: *true* atau *false*. Ini sering digunakan dalam konteks pertanyaan ya/tidak atau pilihan biner lainnya. Misalnya, pengguna dapat diminta untuk menjawab apakah mereka memiliki kendaraan atau tidak. Dalam Java, *input* ini dapat diambil menggunakan metode *nextBoolean()* dari *class Scanner*.

d. *Input Karakter (Character)*

Input karakter adalah jenis *input* yang memungkinkan pengguna untuk memasukkan satu karakter tunggal. Meskipun jarang digunakan secara langsung dalam banyak aplikasi, *input* karakter bisa berguna dalam situasi tertentu, seperti saat meminta pengguna untuk memilih opsi dari daftar terbatas (misalnya, 'Y' untuk ya dan 'N' untuk tidak). Dalam Java, *input* karakter sering kali dibaca sebagai *string* dan kemudian dikonversi menjadi karakter.

e. *Input Pilihan (Choice Input)*

Input pilihan melibatkan situasi di mana pengguna diminta untuk memilih dari beberapa opsi yang telah ditentukan sebelumnya. Ini sering kali dilakukan melalui menu atau *prompt* yang menampilkan beberapa pilihan kepada pengguna. Misalnya, dalam aplikasi berbasis teks, pengguna mungkin diminta untuk memilih antara beberapa tindakan dengan memasukkan nomor terkait pilihan tersebut.

f. *Input File*

Input file memungkinkan pengguna untuk mengunggah atau memilih *file* dari sistem mereka untuk diproses oleh program. Ini sering digunakan dalam aplikasi

yang memerlukan pemrosesan data eksternal, seperti pengunggahan gambar, dokumen teks, atau *spreadsheet*. Dalam Java, pengelolaan *input file* biasanya dilakukan menggunakan *class File*, *FileReader*, atau *BufferedReader*.

g. *Input Data Terstruktur*

Input data terstruktur mencakup masukan yang mengikuti format tertentu, seperti JSON atau XML. Ini sering digunakan dalam aplikasi web dan layanan API di mana data dikirim dalam format terstruktur agar mudah diproses oleh program. Dalam Java, terdapat *library* seperti Gson atau Jackson yang membantu dalam menguraikan dan memproses *input data terstruktur* ini.

2.4 Kelebihan Dan Kekurangan Dari User Input

Input pengguna dalam pemrograman memiliki variasi yang luas, masing-masing dengan kelebihan dan kekurangan yang unik. *Input* teks (*string*), misalnya, sangat fleksibel dalam memasukkan informasi apa pun, seperti nama atau alamat, tetapi rentan terhadap kesalahan ketik yang dapat mempengaruhi proses selanjutnya. Sementara itu, *input* angka (*integer*, *float*, *double*) berguna untuk perhitungan matematika dan analisis data, tetapi error dapat timbul jika pengguna memasukkan data yang tidak valid, seperti huruf saat diminta angka. *Boolean input* mempermudah membuat keputusan logis dengan hanya dua kemungkinan nilai (*true* atau *false*), tetapi terbatas dalam informasi yang dapat disampaikan, sehingga tidak cocok untuk situasi yang memerlukan lebih banyak detail.

Choice input memberikan opsi terstruktur bagi pengguna untuk memilih dari beberapa pilihan, tetapi jika pilihan tidak cukup jelas atau terbatas, ini bisa membingungkan dan mengurangi pengalaman pengguna. *File input* memungkinkan pengunggahan data eksternal yang berguna dalam aplikasi seperti pengolahan dokumen atau gambar, tetapi juga menambah kompleksitas dalam penanganan dan validasi *file* tersebut. Terakhir, data terstruktur seperti JSON atau XML memfasilitasi pertukaran data yang terorganisir antara sistem, tetapi memerlukan pemahaman format tertentu dan proses parsing yang tepat agar dapat digunakan dengan efektif. Oleh karena itu, setiap jenis *input* harus dipilih berdasarkan kebutuhan aplikasi untuk menciptakan interaksi yang efektif dan responsif, serta meningkatkan pengalaman pengguna secara keseluruhan.

BAB III

PERMASALAHAN

3.1 Permasalahan

a. Buatlah *Inputan* biodata mahasiswa dengan ketentuan nama, nim, alamat, jeniskelamin (imputannya hari 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 adalah program Java membuat *inputan* biodata mahasiswa

```
1 import java.util.Scanner;
2 public class BiodataMahasiswa {
3     public static void main(String[] args) {
4         Scanner input = new Scanner(System.in);
5         System.out.print("$:Masukkan Nama: ");
6         String nama = input.nextLine();
7         System.out.print("$:Masukkan NIM: ");
8         String nim = input.nextLine();
9         System.out.print("$:Masukkan Alamat: ");
10        String alamat = input.nextLine();
11        System.out.print("$:Masukkan Jenis Kelamin (L/P): ");
12        char jkInput = input.next().charAt(0);
13        String jenisKelamin;
14        if (jkInput == 'L' || jkInput == 'l') {
15            jenisKelamin = "Laki-laki";
16        } else if (jkInput == 'P' || jkInput == 'p') {
17            jenisKelamin = "Perempuan";
18        } else {
19            jenisKelamin = "Tidak diketahui";
20        }
21        System.out.println("\n--- Biodata Mahasiswa ---");
22        System.out.println("Saya " + nama + " dengan NIM " + nim + " yang tinggal di " + alamat + ",");
23        System.out.println("Saya seorang " + jenisKelamin + ".");
24    }
25 }
```

Gambar 2. 1 Program Biodata Mahasiswa

Program Java di atas adalah aplikasi sederhana untuk mengumpulkan biodata mahasiswa. Menggunakan *class Scanner*, program meminta pengguna untuk memasukkan nama, NIM, alamat, dan jenis kelamin (L/P). *Input* jenis kelamin diproses untuk menentukan deskripsi yang sesuai. Setelah semua data terkumpul, program menampilkan biodata dalam format terstruktur, mencakup nama, NIM, alamat, dan jenis kelamin. Jika *input* jenis kelamin tidak valid, program akan mencetak "Tidak diketahui".

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 91
cuments\Veri Shofiyah Zazqia\PRAKTIKUM SEM. 3\PROGRAM 0002 cmd /C "C:\Users\Abdul Rohman\AppData\Roaming\Code\User\globalStorage\pleiades.java-extension-pack-jdk\java\latest\bin\java.exe" --enable-preview -XX:
ShowCodeDetailsInExceptionHandlerMessages -cp "C:\Users\Abdul Rohman\AppData\Roaming\Code\User\workspaceStorage\59b79f8ba5daf38c635f945f0c8ef05\redhat_java\jdk_ws\PROGRAM_000_3ebcb13\bin" BiodataMahasiswa
Masukkan Nama: Ani Shofiyah Zazqia
Masukkan NIM: 2302310184
Masukkan Alamat: Pangluros Kelbung Sepulu Bangkalan
Masukkan Jenis Kelamin (L/P): P

--- Biodata Mahasiswa ---
Saya Ani Shofiyah Zazqia dengan NIM 2302310184 yang tinggal di Pangluros Kelbung Sepulu Bangkalan,
Saya seorang Perempuan.
```

Gambar 2. 2 Output Program Biodata Diri

Gambar diatas merupakan *Output* dari program Java ini adalah tampilan biodata mahasiswa yang telah dimasukkan oleh pengguna. Setelah pengguna memberikan *input* untuk nama, NIM, alamat, dan jenis kelamin, program mencetak informasi tersebut dalam format yang terstruktur.

BAB V

PENUTUP

5.1 Kesimpulan

user input dalam pemrograman Java merupakan aspek penting yang memungkinkan interaksi dinamis antara pengguna dan aplikasi. Berbagai metode seperti *Scanner*, *BufferedReader*, dan *Console* menawarkan cara yang berbeda untuk mengambil *input*, masing-masing dengan kelebihan dan kekurangan. Jenis-jenis *input*, termasuk teks, angka, *boolean*, pilihan, *file*, dan data terstruktur, memberikan fleksibilitas dalam pengumpulan informasi. Program sederhana seperti contoh biodata mahasiswa menunjukkan bagaimana data pengguna dapat diolah dan ditampilkan secara efektif. Dengan memahami cara mengelola *input* pengguna dan *Output* yang dihasilkan, pengembang dapat merancang aplikasi yang lebih responsif dan *user-friendly*, meningkatkan pengalaman pengguna secara keseluruhan.

5.2 Saran

Saran untuk pengembang yang bekerja dengan *user input* dalam pemrograman Java adalah untuk selalu mempertimbangkan validasi dan penanganan kesalahan saat mengumpulkan data dari pengguna. Pastikan untuk memberikan instruksi yang jelas mengenai format *input* yang diharapkan, sehingga pengguna dapat memasukkan data dengan benar. Selain itu, gunakan metode yang sesuai berdasarkan konteks aplikasi; misalnya, *Scanner* untuk *input* sederhana dan *BufferedReader* untuk pembacaan data besar. Implementasikan logika untuk menangani *input* yang tidak valid, seperti menampilkan pesan kesalahan yang informatif. Terakhir, pertimbangkan pengalaman pengguna secara keseluruhan dengan membuat antarmuka *input* yang intuitif dan responsif, agar pengguna merasa nyaman dan puas saat berinteraksi dengan aplikasi.



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

MODUL 3

KONDISIONAL PERULANGAN DAN PERCABANGAN



Java™

ZAIFUR ROHMAN

2302310175

BAB I

PENDAHULUAN

1.1 Latar Belakang

Kondisional, perulangan, dan percabangan adalah tiga konsep fundamental dalam pemrograman Java yang sangat penting untuk mengontrol alur eksekusi program. Kondisional memungkinkan program untuk mengambil keputusan berdasarkan kondisi tertentu, menggunakan struktur seperti *if*, *else*, dan *switch*. Dengan kondisional, *programmer* dapat menentukan jalur mana yang akan diambil dalam situasi yang berbeda, sehingga meningkatkan fleksibilitas dan responsivitas aplikasi. Perulangan, di sisi lain, digunakan untuk mengeksekusi blok kode secara berulang hingga kondisi tertentu terpenuhi. Struktur perulangan seperti *for*, *while*, dan *do-while* memungkinkan pengulangan yang efisien, mengurangi redundansi dalam kode dan memudahkan pengelolaan data dalam jumlah besar. Percabangan mengacu pada kemampuan untuk membagi alur eksekusi program ke dalam beberapa cabang berdasarkan hasil evaluasi kondisi. Dengan menggunakan percabangan, *programmer* dapat menangani berbagai kemungkinan hasil dan meresponsnya dengan cara yang sesuai. Ketiga konsep ini saling terkait dan membentuk dasar logika pemrograman yang kuat, memungkinkan pengembangan aplikasi yang kompleks dan interaktif dengan lebih mudah.

1.2 Tujuan Praktikum

- a. Mahasiswa dapat memahami dan menggunakan kondisional, perulangan dan percabangan.

BAB II

DASAR TEORI

2.1 Kondisional

Kondisional adalah struktur yang memungkinkan program untuk membuat keputusan berdasarkan kondisi tertentu. Dengan menggunakan pernyataan kondisional, *programmer* dapat mengarahkan alur eksekusi program ke jalur yang berbeda tergantung pada hasil evaluasi dari kondisi yang diberikan.

a. Pengertian Kondisional

Kondisional dalam Java merujuk pada struktur yang memungkinkan program untuk mengambil keputusan berdasarkan evaluasi kondisi tertentu. Dalam konteks pemrograman, kondisional berfungsi sebagai alat untuk mengontrol alur eksekusi program, sehingga *programmer* dapat menentukan tindakan yang akan diambil berdasarkan hasil dari suatu kondisi.

Jika kondisi yang dievaluasi bernilai benar (*true*), maka instruksi tertentu akan dijalankan, sebaliknya, jika kondisi tersebut bernilai salah (*false*), program dapat memilih untuk menjalankan instruksi lain atau tidak melakukan apa-apa sama sekali. Dengan demikian, kondisional sangat penting dalam menciptakan logika yang dinamis dan interaktif dalam aplikasi.

b. Jenis Pernyataan Kondisional

Ada beberapa jenis pernyataan kondisional dalam Java, di antaranya adalah *if*, *if-else*, *nested if*, dan *switch-case*.

1. *If Statement*

Pernyataan *if* adalah bentuk paling dasar dari kondisional. Ini digunakan untuk mengevaluasi satu kondisi; jika kondisi tersebut benar (*true*), maka blok kode di dalamnya akan dieksekusi. Contohnya, jika kita ingin memeriksa apakah seorang siswa lulus berdasarkan nilai yang diperoleh, kita bisa menggunakan pernyataan *if* untuk mengeksekusi instruksi tertentu jika nilai tersebut memenuhi syarat kelulusan, seperti lebih besar atau sama dengan 75. Jika tidak ada instruksi yang perlu dijalankan ketika kondisi tidak terpenuhi, maka tidak ada tindakan lebih lanjut yang diambil.

2. *If-Else Statement*

Pernyataan *if-else* memperluas fungsionalitas dari pernyataan *if* dengan menambahkan alternatif. Jika kondisi dalam pernyataan *if* tidak terpenuhi, maka blok kode dalam bagian *else* akan dieksekusi. Ini berguna ketika kita ingin memberikan dua kemungkinan hasil berdasarkan evaluasi kondisi. Misalnya, jika nilai siswa lebih besar atau sama dengan 75, dia akan dinyatakan lulus; jika tidak, dia dinyatakan tidak lulus. Dengan cara ini, *programmer* dapat menangani dua skenario yang berbeda secara langsung.

3. *Nested If*

Pernyataan *nested if* memungkinkan penggunaan pernyataan *if* di dalam pernyataan *if* lainnya. Ini berguna ketika ada beberapa kondisi yang perlu dievaluasi secara berurutan. Misalnya, kita bisa memeriksa apakah seorang siswa lulus dan kemudian mengevaluasi apakah dia mendapatkan beasiswa berdasarkan kriteria tambahan seperti penghasilan orang tua. Dengan menggunakan *nested if*, kita dapat menambahkan logika lebih kompleks dan membuat keputusan berdasarkan beberapa parameter sekaligus.

4. *Switch-Case*

Pernyataan *switch-case* adalah alternatif dari serangkaian pernyataan *if-else* yang lebih sederhana ketika ada banyak kemungkinan nilai untuk satu variabel. Dengan *switch-case*, dapat membandingkan nilai variabel dengan beberapa kasus (*case*) dan mengeksekusi blok kode yang sesuai dengan nilai tersebut. Misalnya, dalam permainan video, kita bisa menggunakan *switch-case* untuk menentukan arah gerakan karakter berdasarkan *input* pengguna. Setiap *case* harus diakhiri dengan pernyataan *break* untuk mencegah eksekusi berlanjut ke *case* berikutnya.

c. Manfaat Pernyataan Kondisional

Kondisional dalam pemrograman Java memiliki manfaat yang sangat penting dalam mengontrol alur eksekusi program berdasarkan evaluasi kondisi tertentu. Dengan menggunakan struktur kondisional seperti *if*, *else if*, dan *switch-case*, *programmer* dapat membuat keputusan yang dinamis, memungkinkan aplikasi untuk merespons berbagai situasi dan *input* dari pengguna dengan cara yang tepat. Salah satu manfaat utama dari kondisional adalah kemampuannya untuk meningkatkan interaktivitas program.

2.2 Perulangan

Perulangan dalam Java adalah mekanisme yang memungkinkan eksekusi berulang dari suatu blok kode berdasarkan kondisi tertentu. Ini sangat penting dalam pemrograman karena memungkinkan pengulangan instruksi tanpa harus menuliskannya berulang kali, sehingga dapat meningkatkan efisiensi dan keterbacaan kode.

a. Pengertian Perulangan

Perulangan dalam Java, atau yang sering disebut sebagai "*looping*," adalah konsep dasar dalam pemrograman yang memungkinkan eksekusi satu atau beberapa baris kode secara berulang berdasarkan kondisi tertentu. Dengan menggunakan perulangan, *programmer* dapat menghindari penulisan kode yang sama berulang kali, sehingga dapat meningkatkan efisiensi dan keterbacaan program. Perulangan sangat berguna dalam berbagai situasi, seperti saat mengolah data dalam *array*, melakukan perhitungan berulang, atau memproses *input* yang dikirim oleh pengguna.

b. Jenis-jenis Perulangan

Dalam Java, terdapat tiga jenis perulangan utama: *for*, *while*, dan *do-while*. Masing-masing jenis perulangan memiliki sintaksis dan penggunaan yang berbeda.

1. *For*

Perulangan *For* adalah jenis perulangan yang paling umum digunakan ketika jumlah iterasi sudah diketahui sebelumnya. Dalam perulangan ini, kita mendefinisikan tiga bagian penting: inisialisasi variabel, kondisi yang harus dipenuhi untuk melanjutkan perulangan, dan perubahan yang dilakukan pada variabel setelah setiap iterasi. Hal ini membuat perulangan *for* sangat efisien untuk situasi di mana kita ingin mengulang suatu proses dengan jumlah langkah pasti, seperti mengiterasi elemen dalam *array* atau melakukan penghitungan tertentu.

2. *While*

Perulangan digunakan ketika jumlah iterasi tidak diketahui sebelumnya dan bergantung pada kondisi tertentu yang mungkin berubah selama eksekusi. Pada perulangan ini, kondisi diperiksa sebelum setiap iterasi; jika kondisi bernilai *true*, maka blok kode di dalamnya akan dieksekusi. Ini menjadikan *while* sangat berguna dalam situasi di mana kita perlu terus melakukan suatu proses hingga kondisi

tertentu tidak lagi terpenuhi, seperti saat membaca data dari pengguna sampai mereka memilih untuk berhenti.

3. *Do-While*

Perulangan *Do-While* mirip dengan *while*, tetapi dengan perbedaan utama bahwa kondisi diperiksa setelah eksekusi blok kode. Ini memastikan bahwa blok kode akan dieksekusi setidaknya satu kali sebelum evaluasi kondisi dilakukan. Perulangan *do-while* sangat berguna ketika kita ingin menjamin bahwa suatu aksi dilakukan minimal satu kali, seperti meminta *input* dari pengguna setidaknya sekali sebelum memeriksa apakah mereka ingin melanjutkan atau tidak.

c. Manfaat Perulangan

Perulangan dalam pemrograman Java memiliki berbagai manfaat yang signifikan, menjadikannya salah satu konsep dasar yang sangat penting.

1. Menghindari Pengulangan Kode

Perulangan memungkinkan eksekusi berulang dari blok kode tanpa perlu menulis instruksi yang sama berulang kali, sehingga menghemat waktu dan usaha.

2. Meningkatkan Keterbacaan Kode

Dengan menggunakan perulangan, kode menjadi lebih ringkas dan mudah dibaca, karena tidak ada pengulangan instruksi yang tidak perlu.

3. Memudahkan Pengolahan Data

Perulangan sangat berguna untuk mengiterasi melalui koleksi data seperti *array* atau daftar, memungkinkan pemrosesan elemen jumlah besar secara efisien.

4. Pelaksanaan Perhitungan Berulang

Perulangan memungkinkan pelaksanaan operasi matematis yang diperlukan dalam algoritma tertentu, seperti penghitungan total dari jumlah angka atau rata-rata dari sekumpulan angka.

5. Validasi *Input* Pengguna

Dengan perulangan, program dapat terus meminta *input* dari pengguna hingga nilai yang valid diberikan, meningkatkan interaktivitas dan keandalan aplikasi.

6. Penerapan Logika yang Kompleks

Perulangan membantu dalam menangani logika program yang lebih kompleks dengan cara yang terstruktur, memungkinkan *programmer* untuk mengeksekusi blok kode berdasarkan kondisi tertentu.

7. Penggunaan Sumber Daya yang Efisien

Dengan mengurangi jumlah baris kode dan meningkatkan efisiensi eksekusi, perulangan membantu penggunaan sumber daya sistem secara lebih optimal.

8. Fleksibilitas dalam Algoritma

Perulangan memberikan fleksibilitas dalam merancang algoritma, memungkinkan *programmer* untuk mengubah kondisi dan cara iterasi sesuai kebutuhan aplikasi.

2.3 Percabangan

Percabangan dalam Java adalah struktur yang memungkinkan program untuk mengambil keputusan berdasarkan kondisi tertentu, sehingga alur eksekusi program dapat bercabang. Pernyataan ini dikenal juga sebagai "*Control Flow*" atau "Struktur Kondisi". Dalam Java, terdapat beberapa jenis percabangan, termasuk *if*, *if-else*, *else if*, dan *switch-case*.

Pernyataan *if* digunakan untuk mengevaluasi suatu kondisi; jika kondisi tersebut benar, maka blok kode di dalamnya akan dieksekusi. Jika tidak, *programmer* dapat menggunakan *else* untuk menentukan tindakan alternatif. Dengan *else if*, beberapa kondisi dapat diperiksa secara berurutan, memungkinkan penanganan lebih dari dua kemungkinan hasil. Di sisi lain, pernyataan *switch-case* menyediakan cara yang lebih terstruktur untuk menangani banyak kemungkinan nilai dari suatu variabel, dengan membandingkan nilai tersebut terhadap beberapa kasus yang telah ditentukan. Selain itu, percabangan juga membantu dalam menangani logika kompleks dengan cara yang terorganisir, memungkinkan *programmer* untuk mengelompokkan berbagai kondisi dan tindakan yang relevan.

Meskipun percabangan dan kondisional sering digunakan secara bergantian, keduanya memiliki perbedaan. Kondisional merujuk pada mekanisme yang digunakan untuk mengevaluasi kondisi dan menentukan apakah suatu blok kode harus dieksekusi. Ini mencakup semua jenis pernyataan yang memungkinkan pengambilan keputusan berdasarkan nilai boolean (*true* atau *false*). Sementara itu, percabangan adalah istilah yang lebih luas yang mencakup penggunaan struktur kondisional untuk mengarahkan alur eksekusi program. Dengan kata lain, semua percabangan melibatkan kondisional, tetapi percabangan lebih fokus pada pengaturan alur eksekusi berdasarkan hasil evaluasi dari satu atau beberapa kondisi.

BAB III

PERMASALAHAN

3.1 Permasalahan

- a. Menentukan bilangan ganjil dan genap mulai 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 adalah program menentukan bilangan ganjil dan genap mulai dari 1 sampai 100.

```
1 public class GanjilGenap {
2     public static void main(String[] args) {
3         System.out.println("Bilangan Genap dari 1 sampai 100:");
4         for (int i = 1; i <= 100; i++) {
5             if (i % 2 == 0) {
6                 System.out.print(i + " ");
7             }
8         }
9
10        System.out.println("\n\nBilangan Ganjil dari 1 sampai 100:");
11        for (int i = 1; i <= 100; i++) {
12            if (i % 2 != 0) {
13                System.out.print(i + " ");
14            }
15        }
16    }
17 }
```

Gambar 3. 1 Program Bilangan Ganjil genap

Program di atas adalah program Java sederhana untuk menentukan dan mencetak bilangan genap dan ganjil dari 1 sampai 100. Program ini menggunakan *loop for* untuk melakukan iterasi dari angka 1 hingga 100. Dalam *loop* pertama, bilangan genap ditentukan dengan memeriksa apakah angka habis dibagi 2 menggunakan kondisi $i \% 2 == 0$. Jika kondisi ini terpenuhi, angka tersebut dicetak di baris yang sama menggunakan `System.out.print()`. Setelah itu, program mencetak bilangan ganjil dengan menggunakan *loop* kedua, yang memeriksa apakah angka tidak habis dibagi 2 dengan kondisi $i \% 2 != 0$. Bilangan ganjil yang memenuhi kondisi ini juga dicetak di baris yang sama.

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 68
D:\Documents\Ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP> cmd /C ""C:\Users\Abdul Rohman\AppData\Roaming\Code\User\globalSt
orage\pleiades.java-extension-pack-jdk\java\latest\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Use
rs\Abdul Rohman\AppData\Roaming\Code\User\workspaceStorage\5a6e39fde3ff4d603c425a397c5a7517\redhat.java\jdt_ws\PROGRAM_OOP_dbfe72f4\b
in" GanjilGenap
Bilangan Genap dari 1 sampai 100:
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92
94 96 98 100
Bilangan Ganjil dari 1 sampai 100:
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91
93 95 97 99
D:\Documents\Ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP>
```

Gambar 3. 2 Output Program

Output dari program ini terdiri dari dua bagian. Bagian pertama mencetak daftar bilangan genap dari 1 hingga 100, misalnya 2 4 6 8 ... 100. Bagian kedua mencetak daftar bilangan ganjil dari 1 hingga 100, misalnya 1 3 5 7 ... 99. Hasilnya

kedua bagian dipisahkan oleh dua baris kosong untuk memudahkan pembacaan.

Program penilaian sesuai *grade* yang ditentukan

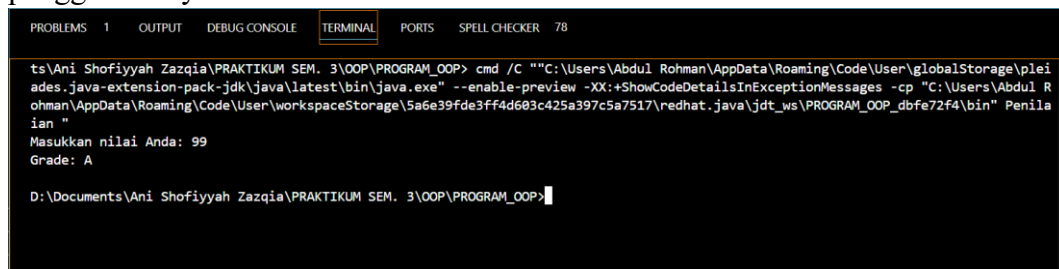
```

1  import java.util.Scanner;
2
3  public class Penilaian {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6
7          System.out.print("Masukkan nilai Anda: ");
8          int nilai = scanner.nextInt();
9
10         if (nilai > 80) {
11             System.out.println("Grade: A");
12         } else if (nilai > 70) {
13             System.out.println("Grade: B");
14         } else if (nilai > 60) {
15             System.out.println("Grade: C");
16         } else if (nilai >= 30) {
17             System.out.println("Grade: D");
18         } else {
19             System.out.println("Grade: Tidak Lulus");
20         }
21
22         scanner.close();
23     }
24 }

```

Gambar 3. 3 Program Penilaian

Program ini membaca nilai pengguna menggunakan *Scanner* dan menentukan *grade* berdasarkan nilai tersebut. Jika nilai lebih dari 80, *grade* adalah A; lebih dari 70 adalah B; lebih dari 60 adalah C; 30 hingga 60 adalah D; dan di bawah 30, pengguna dinyatakan Tidak Lulus.



```

PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER  78
ts\Ani Shofiyyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP> cmd /C ""C:\Users\Abdul Rohman\AppData\Roaming\Code\User\globalStorage\plei
ades.java-extension-pack-jdk\java\latest\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Abdul R
ohman\AppData\Roaming\Code\User\workspaceStorage\5a6e39fde3ff4d683c425a397c5a7517\redhat.java\jdt_ws\PROGRAM_OOP_dbfe72f4\bin" Penila
ian "
Masukkan nilai Anda: 99
Grade: A

D:\Documents\Ani Shofiyyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP>

```

Gambar 3. 4 Output Program

Hasil *grade* ditampilkan sesuai kondisi yang terpenuhi, dan program menutup objek *Scanner* setelah *input* selesai. Contoh *Output*: jika nilai yang dimasukkan 85, program mencetak *Grade: A*.

BAB V

PENUTUP

5.1 Kesimpulan

Dalam pemrograman Java, konsep kondisional, perulangan, dan percabangan memainkan peran yang sangat penting dalam mengontrol alur eksekusi program dan meningkatkan interaktivitas aplikasi. Kondisional memungkinkan *programmer* untuk membuat keputusan berdasarkan evaluasi kondisi tertentu, menggunakan struktur seperti *if*, *else if*, dan *switch-case*, sehingga program dapat merespons *input* pengguna dengan cara yang tepat. Di sisi lain, perulangan menyediakan mekanisme untuk mengeksekusi blok kode secara berulang, yang sangat berguna dalam pengolahan data dan pelaksanaan perhitungan berulang tanpa perlu menulis kode yang sama berulang kali. Sementara itu, percabangan menggabungkan elemen kondisional untuk menangani berbagai kemungkinan hasil dengan cara yang terorganisir, memungkinkan aplikasi untuk mengambil tindakan yang sesuai berdasarkan situasi yang dihadapi. Secara keseluruhan, pemahaman dan penerapan ketiga konsep ini memungkinkan *programmer* untuk menulis kode yang lebih efisien, fleksibel, dan responsif terhadap kebutuhan pengguna, serta menghasilkan aplikasi yang lebih kompleks dan interaktif.

5.2 Saran

Penting untuk memahami dan menguasai konsep kondisional, perulangan, dan percabangan dengan baik, karena ketiga elemen ini merupakan fondasi dalam pengembangan perangkat lunak yang efektif. Praktikkan penggunaan berbagai struktur ini dalam proyek kecil untuk memperkuat pemahaman dan kemampuan problem-solving. Selain itu, selalu pertimbangkan untuk menulis kode yang bersih dan terstruktur agar lebih mudah dipahami dan dipelihara di masa depan. Terakhir, eksplorasi lebih lanjut tentang algoritma dan logika pemrograman dapat membantu meningkatkan keterampilan pemrograman secara keseluruhan.



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

MODUL 4

JAVA ARRAY



Java™

ZAIFUR ROHMAN

2302310175

BAB I

PENDAHULUAN

1.1 Latar Belakang

Array dalam Java merupakan struktur data yang sangat penting dan sering digunakan untuk menyimpan sekumpulan elemen yang memiliki tipe data yang sama. Konsep *array* memungkinkan *programmer* untuk mengelola data dengan lebih efisien, terutama ketika berhadapan dengan jumlah data yang besar. Tanpa adanya *array*, *programmer* harus mendeklarasikan banyak variabel terpisah untuk menyimpan data yang serupa, yang tentunya tidak praktis dan sulit dikelola. Dalam pemrograman Java, terdapat berbagai cara untuk melakukan iterasi melalui elemen dalam *array*, masing-masing dengan kelebihan dan kekurangan yang unik. Metode pertama adalah *loop for* tradisional, yang memberikan kontrol penuh atas proses iterasi dan memungkinkan akses langsung ke indeks.

Array di Java dianggap sebagai objek dan memiliki ukuran tetap yang ditentukan saat *array* dibuat. Setiap elemen dalam *array* dapat diakses menggunakan indeks, yang dimulai dari 0. Dengan menggunakan *array*, pengelolaan data menjadi lebih terstruktur dan mudah diakses, serta memudahkan dalam melakukan operasi seperti iterasi dan manipulasi data. Selain itu, Java juga mendukung *array* multidimensi, yang memungkinkan penyimpanan data dalam bentuk tabel atau matriks, memberikan fleksibilitas tambahan dalam pengolahan data kompleks.

1.2 Tujuan

- a. Mahasiswa dapat mendeklarasikan dan membuat *Array*

BAB II

DASAR TEORI

2.1 Pengertian *Array*

Array dalam Java adalah struktur data yang memungkinkan penyimpanan sekumpulan elemen dengan tipe data yang sama dalam satu variabel. Konsep ini sangat penting dalam pemrograman karena membantu pengelolaan data secara efisien dan terorganisir. Setiap elemen dalam *array* diakses menggunakan indeks, yang dimulai dari 0, sehingga elemen pertama berada di posisi 0, elemen kedua di posisi 1, dan seterusnya. Dengan menggunakan *array*, *programmer* dapat menyimpan dan mengelola data dalam jumlah besar tanpa perlu mendeklarasikan banyak variabel terpisah, yang tentunya lebih praktis. *Array* juga mendukung berbagai tipe data, termasuk tipe primitif seperti *int*, *float*, dan *char*, serta objek seperti *String*. Selain itu, Java mendukung *array* multidimensi, yang memungkinkan penyimpanan data dalam bentuk tabel atau matriks. Penggunaan *array* sangat krusial dalam banyak algoritma dan struktur data lainnya, menjadikannya salah satu konsep dasar yang harus dipahami oleh setiap *programmer* Java.

2.2 Deklarasi *Array*

Deklarasi *array* dalam Java adalah langkah awal untuk menggunakan struktur data ini, yang memungkinkan penyimpanan sekumpulan elemen dengan tipe data yang sama. Proses deklarasi terdiri dari beberapa langkah penting yang perlu dipahami secara rinci.

a. Menentukan Tipe Data

Langkah pertama dalam deklarasi *array* adalah menentukan tipe data elemen yang akan disimpan dalam *array*. Tipe data ini bisa berupa tipe primitif seperti *int*, *float*, *char*, atau tipe objek seperti *String*. Penentuan tipe data ini sangat penting karena akan mempengaruhi cara pengolahan dan penyimpanan data dalam memori. Misalnya, jika kita ingin menyimpan bilangan bulat, kita akan menggunakan tipe data *integer (int)*.

b. Menentukan Nama *Array*

Setelah menentukan tipe data, langkah berikutnya adalah memberikan nama pada *array*. Nama *array* harus mengikuti aturan penamaan variabel di Java, yang berarti tidak boleh diawali dengan angka, tidak boleh mengandung spasi, dan tidak boleh menggunakan karakter khusus kecuali garis bawah (_). Nama *array* sebaiknya deskriptif untuk mencerminkan isi dari *array* tersebut. Misalnya, kita bisa menggunakan nama angka untuk menyimpan bilangan bulat.

c. Mengalokasikan Memori untuk *Array*

Setelah mendeklarasikan nama dan tipe data, kita perlu mengalokasikan memori untuk *array* dengan menggunakan kata kunci *new*. Pada tahap ini, kita juga harus menentukan ukuran *array*, yaitu jumlah elemen yang dapat disimpan. Ukuran ini bersifat tetap setelah *array* dibuat.

d. Deklarasi dan Inisialisasi dalam Satu Langkah

Java juga memungkinkan kita untuk menggabungkan proses deklarasi dan inisialisasi nilai awal dalam satu langkah. Ini dilakukan dengan menuliskan nilai-nilai yang ingin dimasukkan ke dalam *array* di dalam kurung kurawal {}. Contohnya, jika kita ingin membuat *array* angka dengan nilai awal 10, 20, 30, 40, dan 50.

e. Mengakses Elemen *Array*

Setelah mendeklarasikan dan menginisialisasi *array*, kita dapat mengakses elemen-elemennya menggunakan indeks. Indeks dimulai dari 0 hingga $n-1$ (di mana n adalah jumlah elemen dalam *array*).

2.3 Jenis-jenis *Array*

Array dalam Java dapat dibedakan menjadi beberapa jenis berdasarkan dimensi dan struktur penyimpanannya. Berikut adalah penjelasan rinci mengenai jenis-jenis *array* yang umum digunakan dalam pemrograman Java.

a. *Array* Satu Dimensi

Array satu dimensi adalah jenis *array* yang paling sederhana dan paling sering digunakan. *Array* ini menyimpan elemen dalam satu baris, sehingga semua elemen dapat diakses menggunakan satu indeks. Setiap elemen dalam *array* satu dimensi diakses menggunakan satu indeks, yang memudahkan pengelolaan data sekuensial.

Array ini sangat berguna untuk menyimpan kumpulan data yang memiliki hubungan langsung, seperti daftar nilai atau nama.

b. *Array* Dua Dimensi

Array dua dimensi adalah *array* yang memiliki dua indeks, memungkinkan penyimpanan data dalam bentuk tabel atau matriks. *Array* ini memiliki dua indeks, yaitu satu untuk baris dan satu untuk kolom, sehingga setiap elemen dapat diakses dengan kombinasi kedua indeks tersebut. *Array* dua dimensi sering digunakan dalam aplikasi yang memerlukan representasi data terstruktur, seperti grafik atau tabel statistik, di mana hubungan antar data lebih kompleks. Setiap elemen dalam *array* dua dimensi diakses dengan menggunakan dua indeks, satu untuk baris dan satu untuk kolom.

c. *Array* Multidimensi

Array multidimensi adalah jenis *array* yang dapat memiliki lebih dari dua dimensi. Ini berarti kita dapat membuat *array* tiga dimensi, empat dimensi, atau lebih, tergantung pada kebutuhan aplikasi. *Array* ini memungkinkan penyimpanan data dalam format yang lebih kompleks, seperti kubus atau volume. Dengan menggunakan *array* multidimensi, *programmer* dapat mengelola data yang memiliki lebih dari dua atribut atau variabel secara bersamaan. Meskipun penggunaannya lebih jarang dibandingkan dengan *array* satu dan dua dimensi, *array* multidimensi sangat berguna dalam aplikasi tertentu yang memerlukan representasi data yang lebih kompleks.

2.4 Manfaat Menggunakan *Array*

Penggunaan *array* dalam pemrograman memiliki banyak manfaat yang signifikan, menjadikannya salah satu struktur data yang paling penting dan sering digunakan. Berikut adalah penjelasan rinci mengenai manfaat menggunakan *array* dalam pemrograman:

a. Pengelolaan Data Terstruktur

Salah satu manfaat utama dari *array* adalah kemampuannya untuk mengelola data terstruktur. Dengan *array*, *programmer* dapat menyimpan sekumpulan elemen dengan tipe data yang sama dalam satu variabel. Hal ini memudahkan pengorganisasian data, terutama ketika bekerja dengan kumpulan data yang besar.

Misalnya, jika kita perlu menyimpan daftar nilai siswa, kita bisa menggunakan *array* untuk menyimpan semua nilai tersebut dalam satu struktur. Ini mengurangi kebutuhan untuk mendeklarasikan banyak variabel terpisah, yang bisa membingungkan dan sulit dikelola. Dengan *array*, data dapat diakses dan dimanipulasi dengan cara yang lebih sistematis dan teratur.

b. Akses yang Cepat dan Efisien

Array menawarkan akses yang cepat dan efisien terhadap elemen-elemennya. Setiap elemen dalam *array* dapat diakses langsung menggunakan indeks, yang berarti bahwa waktu akses untuk mengambil elemen tertentu adalah konstan ($O(1)$). Ini sangat penting dalam aplikasi yang memerlukan kecepatan tinggi, seperti algoritma pencarian dan pengurutan. Ketika data disimpan dalam *array*, *programmer* dapat dengan mudah mengambil, memodifikasi, atau menghapus elemen tanpa harus melakukan pencarian yang rumit. Kecepatan akses ini menjadikan *array* pilihan ideal untuk aplikasi yang memerlukan performa yang lebih tinggi.

c. Penggunaan Memori yang Efisien

Array juga menawarkan penggunaan memori yang efisien. Elemen-elemen dalam *array* disimpan secara kontigu di dalam memori, yang mengurangi *overhead* dibandingkan dengan struktur data lain seperti *linked list*. Dalam *linked list*, setiap elemen (*node*) memiliki *pointer* ke elemen berikutnya, yang membutuhkan lebih banyak ruang memori untuk menyimpan *pointer* tersebut. Sebaliknya, *array* hanya membutuhkan ruang untuk elemen itu sendiri. Selain itu, penyimpanan kontigu ini memungkinkan *array* memanfaatkan *cache* memori dengan lebih baik, sehingga meningkatkan kecepatan akses data.

d. Dukungan untuk Implementasi Algoritma

Array mendukung implementasi berbagai algoritma dasar dalam pemrograman. Banyak algoritma pencarian (seperti *binary search*) dan algoritma pengurutan (seperti *quicksort* dan *mergesort*) dirancang khusus untuk bekerja dengan *array*. Dengan menggunakan *array* sebagai struktur dasar, *programmer* dapat menerapkan algoritma-algoritma ini dengan lebih mudah dan efisien. Hal ini menjadikan *array* sebagai fondasi penting dalam pengembangan algoritma kompleks serta pengolahan data secara umum.

e. Dukungan untuk *Array* Multidimensi

Array juga memiliki kemampuan untuk mendukung multidimensi, memungkinkan penyimpanan data dalam bentuk tabel atau matriks. Ini sangat berguna dalam aplikasi yang memerlukan representasi data dua dimensi atau lebih, seperti pemrosesan citra, grafik komputer, dan simulasi fisika.

2.5 Metode Iterasi *Array*

Ada beberapa cara untuk melakukan iterasi melalui elemen dalam *array* di Java untuk mengakses dan memanipulasi setiap elemen dengan mudah.

a. *Loop for* Tradisional

Loop for tradisional adalah salah satu metode paling umum dan fleksibel untuk melakukan iterasi melalui elemen dalam *array*. Dalam pendekatan ini, *programmer* mendeklarasikan variabel indeks yang digunakan untuk mengontrol jumlah iterasi. Sintaks dasar dari *loop* ini melibatkan tiga bagian: inisialisasi indeks, kondisi pengulangan, dan pernyataan *increment/decrement*.

b. *Loop for-each*

Loop for-each, juga dikenal sebagai *enhanced for loop*, adalah metode yang dirancang khusus untuk iterasi melalui koleksi, termasuk *array*. Dalam pendekatan ini, *programmer* tidak perlu mengelola indeks secara manual, cukup dengan mendeklarasikan variabel sementara yang akan menyimpan nilai elemen saat ini dalam setiap iterasi.

c. *Loop Bersarang (Nested Loop)*

Loop bersarang digunakan ketika bekerja dengan *array* multidimensi, seperti *array* dua dimensi atau lebih. Dalam hal ini, *loop* luar digunakan untuk mengiterasi melalui baris-baris dalam *array*, sementara *loop* dalam digunakan untuk mengiterasi melalui kolom pada setiap baris.

d. Stream API

Sejak diperkenalkannya Java 8, Stream API memberikan cara fungsional untuk melakukan iterasi melalui *array* dan koleksi lainnya. Dengan menggunakan metode seperti *Arrays.stream()*, *programmer* dapat mengonversi *array* menjadi stream dan menerapkan berbagai operasi fungsional seperti filter, map, dan *forEach* secara deklaratif.

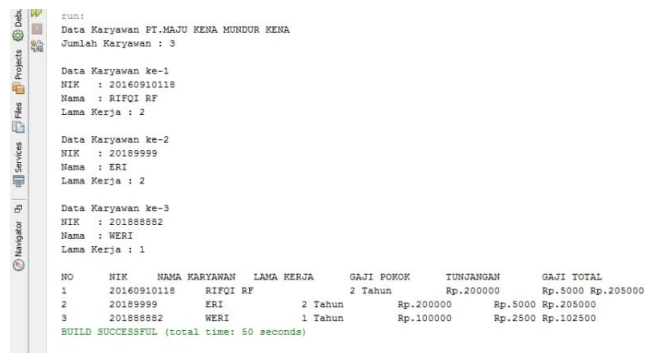
BAB III

PERMASALAHAN

3.1 Permasalahan

a. Buatlah program yang *Outputnya* sebagai berikut :

note : tunjangan = 25% dari gaji pokok



```
TUMI:
Data Karyawan PT.MAJU KENA MUNDUR KENA
Jumlah Karyawan : 3

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

Data Karyawan ke-2
NIK : 20189999
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    RIFOI RF      2 Tahun      Rp.200000    Rp.5000    Rp.205000
2      20189999      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)
```

BAB IV IMPLEMENTASI

4.1 Implementasi

a. Berikut adalah program menghitung gaji karyawan

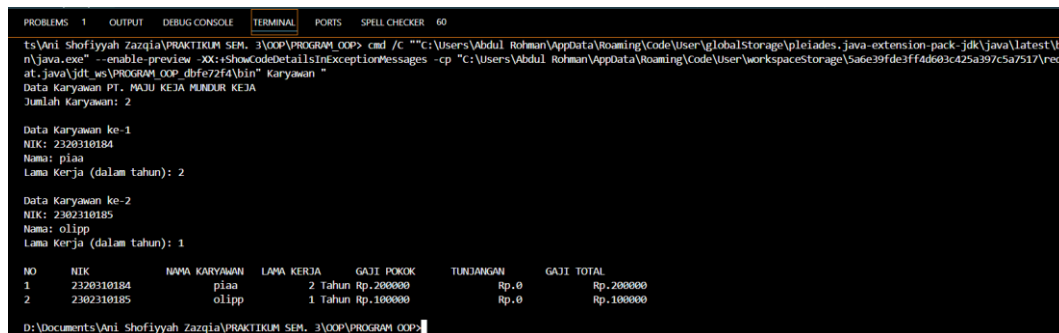
```

1 import java.util.Scanner;
2 public class Karyawan {
3     public static void main(String[] args) {
4         Scanner scanner = new Scanner(System.in);
5         System.out.println("Data Karyawan PT. MAJU KERJA PRIBADI KEJA");
6         System.out.print("Jumlah Karyawan: ");
7         int jumlahKaryawan = scanner.nextInt();
8         String[] nama = new String[jumlahKaryawan];
9         String[] nik = new String[jumlahKaryawan];
10        int[] lamaKerja = new int[jumlahKaryawan];
11        double[] gajiPokok = new double[jumlahKaryawan];
12        double[] tunjangan = new double[jumlahKaryawan];
13        double[] gajiTotal = new double[jumlahKaryawan];
14        for (int i = 0; i < jumlahKaryawan; i++) {
15            System.out.println("Data Karyawan ke- " + (i + 1));
16            System.out.print("NIK: ");
17            nik[i] = scanner.next();
18            System.out.print("Nama: ");
19            nama[i] = scanner.next();
20            System.out.print("Lama Kerja (dalam tahun): ");
21            lamaKerja[i] = scanner.nextInt();
22            if (lamaKerja[i] > 1) {
23                gajiPokok[i] = 200000;
24            } else {
25                gajiPokok[i] = 100000;
26            }
27            tunjangan[i] = 25/100 * gajiPokok[i];
28            gajiTotal[i] = gajiPokok[i] + tunjangan[i];
29        }
30        System.out.println("\nNO\tNIK\tNAMA KARYAWAN\tLAMA KERJA\tGAJI POKOK\tTUNJANGAN\tGAJI TOTAL");
31        for (int i = 0; i < jumlahKaryawan; i++) {
32            System.out.printf("No\t%5s\t%15s\t%5s\t%10s\t%10s\t%10s\n",
33                (i + 1), nik[i], nama[i], lamaKerja[i], gajiPokok[i], tunjangan[i], gajiTotal[i]);
34        }
35        scanner.close();
36    }
37 }

```

Gambar 4. 1 Program Menghitung Gaji

Program ini mengelola data karyawan dengan meminta *input* jumlah karyawan, NIK, nama, dan lama kerja. Berdasarkan lama kerja, gaji pokok dihitung (Rp100.000 atau Rp200.000), tunjangan sebesar 25% dari gaji pokok ditambahkan, lalu dihitung gaji total. Data karyawan ditampilkan dalam tabel berisi NIK, nama, lama kerja, gaji pokok, tunjangan, dan gaji total.



```

ts\ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP> cmd /c "C:\Users\Abdul Rohman\AppData\Roaming\Code\User\globalStorage\pleiades.java-extension-pack-jdk\java\latest\bin\java.exe" --enable-preview -XX:ShowCodeDetailsInExceptionMessages -cp "C:\Users\Abdul Rohman\AppData\Roaming\Code\User\workspaceStorage\Sage39fde3ff4d603c425a397c5a7517\redh at:java\jdk_ws\PROGRAM_OOP_dbg72f4\bin" Karyawan
Data Karyawan PT. MAJU KERJA PRIBADI KEJA
Jumlah Karyawan: 2

Data Karyawan ke-1
NIK: 2320310184
Nama: piaa
Lama Kerja (dalam tahun): 2

Data Karyawan ke-2
NIK: 2302310185
Nama: olipp
Lama Kerja (dalam tahun): 1

NO      NIK      NAMA KARYAWAN      LAMA KERJA      GAJI POKOK      TUNJANGAN      GAJI TOTAL
1       2320310184      piaa      2 Tahun      Rp.200000      Rp.0      Rp.200000
2       2302310185      olipp      1 Tahun      Rp.100000      Rp.0      Rp.100000

```

Gambar 4. 2 Output Program

Output program ini berupa tabel yang menyusun data setiap karyawan secara rapi, menampilkan informasi lengkap mulai dari nomor urut hingga gaji total masing-masing karyawan sesuai dengan lama kerjanya.

BAB V

PENUTUP

5.1 Kesimpulan

Dalam pemrograman Java, terdapat berbagai cara untuk melakukan iterasi melalui elemen dalam *array*, masing-masing dengan kelebihan dan kekurangan yang unik. Metode pertama adalah *loop for* tradisional, yang memberikan kontrol penuh atas proses iterasi dan memungkinkan akses langsung ke indeks. Selanjutnya, *loop for-each* menawarkan cara yang lebih sederhana dan bersih untuk mengakses nilai elemen tanpa memerlukan pengelolaan indeks secara manual, meskipun tidak memungkinkan modifikasi elemen selama iterasi. Untuk *array* multidimensi, *loop* bersarang menjadi pilihan yang efektif, meskipun kompleksitasnya meningkat seiring bertambahnya dimensi. Terakhir, Stream API yang diperkenalkan di Java 8 memberikan pendekatan fungsional yang modern dan deklaratif untuk pemrosesan data, memungkinkan penggunaan operasi lanjutan dan potensi pemrosesan paralel. Dengan memahami berbagai metode ini, *programmer* dapat memilih teknik iterasi yang paling sesuai dengan kebutuhan aplikasi mereka, meningkatkan efisiensi dan keterbacaan kode.

5.2 Saran

Dalam memilih metode iterasi untuk *array* dalam Java, penting bagi *programmer* untuk mempertimbangkan konteks dan kebutuhan spesifik dari aplikasi yang sedang dikembangkan. Jika kontrol penuh atas indeks dan logika iterasi diperlukan, maka *loop for* tradisional adalah pilihan yang tepat. Namun, jika tujuan utama adalah untuk meningkatkan keterbacaan kode dan mengakses nilai elemen tanpa memikirkan indeks, maka *loop for-each* dapat menjadi solusi yang lebih efisien. Untuk pengolahan data dalam *array* multidimensi, penggunaan *loop* bersarang harus dilakukan dengan hati-hati untuk menghindari kompleksitas yang berlebihan.



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

MODUL 5

PEMROGRAMAN BERORIENTASI OBJEK



Java™

ZAIFUR ROHMAN

2302310175

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pemrograman Berorientasi Objek (PBO) dalam Java merupakan sebuah paradigma pemrograman yang menekankan penggunaan objek sebagai elemen utama dalam pengembangan perangkat lunak. Java, sebagai bahasa pemrograman yang sepenuhnya mendukung prinsip PBO, mengorganisasi kode berdasarkan konsep-konsep seperti kelas, objek, pewarisan, polimorfisme, enkapsulasi, dan abstraksi.

Di dalam Java, kelas berfungsi sebagai cetak biru yang mendefinisikan atribut dan metode yang dimiliki oleh objek. Dengan menerapkan enkapsulasi, pengembang dapat melindungi data sensitif dalam objek melalui pengaturan akses yang terbatas. Selain itu, konsep pewarisan memungkinkan suatu kelas untuk mewarisi sifat dan perilaku dari kelas lain, sehingga memudahkan pengelolaan dan penggunaan kembali kode. Polimorfisme memberikan fleksibilitas dalam mendefinisikan perilaku berbeda untuk objek yang bervariasi, sedangkan abstraksi memungkinkan pengembang untuk fokus pada desain tingkat tinggi tanpa terjebak dalam rincian implementasi. Melalui pendekatan ini, Java membantu pengembang menciptakan perangkat lunak yang modular, terstruktur, dan mudah untuk dikembangkan di masa depan.

1.2 Tujuan

- a. Mahasiswa dapat memahami dan menggunakan *encapsulation*
- b. Mahasiswa dapat membuat paket dalam Java

BAB II

DASAR TEORI

2.1 Pengertian OOP

Pemrograman Berorientasi Objek (OOP) merupakan pendekatan dalam pemrograman yang berfokus pada objek, di mana setiap objek memiliki atribut (data) dan metode (perilaku) yang saling berinteraksi. OOP dirancang untuk mempermudah proses pengembangan perangkat lunak dengan mengikuti model yang lebih intuitif dan menyerupai kehidupan sehari-hari. Dalam OOP, program terdiri dari kombinasi objek-objek kecil yang dapat saling berkomunikasi, memungkinkan pengembang untuk membangun sistem yang lebih kompleks dengan cara yang terstruktur dan modular. Konsep-konsep utama dalam OOP mencakup enkapsulasi, pewarisan, polimorfisme, dan abstraksi, yang secara kolektif berkontribusi pada penciptaan kode yang lebih mudah untuk dipelihara, dapat digunakan kembali, dan lebih aman dari kesalahan.

2.2 Konsep Dasar OOP

Ada empat konsep utama dalam Pemrograman Berorientasi Objek (OOP) yang menjadi dasar pengembangan perangkat lunak. Berikut adalah penjelasan rinci mengenai masing-masing konsep tersebut:

a. Abstraksi

Abstraksi adalah konsep yang memungkinkan *programmer* untuk menyembunyikan detail implementasi dari objek dan hanya menampilkan informasi yang relevan kepada pengguna. Dalam OOP, abstraksi membantu memfokuskan perhatian pada fungsi penting dari objek tanpa harus memahami semua detail internalnya.

b. Enkapsulasi

Enkapsulasi adalah teknik yang digunakan untuk membungkus data (atribut) dan metode dalam satu unit, yaitu kelas, serta membatasi akses ke data tersebut dari luar kelas. Konsep ini bertujuan untuk melindungi data dari modifikasi yang tidak diinginkan dan menjaga integritas informasi. Dalam OOP, enkapsulasi dicapai

dengan menggunakan modifier akses seperti *private*, *protected*, dan *public*. Sebagai contoh, dalam sebuah kelas *AkunBank*, atribut seperti saldo dapat dideklarasikan sebagai *private*, sehingga hanya dapat diakses melalui metode publik (*getter/setter*). Hal ini memastikan bahwa saldo tidak dapat diubah sembarangan oleh pengguna, sehingga menjaga keamanan dan konsistensi data.

c. Pewarisan (*Inheritance*)

Pewarisan adalah suatu mekanisme yang memungkinkan sebuah kelas (*subclass*) untuk mewarisi atribut dan metode dari kelas lain (*superclass*). Konsep ini menciptakan hubungan hierarkis antara berbagai kelas, sehingga *subclass* dapat memanfaatkan kembali kode dari *superclass* tanpa harus menulis ulang. Dengan adanya pewarisan, *programmer* dapat menghindari redundansi kode dan mendukung penggunaan kembali kode secara efisien. Sebagai contoh, sebuah kelas *Kendaraan* dapat berfungsi sebagai *superclass* yang memiliki metode umum seperti *bergerak()*, sedangkan kelas *Mobil* dan *Sepeda* dapat berperan sebagai *subclass* yang mewarisi karakteristik dari *Kendaraan* dan mengimplementasikan atau mengoverride metode tersebut sesuai dengan kebutuhan spesifik mereka.

d. Polimorfisme

Polimorfisme adalah konsep yang memungkinkan objek dari berbagai kelas untuk memberikan respons yang berbeda terhadap panggilan metode yang sama. Hal ini memberikan fleksibilitas dalam pemrograman, karena objek-objek tersebut dapat digunakan secara bergantian tergantung pada konteksnya. Polimorfisme dapat diwujudkan melalui dua cara utama, *method overloading*, di mana beberapa metode dengan nama yang sama tetapi parameter yang berbeda didefinisikan dalam satu kelas; dan *method overriding*, di mana *subclass* mengganti implementasi metode dari *superclass* untuk memenuhi kebutuhan spesifik mereka. Sebagai contoh, jika kita memiliki kelas *Hewan* dengan metode *suara()*, *subclass* seperti *Kucing* dan *Anjing* dapat mengoverride metode tersebut untuk menghasilkan suara yang sesuai dengan jenis hewan masing-masing.

2.3 Komponen Dasar dalam OOP

Objek adalah entitas yang memiliki data dan perilaku tertentu. Setiap objek terdiri dari dua karakteristik utama:

a. Objek

Objek adalah entitas yang memiliki data dan perilaku tertentu dalam konteks pemrograman. Setiap objek merupakan instansiasi dari kelas dan memiliki dua karakteristik utama, yaitu atribut dan metode.

1. Atribut

Atribut adalah data atau informasi yang dimiliki oleh suatu objek. Mereka berfungsi untuk menyimpan karakteristik atau keadaan dari objek tersebut. Dalam OOP, atribut dapat dianggap sebagai variabel yang ada di dalam kelas dan merepresentasikan nilai-nilai tertentu yang terkait dengan objek. Misalnya, dalam kelas Mobil, atribut dapat mencakup warna, merek, dan tahun pembuatan. Atribut ini menyimpan informasi spesifik tentang setiap instans objek Mobil. Dengan adanya atribut, *programmer* dapat mengelompokkan data dengan lebih terstruktur, sehingga memudahkan pengaksesan dan manipulasi data di dalam program.

2. Metode

Metode, di sisi lain, adalah fungsi atau perilaku yang dapat dilakukan oleh objek. Mereka mendefinisikan tindakan yang dapat dilakukan oleh objek dan bagaimana objek tersebut berinteraksi dengan data yang dimilikinya. Dalam kelas Mobil, beberapa contoh metode mungkin termasuk jalan(), berhenti(), atau belok(). Metode ini menggambarkan cara kerja objek dan memungkinkan *programmer* untuk mengatur perilaku objek secara terstruktur. Polimorfisme dan *method overriding* adalah dua konsep penting dalam Pemrograman Berorientasi Objek (OOP), namun keduanya memiliki perbedaan yang mendasar. Berikut adalah penjelasan mengenai perbedaan antara keduanya.

a. Polimorfisme

Polimorfisme adalah prinsip dalam OOP yang memungkinkan objek untuk memiliki banyak bentuk atau perilaku. Dalam konteks Java, polimorfisme dapat dibagi menjadi dua jenis: polimorfisme statis dan polimorfisme dinamis. Polimorfisme statis terjadi pada waktu kompilasi, biasanya melalui *method overloading*, di mana beberapa metode dalam satu kelas memiliki nama yang sama tetapi berbeda dalam parameter (jumlah atau tipe). Sebaliknya, polimorfisme dinamis terjadi pada waktu *runtime*, yang dicapai melalui *method overriding*, di

mana *subclass* dapat memberikan implementasi spesifik untuk metode yang sudah didefinisikan di *superclass*.

b. *Method Overriding*

Method overriding adalah teknik di mana *subclass* menyediakan implementasi spesifik untuk metode yang sudah ada di *superclass*. Ini memungkinkan *subclass* untuk mengubah atau memperluas perilaku dari metode *superclass* sesuai dengan kebutuhan spesifiknya. Dalam *method overriding*, nama metode, tipe parameter, dan urutan parameter harus sama persis dengan yang ada di *superclass*. Hal ini memungkinkan JVM (*Java Virtual Machine*) untuk menentukan metode mana yang akan dipanggil pada saat *runtime* berdasarkan tipe objek yang digunakan. Dengan demikian, *method overriding* merupakan contoh dari polimorfisme dinamis, karena keputusan tentang metode mana yang akan dieksekusi ditentukan saat *runtime*.

b. Kelas

Kelas berfungsi sebagai cetak biru atau prototipe untuk menciptakan objek dalam pemrograman berorientasi objek (OOP). Kelas mendefinisikan tipe data yang akan digunakan oleh objek, termasuk atribut dan metode yang dimiliki. Sebagai contoh, sebuah kelas Mobil dapat memiliki atribut seperti warna, merek, dan tahun, serta metode seperti jalan(), berhenti(), dan belok(). Ketika seorang *programmer* membuat objek dari kelas ini, mereka menciptakan instansiasi spesifik dari kelas tersebut dengan nilai tertentu untuk atributnya.

Kelas berperan sebagai kerangka kerja yang memungkinkan *programmer* untuk mengelompokkan data dan fungsi terkait secara logis. Ini memudahkan pengorganisasian kode dan pengembangan aplikasi yang lebih kompleks. Dalam OOP, kelas tidak hanya menyimpan informasi tentang atribut dan metode, tetapi juga menyediakan struktur yang memungkinkan interaksi antar objek. Dengan cara ini, *programmer* dapat membangun aplikasi yang lebih terstruktur dan mudah dipelihara.

2.1 Manfaat Menggunakan OOP

Pemrograman Berorientasi Objek memiliki beberapa fungsi dan manfaat yang signifikan dalam pengembangan perangkat lunak, yang meningkatkan

efisiensi dan efektivitas, tetapi juga memberikan struktur yang lebih baik untuk kode, Berikut adalah penjelasan mengenai fungsi dan manfaat dari OOP:

a. Mempermudah Pengembangan Program

Salah satu fungsi utama OOP adalah mempermudah pekerjaan *programmer* dalam mengembangkan program. Dengan menggunakan model yang mengikuti struktur kehidupan sehari-hari, OOP memungkinkan *programmer* untuk membangun aplikasi yang lebih intuitif dan mudah dipahami. Dalam OOP, objek besar dibentuk dari gabungan beberapa objek kecil yang saling berkomunikasi dan bertukar informasi untuk mencapai hasil akhir, sehingga memudahkan dalam pengelolaan kompleksitas sistem.

b. Meningkatkan Reusabilitas Kode

OOP memungkinkan penggunaan kembali kode yang telah ditulis sebelumnya. Dengan konsep pewarisan, *programmer* dapat membuat kelas baru yang mewarisi atribut dan metode dari kelas yang sudah ada, sehingga tidak perlu menulis ulang kode yang sama. Ini tidak hanya menghemat waktu tetapi juga meningkatkan konsistensi dan dapat mengurangi kemungkinan kesalahan dalam saat kode dijalankan.

c. Modularitas dan Pemeliharaan

OOP mendukung modularitas, di mana setiap objek dapat dianggap sebagai modul terpisah dengan fungsionalitasnya sendiri. Hal ini membuat pemeliharaan kode menjadi lebih mudah karena setiap modul dapat diperbarui atau diperbaiki tanpa mempengaruhi modul lainnya.

d. Keamanan Data

Konsep enkapsulasi dalam OOP membantu melindungi data dengan menyembunyikan detail implementasi dari objek lain. Akses ke data dibatasi melalui modifier akses seperti *private*, *protected*, dan *public*, sehingga hanya metode tertentu yang dapat mengakses atau memodifikasi data tersebut. Ini meningkatkan keamanan informasi dan mencegah perubahan data yang tidak sah.

e. Meningkatkan Fleksibilitas dan Dinamika

Polimorfisme, salah satu prinsip OOP, memungkinkan objek dari kelas yang berbeda untuk merespons dengan cara yang berbeda terhadap panggilan metode yang sama.

BAB III

PERMASALAHAN

3.1 Permasalahan

- a. Buatlah sebuah program java yang mengimplementasikan konsep dasar OOP *class* dan *Object* (atribut dan *method*), setiap mahasiswa tidak boleh sama programnya!

BAB IV

IMPLEMENTASI

4.1 Implementasi

```
1 class Kendaraan {
2     public void bunyiKlakson() {
3         System.out.println("Klakson umum: Beep Beep!");
4     }
5 }
6 class Mobil extends Kendaraan {
7     @Override
8     public void bunyiKlakson() {
9         System.out.println("Klakson mobil: Honk Honk!");
10    }
11 }
12 class Motor extends Kendaraan {
13     @Override
14     public void bunyiKlakson() {
15         System.out.println("Klakson motor: Tiiitt Tiiitt!");
16    }
17 }
18 class Scoopy extends Motor {
19     @Override
20     public void bunyiKlakson() {
21         System.out.println("Klakson Scoopy: Ngeng Ngengg!");
22    }
23 }
24 public class moduls {
25     Run (Debug)
26     public static void main(String[] args) {
27         Kendaraan kendaraanUmum = new Kendaraan();
28         Kendaraan mobil = new Mobil();
29         Kendaraan motor = new Motor();
30         Motor scoopy = new Scoopy(); // membuat objek dari class Scoopy
31         kendaraanUmum.bunyiKlakson();
32         mobil.bunyiKlakson();
33         motor.bunyiKlakson();
34         scoopy.bunyiKlakson();
35     }
36 }
```

Gambar 5. 1 Program OOP

Program ini menunjukkan konsep polimorfisme dan pewarisan dalam Java. Kelas Kendaraan adalah induk dengan metode bunyiKlakson(), sementara Mobil, Motor, dan Scoopy sebagai turunan mengoverride metode tersebut dengan suara klakson berbeda. Di metode *main*, beberapa objek dibuat, kendaraanUmum dari Kendaraan, serta mobil, motor, dan scoopy dari kelas turunan. Saat bunyiKlakson() dipanggil, Java menjalankan implementasi sesuai kelas objek sebenarnya, sehingga setiap objek menghasilkan suara klakson unik. Hal ini mencerminkan kemampuan polimorfisme untuk memberikan perilaku berbeda pada metode yang sama.

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 118
Klakson motor: Tiiitt Tiiitt!
Klakson Scoopy: Ngeng Ngengg!

D:\Documents\Ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP>
D:\Documents\Ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP>
D:\Documents\Ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP> d: && cd "d:\documents\Ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP" && cmd /c "C:\Users\Abdul Rohman\AppData\Roaming\Code\User\globalStorage\plugins\java-extension-pack-jdk\java\latest\bin\java.exe" --enable-preview -XX:ShowCodeDetails=IOExceptionMessages -cp "C:\Users\Abdul Rohman\AppData\Roaming\Code\User\workspaceStorage\5a6e39fde3ffade6ac46a8a0f5a717e0eb4c\java\jdt_ws\Program_OOP_dife72f4\bin" moduls -
Klakson umum: Beep Beep!
Klakson mobil: Honk Honk!
Klakson motor: Tiiitt Tiiitt!
Klakson Scoopy: Ngeng Ngengg!

d:\Documents\Ani Shofiyah Zazqia\PRAKTIKUM SEM. 3\OOP\PROGRAM_OOP>
```

Gambar 5. 2 Output Program

Output program menampilkan suara klakson sesuai kelas objek: "Klakson umum: Beep Beep!", "Klakson mobil: Honk Honk!", "Klakson motor: Tiiitt Tiiitt!", dan "Klakson Scoopy: Ngeng Ngengg!". Polimorfisme memastikan metode sesuai dengan kelas aktual objek.

BAB V

PENUTUP

5.1 Kesimpulan

OOP adalah paradigma yang sangat efektif dalam pengembangan perangkat lunak, dengan empat konsep dasar yaitu objek, kelas, polimorfisme, dan enkapsulasi yang saling berinteraksi untuk menciptakan sistem yang terstruktur dan modular. Objek berfungsi sebagai entitas yang memiliki data dan perilaku, sementara kelas bertindak sebagai cetak biru untuk menciptakan objek tersebut. Polimorfisme memungkinkan objek untuk memiliki banyak bentuk dan perilaku, memberikan fleksibilitas dalam penggunaan metode, sedangkan *method overriding* menjadi salah satu cara untuk mencapai polimorfisme dinamis. Selain itu, OOP menawarkan berbagai manfaat seperti reusabilitas kode, keamanan data melalui enkapsulasi, serta kemudahan pemeliharaan dan pengembangan aplikasi yang kompleks. Dengan menerapkan prinsip-prinsip OOP, *programmer* dapat menciptakan aplikasi yang lebih efisien, mudah dipelihara, dan mampu beradaptasi dengan perubahan kebutuhan di dunia teknologi yang terus berkembang.

5.2 Saran

Penting untuk memahami dan menguasai konsep dasar OOP secara mendalam, termasuk objek, kelas, polimorfisme, dan enkapsulasi, untuk merancang struktur kelas yang jelas dan logis, sehingga memudahkan pengelolaan kode dan kolaborasi dalam tim. Selain itu, disarankan untuk menerapkan prinsip desain yang baik, seperti *Single Responsibility Principle* dan *Open/Closed Principle*, untuk meningkatkan kualitas kode dan meminimalkan ketergantungan antar komponen. Menggunakan alat bantu seperti diagram kelas dapat membantu dalam visualisasi hubungan antar objek dan mempermudah proses pengembangan. Terakhir, selalu lakukan pengujian unit pada metode yang diimplementasikan untuk memastikan bahwa setiap bagian dari aplikasi berfungsi dengan baik dan sesuai dengan harapan. Dengan mengikuti saran-saran ini, pengembang dapat memaksimalkan manfaat OOP dalam menciptakan aplikasi yang *robust* dan *scalable*.



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

MODUL 6

CONSTRUCTOR DAN INHERITANCE



Java™

ZAIFUR ROHMAN

2302310175

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam pemrograman berorientasi objek, konsep *Constructor* dan *Inheritance* memainkan peran yang sangat penting dalam pengembangan aplikasi yang efisien dan terstruktur. *Constructor* adalah metode khusus yang digunakan untuk menginisialisasi objek saat dibuat, memastikan bahwa semua atribut objek memiliki nilai awal yang sesuai. Hal ini sangat krusial dalam menghindari kesalahan yang mungkin terjadi akibat penggunaan objek yang belum terinisialisasi.

Di sisi lain, *Inheritance* memungkinkan pengembang untuk menciptakan hierarki kelas yang lebih terorganisir, di mana *subclass* dapat mewarisi atribut dan metode dari *superclass*. Ini tidak hanya mengurangi duplikasi kode, tetapi juga memfasilitasi pemeliharaan dan pengembangan lebih lanjut dari aplikasi. Dengan menggunakan *Inheritance*, *programmer* dapat membuat kelas yang lebih spesifik berdasarkan kelas umum, sehingga mempermudah pengelolaan kompleksitas dalam kode. Keduanya, *Constructor* dan *Inheritance*, adalah fondasi penting dalam bahasa pemrograman Java, mendukung prinsip-prinsip desain perangkat lunak yang baik dan memfasilitasi pengembangan aplikasi yang *robust* dan *scalable*.

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 *Constructor*

Constructor merupakan elemen penting dalam desain perangkat lunak berorientasi objek, membantu dalam menciptakan kode yang lebih modular, terorganisir, dan mudah dikelola.

a. Pengertian *Constructor*

Constructor dalam Java adalah metode khusus yang digunakan untuk menginisialisasi objek saat objek tersebut dibuat. Setiap kali sebuah objek dari kelas diinstansiasi menggunakan kata kunci *new*, *Constructor* akan dipanggil secara otomatis. Nama *Constructor* harus sama dengan nama kelas dan tidak memiliki tipe pengembalian, termasuk *void*. Terdapat dua jenis *Constructor*, *Constructor default* yang tidak memiliki parameter dan secara otomatis disediakan oleh Java jika tidak ada *Constructor* yang didefinisikan, serta *Constructor* dengan parameter yang memungkinkan pengembang untuk memberikan nilai awal pada atribut objek saat objek tersebut dibuat. Dengan menggunakan *Constructor*, pengembang dapat memastikan bahwa semua atribut objek terisi dengan nilai yang tepat sebelum objek digunakan, sehingga meningkatkan konsistensi dan keandalan data dalam aplikasi. *Constructor* memainkan peran penting dalam mendukung prinsip-prinsip desain perangkat lunak berorientasi objek, seperti enkapsulasi dan penggunaan kembali kode, serta membantu menciptakan kode yang lebih terstruktur dan mudah dikelola.

b. jenis jenis *Constructor*

Dalam Java, terdapat dua jenis *Constructor* yang umum digunakan: *default Constructor* dan *parameterized Constructor*.

1. *Default Constructor*

Default Constructor adalah jenis *Constructor* yang tidak memiliki parameter. Jika Anda tidak mendefinisikan *Constructor* dalam kelas, Java secara otomatis menyediakan *default Constructor*. *Constructor* ini biasanya digunakan untuk memberikan nilai *default* pada atribut objek. Misalnya, jika Anda memiliki kelas *Mobil* yang tidak memiliki *Constructor* yang didefinisikan, Java akan membuatkan

Constructor default yang menginisialisasi semua atribut dengan nilai *default*nya (seperti *null* untuk objek dan 0 untuk tipe numerik). Dengan menggunakan *default Constructor*, *programmer* dapat memastikan bahwa objek selalu memiliki nilai awal yang konsisten, meskipun nilai tersebut mungkin tidak spesifik. Contoh implementasi *default Constructor* dapat dilihat pada kelas Mobil yang menginisialisasi atribut merk dengan nilai "Tidak Diketahui" jika objek dibuat tanpa parameter.

2. *Parameterized Constructor*

Parameterized Constructor adalah jenis *Constructor* yang menerima satu atau lebih parameter, yang digunakan untuk menginisialisasi atribut objek dengan nilai tertentu saat objek tersebut dibuat. *Parameterized Constructor* memberikan fleksibilitas kepada pengembang untuk memberikan nilai awal yang berbeda untuk setiap objek yang dibuat, memungkinkan inisialisasi yang lebih spesifik dan relevan sesuai kebutuhan. Misalnya, pada kelas Mobil, Anda dapat mendefinisikan *parameterized Constructor* yang menerima parameter seperti merk dan tahun. Ketika objek dari kelas tersebut dibuat, pengguna dapat memberikan nilai-nilai ini, sehingga setiap objek dapat memiliki atribut yang sesuai dengan konteks penggunaannya. Dengan cara ini, *parameterized Constructor* membantu meningkatkan keandalan dan konsistensi data dalam aplikasi, karena setiap objek diinisialisasi dengan informasi yang tepat saat dibuat.

c. Pentingnya *Constructor*

Constructor memiliki peranan yang sangat penting dalam pemrograman berorientasi objek, terutama dalam bahasa Java. Pertama-tama, *Constructor* memungkinkan *programmer* untuk menginisialisasi objek dengan nilai awal yang tepat saat objek tersebut dibuat. Dengan menggunakan *Constructor*, pengembang dapat memastikan bahwa semua atribut objek memiliki nilai yang valid dan konsisten sebelum objek digunakan dalam program. Hal ini sangat penting untuk mencegah kesalahan yang mungkin terjadi akibat penggunaan objek yang belum terinisialisasi atau memiliki nilai *default* yang tidak diinginkan.

Selain itu, *Constructor* juga meningkatkan efisiensi dan keamanan program. Dengan memberikan nilai *default* pada atribut melalui *Constructor*, *programmer* dapat menghindari potensi bug yang diakibatkan oleh nilai-nilai yang tidak terduga.

Misalnya, jika sebuah objek memerlukan data dari pengguna saat dibuat, *parameterized Constructor* memungkinkan *programmer* untuk menerima *input* tersebut secara langsung, sehingga memudahkan pengaturan dan pengelolaan data. Ini juga membantu dalam menyederhanakan kode, karena *programmer* tidak perlu menulis kode tambahan untuk mengatur nilai atribut setelah objek dibuat.

Constructor juga mendukung prinsip desain perangkat lunak yang baik, seperti enkapsulasi dan penggunaan kembali kode. Dengan membatasi cara objek diinisialisasi melalui *Constructor*, *programmer* dapat menjaga integritas data dan memastikan bahwa objek dikelola dengan cara yang konsisten. Secara keseluruhan, keberadaan *Constructor* dalam sebuah kelas adalah elemen kunci untuk menciptakan aplikasi yang *robust* dan *scalable*, serta memfasilitasi pengembangan perangkat lunak yang lebih terstruktur dan mudah dipelihara.

2.2 *Inheritance*

Inheritance adalah salah satu konsep utama dalam pemrograman berorientasi objek (*Object-Oriented Programming*) yang memungkinkan sebuah kelas (kelas turunan atau anak) untuk mewarisi properti dan perilaku dari kelas lain (kelas induk atau orang tua).

a. Definisi *Inheritance*

Inheritance adalah konsep fundamental dalam pemrograman berorientasi objek (OOP) yang memungkinkan sebuah kelas (dikenal sebagai *subclass* atau *child class*) untuk mewarisi atribut dan metode dari kelas lain (dikenal sebagai *superclass* atau *parent class*). *Inheritance* bukan hanya sekedar teknik pewarisan namun juga strategi desain sistem yang kuat dalam pemrograman berbasis objek. Dengan menggunakan *Inheritance*, *subclass* dapat mengakses dan memanfaatkan sifat serta perilaku yang telah didefinisikan dalam *superclass* tanpa perlu menulis ulang kode tersebut. Konsep ini mendukung pengorganisasian kode yang lebih baik melalui pembuatan hierarki kelas, di mana kelas yang lebih umum dapat memiliki kelas-kelas yang lebih spesifik di bawahnya. Misalnya, sebuah kelas Hewan dapat menjadi *superclass* untuk kelas-kelas seperti Kucing, Anjing, dan Burung, yang semuanya mewarisi karakteristik dasar dari Hewan tetapi juga dapat memiliki perilaku unik mereka sendiri. *Inheritance* tidak hanya meningkatkan reusabilitas

kode, tetapi juga memfasilitasi pengembangan perangkat lunak yang lebih modular dan mudah dipelihara, memungkinkan *programmer* untuk memperluas fungsionalitas aplikasi dengan lebih efisien.

b. Jenis-jenis *Inheritance*

Jenis-jenis *Inheritance* dalam pemrograman berorientasi objek (OOP) adalah sebagai berikut:

1. *Single Inheritance*

Single Inheritance adalah jenis *Inheritance* di mana sebuah *subclass* mewarisi atribut dan metode dari satu *superclass* saja. Ini adalah bentuk paling sederhana dari pewarisan, yang memungkinkan kelas anak untuk memiliki semua fitur dari kelas induk tanpa kompleksitas tambahan. Misalnya, jika kita memiliki kelas Hewan sebagai *superclass*, maka kelas Kucing dapat menjadi *subclass* yang mewarisi semua atribut dan metode dari kelas Hewan.

2. *Multilevel Inheritance*

Multilevel Inheritance terjadi ketika sebuah *subclass* mewarisi dari *superclass*, dan *subclass* tersebut juga dapat menjadi *superclass* bagi *subclass* lainnya. Ini menciptakan rantai pewarisan yang berlapis-lapis. *Multilevel Inheritance* memungkinkan pengembang untuk membangun hierarki yang lebih kompleks dan terstruktur.

3. *Hierarchical Inheritance*

Hierarchical Inheritance adalah jenis *Inheritance* di mana banyak *subclass* mewarisi dari satu *superclass* tunggal. Dalam hal ini, satu kelas induk memiliki beberapa kelas anak yang masing-masing dapat memiliki atribut dan metode spesifik mereka sendiri. *Hierarchical Inheritance* memudahkan pengelompokan fitur-fitur umum dalam satu *superclass* sambil memberikan fleksibilitas kepada *subclass* untuk menambahkan atau mengubah perilaku mereka sendiri.

4. *Multiple Inheritance*

Multiple Inheritance adalah konsep di mana sebuah *subclass* dapat mewarisi dari lebih dari satu *superclass*. Meskipun ini merupakan fitur yang kuat dalam banyak bahasa pemrograman berorientasi objek, Java tidak mendukung *multiple Inheritance* secara langsung untuk mencegah ambiguitas dan konflik yang mungkin muncul ketika dua *superclass* memiliki metode dengan nama yang sama.

5. *Hybrid Inheritance*

Hybrid Inheritance adalah gabungan dari berbagai jenis *Inheritance* lainnya, seperti multilevel dan *Hierarchical Inheritance*. Dalam *Hybrid Inheritance*, sebuah *subclass* dapat mewarisi dari lebih dari satu *superclass* yang berbeda dengan cara yang terstruktur. *Hybrid Inheritance* memungkinkan pengembang untuk merancang sistem yang sangat fleksibel dan modular dengan memanfaatkan kekuatan berbagai jenis pewarisan sekaligus.

c. Manfaat *Inheritance*

Inheritance dalam pemrograman berorientasi objek (OOP) memiliki berbagai manfaat yang signifikan, yang mendukung pengembangan perangkat lunak yang lebih efisien dan terstruktur. Berikut adalah beberapa manfaat utama dari *Inheritance*:

1. Reusabilitas Kode

Inheritance memungkinkan penggunaan kembali kode dari *superclass* di *subclass*, mengurangi duplikasi dan kesalahan.

2. Pengorganisasian Kode

Inheritance memungkinkan pengembang untuk membuat hierarki kelas yang jelas dan terstruktur. Dengan ini, kelas yang lebih umum dapat memiliki *subclass* yang lebih spesifik, sehingga memudahkan pemahaman dan pengelolaan kode.

3. Modularitas

Memecah kode menjadi modul-modul kecil yang lebih mudah dikelola dan dipelihara. Dengan *Inheritance*, kode dapat dibagi menjadi modul-modul yang lebih kecil dan mudah diatur.

4. Memperluas Fungsionalitas

Subclass dapat memperluas atau memodifikasi fungsionalitas yang diwarisi dari *superclass* tanpa mengubah kode *superclass* itu sendiri.

5. Polimorfisme

Memungkinkan objek dari kelas berbeda untuk digunakan sebagai objek dari kelas induk, meningkatkan fleksibilitas desain.

6. Abstraksi

Kelas abstrak dapat digunakan sebagai blueprint untuk *subclass*, mendefinisikan metode yang harus diimplementasikan oleh *subclass*.

BAB III

PERMASALAHAN

3.1 Permasalahan

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

BAB IV

IMPLEMENTASI

4.1 Implementasi

```

1 import java.util.Scanner;
2 class Starbuck {
3     private int uang;
4     public Starbuck() {
5         this.uang = 0;
6     }
7     public void pesan(String nama, int jumlahPesanan) {
8         System.out.println("Nama pelanggan : " + nama);
9         if (jumlahPesanan < 0) {
10             System.out.println("Pesanan negatif. Tidak boleh memesan.");
11         }
12         System.out.println("Jumlah stok : " + stok);
13         if (stok < jumlahPesanan) {
14             System.out.println("Pesanan habis!!!. " + nama + " memesan " + jumlahPesanan + " nasi sanger.");
15         }
16         System.out.println("Jumlah stok : " + stok);
17     }
18     public int getStok() {
19         return stok;
20     }
21 }
22 public class Starbuck {
23     public static void main(String[] args) {
24         Scanner scanner = new Scanner(System.in);
25         Starbuck starbuck = new Starbuck();
26
27         System.out.println("Selamat datang di Starbuck!");
28         System.out.println("Stok nasi hari ini : " + starbuck.getStok());
29
30         while (starbuck.getStok() > 0) {
31             System.out.println("Masukkan nama pelanggan : ");
32             String nama = scanner.nextLine();
33
34             System.out.println("Masukkan jumlah pesanan untuk " + nama + " : ");
35             int jumlah = scanner.nextInt();
36             scanner.nextLine();
37
38             starbuck.pesan(nama, jumlah);
39
40             if (starbuck.getStok() < 0) {
41                 System.out.println("Stok habis! Tidak dapat melayani pesanan lagi.");
42             }
43             scanner.close();
44         }
45     }
46 }

```

Gambar 6. 1 *Program Constructor*

Program ini adalah simulasi pemesanan kopi di Starbuuk, di mana stok awal kopi ditetapkan sebesar 100 menggunakan konstruktor *default*. Dalam setiap iterasi, pengguna memasukkan nama pelanggan dan jumlah pesanan kopi. Program memeriksa apakah stok mencukupi. Jika cukup, stok akan dikurangi, dan pesanan berhasil. Jika tidak cukup, pesanan gagal dan stok tidak berubah. Proses ini berulang hingga stok habis, kemudian program akan menampilkan pesan bahwa pemesanan tidak dapat dilayani lagi.

[illegible]

Gambar 6. 2 *Output Program*

Output program dimulai dengan menyapa pengguna dan menampilkan stok awal (100). Untuk setiap pelanggan, program mencetak nama mereka, status pesanan (berhasil atau gagal), dan stok yang tersisa setelah pesanan. Jika stok habis, program akan mencetak pesan bahwa pemesanan tidak bisa dilakukan lagi, kemudian berhenti.

BAB V

PENUTUP

5.1 Kesimpulan

Constructor dalam Java berfungsi untuk menginisialisasi objek saat dibuat, memastikan bahwa semua atribut objek memiliki nilai yang sesuai. Ada dua jenis *Constructor*, yaitu *default Constructor*, yang tidak memiliki parameter dan secara otomatis disediakan oleh Java, serta *parameterized Constructor*, yang memungkinkan pengembang memberikan nilai awal saat objek diinstansiasi.

Di sisi lain, *Inheritance* adalah konsep yang memungkinkan sebuah kelas (*subclass*) mewarisi atribut dan metode dari kelas lain (*superclass*). Ini memberikan keuntungan dalam hal reusabilitas kode, karena *subclass* dapat menggunakan kembali fungsionalitas dari *superclass* tanpa perlu menulis ulang kode. Selain itu, *Inheritance* membantu dalam pengorganisasian kode dengan menciptakan hierarki kelas yang jelas, sehingga memudahkan pemahaman dan pemeliharaan.

Terdapat beberapa jenis *Inheritance*, seperti *Single Inheritance*, *multilevel Inheritance*, *Hierarchical Inheritance*, *multiple Inheritance* (yang dapat diatasi dengan interface di Java), dan *Hybrid Inheritance*. Masing-masing jenis ini memiliki cara tersendiri dalam membangun hubungan antara kelas-kelas.

5.2 Saran

Untuk memanfaatkan konsep *Constructor* dan *Inheritance* secara efektif dalam pengembangan perangkat lunak, penting bagi pengembang untuk memahami bagaimana kedua fitur ini dapat meningkatkan kualitas kode. Disarankan agar pengembang merancang kelas dengan baik, menggunakan *Constructor* untuk memastikan objek diinisialisasi dengan benar, dan memanfaatkan *Inheritance* untuk menciptakan hierarki kelas yang logis dan terstruktur. Selain itu, penggunaan interface dapat membantu mengatasi keterbatasan *multiple Inheritance* di Java, memungkinkan fleksibilitas dalam desain. Dengan menerapkan prinsip-prinsip ini, pengembang dapat menghasilkan aplikasi yang lebih modular dan mudah dipelihara.

A. KESIMPULAN

Dalam praktikum ini, kami telah mengeksplorasi berbagai konsep fundamental dalam pemrograman Java yang penting untuk pengembangan perangkat lunak yang efektif. Penggunaan *user input* memungkinkan program berinteraksi langsung dengan pengguna, memberikan fleksibilitas dalam pengolahan data. Kami belajar mengimplementasikan metode untuk menerima dan memvalidasi *input*, meningkatkan pengalaman interaktif.

Konsep perulangan dan percabangan menjadi kunci dalam mengontrol alur eksekusi program. Dengan struktur perulangan seperti *for* dan *while*, kami dapat mengulangi proses hingga kondisi tertentu terpenuhi, sedangkan percabangan menggunakan *if*, *else if*, dan *switch* memungkinkan pengambilan keputusan berdasarkan kondisi yang ada.

Kami juga mempelajari penggunaan *array* untuk menyimpan dan mengelola sekumpulan data secara efisien. Selain itu, pemahaman tentang *Object Oriented Programming* (OOP) membantu kami mengorganisir kode menjadi lebih modular dan mudah dipelihara. Dalam OOP, kami belajar tentang konstruktor untuk menginisialisasi objek dan pewarisan (*Inheritance*) yang menciptakan hierarki kelas kompleks, mengurangi duplikasi kode dan mempercepat pengembangan fitur.

B. SARAN

Berdasarkan pengalaman praktikum pemrograman Java ini, terdapat beberapa saran untuk meningkatkan efektivitas pembelajaran ke depan. Pertama, peserta disarankan untuk lebih banyak berlatih secara mandiri dengan proyek kecil yang mengintegrasikan konsep seperti OOP, *array*, dan struktur kontrol. Ini akan memperkuat pemahaman dan keterampilan dalam menerapkan teori ke praktik. Selain itu, kolaborasi dalam kelompok kecil dapat menjadi metode efektif untuk berbagi pengetahuan dan menyelesaikan masalah bersama, mempercepat proses belajar. Selanjutnya, penting untuk memperdalam pemahaman tentang *debugging* dan teknik pengujian agar program yang dibuat berfungsi dengan baik dan bebas dari kesalahan. Menggunakan alat pengembangan seperti IDE yang mendukung fitur *debugging* sangat membantu dalam proses ini.

DAFTAR PUSTAKA

- Sianipar, R. H. (2015). Pemrograman Java untuk Programmer (Vol. 1). Penerbit ANDI.
- Widayat, A., Prasetya, E., Always, G. M., Anggoro, M. G., & Iksari, I. H. (2023). Pemahaman Konsep Dasar Java: Pendekatan Praktis untuk Mahasiswa. Buletin Ilmiah Ilmu Komputer dan Multimedia (BIKMA), 1(4), 619-622.
- Kadir, A. (2020). Logika Pemrograman Java. Elex Media Komputindo.
- Yuana, R. A., & Kom, M. (2015). Pemrograman Java. Informatika, 1, 77.
- Leitch Robert A.,K. Roscoe Davis. (2005). Analisis &Desain. Andi.Yogyakarta
- Rohma Fathur and Kurniawan, D. (2017). "Pengukuran Kualitas Website Badan Nasional Penanggulangan Bencana Menggunakan Metode Webqual 4.0" Available at:<http://ejournal.nusamandiri.ac.id/ejurnal/index.php/jitk/article/view/432/321>. Diunduh tanggal 20 Nopember 2018.
- Sauro, J. (2011). First Click Testing. Available at: <https://www.usability.gov/howtoand-tools/methods/first-click-testing.html>. Diakses tanggal 20 Nopember 2018.
- Septiani, Desty dkk. (2018). Analisis Dan Perancangan Sistem Pengisian Kartu Rencana Study (KRS) Untuk Jurusan Teknik Informatika Dan Sistem Informasi Kampus Tanri Abeng University. Available at: <http://journal.uinjkt.ac.id/index.php/aism>
- Putra, Y. W. S., Handayani, R. D., Astuti, D., Arifah, F. N., Nasution, E., Sutjiningtyas, S., ... & Sukma, F. (2024). Pemrograman Berorientasi Objek Dengan Java dan Netbeans Ide. TOHAR MEDIA.
- Suryana, T. (2023). Materi 2 Perulangan Dalam Java Script.
- Enterprise, J. (2015). Mengenal java dan database dengan netbeans. Elex Media Komputindo.
- Pratiwi, E. L. (2020). Konsep Dasar Algoritma Dan Pemrograman Dengan Bahasa Java. Poliban Press.
- Khannedy, E. K. (2011). Belajar Java Dasar. Bandung: Strip Bandung.

- Wajhillah, R., Wibowo, A., & Hermaliani, E. H. (2011). Visualisasi Logika Algoritma Pengurutan Data Menggunakan Java. SNIT 2011, 1(1), 280-288.
- Midkiff, S. P., Moreira, J. E., & Snir, M. (1998). Optimizing array reference checking in Java programs. IBM Systems journal, 37(3), 409-453.
- Sari, N., Gunawan, W. A., Sari, P. K., Zikri, I., & Syahputra, A. (2022). Analisis algoritma bubble sort secara ascending dan descending serta implementasinya dengan menggunakan bahasa pemrograman java. ADI Bisnis Digital Interdisiplin Jurnal, 3(1), 16-23.
- Wu, C. T. (2006). An introduction to Object-Oriented Programming with Java. Tata McGraw-Hill Publishing Company Limited.
- Hadiprakoso, R. B. (2021). Pemrograman Berorientasi Objek: Teori dan implementasi dengan Java. RBH.
- Retnoningsih, E., Shadiq, J., & Oscar, D. (2017). Pembelajaran Pemrograman Berorientasi Objek (Object Oriented Programming) Berbasis Project Based Learning. Informatics For Educators And Professional: Journal of Informatics, 2(1), 95-â.
- Enterprise, J. (2015). Mengenal java dan database dengan netbeans. Elex Media Komputindo.
- Wibowo, K. (2015). Analisa Konsep Object Oriented Programming Pada Bahasa Pemrograman PHP. Jurnal Khatulistiwa Informatika, 3(2).
- Rojat, M. R. (2022). Pembelajaran Pemrograman Berorientasi Objek (Object Oriented Programming) Berbasis Project Based Learning. Jurnal Portal Data, 2(7).