LAPORAN PRAKTIKUM PEKAN 7

MATA KULIAH ALGORITMA DAN PEMROGRAMAN

DOSEN PENGAMPU :

WAHYUDI, S.T M.T

OLEH :

MUHAMMAD HANS NAFIS

NIM : 2511532027

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

KATA PENGANTAR

Puji syukur kepada Allah SWT atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Laporan Praktikum Pekan 7 ini tepat pada waktunya. Laporan ini disusun untuk memenuhi salah satu tugas praktikum mata kuliah Algoritma dan Pemograman, dengan topik pembahasan mengenai penggunaan *for* dalam menentukan bilangan prima, *Object-Oriented Programming*, dan manipulasi *String* pada Bahasa Pemrograman Java.

Melalui praktikum ini, penulis mempelajari bagaimana *for* digunakan untuk menentukan apakah bilangan tersebut merupakan bilangan prima, penjelasan *Object-Oriented Programming* beserta contoh nya, dan cara memanipulasi *String*. Penulis berharap laporan ini dapat menambah wawasan serta pemahaman dalam penerapan konsep dasar pemrograman Java.

Penulis menyadari bahwa laporan ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun demi perbaikan di masa mendatang. Akhir kata, penulis mengucapkan terima kasih kepada Bapak Wahyudi S.T. M.T. selaku dosen pengampu mata kuliah Algoritma dan Pemrograman, asisten laboratorium, serta teman-teman praktikum dan pihak lain yang turut mendukung penulisan laporan ini.

Rabu, 12 November 2025

Muhammad Hans Nafis

DAFTAR ISI

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Pemrograman Java merupakan salah satu bahasa pemrograman berorientasi objek yang banyak digunakan dalam dunia akademik maupun industri karena sifatnya yang multiplatform, aman, dan mudah dipahami. Dalam proses pembelajaran pemrograman dasar hingga berorientasi objek, mahasiswa perlu memahami berbagai konsep penting seperti struktur kontrol, penggunaan metode (*method*), manipulasi *string*, serta penerapan prinsip *Object-Oriented Programming* (OOP) seperti enkapsulasi dan pemanggilan objek antar kelas.

Dalam proses pembelajaran, mahasiswa diperkenalkan pada berbagai konsep penting seperti penggunaan variabel, percabangan, perulangan, fungsi (*method*), hingga pembuatan kelas dan objek. Konsep-konsep tersebut melatih mahasiswa untuk berpikir sistematis, menyelesaikan masalah secara logis, serta mampu mengimplementasikan teori pemrograman ke dalam bentuk program nyata. Oleh karena itu, praktikum ini menjadi sarana untuk mengasah kemampuan analisis dan penerapan konsep dasar pemrograman menuju pengembangan sistem yang lebih kompleks di masa depan.

1.2 Tujuan Praktikum

Tujuan dari pelaksanaan praktikum antara lain sebagai berikut :

1. Melatih kemampuan logika melalui penerapan perulangan (*for*) dan struktur kendali (*if–else*) dalam program.
2. Mengimplementasikan konsep kelas dan objek dalam pemrograman berorientasi objek.
3. Menerapkan prinsip enkapsulasi melalui penggunaan *setter* dan *getter*.
4. Mengenal berbagai metode manipulasi *string*.

1.3 Manfaat Praktikum

Manfaat dari pelaksanaan praktikum antara lain sebagai berikut :

1. Menambah pemahaman mahasiswa terhadap dasar-dasar pemrograman Java secara praktis.

2. Membiasakan penggunaan konsep *object-oriented programming* untuk pemecahan masalah.
3. Memperkuat logika algoritmik melalui implementasi fungsi matematis dan pengolahan data teks.

BAB 2

PEMBAHASAN

2.1 *For* dalam Menentukan Bilangan Prima

Dalam bahasa pemrograman Java, *for loop* adalah salah satu bentuk struktur perulangan (*looping*) yang digunakan untuk mengeksekusi satu atau beberapa pernyataan (*statements*) secara berulang, selama kondisi tertentu terpenuhi**.** Struktur ini sangat berguna ketika jumlah pengulangan sudah diketahui sebelumnya (*counted loop*).

Sintaks *for* :

for (inisialisasi; test; update) {

statement;

}

Bagian inisialisasi digunakan untuk menginisialisasi variabel penghitung yang akan digunakan dalam perulangan. Inisialisasi hanya dilakukan sekali. Bagian *test* merupakan ekspresi logika yang akan diperiksa sebelum setiap iterasi. Jika bernilai *true*, blok pernyataan di dalam *loop* akan dijalankan. Jika bernilai *false*, perulangan akan berhenti. Bagian *update* digunakan untuk mengubah nilai variabel penghitung setelah setiap satu kali iterasi selesai, biasanya dengan menambah nilai/*increment* (variabel++) atau mengurangi nilai/*decrement* (variabel--).

Ada hal baru yang dipelajari pada penggunaan *for* kali ini, yaitu *public static boolean*. Ini merupakan kombinasi modifier (*public* dan *static*) dan tipe data (*boolean*) untuk mendeklarasikan metode yang dapat diakses dari mana saja, milik kelasnya sendiri bukan objek, dan mengembalikan nilai kebenaran. Kombinasi ini memungkinkan pemanggilan metode langsung menggunakan nama kelas tanpa perlu membuat objek terlebih dahulu.

Contoh kode program :

```
	public static boolean isPrime(int n) {
		int factors = 0;
		for (int i = 1; i <= n; i++) {
			if (n % i == 0) {
				factors++;
			}
		}
		return (factors == 2);
	}
	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);
```

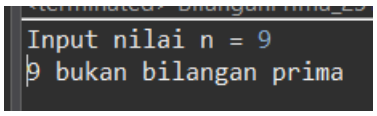
```
            System.out.print("Input nilai n = ");
            int a = input.nextInt();
            if (isPrime(a)) {
                  System.out.println(a + " bilangan prima");
            }else {
                  System.out.println(a + " bukan bilangan prima");
            }
            input.close();
      }

}
```

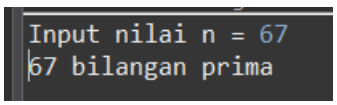
Program Java di atas digunakan untuk menentukan apakah suatu bilangan merupakan bilangan prima atau bukan. Fungsi isPrime(int n) berperan sebagai pemeriksa utama. Di dalam fungsi ini, variabel *factors* digunakan untuk menghitung banyaknya faktor dari bilangan n. Program melakukan perulangan dari 1 hingga n dan setiap kali menemukan bilangan yang habis membagi n (n % i == 0), maka *factors* akan bertambah satu. Setelah perulangan selesai, fungsi akan mengembalikan nilai true jika jumlah faktor sama dengan dua, karena bilangan prima hanya memiliki dua faktor, yaitu 1 dan dirinya sendiri.

Pada bagian *main*, program meminta pengguna untuk memasukkan sebuah nilai n melalui input. Nilai tersebut kemudian dikirim ke fungsi isPrime() untuk diperiksa. Jika hasil pengembalian fungsi bernilai *true*, maka program mencetak bahwa bilangan tersebut adalah "bilangan prima". Sebaliknya, jika hasilnya *false*, maka akan ditampilkan bahwa bilangan tersebut "bukan bilangan prima". Setelah proses selesai, objek *Scanner* ditutup menggunakan input.close() untuk mencegah kebocoran sumber daya. Program ini sederhana dan efektif untuk bilangan kecil, meskipun untuk bilangan besar akan kurang efisien karena memeriksa seluruh pembagi dari 1 sampai n.

Output :



Gambar 2.1



Gambar 2.2

2.2 *Object-Oriented Programming*

Secara umum, OOP adalah paradigma pemrograman yang mengorganisir kode menjadi objek-objek yang menggabungkan data (atribut, properti) dan perilaku (metode) dalam satu kesatuan. Objek mewakili entitas (nyata atau abstrak) dengan keadaan (state) dan kemampuan (behavior). Kelas (*class*) adalah "cetakan" atau blueprint untuk objek-objek tersebut: mendefinisikan atribut + metode. OOP sering digunakan untuk sistem yang besar, kompleks, yang membutuhkan perawatan (*maintainability*), skalabilitas, dan reuse kode.

Ada empat konsep inti yang sering menjadi "pilar" OOP, yaitu :

1. Encapsulation

   Konsep yang mengikat data (atribut) dan kode (metode) dalam satu unit (kelas), serta menyembunyikan bagian-internal objek agar hanya lewat antarmuka tertentu (*public methods*) yang diakses dari luar, yang bermanfaat untuk mencegah akses langsung ke data yang seharusnya "*private*", sehingga meningkatkan keamanan dan konsistensi data, serta memudahkan perubahan implementasi internal tanpa memengaruhi bagian lain dari program yang menggunakan kelas tersebut.

2. Abstraction

   Konsep yang menyederhanakan kompleksitas dengan hanya menampilkan detail yang relevan untuk pengguna objek, dan menyembunyikan detail implementasi yang tidak perlu diketahui. Manfaatnya adalah fokus pada apa yang dilakukan objek, bukan bagaimana ia melakukannya dan membantu menjaga modul-kode tetap bersih dan mudah dipahami.

3. Inheritance

   Kemampuan suatu kelas baru (subkelas) untuk mewarisi atribut + metode dari kelas yang sudah ada (superkelas). Bermanfaat untuk *reuse* kode dan membentuk hierarki kelas yang menggambarkan hubungan "adalah-sebuah" (*is-a*).

4. Polymorphism

   Kemampuan objek untuk "banyak bentuk", artinya metode yang sama bisa bekerja dengan cara berbeda tergantung objek yang memanggilnya, sehingga kode menjadi lebih fleksibel dan generik.

Contoh kode program :

```
public class Mahasiswa_2511532027 {
	//variabel global
	private int nim;
	private String nama,nim2;
	//membuat mutator (setter)
	public void setNim (int nim) {
		this.nim=nim;
	}
	public void setNim2 (String nim2) {
		this.nim2=nim2;
	}
	public void setNama (String nama) {
		this.nama=nama;
	}
	//membuat accesor (getter)
	public int getNim() {
		return nim;
	}
	public String getNim2() {
		return nim2;
	}
	public String getNama() {
		return nama;
	}
	//metode main
	public void Cetak() {
		System.out.println("Nim : "+nim);
		System.out.println("Nama : "+nama);
	}
	public void Cetak2() {
		System.out.println("Nim : "+nim2);
		System.out.println("Nama : "+nama);
	}
}
```

Program di atas berperan sebagai kelas utama (blueprint) yang mendefinisikan struktur dari objek *Mahasiswa*. Di dalamnya terdapat tiga atribut yaitu nim, nim2, dan nama yang semuanya bersifat private, sehingga hanya bisa diakses melalui metode publik seperti *setter* (setNim, setNim2, setNama) dan *getter* (getNim, getNim2, getNama). Hal ini menunjukkan konsep enkapsulasi dalam OOP, di mana data disembunyikan dan hanya bisa dimanipulasi melalui metode khusus. Selain itu, kelas ini juga memiliki dua metode tambahan, yaitu Cetak() dan Cetak2(), yang digunakan untuk menampilkan data mahasiswa ke layar. Dengan demikian, file ini menjadi dasar atau cetakan bagi objek mahasiswa yang akan digunakan pada program lain.

```java
public class PanggilMahasiswa_2511532027 {
	public static void main(String[] args) {
		Mahasiswa_2511532027 a = new Mahasiswa_2511532027();
		a.setNim(23532);
		a.setNama("Rahmat");
		System.out.println(a.getNim());
		System.out.println(a.getNama());
		a.Cetak();
	}
}
```

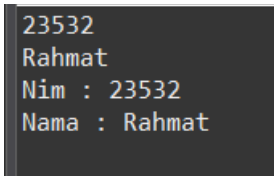
Program di atas berperan sebagai kelas pemanggil (*driver class*) yang mendemonstrasikan cara menggunakan kelas Mahasiswa_2511532027. Di dalam metode main, program membuat sebuah objek a dari kelas Mahasiswa_2511532027, kemudian memanggil metode setter untuk mengisi data seperti nim dan nama. Selanjutnya, program menampilkan data tersebut dengan memanggil metode getter dan Cetak(). Proses ini memperlihatkan bagaimana objek dibuat (instansiasi) dan digunakan dalam OOP, sekaligus menggambarkan hubungan antar kelas, yaitu kelas utama (Mahasiswa) digunakan oleh kelas lain (PanggilMahasiswa) untuk menjalankan fungsinya.

```java
public class PanggilMahasiswa2_2511532027 {
	public static void main (String[] args) {
		Scanner input = new Scanner(System.in);
		System.out.print("NIM : ");
		String x = input.nextLine();
		System.out.print("Nama : ");
		String y = input.nextLine();
		Mahasiswa_2511532027 a = new Mahasiswa_2511532027();
		a.setNim2(x);
		a.setNama(y);
		if (x.startsWith("25")) {
			System.out.println(y + " anda angkatan 2025");
		}
		if (x.contains("1153")) {
			System.out.println("Anda Mahasiswa Informatika");
		}
		a.Cetak2();
		input.close();
	}
}
```

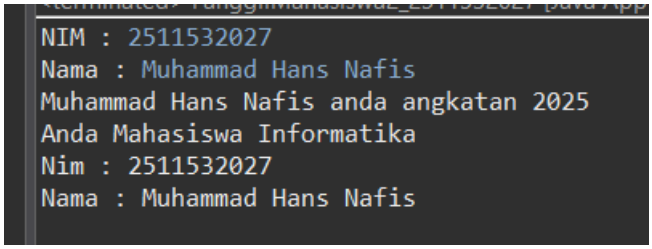
Program di atas berperan sebagai kelas pemanggil, namun dengan fitur yang lebih interaktif. Program meminta pengguna untuk memasukkan data NIM dan Nama melalui input keyboard menggunakan kelas Scanner. Setelah itu, data disimpan ke dalam objek Mahasiswa_2511532027 melalui metode setter, lalu dianalisis menggunakan kondisi if untuk menentukan angkatan dan jurusan berdasarkan pola NIM. Terakhir, data dicetak ke layar dengan memanggil metode

Cetak2(). File ini menunjukkan reusabilitas dalam OOP, di mana kelas Mahasiswa dapat digunakan kembali di program lain tanpa perlu menulis ulang struktur data atau fungsinya, sekaligus memperlihatkan penerapan enkapsulasi dan interaksi antar objek secara nyata.

Output :



Gambar 2.3



Gambar 2.4

2.3 *String*

Dalam Java, *String* adalah tipe data non-primitif yang digunakan untuk menyimpan sekumpulan karakter seperti huruf, angka, atau simbol. *String* bisa dianggap sebagai *array* dari karakter (*char*), tetapi memiliki banyak metode bawaan yang memudahkan manipulasi teks, seperti mencari panjang *string*, menggabungkan *string*, mengambil *substring*, mengganti karakter, dan sebagainya.

String memiliki serangkaian metode bawaan yang dapat digunakan, beberapa contohnya sebagai berikut :

1. Akses karakter
    a. charAt(int index) — Mengembalikan karakter (tipe *char*) di posisi indeks yang ditentukan (indeks mulai dari 0).
    b. codePointAt(int index) — Mengembalikan nilai Unicode (kode *point*) karakter di indeks yang diberikan.
    c. codePointBefore(int index) — Mengembalikan nilai Unicode untuk karakter sebelum indeks yang diberikan.
    d. codePointCount(int beginIndex, int endIndex) — Menghitung jumlah kode point Unicode antara dua indeks.

e. length() — Mengembalikan jumlah karakter (jumlah *char*) dalam *string*.

2. Perbandingan / pemeriksaan kesetaraan
    a. compareTo(String anotherString) — Membandingkan dua *string* secara leksikografis (urutan berdasarkan karakter). Menghasilkan nilai int (<0, =0, >0) tergantung urutan.
    b. compareToIgnoreCase(String str) — Sama seperti di atas, tetapi mengabaikan perbedaan huruf besar/kecil.
    c. equals(Object another) — Mengecek apakah dua *string* persis sama (termasuk huruf besar/kecil). Menghasilkan boolean.
    d. equalsIgnoreCase(String another) — Sama seperti equals, tetapi mengabaikan huruf besar/kecil.
    e. contentEquals(CharSequence cs) — Mengecek apakah *string* ini berisi urutan karakter yang sama sebagai CharSequence yang diberikan.
    f. regionMatches(...) — Memeriksa apakah *region* (sub-bagian) dari dua *string* sama.
3. Pemeriksaan konten / lokasi
    a. contains(CharSequence s) — Mengecek apakah *string* ini mengandung urutan karakter yang diberikan. Mengembalikan boolean.
    b. startsWith(String prefix) — Mengecek apakah *string* ini dimulai dengan urutan karakter tertentu.
    c. endsWith(String suffix) — Mengecek apakah *string* ini diakhiri dengan urutan karakter tertentu.
    d. indexOf(...) — Menemukan indeks kemunculan pertama dari karakter atau *string* yang diberikan. Mengembalikan indeks (int) atau –1 jika tidak ditemukan.
    e. lastIndexOf(...) — Menemukan indeks kemunculan terakhir karakter/*string* yang diberikan.
    f. isEmpty() — Mengecek apakah *string* memiliki panjang 0 (tidak ada karakter).

g. matches(String regex) — Mengecek apakah seluruh *string* cocok dengan ekspresi reguler yang diberikan.

4. Mengubah / memanipulasi *string*
    a. concat(String str) — Menggabungkan dua *string*. Sama dengan menggunakan operator +.
    b. replace(char oldChar, char newChar) atau replace(CharSequence target, CharSequence replacement) — Mengganti karakter atau urutan karakter yang ditemukan dengan yang baru.
    c. replaceAll(String regex, String replacement) — Mengganti semua *substring* yang cocok dengan ekspresi reguler dengan substring pengganti.
    d. replaceFirst(String regex, String replacement) — Mengganti hanya kemunculan pertama yang cocok dengan *regex*.
    e. substring(int beginIndex) atau substring(int beginIndex, int endIndex) — Mengambil bagian dari *string* dari indeks tertentu sampai akhir atau sampai indeks akhir tertentu.
    f. toLowerCase() / toUpperCase() — Mengonversi seluruh *string* menjadi huruf kecil atau huruf besar.
    g. trim() — Menghapus spasi (white space) dari awal dan akhir *string*.
    h. toCharArray() — Mengubah *string* menjadi *array* karakter (char[]).
    i. getBytes() — Mengubah *string* menjadi *array byte* (byte[]) berdasarkan *encoding* default atau *encoding* tertentu.
    j. join(CharSequence delimiter, CharSequence... elements) — Menggabungkan sejumlah *string*/karakter dengan pemisah (*delimiter*).
    k. format(Locale locale, String format, Object... args) — Membuat *string* yang sudah diformat (mirip dengan printf) berdasarkan format dan argumen.
    l. valueOf(...) — Metode statik (umumnya) yang mengembalikan representasi *string* dari suatu nilai (integer, boolean, objek).
    m. toString() — Mengembalikan *string* itu sendiri; karena String sudah objek, biasanya ini hanya pengenalan kembali.

5. Lain-lain
    a. hashCode() — Mengembalikan nilai *hash integer* untuk *string*, digunakan misalnya dalam struktur data *hash* (*hash table*).
    b. getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) — Menyalin karakter dari *string* ke dalam *array* char[].
    c. subSequence(int beginIndex, int endIndex) — Mirip dengan substring, tetapi mengembalikan objek CharSequence (antarmuka) bukan secara spesifik String.
    d. offsetByCodePoints(int index, int codePointOffset) — Metode yang mempertimbangkan kode point Unicode (yang mungkin terdiri dari lebih dari 1 char) ketika menghitung offset.
    e. intern() — Mengembalikan representasi *canonical* dari *string*: jika sudah ada *string* dengan konten yang sama di "*pool*" *string*, maka string yang sudah ada akan dikembalikan; jika belum ada, *string* baru ditambahkan ke pool. (Metode ini berguna untuk menghemat memori jika banyak string identik).

Contoh kode progrm :

```
	public static void main (String[] args) {
		String salam = "Assalamualaikum";
		System.out.println("panjang salam adalah : " +
salam.length());
		System.out.println(salam.toUpperCase());
		System.out.println(salam.toLowerCase());
		System.out.println(salam.indexOf("salam"));
	}
}
```

Program ini mendefinisikan sebuah string bernama salam dengan nilai "Assalamualaikum", lalu menampilkan panjang teks tersebut menggunakan metode .length(). Selanjutnya, program mencetak hasil konversi huruf menjadi kapital dengan .toUpperCase(), huruf kecil dengan .toLowerCase(), dan posisi indeks awal dari *substring* "salam" di dalam *string* utama menggunakan .indexOf("salam"). Program ini bertujuan untuk menunjukkan penggunaan beberapa metode bawaan kelas *String* dalam Java yang berfungsi untuk memanipulasi dan menganalisis teks.

Output :

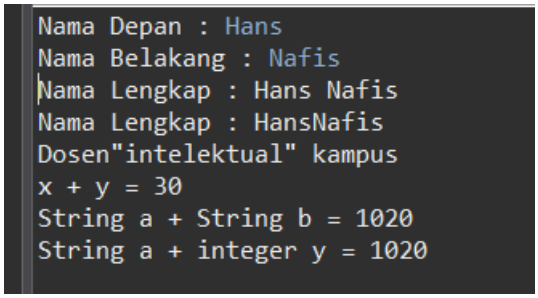
```
panjang salam adalah : 15
ASSAI AMUAI ATKUM
assalamualaikum
2
```

Gambar 2.5

```
	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);
		System.out.print("Nama Depan : ");
		String firstName = input.nextLine();
		System.out.print("Nama Belakang : ");
		String lastName = input.nextLine();
		String txtl = "Dosen\"intelektual\" kampus";
		System.out.println("Nama Lengkap : " + firstName + " " +
lastName);
		System.out.println("Nama Lengkap : " +
firstName.concat(lastName));
		System.out.println(txtl);
		int x = 10;
		int y = 20;
		int z = x + y;
		System.out.println("x + y = " + z);
		String a = "10";
		String b = "20";
		String c = a + b;
		System.out.println("String a + String b = " + c);
		String v = a + y;
		System.out.println("String a + integer y = " + v);
		input.close();
	}
}
```

Program ini memperlihatkan penggunaan kelas *String* dan operasi dasar antara *string* dengan tipe data lain. Program meminta pengguna untuk memasukkan nama depan dan nama belakang, lalu menampilkan hasil gabungannya menggunakan dua cara: dengan operator + dan dengan metode .concat(). Selain itu, terdapat *string txtl* yang menunjukkan penggunaan tanda kutip ganda di dalam teks dengan karakter escape \". Program ini juga mendemonstrasikan perbedaan antara penjumlahan numerik (int x + int y) yang menghasilkan hasil aritmetika, dan penggabungan teks (String a + String b atau String a + int y) yang menghasilkan *string* baru. Secara keseluruhan, program ini bertujuan untuk memperkenalkan konsep manipulasi string dan interaksi antara *string* dan tipe data primitif dalam Java.

Output :



Gambar 2.6

BAB 3
KESIMPULAN

Dari seluruh rangkaian praktikum yang telah dilakukan, dapat disimpulkan bahwa bahasa pemrograman Java memberikan dasar yang kuat dalam memahami struktur program yang terorganisir serta konsep pemrograman berorientasi objek. Melalui implementasi berbagai program seperti pengecekan bilangan prima, pengelolaan data mahasiswa, hingga manipulasi string, mahasiswa belajar menggabungkan logika algoritmik dengan konsep kelas, objek, metode, serta enkapsulasi untuk membangun program yang efisien dan mudah dipahami. Praktikum ini juga menumbuhkan keterampilan berpikir sistematis dalam menyusun kode menggunakan struktur kontrol seperti perulangan dan percabangan, serta memperkenalkan penggunaan fungsi-fungsi bawaan Java untuk memproses data teks. Dengan demikian, kegiatan ini tidak hanya melatih kemampuan teknis menulis program, tetapi juga memperdalam pemahaman konseptual terhadap cara kerja dan keunggulan paradigma berorientasi objek dalam pengembangan perangkat lunak.

Sebagai saran, disarankan mahasiswa lebih aktif berlatih menulis dan menjalankan program secara mandiri di luar jam praktikum. Hal ini penting karena keterampilan pemrograman hanya dapat dikuasai dengan sering berlatih. Dengan demikian, diharapkan pada praktikum berikutnya mahasiswa dapat lebih cepat memahami materi, meminimalisir kesalahan sintaks, serta mampu mengembangkan program yang lebih kompleks dan bermanfaat.

DAFTAR PUSTAKA

[1] https://www.petanikode.com/java-perulangan/ [Diakses 12 November 2025]

[2] https://www.duniailkom.com/tutorial-belajar-java-perulangan-for-bahasa-java/ [Diakses 12 November 2025]

[3] https://www.datacamp.com/doc/java/boolean [Diakses 12 November 2025]

[4] https://www.datacamp.com/doc/java/static [Diakses 12 November 2025]

[5] https://www.datacamp.com/doc/java/public [Diakses 12 November 2025]

[6] https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP [Diakses 12 November 2025]

[7] https://www.geeksforgeeks.org/dsa/introduction-of-object-oriented-programming/ [Diakses 12 November 2025]

[8] https://docs.oracle.com/javase/8/docs/api/java/lang/String.html [Diakses 12 November 2025]

[9] https://www.w3schools.com/java/java_ref_string.asp [Diakses 12 November 2025]

LAMPIRAN