Laboratorium Komputer
Program Studi Informatika

MODUL PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK

Tim Dosen Matakuliah Pemrograman Berorientasi Objek

Program Studi Informatika
Fakultas Teknik
Universitas Sulawesi Barat
2025

VISI DAN MISI PROGRAM STUDI INFORMATIKA FAKULTAS TEKNIK UNIVERSITAS SULAWESI BARAT

Visi

"Menjadi program studi yang unggul di bidang informatika yang berdaya saing di tingkat lokal, nasional, dan global".

Misi

1. Menyelenggarakan proses pembelajaran yang berkualitas guna berdaya saing di tingkat nasional dan internasional.
2. Melaksanakan penelitian berfokus pada Kecerdasan buatan, rekayasa perangkat lunak dan Jaringan yang inovatif, bermutu, bermanfaat.
3. Meningkatkan kontribusi bidang informatika yang berdampak pada masyarakat luas.
4. Meningkatkan jejaring mitra dalam mendukung kualitas pendidikan, penelitian, dan pengabdian.

Tujuan

1. Menghasilkan lulusan kompeten di bidang Informatika yang berakhlak mulia, inovatif, profesional, mandiri serta memiliki semangat technopreneurship yang berdaya saing di tingkat nasional dan internasional.
2. Menghasilkan karya penelitian di bidang informatika berfokus pada Kecerdasan buatan, rekayasa perangkat lunak dan Jaringan yang inovatif, bermutu, dan bermanfaat.
3. Menyebarluaskan hasil penelitian bidang informatika guna memberikan kontribusi bagi peningkatan mutu kehidupan masyarakat.
4. Terwujudnya kerjasama berkelanjutan dengan stakeholder, dunia industri dan lembaga lainnya untuk berkontribusi dalam peningkatan mutu pendidikan, penelitian dan kehidupan masyarakat

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Allah yang Maha Pengasih lagi Maha Penyayang yang telah memberikan rahmat dan karunia-Nya, sehingga kami dapat menyelesaikan penyusunan modul praktikum Pemrograman Berorientasi Objek ini. Modul praktikum ini adalah rangkuman materi untuk mendukung kegiatan praktikum mata kuliah Pemrograman Berorientasi Objek yang terdiri dari sepuluh kegiatan praktikum luring.

Dengan terselesaikannya modul praktikum ini, kami mengucapkan terimakasih banyak kepada semua pihak yang terlibat dalam penyusunan modul praktikum ini.

Demikian modul praktikum Pemrograman Berorientasi Objek ini kami susun, besar harapan kami modul praktikum ini dapat memberi manfaat dalam meningkatkan pengetahuan dan kemampuan pemrograman khususnya Konsep dan Penerapan Pemrograman Berorientasi Objek pada Bahasa Pemrograman Java serta menguatkan kemampuan berpikir komputasional bagi Mahasiswa Informatika. Tak lupa, kami juga mengharapkan kritik dan saran perbaikan untuk penyusunan di masa mendatang.

Majene, 11 April 2025

Tim Penyusun

TIM PENYUSUN

Ir. Sugiarto Cokrowibowo, S.Si., MT
A. Amirul Asnan Cirua, S.T., M.Kom
Mahmuddin
Rich Rona Selamat Marpaung
Deananda
Ahmad Khanif Izzah Arifin
Baso Muchtar Fajar Alghifari
Muhammad Zuhdi

TATA TERTIB PRAKTIKUM

1. Praktikan (mahasiswa peserta praktikum) hadir 10 menit sebelum Praktikum dimulai dengan membawa: Tugas Pendahuluan, Hard Copy Modul Praktikum, berpakaian rapi dan sopan dan menggunakan Name Tag (tertulis nama dan NIM).
2. Kegiatan praktikum terdiri dari: Tugas Pendahuluan, Pre-Test (responsi), Praktikum (Materi Praktikum) dan Tugas Praktikum.
3. Tugas Pendahuluan dan Hard Copy Modul Praktikum Wajib dibawa sebelum masuk ke Lab untuk mengikuti praktikum.
4. Soal-Soal Tugas Pendahuluan dan Tugas Praktikum telah tersedia di dalam Modul untuk setiap praktikum. Jika ada revisi atau tambahan soal akan disampaikan minimal satu hari sebelum praktikum dimulai.
5. Tugas Pendahuluan ditulis menggunakan tangan sendiri (hard writing), di kertas A4 dan diberi sampul sesuai format yang diberikan serta dikerjakan paling lambat sebelum masuk ke pintu Lab sesuai jadwal masing-masing Praktikum.
6. Modul Praktikum Pemrograman Berorientasi Objek serta Name Tag Wajib dibawa dan digunakan di setiap praktikum dan asistensi.
7. Praktikan harus berpakaian rapi dan sopan (celana panjang/rok panjang dan baju kemeja, meskipun menggunakan jaket atau jas almamater tetap harus mengenakan baju kemeja) serta mengedepankan disiplin dan etika di setiap kegiatan yang berhubungan dengan praktikum termasuk konsultasi ke asisten (asistensi).
8. Praktikum dan Asistensi hanya dilaksanakan di lingkungan Fakultas Teknik pada siang hari mulai dari pukul 07.00 sampai 17.30.
9. Responsi (pretest) dilaksanakan sebelum memulai praktikum, di dalam ruangan Lab setelah praktikan masuk ke ruangan lab dan pemeriksaan di pintu masuk berakhir.
10. Praktikan dengan nilai responsi ≤ 50 dinyatakan GAGAL RESPON dan tidak diperkenankan mengikuti praktikum (dipersilahkan keluar dari ruangan/tidak perlu mengikuti dan melanjutkan praktikum pada jadwal tersebut

11. Praktikan yang dinyatakan GAGAL RESPON pada suatu praktikum maka tidak perlu mengumpulkan Tugas Praktikum dan diberikan nilai tugas praktikum = 0.
12. Tugas Praktikum harus diasistensikan minimal satu kali dan dinyatakan diterima (ACCEPTED) oleh Dosen/Asisten sebelum dikumpul.
13. Tugas Praktikum yang terlambat dikumpulkan atau tidak dinyatakan diterima oleh Dosen/Asisten (tidak mendapatkan ACCEPTED) maka diberikan nilai tugas praktikum = 0 (dianggap tidak mengumpulkan tugas praktikum).
14. Tugas Praktikum harus dikumpulkan paling lambat 24 Jam sebelum Praktikum berikutnya dimulai. Jika terlambat mengumpulkan tugas praktikum maka diberikan nilai tugas praktikum = 0 pada praktikum tersebut.
15. Praktikan yang diberikan nilai tugas praktikum = 0 pada suatu praktikum maka dinyatakan GAGAL PRAKTIKUM pada praktikum tersebut.
16. Praktikan yang tidak mengikuti praktikum maka dinyatakan GAGAL PRAKTIKUM.
17. Praktikan yang telah mendapatkan GAGAL PRAKTIKUM sebanyak empat kali dengan alasan apapun (GAGAL RESPON, tidak hadir, tidak mengikuti praktikum, terlambat mengumpulkan tugas praktikum, tidak mendapatkan ACCEPTED tugas praktikum, tidak mengumpulkan tugas praktikum maupun pelanggaran lainnya) maka dinyatakan GAGAL LAB untuk Praktikum Pemrograman Berorientasi Objek Semester genap 2023/2024 dan tidak perlu melanjutkan ke praktikum berikutnya.
18. Ada tiga kategori GAGAL di dalam praktikum ini: GAGAL RESPON, GAGAL PRAKTIKUM, dan GAGAL LAB.
19. GAGAL RESPON terjadi jika nilai Responsi Praktikan ≤ 50.
20. GAGAL PRAKTIKUM dapat disebabkan oleh: GAGAL RESPON, nilai tugas praktikum = 0, Terbukti Melanggar Tata Tertib Praktikum, Melakukan Kecurangan saat Responsi, tidak hadir, tidak mengikuti praktikum, terlambat mengumpulkan tugas praktikum, tidak mendapatkan ACCEPTED tugas praktikum, tidak mengumpulkan tugas praktikum.
21. GAGAL LAB terjadi jika praktikan telah memperoleh GAGAL PRAKTIKUM sebanyak empat kali

22. Pelanggaran berat seperti pengrusakan alat-alat dan laboratorium akan ditindak sesuai peraturan yang berlaku di masing-masing Laboratorium.

BELAJAR PEMROGRAMAN DENGAN CEPAT DAN EFISIEN

Ini pertanyaan yang sering ada di benak mahasiswa informatika: “Bagaimana sih caranya belajar dan menguasai pemrograman dengan cepat dan efisien?”

Walaupun tidak mudah, tetapi sebenarnya hal ini tidak sesulit yang mungkin orang perkirakan jika kita bersedia menjalani dengan penuh dedikasi, semangat, dan konsistensi. Berikut ini beberapa tips yang mungkin dapat dilakukan agar perjalanan kalian dalam mempelajari pemrograman bisa lebih cepat dan efisien.

1. KENALI DAN CINTAI

Langkah pertama dalam belajar seharusnya dimulai dengan mengenal dan mencintai agar langkah berikutnya menjadi ringan. Dimulai dengan mengenali dan mencari tau apa saja yang berkaitan dengan pemrograman, benefit apa saja yang akan kita dapatkan setelah menguasai pemrograman, sejarah pemrograman yang sedang kita pelajari, dari mana kita bisa mempelajarinya, dimana kita bertanya jika mendapatkan kesulitan, sampai tools apa saja yang akan kita butuhkan untuk mempelajari pemrograman, dll. Kita cari tahu semua itu agar bisa lebih kenal. Selanjutnya mencintai. Mencintai pemrograman. Beberapa hal dalam hidup kita memang terjadi tanpa kita diberi kesempatan untuk memilih dan mungkin tidak sesuai dengan yang kita harapkan, atau mungkin kita merasa telah salah memilih dan terlambat menyadari. Hal terbaik dari semua itu adalah bahwa kita harus memulai belajar untuk mencintai atau bahkan mungkin memaksakan diri untuk mencintai. Mencintai pemrograman.

2. PELAJARI KONSEP SKIL DASAR (FUNDAMENTAL) PROGRAMMING

Langkah awal yang tidak boleh terlewatkan adalah memahami konsep dasar programming. Jika ingin memahami tahap selanjutnya yang lebih kompleks dalam pemrograman anda harus memiliki dasar-dasar yang kuat. Dasar-dasar pemrograman ini seperti: algoritma, struktur data, variabel, tipe data, operasi, input/output, logika, looping, conditional, dan penulisan sintaks program. Dengan penguasaan fundamental yang kuat ini akan memungkinkan anda menjadi mudah beradaptasi, meminimalisir stuck dalam belajar, dan materi selanjutnya yang lebih kompleks akan menjadi mudah

dipahami. Jika kita memiliki skill fundamental yang kuat maka kelak tidak akan sulit bagi kita untuk menguasai framework yang lebih kompleks.

3. FOKUS PADA SATU BAHASA PEMROGRAMAN

Fokuslah pada satu bahasa pemrograman dulu karena ini akan membuat proses belajar programming anda menjadi lebih cepat dan efisien. Ada banyak bahasa pemrograman yang mungkin cukup menggoda untuk ingin anda pelajari dan kuasai dengan sesegera mungkin, tetapi akan menjadi berat bagi pemula jika ingin menguasainya sekaligus dalam waktu bersamaan jika belum memiliki fundamental programming yang kuat. Umumnya bahasa pemrograman dibuat sebagai bahasa pemrograman general purpose artinya bahasa-bahasa pemrograman tersebut secara fungsional dapat saling menggantikan dan berarti juga dengan menguasai satu bahasa pemrograman maka secara konsep kita juga dapat menguasai bahasa pemrograman yang lain. Dengan menguasai satu bahasa pemrograman disertai penguasaan fundamental programming yang kuat maka masalah bahasa pemrograman yang lain itu sebenarnya hanya MASALAH SINTAKSIS, hanya masalah penulisan sintaks source code saja. Jadi sebenarnya dengan menguasai satu bahasa pemrograman akan SANGAT memungkinkan anda menjadi seorang Polyglot bahasa pemrograman. Jika saat anda berada di semester akhir program studi informatika dan sudah banyak mempelajari bahasa pemrograman tapi belum ada satupun yang dikuasai, coba evaluasi mungkin memang belum ada satupun bahasa pemrograman yang betul-betul anda pelajari dan terlalu menginginkan segera menguasai semuanya.

4. LEARNING BY DOING

Materi dan konsep pemrograman dapat kita pelajari dengan membacanya seperti Novel, tetapi kesalahan kita adalah ketika kita berhenti sampai di situ, yaitu hanya membaca buku/materi dan melihat contoh kode program di layar monitor tanpa mempraktekkan secara langsung. Karena ketika kita menulis kode, mengoptimalkan kode, mencoba solusi yang berbeda, berjumpa dengan Error, itu akan membuat kemampuan berpikir logis, kemampuan berpikir komputasional dan problem solving anda meningkat dari hari ke hari. Semakin sering anda berlatih maka akan semakin lihai anda dalam menyusun kode pemrograman dan menyelesaikan error program.

Selain itu anda dapat mencoba untuk membuat project pribadi, misalnya project portfolio atau project yang berkaitan dengan rencana riset anda, atau project mandiri lainnya, karena hal ini akan membuat anda bisa belajar memahami pemrograman dengan lebih cepat. Ketika anda melihat sebuah aplikasi atau sistem yang mungkin dibuat dengan entah bahasa pemrograman apa, bahasa pemrograman yang tidak anda ketahui. Cobalah buat project mandiri dari hal tersebut dan membangunnya dengan bahasa pemrograman yang sedang anda pelajari.

5. GUNAKAN METODE CODE BY HAND

Ini mungkin terdengar aneh bagi sebagian orang karena mungkin menurutnya seharusnya coding dilakukan menggunakan komputer, bukan secara manual dengan menulis. Namun sebenarnya coding manual dengan tulisan tangan akan memberi anda pemahaman tentang sintaksis dan algoritma. Selain itu hal ini dapat membantu anda dalam proses interview kerja karena tidak sedikit interview kerja di bidang pemrograman akan meminta untuk membuat coding secara manual dengan tulisan tangan. Walaupun terlihat klasik, tapi sebenarnya test semacam ini cukup efektif untuk melihat kemahiran seorang programmer.

6. GUNAKAN PERTANYAAN BAGAIMANA JIKA

Program anda sudah berhasil berjalan dengan benar. Anda telah berhasil mereplikasi contoh kode dari buku atau materi lainnya. **Jangan berhenti** sampai disitu, lakukan sedikit lagi pertanyaan analitik BAGAIMANA JIKA???. Buat modifikasi pada program yang telah anda buat. Temukan kemungkinan bug dan errornya. Lakukan perkiraan Error Logika atau Runtime Error yang mungkin akan terjadi JIKA... Selalu tanyakan pada diri anda: **BAGAIMANA JIKA...???**

7. JANGAN MALU BERTANYA

Jika menemukan kebuntuan, tak perlu malu bertanya. Kita masih dalam tahap belajar. Bertanya tidak akan mengurangi apapun dalam diri kita. Jangan malu bertanya. Tanyakan kepada dosen ataupun teman jika kita mengalami masalah pemrograman. Karena dosen dan teman bisa menjadi shortcut kita untuk lebih cepat lagi menguasai pemrograman. Jikapun kita belum berhasil mendapatkan informasi dan jawaban dari teman dan dosen setidaknya kita telah berkontribusi membagikan informasi tentang masalah pemrograman

tersebut. Diskusilah, jangan malu bertanya, dan sebaliknya jika anda tahu juga jangan sungkan untuk membagikan informasi yang anda miliki tentang pemrograman.

8. IKUTILAH ONLINE COURSE

Mengikuti online course adalah salah satu cara lain yang dapat anda pilih jika ingin belajar pemrograman karena umumnya online course memang dirancang secara terstruktur yang memungkinkan anda untuk menguasai pemrograman secara singkat. Materi online course dirancang untuk mempersiapkan diri memasuki dunia kerja berdasarkan pengalaman dari si pembuat online course, jadi ada semacam learning path dan shortcut belajar di sana.

9. TONTON YOUTUBE FOSALGO

Tips belajar yang tidak kalah pentingnya adalah nonton youtube di channel FOSALGO. Ada banyak materi, contoh penerapan dan konsep pemrograman di channel tersebut terutama yang berkaitan dengan Pemrograman Java dan Pemrograman Berorientasi Objek. Selain itu dari channel FOSALGO ini kalian bisa bertanya dan bertemu langsung dengan si pembuatnya di Program Studi Informatika Unsulbar :D :D :D .

10. JANGAN LUPA ISTIRAHAT

Belajar programming dengan duduk di depan komputer selama berjam-jam dan mencoba memahami semuanya sekaligus tentunya hal yang sangat melelahkan. Selain itu, kebiasaan tersebut juga tidak baik karena tubuh juga perlu istirahat. Cobalah untuk belajar sedikit demi sedikit, tetapi konsisten setiap harinya. Selain itu, cobalah untuk menghilangkan semua jenis gangguan, seperti mematikan notifikasi telepon, notifikasi email, dan coba isolasi diri agar anda fokus. Dengan melakukan hal-hal tersebut, anda akan menghemat banyak waktu dan terhindar dari burnout ataupun frustasi.

Nah, demikianlah beberapa tips yang dapat membantu proses belajar programming anda lebih cepat dan efektif. Mungkin di tengah perjalananmu menjadi seorang programmer akan merasakan stuck ataupun burnout, tetapi perlu diingat bahwa menjadi seorang programmer yang andal juga butuh konsistensi untuk terus berlatih.

Tetap Semangat...
Jangan Menyerah...
Jangan Lupa Berdoa dan Bersyukur...
Berjuanglah...

DAFTAR ISI

VISI DAN MISI PROGRAM STUDI INFORMATIKA ii
KATA PENGANTAR iii
TIM PENYUSUN iv
TATA TERTIB PRAKTIKUM v
BELAJAR PEMROGRAMAN DENGAN CEPAT DAN EFISIEN viii
DAFTAR ISI xiii
RINGKASAN PRAKTIKUM xiv
PRAKTIKUM 1: INTRODUCTION 1
PRAKTIKUM 2: CLASS AND OBJECT 28
PRAKTIKUM 3: ENCAPSULATION, INHERITANCE AND POLYMORPHISM 62
PRAKTIKUM 4: GRAPHICAL USER INTERFACE JAVAFX 85
PRAKTIKUM 5: ALGORITHMS AND DATA STRUCTURES I 114
PRAKTIKUM 6: ALGORITHMS AND DATA STRUCTURES II 143
PRAKTIKUM 7: HEURISTIC SEARCH ALGORITHM 170
PRAKTIKUM 8: METAHEURISTIC SEARCH ALGORITHMS & MACHINE LEARNING 178

RINGKASAN PRAKTIKUM

Nama Mata Kuliah : Pemrograman Berorientasi Objek
Jumlah Pertemuan Praktikum : 8

Tabel Materi Praktikum

Pertemuan Ke-	Judul Materi Praktikum
1	INTRODUCTION
2	CLASS AND OBJECT
3	ENCAPSULATION, INHERITANCE AND POLYMORPHISM
4	GRAPHICAL USER INTERFACE JAVAFX
5	ALGORITHMS AND DATA STRUCTURES I
6	ALGORITHMS AND DATA STRUCTURES II
7	HEURISTIC SEARCH ALGORITHM
8	METAHEURISTIC SEARCH ALGORITHM & MACHINE LEARNING

PRAKTIKUM 1: INTRODUCTION

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator Penilaian

1. Mahasiswa mampu menjelaskan konsep dasar pemrograman berorientasi objek
2. Mahasiswa mampu menjelaskan dasar pemrograman Java
3. Mahasiswa mampu menggunakan berbagai tipe data primitif dan non primitif untuk membuat variabel
4. Mahasiswa mampu membuat operasi sederhana di dalam method main
5. Mahasiswa mampu membuat class menggunakan Java
6. Mahasiswa mampu membuat field dan method
7. Mahasiswa mampu membuat variabel di dalam class (field)
8. Mahasiswa mampu membuat variabel di dalam method (local variable)
9. Mahasiswa mampu membuat variabel untuk input method (parameter)
10. Mahasiswa mampu membuat operasi di dalam method
11. Mahasiswa mampu membuat operasi conditional di dalam method
12. Mahasiswa mampu membuat operasi looping di dalam method
13. Mahasiswa mampu membuat input-output program melalui file dan console window
14. Mahasiswa mampu menjelaskan tentang String dan Regular Expression

B. Tugas Pendahuluan

1. Gambarkan hubungan antara state dan behavior pada object di dunia nyata dengan field dan method pada object di software.
2. Tuliskan dan jelaskan tiga pilar object oriented programming.
3. Tuliskan empat macam visibilitas yang ada di java.
4. Jelaskan konsep dasar tentang class di java.
5. Tuliskan 50 keyword java.
6. Apa yang dimaksud: keyword java bersifat case-sensitive.
7. Tuliskan 8 tipe data primitif yang ada di java beserta range/ jangkauannya.
8. Tuliskan paling sedikit 20 tipe data non primitif yang ada di java.

9. Jelaskan apa yang dimaksud dengan Instance Variables, Class Variables, Local Variables, dan Parameter Variables di java.
10. Buatlah tabel operator-operator yang tersedia di Java.
11. Tuliskan disertai contoh program untuk semua bentuk variasi CONDITIONAL yang ada di Java.
12. Tuliskan disertai contoh program untuk semua bentuk variasi LOOP yang ada di Java.
13. Buatlah 6 program dengan variasi berbeda untuk menangani Input-Output di java melalui Console Window dan File.
14. Buatlah sebuah method sederhana kemudian panggil method tersebut melalui method main.
15. Jelaskan perbedaan antara method void dan method non-void.
16. Jelaskan cara kerja manual lalu susunlah algoritma sieve of Eratosthenes untuk mencari semua bilangan prima yang lebih kecil dari n. Kemudian susunlah source code java untuk mengimplementasikan algoritma sieve of Eratosthenes. Program akan menerima input berupa bilangan bulat n, kemudian program akan menampilkan semua bilangan prima antara 1 sampai n.
17. Buatlah program untuk menjumlahkan bilangan yang sangat besar. Buatlah sebuah variabel a berisi bilangan yang sangat besar: 10000000000000000000000000000000000000001. (ada 40 angka nol di antara angka 1). Lalu jumlahkan nilai a dengan variabel b berisi bilangan 22. Lalu tampilkan hasil penjumlahannya. Programmer diminta untuk menentukan sendiri tipe data dari variabel a dan b kemudian membuat operasi penjumlahannya.

    a = 10000000000000000000000000000000000000001
    b = 22
    c = a + b
    cetak nilai c

C. Ringkasan Materi

Pada praktikum ini kita akan membahas tentang dasar-dasar pemrograman Java dan Pemrograman Berorientasi Objek meliputi: Tipe Data dan Operator di Java, Input dan Output, Operator Looping dan Operator Conditional.

BAHASA PEMROGRAMAN JAVA

Berikut ini 50 keyword (kata cadangan) Java

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

*	not used	|	***	added in 1.4
**	added in 1.2	|	****	added in 5.0

TIPE DATA (DATA TYPE)_____

Tipe data adalah tipe atau jenis dari data yang digunakan untuk menangani variabel. Kita dapat menganalogikan variabel dan tipe data ini seperti sebuah wadah (variabel) dan jenis atau bahan untuk membuat wadah (tipe data) sedangkan nilai yang tersimpan (value) dari variabel dapat dianalogikan sebagai material atau bahan atau benda yang tersimpan di dalam wadah. Ada macam macam wadah: ember, keranjang, kantong kresek, botol, gelas kaca, baskom, dll. Ini menggambarkan bermacam-macam variabel untuk menyimpan value. Wadah wadah ini memiliki bentuk, volume, dan bahan untuk membuat wadah yang berbeda-beda. Perbedaan bentuk, volume dan bahan ini adalah analogi tipe data yang berbeda-beda. Sehingga value (nilai data) yang dapat disimpan pada variabel(wadah) juga tergantung dari tipe datanya (tipe wadahnya) kita akan kesulitan untuk menyimpan air di keranjang, tetapi keranjang dapat digunakan untuk menyimpan buah. Demikian pula Variabel, Tipe Data, dan Value. Kita membutuhkan Tipe Data yang tepat untuk menyimpan Value. Tipe data double (decimal) dapat digunakan untuk menyimpan bilangan bulat (int), tetapi tidak semua bilangan decimal dapat disimpan dalam variabel bertipe int.

Analogi untuk hubungan antara **tipe data**, **variabel**, dan **nilai variabel** dapat kita lihat pada sebuah Keranjang Rotan berisi 12 Mangga. Disini kita dapat menganalogikan Keranjang sebagai sebuah variabel, dan keranjang tersebut terbuat dari rotan artinya tipe datanya adalah rotan yang tentunya tidak dapat kita gunakan untuk menyimpan air tapi dapat kita gunakan untuk menyimpan mangga. Berikutnya mangga sebanyak 12 buah itu dapat kita analogikan sebagai nilai dari variabel.

8 Tipe Data Primitif dan Tipe Data Non-Primitif di Java.

Pada Bahasa pemrograman Java ada dua jenis tipe data: Tipe Data Primitif (System-defined data types) dan Tipe Data Non-primitif (User-defined data types) (Karumanchi, 2020). Ada 8 tipe data primitif di Java: byte, short, int, long, float, double, char, dan boolean. Sedangkan untuk tipe data non-primitif ada lebih banyak lagi dan tidak terhitung jumlahnya dikarenakan selain dari yang telah tersedia di Java (JDK) programmer juga dapat membuat sendiri tipe data non primitif ini (User-defined data types). Untuk membuat sendiri tipe data non-primitif ini di Java dilakukan dengan cara membuat Class, Enum, ataupun Interface. Istilah lain untuk tipe data non-primitif di java sering juga disebut tipe data reference. Beberapa tipe data reference yang telah tersedia ji Java diantaranya: Array, String, Tipe Wrapper (Byte, Short, Integer, Long, Float, Double, Character, dan Boolean). Kita juga dimungkinkan untuk membuat tipe data non-primitif kita sendiri, sehingga jumlah tipe data non-primitif di java ini menjadi tak berhingga banyaknya.

Catatan penting: di Java (byte, short, int, long, float, double, char, dan boolean merupakan tipe primitif) berbeda dengan Wrapper (Byte, Short, Integer, Long, Float, Double, Character, dan Boolean yang merupakan tipe non-primitif)

Berikut ini 8 tipe data primitif beserta range datanya di java:

Bilangan Bulat (Integer)

Ada empat tipe data primitif di java yang dapat digunakan untuk menangani bilangan bulat sesuai dengan Panjang jangkauan nilainya:

1. byte: length = 1 byte atau 8 bits. Nilai MIN = -128 dan nilai MAX = 127 (inclusive).
2. short: length = 2 byte atau 16 bits. Nilai MIN = -32,768 dan nilai MAX = 32,767 (inclusive).
3. int: length =4 byte atau 32 bits. Nilai MIN = -231 dan nilai MAX = 231-1

4. long: length = 8 byte atau 64 bits. Nilai MIN = -263 dan nilai MAX = 263-1

Bilangan Riil (Real Number)

Java memiliki dua tipe data primitif untuk menangani bilangan real yaitu tipe double dan float.

5. float: length = 4 byte atau 32 bits.
6. double: length = 8 byte atau 64 bits.

Char dan Boolean

7. char: length = 2 byte atau 16 bits
8. boolean: length = 1 bit

VARIABEL_____

Telah disebutkan sebelumnya bahwa dari sisi pemrograman, object akan menyimpan state menjadi variabel dan behavior akan dijadikan sebagai method.

```
//variables
String warna;
String jenisTransmisi;
double kecepatan;
```

Pada Bahasa pemrograman java didefinisikan empat jenis variabel berikut: Instance Variables, Class Variables, Local Variables, dan Parameter Variables.

1. Instance Variables atau sering disebut juga non-static fields (dideklarasikan tanpa menggunakan modifier static) merupakan variable-variabel yang unik untuk setiap object (setiap object akan memiliki nilai Instance variables-nya sendiri)
2. Class Variables atau sering disebut juga static field adalah variabel yang dideklarasikan menggunakan modifier static. Secara teori hanya ada satu copy untuk Class Variable berapapun banyaknya object hasil instan dari class tersebut. Class Variables ini dapat digunakan tanpa kita harus terlebih dahulu membuat instan dari classnya.
3. Local Variables mirip seperti bagaimana object menyimpan nilai state ke dalam field, sebuah method juga akan menyimpan nilai state sementara (temporary state) di dalam local variables (variabel lokal). Local variables ini hanya dapat dikenali atau diakses di dalam method tempat dimana local variable ini di deklarasikan. Cara mendeklarasikan local variables ini sama

dengan cara mendeklarasikan field namun tidak dapat diberi modifier static, modifier visibilitas (public, private, protected) maupun modifier final dan abstract. Local Variables ini terbungkus oleh method tempat dimana mereka di deklarasikan dan tidak dapat dikenali dan dipanggil dari method lain, class lain maupun melalui object hasil instan dari class di mana method tersebut berada. Local Variables terbungkus di dalam method tetapi dapat mengakses field yang ada di kelas tersebut dan dapat menerima input dari parameter method serta dapat menghasilkan output yang dikirim ke luar method menggunakan keyword return.

4. Parameter Variables atau biasa disebut parameter saja, merupakan variabel variabel yang menjadi input sebuah method. Variabel ini dituliskan bersama deklarasi tipe datanya di dalam kurung yang diletakkan setelah nama method. Kita telah menggunakan parameter ini sebelumnya untuk membuat method main. public static void main(String[ ] args){ }. Deklarasi variabel yang diapit oleh tanda kurung String[ ] args yang disebut sebagai parameter. Sifat dari parameter ini sama seperti local variables hanya saja cara mendeklarasikannya yang berbeda. Parameter hanya dikenali secara local di dalam method tetapi nilainya diperoleh saat method tersebut dipanggil, karena itu parameter ini disebut sebagai input method.

Baik local variables maupun parameter variables keduanya bukan merupakan fields dari sebuah object karena local variables dan parameter variables hanya digunakan di dalam method. Istilah fields atau properties atau attributes biasanya hanya digunakan untuk dua jenis variabel awal yaitu Instance Variables dan Class variables. Jadi kalau kita menyebut fields atau properties atau attributes berarti yang kita maksud adalah Instance Variables dan Class variables yaitu variabel yang dikenali/dapat digunakan di seluruh bagian class tempat dimana variabel tersebut didefinisikan. Namun ke-empat jenis variabel di atas (Instance Variables, Class Variables, Local Variables, dan Parameter Variables) kesemuanya kita sebut secara umum sebagai variables (Mawlood-Yunis, 2022) (Oracle, 2022) .

```
class SepedaMotor{
        //field
        static String warna = “HITAM”;
        static String jenisTransmisi = “MANUAL”
```

```
        double kecepatan = 0;

        //method
        double speedUp(double increment){
                double velocity = kecepatan + increment;
                return velocity;
        }
}
```

Dari class SepedaMotor diatas dapat kita identifikasi empat jenis variabel yang ada di dalamnya:

1. Instance Variable: double kecepatan = 0;
2. Class Variable: static String warna = “HITAM”; dan static String jenisTransmisi = “MANUAL”;
3. Local Variable: double velocity
4. Parameter: double increment

Secara sederhana deklarasi sebuah variabel dapat dilakukan dengan menuliskan Tipe Data diikuti dengan Nama Variabelnya.

```
TipeData namaVariabel;
```

Ada beberapa hal terkait penamaan variabel yang ada di java sebagai berikut:

1. Nama variabel di java bersifat case-sensitive sehingga nama variabel dengan case berbeda (uppercase dan lowercase dianggap variabel yang berbeda pula meski jika dibaca textnya sama. Nama variabel JenisTransmisi, jenistransmisi, JENIS_TRANSMISI, dan jenisTransmisi keempatnya dianggap sebagai empat variabel berbeda karena penulisan casenya berbeda.
2. Nama variabel dapat terdiri dari deretan huruf dan angka dan symbol underscore (_) dan symbol $ namun kita disarankan untuk menggunakan nama variabel yang menggunakan huruf dan sebisa mungkin menghindari penggunaan symbol $. Nama variabel juga dapat diawali dengan symbol _ dan $ namun lebih disarankan untuk mengawali nama variabel dengan huruf. Nama variabel tidak boleh diawali dengan angka.
3. Nama variabel dibuat menggunakan huruf lowercase (huruf kecil) jika

nama variable terdiri dari dua kata atau lebih maka kata kedua dan seterusnya ditulis dengan diawali huruf uppercase (huruf kapital)

4. Untuk penamaan variabel yang berupa variabel final (konstanta) maka semua hurufnya adalah huruf kapital dan jika terdiri dari dua kata maka digunakan pemisah berupa symbol _ (underscore).
5. Disarankan untuk membuat nama variabel yang bermakna Ketika dibaca misalnya: alas, tinggi, panjang, luas dan menghindari penggunaan nama variabel a, t, p, l. Hal ini agar source code kita lebih bersih dan lebih mudah terbaca nantinya apalagi jika kita bekerja dalam sebuah tim programmer.
6. Kita tidak boleh menggunakan keyword java sebagai nama variabel (Oracle, 2022).

Berikut ini contoh penulisan nama variabel yang disarankan:

```
int cadence;
double speed;
int gear;
String namaHari;
int jenisTransmisi;
final double PI = 3.14;
final int MAX_SPEED = 245;
int color = 223;
double luasLingkaran = 356.231;
double kelilingPersegiPanjang;
```

Field-field di java akan mendapatkan nilai default saat dideklarasikan meskipun tidak diberi nilai inisial (nilai awal). Berikut ini nilai default field di java:

1. Tipe Boolean akan diinisialisasi dengan nilai default false.
2. Tipe primitif yang menangani angka (byte, short, int, long, float, dan double) akan diinisialisasi dengan nilai default = 0 atau 0.0.
3. Tipe char akan diinisialisasi dengan sebuah karakter space.
4. Tipe Class akan diinisialisasi dengan nilai default null (Mawlood-Yunis, 2022). Yang termasuk ke dalam tipe class ini adalah semua tipe non-primitif diantaranya: String, Array, ArrayList, LinkedList, BigInteger, BigDecimal, tipe wrapper (Byte, Short, Integer, Long, Float, Double, Boolean, dan Character), serta class yang dibuat oleh programmer. Variabel bertipe class ini sering kita sebut sebagai object. Disini kita punya satu definisi tambahan bahwa

objek adalah variabel yang tipe datanya bertipe class atau dengan kata lain class adalah tipe data untuk object.

Kita dapat melihat disini bahwa perbedaan variabel yang bertipe data primitif dan yang bertipe class adalah bahwa variabel yang bertipe data primitif dapat dibuat dan diinisialisasi tanpa menggunakan keyword new. Variabel dengan tipe data non-primitif akan mendapat nilai default = null.

Pada tipe data primitif kita mengenal bagian dari source code yang disebut dengan literal. Literal adalah source code yang merepresentasikan nilai tetap (fixed value). Literal ini dituliskan secara langsung ke source code bisanya untuk memberikan nilai pada variabel bertipe data primitif dan literal ini tidak membutuhkan proses komputasi. Literal ini mirip dengan simbol-simbol khusus yang ada di Regular Expression. Materi lebih jauh tentang literal dapat dibaca di situs Oracle berikut:
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

berikut ini beberapa contoh penggunaan literal untuk memberi nilai pada variabel bertipe data primitif:

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

angka 26 jika dalam decimal dapat ditulis sebagai berikut:

```
int decVal = 26;
```

angka 26 jika dalam hexadecimal dapat ditulis sebagai berikut:

```
int hexVal = 0x1a;
```

angka 26 jika dalam binary dapat ditulis sebagai berikut:

```
int binVal = 0b11010;
double d1 = 123.4;
```

nilai 123.4 jika ditulis dalam notasi saintifik dapat menggunakan literal berikut:

```
double d2 = 1.234e2;
```

atau jika menggunakan tipe float maka kita gunakan literal f berikut:

```
float f1 = 123.4f;
```

penggunaan symbol _ (underscore) yang disisipkan pada nilai bilangan bulat tidak akan mempengaruhi nilainya. Symbol ini dapat kita gunakan untuk menjadi pemisah ribuan, jutaan yang akan lebih memudahkan programmer dalam membaca nilainya.

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

TYPECASTING_____

Typecasting atau biasa disebut casting saja adalah sebuah cara untuk mengambil nilai dari sebuah variabel dan dimasukkan ke dalam variabel yang baru dengan nilai ekuivalen yang berbeda tipe datanya. Typecasting ini mirip dengan konversi nilai variabel yang tipe datanya berbeda. Type casting dilakukan menggunakan operator berupa symbol tanda kurung berisi tipe data tujuan. Misalnya kita akan melakukan typecasting dari integer ke double, maka dapat kita lakukan seperti berikut ini:

```
int x = 27;
double y = (double) x;
```

tanda kurung berisi tipe data tujuan casting kita letakkan di depan variabel atau nilai yang akan kita casting. Operasi casting ini akan memotong nilai integer dari nilai variabel decimal asal dan bukan melakukan pembulatan.

```
float x = 2.9f;
int y = (int) x;
System.out.println(y);
```

Operasi typecasting di atas akan menghasilkan nilai variabel y = 2 bukan 3 yang merupakan pembulatan ke atas dari nilai 2.9. sebaliknya bilangan bulat yang dicasting ke tipe decimal akan mendapatkan tambahan .0 di belakang angka bulatnya. Jadi typecasting tidak sama dengan pembulatan bilangan

> Diluar topik typecasting, untuk pembulatan bilangan kita dapat menggunakan method Math.ceil( ), Math.floor( ), Math.round( ), dll.

TYPE ASSIGNMENT

Pada Bahasa pemrograman java tidak semua variabel dengan tipe data (tipe data primitif) berbeda dapat di-assign ke tipe data lain. Untuk dapat di-assign ke tipe data lain, sebuah variabel atau nilai harus memiliki jangkauan tipe data (length) yang lebih kecil atau sama dengan tipe data tujuan (Mawlood-Yunis, 2022).

```
public class VariableAssignment {
        public static void main(String args[]) {
                int x = 2; double y;
                y = x; // correct (boleh)
                x = y; // mismatch type (tidak boleh dilakukan typecasting)
                x = (int) y; // explicit type casting
                byte b;
                short s;
                s = b; // correct
                b = s; // mismatch type
                s = x;// mismatch type
                b = x; // mismatch type
                x = b; // correct assignment
                x = s; // correct assignment
                long l;
                x = l; // mismatch type
                l = x;
                double d;
                float f;
                d = l;
```

```
                l = d; // mismatch type
                f = l;
                l = f; // mismatch type
                char c = 'a';
                char a = 97;
                System.out.println(a); // print character a
                int charvalue = (int) c;
                boolean bool1 = true;
                boolean bool2 = false;
                bool1 = 0; // type mismatch cannot convert from int to boolean
                bool2 = 1; // type mismatch cannot convert from int to boolean
                bool1 = (boolean) 1; // cannot cast from int to boolean
        }
}
```

STRING_____

Tipe String digunakan untuk merepresentasikan deretan karakter atau text yang sering diperlakukan layaknya tipe data primitif, namun sebenarnya String adalah sebuah class dan bukan tipe data primitif. String adalah tipe reference (non primitif) khusus yang telah disediakan oleh JDK. Sebagai tipe data, String dituliskan menggunakan huruf depan kapital: String dan bukan string. Sebagai sebuah class, di dalam String sudah tersedia method-method untuk melakukan operasi String seperti: compareTo(), charAt(index), contains(), equals(), length(), equalsIgnoreCase(), hashKode(), isBlank(), isEmpty(), split(regex), repeat( ), replace( ), trim( ), toCharArray( ), toLowerCase(), toUpperCase( ), valueOf( ) dan masih banyak lagi. Dokumentasi untuk tipe String dapat dilihat pada situs Dokumentasi JDK:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

STRING CONCATENATION_____

Di java kita dapat menggunakan operator plus (+) untuk menggabungkan (atau lebih tepatnya menyambung) dua String atau lebih membentuk String yang lebih Panjang (Mawlood-Yunis, 2022). Operasi penyambungan string ini disebut dengan concatenation. Dalam class String sebenarnya juga sudah disediakan method concat(str) yang kegunaanya sama dengan operator plus ini.

```
String str1 = “Informatika”;
String str2 = “Universitas Sulawesi Barat”;
String str3 = str1 + str2;
System.out.println(str3)
```

String dapat di-concatenate dengan angka dan akan menghasilkan String sebagai hasil akhir concatenation.

```
int number = 2022;
String tahun = “Tahun “;
System.out.println(tahun + number);
```

Kita dapat melakukan rantai operasi concatenation:

```
String data = “Al Fonso “+12.079+”De Alber Queue”+” Bara Baraya Papap”+” Papap Pung”+” Dimensi “+2004;
System.out.println(data);
```

atau kita juga dapat melakukan rantai operasi concatenation ini langsung di dalam method print

```
System.out.println(“Al Fonso “+12.079+”De Alber Queue”+” Bara Baraya Papap”+” Papap Pung”+” Dimensi “+2004);
```

Method concat(String str) juga dapat digunakan untuk menyambung String.

```
String str1 = "Kata Pertama";
String str2 = "Kata Perdua";
String str3 = str1.concat(str2);
```

Untuk stream data yang besar dan memerlukan banyak operasi concatenation kita disarankan untuk tidak melakukannya menggunakan tipe data String, sebaiknya kita gunakan tipe StringBuffer atau StringBuilder setelah semua operasi concatenation-nya selesai baru kita kembalikan ke tipe String menggunakan method .toString( ).

KONVERSI TIPE DATA PRIMITIF DARI DAN KE STRING _____

Konversi variabel dari tipe data primitif ke String dapat dilakukan menggunakan method valueOf() yang telah disediakan di class String. Ada 9 overloading method yang dapat digunakan:

1. valueOf(boolean b)
2. valueOf(char c)
3. valueOf(char[] data)
4. valueOf(char[] data, int offset, int count)
5. valueOf(double d)
6. valueOf(float f)
7. valueOf(int i)
8. valueOf(long l)
9. valueOf(Object obj)

```
double x = 27.891;
String str = String.valueOf(x);
```

Selain menggunakan method valueOf( ) kita juga dapat menggunakan operator concatenation untuk menkonversi tipe data primitif ke String yaitu dengan cara melakukan concatenasi nilai variabel bertipe data primitif dengan String “”.

```
double x = 27.891;
String str = “”+x;
atau
double x = 27.891;
String str = x+“”;
```

Selanjutnya, untuk melakukan konversi tipe data String ke tipe data primitif dapat dilakukan menggunakan class wrapper (class pembungkus) dari masing-masing tipe data primitif (Byte, Short, Integer, Long, Float, Double, dan Boolean). Contoh untuk mengkonversi String ke int dapat dilakukan dengan cara berikut:

```
String str = "12";
int value = Integer.parseInt(str);
```

untuk mengkonversi String ke double dapat dilakukan dengan cara berikut:

```
String str = "12.89";
double value = Double.parseDouble(str);
```

SPLIT STRING_____

Operasi split string digunakan untuk memecah sebuah string (split) menjadi array string berdasarkan pemisah (delimiter) berupa symbol tertentu. Symbol pemisah ini bisa berupa karakter, atau deretan karakter, symbol titik dua (:), simbol titik koma (;), dll. Pada contoh berikut kita melakukan split string menggunakan simbol spasi (\\s).

```
import java.util.Arrays;
public class Test{
	public static void main(String[] args) {
		String strA = "Jika Kamu tidak sanggup menahan lelahnya belajar
	maka kamu harus sanggup menahan perihnya kebodohan";
		String[] arrStrB = strA.split("\\s");
		for (String s : arrStrB) {
			System.out.println(s);
		}
	}
}
```

kita juga dapat melakukan operasi split string menggunakan regex yang lebih kompleks misalnya kita menggunakan beberapa string delimiter sekaligus:

```
import java.util.Arrays; public class Test{
	public static void main(String[] args) {
		String strA = "Jika Kamu;tidak sanggup:menahan lelahnya belajar
	maka kamu harus;sanggup menahan:perihnya-kebodohan";
		String[] arrStrB = strA.split("\\:|\\;|-|\\s");
		for (String s : arrStrB) {
			System.out.println(s);
		}
	}
}
```

MAIN METHOD_____

Method main merupakan salah satu method khusus di Java yang memiliki aturan penulisan khusus pula. Class yang memiliki method main biasa disebut pula dengan main class (bukan istilah resmi). Class yang memiliki method main akan dianggap sebagai program (aplikasi java), dalam artian class tersebut akan menjadi class yang dapat di-run. Method main akan menjadi method yang pertama dipanggil saat program dijalankan. Berikut ini beberapa struktur sederhana dari method main.

1. public static void main(String[ ] args){ }
2. public static void main(String... args){ }
3. static public void main(String... args){ }
4. static public void main(String[ ] args){ }

jika dilihat dari strukturnya maka kita ketahui bahwa method main ini adalah method yang diawali dengan modifier static dan public. Modifier static diberikan agar method main ini dapat digunakan/dipanggil tanpa harus diinstan classnya (class dimana method main ini berada), sedangkan modifier public berarti method main ini visibilitasnya public sehingga method main akan terlihat (dapat diakses) dari semua class meskipun dari package berbeda.

Selanjutnya masih dibagian struktur deklarasi, method main ini memiliki tipe output void dan tipe parameter input berupa array String (String[ ] args).

output : void
input : array String

Ingat, Array String yang digunakan disini bukan sekedar tipe String, tetapi juga merupakan array. Kalian masih ingat dengan struktur data array bukan???.
Dari dalam method main ini kita dapat melakukan operasi seperti method pada umumnya, hanya saja penulisan struktur dasarnya (modifier, input, output, dan penamaan method) yang harus mengikuti aturan penulisan khusus. Kita dapat membuat instan dari class lain di method main ini, memanggil field dan method yang ada di object hasil instan class, kita bahkan dapat membuat operasi layaknya pada pemrograman prosedural.

INPUT-OUTPUT_____

Ada beberapa cara berbeda yang dapat digunakan untuk melakukan operasi input dan output di Java diantaranya sebagai berikut:

1. Input dari keyboard menggunakan class java.util.Scanner
2. Input dari file menggunakan class java.util.Scanner
3. Menulis output di console window menggunakan System.out.println() dan System.out.print( )
4. Menulis output di console window menggunakan System.out.printf()
5. Menulis string ke file menggunakan java.io.FileWriter dan java.io.BufferedWriter
6. Membaca string dari file menggunakan java.io.FileReader dan java.io.BufferedReader

OPERATOR_____

Operator adalah symbol khusus yang digunakan untuk melakukan operasi. Berikut ini tabel operator-operator yang tersedia di Java:

Operator Precedence	
Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR (XOR)	^
bitwise inclusive OR	|
logical AND	&&

logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

CONDITIONAL_____

Ada beberapa variasi operasi kondisional yang dapat digunakan sesuai dengan kebutuhan diantaranya:

1. if
2. If-else if
3. If-else if-else
4. swith-case
5. ternary

LOOP_____

Ada beberapa cara yang dapat kita lakukan untuk membuat LOOP (perulangan) diantaranya menggunakan:

1. for
2. for each
3. while
4. do-while

METHOD_____

Method adalah blok kode yang hanya dijalankan ketika dipanggil. Kita dapat memasukkan input ke dalam sebuah method melalui variabel yang disebut dengan parameter. Method berisi operasi-operasi atau aksi. Method juga sering disebut dengan function. Method dibuat dengan tujuan untuk penggunaan kode berulang (to reuse kode), dimana kita cukup sekali saja membuat kode untuk digunakan berulang kali.

Method dibuat di dalam sebuah class. Method didefinisikan menggunakan nama method yang diikuti tanda kurung () berisi parameter method kemudian diikuti dengan tanda kurung kurawal { } berisi blok statement atau operasi yang akan dikerjakan oleh method. Untuk method yang memiliki nilai kembalian (return value) maka return value ini harus dideklarasikan di bagian akhir method sebelum tanda kurung kurawal di tutup. Nama lain untuk method yang memiliki

return value ini disebut dengan method non-void.

pada kode program latihan sebelumnya sebenarnya kita telah berlatih dengan menggunakan beberapa method yang sudah disediakan di JDK. Seperti method println() yang berada di class System. Kita juga telah sering menggunakan method main(). Berikut ini kita akan berlatih membuat method kita sendiri. Tapi sebelum itu, perhatikan program berikut ini.

```
public class Program019 {
	public static void main(String[] args) {
		double nilaiA = 12.654;
		double nilaiB = 86.217;
		double nilaiC = (nilaiA*nilaiA + nilaiB*nilaiB) / 2*nilaiA*nilaiB;
		System.out.println(nilaiC);
		double nilaiD = 34.6;
		double nilaiE = 82.12;
		double nilaiF = (nilaiD*nilaiD + nilaiE*nilaiE) / 2*nilaiD*nilaiE;
		System.out.println(nilaiF);
		double nilaiG = 764.31;
		double nilaiH = 5.95;
		double nilaiK= (nilaiG*nilaiG + nilaiH*nilaiH) / 2*nilaiG*nilaiH;
		System.out.println(nilaiK);
	}
}
```

Perhatikan kode di atas. Terlihat bahwa nilaiC, nilaiF, dan nilaiK diperoleh menggunakan formula yang sama. Formula ini dapat kita sederhanakan penulisannya ke dalam method dan programnya menjadi seperti berikut:

```
public class Program020 {
	public static void main(String[] args) {
		double nilaiA = 12.654;
		double nilaiB = 86.217;
		double nilaiC = hitung(nilaiB, nilaiB);
		System.out.println(nilaiC);
```

```
                double nilaiD = 34.6;
                double nilaiE = 82.12;
                double nilaiF = hitung(nilaiD, nilaiE);
                System.out.println(nilaiF);
                double nilaiG = 764.31;
                double nilaiH = 5.95;
                double nilaiK = hitung(nilaiG, nilaiH);
                System.out.println(nilaiK);
        }
        public static double hitung(double a, double b){
                double c = (a*a + b*b) / 2*a*b;
                return c;
        }
}
```

Disini kita membuat method hitung( ) untuk mendelegasikan operasi

```
public static double hitung(double a, double b){
        double c = (a*a + b*b) / 2*a*b;
        return c;
}
```

Method hitung() ini kemudian dipanggil melalui method main()
double nilaiC = hitung(nilaiB, nilaiB);

REGULAR EXPRESSION_____

Ekspresi Reguler (Regular Expression) atau biasa disingkat dengan Regex adalah pola pencarian (search pattern) untuk string. Regex adalah deretan karakter yang membentuk pola pencarian. Pola pencarian string ini bisa berupa apa saja, karakter sederhana, maupun ekspresi kompleks yang terdiri dari karakter khusus yang digunakan untuk mendeskripsikan pola (pattern).

Regex dapat digunakan untuk melakukan pencarian (searching), edit, dan manipulasi text/string. Proses ini disebut dengan penerapan Regex pada text/string.

Regex diterapkan pada text dari kiri ke kanan. Sekali sebuah source karakter pada text digunakan maka karakter tersebut tidak dapat digunakan lagi. Sebagai contoh penerapan Regex "aba" akan cocok (match) dengan text "abababababa" hanya dua kali (aba_aba__).

Untuk bekerja dengan Regex di Java dapat kita lakukan dengan meng-import package java.util.regex yang telah menyediakan class: Pattern, Matcher dan PatternSystaxException.

Dokumentasi untuk Regex di Java dapat ditemukan disini:

1. https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/regex/package-summary.html
2. https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/regex/Pattern.html
3. https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/regex/Matcher.html
4. https://www.vogella.com/tutorials/JavaRegularExpressions/article.html
5. https://www.w3schools.com/java/java_regex.asp
6. https://www.javatpoint.com/java-regex
7. https://regex101.com/r/cS6hE8/1/codegen?language=java

Metacharacters

Metacharacter adalah karakter dengan makna khusus di regular expression:

Metacharacter	Deskripsi
|	Digunakan untuk menemukan sebuah pola yang sesuai dengan salah satu pilihan pola yang dipisahkan oleh simbol pipe (|) seperti contoh: ayam|kucing|sapi
.	Digunakan untuk menemukan sebuah karakter (karakter apapun itu)
^	Digunakan untuk menemukan string di awal baris yang dimulai dengan kata atau karakter yang ditulis setelah simbol ^. Contohnya "^halo"
$	Digunakan untuk menemukan string diakhir baris yang diakhiri dengan kata atau simbol yang ditulis sebelum simbol $. Contohnya: "world$"

\d	Digunakan untuk menemukan digit atau angka
\s	Digunakan untuk menemukan spasi (whitespace character)
\b	Digunakan untuk menemukan boundary kata

Quantifiers

Quantifiers digunakan untuk mendefinisikan kuantitas atau jumlah.

Quantifiers	Deskripsi
n+	Digunakan untuk menemukan string yang mengandung paling sedikit satu karakter n
n*	Digunakan untuk menemukan string yang mengandung karakter n ataupun boleh tidak mengandung karakter n sama sekali
n?	Digunakan untuk menemukan string yang mengandung n atau tidak sama sekali
n{x}	Digunakan untuk menemukan string n yang berderet sebanyak x (tepat sebanyak x)
n{x,y}	Digunakan untuk menemukan string n dengan pengulangan paling sedikit sebanyak x tapi tidak lebih dari y
n{x,}	Digunakan untuk menemukan string n yang diulang paling sedikit sebanyak x

Pada contoh program berikut ini kita akan mengekstrak text dari dalam text:

```
{
        "Kode MK": "USB0913",
        "Matakuliah": "Wawasan Sosial Budaya",
        "SKS": 3,
        "Nilai": "A"
}
```

Kita akan mengekstrak text yang berada diantara tanda kurung kurawal menjadi pasangan key dan value:

```
Kode MK = USB0913
Matakuliah = Wawasan Sosial Budaya
SKS = 3
Nilai = A
```

Berikut ini kode program javanya:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Regex010 {
	public static void main(String[] args) {
		String text = """
			{
				"Kode MK": "USB0913",
				"Matakuliah": "Wawasan Sosial Budaya",
				"SKS": 3,
				"Nilai": "A"
			}
		""";
		Pattern p =
		Pattern.compile("\"([^\"]+)\":\\s*\"*([^\"^\\,]+)\"*\\,*");
		Matcher m = p.matcher(text);
		while (m.find()) {
			String key = m.group(1).trim();
			System.out.print(key);
			String value = m.group(2).trim();
			System.out.print(" = "+value); System.out.println();
		}
	}
}
```

Cobalah untuk menemukan makna dari Regex:
"\"([^\"]+)\":\\s*\"*([^\"^\\,]+)\"*\\,*";

MATERI LANJUTAN

Java Language Basic

URL: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html
OOP Zetcode: https://zetcode.com/java/oop/

Selamat Belajar, Jangan Lupa Berdoa dan Bersyukur. Terimakasih...

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. Tugas Praktikum

1. Pilih 10 object di dunia nyata, lakukan identifikasi states dan behaviors dari objects tersebut. Lalu buatlah class untuk masing-masing object tersebut. Kemudian buat sebuah class Main (class yang dilengkapi dengan method main( )) dan dari method main tersebut buat minimal 10 object (instan class) dari masing-masing class yang telah dibuat serta panggil method-method yang tersedia di dalamnya.
2. Buatlah class untuk membuat object geometri: segitiga, persegi panjang, persegi, lingkaran, jajargenjang, trapesium, belah ketupat dan layang-layang. Setiap class harus dilengkapi dengan method untuk men-set nilai variabel variabel yang ada di dalam class. selain itu setiap class juga harus memiliki method hitungLuas( ), hitungKeliling( ), dan printState( ) untuk menampilkan nilai variabel-variabel terkini.
3. Buatlah program untuk menghitung luas dan keliling bangun datar: persegi, persegi panjang, segitiga, jajar genjang, trapesium, layang-layang, belah ketupat dan lingkaran. Sesuaikan variabel input yang dibutuhkan. input variabel dilakukan melalui console window.
4. Buatlah program untuk menghitung debit air dengan input consol terdiri dari nilai luas permukaan dan kecepatan aliran air.
5. Buatlah program untuk menampilkan sejumlah n baris String "INFORMATIKA" dengan nilai variabel n diinput melalui console window.
6. Buatlah program untuk menentukan nilai MIN dan MAX jika inputnya berupa dua buah bilangan a dan b yang diinput melalui console window.
7. Buatlah program untuk menentukan nilai MIN dan MAX jika inputnya terdiri dari 5 variabel a, b, c, d, dan e yang diinput melalui console.
8. Buatlah program transkrip nilai untuk menginput nilai dan menampilkan data transkrip (lakukan identifikasi terhadap transkrip kalian masing-

masing di SIAKAD Unsulbar) lalu demokan hasil Aplikasi yang dibuat menggunakan data transkrip kalian. Desain dan susun program sekreatif mungkin. Lengkapi fitur-fiturnya dengan operasi CRUD berbasis Array/ArrayList/LinkedList/Vector (tanpa menggunakan file maupun database server/DBMS)

9. Buatlah sebuah class kemudian di dalam class tersebut buatlah masing masing 100 variable untuk empat jenis variable (instance variables, class variables, local variables, dan parameters) jadi total ada 400 variables yang dibuat dengan berbagai varian modifier visibilitas.
10. Buatlah sebuah class Main kemudian di dalam method main( ) di kelas tersebut buatlah penggunaan operasi-operasi String: charAt(int index), compareTo( ), compareToIgnoreCase( ), concat( ), contains( ), copyValueOf( ), equals( ), equalsIgnoreCase( ), format( ), hashKode( ), indexOf( ), isBlank( ), length( ), lines( ), matches( ), repeat( ), replace( ), replaceAll( ), replaceFirst( ), split( ), substring(int beginIndex), substring(int beginIndex, int endIndex), toCharArray( ), toLowerCase( ), toString( ), toUpperCase( ), trim( ), valueOf( ). Beri penjelasan untuk penggunaan masing-masing method, jika terdapat lebih dari satu varian parameter, maka buat semua variannya. Misalnya untuk method valueOf( ) ada valueOf(double d), valueOf(long l), dll. Maka gunakan semua method valueOf( ) tersebut.
11. Buatlah program Java untuk menampilkan bilangan bulat diantara dua bilangan bulat m dan n secara ascending (tidak termasuk bilangan m dan n, nilai m tidak harus lebih kecil dari n). Input program adalah bilangan bulat m dan n. outputnya adalah barisan bilangan yang dipisahkan menggunakan <,>.

    Contoh IO sebagai berikut:

    ```
    17
    2
    3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
    ```

12. Buatlah program Java untuk menampilkan penjumlahan bilangan bulat diantara dua bilangan bulat m dan n secara ascending (tidak termasuk bilangan m dan n, nilai m tidak harus lebih kecil dari n). Input program adalah bilangan bulat m dan n. outputnya adalah barisan bilangan yang dipisahkan menggunakan simbol “+” kemudian sebelum hasil

penjumlahan ditampilkan terlebih dahulu dituliskan symbol "=" baru kemudian hasil penjumlahannya.
Contoh Input Output sebagai berikut:

```
7
2
3 + 4 + 5 + 6 = 18
```

13. Buatlah sebuah program kriptografi sederhana dengan scenario berikut. Program akan menerima sebuah kalimat (terdiri dari beberapa kata) kemudian anda diminta untuk memanipulasi semua huruf vocal yang ada pada kalimat tersebut dengan format:
    a menjadi ava
    i menjadi ivi
    u menjadi uvu
    e menjadi eve
    o menjadi ovo
    berikut ini contoh Input-Output programnya:

```
selamat belajar java
sevelavamavat bevelavajavar javavava
```

```
informatika unsulbar
ivinfovormavativikava uvunsuvulbavar
```

14. Buatlah program untuk mengidentifikasi Kata terpanjang yang ada pada sebuah kalimat. Program akan menerima input berupa kalimat yang terdiri dari beberapa kata. Setiap kata akan dipisahkan oleh spasi. Selanjutnya program akan menampilkan nilai panjang dari kata terpanjang diikuti dengan kata yang dimaksud. Panjang kata berarti banyaknya karakter yang ada pada kata tersebut. Misalnya kata: berusaha panjangnya adalah 8 karena ada delapan karakter di dalam kata berusaha. Berikut ini contoh IO programnya:

```
Sedang berusaha menjadi lebih baik lagi bekerja keras dan pantang menyerah bertanggungjawab percaya diri 16 bertanggungjawab
```

15. Buatlah Program Kriptografi sederhana untuk Algoritma Caesar Cipher dan Vigenere Cipher (masing-masing disertai encrypt dan decrypt). Jelaskan langkah-langkah manual dari kedua algoritma tersebut, susun algoritmanya, buat flowchartnya menggunakan flowgorithm, kemudian buatlah source code untuk mengimplementasikan kedua algoritma Kriptografi tersebut.
16. Buatlah program Java yang dilengkapi dengan input Scanner untuk membuat semua pola (pattern) berikut (silahkan download lalu buka menggunakan notepad/notepad++ untuk melihat hasilnya):

    https://gist.github.com/FOSSALGO/c7a02f44b09fe96654da4f027e2b84af

PRAKTIKUM 2: CLASS AND OBJECT

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator Penilaian

1. Mahasiswa mampu menjelaskan konsep class dan object di OOP.
2. Mahasiswa mampu membuat class.
3. Mahasiswa mampu membuat object dari class.
4. Mahasiswa mampu membuat dan mengetahui penggunaan Constructor.
5. Mahasiswa mampu membuat dan mengetahui penggunaan method getter dan setter.
6. Mahasiswa mampu membuat inner class.
7. Mahasiswa mampu menggunakan keyword instanceof.
8. Mahasiswa mampu menjelaskan struktur dari method.
9. Mahasiswa mampu menjelaskan tentang method void dan non-void.
10. Mahasiswa mampu membuat method void dan non-void.
11. Mahasiswa mampu membuat method dengan output menggunakan berbagai tipe data primitif dan non-primitif.
12. Mahasiswa mampu membuat method dengan berbagai tipe output dan berbagai tipe parameter (input).
13. Mahasiswa mampu membuat method static.
14. Mahasiswa mampu membuat method main( ).
15. Mahasiswa mampu membuat OVERLOADING METHOD.
16. Mahasiswa mampu membuat OVERRIDING METHOD.
17. Mahasiswa mampu membuat OVERLOADING CONSTRUCTOR.
18. Mahasiswa mampu membuat method dengan menggunakan modifier: public, private, protected, static, final, abstract.

B. Tugas Pendahuluan

1. Jelaskan konsep dasar tentang class di dalam Pemrograman Berorientasi Objek!
2. Jelaskan konsep dasar tentang object di dalam Pemrograman Berorientasi Objek!

3. Jelaskan bagaimana cara membuat object dari class!
4. Apa yang dimaksud dengan membuat instan dari class (menginstan class)?
5. Berapa maksimal jumlah object yang dapat kita buat dari sebuah class?
6. Jelaskan apa yang dimaksud dengan constructor, apa kegunaan constructor, bagaimana cara membuat constructor, bagaimana cara menggunakan constructor, dan apa yang dimaksud dengan overloading constructor?
7. Apa yang dimaksud dengan constructor default?
8. Tuliskan Langkah-langkah untuk membuat sebuah class
9. Tuliskan Langkah-langkah untuk membuat method
10. Tuliskan Langkah-langkah untuk membuat variabel
11. Gambarkan dan jelaskan struktur dari sebuah method
12. Apa kegunaan dari operator instanceof?
13. Apa perbedaan method void dan non-void? Berikan masing masing 3 contoh untuk jenis method tersebut!
14. Jelaskan kegunaan dari modifier static? Berikan masing-masing 4 contoh penggunaan modifier static pada variabel dan method.
15. Jelaskan perbedaan antara Instance Variables, Class Variables, Local Variables, dan Parameter Variables.
16. Dapatkah kita membuat class di dalam sebuah class? jelaskan cara membuatnya jika hal tersebut dapat dilakukan.
17. Apakah sebuah class adalah termasuk tipe data di java? Jelaskan alasan anda.
18. Jelaskan tentang class: java.lang.Object. apa kegunaan dari class tersebut?, method apa saja yang tersedia di dalamnya? Dan apakah kita perlu meng import class java.lang.Object agar method-method yang ada di dalamnya dapat kita gunakan?
19. Jelaskan kesamaan dan perbedaan OVERLOADING dan OVERRIDING?
20. Jelaskan apa saja kegunaan keyword return di dalam sebuah method?

C. Ringkasan Materi

Pada praktikum ini kita akan membahas tentang dasar-dasar pemrograman Java dan Pemrograman Berorientasi Objek yaitu Class dan Object

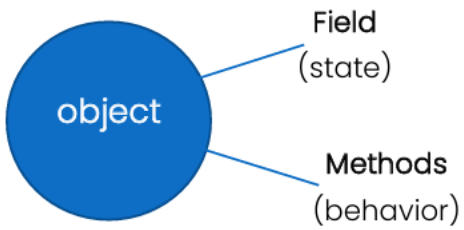
OBJECT_____

Object adalah kunci untuk memahami teknologi object-oriented. Sekarang

lihatlah disekeliling kita, dan kita akan menemukan banyak contoh object di dunia nyata: kucing, meja, kursi, buku, mobil, tas, smartphone, sepeda, motor, baju, lemari, kotak pensil, termos kopi, laptop, ayam, jonga, sapi, serangga, gelas, senter, dll.

Object di dunia nyata memiliki dua karakteristik yang semua object memilikinya yaitu mereka semua memiliki **state** dan **behavior**. **State** untuk kucing ada: nama, warna, varietas dan **behavior** untuk kucing diantaranya ada: menggerakkan ekor, melompat, berjalan, berlari, menerjang, menghempas batu karang, dll. Demikian pula pada object motor ada **state** (diameter roda, jenis transmisi, kapasitas bahan bakar, dll) dan terdapat **behavior** (pengereman, tingkatkan kecepatan, belok kiri, belok kanan, dll). Identifikasi **state** dan **behavior** dari object di dunia nyata (real world) adalah cara terbaik sebagai langkah awal memahami terminologi dalam **object oriented programming** (Oracle, 2022). Sekarang coba buat daftar object yang ada di sekitar kemudian lakukan identifikasi **state** dan **behavior** yang dimiliki oleh object tersebut.

Konsep **object** di dunia nyata ini diadopsi ke dalam konsep **object** dalam pemrograman (software) dimana istilah **state** pada real-object kemudian kita sebut sebagai **fields** (fields atau variables atau properties atau data) sedangkan istilah **behavior** pada real-object kemudian kita sebut sebagai **methods** (methods atau functions atau procedure atau operations).



Secara sederhana kita dapat katakan bahwa istilah object di dalam pemrograman adalah sebuah teknik pembungkusan data dan function (function untuk mengubah/mengolah nilai pada data tersebut). Untuk memudahkan dan menyederhanakan penggunaan istilah, data ini adakalanya kita sebut sebagai field, variable, property, attributes, dalam java kita akan lebih sering menggunakan istilah field dan variable. Sedangkan istilah function adakalanya kita sebut sebagai method, operation, procedure (meskipun sedikit berbeda dengan function), dalam java kita akan lebih sering menggunakan istilah method untuk menggantikan istilah operation, function dan procedure.

Field = variable, property, attribute, data, (pada object nyata = state) Method = function, procedure, operation, (pada object nyata = behavior)

Kita dapat mengatur visibilitas variable dan method yang ada pada object. Teknik ini dikenal dengan istilah encapsulation (pembungkusan). Encapsulation dapat diberikan kepada field, method dan class. Encapsulation variable dan method ini dilakukan menggunakan package dan modifier visibilitas (access modifier): public, private, protected, dan default.

CLASS_____

Class adalah cetak biru (blueprint) untuk membuat object. Karena class adalah sebuah blueprint atau prototype untuk membuat object maka dari sebuah class kita dapat membuat banyak object. Sebuah class akan memodelkan sebuah object di dunia nyata. Kita mengambil contoh objek sepeda motor. Ada banyak objek sepeda motor dilihat dari warna, jenis transmisi, kecepatan, dll. Dari berbagai objek sepeda motor ini, kita dapat melakukan generalisasi untuk membentuk sebuah blueprint Sepeda Motor. Generalisasi model sepeda motor ini kita sebut dengan class.

Sederhananya kita dapat menyebutkan bahwa class adalah:

- **Class** adalah **blueprint** untuk membuat object
- **Class** adalah **template** atau cetakan untuk membuat object
- Dari sebuah class kita dapat membuat **banyak** object
- Object dari sebuah class **dibuat menggunakan** keyword **new**
- Object yang dibuat dari sebuah class biasa juga disebut **hasil instan** dari class
- Istilah **menginstan class** berarti sama dengan **membuat object** dari class
- Class juga merupakan **tipe data** (struktur data) untuk membuat object

Siklus hidup sebuah class diawali dari **object** di dunia nyata (object from real world) → kemudian dibuatkan source code class (software) → kemudian diinstan menjadi object (software).

Telah kita bahas pada praktikum sebelumnya bahwa object di dunia nyata memiliki **state** dan **behavior**. Sedangkan object di dalam program (software) memiliki **field** dan **method**. Untuk membuat class dari object di dunia nyata dapat

kita lakukan dengan terlebih dahulu melakukan identifikasi (abstraksi) terhadap **state** dan **behavior** yang dimiliki oleh **object** di dunia nyata ini. Setelah kita membuat classnya, maka dari sebuah class tersebut kita dapat membuat banyak object (object yang dimaksud di bagian ini adalah object **software** bukan real world object).

Kita mulai dengan sebuah real world object: Bangun Datar **Kotak**.

Selanjutnya kita lakukan identifikasi **state** dan **behavior** yang ada pada object kotak:

State : warna, panjang, lebar, keliling, dan luas

Behavior : hitung luas dan hitung keliling, set Panjang, set lebar, set warna.

Dari hasil identifikasi tersebut kita dapat membuat sebuah bagan untuk mengelompokkan **state** dan **behavior** real world object **Kotak** sebagai berikut:

Kotak
warna panjang lebar keliling luas
setWarna( ) setPanjang( ) setLebar( ) hitungLuas( ) hitungKeliling( )

Pada kompartemen pertama bagan di atas kita menuliskan nama real world object yang sekaligus nanti akan menjadi nama classnya. Pada kompartemen yang kedua kita menuliskan semua **state** hasil identifikasi kita, dan pada

kompartemen yang terakhir kita menuliskan semua behavior hasil identifikasi. Di kompartemen yang terakhir ini nama behaviornya diikuti dengan tanda kurung buka tutup dimana nanti behavior ini akan kita transformasi menjadi method. Setelah melakukan identifikasi selanjutnya kita dapat dengan mudah membuat class dari real world object **Kotak** sebagai berikut:

Kita buat fields-nya terlebih dahulu:

```
public class Kotak {
	String warna;
	double panjang;
	double lebar;
	double keliling;
	double luas;
}
```

Kemudian kita tambahkan method-methodnya menjadi seperti berikut ini:

```
public class Kotak {
	String warna;
	double panjang;
	double lebar;
	double keliling;
	double luas;

	void setWarna(String w) {

	}

	void setPanjang(double p) {

	}

	void setLebar(double l) {

	}
```

```
        void hitungLuas() {

        }

        void hitungKeliling() {

        }
}
```

Selanjutnya kita lengkapi operasi di dalam method-method yang telah dibuat menjadi seperti berikut ini:

```
public class Kotak {
        String warna;
        double panjang;
        double lebar;
        double keliling;
        double luas;

        void setWarna(String w) {
                warna = w;
        }

        void setPanjang(double p) {
                panjang = p;
        }

        void setLebar(double l) {
                lebar = l;
        }

        void hitungLuas() {
                luas = panjang * lebar;
        }

        void hitungKeliling() {
```

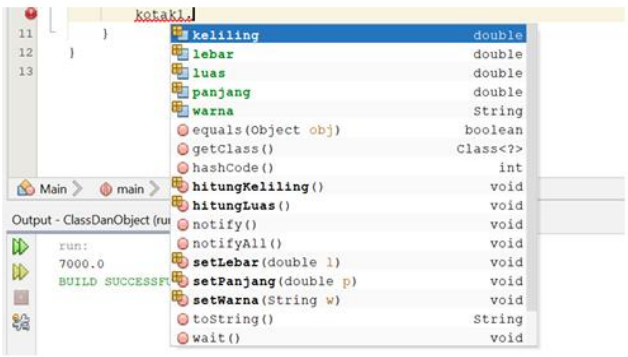
```
            keliling = panjang + lebar + panjang + lebar;
        }
}
```

Kita akan Kembali melakukan penyesuaian untuk class **Kotak** ini namun sekarang kita akan coba terlebih dahulu untuk menginstan **class Kotak** untuk membuat beberapa object dari class **Kotak** ini.

Buat sebuah **Main Class**. Istilah Main Class kita gunakan untuk merujuk pada class yang di dalamnya terdapat method main( ). Kemudian di dalam method main( ) kita buat beberapa instan dari class **Kotak** menggunakan keyword **new**. Ingat keyword **new** adalah keyword yang digunakan untuk **membuat object** dari **class**.

```
public class Main {
        public static void main(String[] args) {
            Kotak kotak1 = new Kotak();
            Kotak kotak2 = new Kotak();
            Kotak kotak3 = new Kotak();
            Kotak kotak4 = new Kotak();
        }
}
```

Jika kita menggunakan IDE Netbeans menuliskan nama salah satu object kotak misalnya **kotak1** setelah objectnya diinstan kemudian kita menuliskan tanda . (atau key [ctrl][space] setelah tanda .) maka akan terlihat semua **field** dan **method** public yang tersedia di class Kotak.



Anda mungkin akan bertanya: kita belum membuat method: equals(), getClass(), hashKode(), notify(), notifyAll(), toString() dan wait() tapi mengapa terlihat juga method-method itu? Ini adalah salah satu kelebihan dan fitur dari object oriented programming yang disebut **inheritance**. Dalam java semua class secara default akan menjadi **subclass** atau turunan dari class **Object**. Semua method dan field public dan protected dapat terlihat dari subclassnya. **Field**, **method**, dan **interclass** biasa juga diistilahkan dengan member class. semua member class public dan protected dapat terlihat dan digunakan oleh subclass-nya.

> ada sebuah class di package java.lang yang Bernama **Object**. Filenya: **Object.java** dan semua class di package java.lang tidak perlu di-import saat kita membuat class. package java.lang ini akan otomatis di-import juga. Saya berharap anda tidak bingung disini bahwa ada class yang Bernama **Object** di package java.lang atau lengkapnya ditulis **java.lang.Object**;

Berikut ini method-method yang tersedia di class java.lang.**Object**;

- public String toString()
- public int hashKode()
- public boolean equals (Object obj)
- public Class getClass()
- protected Object clone()
- public void notify()
- public void notifyAll()
- public void wait()
- public void wait(long timeout)
- public void wait(long timeout, int nanos)

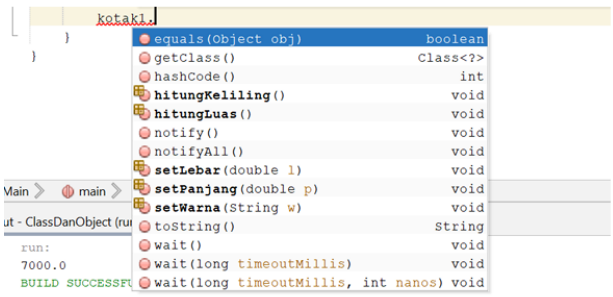
fitur **inheritance** di java memungkinkan kita untuk menggunakan field dan method yang ada di **superclass** untuk digunakan maupun didefinisikan ulang (overriding) di **subclass**. Kita akan Kembali memperdalam pengetahuan tentang **inheritance** ini di praktikum berikutnya InsyaAllah.

Kembali ke class **Kotak**. Di class Kotak kita dapat mengatur agar field: warna, panjang, lebar, keliling dan luas tidak dapat terlihat atau diakses langsung dari **object** hasil instannya yaitu dengan menggunakan mekanisme pembungkusan

(**encapsulation**) menggunakan modifier visibilitas **private**. Tambahkan modifier private ke setiap field yang telah kita buat sebelumnya menjadi:

```
private String warna;
private double panjang;
private double lebar;
private double keliling;
private double luas;
```

sekarang dari object **kotak1** kita tidak dapat lagi melihat field-field tersebut.



Selanjutnya bagaimana jika kita hendak mengakses field tersebut misalnya kita ingin merubah nilai warna, panjang, dan lebar? Kita dapat membuat method **setter** untuk field warna, panjang, dan lebar di class **Kotak**. Method **getter** dan **setter** ini digunakan untuk mengubah dan mengambil nilai field/variabel yang ada di dalam object khususnya field yang visibilitasnya private.

```
void setWarna(String w) {
        warna = w;
}

void setPanjang(double p) {
        panjang = p;
}

void setLebar(double l) {
        lebar = l;
}
```

atau selain menggunakan method **setter** kita juga dapat menetapkan nilai awal untuk field-field ini melalui method **constructor**. Method **constructor** (biasa juga disebut **constructor** saja tanpa kata depan method) merupakan method khusus juga di java. sebelumnya kita telah menggunakan method khusus di java yaitu method main( )). Method khusus ini tentunya memiliki cara penulisan yang khusus pula. Method **constructor** ditulis dengan nama yang sama dengan nama class tempat **constructor** tersebut dibuat. Method **constructor** ditulis tanpa disertai dengan deklarasi output method. Dalam satu class kita diperbolehkan memiliki beberapa method **constructor** (**overloading constructor**). Jika kita tidak membuat **constructor** maka untuk menginstan class kita dapat menggunakan **constructor default**. **Constructor** adalah method yang pertama dipanggil saat sebuah object dibuat dari sebuah class. Saat membuat object method **constructor** ini kita tuliskan (dipanggil) tepat di sebelah kanan setelah kita menuliskan keyword **new**. Jadi sebenarnya tadi kita telah menggunakan **constructor default** pada saat membuat object kotak:

```
Kotak kotak1 = new Kotak( );
Kotak kotak2 = new Kotak( );
Kotak kotak3 = new Kotak( );
Kotak kotak4 = new Kotak( );
```

Bagian yang dicetak tebal di atas adalah method **constructor** atau lebih tepatnya **default constructor**. Perhatikan bahwa nama method constructor **sama** dengan nama class-nya yaitu class **Kotak**.

Constructor dapat digunakan untuk menetapkan nilai awal (nilai inisial) untuk **fields** yang ada di dalam class, dan untuk sebuah class kita dapat membuat beberapa varian **constructor** sesuai kebutuhan dan mengikuti aturan **overloading method**. Disini saya membuat tiga buah constructor untuk class Kotak sebagai berikut:

```
public Kotak() {

}
```

```
public Kotak(String w) {
	warna = w;
}

public Kotak(String w, double p, double l) {
	warna = w;
	panjang = p;
	lebar = l;
}
```

Karena sekarang kita sudah memiliki beberapa varian **constructor** jadi kita juga dapat membuat object kotak dengan beberapa cara dengan memanggil **constructor** sembari **menginisialisasi** field di object kotak yang kita buat.

```
public class Main {
	public static void main(String[] args) {
		Kotak kotak1 = new Kotak();
		Kotak kotak2 = new Kotak("Biru");
		Kotak kotak3 = new Kotak("Hitam", 12, 7);
		Kotak kotak4 = new Kotak();
	}
}
```

Jika tadi kita telah membuat method **setter** untuk **menetapkan nilai field** pada object selanjutnya kita juga dapat membuat method **getter** untuk mengambil nilai field dari **object**. Berbeda dengan method **setter** yang biasanya memiliki output void dan input nya berisi **parameter**, pada method **getter** biasanya parameternya kosong (tidak menggunakan parameter, tetapi outputnya berisi nilai field yang di **get**. Kita akan menambahkan method **getter** untuk field luas dan keliling di class Kotak. Method getter yang kita buat ini akan dilengkapi juga dengan operasi untuk menghitung luas dan keliling jadi kita melakukan **getter** sembari menghitung luas dan keliling kotak.

```
public double getKeliling() {
	keliling = panjang + lebar + panjang + lebar;
	return keliling;
```

```
}

public double getLuas() {
        luas = panjang * lebar; return luas;
}
```

Sekarang class Kotak kita menjadi:

```
public class Kotak {
        private String warna;
        private double panjang;
        private double lebar;
        private double keliling;
        private double luas;

        public Kotak() {

        }

        public Kotak(String w) {
                warna = w;
        }

        public Kotak(String w, double p, double l) {
                warna = w;
                panjang = p;
                lebar = l;
        }

        public void setWarna(String w) {
                warna = w;
        }

        public void setPanjang(double p) {
                panjang = p;
        }
```

```
    public void setLebar(double l) {
        lebar = l;
    }

    public double getKeliling() {
        keliling = panjang + lebar + panjang + lebar;
        return keliling;
    }

    public double getLuas() {
        luas = panjang * lebar; return luas;
    }
}
```

Meskipun nama variabel harus unik, namun di Java kita diperbolehkan untuk membuat nama variabel local yang sama dengan nama fieldnya dan untuk membedakannya kita dapat menggunakan keyword **this**. Keyword **this** disini digunakan untuk membedakan **fields** dan **local variables**. Field akan diberi awalan **this** untuk membedakannya dengan **local variable**. Sehingga kita dapat membuat method **constructor** dan **setter** kita menjadi seperti berikut ini:

```
public Kotak(String warna) {
    this.warna = warna;
}

public Kotak(String warna, double panjang, double lebar) {
    this.warna = warna;
    this.panjang = panjang;
    this.lebar = lebar;
}

public Kotak(double panjang, double lebar) {
    this.panjang = panjang;
    this.lebar = lebar;
```

```
}

public void setWarna(String warna) {
        this.warna = warna;
}

public void setPanjang(double panjang) {
        this.panjang = panjang;
}

public void setLebar(double lebar) {
        this.lebar = lebar;
}
```

Perhatikan penulisan: **this.panjang** = panjang;
yang tercetak tebal: **this.panjang** adalah **field** (instance variable) sedangkan yang tercetak tipis di sebelah kanan symbol = yaitu: Panjang adalah **local variable** yang berasal dari **parameter**.

Selanjutnya kita dapat melakukan sedikit overriding terhadap method **toString()** yang berasal dari class java.lang.Object; method **toString( )** ini akan kita gunakan untuk menampilkan informasi state object.

```
public String toString() {
        String info = "KOTAK: warna= " + warna
                + ", panjang= " + panjang
                + ", lebar= " + lebar
                + ", luas= " + luas
                + ", keliling= " + keliling;
        return info;
}
```

dan sekarang class Kotak kita secara lengkap dituliskan sebagai berikut:

```
public class Kotak {
        private String warna;
```

```
private double panjang;
private double lebar;
private double keliling;
private double luas;

public Kotak() {

}

public Kotak(String warna) {
        this.warna = warna;
}

public Kotak(String warna, double panjang, double lebar) {
        this.warna = warna;
        this.panjang = panjang;
        this.lebar = lebar;
}

public Kotak(double panjang, double lebar) {
        this.panjang = panjang;
        this.lebar = lebar;
}

public void setWarna(String warna) {
        this.warna = warna;
}

public void setPanjang(double panjang) {
        this.panjang = panjang;
}

public void setLebar(double lebar) {
        this.lebar = lebar;
}
```

```
    public double getKeliling() {
        keliling = panjang + lebar + panjang + lebar;
        return keliling;
    }

    public double getLuas() {
        luas = panjang * lebar; return luas;
    }

    public String toString() {
        String info = "KOTAK: warna= " + warna
            + ", panjang= " + panjang
            + ", lebar= " + lebar
            + ", luas= " + luas
            + ", keliling= " + keliling;
        return info;
    }
}
```

semua **field public** dan **method public** di class Kotak dapat dipanggil dari object hasil instannya dengan menuliskan simbol (.) setelah nama object hasil instannya.

Kita dapat mensimulasikan Kembali pembuatan object di class Main beserta pemanggilan method-method yang telah kita buat di class Kotak:

```
public class Main {
    public static void main(String[] args) {
        Kotak kotak1 = new Kotak();
        Kotak kotak2 = new Kotak("Biru");
        Kotak kotak3 = new Kotak("Hitam", 12, 7);
        Kotak kotak4 = new Kotak();
        kotak2.setPanjang(12);
        kotak2.setPanjang(7);
        double luas = kotak2.getLuas();
        System.out.println("Luas Kotak_2= "+luas);
```

```
                kotak3.getLuas();
                kotak3.getKeliling();
                String info = kotak3.toString();
                System.out.println(info);
                System.out.println(kotak1.toString());
                System.out.println(kotak4.toString());
        }
}
```

Pada java kita juga dapat membuat class di dalam sebuah class. Jadi sebuah class tidak hanya dapat berisi **field** dan **method** tetapi juga dapat berisi **class**. Class di dalam class ini disebut dengan **inner class**.

Member of class: **fields**, **methods**, dan **inner class**

Pada java juga kita dapat mengidentifikasi (mendeteksi) sebuah **object** dibuat atau diinstan dari class apa. Operasi ini dilakukan menggunakan operator **instanceof**.

Pada java juga tersedia keyword **static** yang dapat digunakan untuk membuat **static field** (class variable) dan **static method**. **static field** dan **static method** ini nantinya akan dapat dipanggil langsung dari method di class lain tanpa harus dibuat object-nya (tanpa harus diinstan). Best practice untuk praktikum ini: praktikan diminta untuk mempelajari penggunaan **inner class**, operator **instanceof** dan keyword **static** secara mandiri.

Member class yang diberi modifier static dapat digunakan (dipanggil) dari class lain tanpa harus dibuat object-nya terlebih dahulu (tanpa menggunakan keyword **new**) langsung dengan menuliskan nama classnya diikuti dengan tanda titik (.) berikutnya nama member class yang akan dipanggil.

Keyword **new** digunakan untuk membuat object-object dari class. istilah membuat object ini sering juga disebut dengan membuat instan class. Komunikasi data pada pemrograman berorientasi objek dilakukan melalui object bukan secara langsung berkomunikasi melalui file. Selain dapat dibuat

objectnya, dari sebuah class kita juga dapat membuat turunannya. Istilah **penurunan** class atau **pewarisan** class dalam pemrograman berorientasi object ini disebut dengan **inheritance**. Sebuah class hasil **pewarisan** class juga dapat memiliki beberapa sifat atau karakteristik yang berbeda antara class **superclass** dan **subclass**. Sebuah class juga dapat memiliki method-method yang Namanya sama tetapi parameter inputnya berbeda-beda. Beberapa subclass yang berasal dari **superclass** yang sama dapat juga memiliki perilaku atau operasi berbeda-beda. Kemampuan membuat varian method dan karakteristik class dan object di dalam pemrograman berorientasi object ini disebut dengan istilah **polymorphism**. Jadi secara sederhana konsep klasik dari Pemrograman Berorientasi Objek (Object Oriented Programming) terdiri dari tiga pilar(fitur) utama yaitu: **encapsulation**, **Inheritance**, dan **Polymorphism**. Pada praktikum ini tiga fitur Object Oriented Programming (OOP) akan kita kerjakan menggunakan bahasa pemrograman Java.

METHOD_____

Pada materi praktikum kali ini kita akan kembali membahas tentang method dengan lebih dalam. Kita dapat mengilustrasikan method sebagai sebuah program sederhana (meskipun tak selalu sederhana seperti isyarat yang tak sempat disampaikan awan kepada hujan yang membuatnya tiada). Dalam bahasa pemrograman lain biasanya ada istilah: **procedure**, **function**, dan **operation**. Nah di dalam java semua itu disebut **method**, tinggal nanti bagaimana kita membedakan bentuk penulisan outputnya.

Kalau menurut definisi klasik sebuah program itu minimal terdiri dari tiga bagian: **INPUT**, **PROSES**, **OUTPUT** (definisi klasik dari sebuah program). Demikian pula pada method tiga bagian ini harus ada dan di deklarasikan (kecuali method **constructor** yang tidak memiliki deklarasi output).

Jadi sebuah **method** itu punya: **INPUT**, **PROSES**, **OUTPUT**

Secara Struktur

INPUT: kita sebut sebagai **parameter variabel** (biasa disebut **parameter** saja) dari method yang ditulis di dalam kurung setelah nama method (di sebelah kanan nama method).

PROSES: adalah bagian dari **body** method yang ditulis di dalam **block kurung kurawal**.

OUTPUT: dideklarasikan di sebelah kiri nama method dan nilainya ditetapkan menggunakan *keyword* **return** di bagian akhir blok method.

OUTPUT namaMethod (INPUT) {PROSES}

Perhatikan method berikut ini:

```
double hitungLuasKotak(double panjang, double lebar){
        luas = panjang * lebar;
        return luas;
}
```

Pada method **hitungLuasKotak** di atas terdapat dua input yaitu parameter panjang dan lebar yang masing-masing bertipe double. Output dari method **hitungLuasKotak** adalah bertipe double dan nilai outputnya kita set dari nilai variabel luas menggunakan operasi **return** luas. Proses yang ada di dalam method **hitungLuasKotak** adalah operasi perhitungan luas = panjang * lebar.

INPUT: panjang, lebar
PROSES: luas = panjang x lebar
OUTPUT: luas

Sampai disini seharusnya kita sudah paham tentang struktur method dan mampu membedakan bagian-bagian **INPUT**, **PROSES**, dan **OUTPUT**-nya.

METHOD VOID DAN NON-VOID_____
Istilah method void dan non-void ini bukan istilah resmi dari Java. Istilah ini biasa digunakan oleh para programmer yang sebelumnya lebih sering menggunakan bahasa pemrograman terstruktur dimana pada pemrograman terstruktur terdapat istilah **procedure** dan **function** yang telah disebutkan sebelumnya bahwa di java keduanya (**procedure** dan **function**) disebut sebagai method. Bedanya adalah kalau di procedure itu tidak memiliki output maka method-nya

kita deklarasikan sebagai **void** sedangkan untuk **function** itu punya output maka method-nya nanti kita deklarasikan dengan tipe data selain void (non-void) boleh kita gunakan int, double, String, NamaClass, ArrayList, tipe wrapper dan sebagainya selain tipe void.

Procedure = Method Void
Function = Method Non-Void

Sampai disini sebenarnya kita telah banyak membuat method void dan non-void. Salah satu method void (**procedure**) yang paling sering kita buat adalah method main( ). Perhatikan penulisan lengkap dari method main( )

```
public static void main(String[ ] args){ }
```

dari kelas Kotak kita di materi sebelumnya:

```
public class Kotak {
	private String warna;
	private double panjang;
	private double lebar;
	private double keliling;
	private double luas;

	public Kotak() {

	}

	public Kotak(String warna) {
		this.warna = warna;
	}

	public Kotak(String warna, double panjang, double lebar) {
		this.warna = warna;
		this.panjang = panjang;
		this.lebar = lebar;
	}
```

```
    public Kotak(double panjang, double lebar) {
        this.panjang = panjang;
        this.lebar = lebar;
    }

    public void setWarna(String warna) {
        this.warna = warna;
    }
    public void setPanjang(double panjang) {
        this.panjang = panjang;
    }

    public void setLebar(double lebar) {
        this.lebar = lebar;
    }

    public double getKeliling() {
        keliling = panjang + lebar + panjang + lebar;
        return keliling;
    }

    public double getLuas() {
        luas = panjang * lebar;
        Return luas;
    }

    public String toString() {
        String info = "KOTAK: warna= " + warna
        + ", panjang= " + panjang
        + ", lebar= " + lebar
        + ", luas= " + luas
        + ", keliling= " + keliling;
        return info;
    }
}
```

Umumnya (meski tak selalu) method setter adalah merupakan method void:

```
public void setWarna(String warna) {
        this.warna = warna;
}

public void setPanjang(double panjang) {
        this.panjang = panjang;
}

public void setLebar(double lebar) {
        this.lebar = lebar;
}
```

dan method getter adalah method non-void:

```
public double getKeliling() {
        keliling = panjang + lebar + panjang + lebar;
        return keliling;
}

public double getLuas() {
        luas = panjang * lebar;
        return luas;
}
```

Method toString( ) termasuk method non-void karena outputnya bukan void juga:

```
public String toString() {
        String info = "KOTAK: warna= " + warna
        + ", panjang= " + panjang
        + ", lebar= " + lebar
        + ", luas= " + luas
        + ", keliling= " + keliling;
        return info;
}
```

CONSTRUCTOR DAN OVERLOADING CONSTUCTOR_____

Telah dibahas dalam praktikum sebelumnya bahwa **constructor** adalah method yang dipanggil setelah kita menuliskan keyword **new** saat membuat object. **Constructor** juga merupakan method khusus tanpa deklarasi output dan nama method **constructor** harus sama dengan nama **class**. Jadi class kita bernama Kotak maka nama method **constructor**-nya adalah **Kotak**( ). Di dalam sebuah kelas kita dapat membuat lebih dari satu constructor. Kemampuan untuk membuat method dengan nama yang sama tetapi parameter/urutan tipe parameternya berbeda ini disebut dengan istilah overloading. Jadi pada saat kita melakukan **overloading** terhadap suatu method maka tipe dan urutan parameter dari method yang di **overload** tersebut harus berbeda agar tidak terjadi ambiguitas pada saat pemanggilan method. **Overloading** tidak hanya dapat dilakukan pada method **constructor**. Method-method lain juga dapat kita overloading asalkan mengikuti aturan penulisan parameter yang telah ditetapkan dan sekaligus menghindari ambiguitas.

Berikut ini contoh penulisan overloading method yang ambigu dan harus dihindari

```
public Kotak(String warna) {
        this.warna = warna;
}
//dan
public Kotak(String color) {
        this.warna = color;
}
```

Kedua method di atas, meskipun dibuat dengan nama parameter berbeda, tetapi tipe data parameternya sama maka hanya dianggap sebagai method yang sama, apalagi di bagian parameter kedua method tersebut tidak ada parameter lain yang dapat kita gunakan untuk membuat variasi **urutan** parameter.

Method berikut ini juga ambigu:

```
public Kotak(double panjang, double lebar) {
        this.panjang = panjang;
```

```
        this.lebar = lebar;
}
//dan
public Kotak(double lebar, double panjang) {
        this.lebar = lebar;
        this.panjang = panjang;
}
```

Kedua overloading method di atas juga ambigu karena meskipun parameternya terdiri dari dua parameter yang sudah diubah urutannya tetapi keduanya bertipe data sama maka tetap dianggap sama dan menyebabkan terjadi ambiguitas.

Berikut ini contoh penulisan parameter untuk overloading method yang diperbolehkan:

```
public void myMethod(int a){ }
public void myMethod(double a){ }
public void myMethod(int a, int b){ }
public void myMethod(double a, int b){ }
public void myMethod(double a, double b){ }
public void myMethod(int a, double b){ }
public void myMethod(int a, int b, int c){ }
public void myMethod(int a, int b, String s){ }
public void myMethod(String s, int a, int b){ }
public void myMethod(int a, String s, int b){ }
```

dan berikut ini contoh penulisan overloading method yang ambigu

```
public void myMethod(int a){ }
public void myMethod(int b){ }
```

```
public void myMethod(double a){ }
public void myMethod(double b){ }
```

```
public void myMethod(int a, int b, int c){ }
```

```
public void myMethod(int b, int c, int a){ }
public void myMethod(int c, inta, int b){ }
```

```
public void myMethod(int a, int b, String s){ }
public void myMethod(int b, int a, String s){ }
```

Berikut ini contoh overloading constructor yang telah kita lakukan sebelumnya:

```
public Kotak() {

}
public Kotak(String warna) {
        this.warna = warna;
}
public Kotak(String warna, double panjang, double lebar) {
        this.warna = warna;
        this.panjang = panjang;
        this.lebar = lebar;
}

public Kotak(double panjang, double lebar) {
        this.panjang = panjang;
        this.lebar = lebar;
}
```

METHOD GETTER DAN SETTER_____

Method getter dan setter tidak memiliki aturan penulisan khusus di java hanya saja umumnya method setter merupakan method void sedangkan method getter merupakan method non-void. Method getter digunakan untuk mengambil nilai yang terbungkus di dalam class saat kita tidak memiliki akses langsung ke field/variabel yang akan kita ambil nilainya (mungkin karena terbungkus oleh visibilitas private). Demikian pula method setter kita gunakan untuk menge-set nilai dari field yang tidak dapat kita akses secara langsung dari luar class atau dari luar object hasil instan classnya. Berikut ini method getter dan setter yang telah kita buat di class **Kotak** sebelumnya:

```
public void setWarna(String warna) {
        this.warna = warna;
}

public void setPanjang(double panjang) {
        this.panjang = panjang;
}

public void setLebar(double lebar) {
        this.lebar = lebar;
}

public double getKeliling() {
        keliling = panjang + lebar + panjang + lebar;
        return keliling;
}

public double getLuas() {
        Luas = panjang * lebar;
        return luas;
}
```

Sama seperti pada **field** pada **method** juga kita dapat menambahkan modifier: **static**, **final**, **abstract** serta modifier visibilitas **public**, **private**, **protected**. Modifier visibilitas ini akan kita bahas lebih jauh di bagian praktikum **encapsulation**.

Modifier **static** akan memungkinkan kita untuk menggunakan sebuah **method** atau **field** tanpa terlebih dahulu harus menginstan class tempat dimana **method** dan **field** itu berada. Pada saat membuat method static, kita tidak dapat menggunakan keyword **this** di dalam method-nya.

Member class yang diberi modifier **static** akan memberikan kita akses langsung ke member dari class tersebut tanpa harus membuat objek dari class-nya jika akses langsung ini kita lakukan dari luar class tempat member tersebut berada maka ini sedikit bertentangan dengan prinsip encapsulation yang ada di Java, apalagi yang kita akses tersebut adalah variabel class. Dalam prinsip dasar

encapsulation adalah bahwa semua field/variabel disarankan untuk menggunakan modifier private atau protected atau default saja, dan ketika kita mempersiapkan perubahan untuk nilai field tersebut, kita lakukan melalui method public agar perubahan nilai field atau variabel tersebut tetap terkendali dan tidak sembarang method yang dapat mengubah nilainya dari luar class seenaknya tanpa mempertimbangkan kemungkinan pengaruhnya terhadap operasi lain di dalam internal class tempat field atau variabel tersebut berada. Seharusnya nilai dari field hanya boleh diubah melalui method yang ada di dalam class pembungkus field tersebut (baik itu method setter, method constructor, maupun method lain yang ada di dalam class yang sama).

Dalam **encapsulation**: jangan berikan akses data dari luar class/object kecuali yang dibutuhkan saja atau pastikan keamanan untuk semua data(variabel) dan method yang akan diberi akses dari luar. Aman yang dimaksud disini adalah tingkat dependensi atau kebergantungan operasi atau variabel lain terhadap variabel yang akan di beri akses dari luar.

Contoh class yang memiliki method static:

```
public class BangunDatarLingkaran {
        static double luas;
        static double keliling;

        static double hitungLuas(double radius){
                luas = Math.PI*radius*radius;
                Return luas;
        }

        static double hitungKeliling(double radius){
                keliling = 2.0 * Math.PI * radius;
                return keliling;
        }
}
```

Method hitungLuas( ) dan hitungKeliling( ) pada class **BangunDatarLingkaran** di atas merupakan method static, sehingga kita dapat menggunakan kedua method ini bahkan dari class lain tanpa harus membuat instannya terlebih dahulu.

```
public class Main {
	public static void main(String[] args) {
		double luasLingkaran = BangunDatarLingkaran.hitungLuas(49);
		System.out.println("Luas Lingkaran = "+luasLingkaran);
		Double kelilingLingkaran =
		BangunDatarLingkaran.hitungKeliling(49);
		System.out.println("Keliling Lingkaran = "+kelilingLingkaran);
	}
}
```

Pada method main( ) di atas kita langsung memanggil method hitungLuas secara langsung dari class-nya dengan cara menuliskan nama class-nya (class BangunDatarLingkaran) diikuti dengan tanda titik kemudian diikuti dengan nama method dan parameternya:

```
BangunDatarLingkaran.hitungLuas(49);
```

Perhatikan bahwa disini kita sama sekali tidak membuat object-nya menggunakan keyword **new**. Sama halnya dengan **static method**, pada **static field** (class variable) kita dapat menggunakan **static field** tanpa harus membuat instannya terlebih dahulu.

OVERRIDING METHOD_____

Overriding adalah suatu fitur dalam bahasa pemrograman Java yang memungkinkan kita untuk membuat method dengan nama yang sama dimana salah satu method ini berada di **subclass** sementara method lainnya berada di **superclass**. Overriding adalah proses mendefinisikan ulang suatu method yang dimiliki oleh **superclass**. Pendefinisian ulang ini dilakukan di dalam **subclass**. Jadi method yang didefinisikan ulang tersebut sebenarnya sudah ada di superclass dan tanpa didefinisikan ulang kitapun dapat langsung menggunakannya. Mendefinisikan ulang method dari superclass ini dilakukan jika kita membutuhkan perilaku atau operasi yang berbeda dari method yang ada di superclass. Yang dimaksud mendefinisikan ulang disini adalah merubah operasi atau membuat operasi yang baru di dalam method tetapi output, nama method, dan parameter

input methodnya sama dengan yang ada di class super.

Pada praktikum sebelumnya sebenarnya kita telah melakukan **overriding method** yaitu pada method **toString( )**; method toString( ) sebenarnya adalah method yang ada pada class: java.lang.Object; dan class java.lang.Object; ini adalah class yang secara default merupakan super class dari semua class yang walaupun tidak di deklarasikan semua class yang kita buat akan menjadi subclass dari class **java.lang.Objec**t ini. Nah di dalam class **java.lang.Object** ini ada method toString() lengkapnya: **public String toString()**. Tanpa melakukan **override** kita sudah bisa langsung menggunakan/memanggil method toString() ini. Tetapi secara default method toString() dari class **java.lang.Object** ini hanya akan menampilkan nilai kode has dari object BangunDatarLingkaran@723279cf

Agar method toString( ) ini dapat melakukan operasi lain, jadi kita bermaksud agar nantinya dapat memanggil method toString( ) ini dari object tetapi hasilnya adalah String info state atau nilai-nilai variabel yang ada di dalam object maka saya melakukan modifikasi proses terhadap method toString( ) ini dan modifikasi inilah yang kita istilahkan dengan **overriding**. Singkatnya **overriding** adalah membuat method yang namanya sama dengan method yang ada di **superclass** tetapi proses di dalamnya berbeda.

```
public String toString() {
        String info = "KOTAK: warna= " + warna
        + ", panjang= " + panjang
        + ", lebar= " + lebar
        + ", luas= " + luas
        + ", keliling= " + keliling;
        return info;
}
```

Jadi kalau pada **OVERLOADING** kesamaan nama method terjadi di dalam satu class yang sama sedangkan pada **OVERRIDING** kesamaan nama methodnya terjadi antara **superclass** dan **subclass**. Perhatikan contoh source code berikut:

Kita membuat subuah **superclass** bernama **SuperClassBangunDatar** yang nantinya akan kita turunkan ke **subclass** bernama **SubClassPersegi:**

```
public class SuperClassBangunDatar {
	void cetakInfo(){
		System.out.println("BANGUN DATAR");
	}
}
```

Selanjutnya kita buat subclass bernama **SubClassPersegi** untuk melakukan penurunan sebuah class (**inheritance**) kita menggunakan keyword **extends**:

```
public class SubClassPersegi extends SuperClassBangunDatar {
	@Override void cetakInfo() {
		System.out.println("PERSEGI");
	}
}
```

Kemudian kita buat method main di class Main berikut:

```
public class Main {
	public static void main(String[] args) {
		System.out.println("Object_1 ----------------------------");
		SubClassPersegi objek1 = new SubClassPersegi();
		objek1.cetakInfo();
		System.out.println("\nObject_2 ----------------------------");
		SuperClassBangunDatar objek2 = new SuperClassBangunDatar();
		objek2.cetakInfo();
		System.out.println("\nObject_3 ----------------------------");
		SuperClassBangunDatar objek3 = new SubClassPersegi();
		objek3.cetakInfo();
	}
}
```

Perhatikan bahwa objek1, objek2, dan objek3 dibuat dengan cara berbeda-beda tetapi ketiga object tersebut merupakan instan dari superclass SuperClassBangunDatar hal ini dapat kita uji menggunakan operator binary **instanceof**. Dan method cetakInfo( ) telah mengalami **overriding**. Silahkan didiskusikan di dalam kelas tentang perilaku hasil **overriding** ini.

METHOD REKURSIF_____

Method rekursif adalah sebuah method (function) yang di dalam body methodnya ada pemanggilan terhadap dirinya sendiri (method yang memanggil dirinya sendiri). Berikut ini kita akan membuat method factorial yang di dalam method factorial itu juga kita memanggil method factorial:

```
public class Main {
	static int factorial(int n) {
		if (n == 0) {
			return 1;
		} else {
			return (n * factorial(n - 1));
		}
	}
	public static void main(String[] args) {
		int n = 12;
		int faktorial = factorial(n);
		System.out.println("Faktorial(" + n + ") = " + faktorial);
	}
}
```

Selamat Belajar, Jangan Lupa Berdoa dan Bersyukur. Terimakasih...

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. Tugas Praktikum

1. Temukan 20 object di dunia nyata, lakukan identifikasi terhadap state dan behaviornya, lalu buat bagan classnya, kemudian susunlah kode java untuk class yang akan menjadi blueprint dari 20 object tersebut.
2. Buat 20 class Main (nama filenya tidak harus persis Main.java), lalu dari method main class tersebut buat masing-masing 30 object dari 20 class yang telah dibuat pada tugas nomor 1 (jadi nanti kita akan punya 600 object hasil instan). Selanjutnya lakukan pemanggilan method-method yang ada di object hasil instan.
3. Buatlah sebuah class yang di dalamnya terdapat beberapa inner class

(lengkap dengan field dan method di dalamnya), kemudian buatlah class Main yang di dalamnya terdapat method main berisi operasi instan object dari masing-masing inner class yang telah dibuat. Lakukan pemanggilan method yang ada di object hasil instan dari inner class.

4. Buatlah sebuah class yang di dalamnya terdapat static field dan static method lalu kemudian buatlah sebuah class main untuk menggunakan/memanggil static field dan static method yang telah dibuat tanpa melakukan operasi instan class (tanpa menggunkan keyword new). Sampai disini apa yang telah anda pelajari tentang static? Tuliskan jawaban anda tentang static ini di bagian akhir jawaban soal nomor 4.
5. Tulis kembali class Kotak yang kita buat di Teori Pendukung Praktikum ini. Buatlah sebuah ArrayList bertipe Generik dari class Kotak ini. Tambahkan/masukkan beberapa object Kotak ke dalam ArrrayList (object object kotak yang telah dibuat akan dimasukkan ke dalam object arrayList). Jadi nanti object arrayList akan berisi beberapa object kotak. Kemudian lakukan pemanggilan method yang ada di object kotak tetapi pemanggilan tersebut dilakukan dengan cara memanggil object arrayList terlebih dahulu (tidak dipanggil secara langsung dari object kotak-nya). Yang dimaksud pemanggilan method secara langsung dari object kotak itu seperti ini: kotak1.setPanjang(12); bukan seperti ini yang harus dilakukan, tetapi seharusnya arrayList.get(8).setPanjang(12);
6. Buatlah sebuah class Main kemudian di dalam class tersebut buatlah sebuah method main( ) dan sebuah method recursif non-static untuk menampilkan bilangan fibonacci ke-n. kemudian panggil method recursif tersebut dari dalam method main yang telah kita buat.
7. Buatlah sebanyak 10 superclass yang nantinya siap diturunkan ke subclass. Trik untuk membuat superclass adalah dengan melakukan generalisasi, misalnya untuk class-class PersegiPanjang, Persegi, Lingkaran, Trapesium, Segitiga kita dapat mebuat sebuah generalisasi yaitu BangunDatar. Maka class BangunDatar ini nanti bisa menjadi superclass untuk class-class PersegiPanjang, Persegi, Lingkaran, Trapesium, dan Segitiga.
8. Untuk masing-masing superclass pada tugas nomor 7, buatlah minimal 2 subclassnya.
9. Di setiap subclass pada nomor 8 buatlah constructor, overloading constructor, overloading, overriding, method getter dan setter.
10. Buat sebuah method class Main berisi method main( ) untuk membuat

instan dari semua subclass yang telah dibuat pada nomor 8.

11. Buatlah sebuah class bernama ImageProcessing. Di dalam class ImageProcessing tersebut buatlah sebuah field bertipe array int tiga dimensi untuk menyimpan nilai pixel R,G, dan B. tambahkan pula sebuah method imgToGray( ) di dalam class ImageProcessing output dari method imgToGray( ) adalah array bertipe int dua dimensi yang akan berisi hasil grayscale dari image RGB. Lengkapi class ImageProcessing dengan constructor, method toString( ) serta method getter dan setter. Selanjutnya buatlah sebuah method main( ) di class berbeda, kemudian panggil semua member class yang ada di class ImageProcessing. Gunakan data dummy untuk mensimulasikan data RGB.
12. Buatlah sebuah class bernama SpaceCraft.

SpaceCraft
- xPos : int - yPos :
+ up( ) : void + down( ) : void + left( ) : void + right( ) : void

Class SpaceCraft memiliki dua field dengan modifier private (ditandai dengan simbol "-" di depan nama field pada diagram class). selanjutnya class SpaceCraft tersebut memiliki empat method dengan modifier public (ditandai dengan simbol "+" di depan nama field pada diagram class). variabel xPos dan yPos digunakan untuk menyimpan posisi SpaceCraft pada ruang 2D diagram cartesian. Method up( ) akan berisi operasi untuk mengurangi nilai yPos sebesar 1 satuan atau ditulis dengan: yPos = yPos – 1; demikian pula untuk method-method lainnya:

a. up( ) berisi operasi yPos = yPos -1;
b. down( ) berisi operasi yPos = yPos +1;
c. left( ) berisi operasi xPos = xPos -1;
d. right( ) berisi operasi xPos = xPos +1;

PRAKTIKUM 3: ENCAPSULATION, INHERITANCE AND POLYMORPHISM

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator Penilaian

1. Mahasiswa mampu menjelaskan konsep Encapsulation.
2. Mahasiswa mampu menjelaskan perbedaan Visibilitas: default, public, private, protected.
3. Mahasiswa mampu membuat class menggunakan modifier: public, private, protected.
4. Mahasiswa mampu membuat variabel menggunakan modifier: default, public, private, protected.
5. Mahasiswa mampu membuat method menggunakan modifier: default, public, private, protected 6. Mahasiswa mampu membuat inner class menggunakan modifier: default, public, private, protected.
6. Mahasiswa mampu menjelaskan konsep Inheritance (Superclass(Parent), Subclass(Child), abstract class, final class).
7. Mahasiswa mampu menjelaskan perbedaan Menurunkan sebuah class dan Membuat Instance dari sebuah class (membuat object).
8. Mahasiswa mampu membuat super class.
9. Mahasiswa mampu menggunakan keyword this dan keyword super.
10. Mahasiswa mampu membuat class menggunakan keyword abstract.
11. Mahasiswa mampu membuat class menggunakan keyword final.
12. Mahasiswa mampu membuat Subclass menggunakan keyword extends.
13. Mahasiswa mampu membuat Subclass menggunakan keyword implements.
14. Mahasiswa mampu membuat Interface.
15. Mahasiswa mampu membuat sub-interface dari beberapa super interface
16. Mahasiswa mampu Mampu menjelaskan konsep polymorphism dalam pemrograman berorientasi objek.

17. Mahasiswa mampu menggunakan method OVERLOADING dan OVERRIDING untuk melakukan Polymorphism.

B. Tugas Pendahuluan

1. Jelaskan konsep dasar tentang encapsulation!
2. Buatlah tabel modifier Level Akses terhadap member class (field, method, inner class) dari class yang sama (class itu sendiri), class lain di dalam satu package, dari subclass, dan world (dari class di luar package). Selanjutnya beri penjelasan tabel yang telah dibuat.
3. Tuliskan modifier visibilitas apa saya yang dapat digunakan untuk membuat class.
4. Tuliskan modifier visibilitas apa saya yang dapat digunakan untuk membuat inner class.
5. Tuliskan modifier visibilitas apa saya yang dapat digunakan untuk membuat method?
6. Tuliskan modifier visibilitas apa saya yang dapat digunakan untuk membuat field.
7. Tuliskan modifier visibilitas apa saya yang dapat digunakan untuk membuat constructor.
8. jelaskan tujuan encapsulation dalam pemrograman berorientasi objek.
9. Jelaskan konsep dasar tentang inheritance.
10. Jelaskan perbedaan antara menurunkan sebuah class dan membuat object dari sebuah class.
11. Jelaskan perbedaan penggunaan keyword abstract dan final.
12. Jelaskan perbedaan penggunaan keyword this dan super dalam kaitannya dengan inheritance di Java.
13. Jelaskan konsep dasar tentang interface dalam pemrograman berorientasi objek di Java, apa tujuan dari pembuatan interface, serta jelaskan perbedaan antara interface dan super class di Java.
14. Jelaskan kesamaan dan perbedaan antara Interface dan Class Abstract di Java.
15. Jelaskan tentang penggunaan keyword extends di Java.
16. Jelaskan tentang penggunaan keyword implements di Java.
17. Jelaskan cara menurunkan sebuah class di Java, berikan penjelasan disertai contoh kode programnya.
18. Jelaskan cara menurunkan sebuah interface di Java, berikan penjelasan disertai contoh kode programnya.

19. Apakah sebuah class di Java diperbolehkan meng-ekstends lebih dari satu super class? Jelaskan disertai contoh kode program Java.
20. Apakah sebuah class di Java diperbolehkan meng-implements lebih dari satu interface? Jelaskan disertai contoh kode program Java.
21. Apakah sebuah interface diperbolehkan untuk meng-ekstends lebih dari satu super interface? Jelaskan disertai contoh kode program Java.
22. Apa saja member dari abstract class?
23. Apa saja member dari interface?
24. Apa persamaan dan perbedaan antara Interface, GUI, dan CLI?
25. Jelaskan tentang Overloading Method di Java.
26. Jelaskan tentang Overriding Method di Java.
27. Jika terdapat dua method yang sama (nama methodnya sama, tipe parameter inputnya sama dan tipe outputnya sama) saat melakukan overriding method maka method manakah yang akan dipanggil saat kita memanggil method tersebut dari object yang dibuat dari subclassnya. Apakah yang terpanggil adalah method dari subclass atau superclassnya?. Jelaskan disertai dengan contoh kode program.
28. Buatlah sebuah superclass abstract bernama class BangunDatar yang berisi dua field yaitu: protected double keliling dan protected double luas

```
public abstract class BangunDatar {
        protected double keliling;
        protected double luas;
}
```

Selanjutnya buatlah sebuah interface Geometri berikut

```
public interface Geometri {
        final double PI = 3.14;
        public void hitungKeliling();
        public void hitungLuas();
        public double getKeliling();
        public double getLuas();
}
```

Tugas anda adalah membuat 9 buah class: **PersegiPanjang**, **Persegi**, **SegiTiga**, **Trapesium**, **JajarGenjang**, **Lingkaran**, **LayangLayang**, dan **BelahKetupat**. 8 class yang anda buat ini harus meng-extends abstract class BangunDatar dan meng-implements interface Geometri. Di setiap

class yang anda buat anda diperbolehkan untuk menambahkan field sesuai kebutuhan tetapi filed keliling dan luas harus diambil dari super class BangunDatar (tidak diperbolehkan membuat field keliling dan luas sendiri). Anda diperbolehkan menambahkan method constructor dan method untuk inisialisasi variabel lainnya, tetapi method getLuas( ) dan getKeliling( ) tidak boleh diubah operasinya dan harus ditulis seperti berikut:

```
@Override
public double getKeliling() {
        return keliling;
}

@Override
public double getLuas() {
        return luas;
}
```

Selanjutnya, karena formula untuk luas dan keliling dari 9 class bangun datar tersebut berbeda-beda, maka anda harus melakukan overriding dan memodifikasi operasi pada method hitungKeliling() dan hitungLuas().

C. Ringkasan Materi

ENCAPSULATION_____

Kita telah menyinggung dan sedikit membahas tentang encapsulation pada praktikum sebelumnya. Encapsulation sering pula disebut dengan Data Hiding dimana kita mengatur akses private untuk field atau data dan hanya memberikan akses public melalui method tertentu dan untuk mengakses data tertentu saja (Samoylov, 2022).

Encapsulation (enkapsulasi) merupakan salah satu pilar Object Oriented Programming. Encapsulation adalah suatu cara untuk mengatur atau membatasi akses terhadap member class dari luar class. **Member** class ini dapat berupa: **field**, **method** maupun **inner class**. Akses terhadap member class bergantung pada acces modifier atau modifier visibilitas yang diberikan pada member tersebut. Ada tiga macam modifier visibilitas yang dapat digunakan yaitu: **public**, **private**, **protected**. Sebenarnya ada modifier default untuk semua member yaitu ketika kita tidak menambahkan modifier visibilitas ke sebuah member berarti visibilitasnya default.

Enkapsulasi memberi jaminan pada member class agar tidak diganggu oleh programmer lain kecuali member class yang disediakan untuknya. Dengan demikian program akan menjadi lebih bersih (clean) dari gangguan maupun kesalahan (error), termasuk kesalahan dalam penggunaan object. Selain itu, dengan pengaturan akses seperti ini maka programmer pembuat class akan lebih mudah melakukan perbaikan terhadap internal class tanpa mengganggu implementasi program yang telah dan terlanjur sudah dilakukan oleh client maupun programmer lain (Suarga, 2009).

Pada bahasa pemrograman java disediakan dua macam modifier ada yang merupakan **Access Modifier** dan ada pula **Non-Access Modifier**.

Access Modifier: **public**, **private**, **protected**, default.
Non-access Modifier: **final**, **abstract**, **static**, **transient**, **synchronized**, **volatile**.

Encapsulation dilakukan menggunakan **Acces Modifier** atau biasa juga disebut **modifier visibilitas**. Jadi tidak semua modifier memiliki kegunaan sebagai modifier untuk melakukan encapsulation. Hanya Modifier Visibilitas saja yang digunakan.

Tabel modifier Level Akses terhadap member class (field, method, inner class) dari class yang sama (class itu sendiri), class lain di dalam satu package, dari subclass (di luar package), dan world (dari class di luar package dan bukan subclass).

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

- Untuk sebuah Class kita dapat menggunakan modifier: **public**, **default**.
- Untuk Inner Class kita dapat menggunakan modifier: **public**, **protected**, **default** dan **private**.
- Untuk **method**, **field**, dan **constructor** kita dapat menggunakan modifier: **public**, **protected**, **default** dan **private** (w3schools, 2022).

Tips dalam menentukan level akses terhadap class maupun member class:

- Jika programer lain menggunakan class kita, kita harus memastikan tidak akan terjadi kesalahan akibat penyalahgunaan field dan method. Hal ini dapat kita atur menggunakan access modifier.
- Utamakan menggunakan modifier private kecuali jika kita memiliki alasan kuat untuk tidak menggunakannya.
- Hindari penggunaan filed public kecuali untuk konstanta.
- Dalam latihan biasanya kita lebih sering menggunakan visibilitas public, tetapi untuk program yang akan dirilis (level production) sebaiknya hindari visibilitas public ini kecuali "terpaksa" (Oracle, The Java Tutorials, 2022).

INHERITANCE_____

Sebelum memulai membahas tentang **inheritance** mari kita membuat projectnya dahulu. Buatlah project bernama **projectinheritance**. Kemudian di dalam project tersebut buatlah dua package: **myprogram** dan **test**. Selanjutnya tuliskan source code berikut.

Class: myprogram.Manusia

```
package myprogram;

public abstract class Manusia {
        protected String nama;
        protected String alamat;

        protected String getNama() {
                return nama;
        }

        protected void setNama(String nama) {
                this.nama = nama;
        }

        protected String getAlamat() {
```

```
		return alamat;
	}

	protected void setAlamat(String alamat) {
		this.alamat = alamat;
	}
}
```

Class: myprogram.Mahasiswa

```
package myprogram;
public class Mahasiswa extends Manusia {
	private String nim;
	public String getNim() {
		return nim;
	}

	public void setNim(String nim) {
		this.nim = nim;
	}

	public String getNama() {
		return super.getNama();
	}

	public void setNama(String nama) {
		super.setNama(nama);
	}

	public String getAlamat() {
		return super.getAlamat();
	}

	public void setAlamat(String alamat) {
		super.setAlamat(alamat);
	}
```

```
	@Override public String toString() {
		String info = "Nama : " + nama
		+ "\nNIM : " + nim
		+ "\nAlamat : " + alamat;
		return info;
	}
}
```

Class: test.Main

```
package test;
import myprogram.Mahasiswa;
public class Main {
	public static void main(String[] args) {
		Mahasiswa m0 = new Mahasiswa();
		m0.setNama("Budi");
		m0.setNim("D0212212");
		m0.setAlamat("Kandemeng");
		System.out.println(m0);
	}
}
```

Pewarisan (inheritance) merupakan konsep dalam Object Oriented Programming yang memungkinkan kita untuk membuat class dari class yang telah ada atau telah dibuat sebelumnya. Class hasil penurunan class ini akan memiliki sifat (field, method, innerclass) yang dimiliki oleh superclass-nya. Member class dari superclass dengan modifier private tidak dapat diakses dari subclass. Member class di superclass dengan modifier default hanya dapat diakses dari subclass yang berada di package yang sama dengan superclassnya.

Class yang menjadi class orang tua (parent) biasa juga disebut **superclass** atau **class dasar**.

Class yang menjadi class anak (child) biasa juga disebut dengan **subclass**. Dalam pembahasan di praktikum ini kita akan lebih sering menggunakan istilah **subclass** dan **superclass**.

Keyword yang digunakan untuk melakukan inheritance adalah keyword **extends** ini berbeda dengan untuk membuat object yang menggunakan keyword **new**. Jadi inheritance berbeda dengan membuat object (instance).

Selain keyword **extends** di java ada keyword yang penggunaannya hampir mirip dengan **extends** yaitu keyword **implements**. Jika **extends** digunakan untuk melakukan pewarisan dari suatu superclass ke subclass, maka **implements** digunakan untuk mengimplementasi **interface**. Untuk membuat sebuah subclass kita hanya diijinkan untuk meng-**extends** satu superclass saja, sedangkan untuk penggunaan interface menggunakan **implements** kita diperbolehkan menggunakan lebih dari satu **implements**. Tapi jika yang akan kita turunkan adalah sebuah interface yang akan diturunkan ke subinterface, maka disini kita diperbolehkan membuat **subinterface** yang **extends** ke lebih dari satu **interface**. Inti sederhananya sebuah subclass hanya diijinkan memiliki satu superclass.

Interface mirip dengan class tetapi di dalam interface tidak terdapat operasi. Interface **HANYA** berisi deklarasi konstanta (variabel final) dan deklarasi method tanpa body. Jadi method-method di dalam interface itu tidak memiliki operasi hanya sekedar nama dan parameter serta tipe output saja dan modifier method di interface itu hanya diperbolehkan menggunakan modifier default dan public. Jika tidak memiliki field selain konstanta dan methodnya juga tidak memiliki operasi, lalu untuk apa interface ini dibuat? Interface ini adalah semacam protokol yang digunakan untuk komunikasi antar object. Jadi di dalam interface itu sudah ditentukan apa saja yang **HARUS** disiapkan jika dua object akan saling bertukar data. Perlu diingat pula bahwa istilah **interface** disini bukanlah Graphical User Interface (GUI) maupun Command Line Interface (CLI). Ini adalah interface (antarmuka) pada level pemrograman jadi program yang akan saling bertukar data sedangkan GUI dan CLI itu gunanya untuk menghubungkan manusia dan komputer.

Interface dibuat dijava menggunakan awalan keyword **interface** dan nanti ketika hendak digunakan maka dituliskan setelah keyword **implements** pada deklarasi class.

Interface: myprogram.Cetak

```
package myprogram;
public interface Cetak {
	public final String PROJECT_NAME = "PROJECT INHERITANCE";
	public void cetakInfo();
}
```

Kemudian interface ini akan kita coba implementasikan di subclass Mahasiswa

Class: myprogram.Mahasiswa

```
package myprogram;
public class Mahasiswa extends Manusia implements Cetak{
	private String nim;
	public String getNim() {
		return nim;
	}

	public void setNim(String nim) {
		this.nim = nim;
	}

	public String getNama() {
		return super.getNama();
	}

	public void setNama(String nama) {
		super.setNama(nama);
	}

	public String getAlamat() {
		return super.getAlamat();
	}
```

```
    public void setAlamat(String alamat) {
        super.setAlamat(alamat);
    }

    @Override
    public String toString() {
        String info = "Nama : " + nama
        + "\nNIM : " + nim
        + "\nAlamat : " + alamat;
        return info;
    }

    @Override
    public void cetakInfo() {
        System.out.println(this);
    }
}
```

Saat menggunakan interface kita akan **dipaksa** untuk **meng-override** semua method yang ada di interface yang kita implements.

Selain interface sebenarnya kita juga telah menggunakan superclass yang mirip dengan interface yaitu **abstract superclass**. **Abstract superclass** adalah class yang dalam deklarasinya kita tambahkan modifier **abstract**. Modifier **abstract** ini akan mengakibatkan kita tidak dapat membuat object (instan) dari class abstract ini. Untuk menggunakan class **abstract** kita harus terlebih dahulu melakukan inheritance menggunakan keyword extends lalu nanti class hasil inheritance ini yang kita buatkan object-nya. Pada contoh source code di atas kita memiliki class **abstract** bernama Manusia, dan telah kita inheritance ke subclass bernama Mahasiswa.

Perhatikan varian operasi berikut:

```
Mahasiswa man1 = new Mahasiswa();//boleh
Manusia man2 = new Manusia();//tidak diperbolehkan
Manusia man3 = new Mahasiswa();//boleh
```

Pembuatan object man2 diatas tidak diperbolehkan karena class Manusia adalah class **abstract**.

Kebalikan dari keyword **abstract** adalah keyword **final**. pada sebuah class modifier final class akan mengakibatkan class tersebut tidak dapat diturunkan (diwariskan) dan hanya dapat dibuat instannya. Jadi keyword final ini selain dapat kita gunakan untuk membuat konstanta juga dapat digunakan untuk membuat class yang tidak boleh diwariskan dan hanya boleh dibuatkan object-nya.

POLYMORPHISM_____

Polymorphism (polimorfisma) dapat diartikan sebagai banyak bentuk atau satu nama tetapi memiliki banyak bentuk. Polymorphism merupakan salah satu pilar atau fitur dari object oriented programming yang memungkinkan class atau object memiliki beberapa method dengan nama yang sama tetapi operasi maupun parameter dari method tersebut berbeda.

Polymorphism dibuat menggunakan teknik **overloading** dan **overriding** yang telah kita bahas di praktikum sebelumnya.

Overloading yang paling sering kita temui mungkin adalah **overloading constructor** dan **overriding** yang paling sering kita jumpai mungkin adalah overriding method toString(). Jika kalian masih samar-samar tentang overloading dan overriding silahkan pelajari kembali materi praktikum sebelumya. Overloading adalah teknik pemrograman berupa pembuatan method dengan nama yang sama di dalam satu class yang sama, sementara Overriding adalah pembuatan method di subclass yang namanya sama dengan method yang ada di superclass-nya atau di interface-interface yang di-implements di subclass. Pada saat mengimplements sebuah interface, kita diharuskan untuk melakukan override terhadap semua method yang ada di interface tersebut. Method-method hasil dari overloading dan overriding ini bisa saja memiliki parameter input dan operasi berbeda sehingga ketika dijalankan akan menghasilkan nilai atau perilaku yang berbeda dari method asalnya.

Buatlah sebuah project bernama **projectpolymorphism**. Lalu di dalam project tersebut buatlah package **unsulbar** dan package **test**. Buat sebuah kelas bernama **Latihan** di package **unsulbar**. Kemudian masih di package unsulbar buatlah sebuah package bernama **informatika** di dalam package **unsulbar**.

Lengkapi class unsurbar.Latihan menjadi sebagai berikut:
Class: unsulbar.Latihan

```
package unsulbar;
public class Latihan {
	public String plus(String str1, String str2) {
		return str1 + str2;
	}

	public double plus(double a, double b) {
		return a + b;
	}

	public int plus(int a, int b) {
		return a + b;
	}
}
```

Pada class **Latihan** ini kita melakukan overloading terhadap method **plus( )**. Method plus( ) kita overloading menjadi tiga buah method yang namanya sama tetapi operasi, parameter input, dan outputnya berbeda. Jika parameternya adalah dua buah String maka outputnya akan menghasilkan concatenation atau penggabungan dari dua string yang diinputkan. Demikian pula jika parameter inputnya adalah tipe double maka outputnya adalah hasil penjumlahan dua bilangan bertipe double.

Selanjutnya pada package testing kita membuat class main bernama Test01 untuk menguji hasil overloading method yang kita buat serta melihat perilaku polymorphism menggunakan overloading.

Class: test.Test01

```
package test;
import unsulbar.Latihan;
public class Test01 {
	public static void main(String[] args) {
		Latihan latihan = new Latihan();
		String s1 = "Informatika";
		String s2 = "Unsulbar";
		String s3 = latihan.plus(s1, s2);
		System.out.println(s3);
		int bil = 1234;
		int bul = 56789;
		int bilbul = latihan.plus(bil, bul);
		System.out.println(bilbul);
		double d1 = 12.34;
		double d2 = 567.89;
		double d3 = latihan.plus(d1, d2);
		System.out.println(d3);
	}
}
```

Berikutnya kita akan membuat operasi **polymorphism** menggunakan **overriding** method dari superclass dan interface.

Buatlah sebuah **abstract class** dan sebuah **interface** di dalam package unsulbar.informatika. superclass dan interface ini akan kita gunakan untuk membuat subclass bangun datar: Lingkaran, Persegi, dan Persegi Panjang.

Class: unsulbar.informatika.BangunDatar

```
package unsulbar.informatika;
public abstract class BangunDatar {
	protected double luas, keliling;
	public abstract double hitungLuas();
	public abstract double hitungKeliling();
}
```

Interface: unsulbar.informatika.InfoBangunDatar

```
package unsulbar.informatika;
public interface InfoBangunDatar {
	public final double PI = 3.14;
	public void printInfo();
	public void printTime();
}
```

Selanjutnya kita akan melakukan inheritance dari superclass BangunDatar dan interface InfoBagunDatar untuk membuat subclass: Lingkaran, Persegi, dan PersegiPanjang serta membuat **overriding** method di dalamnya.

Class: unsulbar.Lingkaran

```
package unsulbar;
import java.util.Date;
import unsulbar.informatika.BangunDatar;
import unsulbar.informatika.InfoBangunDatar;

public class Lingkaran extends BangunDatar implements InfoBangunDatar {
	private final double radius;
		public Lingkaran(double radius) {
		this.radius = radius;
	}
	@Override
	public double hitungLuas() {
		luas = PI * radius * radius; return luas;
	}
	@Override
	public double hitungKeliling() {
		keliling = 2 * PI * radius; return keliling;
	}

	@Override
```

```
        public void printInfo() {
                System.out.println("Bagun Datar LINGKARAN. radius = " + radius + ";
                Luas = " + luas + ";
                Keliling = " + keliling);
        }

        @Override
        public void printTime() {
                Date date = new Date();
                System.out.println("Lingkaran-" + date.toString());
        }
}
```

Class: unsulbar.Persegi

```
package unsulbar;
import java.util.Date;
import unsulbar.informatika.BangunDatar;
import unsulbar.informatika.InfoBangunDatar;

public class Persegi extends BangunDatar implements InfoBangunDatar {
        private final double sisi;

        public Persegi(double sisi) {
                this.sisi = sisi;
        }

        @Override
        public double hitungLuas() {
                luas = sisi * sisi;
                Return luas;
        }

        @Override
        public double hitungKeliling() {
                keliling = 4 * sisi; return keliling;
```

```
        }

        @Override
        public void printInfo() {
                System.out.println("Bagun Datar PERSEGI. Sisi = " + sisi + ";
                Luas = " + luas + ";
                Keliling = " + keliling);
        }

        @Override
        public void printTime() {
                Date date = new Date();
                System.out.println("Persegi-"+date.toString());
        }
}
```

Class: unsulbar.PersegiPanjang

```
package unsulbar;
import java.util.Date;
import unsulbar.informatika.BangunDatar;
import unsulbar.informatika.InfoBangunDatar;

public class PersegiPanjang extends BangunDatar implements
InfoBangunDatar {
        private final double panjang, lebar;

        public PersegiPanjang(double panjang, double lebar) {
                this.panjang = panjang;
                this.lebar = lebar;
        }

        @Override
        public double hitungLuas() {
                luas = panjang * lebar;
                return luas;
```

```
        }

        @Override
        public double hitungKeliling() {
                keliling = panjang + lebar + panjang + lebar;
                return keliling;
        }

        @Override
        public void printInfo() {
                System.out.println("Bagun Datar PERSEGI PANJANG. Panjang = " +
                panjang + ";
                Lebar = " + lebar + ";
                Luas = " + luas + ";
                Keliling = " + keliling);
        }

        @Override
        public void printTime() {
                Date date = new Date();
                System.out.println("PersegiPanjang-" + date.toString());
        }
}
```

Kita telah melakukan overriding terhadap method: hitungKeliling( ), hitungLuas( ), printInfo( ), dan printTime( ). Keempat method ini memiliki perilaku/operasi berbeda-beda di setiap subclass yang telah kita buat.

Berikutnya kita akan membuat class Main bernama Test02 di package test. Di dalam method main() class Test02 kita akan membuat object dari class Lingkaran, Persegi, dan PersegiPanjang untuk melihat hasil overriding method yang telah dilakukan.

```
Lingkaran lingkaran01 = new Lingkaran(27);
Persegi persegi01 = new Persegi(162);
```

```
PersegiPanjang persegiPanjang01 = new PersegiPanjang(32, 75);
System.out.println("-----------------------------------");
lingkaran01.hitungLuas();
lingkaran01.hitungKeliling();
lingkaran01.printInfo();
lingkaran01.printTime();  System.out.println("-----------------------------------
----"); persegi01.hitungLuas();
persegi01.hitungKeliling();
persegi01.printInfo();
persegi01.printTime(); System.out.println("-----------------------------------
--"); persegiPanjang01.hitungLuas();
persegiPanjang01.hitungKeliling();
persegiPanjang01.printInfo();
persegiPanjang01.printTime();
```

Kita juga dapat membuat object dari superclass tetapi dengan cara memanggil constructor dari subclass-nya seperti ini.

```
BangunDatar lingkaran02 = new Lingkaran(27);
BangunDatar persegi02 = new Persegi(162);
BangunDatar persegiPanjang02 = new PersegiPanjang(32, 75);
```

Tetapi dengan cara seperti ini kita hanya dapat menggunakan field dan method yang ada di superclassnya. Seperti pada contoh ini, karena pada class BangunDatar tidak tersedia method printInfo() dan printTime(), maka kita tidak dapat melakukan hal berikut:

```
persegiPanjang02.printInfo();
persegiPanjang02.printTime();
```

source code lengkap untuk class Test02 adalah sebagai berikut.

Class: test.Test02

```
package test;
import unsulbar.Lingkaran;
```

```
import unsulbar.Persegi;
import unsulbar.PersegiPanjang;
import unsulbar.informatika.BangunDatar;
public class Test02 {
	public static void main(String[] args) {
		Lingkaran lingkaran01 = new Lingkaran(27);
		Persegi persegi01 = new Persegi(162);
		PersegiPanjang persegiPanjang01 = new PersegiPanjang(32, 75);
		System.out.println("-------------------------------------");
		lingkaran01.hitungLuas();
		lingkaran01.hitungKeliling();
		lingkaran01.printInfo();
		lingkaran01.printTime();  System.out.println("----------------------
		-----------------"); persegi01.hitungLuas();
		persegi01.hitungKeliling();
		persegi01.printInfo();
		persegi01.printTime();  System.out.println("----------------------
		---------------"); persegiPanjang01.hitungLuas();
		persegiPanjang01.hitungKeliling();
		persegiPanjang01.printInfo();
		persegiPanjang01.printTime();
		BangunDatar lingkaran02 = new Lingkaran(27);
		BangunDatar persegi02 = new Persegi(162);
		BangunDatar persegiPanjang02 = new PersegiPanjang(32, 75);
	}
}
```

Sampai disini kita telah mempelajari **tiga** fitur utama dari **Object Oriented Programming** yaitu: **Inheritance**, **Encapsulation dan Polymorphism**. Mahasiswa diharapkan telah memiliki pemahaman yang kuat tentang konsep dan penerapan dasar Pemrograman Berorientasi Objek sehingga siap untuk menghadapi teknologi yang menerapkan Pemrograman Berorientasi Objek termasuk yang dibuat menggunakan bahasa pemrograman lain selain Java.

Selamat Belajar, Jangan Lupa Berdoa dan Bersyukur. Terimakasih...

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. Tugas Praktikum

1. Buatlah sebuah project java bernama projectencapsulation.
2. Di dalam project projectencapsulation buatlah dua buah package: myprogram dan test.
3. Di dalam package test buatlah sebuah class Main dengan method main() di dalamnya.
4. Di dalam package myprogram buatlah sebuah abstract class bernama class Manusia yang memiliki field protected: nama, alamat, kemudian siapkan method protected getter dan protected setter untuk field nama, alamat.
5. Di dalam package myprogram buatlah sebuah class lagi bernama class Mahasiswa yang merupakan subclass dari class Manusia. Buatlah field private nim di class Mahasiswa. Buat method public getter dan public setter untuk field: nama, alamat, dan nim. Untuk method getter dan setter untuk field nama dan alamat harus berisi pemanggilan terhadap method getter dan setter field nama dan alamat yang telah kita buat dengan class super di class Manusia, jadi disini kita akan memperagakan penggunaan keyword super. Lakukan overriding terhadap method toString() untuk menuliskan String info: Nama, NIM, dan Alamat.
6. Di dalam package test, pada method main() milik class Main buatlah Array List berisi 20 object dari class Mahasiswa kemudian tampilkan nilai field fieldnya dengan cara memanggil method toString() dari masing-masing object Mahasiswa di dalam looping FOR-EACH.
7. Buatlah sebuah project java.
8. Di dalam project tersebut buatlah 10 buah packege berbeda: paket1, paket2, paket3, paket4, paket5, paket6, paket7, paket8, paket9, paket10.
9. Di dalam package-package itu buatlah masing-masing sebuah superclass lima diantaranya harus merupakan superclass abstract.
10. Buatlah sebuah package lagi di dalam project yang kita buat tadi, sebuah package bernama paketinterface. di dalam package interface ini buatlah sebuah interface bernama MyInterface yang berisi method bernama String getStringInfoState( );

11. Buatlah package test di dalam project yang kita buat tadi. Di dalam package test ini buatlah 10 class yang masing-masing berupakan subclass dari class class yang berada di paket1, paket2, paket3, paket4, paket5, paket6, paket7, paket8, paket9, paket10. Jadi masing-masing superclass yang telah kita buat akan memiliki satu subclass. Semua subclass ini akan meng-implements interface MyInterface.
12. Buatlah sebuah class Main di package test untuk memperlihatkan (mensimulasikan) hasil penggunaan superclass, subclass dan interface dari tugas nomor 7 sampai 11.
13. Buatlah program animasi sederhana menggunakan dua cara berbeda: 1) dengan cara melakukan extends terhadap class java.lang.Thread. 2) dengan cara melakukan implements terhadap interface java.lang.Runnable.
14. Buatlah project baru yang berbeda dari project di tugas nomor sebelumnya. Di dalam project tersebut buatlah sebuah package bernama latihan.inheritance. Di dalam package tersebut (latihan.inheritance) buatlah tiga buah interface (InterfaceA, InterfaceB, dan InterfaceC). Selanjutnya buatlah sebuah package baru bernama test.inheritance.
15. Buatlah empat buah interface berbeda (InterfacePertama, InterfaceKedua, InterfaceKetiga, InterfaceKeempat) , kemuadian InterfaceKeempat harus extends ke InterfacePertama, InterfaceKedua, dan InterfaceKetiga.
16. Buatlah dua buah class Abstact (AbstractClassPertama dan AbstractClassKedua) lalu buatlah sebuah class lagi (CHildClass) jadi hanya ada tiga class yang dibuat. Aturlah sedemikian rupa agar ChildClass ini menjadi turunan dari AbstractClassPertama dan AbstractClassKedua. Ingat hanya ada tiga buah class yang boleh dibuat. Tidak ada pembuatan/penggunaan interface.
17. Lakukan overloading method terhadap 10 method berbeda (jadi minimal kita akan membuat 20 method). Buatlah sebuah Main Class untuk melihat hasil overloading ini.
18. Lakukan overriding method dengan ketentuan: 10 abstract method berasal dari abstract class, 10 method lagi (bukan abstract method) dari abstract class dan 10 method berasal dari interface pertama dan 10 method dari interface kedua (jadi ada dua interface yang dibuat). Abstract Class dan dua Interface ini selanjutnya digunakan untuk membuat 10 subclass yang extends ke abstract class dan implements ke interface pertama dan kedua.

19. Buatlah sepuluh class (ClassA, ClassB, ClassC, ClassD, ClassE, ClassF, ClassG, ClassH, ClassI, ClassJ). Disetiap class terdapat overriding ke method public String toString() { }. Method toString ini akan mengembalikan (return) String nama class tempat method toString( ) tersebut ditulis. Dengan class yang extends masing-masing sebagai berikut:
    ClassB extends ClassA
    ClassC extends ClassB
    ClassD extends ClassC
    ClassE extends ClassD
    ClassF extends ClassE
    ClassG extends ClassF
    ClassH extends ClassG
    ClassI extends ClassH
    ClassJ extends ClassI
    Buatlah sebuah class lagi yang memiliki method main. di dalam method main buatlah object dari sepuluh class tersebut (ClassA, ClassB, ClassC, ClassD, ClassE, ClassF, ClassG, ClassH, ClassI, ClassJ) lalu lakukan pemanggilan method toString dari sepuluh object yang telah dibuat.
20. Buatlah sebuah Super class yang memiliki method abstrack yang akan di override oleh subclass. Kemudian buatlah minimal 5 sub class yang akan mengoverride method yg telah dibuat pada super class. Lakukan modifikasi pada method yang dioverride sekreatif mungkin. kemudian buatlah sebuah class main yang berisi minimal 10 objek yang di instans dari setiap subclass dan panggil method yang telah di override.

PRAKTIKUM 4: GRAPHICAL USER INTERFACE JAVAFX

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator Penilaian

1. Mahasiswa mampu mengembangkan aplikasi java Graphical User Interface (GUI) menggunakan JavaFX.
2. Mahasiswa mampu menggunakan class-class yang digunakan untuk membuat elemen GUI di JavaFX.
3. Mahasiswa mampu melakukan kustomisasi GUI dengan cara memperluas (extends dan implements) terhadap class-class yang tersedia di library JavaFX.
4. Mahasiswa mampu membuat visualisasi data menggunakan JavaFX.
5. Mahasiswa mampu membuat Program GUI untuk berbagai keperluan.

B. Tugas Pendahuluan

1. Tuliskan minimal 20 class yang ada di library JavaFX beserta kegunaannya sertakan pula method yang ada untuk setiap class dalam membangun komponen GUI!
2. Jelaskan class utama yang harus extend untuk membuat aplikasi JavaFX !
3. Jelaskan tentang class javafx.scene.canvas.Canvas dan class javafx.scene.canvas.GraphicsContext !
4. Jelaskan tentang class AnimationTimer !
5. Tuliskan contoh program untuk membuat sebuah lingkaran dan persegi panjang dengan Canvas dan GraphicsContext!
6. Jelaskan perbedaan antara **INTERFACE** dan **GRAPHICAL USER INTERFACE** di java!

C. Ringkasan Materi

JAVAFX_____

JavaFX adalah sebuah library modern dari Java yang digunakan untuk membangun Graphical User Interface (GUI) secara lebih kaya, fleksibel dan dinamis dibandingkan dengan library pendahulunya seperti AWT dan Swing.

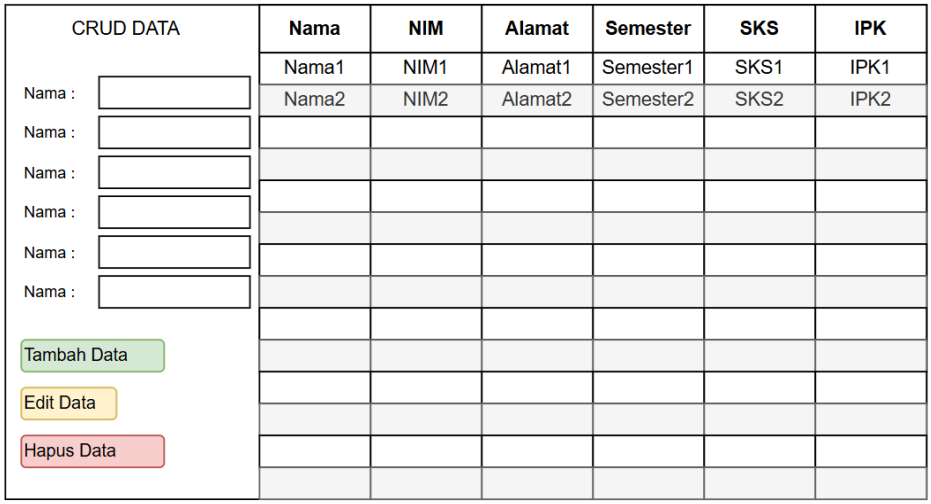
Pada praktikum ini anda diharapkan dapat mengembangkan GUI dan Visualisasi Algoritma menggunakan library JavaFX. Anda diharapkan mampu mengembangkan GUI yang dinamis, responsif dan mampu menampilkan animasi dan visualisasi algoritma. Selain menggunakan JavaFX masih ada library java lainnya yang dapat digunakan untuk membangun GUI di Java diantaranya: Java SWT, AWT, SWING. Kita juga dapat menggunakan secara bersamaan library tersebut jika dibutuhkan.

Berikut ini beberapa URL sumber belajar yang dapat membantu meningkatkan kemampuan pengembangan GUI di Java :

1. **Java Swing** Tutorial: https://zetcode.com/javaswing/
2. **Java 2D Tutorial**: https://zetcode.com/gfx/java2d/
3. **Java 2D Games**: https://zetcode.com/javagames/
4. **JavaFX Tutorial**: https://zetcode.com/gui/javafx/
5. **Login Form Java** : www.youtube.com/watch?v=jHSBrX8lLWk&t=799s
6. Tutorial EECCHHOO: https://eecchhoo.wordpress.com/category/java/java swing/

GUI MENGGUNAKAN JAVAFX_____

Kita akan membuat aplikasi Graphical User Interface sederhana untuk membaca, mengubah, menghapus dan menambah data dari file CSV dan akan menampilkannya ke tabel. Kita buat sketsa GUI-nya terlebih dahulu :



Langkah 1

Buatlah sebuah project JavaFX dengan nama GUI, lalu dalam folder java buat sebuah package bernama CRUD.

Langkah 2

Buatlah sebuah class yang extends ke class javafx.application.Application bernama MyJendela dan juga class Mahasiswa yang berisi field nim, nama, alamat, semester, sks dan ipk.

```
package CRUD;
import javafx.application.Application;
public class MyJendela extends Application {

}
```

```
package CRUD;

public class Mahasiswa {
   String nim,nama,alamat,semester,sks,ipk;

   public String getNim() {
      return nim;
   }

   public void setNim(String nim) {
      this.nim = nim;
   }

   public String getNama() {
      return nama;
   }

   public void setNama(String nama) {
      this.nama = nama;
   }

   public String getAlamat() {
      return alamat;
   }
```

```
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }

    public String getSemester() {
        return semester;
    }

    public void setSemester(String semester) {
        this.semester = semester;
    }

    public String getSks() {
        return sks;
    }

    public void setSks(String sks) {
        this.sks = sks;
    }

    public String getIpk() {
        return ipk;
    }

    public void setIpk(String ipk) {
        this.ipk = ipk;
    }

    public Mahasiswa(String nim, String nama, String alamat, String semester,
String sks, String ipk) {
        this.nim = nim;
        this.nama = nama;
        this.alamat = alamat;
        this.semester = semester;
        this.sks = sks;
        this.ipk = ipk;
    }
}
```

Langkah 3

Lakukan Override method start pada class MyJendela lalu inisialisasi stage beserta class BorderPane dan Scene untuk menampilkan jendela utama.

```
package CRUD;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.stage.WindowEvent;

public class MyJendela extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        //stage
        stage.setTitle("CRUD SEDERHANA");
        stage.initStyle(StageStyle.DECORATED);
        stage.setMaximized(true);
       //Pane & Scene
        BorderPane root =  new BorderPane();
        Scene scene = new Scene(root);
        //stage
        stage.setScene(scene);
        stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
            @Override
            public void handle(WindowEvent windowEvent) {
                Platform.exit();
                System.exit(0);
            }
        });
        stage.show();
    }
}
```

Langkah 4

Langkah selanjutnya kita akan membuat bagian sisi kiri dengan menggunakan VBox yang berisi komponen-komponen seperti HBox, label, button, dan textField.

```
package CRUD;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.stage.WindowEvent;


public class MyJendela extends Application {
   //komponen
   private TextField tNIM,tNama,tAlamat,tSemester,tSKS,tIPK;

   @Override
   public void start(Stage stage) throws Exception {

      //stage
      stage.setTitle("CRUD SEDERHANA");
      stage.initStyle(StageStyle.DECORATED);
      stage.setMaximized(true);

      //Pane & Scene
      BorderPane root =  new BorderPane();
      Scene scene = new Scene(root);

     //Vbox
      VBox side = new VBox(10);
      side.setPadding(new Insets(20,10,10,20));
      side.setPrefWidth(260);
      Label title = new Label("Data Mahasiswa");
      tNIM = new TextField();
      tNama = new TextField();
      tAlamat= new TextField();
      tSemester= new TextField();
```

```
        tSKS= new TextField();
        tIPK= new TextField();

        //Penambahan Komponen Side
        side.getChildren().addAll(
                title,
                createField("NIM \t\t",tNIM),
                createField("Nama \t",tNama),
                createField("Alamat \t",tAlamat),
                createField("Semester ",tSemester),
                createField("SKS \t\t",tSKS),
                createField("IPK \t\t",tIPK)
                );
        // tombol
        Button btnGetData = new Button("Tambah Data");
        Button btnEdit = new Button("Edit Data");
        Button btnDelete = new Button("Hapus Data");
        //menambah tombol kedalam side
        side.getChildren().addAll(btnGetData,btnEdit,btnDelete);

        //penambahan side ke root
        root.setLeft(side);
        stage.setScene(scene);
        stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
            @Override
            public void handle(WindowEvent windowEvent) {
                Platform.exit();
                System.exit(0);
            }
        });
        stage.show();

    }
   // method untuk komponen HBox di side
    private HBox createField(String labelText,TextField textField){
        Label label = new Label(labelText+":");
        textField.setMaxWidth(200);
        HBox field = new HBox(10,label,textField);
        return field;
    }
}
```

Langkah 5

Langkah berikutnya kita akan menambahkan bagian tabel untuk menampilkan data.

```
package CRUD;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.stage.WindowEvent;

public class MyJendela extends Application {
    //komponen
    private TextField tNIM,tNama,tAlamat,tSemester,tSKS,tIPK;
    private TableView <Mahasiswa> table;
    private ObservableList<Mahasiswa> dataList;
    private Mahasiswa selectedMahasiswa = null;
    @Override
    public void start(Stage stage) throws Exception {

        //stage
        stage.setTitle("CRUD SEDERHANA");
        stage.initStyle(StageStyle.DECORATED);
        stage.setMaximized(true);

        //Pane & Scene
        BorderPane root =  new BorderPane();
        Scene scene = new Scene(root);
        //VBox
        VBox side = new VBox(10);
        side.setPadding(new Insets(20,10,10,20));
```

```
        side.setPrefWidth(260);
        Label title = new Label("Data Mahasiswa");
        tNIM = new TextField();
        tNama = new TextField();
        tAlamat= new TextField();
        tSemester= new TextField();
        tSKS= new TextField();
        tIPK= new TextField();

        //Penambahan Komponen Side
        side.getChildren().addAll(
                title,
                createField("NIM \t\t",tNIM),
                createField("Nama \t",tNama),
                createField("Alamat \t",tAlamat),
                createField("Semester ",tSemester),
                createField("SKS \t\t",tSKS),
                createField("IPK \t\t",tIPK)
                );

        //tombol
        Button btnGetData = new Button("Tambah Data");
        Button btnEdit = new Button("Edit Data");
        Button btnDelete = new Button("Hapus Data");
       //menambah tombol kedalam side
       side.getChildren().addAll(btnGetData,btnEdit,btnDelete);

        //table
           String[] columnNames = {"NIM","Nama","Alamat","Semester","SKS","IPK"};
        table = new TableView<>();
        dataList = FXCollections.observableArrayList();
        for (String colName : columnNames){
           TableColumn<Mahasiswa,String> col = new TableColumn<>(colName);
           col.setCellValueFactory(new
PropertyValueFactory<>(colName.toLowerCase()));
           col.setPrefWidth(150);
           table.getColumns().addAll(col);
        }
        table.setItems(dataList);

        //penambahan tabel ke root
```

```
        root.setCenter(table);
        //penambahan side ke root
        root.setLeft(side);
        stage.setScene(scene);
        stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
            @Override
            public void handle(WindowEvent windowEvent) {
                Platform.exit();
                System.exit(0);
            }
        });
        stage.show();
    }
    private HBox createField(String labelText,TextField textField){
        Label label = new Label(labelText+":");
        textField.setMaxWidth(200);
        HBox field = new HBox(10,label,textField);
        return field;
    }
}
```

Langkah 6

Tambahkan style untuk setiap komponen agar kelihatan lebih menarik.

```
package CRUD;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.stage.WindowEvent;
public class MyJendela extends Application {
```

```
//komponen
private TextField tNIM,tNama,tAlamat,tSemester,tSKS,tIPK;
private TableView <Mahasiswa> table;
private ObservableList<Mahasiswa> dataList;
private Mahasiswa selectedMahasiswa = null;
@Override
public void start(Stage stage) throws Exception {
   //stage
   stage.setTitle("CRUD SEDERHANA");
   stage.initStyle(StageStyle.DECORATED);
   stage.setMaximized(true);
   //Pane & Scene
   BorderPane root =  new BorderPane();
   Scene scene = new Scene(root);
   //VBox
   VBox side = new VBox(10);
   side.setPadding(new Insets(20,10,10,20));
   side.setPrefWidth(260);
   Label title = new Label("Data Mahasiswa");
   tNIM = new TextField();
   tNama = new TextField();
   tAlamat= new TextField();
   tSemester= new TextField();
   tSKS= new TextField();
   tIPK= new TextField();

   //Penambahan Komponen Side
   side.getChildren().addAll(
         title,
         createField("NIM \t\t",tNIM),
         createField("Nama \t",tNama),
         createField("Alamat \t",tAlamat),
         createField("Semester ",tSemester),
         createField("SKS \t\t",tSKS),
         createField("IPK \t\t",tIPK)
         );

   //tombol
   Button btnGetData = new Button("Tambah Data");
   Button btnEdit = new Button("Edit Data");
   Button btnDelete = new Button("Hapus Data");
```

```
        side.getChildren().addAll(btnGetData,btnEdit,btnDelete);

        //tabel
        String[] columnNames = {"NIM","Nama","Alamat","Semester","SKS","IPK"};
        table = new TableView<>();
        dataList = FXCollections.observableArrayList();
        for (String colName : columnNames){
            TableColumn<Mahasiswa,String> col = new TableColumn<>(colName);
            col.setCellValueFactory(new
PropertyValueFactory<>(colName.toLowerCase()));
            col.setPrefWidth(150);
            table.getColumns().addAll(col);
        }
        table.setItems(dataList);

        //penambahan tabel ke root
        root.setCenter(table);
        //penambahan side ke root
        root.setLeft(side);
        stage.setScene(scene);
        stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
            @Override
            public void handle(WindowEvent windowEvent) {
                Platform.exit();
                System.exit(0);
            }
        });

        //style
        side.setStyle("-fx-background-color:#2c3e50;");
        title.setStyle("-fx-font-size:18px; -fx-text-fill:white;");
        btnGetData.setStyle("-fx-background-color:#27ea60;-fx-text-fill:white;");
        btnEdit.setStyle("-fx-background-color:#f39c12;-fx-text-fill:white;");
        btnDelete.setStyle("-fx-background-color:#e74c3c;-fx-text-fill:white");
        table.setStyle("-fx-font-size:14px");

        stage.show();


    }
```

```
    private HBox createField(String labelText,TextField textField){
        Label label = new Label(labelText+":");
        textField.setMaxWidth(200);
        HBox field = new HBox(10,label,textField);
        //style
        textField.setStyle("-fx-background-color:transparent; -fx-border-
color:#3498db; -fx-border-width:2px;-fx-text-fill:white;-fx-padding:5px;");
        label.setStyle("-fx-text-fill:white;-fx-font-size:14px;");
        return field;
    }
}
```

Langkah 7

Tambahkan method-method yang diperlukan seperti method untuk membuka file csv,menambah, menghapus, mengedit, menyimpan dan memilih data. Serta tambahkan panggilan method ketika tombol ditekan.

```
package CRUD;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.stage.WindowEvent;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
```

```
public class MyJendela extends Application {
    //komponen
    private TextField tNIM,tNama,tAlamat,tSemester,tSKS,tIPK;
    private TableView <Mahasiswa> table;
    private ObservableList<Mahasiswa> dataList;
    private Mahasiswa selectedMahasiswa = null;
    @Override
    public void start(Stage stage) throws Exception {
        //stage
        stage.setTitle("CRUD SEDERHANA");
        stage.initStyle(StageStyle.DECORATED);
        stage.setMaximized(true);
        //Pane & Scene
        BorderPane root =  new BorderPane();
        Scene scene = new Scene(root);
        //VBox
        VBox side = new VBox(10);
        side.setPadding(new Insets(20,10,10,20));
        side.setPrefWidth(260);
        Label title = new Label("Data Mahasiswa");
        tNIM = new TextField();
        tNama = new TextField();
        tAlamat= new TextField();
        tSemester= new TextField();
        tSKS= new TextField();
        tIPK= new TextField();

        //Penambahan Komponen Side
        side.getChildren().addAll(
                title,
                createField("NIM \t\t",tNIM),
                createField("Nama \t",tNama),
                createField("Alamat \t",tAlamat),
                createField("Semester ",tSemester),
                createField("SKS \t\t",tSKS),
                createField("IPK \t\t",tIPK)
                );

        //tombol
        Button btnGetData = new Button("Tambah Data");
```

```
        Button btnEdit = new Button("Edit Data");
        Button btnDelete = new Button("Hapus Data");

        //aksi Ketika menekan tombol
        btnGetData.setOnAction(e -> tambahData());
        btnEdit.setOnAction(e -> editData());
        btnDelete.setOnAction(e-> hapusData());

        //menambah tombol kedalam side
        side.getChildren().addAll(btnGetData,btnEdit,btnDelete);

        //tabel
        String[] columnNames = {"NIM","Nama","Alamat","Semester","SKS","IPK"};
        table = new TableView<>();
        dataList = FXCollections.observableArrayList();
        for (String colName : columnNames){
            TableColumn<Mahasiswa,String> col = new TableColumn<>(colName);
            col.setCellValueFactory(new
PropertyValueFactory<>(colName.toLowerCase()));
            col.setPrefWidth(150);
            table.getColumns().addAll(col);
        }
        table.setItems(dataList);
        //aksi Ketika memilih data
        table.setOnMouseClicked(e -> pilihMahasiswa());
        //penambahan tabel ke root
        root.setCenter(table);
        //penambahan side ke root
        root.setLeft(side);
        stage.setScene(scene);
        stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
            @Override
            public void handle(WindowEvent windowEvent) {
                Platform.exit();
                System.exit(0);
            }
        });

        //style
        side.setStyle("-fx-background-color:#2c3e50;");
        title.setStyle("-fx-font-size:18px; -fx-text-fill:white;");
```

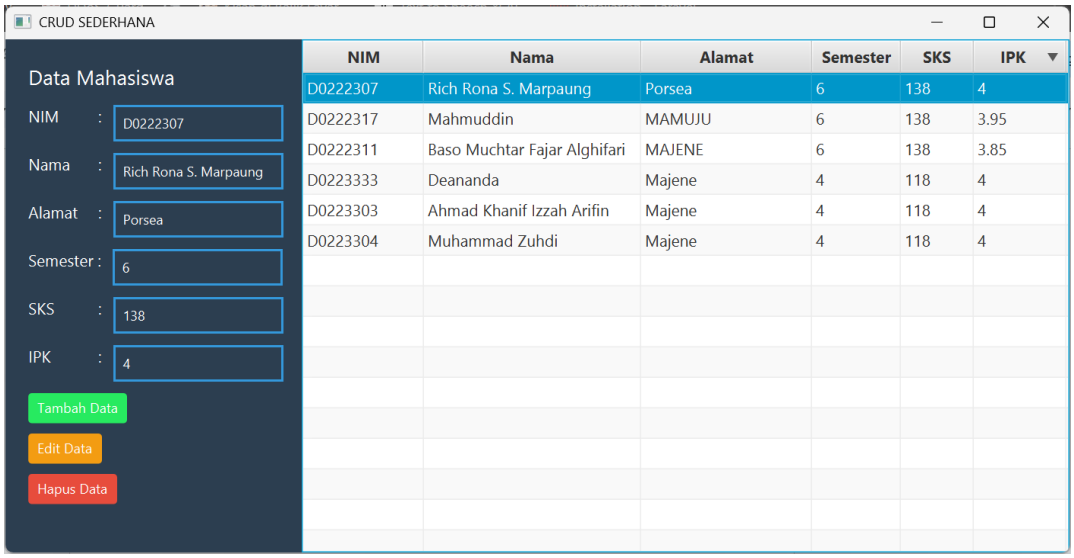
```
        btnGetData.setStyle("-fx-background-color:#27ea60;-fx-text-fill:white;");
        btnEdit.setStyle("-fx-background-color:#f39c12;-fx-text-fill:white;");
        btnDelete.setStyle("-fx-background-color:#e74c3c;-fx-text-fill:white");
        table.setStyle("-fx-font-size:14px");

        muatDataDariCSV();
        stage.show();


    }
    private HBox createField(String labelText,TextField textField){
        Label label = new Label(labelText+":");
        textField.setMaxWidth(200);
        HBox field = new HBox(10,label,textField);

        //style
        textField.setStyle("-fx-background-color:transparent; -fx-border-
color:#3498db; -fx-border-width:2px;-fx-text-fill:white;-fx-padding:5px;");
        label.setStyle("-fx-text-fill:white;-fx-font-size:14px;");
        return field;
    }
    private void muatDataDariCSV(){
        String filename = "database.csv";
        Path path = Path.of(filename);
        ArrayList<Mahasiswa> data = new ArrayList<>();
        try {
            List<String> lines = Files.readAllLines(path);

            for (int i = 1; i < lines.size(); i++){

                String line = lines.get(i);
                if (line.isEmpty()) {
                    continue;
                }
                String[] elements = line.split(";");
                String nim = elements[0];
                String nama = elements[1];
                String alamat = elements[2];
                String semester = elements[3];
                String sks = elements[4];
                String ipk = elements[5];
```

```
            Mahasiswa mhs = new Mahasiswa(nim, nama, alamat, semester, sks,
ipk);
            data.add(mhs);
          }
      } catch (IOException e) {
          throw new RuntimeException(e);
      }
      dataList.clear();
      dataList.addAll(data);
   }
   private void tambahData(){
      if (tNIM.getText().isEmpty() || tNama.getText().isEmpty() ||
tAlamat.getText().isEmpty() ||
            tSemester.getText().isEmpty() || tSKS.getText().isEmpty() ||
tIPK.getText().isEmpty()) {

          System.out.println("Semua field harus diisi!");
          return;
      }
      Mahasiswa mahasiswa = new
Mahasiswa(tNIM.getText(),tNama.getText(),tAlamat.getText(),tSemester.getText
(),tSKS.getText(),tIPK.getText());
      dataList.add(mahasiswa);
      clearField();
      simpanDataKeCSV();
   }
   private void clearField(){
      tNIM.clear();
      tNama.clear();
      tAlamat.clear();
      tSemester.clear();
      tSKS.clear();
      tIPK.clear();

   }

   private void simpanDataKeCSV() {
      try (BufferedWriter writer = new BufferedWriter(new
FileWriter("database.csv"))){
          writer.write("NIM;NAMA;ALAMAT;SEMESTER;SKS;IPK\n");
          for (Mahasiswa mhs : dataList){
```

```
writer.write(String.format("%s;%s;%s;%s;%s\n",mhs.getNim(),mhs.getNama(),
mhs.getAlamat(), mhs.getSemester(),mhs.getSks(),mhs.getIpk()));
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private void pilihMahasiswa(){
        selectedMahasiswa = table.getSelectionModel().getSelectedItem();
        if (selectedMahasiswa!= null){
            tNIM.setText(selectedMahasiswa.getNim());
            tNama.setText(selectedMahasiswa.getNama());
            tAlamat.setText(selectedMahasiswa.getAlamat());
            tSemester.setText(selectedMahasiswa.getSemester());
            tSKS.setText(selectedMahasiswa.getSks());
            tIPK.setText(selectedMahasiswa.getIpk());
        }
    }
    private void  editData(){
        if(selectedMahasiswa !=null){
            selectedMahasiswa.setNim(tNIM.getText());
            selectedMahasiswa.setNama(tNama.getText());
            selectedMahasiswa.setAlamat(tAlamat.getText());
            selectedMahasiswa.setSemester(tSemester.getText());
            selectedMahasiswa.setSks(tSKS.getText());
            selectedMahasiswa.setIpk(tIPK.getText());
            table.refresh();
            clearField();
            simpanDataKeCSV();
        }
    }
    private void hapusData(){
        if(selectedMahasiswa != null){
            dataList.remove(selectedMahasiswa);
            clearField();
            simpanDataKeCSV();
        }
    }
}
```

Langkah 8

Run program yang telah kita buat.



CANVAS DAN GRAPHICSCONTEXT_____

Canvas adalah komponen grafis 2D di JavaFX yang memungkinkan kita untuk menggambar secara manual ke dalam area tampilan. Canvas menyediakan area kosong untuk menggambar secara bebas (low-level drawing). Bukan seperti tombol atau teks biasa, tapi kamu benar-benar menggambar pixel demi pixel. Ibaratnya seperti kanvas lukis tempat kita bisa gambar bentuk, garis, warna, gambar, atau bahkan animasi.

GraphicsContext adalah alat yang digunakan untuk menggambar ke dalam Canvas. Setelah kamu punya Canvas, kamu butuh GraphicsContext untuk bisa menggambar di atasnya.

Pada program berikut ini kita akan berlatih membuat sebuah aplikasi komputer grafis sederhana untuk menampilkan tulisan “Informatika Unsulbar”.
Pertama kita akan membuat sebuah class baru bernama class Tulisan.

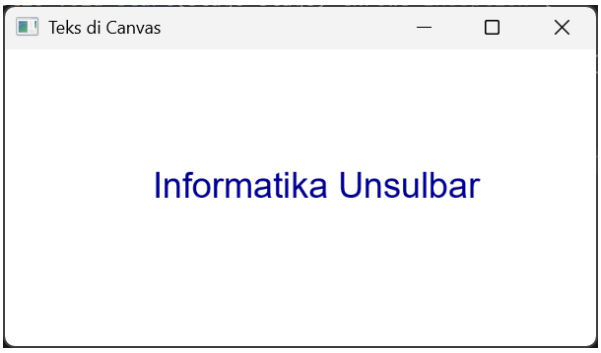
```
package CanvasDanGrapihicsContext;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
```

```
public class Tulisan extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        Canvas canvas = new Canvas(400, 200);
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(Color.DARKBLUE);
        gc.setFont(javafx.scene.text.Font.font("Arial", 24));
        gc.fillText("Informatika Unsulbar", 100, 100);

        Pane root = new Pane(canvas);
        Scene scene = new Scene(root, 400, 200);
        stage.setTitle("Teks di Canvas");
        stage.setScene(scene);
        stage.show();
    }
}
```

Pada class Tulisan kita akan melakukan extends ke class Application, kemudian kita melakukan inisalisasi untuk Canvas dan GraphicsContext, pada contoh di atas kita sudah dapat mengubah warna tulisan menjadi dark blue, dengan font style arial dengan ukuran 24 dan kita menggambar text “Informatika Unsulbar” pada posisi koordinat (100,100).

```
gc.setFill(Color.DARKBLUE);
gc.setFont(javafx.scene.text.Font.font("Arial", 24));
gc.fillText("Informatika Unsulbar", 100, 100);
```

Berikut ini hasil Aplikasi GUI yang dibuat saat aplikasi dijalankan:



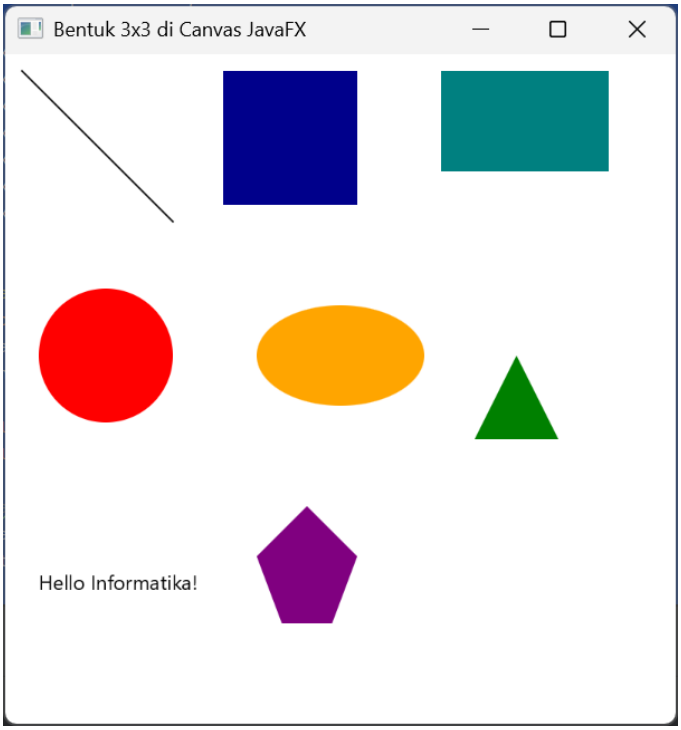
Sangat sederhana dan mudah sekali bukan???!!! Ini adalah langkah awal kita untuk membangun Rich Application Interface.

Selanjutnya kita akan membuat sebuah objek grafis lain nya dengan bentuk dan warna yang berbeda.

```
package CanvasDanGrapihicsContext;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class Bentuk extends Application {
    @Override
    public void start(Stage primaryStage) {
        Canvas canvas = new Canvas(400, 400);
        GraphicsContext gc = canvas.getGraphicsContext2D();

        double cellWidth = 120;
        double cellHeight = 120;

        // Garis (baris 1 kolom 1)
        gc.setStroke(Color.BLACK);
        gc.strokeLine(10, 10, 100, 100);

        // Persegi (baris 1 kolom 2)
        gc.setFill(Color.DARKBLUE);
        gc.fillRect(130, 10, 80, 80);

        // Persegi Panjang (baris 1 kolom 3)
        gc.setFill(Color.TEAL);
        gc.fillRect(260, 10, 100, 60);

        // Lingkaran (baris 2 kolom 1)
        gc.setFill(Color.RED);
        gc.fillOval(20, 140, 80, 80); // width == height → lingkaran

        // Oval (baris 2 kolom 2)
        gc.setFill(Color.ORANGE);
        gc.fillOval(150, 150, 100, 60);
```

```
        // Segitiga (baris 2 kolom 3)
        double[] xSegi3 = {280, 330, 305};
        double[] ySegi3 = {230, 230, 180};
        gc.setFill(Color.GREEN);
        gc.fillPolygon(xSegi3, ySegi3, 3);

        // Teks (baris 3 kolom 1)
        gc.setFill(Color.BLACK);
        gc.fillText("Hello Informatika!", 20, 320);

        // Polygon (baris 3 kolom 2) – segi lima
        double[] xPoly = {150, 180, 210, 195, 165};
        double[] yPoly = {300, 270, 300, 340, 340};
        gc.setFill(Color.PURPLE);
        gc.fillPolygon(xPoly, yPoly, 5);

        Pane root = new Pane(canvas);
        Scene scene = new Scene(root, 400, 400);
        primaryStage.setTitle("Bentuk 3x3 di Canvas JavaFX");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Pada saat kita menjalankan program diatas kita akan mendapatkan hasil seperti gambar berikut ini :



Masih ada banyak lagi objek grafis yang dapat kita buat menggunakan Java Canvas dan GraphicsContext ini kita dapat menggambar titik, garis, kotak, oval, polygon, dll. Kita juga dapat melakukan pengaturan warna, stroke, transparansi, translasi dan masih banyak pengaturan lain lagi yang dapat kita berikan untuk membuat objek grafis dan mengembangkan Custom GUI. Mahasiswa diharapkan dapat mengembangkan diri lebih jauh dengan mempelajari materi-materi berikut:

1. Dokumentasi Java GraphicsContext : https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html

MEMBUAT ANIMASI DENGAN JAVAFX

Dalam JavaFX, animasi dibuat berdasarkan **ilusi gerakan** yang dihasilkan dari **perubahan properti objek secara terus-menerus**. Properti yang bisa diubah tidak terbatas pada posisi, tetapi juga bisa berupa ukuran, warna, transparansi, rotasi, dan sebagainya. Inti dari animasi ini adalah proses *looping* yang berfungsi memperbaharui dan menggambar ulang objek secara berkala, sehingga menghasilkan efek seolah-olah objek sedang bergerak. Dalam dunia pengembangan aplikasi grafis, proses ini dikenal sebagai **animation loop** atau **game loop**, yang juga menjadi dasar dari pengembangan game 2D maupun 3D. JavaFX menyediakan beberapa cara untuk membuat animasi, salah satunya adalah menggunakan kelas **AnimationTimer**. Di JavaFX, metode **handle()** dari

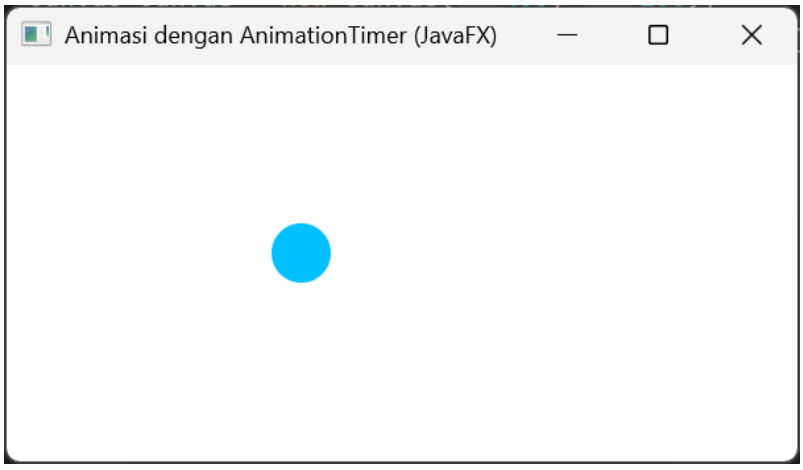
AnimationTimer akan dipanggil terus-menerus oleh sistem (biasanya sekitar 60 kali per detik), dan disinilah logika animasi dijalankan—termasuk untuk memperbaharui posisi objek dan menggambarnya kembali di layar.

Berikut ini kita akan membuat animasi sederhana untuk menggerakkan objek lingkaran secara horizontal menggunakan AnimationTimer.

```
package CanvasDanGrapihicsContext;

import javafx.animation.AnimationTimer;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class Animasi extends Application {
    double x = 0;

    @Override
    public void start(Stage primaryStage) {
        Canvas canvas = new Canvas(400, 200);
        GraphicsContext gc = canvas.getGraphicsContext2D();

        // Loop dengan AnimationTimer
        AnimationTimer timer = new AnimationTimer() {
            @Override
            public void handle(long now) {
                update();
                render(gc);
            }
        };
        timer.start();

        Scene scene = new Scene(new StackPane(canvas), 400, 200);
        primaryStage.setTitle("Animasi dengan AnimationTimer (JavaFX)");
```

```
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void update() {
        x += 2;
        if (x > 400) x = 0;
    }

    private void render(GraphicsContext gc) {
        gc.clearRect(0, 0, 400, 200); // hapus gambar sebelumnya
        gc.setFill(Color.DEEPSKYBLUE);
        gc.fillOval(x, 80, 30, 30); // gambar lingkaran
    }
}
```

Program ini dilakukan dengan menggunakan Class AnimationTimer dan melakukan override pada method handle() yang mana method handle() memanggil method update() dan method rander() yang kita buat untuk melakukan perubahan koordinat x secara terus menerus dengan cara menghapus gambar lama dan membuat gambar baru dengan koordinat baru.



Cara membuat animasi dengan menggunakan AnimationTimer di javaFX tergolong lebih sederhana, meskipun demikian tetap diperlukan pemahaman yang kuat tentang konsep-konsep dasar Pemrograman Berorientasi Objek (PBO) agar dapat memanfaatkannya secara maksimal. Konsep seperti inheritance, polymorphism, encapsulation, serta teknik seperti overriding dan overloading

akan sangat membantu dalam menyusun animasi yang fleksibel, terstruktur, dan mudah dikembangkan lebih lanjut.

MEMBUAT PROGRAM UNTUK INTERAKSI GRAFIS YANG LEBIH KOMPLEKS _____

Pada latihan berikut ini akan mengembangkan class Animasi yang telah kita buat sebelumnya, kita akan menyusun class Animasi ini agar dapat menjalankan AnimationTimer sekaligus berinteraksi dengan input melalui keyboard. Kita membutuhkan latihan ini karena dalam pengembangan game lanjutan tentu kita tidak hanya akan membutuhkan animasi tetapi kita juga membutuhkan interaksi dengan pengguna yang memasukkan input melalui keyboard, mouse, dll.

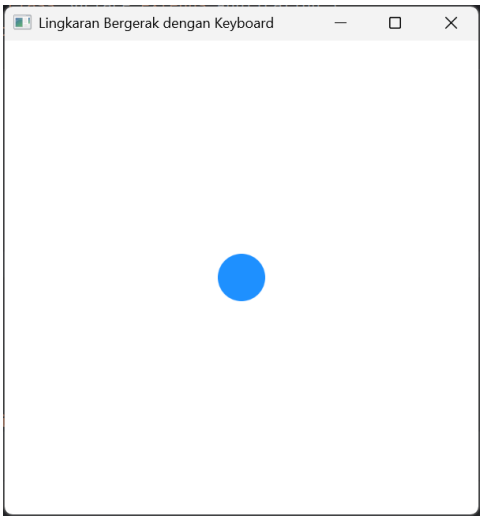
Pada program ini kita akan menambah input keyboard dan mengubah sedikit dari program sebelum nya.

```
package CanvasDanGrapihicsContext;

import javafx.animation.AnimationTimer;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class Surface extends Application {
    private double x = 200;
    private double y = 200;
    private final double radius = 20;
    private final double step = 3;

    // Arah gerakan
    private boolean up, down, left, right;

    @Override
    public void start(Stage primaryStage) {
        Canvas canvas = new Canvas(400, 400);
```

```
    GraphicsContext gc = canvas.getGraphicsContext2D();

    Scene scene = new Scene(new StackPane(canvas), 400, 400);

    // Event untuk menekan tombol
    scene.setOnKeyPressed(e -> {
        if (e.getCode() == KeyCode.UP) up = true;
        if (e.getCode() == KeyCode.DOWN) down = true;
        if (e.getCode() == KeyCode.LEFT) left = true;
        if (e.getCode() == KeyCode.RIGHT) right = true;
    });

    // Event untuk melepas tombol
    scene.setOnKeyReleased(e -> {
        if (e.getCode() == KeyCode.UP) up = false;
        if (e.getCode() == KeyCode.DOWN) down = false;
        if (e.getCode() == KeyCode.LEFT) left = false;
        if (e.getCode() == KeyCode.RIGHT) right = false;
    });

    // Loop animasi menggunakan AnimationTimer
    AnimationTimer timer = new AnimationTimer() {
        @Override
        public void handle(long now) {
            update();
            draw(gc);
        }
    };
    timer.start();

    primaryStage.setTitle("Lingkaran Bergerak Otomatis");
    primaryStage.setScene(scene);
    primaryStage.show();
}

// Update posisi berdasarkan arah
private void update() {
    if (up) y -= step;
    if (down) y += step;
    if (left) x -= step;
    if (right) x += step;
```

```
        // Batas layar (opsional)
        x = Math.max(radius, Math.min(400 - radius, x));
        y = Math.max(radius, Math.min(400 - radius, y));
    }

    // Gambar ulang
    private void draw(GraphicsContext gc) {
        gc.clearRect(0, 0, 400, 400);
        gc.setFill(Color.DODGERBLUE);
        gc.fillOval(x - radius, y - radius, radius * 2, radius * 2);
    }
}
```

dan program GUI yang dihasilkan adalah sebagai berikut:



Coba lakukan pengujian dengan menekan tombol panah di keyboard dan lihat reaksinya. Jika source code anda benar, maka lingkaran biru akan dapat digerakkan menggunakan tombo/kombinasi tombol panah yang ada di keyboard. Untuk materi praktikum JavaFX kita cukupkan sampai di sini. Mahasiswa diharapkan dapat memperdalam dan mengembangkan lebih lanjut dengan memperbanyak latihan dan mengerjakan tugas praktikum.

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. TugasPraktikum

Pelajari dan tingkatkan kemampuan anda dalam mengembangkan custom Graphical User Interface menggunakan JavaFX. Anda diminta untuk membuat Program GUI Java yang menarik untuk menyelesaikan masalah berikut:

1. Buatlah aplikasi JavaFX sederhana yang menampilkan label dan tombol. Ketika tombol ditekan, ubah teks label menjadi "Tombol ditekan!".
2. Buatlah program GUI Java untuk mengkonversi satuan panjang Sistem Internasional (km, hm, dam, m, dm, cm, mm)
3. Buatlah program GUI Java untuk mengkonversi nilai Mata Kuliah dari Nilai Angka (interval 0-100) ke Nilai Huruf (A, A-, B+, B, B-, C, D, E) dan sebaliknya
4. Buatlah program GUI Java untuk mengkonversi nilai Mata Uang Dollar ke Rupiah dan sebaliknya
5. Buatlah program GUI Java untuk membaca dan menyimpan file text download file csv dari: https://gist.github.com/FOSSALGO/665e21fd5761a5a1f8a8c8778e211e25 Buat agar program dapat melakukan fitur manampilkan, menambah, mengubah, menghapus dan menyimpan data. Buat semenarik mungkin !
6. Buatlah program GUI java untuk game snake, puzzle, breakout, tetris, pacman, sokoban, dan space invaders. Referensi dapat dilihat pada situs https://zetcode.com/javagames/ gunakan kreatifitas semaksimal mungkin untuk memodifikasi game tersebut.

PRAKTIKUM 5: ALGORITHMS AND DATA STRUCTURES I

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator penilaian

1. Mahasiswa mampu menjelaskan konsep dasar struktur data Array, ArrayList, Vector, HashSet, dan HashMap.
2. Mahasiswa mampu mengidentifikasi method-method yang ada pada struktur data Array, ArrayList, Vector, HashSet, dan HashMap pada bahasa pemrograman Java.
3. Mahasiswa mampu mengidentifikasi perbedaan antara struktur data Array, ArrayList, Vector, HashSet, dan HashMap pada bahasa pemrograman Java.
4. Mahasiswa mampu mengimplementasikan algoritma sorting pada array atau koleksi data lainnya dalam bahasa pemrograman Java.
5. Mahasiswa mampu menjelaskan dan membuat contoh penggunaan array multidimensi (1D, 2D, dan 3D).

B. Tugas Pendahuluan

1. Jelaskan konsep dasar struktur data Array, ArrayList, Vector, HashSet, HashMap dan Iterator.
2. Lakukan identifikasi dan jelaskan kegunaan dari method-method yang tersedia pada struktur data Array, ArrayList, Vector, HashSet, HashMap dan Iterator yang tersedia di JDK.
3. Jelaskan perbedaan antara Array, ArrayList, dan Vector dari segi kapasitas, sinkronisasi, dan fleksibilitas.
4. Jelaskan pengertian dari algoritma sorting, tujuan penggunaannya dalam struktur data, serta bagaimana konsep dasar algoritma sorting bekerja dalam mengurutkan data.
5. Jelaskan perbedaan antara teknik sorting. Uraikan karakteristik umum, contoh algoritma yang termasuk ke dalam masing-masing kategori, serta kelebihan dan kekurangannya.

6. Buatlah array 1D dan 2D bertipe int, String, dan double, lengkap dengan nilai elemen-elemennya lalu print elemen-elemen yang ada.
7. Buatlah program Java untuk menyimpan dan menampilkan data menggunakan struktur data Array, ArrayList dan Vector. Input dilakukan dengan Scanner dan tampilkan semua datanya.
8. Buatlah program Java untuk menyimpan data berikut menggunakan Vector dan ArrayList: 12, 46.4138729, "Gajah", 'j', 24, '#', "Jerapah", 22.7 .Kemudian tampilkan semua elemen ke console.
9. Buatlah program Java menggunakan ArrayList yang hanya dapat menyimpan data bertipe String, dengan input dari keyboard hingga pengguna memasukkan string "EOF". Tampilkan semua data yang tersimpan.
10. Buatlah program Java untuk menyimpan dan menampilkan data berikut menggunakan HashSet: Apple, Banana, Orange, Apple, Mango. Perhatikan bahwa HashSet tidak menyimpan elemen yang duplikat.
11. Buatlah program Java untuk menyimpan dan menampilkan pasangan data (key-value) menggunakan HashMap. Gunakan input dari Scanner. Misalnya: NIM dan Nama Mahasiswa.
12. Buatlah program Java untuk mengiterasi isi dari ArrayList dan HashSet menggunakan Iterator, lalu tampilkan hasilnya.
13. Implementasikan salah satu algoritma sorting untuk mengurutkan array bertipe int, lalu tampilkan array sebelum dan sesudah diurutkan.

C. Ringkasan Materi

Pada praktikum kali ini kita akan berlatih menggunakan Struktur Data Non-primitif: Array, ArrayList, Vector, HashSet, HashMap, Iterator dan Algoritma Sorting.

ARRAY_____

Ada banyak cara mendeklarasikan array di Java. Array dideklarasikan menggunakan pasangan tanda kurung siku "[]" yang menyatakan dimensi array. Jika menggunakan satu pasang kurung siku "[]" berarti array yang dideklarasikan adalah array satu dimensi (1D), sedangkan jika menggunakan dua pasang tanda kurung siku "[][]" berarti array yang dideklarasikan adalah array 2D, demikian seterusnya "[][][]" untuk array 3D. Kita dapat membuat array dari semua tipe data Primitif dan Non-primitif. Berikut ini contoh-contoh deklarasi array:

```
//Array 1D menggunakan tipe data primitif------
byte[] data01;
```

```
short[] data02;
int[] data03;
long[] data04;
float[] data05;
double[] data06;
char[] data07;
boolean[] data08;

//Array 1D menggunakan tipe data primitif------
byte data11[];
short data12[];
int data13[];
long data14[];
float data15[];
double data16[];
char data17[];
boolean data18[];

//Array 2D menggunakan tipe data primitif------
byte[][] data21;
short[][] data22;
int[][] data23;
long[][] data24;
float[][] data25;
double[][] data26;
char[][] data27;
boolean[][] data28;

//Array 2D menggunakan tipe data primitif------
byte data31[][];
short data32[][];
int data33[][];
long data34[][];
float data35[][];
double data36[][];
char data37[][];
boolean data38[][];

//Array 2D menggunakan tipe data primitif------
byte[] data41[];
short[] data42[];
int[] data43[];
```

```
long[] data44[];
float[] data45[];
double[] data46[];
char[] data47[];
boolean[] data48[];

//Array 3D menggunakan tipe data primitif------
byte[][][] data51;
short[][][] data52;
int[][][] data53;
long[][][] data54;
float[][][] data55;
double[][][] data56;
char[][][] data57;
boolean[][][] data58;

//Array 4D menggunakan tipe data primitif------
byte[][][][] data61;
short[][][][] data62;
int[][][][] data63;
long[][][][] data64;
float[][][][] data65;
double[][][][] data66;
char[][][][] data67;
boolean[][][][] data68;

//Inisialisasi Array 1D dengan tipe data primitif------
//angka di dalam kurung siku setelah kata new dan tipe data adalah panjang
(length/size) dari array
byte[] data71 = new byte[18];// size = 18
short[] data72 = new short[20];// size = 20
int[] data73 = new int[128];// size = 128
long[] data74 = new long[1024];// size = 1024
float[] data75 = new float[212];// size = 212
double[] data76 = new double[97];// size = 97
char[] data77 = new char[14];// size = 14
boolean[] data78 = new boolean[654];// size = 654

//menyimpan nilai ke dalam array byte 1D
byte[] data81 = new byte[4];
data81[0] = 120;//set nilai elemen-0 = 120
data81[1] = 9;//set nilai elemen-1 = 9
```

```
data81[2] = 127;//set nilai elemen-2 = 127
data81[3] = -128;//set nilai elemen-3 = -128

//menyimpan nilai ke dalam array short 1D
short[] data82 = new short[3];
data82[0] = 2;
data82[1] = 8;
data82[0] = 5;
data82[2] = 73;
data82[1] = 6829;

//menyimpan nilai ke dalam array double 1D
double[] data86 = new double[10];
data86[0] = 0;
data86[1] = 1;
data86[2] = 2;
data86[3] = 3;
data86[9] = 18.76;
data86[4] = 4;
data86[5] = 5;
data86[6] = 6;
data86[7] = 7;
data86[8] = 8.0;
data86[9] = 9;

//menyimpan nilai ke dalam array char 1D
char[] data87 = new char[4];
data87[3] = '@';
data87[2] = 'b';
data87[1] = '*';
data87[0] = '#';

//menyimpan nilai ke dalam array boolean 1D
boolean[] data88 = new boolean[6];
data88[0] = true;
data88[3] = false;
data88[1] = false;
data88[2] = true;
data88[4] = false;
data88[3] = true;
data88[5] = true;
```

```
//Array 1D menggunakan tipe data non-primitif------
 //Di Java harus dapat membedakan int dan Integer, long dengan Long
Byte[] data91;
Short[] data92;
Integer[] data93;
Long[] data94;
Float[] data95;
Double[] data96;
Character[] data97;
Boolean[] data98;
```

Inisialisasi array juga dapat dilakukan menggunakan tanda kurung kurawal berpasangan. Tetapi cara ini hanya dapat dilakukan satu kali saja pada saat deklarasi array

```
//inisialisasi nilai elemen array menggunakan kurung kurawal
byte[] data101 = {1,36, 97, 123, 17, 105, -32, 9, -86};
short[] data102 = {2,6,234,123,256,512};
int[] data103 = {6,2,9,34,2987,52};
long[] data104 = {21,54,79,21567,215};
float[] data105 = {9876,2076,12.68f,3.14f,8.31f,3.0f};
double[] data106 = {9876,2076,12.68,3.14,8.31,3.0};
char[] data107 = {'2','8','@','#','%','$','('};
boolean[] data108 = {true,false,false,true,false};

int[] data109 = new int[]{3, 92, 6, 17, 58};
double[] data106 = new double[]{9876,2076,12.68,3.14,8.31,3.0};
```

jika digunakan pada array multi-dimensi maka pasangan kurung kurawalnya juga berlapis. Berikut ini contoh penggunaan kurung kurawal untuk deklarasi array multi dimensi:

```
//inisialisasi nilai elemen array multidimensi menggunakan kurung kurawal
byte[][] data201 = {
 {1, 36, 97, 123, 17, 105, -32, 9, -86},
 {},
 {3, 7, 85, 12},
 {5, 6},
 {7, 4, 3, 2}
};
short[][] data202 = {
```

```
 {24, 65, 3218, 943},
 {7, 2, 9},
 {},
 {2, 8, 12},
 {8, 6}
};
int[][] data203 = {};
int[][] data2031 = {
 {34},
 {12, 34, 56},
 {12, 34, 56, 67},
 {12, 34, 56, 67, 89, 10}
};
long[][] data204 = {
 {1, 2, 3, 4, 5, 6, 7, 8, 9},
 {1, 2, 3, 4, 5, 6, 7, 8},
 {1, 2, 3, 4, 5, 6, 7},
 {1, 2, 3, 4, 5, 6},
 {1, 2, 3, 4, 5},
 {1, 2, 3, 4},
 {1, 2, 3},
 {1, 2},
 {1},
 {}
};
float[][] data205 = {
 {2f, 34.6f, 12f},
 {31, 12.3f, 87f},
 {35, 91.5f, 63f}
};
double[][] data206 = {
 {},
 {3.56, 21.765, 264.287},
 {6.23},
 {3, 4, 5, 6, 7}
};
char[][] data207 = {{'2','@'},{'#','7','#'},{},{'!','$','%'}};
boolean[][] data208 =
{{true,false},{false,false},{true,true,false},{true},{false,true}};
Integer[][] data209 = {
 {2,5,6},
 {2,1,7},
```

```
 {4,5}
};
```

Dari contoh-contoh di atas juga kita dapat melihat bahwa jumlah kolom di setiap barisnya tidak harus sama. Jadi baris pertama boleh terdiri dari dua kolom, baris ketiga empat kolom, bahkan kita dapat membuat baris yang kosong menggunakan { }.

Kita juga dapat membuat array bertipe string ataupun class yang kita buat sendiri. Misalnya kita membuat array menggunakan class Mahasiswa

```
//array menggunakan tipe String
String[] alatTulis = {
   "Buku", "Pulpen", "Pensil", "penghapus", "Penggaris"};

String[][] buah = {
   {"Durian", "Rambutan", "Nangka"},
   {"Mangga", "Jeruk", "Pisang"},
   {"Salak", "Delima"}
};

//array menggunakan class
Mahasiswa[] mhsKelasK = new Mahasiswa[3];
mhsKelasK[0] = new Mahasiswa("D01234", "I Komeng", 87);
mhsKelasK[1] = new Mahasiswa("D01235", "Shine Meng", 94);
mhsKelasK[2] = new Mahasiswa("D01236", "Khan Demeng", 84);

Mahasiswa[] mhsKelasL = {new Mahasiswa("D01234", "I Komeng", 87), new
Mahasiswa("D01235", "Shine Meng", 94), new Mahasiswa("D01236", "Khan
Demeng", 84)
};

Mahasiswa[][] mhsKelasM = {
   {new Mahasiswa("D01234", "I Komeng", 87),new Mahasiswa("D01235", "Shine
Meng", 94),new Mahasiswa("D01236", "Khan Demeng", 84)},
   {},
   { new Mahasiswa("D01237", "Samar", 97), new Mahasiswa("D01238", "Okto", 84)}
};
```

Untuk menampilkan nilai elemen-elemen array dapat dilakukan menggunakan class **Arrays**, looping **for each** dan looping for standar. Untuk array multi-dimensi tentu diperlukan loop bersarang (nested loop).

```
//array menggunakan tipe String
String[] alatTulis = {"Buku", "Pulpen", "Pensil", "penghapus", "Penggaris"};
//mencetak menggunakan class Arrays
System.out.println(Arrays.toString(alatTulis));

System.out.println("-----------------------------------------");
//mencetak menggunakan looping for each
for (String value : alatTulis) {
    System.out.print(value + "; ");
}
System.out.println();

System.out.println("-----------------------------------------");
//mencetak menggunakan for standar
System.out.print("[");
for (int i = 0; i < alatTulis.length; i++) {
    if (i > 0) {
        System.out.print(", ");
    }
    System.out.print(alatTulis[i]);
}
System.out.println("]");

String[][] buah = {
    {"Durian", "Rambutan", "Nangka"},
    {"Mangga", "Jeruk", "Pisang"},
    {"Salak", "Delima"}
};

System.out.println("-----------------------------------------");
//mencetak elemen array multi-dimensi
for (int i = 0; i < buah.length; i++) {
    System.out.print("[");
    for (int j = 0; j < buah[i].length; j++) {
        if (j > 0) {
            System.out.print(", ");
        }
        System.out.print(buah[i][j]);
```

```
    }
    System.out.println("]");
}
```

Source code lengkap untuk deklarasi array dapat dilihat disini:
https://gist.github.com/FOSSALGO/12493c69e78a39b0e47822917f6cf6c9

VECTOR DAN ARRAYLIST_____

Berbeda dengan array yang **size**-nya statis, size pada **Vector** dan **ArrayList** bersifat dinamis sehingga dapat bertambah dan berkurang jumlah elemen yang disimpan di dalamnya. Bukan hanya nilai elemennya yang dapat diubah, tetapi size (banyaknya elemen yang tersimpan) juga dapat berubah atau diatur ulang. Selain itu pada Vector dan ArrayList, jika kita tidak menetapkan tipe yang akan kita simpan, maka kita diperbolehkan menyimpan bermacam-macam tipe object ke dalam sebuah Vector ataupun ArrayList yang sama. Berikut ini beberapa contoh deklarasi Vector dan ArrayList.

Deklarasi Vector

```
//deklarasi Vector
Vector data001;

//deklarasi dan inisialisasi Vector
Vector data002 = new Vector();
data002.add(36);
data002.add("Senin");
data002.add(67.89f);
data002.add(89213);
data002.add(new Mahasiswa("D01235", "Shine Meng", 94));
System.out.println("Vector data002: " + data002);

data002.add("Informatika"); //menambahkan elemen ke vector data002
System.out.println("Vector data002: " + data002);

//deklarasi dan inisialisasi Vector menggunakan class Wrapper Integer
Vector<Integer> data003 = new Vector<>();
data003.add(25);
data003.add(128);
data003.add(256);
data003.add(512);
System.out.println("Vector of Integer: " + data003);

data003.set(1, 678);
```

```
System.out.println("Vector of Integer: " + data003);

//deklarasi dan inisialisasi Vector menggunakan class Wrapper Character
Vector<Character> data004 = new Vector<>();
data004.add('@');
data004.add('#');
data004.add('^');
data004.add('&');
data004.add('%');
data004.add('D');
System.out.println("Vector of Character: " + data004);

int sizeVectorData004 = data004.size();
System.out.println("Size Data004: " + sizeVectorData004);

//deklarasi dan inisialisasi Vector menggunakan class Wrapper Double
Vector<Double> data005 = new Vector<>();
data005.add(12.0);
data005.add(7.43);
data005.add(6.25);
data005.add(198.3251);
System.out.println("Vector of Double: " + data005);

//print Vector menggunakan loop for
System.out.print("Vector of Double: [");
for (int i = 0; i < data005.size(); i++) {
    if (i > 0) {
        System.out.print(", ");
    }
    System.out.print(data005.get(i));
}
System.out.println("]");

//print vector menggunakan loop for each
System.out.print("Vector of Double: [");
int k = 0;
for (Double d : data005) {
    if (k > 0) {
        System.out.print(", ");
    }
    System.out.print(d);
    k++;
```

```
}
System.out.println("]");

//deklarasi dan inisialisasi Vector menggunakan class Mahasiswa
Vector<Mahasiswa> data006 = new Vector<>();
data006.add(new Mahasiswa("D01234", "I Komeng", 87));
data006.add(new Mahasiswa("D01235", "Shine Meng", 94));
data006.add(new Mahasiswa("D01236", "Khan Demeng", 84));
System.out.println("Vector data006: " + data006);

//membuat Vector yang berisi Array
Vector<int[]> data007 = new Vector<>();
int[] arr0 = {2, 3, 7, 5};
data007.add(arr0);
int[] arr1 = {9, 3, 2, 65, 71, 215};
data007.add(arr1);
int[] arr2 = {6, 3, 8};
data007.add(arr2);

//membuat Vector yang berisi Vector
Vector data008 = new Vector();
data008.add(data002);
data008.add(data003);
data008.add(data004);
data008.add(data005);
data008.add(data006);
data008.add(data007);
System.out.println("Vector data008: " + data008);
```

Source code lengkap untuk deklarasi Vector dapat dilihat disini:
https://gist.github.com/FOSSALGO/4f1ba5729afe00b1ae69cdeec92b0d86

Deklarasi ArrayList

```
//deklarasi ArrayList
ArrayList data001;

//deklarasi dan inisialisasi ArrayList
ArrayList data002 = new ArrayList();
data002.add(36);
data002.add("Senin");
data002.add(67.89f);
data002.add(89213);
data002.add(new Mahasiswa("D01235", "Shine Meng", 94));
```

```
System.out.println("ArrayList data002: " + data002);

data002.add("Informatika"); //menambahkan elemen ke ArrayList data002
System.out.println("ArrayList data002: " + data002);

//deklarasi dan inisialisasi ArrayList menggunakan class Wrapper Integer
ArrayList<Integer> data003 = new ArrayList<>();
data003.add(25);
data003.add(128);
data003.add(256);
data003.add(512);
System.out.println("ArrayList of Integer: " + data003);

data003.set(1, 678);
System.out.println("ArrayList of Integer: " + data003);

//deklarasi dan inisialisasi ArrayList menggunakan class Wrapper Character
ArrayList<Character> data004 = new ArrayList<>();
data004.add('@');
data004.add('#');
data004.add('^');
data004.add('&');
data004.add('%');
data004.add('D');
System.out.println("ArrayList of Character: " + data004);

int sizeArrayListData004 = data004.size();
System.out.println("Size Data004: " + sizeArrayListData004);

//deklarasi dan inisialisasi ArrayList menggunakan class Wrapper Double
ArrayList<Double> data005 = new ArrayList<>();
data005.add(12.0);
data005.add(7.43);
data005.add(6.25);
data005.add(198.3251);
System.out.println("ArrayList of Double: " + data005);

//print ArrayList menggunakan loop for
System.out.print("ArrayList of Double: [");
for (int i = 0; i < data005.size(); i++) {
    if (i > 0) {
        System.out.print(", ");
```

```
    }
    System.out.print(data005.get(i));
}
System.out.println("]");

//print ArrayList menggunakan loop for each
System.out.print("ArrayList of Double: [");
int k = 0;
for (Double d : data005) {
    if (k > 0) {
        System.out.print(", ");
    }
    System.out.print(d);
    k++;
}
System.out.println("]");

//deklarasi dan inisialisasi ArrayList menggunakan class Mahasiswa
ArrayList<Mahasiswa> data006 = new ArrayList<>();
data006.add(new Mahasiswa("D01234", "I Komeng", 87));
data006.add(new Mahasiswa("D01235", "Shine Meng", 94));
data006.add(new Mahasiswa("D01236", "Khan Demeng", 84));
System.out.println("ArrayList data006: " + data006);

//membuat ArrayList yang berisi Array
ArrayList<int[]> data007 = new ArrayList<>();
int[] arr0 = {2, 3, 7, 5};
data007.add(arr0);
int[] arr1 = {9, 3, 2, 65, 71, 215};
data007.add(arr1);
int[] arr2 = {6, 3, 8};
data007.add(arr2);

//membuat ArrayList yang berisi ArrayList
ArrayList data008 = new ArrayList();
data008.add(data002);
data008.add(data003);
data008.add(data004);
data008.add(data005);
data008.add(data006);
data008.add(data007);
System.out.println("ArrayList data008: " + data008);
```

Source code lengkap untuk deklarasi ArrayList dapat dilihat disini:
https://gist.github.com/FOSSALGO/d0711f774a18db5fd9f8671621aebd6b

Class Vector dan ArrayList berada di dalam package java.util. yaitu java.util.Vector dan java.util.ArrayList. dokumentasi lengkap untuk fields, constructor dan method yang tersedia pada Vector dan ArrayList dapat dilihat di Dokumentasi JDK Oracle:

Vector: https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html
ArrayList: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

HashSet_____

Struktur Data HashSet pada Java digunakan untuk menyimpan kumpulan item yang setiap item dalam kumpulan data tersebut bersifat unik (tidak ada yang sama). Jadi setiap elemen yang tersimpan di dalam HashSet itu unik. HashSet ini menggunakan terminology himpunan (set) dimana setiap anggota himpunan adalah bersifat unik. Class HashSet berada di package java.util.HashSet. anda dapat melihat dokumentasi lengkap tentang class HashSet di dokumentasi JDK:

https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html

contoh source code untuk menyimpan data menggunakan HashSet:

```
HashSet bendaUnik = new HashSet();
bendaUnik.add(1);
bendaUnik.add("Mahasiswa");
bendaUnik.add("Kamu");
bendaUnik.add(12.3456);
bendaUnik.add('@');
bendaUnik.add("Kamu");
bendaUnik.add('#');
System.out.println(bendaUnik);
```

Kita juga dapat menetapkan agar HashSet hanya dapat digunakan untuk menyimpan data bertipe String saja atau Integer saja dengan cara berikut:

```
HashSet<String> cars = new HashSet<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Toyota");
cars.add("Ford");
cars.add("BMW");
cars.add("Mazda");
cars.add("Toyota");
System.out.println(cars);
```

Berikut ini hasil output dari program di atas:

```
[Volvo, Toyota, Mazda, Ford, BMW]
```

Perhatikan bahwa meskipun kita telah berusaha menambahkan data mobil BMW dan Toyota sebanyak dua kali, ke dua data tersebut hanya di simpan satu kali saja dikarenakan sudah ada data yang sama yang tersimpan lebih dulu dan semua item di HashSet harus unik.

Method **contains( )** dapat digunakan untuk memeriksa apakah sebuah item sudah ada di HashSet atau belum

```
cars.contains("Mazda");
```

method **remove( )** dapat digunakan untuk menghapus item yang ada di HashSet

```
cars.remove("Volvo");
```

untuk membersihkan/mengosongkan HashSet dapat digunakan method **clear( )**

```
cars.clear();
```

berikut ini contoh source code penggunaan Struktur Data HashSet:

```
import java.util.HashSet;

public class TestHashSet {
    public static void main(String[] args) {
        HashSet bendaUnik = new HashSet();
        bendaUnik.add(1);
        bendaUnik.add("Mahasiswa");
        bendaUnik.add("Kamu");
        bendaUnik.add(12.3456);
        bendaUnik.add('@');
        bendaUnik.add("Kamu");
        bendaUnik.add('#');
        System.out.println(bendaUnik);

        HashSet<String> cars = new HashSet<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Toyota");
        cars.add("Ford");
        cars.add("BMW");
        cars.add("Mazda");
```

```
        cars.add("Toyota");
        System.out.println(cars);
        System.out.println("cars.contains(\"Mazda\"): " + cars.contains("Mazda"));
        cars.remove("Volvo");
        System.out.println("Setelah menghapus Volvo: " + cars);
        cars.clear();
        System.out.println("Setelah Clear " + cars);
    }
}
```

Tidak diberikan contoh yang lengkap pada modul ini tentang penggunaan HashSet untuk menyimpan tipe data yang lain, tetapi disini ada contoh tentang pembuatan HashSet dari String menggunakan HashSet. Praktikan diharapkan mampu menggunakan insting programmer untuk membuat HashSet menggunakan tipe generic yang lain: HashSet<Integer>,HashSet<Double>,HashSet<Mahasiswa>, HashSet<Character>, HashSet<Boolean>, HashSet<JButton>, dll.

HashMap_____

Di dalam struktur data HashMap, data disimpan menggunakan format pasangan **key** dan **value**. Jika pada struktur array, Vector, dan ArrayList elemen data (item) diakses menggunakan index (posisi) elemen yang bertipe int, nah pada HashMap, item-itemnya diakses menggunakan key yang bertipe object/class. HashMap dideklarasikan menggunakan **HashMap<Key,Value>**. Dimana Key dan Value berupa object (bertipe class). contoh:

```
HashMap<String, String> data001 = new HashMap<>();
HashMap<Integer, String> data002 = new HashMap<>();
HashMap<String, Integer> data003 = new HashMap<>();
HashMap<Integer, Integer> data004 = new HashMap<>();
HashMap<Integer, Mahasiswa> data005 = new HashMap<>();
```

Meskipun **value** di dalam HashMap tidak unik, tetapi **key** pada HashMap bersifat unik. Untuk menambahkan elemen ke dalam HashMap digunakan method **put(key, value)**.

```
HashMap<String, String> ibuKotaNegara = new HashMap<String, String>();
// tambahkan keys and values (Negara, Kota) menggunakan method put(key, value)
ibuKotaNegara.put("England", "London");
```

```
ibuKotaNegara.put("Germany", "Berlin");
ibuKotaNegara.put("Norway", "Oslo");
ibuKotaNegara.put("USA", "Washington DC");
System.out.println(ibuKotaNegara);
```

Untuk mengakses **value** pada HashMap dapat kita lakukan menggunakan **key** yang bersesuaian dengan value tersebut. Berikut ini contoh pemanggilan menggunakan key "USA":

```
System.out.println(ibuKotaNegara.get("USA"));
```

Kode di atas akan menampilkan String:

```
Washington DC
```

Untuk menghapus item dari HashMap dapat digunakan method **remove(Object key)** atau **remove(Object key, Object value)**. Kita juga dapat mengubah value yang tersimpan pada HashMap menggunakan method **replace(K key, V value)** dan **replace(K key, V oldValue, V newValue)**. Untuk mengetahui ada berapa banyak pasangan **key-value** yang tersimpan di hashMap dapat dilakukan menggunakan method **size( )**. Selain itu juga terdapat method keySet() dan values() yang dapat digunakan untuk memanggil semua Set **key** dan Collection **value** yang ada di HashSet secara terpisah.

Contoh lengkap untuk Latihan HashMap ini adalah sebagai berikut:

```
import java.util.HashMap;

public class TestHashMap {

   public static void main(String[] args) {
      // Create a HashMap object called ibuKotaNegara
      HashMap<String, String> ibuKotaNegara = new HashMap<String, String>();
      // tambahkan keys and values (Negara, Kota) menggunakan method put(key,value)
      ibuKotaNegara.put("England", "London");
      ibuKotaNegara.put("Germany", "Berlin");
      ibuKotaNegara.put("Norway", "Oslo");
      ibuKotaNegara.put("USA", "Washington DC");
      System.out.println(ibuKotaNegara);
      System.out.println(ibuKotaNegara.get("USA"));

      int s = ibuKotaNegara.size();
      System.out.println("Banyaknya Ibu Kota Negara: " + s);
```

```
        //hapus key = England
        ibuKotaNegara.remove("England");
        System.out.println(ibuKotaNegara);

        //menampilkan semua key yang ada di HashMap
        System.out.print("Key Set: ");
        for (String k : ibuKotaNegara.keySet()) {
            System.out.print(k + "; ");
        }
        System.out.println();

        //menampilkan semua Value yang ada di HashMap
        System.out.print("Values: ");
        for (String v : ibuKotaNegara.values()) {
            System.out.print(v + "; ");
        }
        System.out.println();

        //menampilkan pasangan key dan value
        for (String k : ibuKotaNegara.keySet()) {
            System.out.println("key: " + k + " value: " + ibuKotaNegara.get(k));
        }
        System.out.println();

        //clear
        ibuKotaNegara.clear();
        System.out.println(ibuKotaNegara);
    }
}
```

HashMap juga dapat dibuat menggunakan tipe data object yang lain untuk pasangan **key** dan **value** misalnya **<String, Integer>**:

```
public class TestHashMapLain {

    public static void main(String[] args) {

        //membuat object HashMap bernama people
        HashMap<String, Integer> people = new HashMap<String, Integer>();

        //put keys dan values (Name, Age)
```

```
        people.put("John", 32);
        people.put("Steve", 30);
        people.put("Angie", 33);

        //menampilkan isi dari HashMap
        for (String i : people.keySet()) {
            System.out.println("key: " + i + " value: " + people.get(i));
        }
    }
}
```

ITERATOR_____

Iterator adalah interface yang dapat digunakan untuk mengiterasi elemen-elemen pada object yang dibuat dari class yang meng-implement interface **java.util.Collection** dan **java.util.Map** (Collection Framework) seperti: Vector, ArrayList, HashSet, HashMap, Set, List, Queue, Deque, dan LinkedList. Array di java bukan bagian dari Collection Framewok sehingga tidak diiterasi menggunakan interface Iterator. **Iterator** di java bekerja seperti semacam kursor dalam mengiterasi elemen-elemen collection.

Dokumentasi untuk interface Iterator dapat dilihat di:
https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

Berikut ini contoh kode program untuk penggunaan Iterator di ArrayList:

```
import java.util.*;

public class TestJavaIterator {

    public static void main(String[] args) {
        ArrayList<String> namaKota = new ArrayList<String>();

        namaKota.add("Polewali Mandar");
        namaKota.add("Majene");
        namaKota.add("Mamuju");
        namaKota.add("Mamuju Utara");
        namaKota.add("Mamasa");

        // Iterator untuk meng-iterasi namaKota
        Iterator iterator = namaKota.iterator();
        System.out.println("Elemen-elemen namaKota : ");
```

```
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + "; ");
        }
        System.out.println();
    }
}
```

ALGORITMA SORTING_____

Algoritma Sorting mengacu pada penataan ulang larik atau daftar elemen tertentu menurut operator perbandingan pada elemen-elemen tersebut. Operator perbandingan digunakan untuk menentukan urutan baru elemen dalam struktur data masing-masing.

Dasar-dasar penyortiran, diantaranya:

- In-place Sorting: menggunakan ruang konstan untuk menghasilkan keluaran (hanya memodifikasi array yang diberikan).
- Internal Sorting: terjadi Ketika semua data ditempatkan di memori utama atau memori internal.
- External Sorting: Terjadi Ketika semua data yang perlu diurutkan tidak perlu ditempatkan di memori pada suatu waktu.
- Stable Sorting: Ketika dua item yang sama muncul dalam urutan yang sama pada data yang diurutkan seperti pada array.
- Hybrid Sorting: Ketika menggunakan lebih dari satu standar Algoritma Sorting untuk mengurutkan array.

Jenis-Jenis Teknik sortiran, diantaranya:

1. Comparison-based
   Pada algoritma pengurutan berbasis perbandingan, elemen-elemen yang ada akan dibandingkan satu sama lain untuk menentukan urutan yang tepat. Terdapat beberapa algoritma sorting yang termasuk ke dalam kategori Comparison-based yaitu diantaranya Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort dan Heap Sort.
   1) Bubble Sort
      Bubble Sort adalah algoritma pengurutan paling sederhana yang bekerja dengan cara menukar elemen-elemen yang berdekatan secara berulang jika urutannya salah.
      Berikut adalah contoh implementasi Algoritma Bubble Sort dalam program.

```
import java.util.Arrays;

public class BubbleSortExample {

    // Versi Bubble Sort yang telah dioptimalkan
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        boolean swapped;

        // Melakukan iterasi sebanyak n-1 kali
        for (int i = 0; i < n - 1; i++) {
            swapped = false;

            // Melakukan perbandingan dan penukaran elemen
berdekatan
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Menukar elemen jika urutannya salah
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }

            // Jika tidak ada elemen yang ditukar, maka array
sudah terurut
            if (!swapped)
                break;
        }
    }

    // Fungsi untuk mencetak isi array
    public static void printArray(int[] arr) {
        for (int num : arr)
            System.out.print(" " + num);
    }
```

```
    public static void main(String[] args) {
        int[] arr = {64, 34, 25, 12, 22, 11, 90};
        bubbleSort(arr);
        System.out.println("Array yang telah diurutkan:");
        printArray(arr);
    }
}
```

2) Insertion Sort

Insertion sort adalah algoritma pengurutan sederhana yang bekerja dengan memasukkan setiap elemen dari daftar yang tidak diurutkan ke posisi yang benar secara berulang di bagian daftar yang diurutkan.

Berikut adalah contoh implementasi Algoritma Insertion Sort dalam program.

```
public class InsertionSortExample {

    // Fungsi untuk mengurutkan array menggunakan insertion sort
    public static void insertionSort(int[] arr) {
        int n = arr.length;

        for (int i = 1; i < n; ++i) {
            int key = arr[i];  // Elemen yang akan dibandingkan
            int j = i - 1;

            /* Memindahkan elemen arr[0..i-1] yang lebih besar dari key
               ke satu posisi di depan posisi sekarang */
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];  // Geser elemen ke kanan
                j = j - 1;
            }
            arr[j + 1] = key;  // Letakkan key di posisi yang tepat
        }
    }
```

```
    // Fungsi untuk mencetak isi array
    public static void printArray(int[] arr) {
        for (int num : arr)
            System.out.print(num + " ");
        System.out.println();
    }

    // Metode utama
    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6};

        insertionSort(arr);  // Panggil fungsi pengurutan
        System.out.println("Array yang telah diurutkan:");
        printArray(arr);     // Cetak hasil pengurutan
    }
}
```

Pemahaman terhadap algoritma lainnya dalam kategori comparison-based juga sangat dianjurkan, agar dapat memilih strategi pengurutan yang paling optimal untuk berbagai kondisi data.

2. Non-comparison-based: Membandingkan elemen dalam Algoritma Sorting tanpa membandingkan. Terdapat beberapa algoritma sorting yang termasuk ke dalam kategori Non-Comparison-based yaitu diantaranya Counting Sort dan Radix Sort.
    1) Counting Sort

       Counting Sort adalah algoritma pengurutan yang tidak menggunakan perbandingan antar elemen (non-comparison-based). Algoritma ini sangat efisien ketika rentang nilai pada data masukan relatif kecil dibandingkan dengan jumlah elemen yang akan diurutkan. Ide dasar dari Counting Sort adalah menghitung frekuensi kemunculan setiap elemen unik dalam array, lalu menggunakan informasi tersebut untuk menempatkan elemen-elemen ke posisi yang sesuai dalam urutan yang benar.

       Berikut adalah contoh implementasi Algoritma Counting Sort dalam program.

```
import java.util.Arrays;
```

```
public class CountingSortExample {
    public static int[] countSort(int[] inputArray) {
        int n = inputArray.length;
        // Mencari elemen maksimum dari inputArray
        int max = 0;
        for (int i = 0; i < n; i++) {
            if (inputArray[i] > max) {
                max = inputArray[i];
            }
        }

        // Inisialisasi countArray dengan nilai 0
        int[] countArray = new int[max + 1];

        // Menghitung frekuensi kemunculan tiap elemen
        for (int i = 0; i < n; i++) {
            countArray[inputArray[i]]++;
        }

        // Menghitung prefix sum pada countArray
        for (int i = 1; i <= max; i++) {
            countArray[i] += countArray[i - 1];
        }

        // Membuat outputArray berdasarkan countArray
        int[] outputArray = new int[n];

        // Proses pengurutan dimulai dari elemen terakhir (stabil)
        for (int i = n - 1; i >= 0; i--) {
            outputArray[countArray[inputArray[i]] - 1] = inputArray[i];
            countArray[inputArray[i]]--;
        }

        return outputArray;
    }

    public static void main(String[] args) {
```

```
        // Array input
        int[] inputArray = {4, 3, 12, 1, 5, 5, 3, 9};

        // Pemanggilan fungsi Count Sort
        int[] outputArray = countSort(inputArray);

        // Menampilkan array hasil pengurutan
        for (int i = 0; i < outputArray.length; i++) {
            System.out.print(outputArray[i] + " ");
        }
    }
}
```

2) Radix Sort

Radix Sort adalah algoritma pengurutan linear yang mengurutkan elemen dengan memprosesnya digit per digit. Algoritma ini sangat efisien untuk mengurutkan bilangan bulat (integer) atau string yang memiliki panjang kunci tetap.

Berikut adalah contoh implementasi Algoritma Radix Sort dalam program.

```
public class RadixSortExample {

    // Fungsi untuk mendapatkan nilai maksimum dalam array
    static int getMax(int[] arr, int n) {
        int max = arr[0];
        for (int i = 1; i < n; i++)
            if (arr[i] > max)
                max = arr[i];
        return max;
    }

    // Fungsi untuk melakukan Counting Sort berdasarkan digit tertentu (exp)
    static void countSort(int[] arr, int n, int exp) {
        int[] output = new int[n]; // Array hasil output
        int[] count = new int[10]; // Array untuk menghitung frekuensi digit
```

```
        // Menyimpan jumlah kemunculan tiap digit
        for (int i = 0; i < n; i++)
            count[(arr[i] / exp) % 10]++;

        // Mengubah count[i] sehingga berisi posisi akhir digit dalam output[]
        for (int i = 1; i < 10; i++)
            count[i] += count[i - 1];

        // Membangun array output berdasarkan posisi digit
        for (int i = n - 1; i >= 0; i--) {
            output[count[(arr[i] / exp) % 10] - 1] = arr[i];
            count[(arr[i] / exp) % 10]--;
        }

        // Menyalin isi output[] ke arr[]
        for (int i = 0; i < n; i++)
            arr[i] = output[i];
    }

    // Fungsi utama untuk melakukan Radix Sort
    static void radixSort(int[] arr, int n) {
        int max = getMax(arr, n); // Mendapatkan nilai maksimum

        // Melakukan counting sort untuk setiap digit (1s, 10s, 100s, dst.)
        for (int exp = 1; max / exp > 0; exp *= 10)
            countSort(arr, n, exp);
    }

    // Fungsi untuk mencetak isi array
    static void printArray(int[] arr, int n) {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
```

```
    // Main method
    public static void main(String[] args) {
        int[] arr = {170, 45, 75, 90, 802, 24, 2, 66};
        int n = arr.length;

        radixSort(arr, n); // Pemanggilan fungsi pengurutan
        System.out.println("Array setelah diurutkan:");
        printArray(arr, n);
    }
}
```

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. Tugas Praktikum

1. Simpan data berikut dalam Array 2D bertipe double, lalu tampilkan ke console. Hitung juga nilai MIN, MAX, SUM, dan AVERAGE dari data tersebut.

91.35	4.72	26	3.5	8	61	65.36	12	5.6	7.11
27	9.53	549	2.23	17.31	4.25	2.13	83	7	102.4
53.21	3.42	0.21	70.02	819.4	6173	4.25	19.8	17.35	5.768

2. Lakukan penyimpanan untuk data pada soal nomor 1 menggunakan Array 1D, kemudian cetak Kembali ke console, hitung nilai MIN, MAX, SUM, dan AVERAGE dari data tersebut.
3. Lakukan penyimpanan untuk data pada soal nomor 1 menggunakan Vector, kemudian cetak Kembali ke console, hitung nilai MIN, MAX, SUM, dan AVERAGE dari data tersebut.
4. Lakukan penyimpanan untuk data pada soal nomor 1 menggunakan ArrayList, kemudian cetak Kembali ke console, hitung nilai MIN, MAX, SUM, dan AVERAGE dari data tersebut.
5. Susunlah source code java untuk membuat struktur data ArrayList menggunakan Array (tidak menggunakan ArrayList default yang sudah ada di JDK). Lengkapi struktur data ArrayList tersebut dengan operasi-

operasi berikut: add(E e), clear(), get(int index), remove(int index), set(int index, E element), isEmpty(), size(), dan toString()

6. Susunlah source code java untuk membuat struktur data Vector menggunakan Array (tidak menggunakan Vector default yang sudah ada di JDK). Lengkapi struktur data ArrayList tersebut dengan operasi-operasi berikut: add(E e), clear(), get(int index), remove(int index), set(int index, E element), isEmpty(), size(), dan toString()
7. Buatlah program menggunakan HashSet untuk menyimpan data NIM-NIM Mahasiswa.
8. Buatlah program menggunakan HashMap untuk menyimpan data Mahasiswa-Mahasiswa yang terdiri dari NIM dan Nama Mahasiswa, dimana NIM akan dijadikan sebagai key sedangkan Nama Mahasiswa akan dijadikan sebagai value.
9. Buat program untuk menyimpan data pada soal nomor 1 ke dalam ArrayList, kemudian gunakan Iterator untuk menampilkan isi datanya satu per satu.
10. Buat program untuk menyimpan data pada soal nomor 1 ke dalam Array, lalu urutkan menggunakan Bubble Sort dan tampilkan nilai 5 terbesar.
11. Simpan 10 bilangan ke dalam Vector, kemudian tampilkan hasilnya dalam bentuk urutan ascending menggunakan Bubble Sort.

PRAKTIKUM 6: ALGORITHMS AND DATA STRUCTURES II

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator Penilaian

1. Mahasiswa mampu menjelaskan konsep dasar struktur data LINKED LIST, STACK, QUEUE, GRAPH, TREE, dan GRID.
2. Mahasiswa mampu mengidentifikasi method-method yang ada pada struktur data LINKED LIST, STACK, QUEUE, GRAPH, TREE, dan GRID pada bahasa pemrograman Java.
3. Mahasiswa mampu mengidentifikasi perbedaan struktur data LINKED LIST, STACK, QUEUE, GRAPH, TREE, dan GRID pada bahasa pemrograman Java.
4. Mahasiswa mampu membuat kode program untuk penggunaan struktur data LINKED LIST, STACK, QUEUE, GRAPH, TREE, dan GRID pada bahasa pemrograman Java.
5. Mahasiswa mampu menyusun source code untuk membuat struktur data LINKED LIST, STACK, QUEUE, GRAPH, TREE, dan GRID pada bahasa pemrograman Java.

B. Tugas Pendahuluan

1. Jelaskan konsep dasar struktur data Linked List, Stack, Queue, Graph, Tree, dan Grid.
2. Lakukan identifikasi dan jelaskan kegunaan dari method-method yang ada pada struktur data Linked List, Stack, Queue, dan Tree yang tersedia di JDK.
3. Apa perbedaan utama antara Graph berbobot dan Graph tak berbobot? Sertakan contoh situasi penggunaan masing-masing!
4. Jelaskan perbedaan antara struktur data Graph dan Tree. Dalam situasi seperti apa sebaiknya menggunakan Graph dibandingkan Tree, dan kapan sebaliknya Tree lebih tepat digunakan?

5. Jelaskan perbedaan antara struktur data Stack dan Queue. Dalam situasi seperti apa sebaiknya menggunakan Stack dibandingkan Queue, dan kapan sebaliknya Queue lebih tepat digunakan?
6. Jelaskan pengertian struktur data grid dan berikan contoh deklarasi array 2 dimensi di Java untuk merepresentasikan data suhu udara harian selama 7 hari (baris) di 5 kota (kolom).
7. Buatlah program java untuk struktur data Linked List, Stack, Queue, Graph, dan Tree pada bahasa pemrograman Java. Input untuk semua elemen-elemen dilakukan menggunakan java.util.Scanner. program juga harus dilengkapi dengan kode untuk mencetak semua elemen-elemen data yang tersimpan (input, simpan ke struktur data, dan print).
8. Tuliskan source code Java untuk menyimpan dan menampilkan data grid 2 dimensi bertipe String[][] yang berisi nama-nama matakuliah yang diambil oleh 3 mahasiswa pada 4 semester pertama. Tampilkan seluruh isi grid tersebut ke layar dengan format baris dan kolom.
9. Jelaskan tentang mekanisme Langkah operasi LIFO pada Stack.
10. Jelaskan tentang mekanisme Langkah operasi FIFO pada Queue.
11. Buatlah gambar/ilustrasi tentang struktur data Stack lengkap dengan operasi: peek( ), pop( ), dan push( ).
12. Buatlah gambar/ilustrasi tentang struktur data Queue lengkap dengan operasi: offer( ), peek( ), dan pull( ).
13. Jelaskan kegunaan lanjutan dari Struktur Data Stack
14. Jelaskan kegunaan lanjutan dari Struktur Data Queue.
15. Buatlah sebuah class Node dengan field-field sebagai berikut:
    a. public double value;
    b. public ArrayList<Node> children = null;

    lengkapi class Node dengan method constructor, getter, dan setter.

C. Ringkasan Materi

Pada praktikum kali ini kita akan berlatih menggunakan Struktur Data Non-primitif: LINKED LIST, STACK, QUEUE, GRAPH, TREE, dan GRID.

LINKED LIST

Sebelum memulai pembahasan tentang konsep Struktur Data Linked List yang terdiri dari Singly Linked List, Doubly Linked List, dan Circular Linked List disini akan terlebih dahulu kita berkenalan dengan class LinkedList yang sudah tersedia di JDK paket java.util.LinkedList. yang dokumentasinya dapat dilihat di Oracle: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

jadi sebenarnya tanpa membuat Struktur Linked List sendiri sebenarnya di JDK itu sudah ada class LinkedList yang bisa langsung kita gunakan. Sebelum melanjutkan materi, mahasiswa disarankan untuk menonton video Introduction to Java, edisi penggunaan struktur data Linked List yang tersedia di JDK : https://www.youtube.com/watch?v=7QWoRM_cc4I

Arraylist vs Linked List

Struktur Linked List bersifat lebih dinamis dari ArrayList. Jadi pada ArrayList sebenarnya strukturnya dikembangkan dari Array dimana elemen-elemen data tersimpan dan dialamatkan secara kontinyu di dalam array tetapi sizenya dinamis dikarenakan Ketika telah mencapai size maksimumnya ArrayList memiliki mekanisme untuk memperbesar size array yang digunakan sehingga ArrayList memiliki kelebihan dibanding Array yang size-nya statis alih-alih pada ArrayList size-nya dinamis. Tetapi tetap saja, struktur ArrayList dibangun dari Array. Pada Linked List elemen-elemen data (disebut node) saling terkait satu sama lain menggunakan pointer. Tetapi karena di Java tidak menggunakan pointer maka pointer ini digantikan menggunakan object. Tetapi intinya tetap sama yaitu node-node di dalam Linked List saling terkait satu sama lain.

ArrayList berbeda dengan Linked List

Membuat Object Linked List

Sekarang kita akan membuat object Linked List menggunakan class java.util.LinkedList. untuk daftar method yang dapat digunakan untuk memanipulasi Linked List ini dapat dilihat di:
https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html
diantara method-method tersebut adalah sebagai berikut:

1. **add(E e)** digunakan untuk menambahkan elemen di posisi akhir linked list
2. **add(int index, E element)** digunakan untuk menambahkan elemen di posisi tertentu pada linked list
3. **addFirst(E e)** digunakan untuk menambahkan elemen di bagian depan linked list
4. **addLast(E e)** digunakan untuk menambahkan elemen di bagian akhir linked list
5. **clear( )** digunakan untuk menghapus semua elemen yang ada di linked list
6. **clone( )** digunakan untuk mengkloning linked list

7. **contains(Object o)** digunakan untuk memeriksa apakah object tertentu ada di dalam linked list atau tidak
8. **element( )** digunakan untuk membaca/mengakses elemen head (first element) tetapi tidak menghapusnya dari linked list
9. **get(int index)** digunakan untuk membaca/mengakses elemen pada posisi tertentu di linked list
10. **getFirst( )** digunakan untuk membaca/mengakses elemen pertama di linked list
11. **getLast( )** digunakan untuk membaca/mengakses elemen terakhir di linked list
12. **indexOf(Object o**) digunakan untuk membaca/mengakses posisi(index) dari object tertentu di dalam linked list. jika object tersebut tidak ada di linked list maka method ini akan mengembalikan nilai -1. jika ada beberapa object yang sama di dalam list maka yang ditampilkan adalah index dari object yang pertama ditemukan
13. **lastIndexOf(Object o)** mirip seperti indexOf(Object o) tetapi index object terakhir yang ditampilkan
14. **offer(E e)** digunakan untuk menambahkan elemen pada posisi ekor (tail/elemen terakhir) di linked list
15. **offerFirst(E e)** digunakan untuk menambahkan elemen pada posisi front di linked list
16. **offerLast(E e)** digunakan untuk menambahkan elemen di bagian akhir linked list
17. **peek( )** digunakan untuk membaca elemen head (elemen first) tetapi tidak menghapusnya dari linked list.
18. **peekFirst( )** digunakan untuk membaca elemen first tetapi tidak menghapusnya dari linked list.
19. **peekLast( )** digunakan untuk membaca elemen terakhir tetapi tidak menghapusnya dari linked list.
20. **poll( )** digunakan untuk membaca sekaligus menghapus elemen head (elemen first) dari linked list.
21. **pollFirst( )** digunakan untuk membaca sekaligus menghapus elemen first dari linked list
22. **pollLast( )** digunakan untuk membaca sekaligus menghapus elemen terakhir dari linked list

23. **pop( )** digunakan untuk melakukan operasi pop (penghapusan elemen pertama) pada stack yang direpresentasikan menggunakan linked list
24. **push(E e)** digunakan untuk melakukan operasi push (penambahan elemen) pada stack yang direpresentasikan menggunakan linked list
25. **remove( )** digunakan untuk menghapus elemen head pada linked list
26. **remove(int index)** digunakan untuk menghapus elemen pada posisi tertentu di linked list
27. **remove(Object o)** digunakan untuk menghapus object tertentu dari linked list
28. **set(int index, E element)** digunakan untuk mengganti elemen pada posisi tertentu di linked list
29. **size( )** digunakan untuk mengetahui jumlah elemen atau banyaknya elemen yang ada di linked list
30. **toArray( )** dan **toArray(T[] a)** digunakan untuk mengkonversi linked list ke dalam array

Ada cukup banyak method yang disediakan di class java.util.LinkedList dan beberapa method tersebut ternyata memiliki operasi dan kegunaan yang hampir sama, hanya nama methodnya saja yang berbeda. Hal ini dikarenakan pada pengembangan lanjutan class java.util.LinkedList ini juga digunakan untuk mengembangkan Struktur Data lain seperti Stack dan Queue yang memiliki terminology (nama) operasi yang berbeda-beda. Nampaknya class java.util.LinkedList berusaha untuk mengakomodir semua nama-nama operasi/method tersebut.

Berikut ini contoh source code program untuk penggunaan class java.util.LinkedList:

```
package PBO25;
import java.util.LinkedList;
public class TestLinkedList {
    public static void main(String args[]) {
        // membuat object dari class LinkedList
        LinkedList<String> listKata = new LinkedList<String>();

        // memasukkan elemen ke dalam linked list
        listKata.add("Kotak");
        listKata.add("Meja");
        listKata.addLast("Buku");
        listKata.addFirst("Jalan");
        listKata.add(2, "Informatika");
```

```
        listKata.add(2, "Unsulbar");

        System.out.println(listKata);

        listKata.remove("Meja");
        System.out.println(listKata);

        listKata.remove(3);
        System.out.println(listKata);

        listKata.removeFirst();
        System.out.println(listKata);
        listKata.removeLast();
        System.out.println(listKata);
    }
}
```

ADT Linked List

Berikutnya kita akan membahas tentang ADT Linked List. Jadi dapat membuat model operasi untuk Struktur data Linked List kemudian menyusun source code class untuk Linked List kita sendiri (tanpa menggunakan class java.util.LinkedList).

Berdasarkan model data dan hubungan antar node, Linked List dibagi menjadi empat yaitu:

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Circular Doubly Linked List

Sangat disarankan kepada praktikan untuk membaca buku Data Structure and Algorithms made Easy in Java yang ditulis oleh Narasimha Karumanchi agar lebih mudah memahami tentang Struktur Data Linked List ini (Karumanchi, 2020).

Referensi dari Geeksforgeeks juga dapat membantu untuk memahami Struktur Data Linked List dengan cara yang lebih SEDERHANA:
https://www.geeksforgeeks.org/implementing-a-linked-list-in-java-using-class/

Stack_____

Stack adalah struktur data mekanisme penambahan (pemasukan) dan penghapusan (pengeluaran) datanya dilakukan pada salah satu ujung struktur

yang biasanya disebut **TOP** (ujung atas yang terbuka) dengan aturan **LIFO** (Last In First Out). Data yang terakhir dimasukkan ke dalam stack akan menjadi data pertama yang dikeluarkan dari stack. Secara fisik kita dapat membayangkan stack ini seperti tumpukan biskuit oreo di dalam tumbler (silinder) dengan luas permukaan silinder sama luas permukaan oreo ((diameter oreo + 0.01) = diameter silinder) dan hanya ada satu lubang untuk memasukkan dan mengeluarkan oreo dari dalam tumbler.
Sehingga untuk mengeluarkan oreo di lapisan paling dalam, kita harus terlebih dahulu mengeluarkan lapisan-lapisan oreo di atasnya.

Pada stack data dimasukkan dan dikeluarkan melalui pintu yang sama yang disebut **top** (karena diilustrasikan sebagai silinder yang salah satu ujungnya tertutup dan ujung lainnya terbuka dan menghadap ke atas). Sebelum melanjutkan materi ini mahasiswa dipersilahkan untuk mempelajari video tutorial struktur data Stack:
https://www.youtube.com/watch?v=8rDLV9ueVDM

Operasi-operasi dasar yang ada pada struktur data stack diantaranya sebagai berikut:

1. **empty( )**. Digunakan untuk mendeteksi apakah stack dalam keadaan kosong atau tidak
2. **peek( )**. Digunakan untuk melihat elemen/object yang berada di posisi top (posisi paling atas di stack)
3. **pop( )**. Digunakan untuk melihat, mengambil elemen sekaligus menghapusnya dari stack
4. **push( )**. Digunakan untuk memasukkan/menambahkan data ke dalam stack

Penting untuk diingat bahwa mekanisme menambah dan menghapus data di stack menggunakan urutan **LIFO** (Last In First Out) data yang terakhir dimasukkan akan menjadi data yang pertama dikeluarkan dari stack. Pada JDK sudah tersedia class java.util.Stack yang dapat digunakan untuk membuat object stack. Tetapi kita juga dapat membuat class Stack kita sendiri menggunakan Array, Vector, ArrayList, maupun LinkedList.
Berikut ini source code contoh penyimpanan data menggunakan java.util.Stack:

```
package PBO25;
import java.util.Stack;
public class Test01 {
```

```
    public static void main(String[] args) {
        Stack<Double> myStack = new Stack<>();
        myStack.push(23.7);
        System.out.println(myStack);
        myStack.push(3.14);
        myStack.push(254.0);
        myStack.push(1706.09);
        myStack.push(438.721);
        System.out.println(myStack);

        double value = myStack.pop();
        System.out.println("POP: " + value);
        System.out.println(myStack);

        value = myStack.peek();
        System.out.println("PEEK: " + value);
        System.out.println(myStack);

        myStack.push(22.16);
        myStack.push(6.43);
        System.out.println(myStack);
    }
}
```

Kita juga dapat membuat Struktur Data Stack kita sendiri. Struktur data Stack yang kita buat ini dapat kita kembangkan dari Struktur data lain yang sudah ada dan telah kita pelajari sebelumnya seperti array, vector, ArrayList dan LinkedList.

Berikut ini contoh penyusunan source code class Stack kita sendiri menggunakan Linked List bertipe int:

```
package PBO25;
import java.util.ArrayList;
public class Stack {
    private ArrayList<Integer> data = new ArrayList<>();

    public void push(int value) {
```

```
        data.add(value);
    }

    public int peek() {
        int value = data.get(data.size() - 1);
        return value;
    }

    public int pop() {
        int value = data.get(data.size() - 1);
        data.remove(data.size() - 1);
        return value;
    }

    public boolean isEmpty() {
        return data.isEmpty();
    }

    @Override
    public String toString() {
        return data.toString();
    }
}
```

Contoh penggunaan class Stack yang kita buat di class Test02:

```
package PBO25;
public class Test02 {
    public static void main(String[] args) {
        Stack myStack = new Stack();
        myStack.push(237);
        System.out.println(myStack);

        myStack.push(314);
        myStack.push(2540);
        myStack.push(170609);
```

```
        myStack.push(438721);
        System.out.println(myStack);

        double value = myStack.pop();
        System.out.println("POP: " + value);
        System.out.println(myStack);

        value = myStack.peek();
        System.out.println("PEEK: " + value);
        System.out.println(myStack);

        myStack.push(2216);
        myStack.push(643);
        System.out.println(myStack);
        System.out.println("isEmpty: " + myStack.isEmpty());
    }
}
```

Queue_____

Queue adalah struktur data antrian dengan mekanisme urutan operasi menambah dan menghapus datanya menggunakan mekanisme **FIFO** (First In First Out). Dengan mekanisme **FIFO** ini maka data yang pertama dimasukkan ke dalam queue akan menjadi data yang pertama dikeluarkan dari queue. Pada queue data dimasukkan melalui tail dan dikeluarkan melalui head.

Sebelum melanjutkan materi ini mahasiswa dipersilahkan untuk mempelajari video tutorial struktur data Queue berikut:

https://www.youtube.com/watch?v=pWWEm3Q6O6A

Operasi-operasi dasar yang ada pada struktur data queue diantaranya sebagai berikut:

1. **empty( )**. Digunakan untuk mendeteksi apakah queue dalam keadaan kosong atau tidak
2. **peek( )**. Digunakan untuk melihat elemen/object yang berada di posisi head (posisi terdepan di queue)

3. **poll( )**. Digunakan untuk melihat, mengambil elemen sekaligus menghapusnya dari queue. Elemen yang pertama dihapus adalah yang berada di posisi head
4. **offer( )**. Digunakan untuk memasukkan/menambahkan data ke dalam queue. Data ditambahkan melalui bagian tail (ekor).

Pada JDK telah tersedia interface java.util.Queue yang dapat digunakan untuk menyimpan data menggunakan prinsip **FIFO**. Pada interface java.util.Queue telah tersedia method: add( ), element( ), offer(value), peek( ), pool( ), dan remove( ). Queue pada JDK ini adalah interface sehingga kita akan membutuhkan class untuk membuat object queue. Class yang paling umum digunakan adalah class PriorityQueue dan Linked List. Berikut ini contoh penggunaan java.util.Queue untuk melakukan penyimpanan data:

```
package PBO25;
import java.util.LinkedList;
import java.util.Queue;
public class Test03 {
    public static void main(String[] args) {
        Queue<Double> myQueue = new LinkedList<>();

        System.out.println(myQueue);

        myQueue.offer(12.34);
        myQueue.offer(56.78);
        myQueue.offer(123.456);
        System.out.println(myQueue);

        double value = myQueue.poll();
        System.out.println("POLL: " + value);
        System.out.println(myQueue);

        value = myQueue.peek();
        System.out.println("PEEK: " + value);
        System.out.println(myQueue);

        myQueue.offer(3.14);
```

```
        System.out.println(myQueue);

    }
}
```

Kita juga dapat membuat Struktur Data Queue kita sendiri. Struktur data Queue yang kita buat ini dapat kita kembangkan dari Struktur data lain yang sudah ada dan telah kita pelajari sebelumnya seperti array, vector, ArrayList dan LinkedList. Berikut ini contoh penyusunan source code class Queue kita sendiri menggunakan Linked List bertipe int:

```
package PBO25;
import java.util.ArrayList;
public class Queue {
    private ArrayList<Integer> data = new ArrayList<>();

    public void offer(int value) {
        data.add(value);
    }

    public int peek() {
        int value = data.get(0);
        return value;
    }

    public int poll() {
        int value = data.get(0);
        data.remove(0);
        return value;
    }

    public boolean isEmpty() {
        return data.isEmpty();
    }

    @Override
```

```
    public String toString() {
        return data.toString();
    }
}
```

Contoh penggunaan class Queue yang telah kita buat diperlihatkan pada class Test04 berikut:

```
package PBO25;
public class Test04 {
    public static void main(String[] args) {
        Queue myQueue = new Queue();

        System.out.println(myQueue);

        myQueue.offer(1234);
        myQueue.offer(5678);
        myQueue.offer(123456);
        System.out.println(myQueue);

        double value = myQueue.poll();
        System.out.println("POLL: " + value);
        System.out.println(myQueue);

        value = myQueue.peek();
        System.out.println("PEEK: " + value);
        System.out.println(myQueue);

        myQueue.offer(314);
        System.out.println(myQueue);

        System.out.println("isEmpty: " + myQueue.isEmpty());
    }
}
```

Graph_____

Graph adalah struktur data yang digunakan untuk menyimpan hubungan (**adjacency**) antar **vertex** di dalam graph. Hubungan antar **vertex** ini dapat disimpan menggunakan **adjacency matrix** dan **adjacency list**. Visualisasi struktur data graph juga dapat dilihat di:
https://visualgo.net/en/graphds

Graph dibagi menjadi beberapa jenis, di antaranya:

- Graph Berarah (**Directed Graph**): Setiap sisi (**edge**) memiliki arah dari satu simpul ke simpul lainnya.
- Graph Tak Berarah (**Undirected Graph**): Setiap sisi tidak memiliki arah, hubungan bersifat dua arah.
- Graph Berbobot (**Weighted Graph**): Setiap sisi memiliki nilai atau bobot yang menyatakan jarak, biaya, atau nilai tertentu.
- Graph Tak Berbobot (**Unweighted Graph**): Sisi tidak memiliki bobot, semua dianggap bernilai sama.

Apa itu Adjacency Matrix?

Adjacency matrix adalah representasi struktur data graf dalam bentuk array dua dimensi matrix[i][j], di mana i dan j merupakan indeks dari simpul (**vertex**) pada graf. Nilai pada elemen matrix[i][j] menunjukkan keberadaan hubungan (**edge**) antara simpul ke-i dan simpul ke-j.

Untuk graf tak berbobot (**unweighted graph**):

- Jika terdapat hubungan (**edge**) antara dua simpul, maka matrix[i][j] = 1.
- Jika tidak terdapat hubungan antara dua simpul, maka matrix[i][j] = 0.

Pada graf berbobot (**weighted graph**), nilai matrix[i][j] tidak lagi menggunakan 1 dan 0, melainkan:

- matrix[i][j] = w (dengan w merupakan nilai bobot **edge** dari simpul ke-i ke simpul ke-j)
- Jika tidak ada hubungan, umumnya tetap menggunakan 0 tergantung pada konteks aplikasinya.

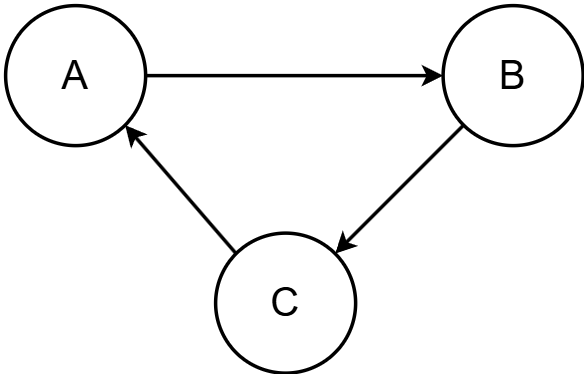
Jenis graf juga mempengaruhi isi dari adjacency matrix:

- Pada graf tak berarah (**undirected graph**), adjacency matrix bersifat simetris, artinya jika matrix[i][j] = 1, maka matrix[j][i] = 1.

- Pada graf berarah (**directed graph**), hubungan hanya satu arah, sehingga matrix[i][j] = 1 belum tentu matrix[j][i] = 1.

Adjacency matrix merupakan cara yang efisien dan sederhana untuk memvisualisasikan keterhubungan antar simpul dalam suatu graf, terutama jika jumlah simpul tidak terlalu besar.

1. Graph Berarah



Dari gambar diatas dapat kita ketahui bahwa dari Keluaran **vertex** A dan masukan ke **vertex** B memiliki arah sehingga graph bisa kita sebut sebagai graph berarah , kemudian lanjut dari **vertex** B masuk ke **vertex** C, dan **vertex** C masuk ke **vertex** A, sehingga **vertex** A memilik tetangga, yaitu **vertex** B, kemudian **vertex** B memiliki tetangga yaitu **vertex** C dan **vertex** C memiliki tetangga yaitu **vertex** A.

Lalu mengapa **vertex** A tidak bisa bertetangga dengan **vertex** C dan **vertex** B tidak bisa bertetangga dengan **vertex** A? karena **vertex** A tidak memiliki **edge** penghubung keluaran ke **vertex** C begitupun juga dengan **vertex** B ke **vertex** A.

Berikut merupakan contoh implementasi Graph di atas ke dalam program.

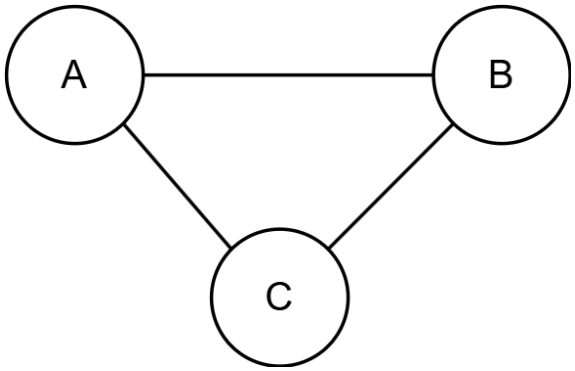
Class DirectedGraph

```
public class DirectedGraph {
    public static void main(String[] args) {
        int[][] adjacencyMatrix = {
            {0, 1, 0}, // A(0) → B(1)
            {0, 0, 1}, // B(1) → C(2)
            {1, 0, 0}  // C(2) → A(0)
        };
```

```
        System.out.println("Directed Graph (Adjacency Matrix):");
        for (int i = 0; i < adjacencyMatrix.length; i++) {
            for (int j = 0; j < adjacencyMatrix[i].length; j++) {
                System.out.print(adjacencyMatrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

2. Graph Tak Berarah

Graph tidak berarah merupakan graph yang setiap **edge** tidak memiliki panah, sehingga setiap **vertex** yang memiliki edge antara satu **vertex** dengan **vertex** yang lain dapat terhubung / bertetangga.



Dari gambar diatas dapat kita ketahui **vertex** A,B,C saling bertetangga karena setiap **edge** nya tidak memiliki panah.

Berikut merupakan contoh implementasi Graph di atas ke dalam program.

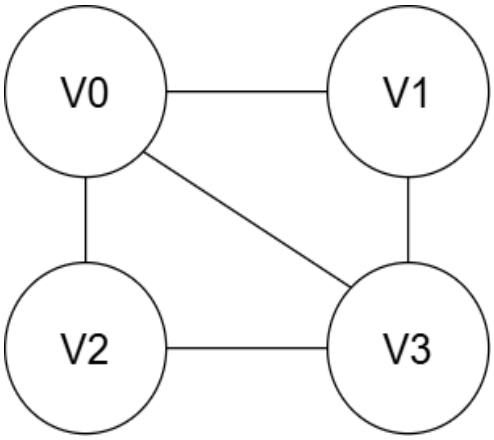
Class UndirectedGraph

```
public class UndirectedGraph {
    public static void main(String[] args) {
        int[][] adjacencyMatrix = {
            {0, 1, 1}, // A terhubung dengan B dan C
            {1, 0, 1}, // B terhubung dengan A dan C
            {1, 1, 0}  // C terhubung dengan A dan B
        };

        System.out.println("Undirected Graph (Adjacency Matrix):");
        for (int i = 0; i < adjacencyMatrix.length; i++) {
            for (int j = 0; j < adjacencyMatrix[i].length; j++) {
```

```
            System.out.print(adjacencyMatrix[i][j] + " ");
        }
        System.out.println();
    }
  }
}
```

3. Graph Tidak Lengkap

Graph tidak lengkap adalah jenis graph yang tidak memiliki semua kemungkinan **edge** (sisi) antar pasangan **vertex**. Artinya, tidak semua **vertex** saling terhubung langsung satu sama lain.



Pada gambar di atas, terdapat 4 vertex, yaitu:

- V0, V1, V2, dan V3

Edge (sisi) yang tampak menghubungkan vertex:

- V0 terhubung ke V1 dan V2
- V1 terhubung ke V3
- V2 terhubung ke V3
- V0 terhubung ke V3 (secara diagonal)

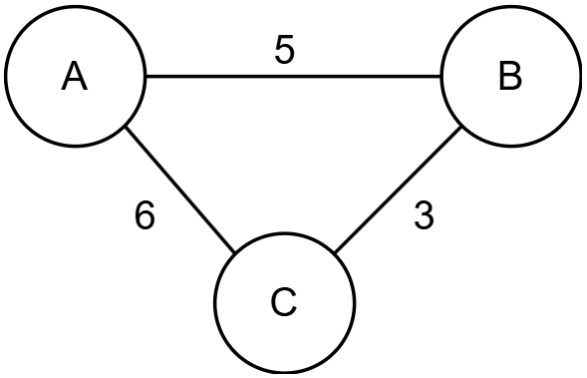
Namun tidak semua pasangan vertex saling terhubung, misalnya:

- Tidak ada edge langsung antara V1 dan V2

Hal ini menunjukkan bahwa graph tersebut adalah graph tidak lengkap, karena jumlah **edge**-nya lebih sedikit dibandingkan jumlah maksimal **edge** yang mungkin pada graph lengkap.

4. Graph Berbobot

Graph Berbobot (**Weighted Graph**) adalah jenis graph di mana setiap **edge** (sisi) memiliki bobot (nilai tertentu) yang merepresentasikan sesuatu, seperti jarak, waktu, biaya, atau kapasitas.



Dari gambar di atas, dapat kita ketahui bahwa **vertex** A ke B memiliki bobot 5, begitu pula sebaliknya. **Vertex** B ke C memiliki bobot 3, dan demikian juga sebaliknya. Sementara itu, **vertex** C ke A memiliki bobot 6, dan sebaliknya pun demikian.

Berikut merupakan contoh implementasi Graph di atas ke dalam program.

Class WeightedGraph

```
public class WeightedGraph {
   public static void main(String[] args) {
      String[] vertex = {"A", "B", "C"};

      // Matriks bobot antar vertex (0 berarti tidak terhubung)
      int[][] matriksBobot = {
         { 0,  5,  6 }, // A
         { 5,  0,  3 }, // B
         { 6,  3,  0 }  // C
      };
      System.out.println("Matriks Bobot Graf (Weighted Graph - Adjacency
Matrix):");

      System.out.print("    ");
      for (String v : vertex) {
         System.out.printf("%-4s", v);
      }
      System.out.println();

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

```
            System.out.printf("%-4s", vertex[i]); // Label baris
            for (int j = 0; j < matriksBobot[i].length; j++) {
                System.out.printf("%-4d", matriksBobot[i][j]);
            }
            System.out.println();
        }
    }
}
```

5. Graph Tak Berbobot

   Dalam pembahasan mengenai graph tak berbobot, kita tidak perlu terlalu mendalami detail khusus, karena pada dasarnya graph tak berbobot memiliki kesamaan dengan graph berarah maupun tak berarah, selama **edge**-nya tidak memiliki nilai (bobot). Artinya, selama **edge** hanya merepresentasikan hubungan antar **vertex** tanpa menyertakan nilai tertentu, maka graph tersebut termasuk dalam kategori graph tak berbobot.

   Demikian pula dalam implementasi programnya, cukup dengan merepresentasikan hubungan antar **vertex**, sebuah graph sudah dapat disebut sebagai graph tanpa bobot.

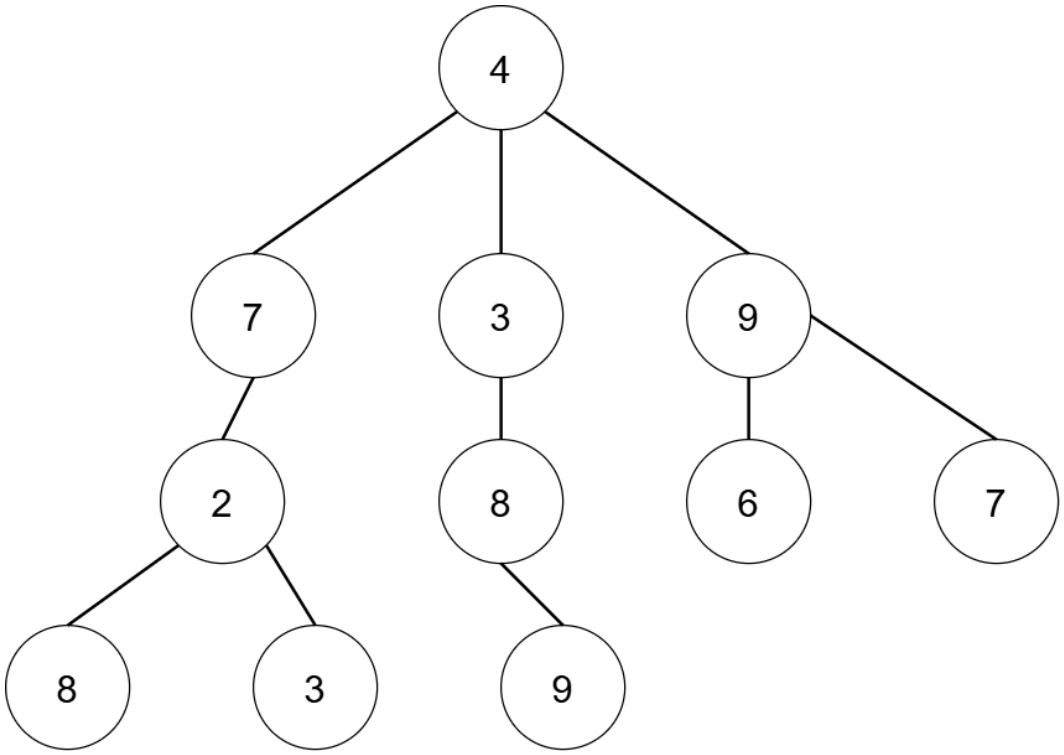
Jika masih bingung, referensi dari w3schools juga dapat membantu untuk memahami Struktur Data Graph:
https://www.w3schools.com/dsa/dsa_theory_graphs.php

Tree_____

Struktur data tree digunakan untuk menyimpan data dalam bentuk struktur pohon/hierarki. Struktur data tree dapat dibangun menggunakan class **Node** yang mirip dengan class **Node** yang digunakan pada Linked List hanya saja pada bagian pointer/address (address part) menggunakan kumpulan **Node** children (node-node anak). **Node** Children ini dapat disimpan menggunakan tipe array, vector, ArrayList, bahkan Linked List sesuai kebutuhan. Berikut ini struktur class **Node** untuk membuat class Tree:

```
import java.util.ArrayList;
public class Node {
    int data;
    ArrayList<Node> children;
}
```

Node pertama pada struktur data tree disebut dengan **root**.



Untuk menyimpan data pada gambar di atas dapat kita lakukan dengan cara berikut:

Class Node

```
import java.util.ArrayList;
public class Node {
    private int data;
    private int level;
    private ArrayList<Node> children;
    public Node(int data) {
        this.data = data;
    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }

    public int getLevel() {
        return level;
```

```
    }

    public void setLevel(int level) {
        this.level = level;
    }

    public void addChild(int data) {
        if (this.children == null) {
            this.children = new ArrayList<>();
        }
        Node newNode = new Node(data);
        newNode.setLevel(level+1);
        this.children.add(newNode);
    }

    public ArrayList<Node> getChildren() {
        return children;
    }

    public void print(){
        for (int i = 0; i < level; i++) {
            System.out.print("--- ");
        }
        System.out.println(data);
        if(this.children!=null){
            for (int i = 0; i < this.children.size(); i++) {
                this.children.get(i).print();
            }
        }
    }
}
```

Class Test05

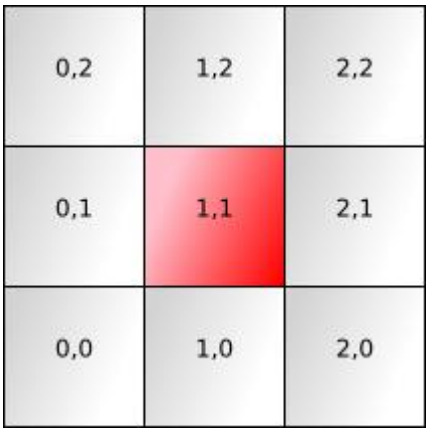
```
public class Test05 {

    public static void main(String[] args) {
        Node root = new Node(4);

        //level 1
        root.addChild(7);
        root.addChild(3);
```

```
        root.addChild(9);

        //level 2
        root.getChildren().get(0).addChild(2);
        root.getChildren().get(1).addChild(8);
        root.getChildren().get(2).addChild(6);
        root.getChildren().get(2).addChild(7);

        //level 3
        root.getChildren().get(0).getChildren().get(0).addChild(8);
        root.getChildren().get(0).getChildren().get(0).addChild(3);
        root.getChildren().get(1).getChildren().get(0).addChild(9);

        //System.out.println(root);
        root.print();

    }
}
```

Jika masih bingung, referensi dari w3schools juga dapat membantu untuk memahami Struktur Data Tree:
https://www.w3schools.com/dsa/dsa_theory_trees.php

GRID_____

Grid adalah struktur data dua dimensi yang menyerupai tabel atau matriks. Grid terdiri dari baris dan kolom, di mana setiap elemen dapat diakses berdasarkan koordinat posisinya. Struktur ini banyak digunakan untuk menyimpan data spasial, tampilan visual (seperti game), serta sistem yang membutuhkan representasi posisi secara dua dimensi.



Gambar di atas merepresentasikan sebuah matriks berukuran 3x3. Kita dapat mengimplementasikan matriks tersebut dalam bentuk program. Grid dapat direpresentasikan dalam bahasa Java menggunakan array dua dimensi (2D array). Elemen-elemen dalam grid dapat memiliki tipe data apa pun, seperti int, char, double atau String.

Class ContohGrid

```
public class ContohGrid {
    public static void main(String[] args) {
        double[][] grid = {
            {0.2, 1.2, 2.2},
            {0.1, 1.1, 2.1},
            {0.0, 1.0, 2.0}
        };

        System.out.println("Isi Grid:");
        for (int baris = 0; baris < grid.length; baris++) {
            for (int kolom = 0; kolom < grid[baris].length; kolom++) {
                System.out.print(grid[baris][kolom] + " ");
            }
            System.out.println();
        }
    }
}
```

Jika masih mengalami kebingungan dalam memahami konsep struktur data Grid, salah satu referensi yang sangat direkomendasikan adalah situs RedBlobGames. Situs ini menyajikan penjelasan yang sangat visual dan interaktif mengenai struktur data Grid dari berbagai perspektif dan bentuk umum, seperti persegi. Penjelasan disertai ilustrasi ini sangat membantu untuk memahami cara kerja dan penerapan grid dalam pengembangan aplikasi maupun game: https://www.redblobgames.com/grids/parts/

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. Tugas Praktikum

1. Diberikan data berikut:

   data

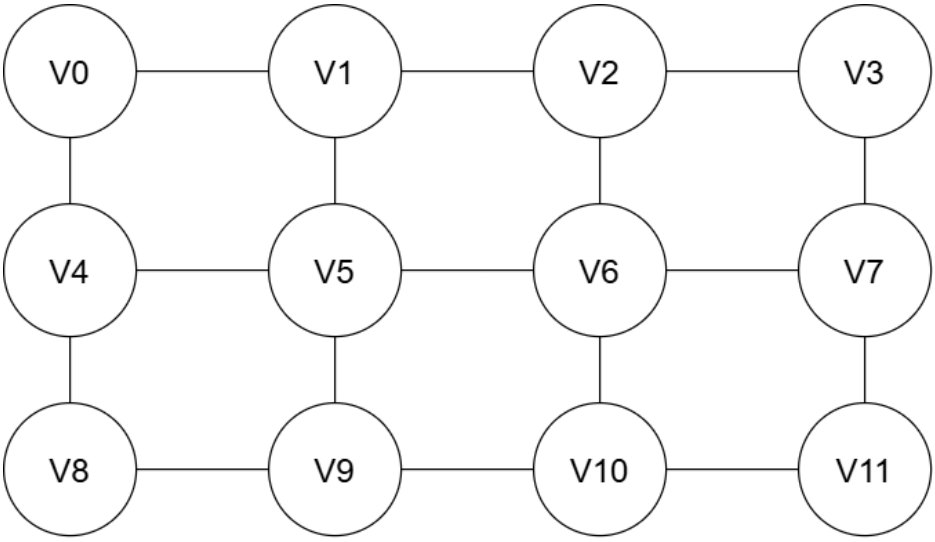
91.35	4.72	26	3.5	8	61	65.36	12	5.6	7.11
27	9.53	549	2.23	17.31	4.25	2.13	83	7	102.4
53.21	3.42	0.21	70.02	819.4	6173	4.25	19.8	17.35	5.768

   Lakukan penyimpanan untuk data di atas menggunakan LinkedList, kemudian cetak Kembali ke console, hitung nilai MIN, MAX, SUM, dan AVERAGE dari data tersebut.

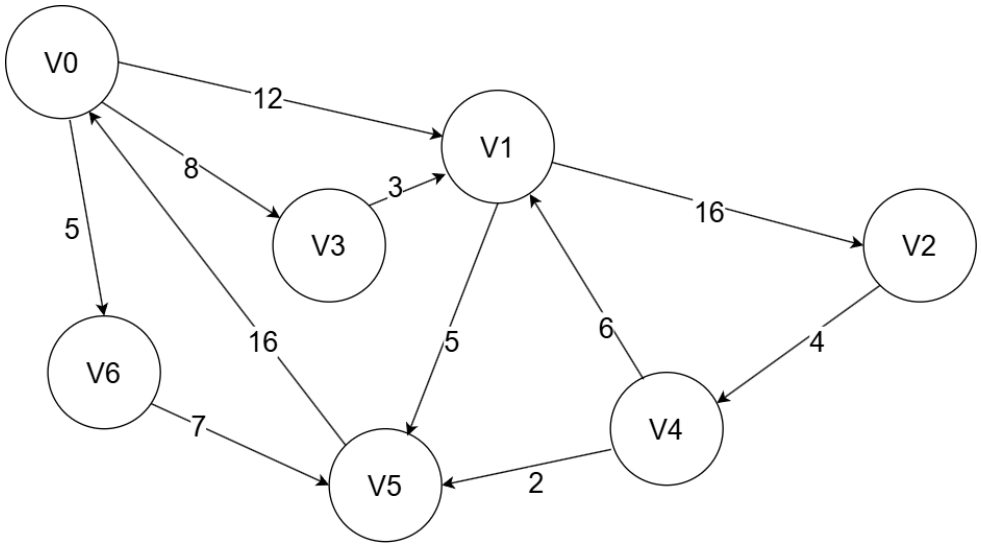
2. Lakukan penyimpanan untuk data di atas menggunakan Stack, kemudian cetak Kembali ke console, hitung nilai MIN, MAX, SUM, dan AVERAGE dari data tersebut.
3. Lakukan penyimpanan untuk data di atas menggunakan Queue, kemudian cetak Kembali ke console, hitung nilai MIN, MAX, SUM, dan AVERAGE dari data tersebut.
4. Buatlah program yang dapat menerima input String dari keyboard/console kemudian program tersebut akan melakukan reverse (membalik) string yang diinputkan menggunakan struktur data stack. Contoh input outputnya

   ```
   Informatika jaya
   ayaj akitamrofnI
   ```

5. Buatlah lima buah class Stack (empat variasi class Stack) menggunakan basis struktur data berbeda: array, vector, ArrayList dan LinkedList. Jangan lupa untuk menerapkan LIFO. Lengkapi semua class Stack yang anda buat dengan method: peek( ), push( ), pop( ), empty( ), toString( ). Buatlah sebuah class Main untuk menguji semua class Stack yang telah dibuat.
6. Buatlah lima buah class Queue (empat variasi class Queue) menggunakan basis struktur data berbeda: array, vector, ArrayList dan LinkedList. Jangan lupa untuk menerapkan FIFO. Lengkapi semua class Queue yang anda buat dengan method: peek( ), offer( ), poll( ), empty( ), toString( ). Buatlah sebuah class Main untuk menguji semua class Queue yang telah dibuat.
7. Buatlah source code untuk menyimpan struktur di bawah ini menjadi struktur graph dengan bantuan Array Adjacency. Nilai 1 akan digunakan sebagai penanda vertex yang saling terhubung (adjacent), sedangkan untuk yang tidak terhubung secara langsung akan diberi nilai 0. Hubungan vertex ke dirinya sendiri juga ditandai dengan nilai 0.



8. Buatlah source code untuk menyimpan struktur graf berbobot dan berarah berikut ini menggunakan array adjacency:



Nilai bobot edge pada graph sekaligus akan digunakan sebagai penanda keterhubungan antar vertex. Hubungan vertex dengan dirinya sendiri diberi nilai 0, demikian pula vertex yang tidak saling adjacent akan ditandai dengan nilai 0.

9. Buatlah program Java untuk merepresentasikan peta dengan struktur data Grid 2 dimensi berisi angka 1 dan 0, di mana angka 1 menunjukkan bahwa lokasi tersebut dapat dilalui dan angka 0 berarti tidak dapat dilalui. Buat program untuk menghitung dan menampilkan jumlah jalur yang dapat dilalui pada peta tersebut.
10. Buatlah program Java untuk merepresentasikan data suhu udara harian selama 7 hari di 5 kota menggunakan struktur data Grid (array 2 dimensi). Lengkapi program dengan input menggunakan Scanner dan tampilkan data suhu dalam bentuk tabel.

11. Buatlah class Tree untuk mengimplementasikan Struktur Data Tree menggunakan class Node yang dibuat pada tugas pendahuluan nomor 10 praktikum 3 ini. Tambahkan method-method berikut ini kedalam class Tree:
    a. addChildNode(Node node). Untuk menambahkan sebuah Node ke Tree
    b. search(double value). Untuk mencari value tertentu di Tree
    c. public String toString( ). Untuk mengkonversi elemen-elemen (node-node) di dalam Tree menjadi String yang kegunaan lanjutannya dapat digunakan untuk mencetak node-node yang ada di dalam Tree.

    Anda dapat memodifikasi class Node dan class Tree sesuai kebutuhan. Tambahkan pula sebuah class Main (class yang di dalamnya ada method main( )) untuk mensimulasikan penyimpanan data dan penggunaan method-method yang telah dibuat.

12. Selesaikan soal programming berikut. Background story: Pada suatu desa, setiap tahun diadakan pemilihan pengurus keamanan desa. Di desa tersebut terdapat beberapa kepala rumah tangga. semuanya diminta untuk menjadi anggota pengurus. Setiap hendak melakukan ronda, seluruh kepala rumah tangga harus berkumpul di salah satu rumah anggota pengurus. Tapi mereka tidak ingin terlalu capek. Sehingga mereka ingin mencari rumah tempat kumpul yang paling dekat dengan rumah lainnya. Peta desa dapat direpresentasikan dengan sebuah grid dengan ukuran 1x1. kemudian lokasi setiap rumah dapat di representasikan dengan lokasi sebuah sel dengan koordinat tertentu.

    Jarak antara suatu sel dengan 8 sel di sekitarnya dihitung menggunakan formula jarak manhattan (manhattan distance). Jarak sel pusat dengan satu sel di sebelahnya (timur, barat, utara, selatan) adalah sebesar 1 unit. Tapi diantara kepala rumah tangga di desa ada 1 kepala rumah tangga (sebut saja si kakek) yang sudah cukup berumur. Si kakek tidak kuat berjalan terlalu jauh. si kakek cuma mampu berjalan sejauh D unit (bolak balik) Cari rumah tempat berkumpul dimana jumlah jarak rumah lain ke rumah kumpul tersebut paling kecil.
    Tugas:
    Realisasikan method berikut:

```
Point findMeetingPlace(Point[] houseLocations, Point kakekHouse, int D){
//tulis jawabanmu dibagian ini
}
```

    Informasi:

Parameter **input** fungsi

houseLocations = array of point. posisi seluruh rumah di desa.

kakekHouse = posisi rumah si kakek

D = jarak kemampuan kakek berjalan (bolak balik)

output fungsi: index rumah yang dijadikan tempat berkumpul

Note: class **Point** memiliki member x dan y yang merupakan elemen koordinat Cartesian.

PRAKTIKUM 7: HEURISTIC SEARCH ALGORITHM

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator Penilaian

1. Mahasiswa mampu menjelaskan konsep pencarian heuristik pada graph dan grid.
2. Mahasiswa mampu memetakan ruang pencarian graph dan grid menjadi struktur data array, arraylist, dan graph.
3. Mahasiswa mampu menjelaskan konsep pencarian heuristik pada graph menggunakan algoritma greedy, DFS, BFS, Backtracking, dan A* (A-Star).
4. Mahasiswa mampu menjelaskan konsep pencarian heuristik pada grid menggunakan algoritma greedy, DFS, BFS, Backtracking, dan A* (A-Star).
5. Mahasiswa mampu memilih struktur data yang tepat untuk menyelesaikan masalah pencarian heuristik menggunakan algoritma greedy, DFS, BFS, Backtracking, dan A* (A-Star).
6. Mahasiswa mampu menyusun source code Java untuk menyelesaikan masalah pencarian heuristik pada graph dan grid menggunakan algoritma greedy, DFS, Backtracking, dan A* (A-Star).
7. Mahasiswa mampu menyusun source code Java untuk menyelesaikan masalah penukaran koin dengan jumlah koin minimum menggunakan algoritma greedy, DFS, BFS, Backtracking, dan A* (A-Star).
8. Mahasiswa mampu menyusun source code Java menggunakan algoritma greedy, DFS, BFS, Backtracking, dan A* (A-Star) untuk menyelesaikan masalah Traveling Salesman Problem (TSP)

B. Tugas Pendahuluan

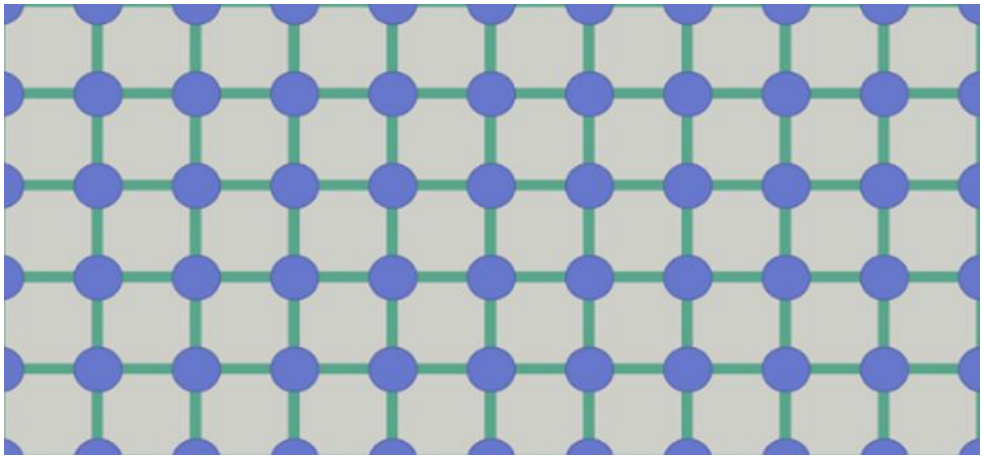
1. Jelaskan tentang konsep dasar ruang pencarian graph dan grid.
2. Gambarkan sebuah graph, dengan ketentuan graph: tidak lengkap, tidak berbobot, tidak berarah dengan jumlah vertex sebanyak 10 vertex.

3. Gambarkan sebuah graph, dengan ketentuan graph: lengkap, tidak berbobot, tidak berarah dengan jumlah vertex sebanyak 10 vertex.
4. Gambarkan sebuah graph, dengan ketentuan graph: tidak lengkap, berbobot, tidak berarah dengan jumlah vertex sebanyak 10 vertex.
5. Gambarkan sebuah graph, dengan ketentuan graph: tidak lengkap, tidak berbobot, berarah dengan jumlah vertex sebanyak 10 vertex.
6. Gambarkan sebuah graph, dengan ketentuan graph: lengkap, berbobot, tidak berarah dengan jumlah vertex sebanyak 10 vertex.
7. Gambarkan sebuah graph, dengan ketentuan graph: lengkap, tidak berbobot, berarah dengan jumlah vertex sebanyak 10 vertex.
8. Gambarkan sebuah graph, dengan ketentuan graph: tidak lengkap, berbobot, berarah dengan jumlah vertex sebanyak 10 vertex.
9. Gambarkan sebuah graph, dengan ketentuan graph: lengkap, berbobot, berarah dengan jumlah vertex sebanyak 10 vertex.
10. Gambarkan masing masing sebuah matrix untuk graph yang telah dibuat (dari nomor 2 hingga nomor 9).
11. Jelaskan cara memetakan ruang pencarian graph ke dalam struktur data.
12. Jelaskan cara memetakan ruang pencarian grid ke dalam struktur data.
13. Jelaskan konsep dasar tentang kerangka kerja algoritma greedy.
14. Jelaskan konsep dasar tentang kerangka kerja algoritma Depth First Search (DFS).
15. Jelaskan konsep dasar tentang kerangka kerja algoritma Breadth First Search (BFS).
16. Jelaskan konsep dasar tentang kerangka kerja algoritma Backtracking.
17. Jelaskan konsep dasar tentang kerangka kerja algoritma A* (A-Star).
18. Diantara Struktur data berikut: Circular Linked List, Stack, Queue, ArrayList. berdasarkan karakteristiknya manakah diantara struktur data tersebut yang paling sesuai untuk digunakan pada algoritma DFS?
19. Diantara Struktur data berikut: Circular Linked List, Stack, Queue, ArrayList. berdasarkan karakteristiknya manakah diantara struktur data tersebut yang paling sesuai untuk digunakan pada algoritma BFS?

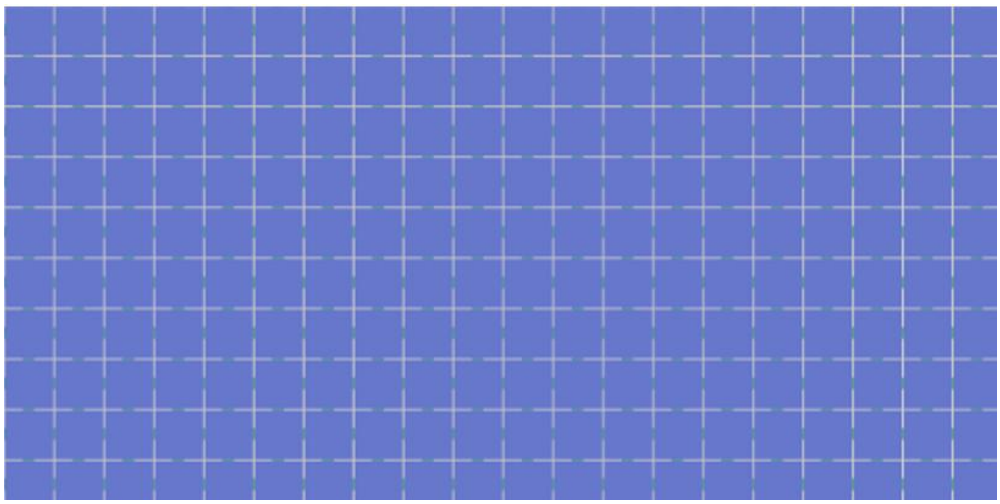
C. Ringkasan Materi

Pada praktikum ini kita akan membahas tentang konsep dari ruang pencarian graph dan grid serta algoritma Greedy, BFS, DFS, Backtracking, dan A* (A-Star). Grid yang dimaksud adalah grid ruang dua dimensi. Sebenarnya grid adalah

bagian dari graph juga. Perbandingan ruang pencarian graph dan grid dapat dilihat di:
https://www.redblobgames.com/pathfinding/grids/graphs.html



Struktur Graph



Struktur Grid

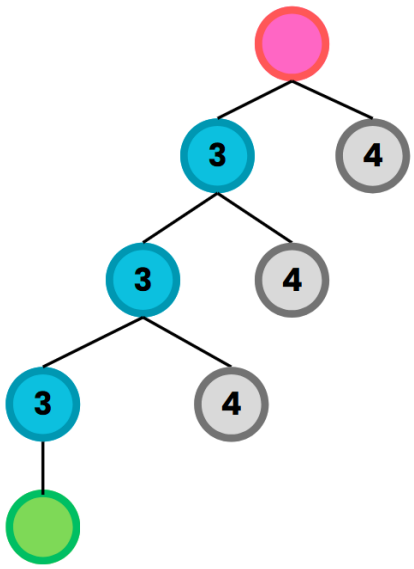
Visualisasi untuk pencarian lintasan terpendek dapat dilihat di QIAO:
https://qiao.github.io/PathFinding.js/visual/

GREEDY_____

Algoritma Greedy adalah algoritma yang menggunakan konsep **rakus** pada penyelesaian masalah. Maksud dari konsep rakus adalah pendekatan penyelesaian masalah yang membuat pilihan optimal lokal pada setiap langkah dengan harapan mencapai solusi optimal global. Dalam konteks pencarian jalur terpendek, algoritma greedy bekerja dengan selalu memilih simpul atau sisi yang

memiliki bobot terkecil pada setiap iterasi, tanpa mempertimbangkan konsekuensi di langkah selanjutnya. Prinsip dasarnya adalah dengan mengeksplorasi simpul-simpul terdekat terlebih dahulu sambil terus memperbarui jarak terpendek yang ditemukan.

Algoritma ini sangat bergantung pada sifat greedy choice property, di mana pilihan terbaik pada saat itu diharapkan mengarah ke solusi keseluruhan yang optimal. Namun, karena tidak mempertimbangkan seluruh struktur graf sekaligus, algoritma greedy bisa gagal dalam kasus tertentu, seperti pada graf dengan siklus berbobot negatif. Meskipun demikian, dalam banyak kasus praktis, pendekatan ini cukup efisien dan mudah diimplementasikan, terutama
untuk graf dengan skala besar yang membutuhkan solusi cepat.



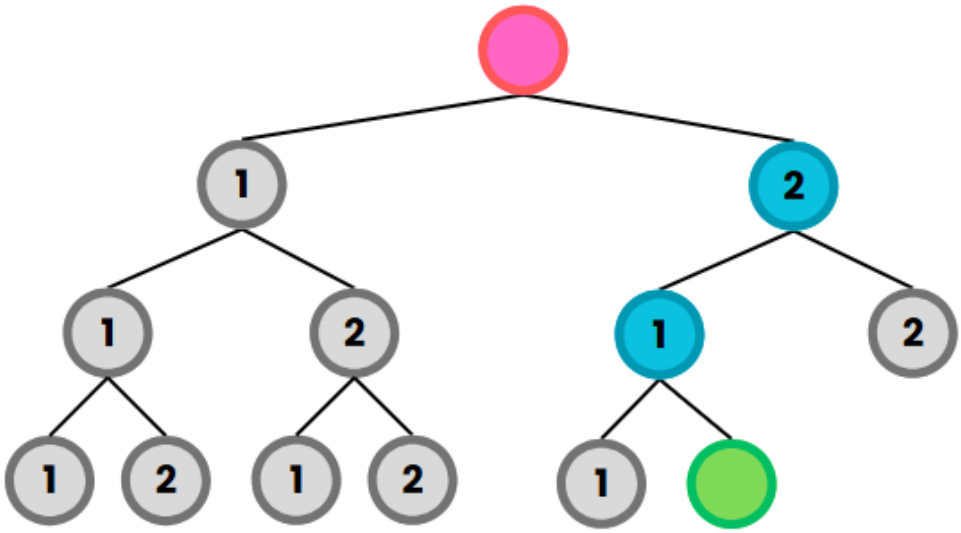
Gambar diatas adalah contoh konsep dasar implementasi algoritma greedy pada Tree dalam menentukan jalur menuju goal (tujuan), dimana algoritma greedy akan mengabaikan pilihan lain yang dianggap tidak optimal pada setiap langkah.

BREADTH FIRST SEARCH_____

Algoritma Breadth First Search (BFS) adalah salah satu algoritma traversing atau pencarian dalam struktur data graf atau pohon yang bekerja dengan mengeksplorasi semua node pada level saat ini sebelum berpindah ke node

pada level berikutnya. Algoritma ini menggunakan pendekatan First-In-First-Out (FIFO) dengan memanfaatkan antrian (queue) untuk menyimpan node yang akan dikunjungi. Karena BFS menjelajahi graf secara melebar (breadth), algoritma ini sangat efektif untuk menemukan jalur terpendek (shortest path) dalam graf tidak berbobot (unweighted graph), di mana setiap edge memiliki nilai yang sama.

Prinsip kerja BFS dimulai dengan memilih node awal (source) dan menandainya sebagai sudah dikunjungi. Selanjutnya, semua node tetangga (neighbor) yang terhubung langsung dengan node awal dimasukkan ke dalam antrian. Proses ini diulang untuk setiap node dalam antrian hingga node tujuan (destination) ditemukan atau seluruh graf telah dijelajahi. Karena BFS selalu mengeksplorasi node terdekat terlebih dahulu, jalur yang ditemukan dari node awal ke node tujuan pasti merupakan jalur terpendek dalam konteks graf tidak berbobot.

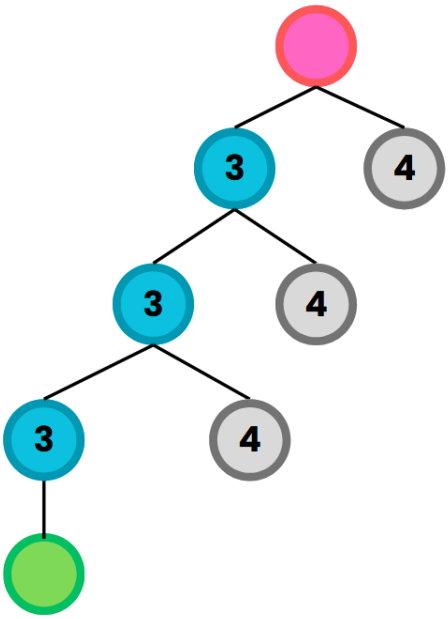


Gambar diatas adalah contoh konsep dasar implementasi algoritma BFS (Breadth First Search) pada Tree dalam menentukan jalur menuju goal (tujuan), dimana algoritma BFS akan mencoba semua pilihan yang ada pada setiap langkah dan berhenti mencoba pilihan selanjutnya ketika telah mencapai goal (tujuan).

DEPTH FIRST SEARCH_____

Algoritma Depth First Search (DFS) adalah salah satu algoritma traversal dan pencarian dalam struktur data graf atau pohon. Algoritma ini bekerja dengan

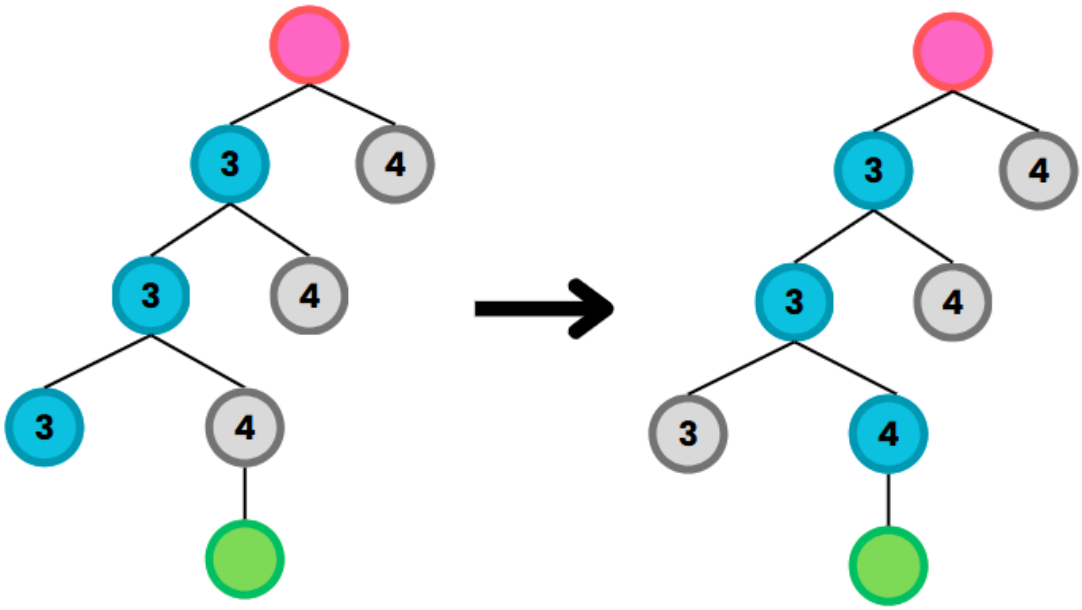
menjelajahi node atau simpul secara mendalam (depth) terlebih dahulu sebelum bergerak ke node yang bertetanggaan. Artinya, DFS akan terus menelusuri cabang atau jalur tertentu sampai menemui simpul yang tidak memiliki anak (leaf node) atau mencapai tujuan. Meskipun DFS efektif untuk menelusuri seluruh graf atau mencari keberadaan suatu jalur, algoritma ini tidak dirancang khusus untuk mencari jalur terpendek dalam graf berbobot atau tanpa bobot karena sifatnya yang tidak memprioritaskan panjang jalur.



Gambar diatas adalah contoh konsep dasar implementasi algoritma DFS (Depth First Search) pada Tree dalam menentukan jalur menuju goal (tujuan), dimana algoritma DFS hampir mirip dengan konsep algoritma greedy, akan tetapi pada algoritma DFS berfokus pada satu pilihan di setiap langkah dan tidak otomatis mengambil pilihan yang optimal yang ada.

BACKTRACKING_____

Algoritma Backtracking adalah teknik penyelesaian masalah dengan mencoba semua kemungkinan solusi secara sistematis dan membuang jalur yang tidak memenuhi kriteria optimal. Dalam konteks pencarian jalur terpendek, backtracking bekerja dengan mengeksplorasi setiap kemungkinan rute dari titik awal ke titik tujuan, lalu membandingkan panjangnya untuk menemukan yang terpendek. Jika suatu jalur melebihi panjang jalur terpendek sementara, algoritma akan "mundur" (backtrack) dan mencoba rute alternatif, sehingga mengurangi waktu komputasi dengan menghindari pencarian yang tidak perlu.



Gambar diatas adalah contoh konsep dasar implementasi algoritma backtracking pada Tree dalam menentukan jalur menuju goal (tujuan), dimana algoritma backtracking akan mundur ketika pilihan yang diambil buntu atau tidak terhubung dengan goal (tujuan)

A* (A-STAR)_____

Algoritma A-Star merupakan salah satu algoritma pencarian jarak yang memiliki kemampuan yang optimal dan komplit dalam memecahkan permasalahan yang berkaitan dengan pencarian atau penentuan sebuah rute dengan jarak terdekat. Secara konsep algoritma A-Star dibagi menjadi dua titik yaitu titik yang dapat dilalui atau biasa disebut dengan Open List dan titik yang tidak dapat dilalui atau biasa disebut dengan Close List. Secara fungsional, Close List berfungsi agar algoritma tersebut tidak melakukan pengecekan kembali pada titik yang telah dilaluinya sehingga proses pencarian yang dilakukan dapat berjalan lebih cepat dan mengurangi adanya proses pengecekan tak terbatas pada tiap titik atau node nya. Biasanya konsep pencarian rute terdekat pada sebuah algoritma adalah dimana algoritma tersebut akan berhenti jika tidak ada lagi Open List atau titik akhir sudah ditentukan.

Berikut adalah rumus perhitungan yang digunakan dalam penerapan metode A-Star:

$$F(n) = g(n) + h'(n)$$

Keterangan:
g(n) = Biaya atau Cost sebenarnya dari Node awal ke Node n.
h'(n) = Biaya atau Cost perkiraan dari Node n ke Node Tujuan.
h'(n) adalah nilai Heuristik yang ditentukan dengan melakukan perhitungan dengan rumus Heuristic Euclidean Distance. Berikut adalah rumus dari HED:

$$d(x+y) = |\sqrt{(x1-x2)^2 + (y1-y2)^2}|$$

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. Tugas Praktikum

1. Buatlah sebuah graph lengkap berbobot tak berarah(10 sampai 20 vertex). Selesaikan Masalah Komputasi Traveling Salesman Problem (TSP) pada graph tersebut menggunakan Algoritma Greedy.
2. Buatlah sebuah graph lengkap berbobot tak berarah(10 sampai 20 vertex). Selesaikan Masalah Komputasi Traveling Salesman Problem (TSP) pada graph tersebut menggunakan Algoritma DFS.
3. Buatlah sebuah graph lengkap berbobot tak berarah(10 sampai 20 vertex). Selesaikan Masalah Komputasi Traveling Salesman Problem (TSP) pada graph tersebut menggunakan Algoritma BFS.
4. Buatlah sebuah graph lengkap berbobot tak berarah(10 sampai 20 vertex). Selesaikan Masalah Komputasi Traveling Salesman Problem (TSP) pada graph tersebut menggunakan Algoritma Backtracking.
5. Buatlah sebuah graph lengkap berbobot tak berarah(10 sampai 20 vertex). Selesaikan Masalah Komputasi Traveling Salesman Problem (TSP) pada graph tersebut menggunakan Algoritma A* (A-Star).
6. Buatlah program untuk menyelesaikan masalah penukaran koin dengan jumlah koin minimum menggunakan algoritma Greedy.
7. Buatlah program untuk menyelesaikan masalah penukaran koin dengan jumlah koin minimum menggunakan algoritma DFS.
8. Buatlah program untuk menyelesaikan masalah penukaran koin dengan jumlah koin minimum menggunakan algoritma BFS.

PRAKTIKUM 8: METAHEURISTIC SEARCH ALGORITHMS & MACHINE LEARNING

Nama Mahasiswa :
NIM :
Hari/Tanggal :
Nama Asisten :
Paraf Asisten :

A. Indikator Penilaian

1. Mahasiswa mampu menjelaskan konsep dasar algoritma metaheuristik dan machine learning secara umum.
2. Mahasiswa mampu membedakan karakteristik algoritma GA, ACO, PSO, ABC, PGA, Neural Network, Naive Bayes, dan Fuzzy Logic.
3. Mahasiswa mampu mengidentifikasi permasalahan yang dapat diselesaikan menggunakan algoritma metaheuristik dan machine learning.
4. Mahasiswa mampu menemukan solusi menggunakan pemrograman berorientasi objek untuk studi kasus OOP on Artificial Intelligence (GA, ACO, PSO, ABC, PGA).
5. Mahasiswa mampu menemukan solusi menggunakan pemrograman berorientasi objek untuk studi kasus OOP on Artificial Intelligence (Naive Bayes, Neural Network (Perceptron, Backpropagation), Fuzzy Logic).

B. Tugas Pendahuluan

1. Kumpulkan 8 paper (lima tahun terakhir) yang masing-masing mewakili Algoritma di Praktikum 8 ini. Lalu buatlah resume untuk masing-masing paper yang telah dikumpulkan. Isi resume: Judul, penulis, Latar belakang, Problem yang diselesaikan, Metode yang digunakan, flowchart, pseudocode, teknik pengujian/validasi yang digunakan, kesimpulan. Setiap mahasiswa tidak boleh menggunakan paper yang sama

C. Ringkasan Materi

Pada praktikum kali ini kita akan berlatih menggunakan Algoritma Pencarian Metaheuristik & Machine Learning: GA, ACO, PSO, ABC, PGA, NEURAL NETWORK, NAIVE BAYES , FUZZY LOGIC.

Sebelum kita mempelajari algoritma-algoritma yang digunakan dalam praktikum ini, ada baiknya kita memahami terlebih dahulu apa itu algoritma pencarian metaheuristik dan machine learning.

Algoritma pencarian **metaheuristik** adalah pendekatan pemecahan masalah yang digunakan untuk mencari solusi optimal atau mendekati optimal dari permasalahan kompleks, khususnya yang sulit diselesaikan dengan algoritma konvensional.

Berbeda dengan algoritma heuristik biasa yang hanya cocok untuk satu jenis masalah tertentu, metaheuristik bersifat lebih umum, fleksibel, dan dapat diterapkan pada berbagai jenis masalah optimasi, seperti penjadwalan, rute perjalanan, pengaturan sumber daya, dan lainnya.

Sementara itu, **Machine Learning** (ML) adalah bagian dari kecerdasan buatan (*Artificial Intelligence*) yang memungkinkan komputer untuk belajar dari data dan membuat keputusan secara otomatis, tanpa diprogram secara eksplisit. Artinya, komputer tidak diberi aturan pasti satu per satu, tapi dia belajar sendiri dari pola yang ditemukan dalam data untuk kemudian digunakan dalam memprediksi atau mengklasifikasi kasus baru.

Genetic Algorithm _____

Algoritma genetika yang dikenalkan oleh Goldberg merupakan metode komputasi yang terinspirasi dari teori evolusi Darwin. Teori tersebut menyatakan bahwa makhluk hidup yang mampu **bertahan** adalah yang paling **kuat**, dan keberlangsungan hidup mereka bergantung pada proses **reproduksi**, **crossover**, serta **mutasi**. Prinsip evolusi ini kemudian diadaptasi menjadi pendekatan komputasional untuk menyelesaikan masalah secara lebih "**alami**". *Genetic Algorithm* (GA) digunakan untuk menyelesaikan berbagai masalah optimasi dan pencarian solusi dalam ruang solusi yang kompleks. Contoh penerapannya meliputi penjadwalan kerja, optimasi rute perjalanan (seperti Traveling Salesman Problem), pengenalan pola, pengaturan jalur robot, serta pengembangan sistem kontrol dalam dunia industri.

Dalam algoritma ini, setiap solusi dinamakan ***chromosome***, dan kumpulan dari solusi-solusi tersebut disebut ***populasi***. Setiap *chromosome* terdiri dari unit-unit penyusun yang dinamakan ***gen***, yang nilainya dapat berupa angka, biner, simbol, maupun karakter, tergantung dari jenis permasalahan yang ditangani. Beberapa metode representasi yang umum digunakan antara lain:

- **Binary encoding:** gen berupa 0 dan 1.
- **Real-valued encoding:** gen berupa angka desimal.
- **Permutation encoding**: gen adalah urutan objek, cocok untuk masalah rute atau urutan tugas.

Proses evolusi *chromosome* berlangsung secara bertahap melalui **generasi**. Di setiap generasi, *chromosome* dievaluasi menggunakan ***fitness function*** untuk menilai seberapa baik solusi yang dihasilkannya. *Fitness* ini menjadi dasar dalam proses **seleksi**, yaitu menentukan *chromosome* mana yang layak dipertahankan ke generasi berikutnya. Mekanisme ini mengikuti prinsip Darwin, yaitu *chromosome* dengan nilai *fitness* lebih tinggi memiliki peluang lebih besar untuk bertahan.
Beberapa metode seleksi yang umum digunakan antara lain:

1. Roulette wheel selection
2. Tournament selection
3. Rank-based selection
4. Boltzmann selection
5. Stochastic Universal Sampling

Keturunan baru atau ***offspring*** dihasilkan melalui proses ***crossover***, yaitu penggabungan antara dua *chromosome* dalam satu generasi. Proses ini bertujuan untuk menciptakan solusi baru dengan menggabungkan karakteristik dari induknya. Proporsi *chromosome* yang mengalami *crossover* diatur oleh parameter bernama ***crossover rate***. Beberapa jenis *crossover* yang sering digunakan adalah:

1. Single-point crossover
2. Two-point crossover
3. Uniform crossover
4. K-Point crossover
5. Partially mapped Crossover
6. Order Crossover
7. Precedence preserving Crossover
8. Shuffle Crossover
9. Reduced Surrogate Crossover
10. Cycle Crossover.

Selain *crossover*, perubahan nilai satu atau beberapa gen dalam *chromosome*, yang dikenal sebagai ***mutasi***, juga penting untuk menjaga keragaman populasi. Proses ini mensimulasikan perubahan genetik alami. Frekuensi terjadinya *mutasi* ditentukan oleh ***mutation rate***. Jenis *mutasi* yang umum antara lain:

1. Bit-flipping mutation
2. Swap mutation
3. Gaussian mutation
4. Displacement Mutation
5. Inversion Mutation
6. Scramble Mutation
7. Reversing Mutation

Seluruh proses tersebut berlangsung dalam beberapa generasi. Setiap generasi baru dihasilkan melalui langkah-langkah berikut:

1. Inisialisasi populasi awal secara acak
2. Evaluasi *fitness* masing-masing *chromosome*
3. Seleksi untuk memilih induk terbaik
4. *Crossover* antar-induk untuk menghasilkan keturunan
5. *Mutasi* untuk memperkenalkan variasi baru
6. Pembentukan populasi generasi berikutnya
7. Ulangi hingga kondisi berhenti terpenuhi (misalnya jumlah generasi atau konvergensi solusi)

Selama proses ini, algoritma akan berupaya memperbaiki kualitas solusi dari waktu ke waktu. Pada akhirnya, *chromosome* terbaik yang tersisa dianggap sebagai solusi optimal atau mendekati optimal dari permasalahan yang sedang diselesaikan.
Beberapa parameter utama yang mempengaruhi performa GA:

- **Population size:** jumlah *chromosome* dalam satu generasi
- **Crossover rate:** seberapa sering *crossover* dilakukan
- **Mutation rate:** probabilitas terjadinya *mutasi*
- **Number of generations**: berapa banyak siklus evolusi yang dilakukan

Pemilihan parameter yang tepat sangat penting untuk keseimbangan antara eksplorasi ruang solusi dan eksploitasi solusi terbaik.
Kelebihan algoritma genetika :

1. Tidak memerlukan informasi turunan fungsi (non-gradient)
2. Mampu menemukan solusi global meskipun ruang solusinya kompleks
3. Fleksibel terhadap berbagai jenis representasi solusi

Kekurangan algoritma genetika :

1. Bisa memerlukan waktu komputasi yang lama
2. Rentan terhadap konvergensi prematur (terjebak di solusi lokal)
3. Memerlukan eksperimen untuk menentukan parameter yang tepat

Ant Colony Optimization _____

Ant Colony Optimization (ACO) merupakan algoritma metaheuristik yang dikembangkan oleh Marco Dorigo, terinspirasi dari **perilaku semut** dalam menemukan jalur tercepat menuju sumber makanan. Dalam dunia nyata, semut berkomunikasi secara tidak langsung melalui jejak **feromon** yang mereka tinggalkan di sepanjang jalur yang mereka tempuh. Jalur yang lebih baik akan memiliki konsentrasi feromon lebih tinggi dan cenderung lebih sering dipilih oleh semut lain. Perilaku ini kemudian dimodelkan secara komputasional untuk menyelesaikan berbagai persoalan optimasi.

ACO digunakan untuk menyelesaikan berbagai permasalahan optimasi kombinatorial dan pencarian jalur terbaik. Beberapa penerapan umum ACO meliputi optimasi logistik, penjadwalan tugas, perancangan jaringan, dan alokasi sumber daya. Algoritma ini sangat cocok diterapkan pada permasalahan yang kompleks, di mana ruang solusinya besar dan sulit ditelusuri secara eksak.

Dalam algoritma ini, solusi dibangun secara bertahap oleh agen-agen yang disebut semut buatan (artificial ants). Setiap semut membangun solusi dengan memilih elemen-elemen berdasarkan probabilitas, yang dihitung dari kombinasi antara jejak feromon dan informasi heuristik (seperti jarak atau bobot). Dua parameter penting yang mempengaruhi proses pemilihan ini adalah:

- **α (alfa)**: tingkat pengaruh feromon
- **β (beta)**: tingkat pengaruh heuristik

Probabilitas semut memilih elemen j dari elemen i dihitung dengan rumus:

$$P_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k \in N_i} [\tau_{ik}]^{\alpha} \cdot [\eta_{ik}]^{\beta}}$$

Di mana:

P_{ij} : Probabilitas semut memilih untuk berpindah dari i ke j

T_{ij} : Intensitas feromon antara elemen i dan j,

n_{ij} : Informasi heuristik (misalnya $1/d_{ij}$ jika berbasis jarak),

N_i : Himpunan elemen yang mungkin dipilih dari posisi saat ini.

α : Parameter yang mengontrol pengaruh pheromone dalam pemilihan jalur.

β : Parameter yang mengontrol pengaruh heuristic (jarak).

Setelah semua semut menyelesaikan solusi mereka, intensitas feromon diperbarui. Jalur-jalur terbaik akan mendapatkan **penguatan (reinforcement)** feromon, sementara semua feromon mengalami penguapan untuk mencegah stagnasi solusi. Proses evaporasi ini mengikuti rumus:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$$

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

Dengan:

ρ : Sebagai laju evaporasi,

$\Delta\tau^{k}_{ij}$: Kontribusi feromon dari semut ke-k (biasanya proporsional terhadap kualitas solusi yang dibangun).

T_{ij} : Nilai feromon pada jalur antara i dan j.

m : Jumlah semut yang melakukan perjalanan dalam satu iterasi.

Seluruh proses berlangsung dalam beberapa iterasi. Setiap iterasi melibatkan langkah-langkah sebagai berikut:

1. Inisialisasi feromon dan parameter algoritma
2. Setiap semut membangun solusi berdasarkan probabilitas
3. Evaluasi kualitas solusi yang dihasilkan
4. Pembaruan jejak feromon
5. Ulangi proses hingga kondisi berhenti tercapai (misalnya jumlah iterasi atau tidak ada perbaikan solusi)

Selama proses tersebut, semut secara kolektif mengeksplorasi ruang solusi dan memperkuat jalur-jalur yang menjanjikan. Pendekatan ini memungkinkan ACO untuk menemukan solusi optimal atau mendekati optimal secara adaptif dan dinamis.

Beberapa parameter utama yang mempengaruhi performa ACO:

- **Jumlah semut (number of ants)**: memengaruhi keragaman solusi yang dibangun
- **Alpha dan Beta**: mengatur keseimbangan antara eksploitasi feromon dan eksplorasi heuristik
- **Evaporation rate (ρ)**: menentukan seberapa cepat feromon menguap
- **Jumlah iterasi**: batas siklus optimasi

Kelebihan Algoritma ACO:

1. Mampu menangani masalah optimasi kombinatorial yang kompleks
2. Tidak memerlukan asumsi matematis yang kaku
3. Adaptif dan fleksibel terhadap perubahan parameter atau struktur masalah

Kekurangan Algoritma ACO:

1. Dapat mengalami konvergensi prematur (terjebak di solusi lokal)
2. Kinerja sangat tergantung pada parameter awal
3. Memerlukan waktu komputasi yang cukup besar untuk masalah berskala besar

Particle Swarm Optimization _____

Particle Swarm Optimization (PSO) adalah algoritma optimasi berbasis populasi yang terinspirasi dari perilaku sosial **sekelompok hewan**, seperti kawanan **burung**, **ikan**, atau **serangga**. PSO meniru bagaimana makhluk-makhluk ini bergerak dan berkomunikasi dalam kelompok untuk mencari sumber makanan atau tujuan tertentu. Dalam algoritma ini, istilah **partikel** digunakan untuk menyebut **satu individu** dalam kelompok tersebut, misalnya satu burung dalam sekawanan burung.

Setiap partikel dalam PSO memiliki dua atribut utama, yaitu **posisi** dan **kecepatan**. Partikel-partikel ini bergerak dalam ruang pencarian (*search space*)

dan saling berbagi informasi. Ketika satu partikel menemukan posisi terbaik (yang memberikan nilai fungsi objektif optimal), partikel lainnya akan terdorong untuk menuju ke arah posisi tersebut, meskipun posisi awalnya berbeda-beda.

PSO termasuk dalam kategori swarm intelligence, yaitu kecerdasan kolektif yang muncul dari interaksi individu-individu sederhana dalam kelompok. Berbeda dengan *Genetic Algorithm* yang berbasis evolusi (*evolution-based*), PSO lebih menekankan pada kerja sama dan pembelajaran sosial antar partikel.

Inspirasi utama PSO berasal dari tiga prinsip perilaku kelompok hewan:

- Kohesi: keinginan untuk tetap dekat dengan kawanan.
- Separasi: menghindari tabrakan dengan individu lain.
- Penyesuaian(Alignment): menyelaraskan arah gerak dengan kelompok.

Dalam konteks PSO, setiap partikel memiliki:

- Personal best (pbest): posisi terbaik yang pernah dicapai oleh partikel itu sendiri.
- Global best (gbest): posisi terbaik yang pernah dicapai oleh seluruh partikel dalam populasi.

Dengan dua informasi tersebut, kecepatan dan arah gerak partikel akan diperbarui secara iteratif agar mendekati solusi terbaik.
Perubahan posisi dan kecepatan partikel dihitung menggunakan persamaan berikut:
Persamaan posisi:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Persamaan kecepatan:

$$v_i(t+1) = w v_i(t) + c_1 r_1 (pbest_i(t) - x_i(t)) + c_2 r_2 (gbest(t) - x_i(t))$$

Keterangan:

$v_i(t)$: kecepatan partikel ke-i pada iterasi ke-t
$x_i(t)$: posisi partikel ke-i pada iterasi ke-t
w : *inertia weight* (pengaruh kecepatan sebelumnya)
$c_1 c_2$: *learning coefficients* (biasanya antara 1.5 – 2.0)
$r_1 r_2$: bilangan acak dari 0 hingga 1
$pbest_i$: posisi terbaik partikel ke-i
$gbest$: posisi terbaik seluruh populasi

Langkah-Langkah dalam Algoritma PSO:

1. Inisialisasi populasi partikel secara acak (posisi dan kecepatan). Evaluasi *fitness* masing-masing partikel.
2. Simpan personal best untuk tiap partikel dan global best untuk seluruh populasi.
3. Perbarui kecepatan dan posisi tiap partikel menggunakan rumus di atas.
4. Ulangi langkah 2–4 hingga iterasi maksimum tercapai atau solusi konvergen.

Beberapa parameter penting di PSO:

1. **Jumlah partikel:** memengaruhi keragaman solusi. Semakin banyak partikel, semakin besar peluang menemukan solusi optimal, namun komputasi jadi lebih berat.
2. **Inertia weight (w):** mengatur keseimbangan eksplorasi dan eksploitasi.
    - Nilai besar → eksplorasi lebih luas
    - Nilai kecil → konvergensi lebih cepat
3. **Learning coefficients (c_1 dan c_2):**
    - c_1 mengatur pengaruh pengalaman individu (eksplorasi lokal)
    - c_2 mengatur pengaruh sosial (eksplorasi global)
4. **Jumlah iterasi:** semakin banyak iterasi, semakin baik kualitas solusi (hingga titik tertentu).

Kelebihan Algoritma Particle Swarm Optimization

1. Struktur sederhana dan mudah diimplementasikan.
2. Tidak memerlukan turunan fungsi (non-gradient).
3. Konvergensi cepat menuju solusi (terutama untuk masalah kontinu).
4. Fleksibel dan dapat disesuaikan untuk berbagai jenis masalah.

Kekurangan Algoritma Particle Swarm Optimization

1. Rentan terhadap konvergensi prematur (terjebak di solusi lokal).
2. Sensitif terhadap pemilihan parameter.
3. Tidak langsung cocok untuk masalah diskrit (perlu modifikasi khusus).

Artificial Bee Colony _____

Algoritma Artificial Bee Colony (ABC) merupakan algoritma optimasi yang diusulkan oleh Dervis Karaboga pada tahun 2005. Algoritma ini terinspirasi dari perilaku **koloni lebah madu** dalam mencari dan mengeksplorasi sumber makanan secara efisien. ABC termasuk dalam kelompok algoritma **Swarm Intelligence** karena melibatkan interaksi antar individu dalam populasi untuk menemukan solusi optimal.

Dalam dunia nyata, lebah madu berkomunikasi satu sama lain untuk menemukan dan mengeksploitasi sumber makanan melalui sebuah tarian yang dikenal sebagai **Waggle Dance.** Informasi yang disampaikan dalam tarian tersebut mencakup:

- Arah terhadap sumber makanan
- Jarak dari sarang
- Kualitas sumber makanan (jumlah nektar)

Semakin tinggi kualitas sumber makanan, maka semakin lama durasi tarian, sehingga lebih banyak lebah yang tertarik untuk menuju lokasi tersebut.
Ketika suatu sumber makanan sudah tidak layak dieksploitasi, lebah akan mencari sumber makanan baru dan membagikan informasi tersebut kepada lebah lainnya. Perilaku inilah yang menjadi dasar mekanisme eksplorasi dan eksploitasi dalam algoritma ABC.

Dalam implementasi ABC, populasi dibagi menjadi tiga jenis lebah:

1. **Employed Bees (Lebah Pekerja):** Mencari sumber makanan (solusi) dan mengevaluasinya berdasarkan kualitas (fitness). Setiap lebah pekerja terkait dengan satu sumber makanan.
2. **Onlooker Bees (Lebah Pengamat):** Menunggu informasi dari lebah pekerja di sarang, lalu memilih sumber makanan berdasarkan kualitas yang disampaikan melalui *waggle dance*. Mereka kemudian mengeksploitasi sumber makanan yang paling menjanjikan.
3. **Scout Bees (Lebah Pengintai):** Mengganti sumber makanan yang buruk (tidak mengalami perbaikan setelah beberapa iterasi) dengan solusi baru yang dihasilkan secara acak.

Langkah-langkah algoritma ABC:

a. **Inisialisasi Awal**

- Tentukan parameter: jumlah populasi, jumlah maksimum iterasi, jumlah lebah pengintai, dan batas stagnasi
- Inisialisasi posisi awal sumber makanan (solusi) secara acak dengan persamaan:

$$x_{ij}=x_j^{min}+rand(0,1)\cdot(x_j^{max}-x_j^{min})$$

Keterangan:

x_{ij} : nilai solusi ke-i pada dimensi ke-j

x_j^{min}, x_j^{max} : batas bawah dan atas dimensi ke-j

$rand(0,1)$: bilangan acak antara 0 dan 1

b. Fase Employed Bee

Setiap lebah pekerja mengeksplorasi solusi baru di sekitar sumber makanan yang telah ditugaskan padanya:

$$v_{ij}=x_{ij}+\phi ij\cdot(x_{ij}-x_{kj})$$

Keterangan:

v_{ij} : solusi baru

Φij : bilangan acak dalam rentang [-1,1]

x_{kj} : solusi lain yang dipilih secara acak, k≠i

Jika solusi baru lebih baik, maka solusi tersebut menggantikan solusi lama.

c. Fase Onlooker Bee

Lebah pengamat memilih sumber makanan berdasarkan probabilitas proporsional terhadap kualitas solusi:

$$P_i = fit_i / \sum^{N}_{j=1} fit_j$$

Keterangan:

P_i : probabilitas solusi ke-i untuk dipilih oleh onlooker bee

fit_i : nilai fitness (kualitas) dari solusi ke-i

$\sum^{N}_{j=1} fit_j$: jumlah total fitness dari semua solusi dalam populasi (N solusi)

N : jumlah total solusi (sumber makanan)

Setelah memilih, onlooker bee mengeksplorasi solusi menggunakan formula yang sama seperti pada fase employed bee.

d. Fase Scout Bee

Jika sebuah solusi tidak membaik setelah sejumlah iterasi tertentu (limit), maka lebah pekerja yang terkait akan menjadi scout bee dan mencari solusi baru secara acak.

e. Memorize the Best Solution

Solusi terbaik dari seluruh iterasi akan disimpan sebagai hasil akhir.

f. Kriteria Berhenti

Proses iterasi berhenti jika jumlah maksimum iterasi telah tercapai atau jika solusi terbaik tidak mengalami perbaikan lebih lanjut.

Kelebihan Algoritma Artificial Bee Colony:

1. Implementasi sederhana dan efisien
2. Mampu menyeimbangkan eksplorasi dan eksploitasi
3. Dapat diterapkan untuk berbagai jenis masalah optimasi

Kekurangan Algoritma Artificial Bee Colony:

1. Dapat terjebak pada solusi lokal
2. Perlu penyesuaian parameter agar hasil optimal
3. Kinerja tergantung pada pemilihan strategi eksplorasi

Parthenogenetic Algorithm _____

Parthenogenetic Algorithm (PGA) merupakan salah satu algoritma evolusioner yang terinspirasi dari proses **reproduksi aseksual alami**, khususnya *parthenogenesis*, yaitu proses berkembang biaknya organisme tanpa pembuahan. Dalam biologi, proses ini terjadi ketika individu betina mampu menghasilkan keturunan tanpa keterlibatan sel jantan. Konsep ini kemudian diadaptasi menjadi pendekatan komputasi untuk menyelesaikan masalah optimasi dengan pendekatan evolusioner yang lebih sederhana dibandingkan Genetic Algorithm (GA), karena tidak memerlukan proses crossover antara dua individu.

PGA digunakan dalam berbagai permasalahan optimasi, seperti perancangan sistem kontrol, klasifikasi data, pengaturan parameter, dan pencarian solusi dalam ruang solusi yang kompleks. Keunggulan PGA terletak pada efisiensinya dalam eksplorasi solusi dan kemampuannya untuk bekerja secara efektif dengan populasi kecil, karena hanya memerlukan satu induk untuk menghasilkan keturunan.

Dalam algoritma ini, setiap solusi disebut **individu** atau **chromosome**, dan evolusi berlangsung melalui proses rekombinasi dan mutasi terhadap individu tersebut. Tidak ada proses seleksi pasangan atau crossover seperti pada GA. Proses utama dari PGA meliputi:

1. **Reproduksi Aseksual (Cloning dan Modifikasi),** Setiap individu mereplikasi dirinya sendiri untuk menghasilkan satu atau lebih keturunan. Keturunan tersebut kemudian dimodifikasi melalui proses mutasi untuk menciptakan variasi genetik.
2. **Mutasi Genetik,** Mutasi dilakukan pada gen-gen tertentu dalam chromosome untuk menciptakan variasi. Mutasi ini merupakan mekanisme utama eksplorasi solusi. Beberapa teknik mutasi yang dapat digunakan antara lain:
    - Gaussian mutation (untuk representasi real-valued)
    - Bit-flipping (untuk representasi biner)
    - Swap mutation (untuk urutan/permutasi)
3. **Evaluasi Fitness,** Setiap individu dan keturunannya dievaluasi menggunakan fungsi objektif atau fitness function. Nilai fitness menunjukkan seberapa baik solusi tersebut menyelesaikan permasalahan yang diberikan.
4. **Seleksi Bertahan,** Individu terbaik dari populasi (baik induk maupun keturunan) dipilih untuk membentuk populasi generasi berikutnya. Proses ini memastikan bahwa solusi terbaik tetap dipertahankan dan diperbaiki dari waktu ke waktu.

Langkah-langkah utama dalam satu siklus generasi PGA adalah:

1. Inisialisasi populasi awal secara acak
2. Evaluasi fitness setiap individu
3. Reproduksi aseksual untuk menghasilkan keturunan dari tiap individu
4. Mutasi pada keturunan untuk menghasilkan variasi
5. Evaluasi fitness keturunan
6. Seleksi individu terbaik untuk generasi berikutnya
7. Ulangi hingga kondisi berhenti tercapai (jumlah generasi atau konvergensi)

Tidak adanya crossover menjadikan PGA lebih sederhana dan komputasinya lebih ringan dibandingkan GA. Namun, variasi solusi sangat tergantung pada kualitas dan skema mutasi yang digunakan.
Beberapa parameter penting dalam PGA:

- **Population size:** jumlah individu dalam setiap generasi
- **Mutation rate**: tingkat probabilitas terjadinya mutasi gen

- Number of clones: jumlah keturunan yang dihasilkan per individu
- Number of generations: batas iterasi proses evolusi

Kelebihan Parthenogenetic Algorithm:

1. Sederhana karena tidak membutuhkan pasangan atau proses crossover
2. Efisien dalam penggunaan populasi kecil
3. Cocok untuk masalah dengan domain solusi yang sempit atau eksplorasi awal

Kekurangan Parthenogenetic Algorithm

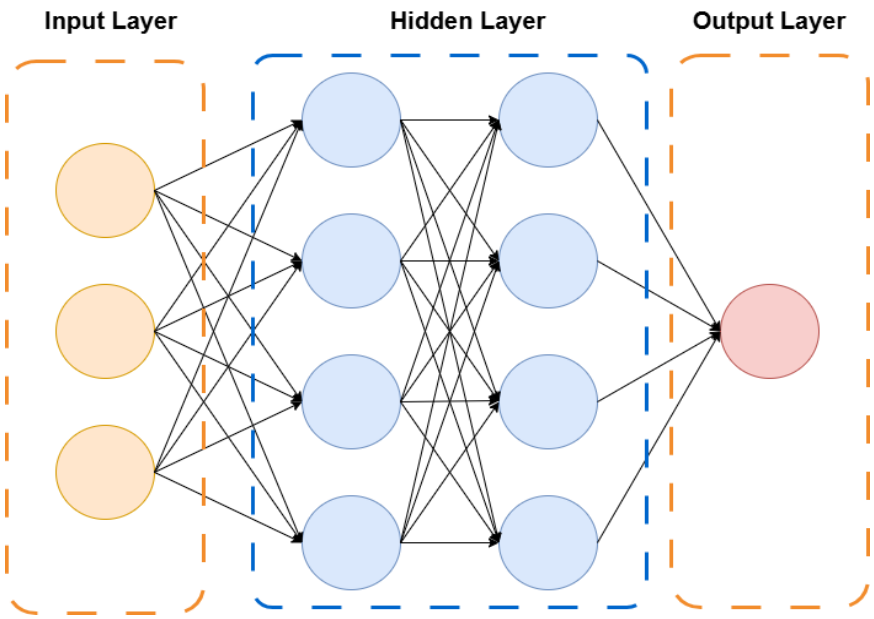
1. Keragaman populasi dapat menurun cepat tanpa skema mutasi yang baik
2. Rentan terhadap stagnasi jika tidak ada mekanisme eksplorasi tambahan
3. Performa sangat bergantung pada parameter mutasi

Neural Network _____

Di dunia kecerdasan buatan, algoritma Neural Network atau Jaringan Syaraf Tiruan merupakan pendekatan komputasi yang terinspirasi oleh cara kerja otak manusia. Algoritma ini dirancang untuk mengenali pola dan hubungan dalam data melalui proses belajar dari pengalaman, serupa dengan bagaimana otak manusia belajar melalui koneksi antar neuron.

A. Struktur Dasar Jaringan Syaraf Tiruan

   Sebuah jaringan syaraf tiruan terdiri dari tiga lapisan utama: lapisan input, lapisan tersembunyi (hidden layer), dan lapisan output. Setiap lapisan terdiri dari sejumlah unit pemroses kecil yang disebut neuron atau node, yang terhubung satu sama lain melalui jalur-jalur yang disebut bobot (weights).



- **Input Layer** menerima data awal yang masuk ke sistem, seperti gambar, angka, atau suara.
- **Hidden Layer** melakukan proses transformasi kompleks dengan menggabungkan input menggunakan bobot dan fungsi aktivasi, menciptakan representasi internal dari data.
- **Output Layer** menghasilkan prediksi atau keputusan akhir dari jaringan, misalnya apakah suatu gambar menunjukkan kucing atau anjing.

B. **Proses Belajar dan Propagasi**

Algoritma neural network bekerja melalui dua proses utama: propagasi maju **(forward propagation)** dan propagasi balik **(backpropagation).**

- Propagasi Maju Data input mengalir dari lapisan input menuju lapisan tersembunyi dan akhirnya ke lapisan output. Di setiap neuron, input dikalikan dengan bobot, dijumlahkan, lalu dilewatkan ke fungsi aktivasi seperti sigmoid atau ReLU (Rectified Linear Unit), yang menentukan apakah neuron tersebut akan "menyala" atau tidak.
- Backpropagation Setelah output dihasilkan, sistem akan mengevaluasi seberapa besar kesalahan (error) antara hasil prediksi dan nilai sebenarnya. Kemudian, sistem secara otomatis menyesuaikan bobot-bobot jaringan secara bertahap untuk mengurangi kesalahan tersebut. Proses penyesuaian ini dilakukan melalui algoritma pembelajaran seperti gradient descent.

C. Kemampuan Adaptif dan Generalisasi

Salah satu kekuatan utama jaringan syaraf tiruan adalah kemampuannya untuk belajar dari data tanpa perlu diprogram secara eksplisit. Semakin banyak data yang digunakan, semakin akurat jaringan dalam mempelajari pola dan membuat prediksi. Neural network sangat efektif dalam menangani masalah yang kompleks dan tidak linear, seperti pengenalan wajah, prediksi cuaca, klasifikasi teks, dan banyak aplikasi lainnya. Namun, jaringan saraf tiruan juga memiliki tantangan, seperti kebutuhan akan data dalam jumlah besar, waktu pelatihan yang lama, serta risiko overfitting jika tidak diatur dengan baik

Naive Bayes _____

Naïve Bayes Classifier adalah metode klasifikasi yang didasarkan pada prinsip Teorema Bayes. Metode ini menggunakan pendekatan probabilistik dan statistik yang diperkenalkan oleh ilmuwan asal Inggris, Thomas Bayes, yang memprediksi kemungkinan suatu peristiwa di masa depan berdasarkan data atau pengalaman di masa lalu. Ciri khas dari Naïve Bayes Classifier adalah adanya asumsi kuat—dan cenderung sederhana (naïf)—bahwa setiap fitur atau kondisi bersifat saling bebas (independen). Dalam istilah yang lebih sederhana, Teorema Bayes adalah rumus matematika sederhana untuk menemukan probabilitas ketika kita mengetahui probabilitas tertentu lainnya.

$$P(A \mid B) = P(B \mid A)P(A)P(B)$$

Keterangan:

P(A | B) : Probabilitas A terjadi dengan bukti bahwa B telah terjadi (probabilitas superior)

P(B | A) : Probabilitas B terjadi dengan bukti bahwa A telah terjadi

P(A) : Peluang terjadinya A

P(B) : Peluang terjadinya B

Metode Naive Bayes digolongkan menjadi beberapa tipe berdasarkan fungsinya

1. Multinomial Naive Bayes
2. Bernoulli Naive Bayes
3. Gaussian Naive Bayes

Berikut adalah cara kerja dari algoritma naive bayes classifier, kita akan coba untuk mengambil studi kasus pembelian laptop. Pada dataset tersebut ada lima

kriteria yaitu umur, penghasilan, status pekerjaan, memiliki rumah, status kredit dan satu label beli komputer. Berikut tabel data

Beberapa kegunaan algoritma naive bayes :

1. Berperan sebagai salah satu metode machine learning berbasis pendekatan probabilistik.
2. Dapat dimanfaatkan dalam sistem diagnosis medis otomatis.
3. Efektif untuk mendeteksi dan menyaring pesan spam.

Beberapa Kelebihan algoritma naive bayes :

1. Dapat digunakan baik untuk data berbentuk angka (kuantitatif) maupun data berbasis kategori atau deskripsi (kualitatif).
2. Tidak memerlukan proses pelatihan data (training) dalam jumlah besar.
3. Proses komputasi berlangsung dengan cepat dan efisien.
4. Proses klasifikasi dokumen dapat disesuaikan secara personal sesuai kebutuhan individu.
5. Cocok untuk digunakan dalam klasifikasi dua kelas (biner) maupun lebih dari dua kelas (multiclass).

Beberapa Kekurangan algoritma naive bayes :

1. Jika nilai probabilitas kondisional suatu fitur adalah nol, maka hasil prediksi keseluruhan juga akan menjadi nol.
2. Asumsi bahwa setiap variabel bersifat independen dapat menurunkan tingkat akurasi, karena dalam kenyataannya sering terdapat korelasi antar variabel.

Fuzzy Logic _____

Algoritma fuzzy adalah metode komputasi yang meniru cara manusia mengambil keputusan dengan mempertimbangkan ketidakpastian dan ambiguitas. Berbeda dengan logika klasik (boolean) yang hanya mengenal dua nilai pasti (0 = salah, 1 = benar), logika fuzzy memungkinkan nilai kebenaran dalam rentang 0 hingga 1. Ini memungkinkan sistem untuk menangani data yang tidak pasti, kabur, atau tidak terdefinisi dengan jelas.

Dalam sebuah sistem berbasis logika fuzzy, terdapat tiga komponen utama yang saling terhubung dan bekerja secara sistematis untuk menghasilkan keputusan

atau output dari sebuah input yang bersifat kabur atau tidak pasti. Ketiga komponen tersebut adalah fuzzifikasi, mesin inferensi fuzzy, dan defuzzifikasi.

1. Fuzzifikasi

   Fuzzifikasi merupakan langkah awal dalam sistem fuzzy yang berfungsi untuk mengubah data input yang bersifat tegas (crisp) menjadi bentuk fuzzy. Input ini biasanya berupa nilai numerik, seperti suhu, kecepatan, atau jarak, yang kemudian dipetakan ke dalam himpunan-himpunan fuzzy seperti dingin, hangat, atau panas. Setiap nilai input tidak hanya masuk ke satu kategori, melainkan bisa memiliki tingkat keanggotaan tertentu pada beberapa kategori sekaligus. Proses ini memungkinkan sistem untuk memahami data input dalam bentuk yang lebih fleksibel dan menyerupai cara manusia menilai sesuatu secara tidak eksak.

2. Inferensi Fuzzy

   Setelah proses fuzzifikasi, nilai-nilai fuzzy yang dihasilkan akan diproses dalam komponen kedua, yaitu mesin inferensi fuzzy. Komponen ini merupakan inti dari sistem fuzzy, di mana sekumpulan aturan logika (biasanya berbentuk IF-THEN) digunakan untuk mengolah input fuzzy menjadi output fuzzy. Aturan-aturan ini mencerminkan pengetahuan atau pengalaman ahli yang dituangkan dalam bentuk logika linguistik. Misalnya, sebuah aturan dapat berbunyi: "Jika suhu panas dan kelembaban tinggi maka kipas dipercepat." Mesin inferensi akan mengevaluasi semua aturan yang relevan dan menggabungkan hasilnya menjadi keluaran fuzzy yang mewakili berbagai kemungkinan tindakan yang dapat diambil.

3. Defuzzyfikasi

   Komponen terakhir adalah defuzzifikasi, yaitu proses mengubah output fuzzy yang diperoleh dari mesin inferensi menjadi nilai tegas (crisp) yang bisa digunakan sebagai keputusan nyata atau dikirim ke perangkat kendali. Karena output fuzzy biasanya berupa gabungan beberapa kemungkinan nilai, defuzzifikasi akan menentukan satu nilai akhir yang paling representatif. Proses ini dilakukan dengan berbagai metode, seperti metode pusat gravitasi (centroid), rata-rata maksimum, atau metode lain yang sesuai dengan kebutuhan sistem. Hasil akhir dari defuzzifikasi inilah yang digunakan sebagai keputusan sistem, seperti menentukan kecepatan kipas, jarak aman, atau tingkat ancaman.

D. Langkah Praktikum

Kerjakan kembali program-program yang ada di ringkasan materi. Diskusikan dengan teman atau dosen jika ada bagian yang kurang jelas atau masih sulit dipahami.

E. Tugas Praktikum

1. Case Study: OOP on Artificial Intelligence - GA
2. Case Study: OOP on Artificial Intelligence - ACO
3. Case Study: OOP on Artificial Intelligence - PSO
4. Case Study: OOP on Artificial Intelligence - ABC
5. Case Study: OOP on Artificial Intelligence - PGA
6. Case Study: OOP on Artificial Intelligence - Neural Network Perceptron
7. Case Study: OOP on Artificial Intelligence - Neural Network Backpropagation
8. Case Study: OOP on Artificial Intelligence - Naive Bayes
9. Case Study: OOP on Artificial Intelligence - Fuzzy Logic

REFERENSI

A. A. A. Cirua and S. Cokrowibowo, "Representasi Chromosome Gray Code Algoritma Genetika pada job shop scheduling problem," no. Senarai, pp. 184–189, 2023.

A. A. Asnan Cirua, I. Nurtanio, and Syafaruddin, "Scheduling Job Machines with Swap Sequence to Minimize Makespan Using Spider Monkey Optimization Algorithm," *Proceeding - 6th Int. Conf. Inf. Technol. Inf. Syst. Electr. Eng. Appl. Data Sci. Artif. Intell. Technol. Environ. Sustain. ICITISEE 2022*, no. December 2022, pp. 291–296, 2022, doi: 10.1109/ICITISEE57756.2022.10057926.

Cokrowibowo, S., Nur, N., & Irmayanti. (2020). Web Page Extraction and Classification using JSOUP and Naive Bayes. IOP Conference Series: Materials Science and Engineering. Retrieved from https://iopscience.iop.org/article/10.1088/1757-899X/875/1/012089/pdf

Darmawan, Cokrowibowo, S., Rustan, M., & Irianti, A. (n.d.). Pengembangan Algoritma Genetika untuk Menyelesaikan Course Scheduling Problem menggunakan Partially Mapped Crossover dan Random Pairs Mutagenesis. Konferensi Nasional Ilmu Komputer (KONIK) 2021. Makassar.

Firgiawan, W., Cokrowibowo, S., Irianti, A., & Gunawan, A. (2021). Performance Comparison of Spider Monkey Optimization and Genetic Algorithm for Traveling Salesman Problem. International Conference on Electronics Representation and Algorithm (ICERA).

Kadir, A. (2004). Dasar pemrograman Java 2. Yogyakarta: Penerbit Andi.

Lowe, D. (2020). Java All In One For Dummies 6th Edition. Hoboken: John Wiley & Sons, Inc.

Mawlood-Yunis, A.-R. (2022). Android for Java Programmers. Switzerland: Springer.

Oracle. (2022, 9 10). The Java Tutorials. Retrieved from Java Documentation:

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html
Oracle. (2022, September 4). The Java™ Tutorials: Object-Oriented Programming Concepts. Retrieved from docs.oracle.com: https://docs.oracle.com/javase/tutorial/java/concepts/index.html

Samoylov, N. (2022). Learn Java 17 Programming Second Edition. Birmingham, UK: Packt Publishing.

Suarga. (2009). Dasar Pemrograman Komputer dalam Bahasa Java. Yogyakarta: Penerbit Andi. w3schools. (2022, 9 10). Java Modifiers. Retrieved from w3schools: https://www.w3schools.com/JAVA/java_modifiers.asp