sada
kurnia pustaka

PEMROGRAMAN Java

Tim Penulis:

Mohamad Yusuf
Hedie Kristiawan
Norbertus Tri Suswanto Saptadi
Hendri Julian Pramana
Agung Yuliyanto Nugroho
Nina Rahayu
Deden Rustiana
Gde Brahupadhya Subiksa
Eka Purnama Harahap
Mohammad Badrul
Alfredo Gormantara
Fahmi Fadillah Septiana
Yosep Bustomi

PEMROGRAMAN JAVA

Mohamad Yusuf
Hedie Kristiawan
Norbertus Tri Suswanto Saptadi
Hendri Julian Pramana
Agung Yuliyanto Nugroho
Nina Rahayu
Deden Rustiana
Gde Brahupadhya Subiksa
Eka Purnama Harahap
Mohammad Badrul
Alfredo Gormantara
Fahmi Fadillah Septiana
Yosep Bustomi

sada
kurnia pustaka

PEMROGRAMAN JAVA

Tim Penulis:
Mohamad Yusuf
Hedie Kristiawan
Norbertus Tri Suswanto Saptadi
Hendri Julian Pramana
Agung Yuliyanto Nugroho
Nina Rahayu
Deden Rustiana
Gde Brahupadhya Subiksa
Eka Purnama Harahap
Mohammad Badrul
Alfredo Gormantara
Fahmi Fadillah Septiana
Yosep Bustomi

Editor : Ajay Supriadi, M.Kom.
Tata Letak : Asep Nugraha, S.Hum.
Desain Cover : Septimike Yourintan Mutiara, S.Gz.
Ukuran : UNESCO 15,5 x 23 cm
Halaman : vii, 185
ISBN : 978-634-7021-69-4
Terbit Pada : Agustus 2025
Anggota IKAPI : No. 073/BANTEN/2023

PENERBIT PT SADA KURNIA PUSTAKA
Jl. Warung Selikur Km.6 Sukajaya – Carenang, Kab. Serang-Banten
Email : sadapenerbit@gmail.com
Website : sadapenerbit.com & repository.sadapenerbit.com
Telpon/WA : +62 838 1281 8431

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya, sehingga buku **"Pemrograman *Java*"** ini dapat diselesaikan dengan baik. Buku ini hadir sebagai panduan praktis dan komprehensif bagi siapa pun yang ingin memulai atau mendalami dunia pemrograman, khususnya menggunakan bahasa pemrograman *Java*.

Pemrograman *Java* telah menjadi salah satu bahasa pemrograman paling populer dan relevan di dunia. Fleksibilitas, portabilitas, dan performanya yang tinggi menjadikan *Java* pilihan utama untuk pengembangan aplikasi lintas platform, mulai dari aplikasi desktop, web, hingga perangkat seluler berbasis Android. Oleh karena itu, menguasai *Java* adalah langkah strategis bagi Anda yang ingin berkarier di bidang teknologi informasi.

Dalam buku ini, kami menyajikan materi secara terstruktur, dimulai dari konsep dasar hingga topik yang lebih mendalam. Anda akan dipandu untuk memahami sintaksis dasar, konsep *Object-Oriented Programming* (OOP), struktur data, hingga pengembangan aplikasi yang lebih kompleks. Setiap bab dilengkapi dengan contoh-contoh kode yang relevan dan mudah dipahami, serta latihan soal untuk menguji pemahaman Anda.

Kami menyadari bahwa buku ini masih memiliki kekurangan. Oleh karena itu, segala kritik dan saran yang membangun akan kami terima dengan lapang dada demi perbaikan di masa mendatang. Semoga buku ini dapat memberikan manfaat sebesar-besarnya bagi para pembaca dan menjadi bekal berharga dalam perjalanan Anda menjadi seorang programmer andal.
Terima kasih.

Selamat Membaca!

Penulis

DAFTAR ISI

BAB 1
PENGENALAN PEMROGRAMAN *JAVA*

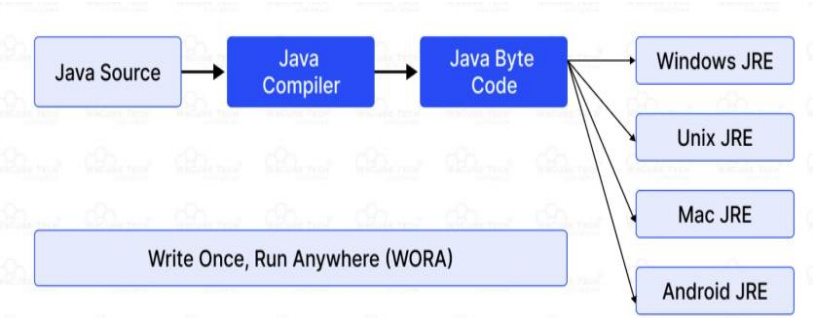
Mohamad Yusuf, S.Kom., M.C.S.
Universitas Mercu Buana

Pendahuluan

Dalam era teknologi yang terus berkembang, bahasa pemrograman menjadi landasan utama untuk menciptakan perangkat lunak, aplikasi, dan sistem yang kita gunakan sehari-hari. Di antara banyak bahasa pemrograman yang ada, *Java* telah terbukti sebagai salah satu yang paling andal, populer, dan bertahan lama.

Java diperkenalkan pada tahun 1995 oleh *Sun Microsystems,* yang sekarang merupakan bagian dari *Oracle Corporation*. Sejak itu, *java* tidak hanya digunakan untuk membuat aplikasi *desktop* sederhana, tetapi juga menjadi basis bagi berbagai sistem besar, mulai dari aplikasi *mobile android* hingga layanan *web* berskala enterprise dan solusi *big data.*

Apa yang membuat *Java* begitu istimewa? Jawabannya terletak pada prinsip dasarnya: "*Write Once, Run Anywhere*." Ini berarti kode yang ditulis dapat dijalankan di berbagai jenis komputer tanpa banyak penyesuaian, asalkan perangkat tersebut mendukung *Java Virtual Machine* (JVM). Prinsip ini menjadikan *Java* pilihan utama bagi para pengembang perangkat lunak di seluruh dunia (Gosling et al., 2015).



Gambar 1.1: *Java Compiler*
Sumber: Diolah Penulis.

Sejarah dan Filosofi *Java*

Java awalnya dikembangkan sebagai bagian dari proyek bernama "*Green Project*" yang dipimpin oleh James Gosling dan timnya di *Sun Microsystems* pada awal 1990-an. Tujuan awalnya adalah membuat bahasa pemrograman untuk perangkat elektronik kecil seperti TV *set-top box* dan *remote control* (WsCube Tech, 2025).

Peran *Java* Dalam Industri Teknologi

Java telah menjadi fondasi bagi industri teknologi modern, dengan penerapan yang luas di berbagai sektor. Dalam pengembangan aplikasi *mobile*, *java* telah menjadi bahasa utama untuk aplikasi *android* selama bertahun-tahun sebelum Kotlin muncul. Di sektor *enterprise*, bank-bank besar seperti *JPMorgan Chase* dan raksasa *e-commerce* seperti Amazon memanfaatkan *java* untuk sistem transaksi, manajemen risiko, dan layanan pelanggan yang menangani jutaan operasi setiap harinya.

Java juga memainkan peran penting dalam *big data* dengan menggunakan *framework* seperti *Hadoop* dan *Spark*, serta dalam pengembangan *web* yang dioptimalkan dengan *server* seperti *Apache Tomcat* dan *framework Spring*, yang mendukung aplikasi dalam skala besar. Perusahaan-perusahaan teknologi terkemuka seperti *Google, Netflix*, dan perusahaan *financial* bergantung pada *Java* untuk sistem kritis mereka. *Google* menggunakan *java* dalam *platform android* serta berbagai layanan *web*-nya, sedangkan *Netflix* mengandalkan *java* untuk arsitektur *microservices* yang mendukung jutaan pengguna *streaming*.

Dengan prinsip "*Write Once, Run Anywhere*", *Java* memungkinkan aplikasi dikembangkan satu kali namun dapat dijalankan di berbagai *platform*, menjadikannya pilihan yang ideal untuk perusahaan yang membutuhkan skalabilitas dan keandalan tinggi dalam operasional sehari-hari mereka.

Keunggulan lain dari *java* terletak pada ekosistem yang kaya dan komunitas pengembang yang besar, yang terus menciptakan alat, pustaka, dan *framework* baru untuk mengikuti perkembangan teknologi terkini. Dari pengembangan aplikasi *cloud-native* hingga sistem *Internet of Things* (IoT), *java* terus beradaptasi dan tetap relevan.

Bagi pemula yang ingin memulai karir di dunia teknologi, menguasai *java* membuka banyak peluang untuk berkontribusi pada proyek-proyek inovatif di berbagai industri, mulai dari *fintech* dan e-*commerce* hingga hiburan dan layanan publik, menjadikannya investasi keterampilan yang sangat berharga di era digital saat ini. *Java* sangat fleksibel dan digunakan dalam berbagai bidang:

Tabel 1.1: Fleksibilitas *Java* Dalam Berbagai Bidang

Bidang Aplikasi	Contoh Teknologi/Framework
Web Development	*Spring Boot*, Jakarta EE, *Java Servlets*
Mobile Apps	Android SDK (*Java*/Kotlin)
Desktop Applications	*Swing*, *Java*FX
Enterprise Systems	*Java* EE (sekarang Jakarta EE), *Microservices*
Big Data	*Hadoop, Apache Spark*
Internet of Things	*Embedded systems* dengan *Java* ME
Game Development	LibGDX, *jMonkeyEngine*

Sumber: Diolah Penulis.

Daftar Pustaka

Ashwani Kumar. (2025). *Java Virtual Machine (JVM): Internal Architecture and Key Concepts Explained*. Https://Prgrmmng.Com/*Java*-Virtual-Machine-Jvm-Architecture-Explained.

ByteCodeAuthor. (2024). *Java From Source to Binary*. Https://Byte-Code.Org/2024/08/20/*Java*-from-Source-to-Binary/.

Francesco Tusa. (2022). *Introduction to Computer Programs: Bytecode and Machine Code*. Https://Thinkobjectoriented.Hashnode.Dev/Introduction-to-Computer-Programs-Bytecode-Machine-Code.

Gaurav Gandhi. (2025). *What is Byte Code in Java?.* Https://Www.Naukri.Com/Code360/Library/Byte-Code-in-*Java*.

Geeksforgeeks. (2025). *Java OOP (Object Oriented Programming) Concepts.* Https://Www.Geeksforgeeks.Org/*Java*/Object-Oriented-Programming-Oops-Concept-in-*Java*/.

Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2015). Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2015). (Oracle Corporation, Ed.; 8th ed., Vol. 8).

Horstmann, C. S. (2019). *Core Java Volume II: Advanced Features (12th ed.).* Pearson Education.

IBM. (2021). *What is the Java Runtime Environment (JRE)?.* Https://Www.Ibm.Com/Think/Topics/Jre.

Lindholm, T., & Yellin, F. (1996). *The Java Virtual Machine*. Addison-Wesley Professional.

Oracle Corporation. (2023). *The Java Virtual Machine Specification. (Inc. Oracle America, Ed.).*

Oracle Documentation. (2023). *Java Platform, Standard Edition Security Developer's Guide.*

WsCube Tech. (2025). Full *History of Java Programming Language (1991-2025).* Https://Www.Wscubetech.Com/Resources/*Java*/History.

PROFIL PENULIS

Mohamad Yusuf, S.Kom., M.C.S.

Lahir di Jakarta, tahun 1976. Pendidikan S1 didapatkan dari Universitas Budi Luhur Jakarta, jurusan Teknik Informatika. Pendidikan S2 didapatkan dari Preston University, Islamabad Pakistan Jurusan Informatika, saat ini penulis sedang menyelesaikan S3 *Data Science* di Universiti Malaysia Kelantan. Penulis memiliki keahlian yang mendalam dalam *Web Technology* dan *Data Science*. Sebagai bagian dari upaya membangun karir sebagai dosen professional di Fakultas Ilmu Komputer Universitas Mercu Buana Jakarta, penulis aktif terlibat dalam riset di bidang spesialisasinya. Beberapa penelitian yang telah dilakukan didanai oleh lembaga perguruan tinggi tempat penulis mengajar. Selain itu, penulis juga rajin menulis buku dengan tujuan memberikan kontribusi positif bagi kemajuan bangsa dan negara tercinta.

Email Penulis: mhd.yusuf@mercubuana.ac.id.

BAB 2
PERSIAPAN LINGKUNGAN PENGEMBANGAN *JAVA*

Hedie Kristiawan, S.Kom., M.M.
Universitas Santo Borromeus

Di bab ini, kita akan bekerja dengan menyiapkan *Java Development Environment* (JDE) yang modern dan fleksibel, yang akan kita gunakan tidak hanya untuk aplikasi dasar tetapi juga untuk menjelajahi AI, *big data*, dan *cloud computing*. Ini merupakan langkah awal menuju pemrograman *java* masa depan.

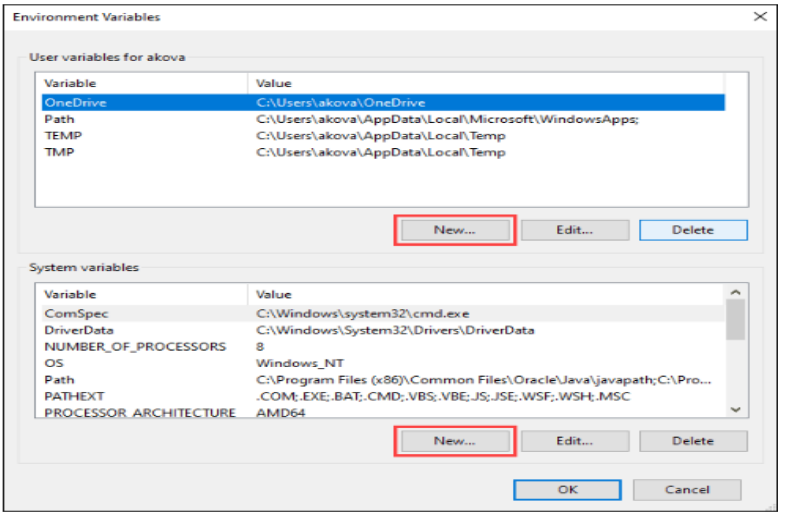
Memahami Peran *Java* di Era Modern & AI

Untuk pertama kali *java* diperkenalkan pada pertengahan tahun 1990an oleh James Gosling dan tim yang berasal dari perusahaan yang bernama *Sun Microsystems* atau lebih dikenal sebagai proyek *Oak* and *Green*, mereka menggunakan filosofi "*Write Once, Run Anywhere*" yang revolusioner di masanya. Pada tahun 1996 *java* versi 1.0 dirilis yang menjadikan bahasa pemrograman *java* berkembang pesat dalam segi fitur, performa, dan portabilitas. *Sun microsystems* mendapatkan pengaruh besar dari komunitas *java* saat mereka merilis sebagian besar kode JVM sebagai *open source* pada tahun 2006 hingga tahun 2007 agar mendukung penggunaan *java* dikenal lebih luas (John Gough, 2000).

Gambar 2.2: *Founders of Sun Microsystems*
Sumber: https://www.mmrao.com/post/108-silicon-valley-why-its-a-unique-place-on-the-planet.

Sejak era J2EE hingga Jakarta EE, *java* menjadi tumpuan dalam penggunaan aplikasi *enterprise* dan menjadi solusi berbasis *cloud* yang jarang mendapatkan kegagalan (*realibel*), ukuran dalam kemampuan yang lebih luas (*scalable*), dan aman. Di era modern ini, berbagai

2. *Setting* E*nvironment Variables*



Gambar 2.8: Menambahkan *Path*
Sumber: https://phoenixnap.com/kb/windows-set-environment-variable.

Untuk melakukannya buka *menu control panel* lalu pilih sistem. Kemudian pilih *advanced system variables*. Setelah instalasi, pastikan variabel lingkungan PATH sudah ditetapkan agar sistem mengenali perintah *java* dan *javac* di *command line*. Dalam direktori instalasi JDK, tambahkan path ke *folder bin*, contoh: C:*Program Files**Java*\jdk-17.0.1*bin*. *Path* harus disesuaikan dengan versi dan lokasi yang anda pilih pada saat instalasi. Jika sudah tekan OK untuk menyimpan perubahan.

Tip instalasi dan *troubleshooting* umum, untuk mendapatkan keamanan dan *update* terbaru, selalu *download installer* dari situs resmi. Pastikan bahwa *Java Development Kit* (JDK) yang digunakan kompatibel dengan IDE yang dipilih. Cek kembali *environment variables* (*JAVA_HOME*, *Path*) jika anda menemukan *error* "JDK *not found*". Sebelum memulai instalasi, pastikan RAM dan ruang disk Anda cukup. Untuk menyimpan konfigurasi yang anda sukai, gunakan fitur import atau *export setting* dari setiap IDE.

Untuk mengembangkan aplikasi *Java*, Anda harus instal JDK. Namun, untuk menjalankan aplikasi *Java* saja, Anda cukup menginstal

JRE (*Java Runtime Environment*). Singkatnya, JDK adalah paket lengkap yang sudah termasuk JRE di dalamnya. Jadi, jika anda ingin belajar membuat program *Java* dari awal, cukup *install* JDK. Dengan begitu, anda sudah bisa menulis kode dan menjalankannya sekaligus. Lingkungan pengembangan yang telah anda siapkan adalah pintu gerbang menuju dunia *coding* yang modern.

Dengan menginstal JDK, Anda tidak hanya mendapatkan alat untuk mengembangkan aplikasi *java*, tetapi juga pondasi yang stabil dan kokoh ditambah dengan IDE modern yang dilengkapi fitur cerdas dan *plugin* AI, anda kini memiliki asisten pribadi yang akan mempercepat proses kerja anda. Persiapan ini lebih dari sekedar instalasi, ini adalah investasi untuk menjadi *programmer* yang efisien, inovatif dan siap menghadapi tantangan di era kecerdasan buatan (AI) (Hidayat, 2024).

Daftar Pustaka

Alfa Satyaputra, M.Sc, E. (2014). *Java for Beginners with Eclipse 4.2 Juno*. Edited by Y. Whindy. Jakarta: Elex Media Komputindo. Available at: https://books.google.co.id/books?id=8NhMDwAAQBAJ&lpg=PP1&ots=mzXFm5SREQ&dq=Cara install JDK (*Java* Development Kit) di windows &lr&hl=id&pg=PP1#v=onepage&q&f=false.

Bhuman Vyas. (2023). *Java*-Powered AI: Implementing Intelligent Systems with Code, *Journal of Science & Technology (India)*, 4(6), pp. 1–12. Available at: https://doi.org/10.55662/jst.2023.4601.

Hidayat, M.S. (2024). Studi Kasus Pengembangan Dan Pemanfaatan Artificial Intelligence Sebagai Penunjang Kegiatan Masyarakat, *Qalam: Jurnal Pendidikan Islam*, 5(2). Available at: https://doi.org/10.57210/qlm.v5i2.348.

Isla Sibanda. (2025). *IDE vs. Traditional Coding: Which Approach Boosts Productivity?*, *Ambassador*. Available at: https://www.getambassador.io/blog/integrated-development-environment-explained.

Jetbrains. (2024). *Software Development Kit*, *IntelliJ IDEA*. Available at: https://www.jetbrains.com/help/idea/sdk.html.

John Gough, D.C. (2000). Implementing Languages Other than *Java* on the *Java* Virtual Machine. Available at: https://eprints.qut.edu.au/30165/.

Mohandas, R. *et al.* (2020). TensorFlow Enabled Deep Learning Model Optimization For Enhanced Realtime Person Detection Using Raspberry Pi Operating At The Edge, *CEUR Workshop Proceedings*, 2771, pp. 157–168.

Otavio Santana. (2025). *Java Enterprise Matters: Why It All Comes Back to Jakarta EE*.

PROFIL PENULIS

Hedie Kristiawan, S.Kom., M.M.

Ketertarikan penulis pada bidang manajemen dan sistem informasi komputer dimulai sejak tahun 2000. Pengalaman ini mengantarkan penulis untuk menempuh studi di Sekolah Tinggi Ilmu Komputer (STMIK) LIKMI Bandung dengan mengambil jurusan Manajemen Sistem Informasi, yang berhasil diselesaikan pada tahun 2005. Penulis kemudian melanjutkan studi pascasarjana di Universitas Pasundan Bandung pada tahun 2015, dengan mengambil program studi Manajemen dan fokus pada bidang Sistem Informasi Bisnis. Sejak tahun 2023, penulis mengabdikan diri sebagai Dosen Bisnis Digital di Universitas Santo Borromeus. Selain aktif mengajar, penulis juga berdedikasi membimbing mahasiswa, salah satunya berhasil menjuarai lomba desain *interface*. Dengan kepakaran di bidang Manajemen dan Sistem Teknologi Informasi, penulis berupaya mewujudkan karier sebagai akademisi dan praktisi profesional. Penulis juga aktif melakukan penelitian dan menulis buku sebagai wujud nyata kontribusi positif bagi kemajuan bangsa dan negara.

Email Penulis: hedie.kristiawan161@gmail.com.

BAB 3
STRUKTUR DASAR PROGRAM *JAVA*

Dr. Ir. Norbertus Tri Suswanto Saptadi, S.Kom., M.T., M.M., IPM.
Universitas Atma Jaya Makassar

Struktur Umum Program *Java*

Program *java* selalu memiliki struktur standar yang harus dan perlu untuk diikuti agar dapat dikompilasi dan dijalankan dengan baik dan benar (Putri *et al.*, 2022). Struktur ini mencakup kelas, metode, dan blok kode yang terorganisir secara rapi. *Java* adalah bahasa yang berbasis objek, sehingga seluruh kode harus berada di dalam kelas (Yanto *et al.*, 2023).

1. Komponen Utama Program *Java*

```
// Nama file: NamaKelas.java
public class NamaKelas {

  // Metode utama-titik awal program dijalankan
  public static void main(String[] args) {
    // Perintah atau instruksi program
    System.out.println("Halo, Dunia!");
  }
}
```

2. Penjelasan Komponen Struktur

Tabel 3.1: Penjelasan Komponen Struktur

Bagian	Fungsi
public class NamaKelas	Mendefinisikan kelas utama dalam program. Nama kelas harus sama dengan nama *file .java*.
{ ... }	Blok kode yang menandai awal dan akhir dari suatu kelas atau metode.
public static void main (*String*[] *args*)	Metode utama yang dieksekusi pertama kali saat program dijalankan.
System.out.println(...)	Perintah untuk mencetak teks ke layar (*console*).

Sumber: Diolah Panulis.

3. Aturan Penamaan dan *File*

a. Nama *file* harus sama persis dengan nama kelas publik (*public class*), termasuk huruf besar-kecil. Contoh: jika nama kelas *HelloWorld*, maka nama *file* harus *HelloWorld.java*.
b. *Java* bersifat *case-sensitive*: *main* dan *main* dianggap berbeda.

10. Gunakan *Java Naming Convention* Secara Konsisten

Elemen	Format	Contoh
Kelas	*PascalCase*	DataSiswa
Metode	*camelCase*	hitungTotal()
Variabel	*camelCase*	jumlahNilai
Konstanta	UPPER_CASE	MAX_NILAI

Kesimpulan: mengikuti konvensi penulisan *Java*: (1) meningkatkan keterbacaan dan kualitas kode, (2) memudahkan kolaborasi dalam tim, (3) menunjukkan profesionalisme sebagai *developer Java*.

Penutup

Membahas pondasi penting yang harus dipahami sebelum menulis program *Java* yang lebih kompleks. Berikut poin-poin utama yang perlu diingat:

1. **Struktur Program *Java* Selalu Dimulai dari Kelas**
   a. Semua kode *Java* harus berada dalam kelas.
   b. Nama kelas harus sesuai dengan nama dari *file* dan ditulis menggunakan *PascalCase*.
2. **Fungsi *Main*() Adalah Titik Masuk Program**
   a. Program *Java* dijalankan mulai dari metode:
   public static void main(*String*[] *args*)
   b. Fungsi ini wajib ada agar program bisa dijalankan.
3. **Komentar Membantu Dokumentasi dan Pemahaman**
   a. Gunakan komentar satu baris (//) atau multi-baris (/* */) untuk menjelaskan bagian penting dari kode.
   b. Komentar dokumentasi (/** */) berguna untuk *API* dan dokumentasi formal.
4. ***Statement* dan Blok Kode Menentukan Logika Program**
   a. *Statement* adalah instruksi tunggal yang diakhiri titik koma,
   b. Blok kode ({}) mengelompokkan beberapa *statement* dalam satu unit logika.
5. ***Package* dan *Import* Membantu Organisasi Proyek**
   a. *Package* digunakan untuk mengelompokkan kelas.
   b. *Import* digunakan untuk mengakses kelas dari *package* lain agar kode lebih ringkas dan terstruktur.

6. Konvensi Penulisan Membuat Kode Lebih Profesional

a. Ikuti standar penamaan seperti *camelCase* untuk variabel, *PascalCase* untuk kelas, dan UPPER_CASE untuk konstanta.
b. Gunakan indentasi 4 spasi dan selalu tulis blok kode {} meskipun hanya satu baris.

Daftar Pustaka

Ida *et al.* (2024). *Pemrograman Berorientasi Objek*, *PT Inovasi Pratama Internasional*. Padang.

Ilham, A.A. *et al.* (2011). Meningkatkan Kinerja *Java* Virtual Machine dengan Mengoptimalkan Peran Garbage Collector, pp. 8–9.

Nita, S. (2021). *Job Sheet Pemrograman Berorientasi Objek With Java (Teori dan Implementasi Java)*. Madiun: UNIPMA Press Universitas PGRI Madiun.

Pratiwi, E.L. (2020). *Konsep Dasar Algoritma dan Pemrograman dengan Bahasa Java*, *Poliban Press*. Banjarmasin: Poliban Press.

Putri, M.P. *et al.* (2022). *Algoritma dan StrukturData*. Bandung: Penerbit Widina Bhakti Persada.

Rachman, A., Rochimah, S. & Sunaryono, D. (2016). Komentar Semi Otomatis untuk Memudahkan Pemahaman pada Bahasa Pemrograman *Java*, *Jurnal Ilmiah NERO*, 2(3), pp. 145–152.

Syamsudin, A. (2020). Analisis Kesalahan Coding Pemrograman *Java* pada Matakuliah Algoritma Pemrograman Mahasiswa Tadris Matematika IAIN Kediri, *Journal Focus Action of Research Mathematic (Factor M)*, 2(2), pp. 102–114.

Wibowo, I.A., Kom, M. and Si, M. (2021). *Tips dan Tricks Wawancara Kerja Menjadi Programming*. Semarang: Universitas Sains & Teknologi Komputer.

Widayat, A. *et al.* (2023). Pemahaman Konsep Dasar *Java*: Pendekatan Praktis untuk Mahasiswa, *Buletin Ilmiah Ilmu Komputer dan Multimedia*, 1(4), pp. 619–622.

Yanto, M. *et al.* (2023). Pemrograman Menggunakan *Java* NetBeans, *BIIKMA: Buletin Ilmiah Ilmu Komputer dan Multimedia*, 1(3), pp. 367–377.

Zain Arif Wildan Sugandi *et al.* (2022). Implementasi Konsep Pemrograman Berorientasi Objek Dalam Aplikasi Pembukuan Keuangan Penjual Jus Buah Menggunakan Bahasa Pemrograman *Java*, *Jurnal IT CIDA*, 8(1), pp. 1–8.

Zulfaa, A. *et al.* (2023). Pengembangan Aplikasi E-Learning Berbasis *Java*/NetBeans Untuk Peningkatan Pembelajaran Interaktif', *BIIKMA: Buletin Ilmiah Ilmu Komputer dan Multimedia*, 1(3), pp. 357–360.

PROFIL PENULIS

Dr. Ir. Norbertus Tri Suswanto Saptadi, S.Kom., M.T., M.M., IPM.

Lahir di Cirebon, Jawa Barat, tanggal 7 Juni 1975. Memiliki Jabatan Fungsional Lektor Kepala, Pembina Tingkat I (IV/b). Berpendidikan Sarjana Komputer (S.Kom.) di Universitas Teknologi Digital Indonesia (UTDI) tahun 1998, Magister Manajemen (M.M.) di Universitas Hasanuddin (UNHAS) tahun 2004, Magister Teknologi Informasi (M.T.) di Universitas Gadjah Mada (UGM) tahun 2007, Insinyur (Ir.) di Pendidikan Profesi Insinyur UNHAS tahun 2020, Insinyur Profesional Madya (IPM.) di Persatuan Insinyur Indonesia (PII) tahun 2021, Doktor (Dr.) di Fakultas Teknik UNHAS tahun 2023, Kursus Kader Pimpinan (Suskapin) XXVI Menwa RI tahun 1997, dan Program Pendidikan Reguler Angkatan (PPRA) LX Lemhannas RI tahun 2020.

Menjadi tenaga pengajar (Dosen) pada Program Studi Teknik Informatika Fakultas Teknologi Informasi Universitas Atma Jaya Makassar (UAJM). Peraih Poster terbaik DPRM Dikti tahun 2016. Dosen berprestasi IKDKI tahun 2020, 2021, dan 2024. Pernah menjabat Kepala UPT Komputer, Kepala BAPSI, Wakil Dekan FT, Dekan FT dan FTI, Wakil Rektor III, Ketua Penjaminan Mutu. Tim PAK Dosen dan Asesor BKD UAJM. *Reviewer International Conference*, *International Journal,* Seminar Nasional, dan Jurnal SINTA. Pemenang Hibah Kemdikbud Penelitian Dosen Muda, Dosen Pemula, Bersaing, Fundamental, dan Strategi Nasional.

Penulis artikel media massa Tribun Timur, Koinonia, Bisnis Sulawesi, Sesawi.net, Mirifica.net, HidupKatolikCom, OMKNet, KatolikanaTV, Jalan Hidup Katolik, dll. Penulis Buku di Kanisius, Sada Kurnia Pustaka, Aksara Sastra Media, *Future Science*, HEI *Publishing*, Mifandi Mandiri Digital, Rey Media Grafika, Widina Salemba, Andi, dan Cendikia Mulia Mandiri. Aktifis organisasi IKA Lemhannas RI LX, IARMI, DPP ISKA, BAPOMI Sulsel, LP3KD Sulsel, IKDKI SulSelTraBar, Komkep KAMS, Komsos KAMS, PUKAT KAMS, TPP KAMS, FMKI KAMS,

UPS KAMS, Pengurus Kebun Sawit Laimbo, FDI, PII Makassar, INAPR, Dewan Keuangan Paroki dan Program Ayo Sekolah Mariso, Animator Laudato Si', TKN GLSI', dll.

Email Penulis: ntsaptadi@gmail.com.

BAB 4
TIPE DATA DAN VARIABEL DALAM *JAVA*

Hendri Julian Pramana, S.Kom., M.Kom.
Universitas Garut

Pendahuluan

Tipe data dan variabel merupakan elemen dasar yang penting dalam pemrograman *java*. Tipe data menentukan jenis nilai yang dapat disimpan, sementara variabel berfungsi sebagai wadah penyimpanan data dalam program. *Java* menggunakan sistem tipe data statis yang kuat, sehingga setiap variabel harus dideklarasikan dengan tipe tertentu sebelum digunakan, memungkinkan deteksi kesalahan sejak awal kompilasi (Cosmina, 2024; Ogihara, 2018).

Bab ini akan membahas identifier, tipe data primitif dan non-primitif, deklarasi dan inisialisasi variabel, konstanta, serta teknik konversi tipe data. Pemahaman topik ini akan membantu pembaca dalam menulis kode program yang benar, efisien, dan sesuai dengan standar bahasa pemrograman *java* (GoalKicker.com, 2018; Schildt, 2014).

Identifier dan *Reserved Word* Pemrograman *Java*

1. Konsep *Identifier*

Dalam bahasa pemrograman, *identifier* adalah nama pengenal yang digunakan untuk menandai elemen-elemen dalam kode program, seperti nama variabel, kelas, metode, dan objek. *Identifier* berperan sebagai label agar program dapat mengenali dan mengakses data atau fungsi tertentu yang telah dibuat. *Identifier* harus bersifat unik dan ditulis mengikuti aturan sintaksis bahasa pemrograman yang digunakan (Ahmadian et al., 2017; Ogihara, 2018).

Gambar 4.1: Ilustrasi Konsep *Identifier*
Sumber: *Generate by Gemini* AI

Dalam konteks pemrograman, *programmer* seperti ibu dalam gambar yang sedang berusaha menjelaskan nama atau penulisan (*"book"* bukan "bukz", *"knife"* bukan "knaif"), serta kegunaan dari

Berikut ini contoh kode yang menggambarkan ketiga jenis tersebut secara utuh:

```java
public class Main {

  // Variabel statis (class variable)
  static String institusi = "Universitas Garut";

  // Variabel instance
  String nama;
  int umur;

  // Konstruktor
  Main(String nama, int umur) {
    this.nama = nama; // menginisialisasi variabel instance
    this.umur = umur;
  }

  // Method
  void tampilkanInfo() {
    // Variabel lokal
    String info = "Mahasiswa Aktif";
    System.out.println("Nama      : " + nama);
    System.out.println("Umur      : " + umur + " tahun");
    System.out.println("Status    : " + info);
    System.out.println("Institusi : " + institusi);
  }

  public static void main(String[] args) {
    Main mhs1 = new Main("Budi", 20);
    mhs1.tampilkanInfo();
  }
}
```

Output Program:

```
Your Output

Nama      : Budi
Umur      : 20 tahun
Status    : Mahasiswa Aktif
Institusi : Universitas Garut
```

Penjelasan:

a. Info adalah variabel lokal, karena hanya berlaku dalam metode tampilkanInfo().
b. Nama dan umur adalah variabel *instance*, karena dideklarasikan dalam kelas namun bukan bagian dari *method* dan tidak menggunakan *static*.
c. Institusi adalah variabel statis, karena dideklarasikan dengan kata kunci *static* dan dimiliki bersama oleh semua objek kelas tersebut.

Konstanta Pada Pemrograman *Java*

Konstanta adalah variabel khusus yang nilainya tetap dan tidak dapat diubah setelah diberikan untuk pertama kalinya. Penggunaan konstanta biasanya untuk menjaga nilai tetap yang universal dalam program, seperti angka `phi`, batas maksimum, atau konfigurasi tetap lainnya. Dalam *Java*, aturan penamaan sebuah konstanta adalah sebagai berikut (Schildt, 2014):

1. Dideklarasikan dengan kata kunci final sebelum tipe data.
2. Dilakukan inisialisasi saat deklarasi.
3. Menggunakan huruf kapital dan pemisah *underscore* [_] sebagai konvensi penamaan.

Contoh Penggunaan:

```
//Deklarasi Konstanta
  final double PI = 3.14159;
  final int MAKS_SKOR = 100;

/* Jika mencoba mengubah nilainya setelah deklarasi,
  maka akan muncul error saat kompilasi: */

  PI = 3.14; // Error: cannot assign a value to final variable 'PI'
```

Daftar Pustaka

Ahmadian, H., Mizuardy, H., & AR, K. (2017). *Mahir Pemrograman Visual Dengan Java*. Unimal Press. http://dx.doi.org/10.1016/j.bpj.2015.06.056%0Ahttps://academic.oup.com/bioinformatics/article-abstract/34/13/2201/4852827%0Ainternal-pdf://semisupervised-3254828305/semisupervised.ppt%0Ahttp://dx.doi.org/10.1016/j.str.2013.02.005%0Ahttp://dx.doi.org/10.10.

Cosmina, I. (2024). *Java* 23 for Absolute Beginners. In *Java 23 for Absolute Beginners*. Apress. https://doi.org/10.1007/979-8-8688-1041-1.

GoalKicker.com. (2018). *Java Notes for Professional*. GoalKicker.com.

Ogihara, M. (2018). Fundamentals of *Java* Programming. In *Fundamentals of Java Programming*. Springer International Publishing. https://doi.org/10.1007/978-3-319-89491-1.

Schildt, H. (2014). *Java: The Complete Reference (9th ed.).* McGraw-Hill Education Group.

PROFIL PENULIS

Hendri Julian Pramana, S.Kom., M.Kom. Adalah seorang akademisi di bidang pengembangan Rekayasa Perangkat Lunak (RPL). Ketertarikan penulis terhadap dunia komputer dan teknologi sudah tumbuh sejak masa sekolah menengah pertama. Belajar secara otodidak dan lembaga-informal yang terbatas tidak menghalangi keinginan untuk melanjutkan pendidikan tinggi di bidang Teknik Informatika setelah lulus Sekolah Menengah Atas di Kota Tasikmalaya. Penulis berhasil menyelesaikan studi Sarjana Komputer dan kemudian melanjutkan pendidikan Magister Komputer pada kampus UDINUS Semarang dan lulus pada tahun 2020. Saat ini, penulis menjadi dosen tetap untuk Prodi Rekayasa Perangkat Lunak, Fakultas Komunikasi dan Informasi (FKOMINFO), Universitas Garut (UNIGA). Penulis mengampu beberapa mata kuliah di antaranya: Sistem Terdistribusi, Pemrograman Bergerak, Sistem Pendukung Keputusan, Peretasan Beretika (*Ethical Hacking*), dan *Design Thinking*. Fokus dan ketertarikan penulis dalam bidang keilmuan meliputi *design thinking*, *decision support system*, *machine learning*, pengenalan objek menggunakan *computer vision*, serta *Internet of Things* (IoT). Selain mengajar, penulis juga berkomitmen untuk terus aktif melakukan penelitian dan meningkatkan pengetahuan melalui kegiatan menulis buku guna memperkuat kontribusi pada dalam pengembangan pengetahuan dan teknologi di Indonesia.

Email Penulis: hendri.jp@uniga.ac.id.

BAB 5
OPERATOR DALAM *JAVA*

Agung Yuliyanto Nugroho, M.Kom., M.Par.
Universitas Cendekia Mitra Indonesia

Pendahuluan

Perkembangan teknologi informasi dan komunikasi yang pesat Dalam dunia pemrograman, operator merupakan elemen dasar yang digunakan untuk membentuk ekspresi dan melakukan manipulasi data. Dalam bahasa pemrograman *java*, operator memainkan peran penting dalam hampir setiap aspek penulisan program, mulai dari perhitungan aritmatika sederhana hingga evaluasi kondisi logika yang kompleks.

Java menyediakan berbagai jenis operator yang memungkinkan programmer untuk mengolah nilai dan variabel secara efisien, konsisten, dan fleksibel. Operator dalam *java* dirancang untuk bekerja dengan berbagai tipe data, baik tipe primitif seperti *int, double*, dan *boolean*, maupun tipe objek. *Java* juga mendukung penggunaan operator dalam bentuk ekspresi yang kompleks, termasuk kombinasi antara operator aritmatika, logika, relasional, dan penugasan.

Pemahaman yang baik tentang jenis-jenis operator, serta cara penggunaannya yang benar, merupakan bagian fundamental dalam penguasaan bahasa *java*. Tanpa pemahaman tersebut, akan sulit bagi seorang *programmer* untuk menulis kode yang benar, efisien, dan mudah dipahami. Bab ini akan membahas berbagai jenis operator yang tersedia dalam *java*, mulai dari operator aritmatika hingga operator khusus seperti *instanceof* dan *operator ternary.*

Dengan mempelajari operator secara sistematis, pembaca diharapkan mampu menggunakannya secara tepat dalam membangun logika dan struktur program yang efektif.

Operator Aritmatika

Operator aritmatika dalam *java* digunakan untuk melakukan operasi matematika dasar seperti penjumlahan, pengurangan, perkalian, pembagian, dan *modulus* (sisa bagi). *Operator* ini bekerja pada tipe data numerik, seperti *int, float, double*, dan *long*.

Operator aritmatika sangat penting karena menjadi dasar dari berbagai operasi perhitungan dalam pemrograman. *Java* menyediakan lima *operator* aritmatika utama, sebagaimana dijelaskan pada tabel berikut:

Keduanya menghasilkan hasil yang sama, tetapi operator *ternary* membuat kode menjadi lebih ringkas dan ekspresif, khususnya dalam kasus yang sederhana.

Contoh lain penggunaan operator *ternary:*

```
int a = 10, b = 20;
int max = (a > b) ? a : b;
System.out.println("Nilai terbesar: " + max);
Output: Nilai terbesar: 20
```

Operator *Instanceof*

Dalam bahasa pemrograman *java*, operator *instanceof* digunakan untuk memeriksa apakah suatu objek merupakan *instance* (turunan) dari kelas tertentu, atau merupakan bagian dari hierarki pewarisan suatu kelas atau antarmuka. Operator ini menghasilkan nilai *boolean true* jika objek tersebut merupakan *instance* dari tipe yang ditentukan, dan *false* jika tidak.

Operator *instanceof* sangat berguna dalam pemrograman berorientasi objek, terutama ketika bekerja dengan pewarisan (*inheritance*), polimorfisme (*polymorphism*), dan *downcasting* objek.

Sintaks Operator *instanceof*

```
objek instanceof NamaKelas
```

1. objek: variabel yang mereferensikan sebuah objek.
2. NamaKelas: tipe data atau kelas yang akan diuji.

Contoh penggunaan dasar:

```
class Hewan {}
class Kucing extends Hewan {}

public class ContohInstanceof {
   public static void main(String[] args) {
      Hewan h = new Kucing();

      System.out.println(h instanceof Kucing); // true
      System.out.println(h instanceof Hewan);  // true
      System.out.println(h instanceof Object); // true
   }
}
```

Daftar Pustaka

Deitel, P. J., & Deitel, H. M. (2017). *Java: How to program (11th ed.).* Pearson Education.

Eckel, B. (2006). *Thinking in Java (4th ed.).* Prentice Hall.

Farrell, J. (2019). *Java programming (9th ed.).* Cengage Learning.

Flanagan, D. (2020). *Java in a nutshell (8th ed.).* O'Reilly Media.

Horstmann, C. S. (2019). *Core Java Volume I-Fundamentals (11th ed.).* Pearson Education.

Liang, Y. D. (2022). *Introduction to Java Programming and Data Structures: Comprehensive version (13th ed.).* Pearson Education.

Lindholm, T., Yellin, F., Bracha, G., & Buckley, A. (2021). *The Java® Virtual Machine Specification: Java SE 17 Edition.* Oracle America, Inc.

McKenzie, B. (2022). *OCP Oracle Certified Professional Java SE 17 Developer Study Guide: Exam 1Z0-829.* Wiley.

Schildt, H. (2018). *Java: The Complete Reference (11th ed.).* McGraw-Hill Education.

PROFIL PENULIS

Agung Yuliyanto Nugroho, M.Kom., M.Par.

Ketertarikan penulis terhadap ilmu komputer dimulai pada tahun 2015 silam. Hal tersebut membuat penulis melanjutkan pendidikan ke Perguruan Tinggi dan berhasil menyelesaikan studi S1 di prodi Teknik Informatika Universitas Teknologi Yogyakarta pada tahun 2018. Dua tahun kemudian, penulis menyelesaikan studi S2 di prodi Teknik Informatika Program Pasca Sarjana Universitas Amikom Yogyakarta dan juga prodi Magister Pariwisata di Sekolah Tinggi Pariwisata Ambarrukmo Yogyakarta. Atas dedikasi dan kerja keras dalam membuat suatu karya, Republik Indonesia Kementerian Hukum Dan Hak Asasi Manusia sudah mencatat ada kurang lebih 100 karya yang sudah tercatat di surat pencatatan ciptaan sebagai salah satu kontribusi dalam melindungi hak kekayaan intelektual.

Email Penulis: agungboiler11@gmail.com.

BAB 6
INPUT DAN *OUTPUT* (I/O) DASAR

Nina Rahayu, S.Kom., M.M., M.T.I.
Universitas Raharja

Pendahuluan

1. Pengertian I/O Dalam Pemrograman

Input adalah proses menerima data dari pengguna atau sumber lain (*keyboard, file*), sedangkan *output* adalah proses menampilkan atau menyimpan data (misalnya ke layar). Studi Sulír *et al.* (2023) menunjukkan bahwa metode I/O ini sangat umum digunakan dalam kode *java* nyata, sehingga pemahaman dasar I/O menjadi sangat penting dalam pembelajaran *Java* awal.

2. Peran I/O Dalam Aplikasi *Java*

Tanpa I/O, program tidak memiliki interaksi dengan pengguna atau data nyata. Menurut Tigina *et al.* (2023), kesalahan dalam I/O termasuk salah satu masalah umum yang menurunkan kualitas kode dalam kursus pemrograman online berbasis *Java* (Tigina *et al,* 2023). Sedangkan metode pengajaran berbasis proyek (seperti yang diuraikan oleh Cai Yang, 2020) memperkuat pemahaman konsep I/O melalui keterlibatan dalam tugas nyata seperti *input* pengguna dan *output* hasil dalam aplikasi mini berbasis teks.

3. Tipe-Tipe *Input* dan *Output* (I/O) di *Java*

Java mendukung beberapa jenis I/O (*Input* dan *Output*), dan masing-masing punya cara penggunaan serta tujuan tersendiri. Berikut adalah 3 tipe umum:

a. *Console* I/O (Fokus utama)

Console I/O adalah proses membaca *input* dari *keyboard* dan menampilkan *output* ke layar (*console*). Ini adalah bentuk I/O paling sederhana dan paling umum digunakan saat belajar dasar *java*. *Console* I/O menggunakan *System.in* dan *System.out* untuk *input*/*output* melalui terminal.

1) *Input*

System.in: untuk *input byte* mentah dari *keyboard*.
Scanner: untuk *input* teks atau angka dari *console* dengan cara yang mudah.
BufferedReader: untuk *input* dari *console* secara efisien (per baris).

Daftar Pustaka

Oracle. (2024). *Class Scanner (Java SE 8 Documentation)*. Oracle Corporation. Access 26/07/2025 (11.30).

Pendergast, M. O. (2006). *Teaching Introductory Programming to IS Students: Java Problems and Pitfalls*. *Journal of Information Technology Education: Research*, Vol. 5, pp. 491–515. doi: https://doi.org/10.28945/261.

Sulir, M., Chodarev, S., & Nosáľ, M. (2023). Outside the Sandbox: A Study of Input/Output Methods in *Java*. *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering (EASE '23), ACM, 2023, pp. 253-258*. doi: https://doi.org/10.1145/3593434.3593501.

Tigina, M., Birillo, A., Golubev, Y., Keuning, H., Vyahhi, N., Bryksin., T. (2023). Analyzing the Quality of Submissions in Online Programming Courses. *ArXiv: Computer Science, Software Engineering.* doi: https://doi.org/10.48550/arXiv.2301.11158.

Yang, C. (2020). Research on *Java* Programming Course Based on CDIO and Iterative Engineering Teaching Pattern. *Recent Advances in Computer Science and Communications,* Volume 13, Issue 3, Jun 2020, p. 519 - 530. doi: https://doi.org/10.2174/2213275912666190819103333.

Yuan, X., Wan, J., An, D., Lu, J., & Yuan, P. (2024). Multi-Method Integrated Experimental Teaching Reform of A Programming Course Based on The OBE-CDIO Model Under The Background of Engineering Education. *Scientific Reports* 14, 16623 (2024). Doi: https://doi.org/10.1038/s41598-024-67667-6.

PROFIL PENULIS

Nina Rahayu, S.Kom., M.M., M.T.I.

Nina Rahayu adalah seorang akademisi dan peneliti di bidang Teknologi Informasi dengan pengalaman lebih dari satu dekade. Saat ini, ia berperan sebagai dosen di Universitas Raharja sejak tahun 2016. Dengan latar belakang pendidikan di bidang Manajemen dan Teknologi Informasi, ia aktif dalam pengajaran, penelitian, dan pengembangan sistem berbasis teknologi. Sebagai seorang peneliti, Nina telah menulis berbagai artikel ilmiah dan berpartisipasi dalam hibah penelitian, termasuk proyek-proyek seperti prediksi biaya kuliah menggunakan kecerdasan buatan, sistem peringatan dini kebakaran berbasis sensor *fusion*, dan pengembangan sistem keamanan data berbasis *blockchain* untuk *e-voting*.

Selain mengajar dan meneliti, Nina juga memiliki pengalaman dalam implementasi sistem teknologi di industri, seperti keterlibatannya dalam proyek ITM di PT Telkom Sigma. Komitmennya terhadap dunia akademik dibuktikan dengan kelanjutan studinya pada program doktoral Ilmu Komputer di Universitas Kristen Satya Wacana, Salatiga, dimana ia terus mengembangkan riset-riset terkini di bidang kecerdasan buatan, sistem informasi, dan pemanfaatan teknologi dalam sektor pendidikan. Dengan dedikasi yang tinggi dalam dunia pendidikan dan teknologi, Nina Rahayu terus berkontribusi dalam inovasi sistem informasi dan kecerdasan buatan, serta berperan dalam mencetak generasi muda yang siap menghadapi tantangan *digital*.

Email Penulis: ninarahayu.niez@gmail.com.

BAB 7
KONTROL ALUR PROGRAM DALAM *JAVA*

Deden Rustiana, M.Kom.
Universitas Raharja

Pendahuluan

Control alur program (*control flow*) adalah salah satu konsep fundamental dalam pemrograman yang memungkinkan kita mengatur bagaimana instruksi-instruksi dalam program dieksekusi. Dalam *java*, kontrol alur program memungkinkan kita untuk membuat keputusan, mengulang blok kode, dan mengatur jalur eksekusi program berdasarkan kondisi tertentu. Secara *default*, program *java* dieksekusi secara sekuensial dari atas ke bawah. Namun dengan menggunakan struktur kontrol alur, kita dapat mengubah urutan eksekusi ini untuk membuat program yang lebih dinamis dan interaktif.

Struktur Percabangan (*Branching*)

Struktur percabangan (*branching*) adalah salah satu bentuk kontrol alur dalam pemrograman yang digunakan untuk menentukan keputusan. Dengan struktur ini, program bisa mengeksekusi blok kode tertentu berdasarkan kondisi tertentu yang bernilai benar (*true*) atau salah (*false*).

1. *If*

Pernyataan if digunakan untuk mengeksekusi blok kode hanya jika suatu kondisi bernilai benar (*true*).

a. Sintaks standar dari pernyataan *if.*

```
if (kondisi) {
    // blok kode yang dijalankan jika kondisi benar
}
```

b. Contoh Penggunaan *if* dalam Program *Java*

```
int nilai = 85;

if (nilai >= 75) {
    System.out.println("Selamat, Anda lulus!");
}
```

c. Penjelasan

Jika variabel nilai lebih besar atau sama dengan 75, maka program mencetak pesan “Selamat, Anda lulus!”. Jika tidak, program tidak melakukan apa pun. Jika hanya ada satu pernyataan, tanda kurung {} boleh dihilangkan, tapi tidak disarankan karena dapat menimbulkan *bug*:

1. *Break*

Digunakan untuk menghentikan perulangan secara paksa meskipun kondisinya masih *true*.

a. Karakteristik dari pernyataan *break.*
   Langsung keluar dari perulangan (bukan hanya satu iterasi, tapi seluruh *loop*). Umumnya dipakai saat terjadi kondisi tertentu yang harus menghentikan *loop*.

b. Contoh penggunaan *break loop* dalam program *java*

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        break;
    }
    System.out.println("i: " + i);
}
```

Perulangan berhenti saat i == 5, karena ada *break.*

2. *Continue*

Xxxxx xxxxxxx xxxxxx xxxxxx xxxxxxxxxxxxxxxxxxxxx xxxxxxxx xxxxxxxx

a. Karakteristik dari pernyataan *continue*
   Tidak menghentikan *loop*, hanya melewati *sisa blok* pada satu putaran. Berguna saat ingin mengabaikan nilai tertentu atau melompati kondisi khusus.

b. Contoh penggunaan *continue loop* dalam program *java*

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

i = 3 tidak dicetak karena iterasi ke-3 dilewati dengan *continue.*

3. *Return*

a. Kegunaan dari pernyataan *return*
   Mengakhiri eksekusi suatu metode (*function*) kapan pun. Bisa juga mengembalikan nilai (kalau metode tidak *void*). Jika dipakai di dalam *loop* dalam sebuah metode, maka program:

1) Langsung keluar dari metode (bukan hanya dari *loop*).
2) Tidak melanjutkan kode setelah *return*.

b. Contoh penggunaan *return loop* dalam program *java*

```
public static void cekAngka() {
    for (int i = 1; i <= 10; i++) {
        if (i == 4) {
            return; // keluar dari metode
        }
        System.out.println("i: " + i);
    }
    System.out.println("Baris ini tidak akan dieksekusi.");
}
```

Kesimpulan

Kontrol alur program adalah fondasi penting dalam pemrograman *Java* yang memungkinkan kita membuat aplikasi yang dinamis dan interaktif. Dengan menguasai struktur percabangan (*if-else*, *switch*), perulangan (*for, while*, *do-while*), dan pernyataan kontrol (*break, continue, return*), kita dapat mengatur logika program dengan efisien. Kunci untuk menguasai kontrol alur program:

1. Memahami kapan menggunakan struktur yang tepat *if-else* untuk kondisi sederhana, *switch* untuk multiple nilai, *for* untuk iterasi yang diketahui, *while* untuk kondisi yang tidak pasti.
2. Menulis kode yang *readable* dan *maintainable*, gunakan variabel *boolean* untuk kondisi kompleks, beri nama yang bermakna.
3. Menghindari *infinite loops*, pastikan kondisi *loop* bisa berubah
4. Mengoptimalkan performa, hindari operasi mahal dalam kondisi *loop*.
5. Praktik yang konsisten, gunakan indentasi yang benar dan konvensi *naming* yang konsisten.

Daftar Pustaka

Bloch, J. (2017). *Effective Java (3rd ed.).* Addison-Wesley Professional.

Deitel, P. J., & Deitel, H. M. (2019). *Java: How to Program (11th ed.).* Pearson Education.

Eckel, B. (2006). *Thinking in Java (4th ed.)*. Prentice Hall.

Gosling, J., Joy, B., Steele, G., Bracha, G., Buckley, A., Smith, D., & Bierman, G. (2021). *The Java Language Specification* (*Java* SE 17 ed.). Oracle America, Inc.

Horstmann, C. S. (2019). *Core Java Volume I - Fundamentals* (11th ed.). Pearson.

Lindholm, T., Yellin, F., Bracha, G., Buckley, A., & Smith, D. (2021). *The Java Virtual Machine Specification (Java SE 17 ed.)*. Oracle America, Inc.

Oracle Corporation. (2023). *The Java™ Tutorials - Control Flow Statements*. Retrieved from https://docs.oracle.com/*java*se/tutorial/*java*/nutsandbolts/flow.html.

Oracle Corporation. (2023). *Java Platform, Standard Edition & Java Development Kit Version 17 API Specification*. Retrieved from https://docs.oracle.com/en/*java*/*java*se/17/docs/api/.

Oracle. (2024). *The Java™ Tutorials-Statement in Java.* https://docs.oracle.com/*java*se/tutorial/*java*/nutsandbolts/if.html.

Pendergast, M. O. (2006). Teaching Introductory Programming to IS Students: *Java* Problems and Pitfalls. *Journal of Information Technology Education: Research,* 5, 491–515. https://www.informingscience.org/Publications/261.

Schildt, H. (2018). *Java: The Complete Reference* (11th ed.). McGraw-Hill Education.

Sierra, K., & Bates, B. (2005). *Head First Java (2nd ed.).* O'Reilly Media.

PROFIL PENULIS

Deden Rustiana, M.Kom.

Penulis merupakan dosen tetap di Universitas Raharja sejak tahun 2016, dengan fokus pada pengajaran dan pengembangan keilmuan di bidang Teknologi Informasi. Beliau mengampu berbagai mata kuliah seperti Sistem Informasi, Pemrograman, Database, serta Manajemen Proyek TI. Selain aktif dalam dunia akademik, Deden juga berkiprah sebagai konsultan IT untuk berbagai perusahaan swasta dan instansi pemerintahan. Pengalaman konsultasinya mencakup pengembangan sistem informasi, audit TI, transformasi digital, serta integrasi teknologi berbasis *web* dan *mobile*. Dengan latar belakang pendidikan Magister Ilmu Komputer dari Universitas Budi Luhur, Deden Rustiana terus berkontribusi dalam pengembangan keilmuan TI, baik melalui pengajaran, penelitian, maupun implementasi langsung di dunia industri dan pemerintahan.

Email Penulis: deden.rustiana@raharja.info.

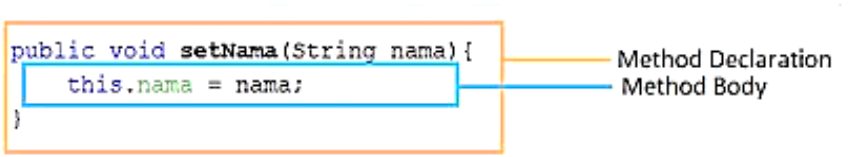
BAB 8
METODE (*FUNCTION*) DALAM *JAVA*

Ir. Gde Brahupadhya Subiksa, S.Kom., M.T.
Politeknik Negeri Bali

Pengertian Metode Dalam *Java*

Pemrograman berorientasi objek (OOP) merupakan salah satu paradigma pemrograman yang paling populer dan banyak digunakan, dengan *java* sebagai salah satu bahasa pemrograman utama yang mendukung OOP (Sugandi, dkk., 2022). Salah satu konsep utama dalam OOP adalah metode (atau dalam bahasa lain sering disebut sebagai fungsi atau *procedures*). Pada *java*, metode digunakan untuk mendefinisikan tindakan yang dapat dilakukan oleh objek. Metode berfungsi untuk mengatur alur eksekusi program, memisahkan logika program, dan memungkinkan program menjadi lebih modular, terstruktur, dan dapat dipelihara dengan mudah. Metode memungkinkan pengembang untuk membagi tugas-tugas besar menjadi unit-unit yang lebih kecil yang bisa dikelola. Tanpa metode, program akan menjadi sangat panjang dan tidak terstruktur, sehingga sulit untuk dipahami dan di-*debug.*

Metode adalah sekumpulan instruksi atau kode yang terorganisir dengan tujuan tertentu dan dapat dipanggil dalam program untuk menjalankan suatu aksi atau tugas. Metode berfungsi untuk mengabstraksi implementasi sebuah proses, sehingga pengembang tidak perlu menulis ulang kode yang sama berulang-ulang. Pada *Java*, metode adalah blok kode yang mendefinisikan suatu aksi yang dapat dilakukan dalam program. Setiap kali metode dipanggil, komputer akan mengeksekusi kode yang ada dalam metode tersebut. Metode dapat dipanggil lebih dari satu kali dalam berbagai bagian program tanpa harus menulis ulang kode yang sama. Misalnya, jika kita ingin menambahkan dua angka dalam program, kita bisa membuat sebuah metode tambah() yang bisa dipanggil kapan saja kita membutuhkan hasil penjumlahan tersebut. Penggunaan metode akan meningkatkan efisiensi, modularitas, dan pemeliharaan kode (Pradeka, dkk., 2024).



Gambar 8.1: Bagian *Method*/Metode Pada *Java*
Sumber: Diolah Penulis.

```
 public static int hitungLuas(int panjang, int
lebar) {
      try {
      int luas = panjang * lebar;
      return luas;
      // Kode yang mungkin menyebabkan pengecualian
      } catch (ExceptionType e) {
      System.out.println("Error, " + e);
    // Penanganan pengecualian
     }
 }
```

Try block: bagian dari kode yang berpotensi menyebabkan pengecualian. Program akan mencoba mengeksekusi kode di dalam blok ini. *Catch block*: bagian ini menangkap pengecualian yang terjadi di dalam blok *try*. Jika pengecualian terjadi, eksekusi program akan beralih ke blok *catch*, dan pengecualian dapat ditangani di sini.

Daftar Pustaka

Asnawi, N. (2023). *Teori dan Praktik Object Oriented Programming Menggunakan Java.*

Endra, R. Y., Kom, S., & Kom, M. (2016). *Belajar Mudah Algoritma Pemrograman Java.* Bandar Lampung: Nida Dwi Karya Publishing.

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

Pradeka, D., Adiwilaga, A., Agustini, D. A. R., & Hidayatullah, M. B. (2024). *Belajar Dasar Pemrograman Berorientasi Objek Serta Create, Read, Update, dan Delete di Database MySQL*. Mega Press Nusantara.

Rahmawati, E., Medina, P., & Azhari, D. S. (2024). Rekursif Dalam Pemrograman Teori Dan Praktek. *Innovative: Journal Of Social Science Research,* 4(4), 5622-5630.

Sianipar, R. H. (2015). *Pemrograman Java Untuk Programmer (Vol. 1).* Penerbit ANDI.

Sugandi, Z. A. W., Nugraha, Y. A., Anam, S. N., & Darmayanti, I. (2022). Implementasi Konsep Pemrograman Berorientasi Objek Dalam Aplikasi Pembukuan Keuangan Penjual Jus Buah Menggunakan Bahasa Pemrograman *Java*. *Jurnal Ilmiah IT CIDA*, 8(1), 1-8.

PROFIL PENULIS

Ir. Gde Brahupadhya Subiksa, S.Kom., M.T.

Praktisi dan Dosen Vokasi pada Program Studi Teknologi Rekayasa Perangkat Lunak, Jurusan Teknologi Informasi, Politeknik Negeri Bali. Penulis lahir di Denpasar tanggal 31 Agustus 1991. Penulis adalah dosen tetap pada Program Studi Teknologi Rekayasa Perangkat Lunak, Jurusan Teknologi Informasi, Politeknik Negeri Bali. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika Universitas Sanata Dharma Yogyakarta dan melanjutkan S2 pada Jurusan Teknik Elektro (Manajemen Sistem Informasi dan Komputer) Universitas Udayana Bali, dan Mengambil Gelar Profesi Insinyur Teknologi Informasi di Universitas Udayana Bali. Penulis menekuni bidang Teknologi Informasi dan Sistem Informasi. Besar harapan penulis dapat berkolaborasi dalam penulisan Jurnal dan Buku. Penulis juga aktif dalam Penelitian dan Publikasi Ilmiah dengan sub bidang keilmuan fokus pada Manajemen Sistem, *Human Computer Interaction*, *Usability* dan *Digital Heritage*.

Email Penulis: brahupadhya@pnb.ac.id.

BAB 9
PEMROGRAMAN BERORIENTASI OBJEK (OOP) DASAR

Eka Purnama Harahap, S.Kom., M.TI.
Universitas Raharja

Apa Itu OOP?

Pemrograman Berorientasi Objek (*Object-Oriented Programming* atau OOP) adalah paradigma pemrograman yang berfokus pada objek sebagai unit dasar penyusun program. Objek merepresentasikan entitas dunia nyata yang memiliki atribut (*data*/*state*) dan perilaku (*method*/*functionality*). Contoh analogi:

Bayangkan sebuah "Mobil":

1. Atribut: warna, merek, kecepatan.
2. Perilaku: melaju(), berhenti(), mundur().

OOP menekankan pada pemodelan objek dan interaksi antar objek, bukan hanya pada urutan instruksi seperti dalam pemrograman prosedural.

Definisi OOP Menurut Para Ahli & Sumber Akademik

Menurut TechTarget (Gilllis & Lewis, 2020–2023), *"Objectoriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic."* Objek memiliki atribut dan *behavior*, dan komunikasi antar objek dilakukan lewat pesan (*method calls*). Menurut Wikipedia, *"Data abstraction is a way … only certain parts of the data are visible… encapsulation refers to … bundling of data with the methods … interacting via specific public methods."* OOP memperlakukan objek sebagai perisai antara *detail* internal dan kode luar, meningkatkan modularitas dan pengelolaan kode.

Studi empiris oleh AlOmar et al. (2021) menunjukkan bahwa *refactoring* dalam proyek OOP dapat secara signifikan meningkatkan *reusability* komponen perangkat lunak. Menurut Wang et al. (2024), penerapan prinsip OOP sangat penting dalam membangun sistem AI dan ML yang modular, mudah dikembangkan, dan dapat diskalakan.

Menurut pandangan sejumlah pengembang (2024), *"Strictly speaking, the only real requirement for* OOP *is that data and behaviours are encapsulated together into objects."* Gardiner menjelaskan bahwa *ad-hoc polymorphism* dan *dynamic dispatch* adalah ciri khas modern OOP.

dengan memecah sistem menjadi bagian-bagian kecil yang disebut objek. Pendekatan ini memberikan banyak manfaat yang sangat dibutuhkan dalam pengembangan sistem skala kecil hingga besar.

1. **Modularitas**
   Setiap objek dapat dikembangkan secara independen dan kemudian digabungkan ke dalam sistem besar. *"Modularity is a key advantage of OOP as it allows compartmentalized development and facilitates collaborative programming"* (Ali, S., *et al.,* 2021).
2. **Reusabilitas**
   Kode yang ditulis dalam bentuk kelas dapat digunakan kembali dalam program lain menggunakan pewarisan. *Object-oriented techniques promote code reuse through inheritance, reducing development time and effort."* (Kumar, A., & Choudhary, R., 2020).
3. **Skalabilitas dan Pemeliharaan Mudah**
   OOP membuat sistem lebih mudah dikembangkan dan diperluas tanpa merusak komponen yang ada.
4. **Enkapsulasi dan Keamanan Data**
   Enkapsulasi melindungi data dengan menyembunyikannya di balik antarmuka publik, meminimalkan risiko perubahan tidak sah.
5. **Abstraksi**
   Abstraksi memungkinkan pengembang untuk fokus pada konsep penting dan mengabaikan *detail* yang tidak relevan.

Daftar Pustaka

Ali, S., et al. (2021). Object-Oriented Programming in Modern Software Development: A Case for Scalability and Maintainability. *Journal of Software Engineering & Applications,* 14(6), 235-248.

AlOmar, E. A., Wang, T., Raut, V., Mkaouer, M. W., Newman, C., & Ouni, A. (2021). *Refactoring for Reuse: An Empirical Study*. arXiv. https://doi.org/10.48550/arXiv.2111.07002.

ArXiv. (2023). *Refactoring Object-Oriented Systems for AI Integration*. Retrieved from https://arxiv.org.

Developer Community Forum. (2024, March 18). *The Essence of OOP: Encapsulation And Dynamic Binding*. StackOverflow Discussion Thread. Retrieved June 30, 2025, from https://stackoverflow.com/questions/essence-of-oop.

Exforsys. (2022). *History of Object Oriented Programming*. Retrieved from https://www.exforsys.com.

Horstmann, C. S. (2019). *Core Java Volume I Fundamentals (11th ed.).* Pearson. Buku Ini Menjelaskan Manfaat OOP Dalam Pengembangan Sistem Besar Berbasis *Java*.

HyperSense Software. (2021). *A brief History of Java And its Role In Enterprise Software.* Retrieved from https://hypersense-software.com

Kumar, A., & Choudhary, R. (2020). A Comparative Study of Object-Oriented and Procedural Programming Paradigms. *International Journal of Computer Applications*, 176(6), 22-27.

Llamas, L. (2023). The Evolution of OOP: From Simula to Kotlin. Medium. Retrieved from https://medium.com/@llamas/evolution-of-oop.

Medium. (2023). *The Evolution of Object-Oriented Programming*. Retrieved from https://medium.com.

Minto Share. (2023). *Why Python is One of The Best OOP Languages*. Retrieved from https://mintoshare.com.

TechTarget. (2024). *Object-oriented programming (OOP)*. Retrieved June 30, 2025, from https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP.

The Dev Workshop. (2022). *The Role of Objective-C in the Apple Ecosystem.* Retrieved from https://thedevworkshop.com.

Wang, T., Bi, Z., Chen, K., Xu, J., Niu, Q., Liu, M., Zhang, S., Pan, X., Wang, J., Feng, P., Wen, Y., Liu, M., & Peng, B. (2024). *Deep Learning and Machine Learning, Advancing Big Data Analytics and Management: ObjectOriented Programming*. arXiv. https://doi.org/10.48550/arXiv.2409.19916.

Wikipedia contributors. (2024, April 1). *Object-Oriented Programming*. Wikipedia. https://en.wikipedia.org/wiki/Object-oriented_programming.

Wikipedia. (2024). *History of Object-Oriented Programming*. Retrieved from https://en.wikipedia.org/wiki/Object-oriented_programming

WIRED. (2020). *How C++ Changed The World*. Retrieved from https://www.wired.com.

PROFIL PENULIS

Eka Purnama Harahap, S.Kom., M.TI. Kelahiran Baringin Jaya, 18 September 1993. Ia merupakan Sarjana lulusan STMIK Raharja pada tahun 2016. Dua tahun kemudian lulus sebagai Magister Teknologi Informasi di STMIK Raharja yang sejak 2019 telah berubah menjadi Universitas Raharja. Saat ini sedang menempuh pendidikan Doktoral Ilmu Komputer di Universitas Kristen Satya Wacana. Sejak 2016 sampai hingga saat ini, ia telah memiliki 10+ Mata kuliah yang diampu, dimana 2 mata kuliah diantaranya yaitu Pemrograman Berbasis Objek dan Pengantar Kecerdasan Buatan. Selama 7 tahun mengabdi, banyak pengalaman yang didapatkan sehingga mengajarkannya lebih mengenal beraneka ragam karakter. Pada Tri Dharma Penelitian, ia aktif menulis karya ilmiah yang terpublikasi di kancah nasional maupun internasional. Diantaranya terdapat 80+ Jurnal Nasional Terakreditasi yang *terindex Google Scholar* dan SINTA serta 18+ Jurnal Internasional *Terindex SCOPUS*. Ia juga merupakan bagian dari *Web of Science Academy*, yakni sebagai *graduate* dan juga mentor. Selain penelitian, ia memiliki 4 hak kekayaan intelektual di bidang teknologi informasi. Bidang minatnya meliputi; *Machine Learning, Artificial Intelligence, Smart Tourism*, dan *Data Analytics*. Motto**:** Al Jazaa Min Jinsil 'Amal.

Email Penulis: ekapurnamaharahap@raharja.info.

BAB 10
INHERITANCE DAN *POLYMORPHISM*

Mohammad Badrul, S.Kom., M.Kom.
Universitas Bina Sarana Informatika

Inheritance/Pewarisan

Pewarisan atau *inheritance* merupakan salah satu karakteristik dari bahasa pemrograman berbasis objek atau *object oriented programming*. Bahasa pemrograman *java* merupakan salah satu bahasa pemrograman yang mengimplementasikan *object oriented programming* (Deitel & Deitel, 2017). *Inheritance* adalah proses pembuatan *class* baru yang akan mewarisi karakter dari *class* yang telah ada.

Pewarisan memungkinkan sebuah kelas yang disebut sebagai *subclass* atau *child class* untuk mewarisi sifat dan perilaku (*attribute* dan *method*) dari kelas lain yang disebut sebagai *superclass* atau *parent class* sehingga akan membentuk hirarki *class*. Dalam hirarki *class*, jika kelas C adalah turunan kelas B, sedangkan kelas B merupakan turunan kelas A, maka secara otomatis *attribute* dan method kelas A juga diwariskan kelas C.

Setiap *subclass* akan mewarisi *state* (*variabel-variabel*) dan *behaviour* (*method-method*) dari *superclass*-nya (Schildt, 2018). Pewarisan merupakan salah satu fitur yang sangat penting di bahasa pemrograman *java* karena dengan fitur ini pengembang dapat memanfaatkan kembali *source code* yang sudah ada, memperluas fungsionalitas, serta mengorganisir *source code* dengan lebih baik (Hidaya, 2024). Contoh *inheritance* di sekitar kita: contoh implementasi dari pewarisan atau *inheritance* di kehidupan kita sehari-hari seperti konsep "mobil", semua mobil memiliki karakteristik atau atribut seperti *merk*, jumlah pintu, warna dan jenis atau pabrikan. Dalam dunia nyata ada banyak jenis atau pabrikan mobil yang ada disekitar kita seperti pabrikan Honda, Toyota, Hyundai, Nissan, Ford, Wuling, BWM, Ferrari dan jenis pabrikan mobil lainnya.

Jenis atau pabrikan ini memiliki sifat dasar yang sama dari "mobil" namun mungkin memiliki *attribute* atau *method* tambahan. Pada konsep *object oriented programming*, *programmer* dapat membuat kelas *parent* atau *super* yang bernama mobil, selanjutnya membuat *class* Honda, Toyota yang dijadikan *subclass* atau *class* turunan yang mewarisi atribut dan *method* dari *superclass* (Sianipar, 2019).

c. *Class* persegiPanjang dan segiTiga

```java
package polymorphism;

/**
 *
 * @author Mohammad Badrul
 */
//class turunan/subclass
public class persegiPanjang extends bangunDatar{
    //create attribut
    int panjang, lebar;

    //create method konstuktor
    public persegiPanjang(int panjang, int lebar) {
        this.panjang = panjang;
        this.lebar = lebar;
    }

    //create method Override
    @Override
    public float luas(){
        return this.panjang * this.lebar;
    }

    //create method Override
    @Override
    public float keliling(){
        return 2 * (this.panjang + this.lebar);
    }

}
```

```java
package polymorphism;

/**
 *
 * @author Mohammad Badrul
 */
//class turunan/subclass
public class segiTiga extends bangunDatar{
    //create attribut
    int alas,tinggi;

    //create method konstruktor
    public segiTiga(int alas, int tinggi) {
        this.alas = alas;
        this.tinggi = tinggi;
    }

    //create method override
    @Override
    public float luas(){
        return (this.alas * this.tinggi)/2;
    }

    //create method override
    @Override
    public float keliling(){
        return this.alas + this.tinggi + this.tinggi;
    }

}
```

Gambar 10.18: *Class* persegiPanjang dan segiTiga
Sumber: Diolah Penulis.

Dua *class* di atas yaitu *class* persegiPanjang dan segiTiga merupakan *class* turunan yang menjadi *class* turunan dari *class* bangun datar. Di dalam *class* turunan ini ada dua *method* yang fungsinya untuk meng-*override method* yang ada di *class* induk sesuai dengan jenis bangun datarnya.

d. *Main Class*

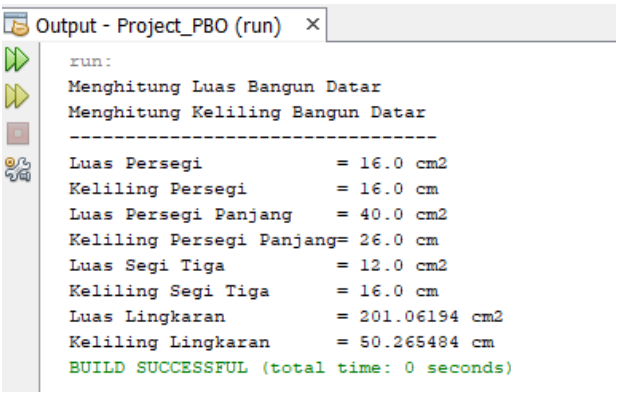
```java
package polymorphism;

/**
 *
 * @author Mohammad Badrul
 */
public class bangunDatarBeraksi {
    public static void main(String[] args) {
        //create object dan instansiasi berikut isi parameternya
        bangunDatar bd = new bangunDatar();
        persegi p = new persegi(4);
        persegiPanjang pp = new persegiPanjang(8, 5);
        segiTiga st = new segiTiga(4, 6);
        lingkaran l = new lingkaran(8);

        //panggil method dan luas Bangun Datar
        bd.luas();
        bd.keliling();
        System.out.println("---------------------------------");
        System.out.println("Luas Persegi             = " + p.luas() + " cm2");
        System.out.println("Keliling Persegi         = " + p.keliling()+ " cm");
        System.out.println("Luas Persegi Panjang     = " + pp.luas() + " cm2");
        System.out.println("Keliling Persegi Panjang= " + pp.keliling()+ " cm");
        System.out.println("Luas Segi Tiga           = " + st.luas()+ " cm2");
        System.out.println("Keliling Segi Tiga       = " + st.keliling() + " cm");
        System.out.println("Luas Lingkaran           = " + l.luas() + " cm2");
        System.out.println("Keliling Lingkaran       = " + l.keliling() + " cm");
    }
}
```

Gambar 10.19: *Main Class*
Sumber: Diolah Penulis.

e. Hasil



Gambar 10.20: Hasil Setelah di *Running*
Sumber: Diolah Penulis.

Daftar Pustaka

Bachtiar, A. M. (2018). *Pemrograman Berorientasi Objek Menggunakan Java.* Informatika.

Cosmina, I. (2018). *Java for Absolute Beginners: Learn to Program the Fundamentals the Java 9+ Way*. Apress.

Deitel, P. J., & Deitel, H. (2017). *Java How to Program, Early Objects (11th Edition).* PE.

Farrell, J. (2018). *Java Programming 9th Edition*. Cengage Learning.

Hidaya, B. (2024). *Pemrograman Java Dasar*. Edu Publisher.

Schildt, H. (2018). *Java: A Beginner's Guide, Eighth Edition 8th Edition*. McGraw Hill.

Sianipar, R. H. (2013). *Teori dan Implementasi Java*. Informatika.

Sianipar, R. H. (2014). *Soal, Kasus dan Penyelesaian Pemrograman Java: Belajar, Memahami, dan Eksplorasi*. Informatika.

Sianipar, R. H. (2019). *Java: Referensi Lengkap Untuk Programer*. Andi Publisher.

PROFIL PENULIS

Mohammad Badrul, M.Kom.

Penulis lahir dari keluarga petani sederhana di daerah Bangkalan tahun 1984. Dari kecil penulis sudah di didik di lingkungan agamis dan sederhana. Setelah lulus dari Sekolah Menengah Pertama dan Sekolah Menengah Atas di lingkungan Pesantren, Sejak di Sekolah Menengah Atas, Penulis memiliki ketertarikan di bidang komputer. Oleh sebab itu penulis melanjutkan program Diploma III atau D3 di Akademi Manajemen Informatika dan Komputer (AMIK) Bina Sarana Informatika di Jakarta dengan Jurusan Komputerisasi Akuntansi. Lulus dari Program Diploma III tahun 2007, Penulis melanjutkan Program Strata Satu (S1) di Sekolah Tinggi Manajemen Informatika dan Komputer (STMIK) Nusa Mandiri di Jakarta Jurusan Sistem Informasi. Setelah lulus program Strata Satu (S1) tahun 2009, Penulis melanjutkan program Strata Dua (S2) di Sekolah Tinggi Manajemen Informatika dan Komputer (STMIK) Nusa Mandiri Jakarta jurusan Ilmu Komputer tahun 2010 dan lulus tahun 2012. Saat ini penulis aktif mengajar di beberapa kampus di Jakarta khususnya Jurusan Sistem Informasi, Informatika dan Teknologi Informasi dengan konsentrasi di bidang *Data Mining, Data Science*, dan Sistem Penunjang Keputusan. Dan untuk mewujudkan karir sebagai dosen profesional, penulis pun aktif sebagai peneliti di bidang kepakarannya tersebut. Beberapa penelitian yang telah dilakukan didanai oleh internal perguruan tinggi dan juga Kemenristek DIKTI.

Email Penulis: mohammad.mbl@bsi.ac.id.

BAB 11
ENCAPSULATION & ACCESS MODIFIER

Alfredo Gormantara, S.Kom., M.Kom.
Universitas Atma Jaya Makassar

Pengertian Enkapsulasi

Encapsulation atau enkapsulasi merupakan salah satu prinsip inti dalam pemrograman berorientasi objek (OOP), terutama dalam bahasa *java*. Prinsip ini melibatkan penggabungan data (atribut) dan perilaku (metode) dalam satu unit yang disebut *kelas*, dengan tujuan menyembunyikan *detail* implementasi dan hanya mengekspos antarmuka publik yang didesain secara ketat. Atribut kelas biasanya dideklarasikan sebagai *private*, sementara akses ke nilai dilakukan melalui metode *getter* dan *setter* publik, yang memungkinkan kontrol akses dan validasi data sebelum modifikasi dilakukan (Suharto 2022).

Menurut studi perbandingan *java* dan *python* (Omer et al. 2023), enkapsulasi di *Java* menghasilkan pengamanan data yang lebih ketat karena penggunaan *modifier* seperti *private, protected*, dan *public*, dibandingkan dengan *python* yang mengandalkan konvensi penamaan seperti _ dan __ untuk menandai atribut privat. Bentuk enkapsulasi ini tidak hanya menjaga integritas objek tetapi juga mempermudah pemeliharaan kode artinya logika internal dapat diubah tanpa mempengaruhi kode eksternal yang bergantung pada antarmuka tetap.

Dengan demikian, enkapsulasi di *java* adalah teknik penting untuk melindungi data, mengontrol akses, dan mendukung pengembangan perangkat lunak yang lebih *maintainable* dan aman.

Pengertian *Access Modifier* (Pengubah Akses)

Access modifier adalah *keyword* dalam bahasa *java* yang digunakan untuk mengatur tingkat aksesibilitas terhadap elemen-elemen dalam program seperti kelas, atribut, *method*, dan konstruktor. Dengan *access modifier, programmer* dapat menentukan sejauh mana anggota kelas tersebut dapat diakses oleh kelas lain, baik yang berada dalam package yang sama maupun di luar *package*. Hal ini memberikan kontrol terhadap interaksi antar bagian program dan membantu menjaga struktur kode tetap terorganisir (Nita 2021).

Tujuan utama penggunaan *access modifier* adalah untuk meningkatkan keamanan dan keteraturan kode, terutama dalam konteks pengembangan perangkat lunak berskala besar. Dengan membatasi akses ke elemen-elemen penting, *programmer* dapat

Dengan demikian, konsep-konsep ini mendasari struktur aplikasi berskala besar agar tetap efisien, aman, dan mudah dikelola baik oleh tim pengembang maupun pengguna akhir.

Kelebihan dan Kekurangan Enkapsulasi

Kelebihan Enkapsulasi:

1. Perlindungan Data (Data Hiding)
   Dengan menjadikan atribut kelas bersifat *private*, data tidak dapat diakses langsung dari luar kelas, sehingga mencegah manipulasi yang tidak diinginkan.
2. Kontrol Penuh Terhadap Akses Data
   Programmer dapat mengontrol bagaimana data dibaca dan diubah melalui *method getter* dan *setter*, yang memungkinkan penerapan validasi sebelum perubahan data dilakukan.
3. Pemeliharaan Kode yang Lebih Mudah
   Perubahan logika atau aturan hanya perlu dilakukan di satu tempat (misalnya dalam *method setter*), sehingga tidak perlu memodifikasi semua bagian program yang mengakses atribut tersebut.
4. Mendukung Prinsip Abstraksi
   Enkapsulasi membantu menyembunyikan kompleksitas internal dan hanya menampilkan antarmuka yang dibutuhkan, memudahkan pengguna kelas untuk fokus pada fungsionalitas utama.

Kekurangan dan Batasan:

1. Kode Tambahan untuk Akses Data
   Enkapsulasi memerlukan pembuatan *method getter* dan *setter* untuk setiap atribut yang perlu diakses, yang dapat menambah panjang kode dan menyebabkan *boilerplate*.
2. Kemungkinan Penyalahgunaan *Setter*
   Jika *setter* tidak dibatasi dengan validasi yang ketat, data masih bisa dimodifikasi dengan nilai yang tidak sesuai, meskipun akses dilakukan secara "resmi".
3. Tingkat Kompleksitas Awal bagi Pemula
   Bagi *programmer* pemula, penggunaan enkapsulasi dapat terasa rumit karena memerlukan pemahaman tambahan mengenai *access modifier* dan struktur kelas.

4. Kinerja Sedikit Lebih Lambat
   Karena semua akses ke atribut harus melalui *method*, terdapat sedikit *overhead* dibandingkan dengan akses langsung, meskipun pada praktiknya ini jarang signifikan.

Kesimpulan

Enkapsulasi merupakan salah satu prinsip fundamental dalam pemrograman berorientasi objek (OOP) yang bertujuan untuk membungkus data dan perilaku dalam satu unit, serta menyembunyikan detail internal dari akses luar. Dengan menerapkan enkapsulasi, pengembang dapat menjaga keamanan dan konsistensi data dengan membatasi akses langsung ke atribut, serta menyediakan method publik untuk mengatur interaksi terhadap data tersebut. Hal ini tidak hanya meningkatkan keamanan program, tetapi juga membuat kode lebih mudah dirawat dan dimodifikasi karena logika internal tidak tersebar ke seluruh sistem.

Access modifier adalah alat teknis dalam *java* yang digunakan untuk mengontrol visibilitas kelas, atribut, dan *method. Modifier* seperti *private, protected,* dan *public* memungkinkan pengembang untuk mengatur siapa saja yang boleh mengakses atau memodifikasi bagian tertentu dari kode.

Access modifier berperan penting dalam mendukung penerapan enkapsulasi, karena memungkinkan pemrogram membatasi atau membuka akses sesuai kebutuhan desain. Dengan memahami dan menggunakan *access modifier* dengan tepat, kita dapat membangun sistem yang lebih modular, aman, dan terstruktur dengan baik.

Daftar Pustaka

Faisal, Mohammad Reza, Universitas Lambung Mangkurat, and Erick Kurniawan. (2023). *Seri Belajar Pemrograman: Pemrograman Dasar Dengan Java.* (March).

Nita, Sekreningsih. (2021). *Pemrograman Berorientasi Objek With 'Java' (Teori & Implementasi Java).* https://eprint.unipma.ac.id/237/1/115. Repository Bu Sekreningtyas.pdf.

Omer, Skala Kamaran, Star Dawood Mirkhan, Nyan Najat Hussein, Aveen Zuber Ali, Tarik Ahmed Rashid, and Poornima Nedunchezhian. (2023). Comparative Analysis of Encapsulation in *Java* And Python: Syntax And Implementation Differences. *The Journal of Duhok University* 26(2):379–89. doi:10.26682/csjuod.2023.26.2.35.

Suharto, Ahmad. 2022. *Teori Dan Praktik Object Oriented Programming Menggunakan Java*.

PROFIL PENULIS

Alfredo Gormantara, S.Kom., M.Kom. adalah dosen tetap di Universitas Atma Jaya Makassar dengan minat riset di bidang *Information Systems, Machine Learning*, dan *Parallel Computing*. Ia menyelesaikan pendidikan sarjana di bidang Teknik Informatika dari Universitas Atma Jaya Makassar dan melanjutkan studi magister di bidang yang sama di Universitas Atma Jaya Yogyakarta. Kepakarannya mencakup pengembangan perangkat keras dan komputasi cerdas, yang sejalan dengan pokok bahasan dalam buku ini. Sebagai dosen profesional dan peneliti aktif, ia telah menghasilkan berbagai penelitian di bidang kepakarannya, baik yang didanai secara internal oleh perguruan tinggi maupun eksternal oleh Kemenristek DIKTI. Penulis juga aktif dalam kegiatan pengabdian kepada masyarakat melalui penerapan teknologi berbasis riset. Buku ini disusun berdasarkan pengalaman akademik dan profesional, serta diharapkan dapat menjadi kontribusi nyata dalam pengembangan ilmu pengetahuan di bidang informatika. Penulis terbuka untuk berkolaborasi dalam penelitian dan pengabdian, dan dapat dihubungi melalui email: alfredo_gormantara@lecturer.uajm.ac.id.

BAB 12
INPUT/OUTPUT (FILE HANDLING)

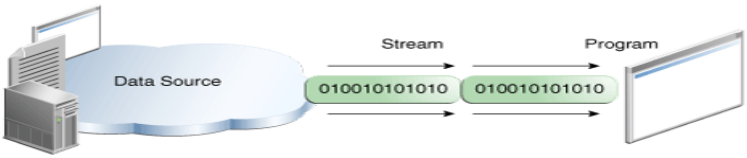
Fahmi Fadillah Septiana, S.Kom., M.Kom.
Universitas Garut

Pendahuluan

Dalam *java*, paket *java.io* mengelola operasi I/O. Paket ini menawarkan kelas-kelas untuk menulis dan membaca data dari berbagai sumber, seperti memori, *file*, atau bahkan jaringan. Ide pokoknya adalah aliran. *Stream* merupakan aliran informasi yang bisa berupa *byte* (*byte stream*) atau karakter (*character stream*) (*Team,* 2024).

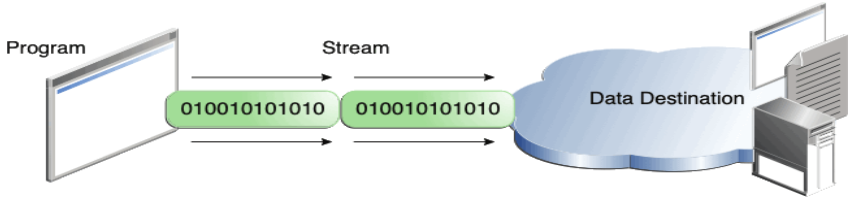
Stream I/O adalah sarana untuk memasukkan data atau tujuan untuk mengeluarkan data. *Stream* dapat mewakili berbagai jenis sumber dan target, termasuk berkas *disk*, perangkat, aplikasi lain, serta susunan memori. *Stream* memungkinkan berbagai jenis data, termasuk *byte* umum, tipe data dasar, karakter lokal, dan objek. Beberapa aliran hanya menyalurkan informasi; yang lain mengolah dan mengubah data dengan cara yang berguna.

Apa pun mekanisme internalnya, semua *stream* menyajikan model sederhana yang sama bagi program yang memanfaatkannya: *Stream* merupakan rangkaian data. Sebuah aplikasi memanfaatkan aliran *input* untuk mengambil data dari suatu sumber, secara berurutan:



Gambar 12.1: Membaca Informasi Ke Dalam Program
Sumber: *Java*, 2024.

Sebuah sistem menggunakan alir keluaran untuk menerapkan data ke suatu perangkat, satu per satu:



Gambar 12.2: Menulis Informasi Dari Sebuah Program
Sumber: *Java*, 2024.

```
        namaBelakang = breader.readLine();
        System.out.println("Nama Saya adalah :");
            System.out.println(namaDepan + " " + namaBelakang);
    }
```

3. *Class JoptionPane*

JoptionPane adalah sebuah *class* yang menyediakan sarana menerima dan menampilkan data berbasis grafis. Model tampilan akan mengikuti tampilan yang disediakan oleh sistem operasi. *JoptionPane* terdapat di dalam *class javax.swing*.

Pentingnya *Try-With-Resources*

Saat bekerja dengan I/O dan berkas, sangat penting untuk selalu menutup *stream* setelah selesai menggunakannya. Jika tidak ditutup, sumber daya sistem bisa bocor, *file* bisa rusak, atau data tidak tersimpan sepenuhnya. Sejak *java* 7, ada konstruksi *try-with-resources* yang sangat direkomendasikan. Ini secara otomatis menutup semua sumber daya yang dapat ditutup (implementasi *AutoCloseable*) setelah blok *try* selesai, bahkan jika terjadi *exception*. Contoh tanpa *try-with-resources* (kurang disarankan)(Horstmann 2013):

```
FileWriter writer = null;
try {
   writer = new FileWriter("myfile.txt");
   writer.write("Hello");
} catch (IOException e) {
   e.printStackTrace();
} finally {
   if (writer != null) {
      try {
         writer.close(); // Harus ditutup di sini
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Contoh dengan *try-with-resources* (sangat direkomendasikan):

```
import java.io.FileWriter;
import java.io.IOException;

public class TryWithResourcesExample {
    public static void main(String[] args) {
        // FileWriter otomatis ditutup setelah blok try selesai
        try (FileWriter writer = new FileWriter("myfile.txt")) {
            writer.write("Hello from try-with-resources!");
            System.out.println("Data written successfully.");
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }
    }
}
```

Sejak *Java* 7, ada API I/O baru yang lebih modern dan efisien, yaitu NIO.2 *(New Input/Output 2*), yang berada di paket *java.nio.file.* API ini menawarkan (Magazine 2003):

1. *Path*: representasi jalur file yang lebih fleksibel dan kuat daripada *file*.
2. *Files*: kelas utilitas dengan metode statis untuk melakukan operasi *file* umum (menyalin, memindahkan, menghapus, membaca semua baris, menulis semua *byte*, dll.) dengan lebih mudah dan aman.
3. *DirectoryStream*: untuk melakukan iterasi pada konten direktori secara efisien.

Pemahaman mendalam tentang konsep *stream*, berbagai kelas I/O, dan pentingnya penanganan *exception* serta penutupan sumber daya adalah kunci untuk menulis program *java* yang kuat dan efisien dalam berinteraksi dengan data dan sistem berkas.

Daftar Pustaka

Bloch, J. (2018). *Effective Java 3rd Edition. Best Practices for Java Platform*.

Evi Lestari Pratiwi. (2020). *Konsep Dasar Algoritma Dan Pemrograman Dengan Bahasa Java*. *Edited by Adi Pratomo*. Banjarmasin Utara: Poliban Press.

Horstmann, Cay S. (2013). *Core Java, Volume I: Fundamentals, 12th Edition.*

Java. (2024). *Java Tutorial.* Retrieved (https://docs.oracle.com/*java*se/tutorial/essential/io/streams.html).

Magazine, Software Development. (2003). *A Brain-Friendly Guide - Head First JAVA 2nd Edition Covers Java 5.0*.

Team, *Java*. (2024). *The Java™ Tutorials*. Retrieved (https://docs.oracle.com/*java*se/tutorial/essential/io/index.html).

PROFIL PENULIS

Fahmi Fadillah Septiana, S.Kom., M.Kom.

Ketertarikan penulis terhadap Pemrograman dan juga Sistem komputer dimulai pada tahun 2013 silam. Hal tersebut membuat penulis memilih untuk masuk ke Sekolah Menengah Kejuruan di SMK Negeri 9 Garut dengan memilih Jurusan Teknik Multimedia (MM) dan berhasil lulus pada tahun 2016. Penulis kemudian melanjutkan pendidikan ke Perguruan Tinggi dan berhasil menyelesaikan studi S1 di prodi Teknik Informatika Institut Teknologi Garut pada tahun 2020. Dua tahun kemudian, penulis menyelesaikan studi S2 di prodi Ilmu Komputer Program Pasca Sarjana Universitas Nusa Mandiri Jakarta. Penulis memiliki kepakaran di bidang *Web Programming* dan *Internet of Things Engineering* (IoT *Engineer*). Dan untuk mewujudkan karir sebagai dosen profesional, penulis pun aktif sebagai Trainer di berbagai sekolah nasional ataupun internasional dan juga menjadi peneliti di bidang kepakarannya tersebut. Beberapa penelitian yang telah dilakukan didanai oleh internal perguruan tinggi dan juga Kemenristek DIKTI. Selain peneliti, penulis juga aktif menulis buku dengan harapan dapat memberikan kontribusi positif bagi bangsa dan negara yang sangat tercinta ini. Atas dedikasi dan kerja keras dalam menulis, Penulis berhasil menerbitkan beberapa jurnal terakreditasi baik ter-*index* sinta ataupun nasional.

Email Penulis: fahmifadillahs@uniga.ac.id.

BAB 13
UNIT TESTING DENGAN *JUNIT*

Yosep Bustomi, S.T., M.Kom.
Universitas Garut

Apa Itu Unit *Testing*?

1. Apa Itu *Unit Testing*?

Terdapat banyak definisi terkait *unit testing*, *unit testing* adalah *testing* otomatis untuk melakukan verifikasi sebagian *code* atau unit, dilakukan dengan cepat dan terisolasi (Khorikov, 2020).

2. Apa Itu *JUnit* 5?

JUnit 5 dirancang dengan mengadopsi arsitektur modular yang terdiri dari tiga komponen utama dan masing-masing memiliki perannya (Bechtold dkk., 2025).

JUnit 5 = *JUnit Platform* + *JUnit Jupiter* + *JUnit Vintage.*

a. *JUnit Platform*

Adalah suatu pondasi atau landasan dari seluruh ekosistem *JUnit* 5. Ini adalah lapisan dasar tempat *test engine* (mesin pengujian) berjalan. *Platform* ini menyediakan API untuk menemukan dan menjalankan tes.

1) Tugasnya memberikan *interface* (antarmuka) bagi IDE, *build tools*, *plugin* CI/CD untuk berinteraksi dengan *JUnit*. Ini memungkinkan alat-alat tersebut menemukan, menjalankan dan melaporkan hasil *test* secara standar, tanpa perlu tahu detail internal dari bagaimana tes itu dilakukan.
2) Analogi mudah *JUnit Platform* sebagai terminal bandara. Ia tidak tahu bagaimana pesawat (*test*) itu dibuat atau jenisnya apa, tetapi ia menyediakan landasan pacu, gerbang dan menara kontrol agar semua jenis pesawat bisa mendarat, take off dan diatur pergerakannya dengan benar.

b. *JUnit Jupiter*

Adalah model pemrograman dan *extension model* untuk menulis tes dan *extensions* di *JUnit* 5. Ini bagian yang paling sering berinteraksi dengan pengembang saat menulis unit *test* baru.

1) Tugas adalah menyediakan semua anotasi dan API yang kita gunakan sehari-hari untuk menulis tes di *JUnit* 5. Ini termasuk *@Test, @BeforeEach, @AfterEach*, *@ParameterizedTest*, dan semua *assertion* yang digunakan (*assertEquals, assertThrows*, dll).

```
@Test
@Tag("UI") // Tes ini terkait UI
 void testHomePageLayout() {
    assertTrue(true); // Asumsi tes UI
}
```

4. *Dynamic Tests*

Dynamic Test merupakan fitur yang memungkinkan kita membuat dan mendaftarkan tes secara dinamis saat *runtime,* bukan saat kompilasi. Ini berbeda dengan *Parameterized Tests* yang menyediakan data inputan untuk tes yang sudah didefinisikan secara statis sebelumnya. Anotasi dari *dynamic tests* ini yaitu *@TestFactory.* Contoh Penggunaan *Dynamic Tests*:

```
@TestFactory
Collection<DynamicTest> dynamicTestsWithCollection() {

 // Ini akan menghasilkan 3 tes dinamis
 return Arrays.asList(
  dynamicTest("Tes Pertama", () -> assertTrue(true)),
  dynamicTest("Tes Kedua", () -> {
    /* lakukan sesuatu */
     assertTrue(true);
  }),
  dynamicTest("Tes Ketiga", () -> assertTrue(true))
 );

}
```

Kesimpulan

Unit testing adalah fondasi penting dalam pengembangan perangkat lunak modern saat ini, sehingga hasil kode yang kita buat sudah sesuai dan terhindar dari kesalahan. *JUnit* adalah *framework* tercepat saat ini untuk ekosistem *java* dalam melakukan *Unit Testing.* Terdapat arsitektur modular di dalam *JUnit* 5 yaitu *JUnit Platform, JUnit Jupiter* dan *JUnit Vintage* sehingga memberikan fleksibilitas yang bisa diandalkan.

Berbagai anotasi seperti *@Test, @BeforeEach* dan *@AfterEach,* yang membantu siklus hidup tes, juga memastikan setiap pengujian dimulai dengan kondisi bersih, selain itu ada *assertion* seperti *assertEquals, assertTrue* dan *assertThrows* yang digunakan untuk verifikasi hasil dari kode sesuai dengan yang kita harapkan.

Selain dari fitur dasar terdapat juga fitur lanjutan seperti *Parameterized Test, Nested Tests, Tagging* dan *Dynamic Test* sehingga kita bisa menyesuaikan dengan kompleksitas objek yang akan dilakukan tes. *Unit Testing* dengan *JUnit* bukan hanya tentang menemukan *bugs* tetapi membawa kita mendapatkan praktik terbaik dalam menulis kode yang bersih, modular dan mudah dipelihara.

Daftar Pustaka

Bechtold, S., Brannen, S., Link, J., Merdes, M., Philipp, M., de Rancourt, J., & Stein, C. (2025, Agustus 2). *JUnit 5 User Guide*. https://junit.org/junit5/docs/current/user-guide/. https://docs.junit.org/current/user-guide/.

Junit-team. (2025a). *build.gradle from junit-jupiter-starter-gradle*. https://github.com/junit-team/junit-examples/blob/r5.13.4/junit-jupiter-starter-gradle/build.gradle.

Junit-team. (2025b, Agustus 2). *pom.xml from junit-jupiter-starter-maven*. https://github.com/junit-team/junit-examples/blob/r5.13.4/junit-jupiter-starter-maven/pom.xml.

Khorikov, Vladimir. (2020). *Unit Testing Principles, Practices, and Patterns*.

PROFIL PENULIS

Yosep Bustomi, S.T., M.Kom.

Seorang praktisi berpengalaman > 15 tahun bergelut di bidang pengembangan perangkat lunak. Selain menjadi praktisi saat ini penulis aktif juga sebagai akademisi menjadi dosen tetap di Program Studi Rekayasa Sistem Komputer, Universitas Garut (UNIGA). Berbagai proyek perangkat lunak yang sudah dikerjakan, baik untuk Perusahaan BUMN dan Perusahaan Swasta yang berfokus di bidang keuangan dan perbankan.

Email Penulis: yosep@uniga.ac.id.

PEMROGRAMAN Java

Pemrograman Java telah menjadi salah satu bahasa pemrograman paling populer dan relevan di dunia. Fleksibilitas, portabilitas, dan performanya yang tinggi menjadikan Java pilihan utama untuk pengembangan aplikasi lintas platform, mulai dari aplikasi desktop, web, hingga perangkat seluler berbasis Android. Oleh karena itu, menguasai Java adalah langkah strategis bagi Anda yang ingin berkarier di bidang teknologi informasi. Dalam buku ini, kami menyajikan materi secara terstruktur, dimulai dari:

1. Pengenalan Pemrograman Java
2. Persiapan Lingkungan Pengembangan Java
3. Struktur Dasar Program Java
4. Tipe Data dan Variabel dalam Java
5. Operator dalam Java
6. Input dan Output (I/O) Dasar
7. Kontrol Alur Program dalam Java
8. Metode (*Function*) dalam Java
9. Pemrograman Berorientasi Objek (OOP) Dasar
10. *Inheritance* dan *Polymorphism*
11. *Encapsulation* & *Access Modifier*
12. Input/Output (*File Handling*)
13. Unit Testing dengan JUnit

sada kurnia pustaka

ISBN 978-634-7021-69-4

9 786347 021694