

BAB I

INTRODUCTION

Sebelum memulai perkuliahan tentang pemrograman komputer, pada bab ini akan disajikan beberapa informasi latar belakang tentang komputer dan bahasa pemrograman. Pada bab ini akan dijelaskan sejarah singkat awal mula komputer hingga komponen-komponen yang terdapat pada komputer masa kini. Selain itu, pada bab ini juga dijelaskan sejarah singkat bahasa pemrograman dari bahasa mesin tingkat rendah hingga bahasa berorientasi obyek seperti saat ini.

1.1 Sejarah Komputer

Manusia telah berevolusi dari masyarakat primitif menjadi masyarakat yang sangat maju dengan terus-menerus menemukan alat. Perkakas batu, bubuk mesiu, roda, dan penemuan lainnya telah mengubah kehidupan manusia secara dramatis. Dalam sejarah belakangan ini, komputer bisa dibilang merupakan penemuan paling penting. Dalam masyarakat yang sangat maju saat ini, komputer memengaruhi kehidupan kita 24 jam sehari: Jadwal kelas dirumuskan oleh komputer, catatan siswa disimpan oleh komputer, ujian dinilai oleh komputer, sistem keamanan dipantau oleh komputer, transaksi pembelian dilayani oleh komputer, dan banyak fungsi lain yang dikendalikan oleh komputer.

Meskipun komputer sejati pertama kali ditemukan pada tahun 1940-an, konsep komputer sebenarnya berusia lebih dari 160 tahun. **Charles Babbage**is dikreditkan dalam menciptakan pendahulu komputer modern. Pada tahun 1823 ia menerima hibah dari pemerintah Inggris untuk membangun perangkat mekanis yang disebutnya *Difference Engine*, yang dimaksudkan untuk menghitung dan mencetak tabel matematika. Perangkat itu didasarkan pada roda yang berputar dan dioperasikan dengan satu engkol. Sayangnya, teknologi saat itu belum cukup maju untuk membangun perangkat tersebut. Dia mengalami kesulitan dan akhirnya meninggalkan proyek tersebut.

Tapi skema yang lebih megah sudah ada padanya. Faktanya, salah satu alasan dia menyerah pada *Difference Engine* mungkin karena mengerjakan konsep barunya untuk mesin yang lebih baik. Dia menyebut perangkat barunya itu dengan *Analytical Engine* atau Mesin Analitik. Perangkat ini juga tidak pernah dibuat. Belum ada teknologi yang dapat digunakan untuk membuat perangkat tersebut menjadi kenyataan. Meskipun tidak pernah dibuat, *Analytical Engine* merupakan pencapaian yang luar biasa karena desainnya pada dasarnya didasarkan pada prinsip dasar yang sama dengan komputer modern. Salah satu prinsip yang menonjol adalah kemampuan programnya. Dengan *Difference Engine*, Babbage hanya dapat menghitung tabel matematika, tetapi dengan *Analytical Engine* dia akan dapat menghitung perhitungan apa pun dengan memasukkan instruksi pada sebuah kartu berlubang (*Punched card*). Metode memasukkan program ke komputer pada kartu berlubang sebenarnya diadopsi untuk mesin nyata dan masih digunakan secara luas hingga tahun 1970-an.

Mesin Analitik tidak pernah dibuat, tetapi program demonstrasi ditulis oleh **Ada Lovelace**, putri penyair Lord Byron. Bahasa pemrograman **Ada** dinamai untuk menghormati Lady Lovelace, programmer komputer pertama.

Pada akhir 1930-an John Atanasoff dari Iowa State University, bersama mahasiswa pascasarjana Clifford Berry, membuat prototipe kalkulator elektronik otomatis pertama.

Salah satu inovasi mesin mereka adalah penggunaan bilangan biner. Pada waktu yang hampir bersamaan, Howard Aiken dari Universitas Harvard sedang mengerjakan Kalkulator Kontrol Urutan Otomatis, yang lebih umum dikenal sebagai **MARK I**, dengan dukungan dari IBM dan Angkatan Laut AS. MARK I sangat mirip dengan *Analytical Engine* dalam desain dan digambarkan sebagai “mimpi Babbage menjadi kenyataan”.

MARK I adalah komputer elektromekanis berbasis relay. Relai mekanis tidak cukup cepat, dan MARK I dengan cepat digantikan oleh mesin berbasis tabung vakum elektronik. Komputer elektronik pertama yang sepenuhnya, **ENIAC I** (*Electronic Numerical Integrator and Calculator*), dibangun di University of Pennsylvania di bawah pengawasan John W. Mauchly dan J. Presper Eckert. Pekerjaan mereka dipengaruhi oleh karya John Atanasoff.

ENIAC I diprogram dengan susah payah dengan mencolokkan kabel ke panel kontrol yang menyerupai papan telepon lama. Pemrograman memakan waktu yang sangat lama dari para teknisi, dan bahkan membuat perubahan sederhana pada sebuah program adalah upaya yang memakan waktu. Sementara aktivitas pemrograman sedang berlangsung, komputer yang menghabiskan banyak dana tersebut tidak mengalami perubahan atau kemajuan. Untuk meningkatkan produktivitasnya, John von Neumann dari Universitas Princeton mengusulkan untuk menyimpan program di memori komputer. Skema program yang disimpan (*stored program*) ini tidak hanya meningkatkan kecepatan komputasi tetapi juga memungkinkan cara penulisan program yang jauh lebih fleksibel. Misalnya, karena suatu program disimpan dalam memori, komputer dapat mengubah instruksi program untuk mengubah urutan eksekusi, sehingga memungkinkan untuk mendapatkan hasil yang berbeda dari satu program.

Komputer pertama kali ini disebut dengan tabung vakum sebagai komputer generasi pertama. Komputer awal, karena ukurannya yang sangat besar sehingga dengan mudah menempati seluruh ruang, bahkan pada masa itu dapat memenuhi ruang bawah tanah sebuah gedung besar. Komputer generasi kedua, dengan transistor yang menggantikan tabung vakum, mulai muncul pada akhir 1950-an. Perbaikan pada perangkat memori juga meningkatkan kecepatan pemrosesan lebih lanjut. Pada awal 1960-an, transistor digantikan oleh sirkuit terintegrasi, dan komputer generasi ketiga muncul. Sebuah sirkuit terintegrasi tunggal pada periode ini menggabungkan ratusan transistor dan memungkinkan pembuatan minikomputer. Komputer mini cukup kecil untuk ditempatkan di atas meja di kantor dan laboratorium individu.

Kemajuan sirkuit terintegrasi memang fenomenal. Sirkuit terpadu berskala besar, umumnya dikenal sebagai chip komputer atau chip silikon, mengemas daya yang setara dengan ribuan transistor dan membuat gagasan tentang "komputer pada chip tunggal" menjadi sebuah kenyataan. Dengan sirkuit terpadu skala besar, mikrokomputer muncul pada pertengahan 1970-an. Mesin yang disebut komputer pribadi sekarang ini adalah keturunan dari mikrokomputer tahun 1970-an. Chip komputer yang digunakan dalam komputer pribadi masa kini

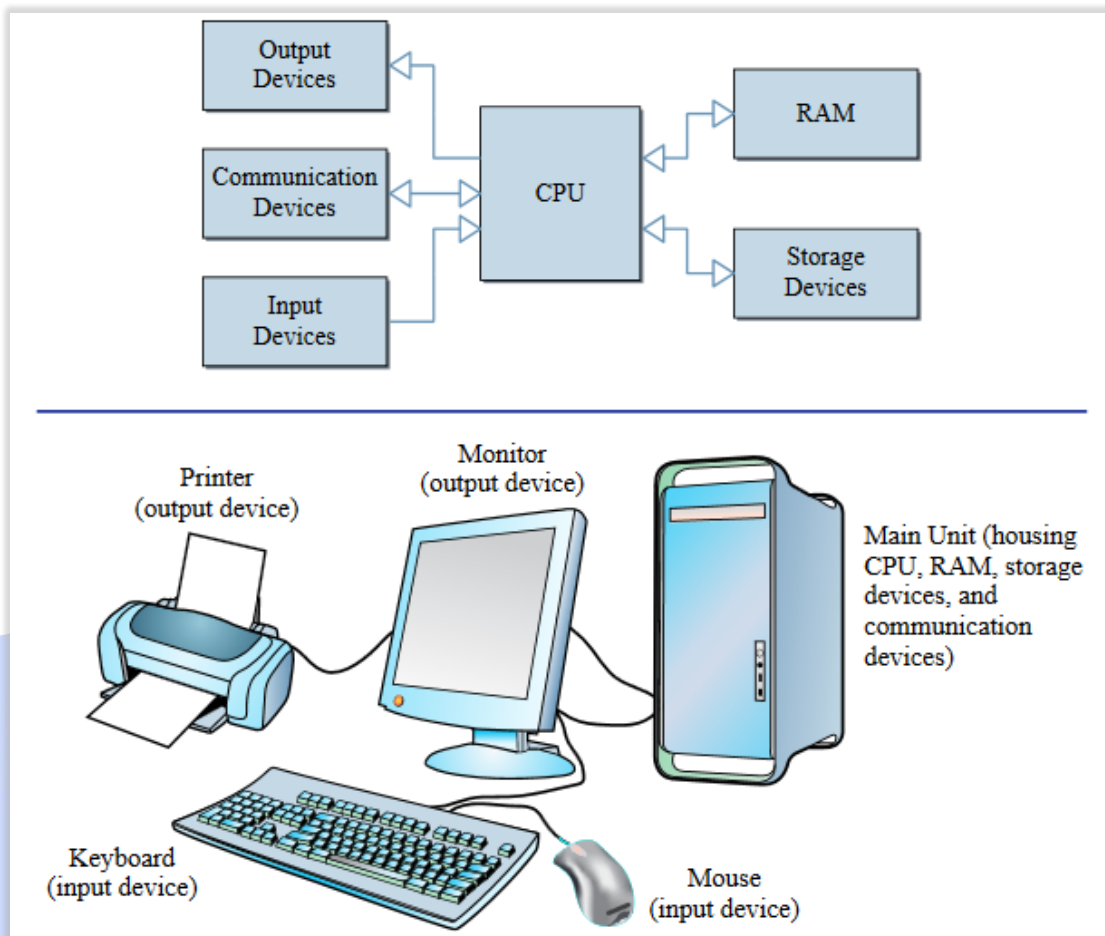
mengemas daya yang setara dengan beberapa juta transistor. Komputer pribadi adalah komputer generasi keempat.

Kata pribadi menggambarkan mesin sebagai perangkat pribadi yang dimaksudkan untuk digunakan oleh individu atau personal. Namun, tidak butuh waktu lama untuk menyadari adanya kebutuhan untuk berbagi sumber daya komputer. Misalnya, mikrokomputer awal membutuhkan sebuah printer khusus. Bukankah lebih masuk akal jika banyak komputer berbagi satu printer? Bukankah juga masuk akal untuk berbagi data antar komputer, daripada menduplikasi data yang sama pada mesin individu? Bukankah menyenangkan mengirim pesan elektronik antar komputer? Gagasan komputer berjaringan muncul untuk memenuhi kebutuhan ini.

Semua jenis komputer terhubung ke sebuah jaringan. Jaringan yang menghubungkan komputer dalam satu gedung atau di beberapa gedung terdekat disebut *Local Area Network* atau LAN. Jaringan yang menghubungkan komputer yang tersebar secara geografis disebut jaringan *Wide Area Network* atau WAN. Jaringan individu ini dapat dihubungkan lebih lanjut untuk membentuk jaringan yang saling berhubungan yang disebut internet. Internet yang paling terkenal hanya disebut Internet. Internet memungkinkan dan mudahnya berbagi informasi di seluruh dunia. Alat yang paling terkenal untuk melihat informasi di Internet adalah browser Web. Browser Web memungkinkan kita menikmati informasi multimedia yang terdiri dari teks, audio, video, dan jenis informasi lainnya.

1.2 Arsitektur Komputer

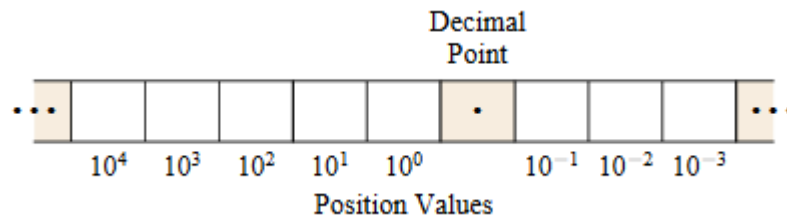
Komputer biasa saat ini memiliki lima komponen dasar: RAM, CPU, perangkat penyimpanan, perangkat I/O (*input/output*), dan perangkat komunikasi. Gambar 1.1 mengilustrasikan lima komponen ini. Sebelum kita menjelaskan komponen-komponen sebuah komputer, kita akan menjelaskan sistem bilangan biner yang digunakan dalam sebuah komputer.



Gambar 1. 1 Tampilan arsitektur yang disederhanakan untuk komputer biasa

1.2.1 Bilangan Biner

Untuk memahami sistem bilangan biner, pertama mari kita tinjau sistem bilangan desimal di mana kita menggunakan 10 digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Untuk merepresentasikan bilangan dalam sistem desimal, kami menggunakan urutan satu atau lebih dari digit ini. Nilai yang diwakili setiap digit dalam urutan bergantung pada posisinya. Misalnya, perhatikan angka 234 dan 324. Angka 2 pada angka pertama mewakili 200, sedangkan angka 2 pada angka kedua mewakili 20. Sebuah posisi dalam suatu barisan memiliki nilai yang merupakan pangkat integral dari 10. Berikut ini diagram menggambarkan bagaimana nilai posisi ditentukan:

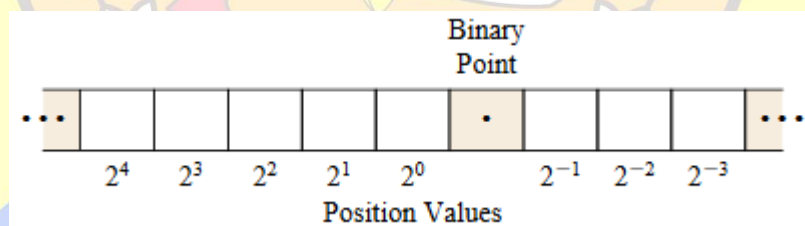


Nilai bilangan desimal (direpresentasikan sebagai deretan digit) adalah jumlah digit, dikalikan dengan nilai posisinya, seperti yang diilustrasikan berikut:

2	4	8	.	7
---	---	---	---	---

$$\begin{aligned}
 & \quad \quad \quad 10^2 \quad 10^1 \quad 10^0 \quad \quad 10^{-1} \\
 &= 2 \times 10^2 + 4 \times 10^1 + 8 \times 10^0 + 7 \times 10^{-1} \\
 &= 2 \times 100 + 4 \times 10 + 8 \times 1 + 7 \times 1/10 \\
 &= 200 + 40 + 8 + 7/10 = 248.7
 \end{aligned}$$

Dalam sistem bilangan desimal, kita memiliki 10 simbol, dan nilai posisinya adalah pangkat integral dari 10. Kita katakan bahwa 10 adalah basis atau radix dari sistem bilangan desimal. Sistem bilangan biner bekerja sama dengan sistem bilangan desimal tetapi menggunakan 2 sebagai basisnya. Sistem bilangan biner memiliki dua digit (0 dan 1) yang disebut bit, dan nilai posisi adalah pangkat dua yang tidak terpisahkan dari 2. Diagram berikut menggambarkan bagaimana nilai posisi ditentukan dalam sistem biner:



Nilai bilangan biner (direpresentasikan sebagai urutan bit) adalah jumlah bit, dikalikan dengan nilai posisinya, seperti yang diilustrasikan berikut:

1	0	1	.	1
---	---	---	---	---

$$\begin{aligned}
 & \quad \quad \quad 2^2 \quad 2^1 \quad 2^0 \quad \quad 2^{-1} \\
 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} \\
 &= 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times 1/2 \\
 &= 4 + 0 + 1 + 1/2 = 5.5
 \end{aligned}$$

*) Konversi bilangan biner ke desimal

Jadi angka biner 101.1 secara numerik setara dengan angka desimal 5.5. Ilustrasi ini menunjukkan cara mengonversi bilangan biner yang diberikan menjadi desimal yang setara. Bagaimana dengan mengonversi bilangan desimal tertentu menjadi padanan binernya?

Langkah-langkah berikut menunjukkan cara mengonversi bilangan desimal (hanya bilangan bulat) menjadi bilangan biner yang setara. Ide dasarnya adalah:

1. Bagi bilangan tersebut dengan 2.
2. Sisanya adalah nilai bit dari posisi 2^0 .
3. Bagi hasil bagi dengan 2.
4. Sisanya adalah nilai bit dari posisi 2^1 .
5. Bagi hasil bagi dengan 2.
6. Sisanya adalah nilai bit dari posisi 2^2 .
7. Ulangi prosedur ini hingga Anda tidak dapat membagi lagi, yaitu hingga hasil bagi menjadi 0.

Diagram berikut menggambarkan konversi angka desimal 25.

Division #5	Division #4	Division #3	Division #2	Division #1
$\begin{array}{r} 0 \\ 2 \overline{) 1} \\ \underline{0} \\ 1 \end{array}$	$\begin{array}{r} 1 \\ 2 \overline{) 3} \\ \underline{2} \\ 1 \end{array}$	$\begin{array}{r} 3 \\ 2 \overline{) 6} \\ \underline{6} \\ 0 \end{array}$	$\begin{array}{r} 6 \\ 2 \overline{) 12} \\ \underline{12} \\ 0 \end{array}$	$\begin{array}{r} 12 \\ 2 \overline{) 25} \\ \underline{24} \\ 1 \end{array}$
2^4	2^3	2^2	2^1	2^0
16	8	0	0	1
+ + + + = 25				

Sistem biner lebih cocok untuk komputer daripada sistem desimal karena jauh lebih mudah merancang perangkat listrik yang dapat membedakan dua status (bit 0 dan 1) daripada 10 status (digit 0 hingga 9). Misalnya, kita dapat mewakili 1 dengan menghidupkan sakelar dan 0 dengan mematikan sakelar. Dalam komputer nyata, 0 diwakili oleh tegangan listrik di bawah level tertentu dan 1 dengan tegangan listrik di atas level ini.

Saat Anda memperhatikan lebih dekat tombol on / off pada komputer dan perangkat elektronik lainnya, Anda akan melihat ikon seperti ini:



Ini adalah representasi dari angka biner 0 dan 1.

1.2.2 RAM

Random Access Memory atau RAM adalah gudang untuk instruksi program dan data yang dimanipulasi oleh program selama eksekusi. RAM dibagi menjadi beberapa sel, dengan setiap sel memiliki alamat yang unik. Biasanya, setiap sel terdiri dari 4 byte (B), dan satu byte (1 B) pada gilirannya terdiri dari 8 bit. Setiap bit, yang bisa hidup atau mati, mewakili satu digit biner. RAM diukur dengan jumlah byte yang dikandungnya. Misalnya, 128 kilobyte (KB) RAM berisi $128 \times 1024 = 131.072$ B karena 1 KB sama dengan $2^{10} = 1024$ B. Perhatikan bahwa 1 K tidak sama dengan 10^3 , meskipun $10^3 = 1000$ adalah pendekatan yang mendekati $2^{10} = 1024$. PC IBM pertama yang diperkenalkan pada tahun 1981 datang dengan 16 KB RAM, dan komputer Macintosh pertama yang diperkenalkan pada tahun 1984 datang dengan RAM 128 KB. Sebaliknya, PC biasa saat ini memiliki RAM mulai dari 1GB (gigabyte) hingga 4GB. Mengingat 1GB sama dengan 1024 MB (megabyte) dan 1 MB sama dengan 1024 KB, kita tahu bahwa 2GB berarti $2 \times 1024 \text{ MB} = 2048 \text{ MB} = 2048 \times 1024 \text{ KB} = 2.097.152 \text{ KB} = 2.097.152 \times 1024 \text{ B} = 2.147.483.648 \text{ B}$. Seiring perkembangan teknologi, RAM semakin bervariasi, mulai dari 4GB, 8GB, hingga 16GB.

1.2.3 CPU

Central processing unit atau CPU adalah otak dari sebuah komputer. CPU adalah komponen yang mengeksekusi instruksi program dengan mengambil instruksi (disimpan dalam RAM), mengeksekusinya, mengambil instruksi berikutnya, mengeksekusinya, dan seterusnya hingga menemukan instruksi untuk dihentikan. CPU berisi sejumlah kecil register, yang merupakan perangkat berkecepatan tinggi untuk menyimpan data atau instruksi sementara. CPU juga berisi *arithmetic logic unit* (ALU), yang melakukan operasi aritmatika seperti

penjumlahan dan pengurangan serta operasi logika seperti membandingkan dua angka.

CPU dicirikan oleh kecepatan jamnya. Misalnya, di Intel Pentium 200, CPU memiliki kecepatan clock 200 megahertz (MHz). Hertz adalah satuan frekuensi yang sama dengan 1 siklus per detik. Siklus adalah periode waktu antara dua status on atau off. Jadi 200 MHz sama dengan 200.000.000 siklus per detik. CPU tercepat untuk komputer pribadi yang tersedia secara komersial adalah sekitar 200 MHz pada tahun 1997. Tetapi pada awal tahun 1998, banyak vendor mulai menjual mesin 300 MHz. Dan hanya dalam 6 bulan, pada pertengahan tahun 1998, komputer pribadi yang paling top adalah mesin 400 MHz. Di akhir tahun 2008, CPU 2,93-GHz (2930-MHz) mulai diluncurkan dan diperkenalkan. Peningkatan kecepatan CPU dalam dua dekade terakhir ini sungguh mencengangkan. Kecepatan clock dari Intel 8080, CPU yang diperkenalkan pada tahun 1974 yang memulai revolusi PC, hanyalah 2 MHz. Sebaliknya, clockspeed dari Intel Pentium 4 yang diperkenalkan pada tahun 2001 adalah 2 GHz (2000 MHz). Di awal tahun 2020, semakin banyak PC atau laptop yang ditenagai processor canggih sehingga memiliki kecepatan clock yang sangat tinggi seperti Ryzen 9 3900X yang memiliki Base Clock 3.9 GHz, dan Maximum Boost Clock mencapai 4.5 GHz. Ada juga Intel Core i9 9900KF yang memiliki Base Clock 3.6 GHz, dan Maximum Boost Clock mencapai 5.0 GHz. Ada juga Intel Core i9-9900 yang memiliki Base Clock 3.1 GHz dan Maximum Boost Clock hingga mencapai 5.0 GHz.

1.2.4 I/O Devices

Perangkat input/output atau I/O memungkinkan komunikasi antara pengguna dan CPU. Perangkat input seperti keyboard dan mouse digunakan untuk memasukkan data, program, dan perintah di CPU. Perangkat keluaran seperti projector, monitor dan printer digunakan untuk menampilkan atau mencetak informasi. Perangkat I/O lainnya termasuk pemindai, pembaca kode batang, pembaca strip magnetik, kamera video digital, dan perangkat antarmuka lainnya.

1.2.5 Storage Devices

Perangkat penyimpanan seperti disk dan tape drive digunakan untuk menyimpan data dan program. Perangkat penyimpanan sekunder disebut memori non volatile, sedangkan RAM disebut memori volatile. Volatile artinya data yang disimpan di suatu perangkat akan hilang ketika daya ke perangkat dimatikan. Menjadi tidak mudah menguap dan jauh lebih murah daripada RAM, penyimpanan sekunder adalah media yang ideal untuk penyimpanan permanen data volume besar. Perangkat penyimpanan sekunder tidak dapat menggantikan RAM, karena penyimpanan sekunder jauh lebih lambat dalam akses data (mengeluarkan data dan menulis data) dibandingkan dengan RAM.

Pada tahun 2008, ada dua jenis disk: hard dan floppy (juga dikenal sebagai disket). Hardisk memberikan kinerja yang lebih cepat dan kapasitas yang lebih besar, tetapi biasanya tidak dapat dilepas; artinya, satu hard disk terpasang secara permanen ke drive disk. Floppy disk, di sisi lain, dapat dilepas, tetapi kinerjanya jauh lebih lambat dan kapasitasnya jauh lebih kecil daripada hard disk. Karena floppy disk standar hanya dapat menyimpan hingga kira-kira 1,44 MB, floppy disk menjadi kurang berguna dalam dunia file gambar dan suara multi megabyte saat ini. Mereka dengan cepat menjadi usang, dan hampir tidak ada orang yang menggunakannya lagi. Media penyimpanan yang dapat dilepas dengan kapasitas yang jauh lebih tinggi seperti zip disk (mampu menampung 100 hingga 250 MB data) menggantikan floppy disk di akhir 1990-an. Teknologi komputer bergerak sangat cepat sehingga zip disk itu sendiri sudah menjadi usang. Bentuk media penyimpanan portabel yang paling umum pada tahun 2008 adalah drive flash USB kompak, juga dikenal sebagai thumb drive, yang kapasitasnya berkisar dari 128 MB hingga 16 GB. Di tahun 2020, kapasitasnya semakin besar, dari 32 GB sampai dengan 256 GB.

Hard disk dapat menyimpan data dalam jumlah besar, biasanya mulai dari 160 GB (gigabyte; 1 GB = 1024 MB) hingga 500 GB untuk PC desktop standar pada tahun 2008. Namun seiring dengan perkembangan teknologi yang sangat cepat, PC standard sekarang sudah menggunakan *Solid-state Drive* (SSD) yang didukung dengan konektor yang semakin canggih dan kapasitas penyimpanan

yang jauh lebih besar, mulai dari 256 GB sampai dengan 512 GB bahkan satuan TB/Terabyte atau bahkan lebih besar.

1.2.6 Communication Devices

Perangkat komunikasi menghubungkan komputer pribadi ke internet. Perangkat komunikasi tradisional untuk komputer di rumah dan di kantor kecil adalah modem. Modem, yang merupakan singkatan dari modulator-demodulator, adalah perangkat yang mengubah sinyal analog menjadi sinyal digital dan digital menjadi analog. Dengan menggunakan modem, sebuah komputer dapat mengirim dan menerima data dari komputer lain melalui saluran telepon. Karakteristik paling penting dari modem adalah kecepatan transmisinya, yang diukur dalam bit per second (bps). Kecepatan tipikal untuk modem adalah 56.000 bps, biasa disebut modem 56K. Dalam kondisi ideal (tidak ada gangguan atau kemacetan saluran), modem 56K dapat mentransfer file 1 MB dalam waktu sekitar 21/2 menit. Namun, seringkali, kecepatan transfer yang sebenarnya jauh lebih rendah daripada maksimum yang mungkin. Yang disebut DSL dan modem kabel bukanlah modem sebenarnya karena mereka mentransfer data secara ketat dalam mode digital, yang memungkinkan kecepatan koneksi yang jauh lebih cepat dari 144K atau lebih. Koneksi satelit-lite berkecepatan tinggi ke Internet juga tersedia sejak tahun 2009.

Perangkat komunikasi untuk menghubungkan komputer ke LAN adalah *network interface card* (NIC). NIC dapat mentransfer data dengan kecepatan yang jauh lebih cepat daripada modem tercepat. Misalnya, jenis NIC yang disebut 10BaseT dapat mentransfer data dengan kecepatan 10 Mbps melalui jaringan. Jaringan tradisional dihubungkan, atau dikabelkan, dengan kabel. Semakin banyak jaringan yang terhubung secara nirkabel, di mana data dibawa melalui gelombang radio. Jaringan nirkabel disebut jaringan WiFi atau 802.11. Seiring perkembangan teknologi, WiFi 3 atau biasa kita sebut 802.11g dirilis pada tahun 2003, dan 802.11n atau WiFi 4 dirilis pada 2009, tahun 2013 dirilis versi 802.11ac atau biasa kita sebut dengan WiFi-5, dan pada tahun 2019 koneksi jaringan WiFi sudah mencapai versi 6 dengan kecepatan transfer data yang sangat cepat.

1.3 Bahasa Pemrograman

Bahasa pemrograman secara luas diklasifikasikan ke dalam tiga level: bahasa mesin, bahasa assembly, dan bahasa level tinggi. Bahasa mesin adalah satu-satunya bahasa pemrograman yang dimengerti CPU. Setiap jenis CPU memiliki bahasa mesinnya sendiri. Misalnya, Intel Pentium dan Motorola PowerPC memahami bahasa mesin yang berbeda. Instruksi bahasa mesin berkode biner dan instruksi mesin level satu yang sangat rendah dapat mentransfer konten dari satu lokasi memori ke register CPU atau menambahkan angka dalam dua register. Jadi, kita harus menyediakan banyak instruksi bahasa mesin untuk menyelesaikan tugas sederhana seperti mencari rata-rata 20 angka. Program yang ditulis dalam bahasa mesin mungkin terlihat seperti ini:

```
10110011 00011001
01111010 11010001 10010100
10011111 00011001
01011100 11010001 10010000
10111011 11010001 10010110
```

Satu tingkat di atas bahasa mesin adalah bahasa assembly, yang memungkinkan pemrograman simbolik "tingkat yang lebih tinggi". Alih-alih menulis program sebagai urutan bit, bahasa assembly memungkinkan programmer untuk menulis program dengan menggunakan kode operasi simbolik. Misalnya, alih-alih 10110011, kita dapat menggunakan MV untuk memindahkan konten sel memori ke register. Kita juga dapat menggunakan nama simbolik, atau mnemonik, untuk register dan sel memori. Program yang ditulis dalam bahasa assembly mungkin terlihat seperti ini:

```
MV    0,    SUM
MV    NUM,  AC
ADD   SUM,  AC
STO   SUM,  TOT
```

Karena program yang ditulis dalam bahasa assembly tidak dikenali oleh CPU, kami menggunakan assembler untuk menerjemahkan program yang ditulis dalam bahasa assembly menjadi bahasa mesin yang setara. Dibandingkan dengan menulis program dalam bahasa mesin, menulis program dalam bahasa

assembly jauh lebih cepat, tetapi tidak cukup cepat untuk menulis program yang rumit.

Bahasa tingkat tinggi dikembangkan untuk memungkinkan pemrogram menulis program lebih cepat daripada saat menggunakan bahasa rakitan. Misalnya, FORTRAN (FORmula TRANslator), bahasa pemrograman yang ditujukan untuk komputasi matematika, memungkinkan pemrogram untuk mengekspresikan persamaan numerik secara langsung sebagai:

$$X = (Y + Z) / 2$$

COBOL (COmmon Business-Oriented Language) adalah bahasa pemrograman yang ditujukan untuk aplikasi pemrosesan data bisnis. FORTRAN dan COBOL dikembangkan pada akhir 1950-an dan awal 1960-an dan masih digunakan. BASIC (Beginners All-purpose Symbolic Instructional Code) dikembangkan secara khusus sebagai bahasa yang mudah dipelajari dan digunakan oleh siswa. BASIC adalah bahasa tingkat tinggi pertama yang tersedia untuk komputer mikro. Bahasa tingkat tinggi terkenal lainnya adalah Pascal, yang dirancang sebagai bahasa akademis. Karena program yang ditulis dalam bahasa tingkat tinggi tidak dikenali oleh CPU, kita harus menggunakan kompiler untuk menerjemahkannya ke bahasa assembly yang setara.

Bahasa pemrograman C dikembangkan pada awal 1970-an di AT&T Bell Labs. Bahasa pemrograman C++ dikembangkan sebagai penerus C di awal 1980-an untuk menambahkan dukungan untuk pemrograman berorientasi objek. Pemrograman berorientasi objek adalah gaya pemrograman yang diterima lebih luas saat ini. Meskipun konsep pemrograman berorientasi objek sudah lama (bahasa pemrograman berorientasi objek pertama, Simula, dikembangkan pada akhir 1960-an), signifikansinya tidak terwujud hingga awal 1980-an. Obrolan ringan, dikembangkan di Xerox PARC, adalah bahasa pemrograman berorientasi objek lain yang terkenal. Bahasa pemrograman yang kami gunakan dalam modul mata kuliah "Pemrograman Komputer 1" ini adalah Java, bahasa pemrograman berorientasi objek terbaru, yang dikembangkan di Sun Microsystems dan sekarang dikembangkan Oracle.

1.4 Java

Java adalah bahasa berorientasi objek baru yang mendapat perhatian luas dari industri dan akademisi. Java dikembangkan oleh James Gosling dan timnya di Sun Microsystems di California. Bahasa ini didasarkan pada C dan C++ dan pada awalnya ditujukan untuk menulis program yang mengontrol peralatan konsumen seperti pemanggang roti, oven microwave, dan lainnya. Bahasa ini pertama kali disebut Oak, dinamai menurut nama pohon oak di luar kantor Gosling, tapi namanya sudah dipakai, jadi tim menamainya Java.

Java sering digambarkan sebagai bahasa pemrograman Web karena penggunaannya dalam menulis program yang disebut applet yang berjalan di dalam browser Web. Artinya, Anda memerlukan browser Web untuk menjalankan applet Java. Applet memungkinkan penyebaran informasi yang lebih dinamis dan fleksibel di Internet, dan fitur ini saja membuat Java menjadi bahasa yang menarik untuk dipelajari. Aplikasi Java adalah program berdiri sendiri lengkap yang tidak memerlukan browser Web. Aplikasi Java dapat dianalogikan dengan program yang kita tulis dalam bahasa pemrograman lain. Dalam semester ini, saya akan menjelaskan menggunakan bahasa Java dengan tujuan untuk mengajarkan dasar-dasar pemrograman berorientasi objek yang dapat diterapkan ke semua bahasa pemrograman berorientasi objek.

Para desainer bahasa Java mengambil pendekatan minimalis; mereka hanya memasukkan fitur yang sangat diperlukan dan menghilangkan fitur yang dianggap berlebihan. Pendekatan minimalis ini membuat Java menjadi bahasa yang lebih mudah dipelajari daripada bahasa pemrograman berorientasi objek lainnya. Java merupakan sarana yang ideal untuk mengajarkan dasar-dasar pemrograman berorientasi objek [1].

BAB II

OOP CONCEPTS AND IMPLEMENTATION

Sebelum kita mulai menulis program yang sebenarnya, Anda perlu mempelajari beberapa konsep dasar *object-oriented programming* (OOP). Tujuan dari bab ini adalah untuk memberi Anda pemahaman tentang pemrograman berorientasi objek dan untuk memperkenalkan dasar konseptual pemrograman berorientasi objek.

2.1 Classes and Objects

Dalam paradigma pemrograman berorientasi obyek, untuk membangun sebuah sistem atau aplikasi, sistem atau aplikasi tersebut akan dipecah menjadi bagian-bagian kecil yang disebut dengan obyek. Tiap-tiap obyek yang terbentuk dapat saling berinteraksi membentuk sebuah sistem yang dapat digunakan. Kelas merupakan *blueprint* atau *prototype* yang ditentukan oleh pengguna untuk membuat obyek-obyek sesuai kebutuhan dan fungsinya.

Segala sesuatu di Java pasti dikaitkan dengan kelas dan objek, bersama dengan atribut dan metodenya. Misalnya: dalam kehidupan nyata, mobil adalah benda. Mobil memiliki atribut, seperti berat dan warna, serta memiliki fungsi/metode, seperti penggerak dan rem. Contoh Membuat Kelas dengan nama `KelasPemrograman`.

```
public class KelasPemrograman {  
  
}
```

Di Java, sebuah objek dibuat dari sebuah kelas. Kita telah membuat kelas bernama `KelasPemrograman`, jadi sekarang kita dapat menggunakan kelas tersebut untuk membuat sebuah objek. Untuk membuat objek `KelasPemrograman`, tentukan nama kelas, diikuti dengan nama objek, dan gunakan kata kunci `new`. Contoh Membuat Objek:

```
public class KelasPemrograman {  
    int nilai = 99;  
  
    public static void main(String[] args) {  
        KelasPemrograman obyek = new KelasPemrograman(); //objek 1  
    }  
}
```

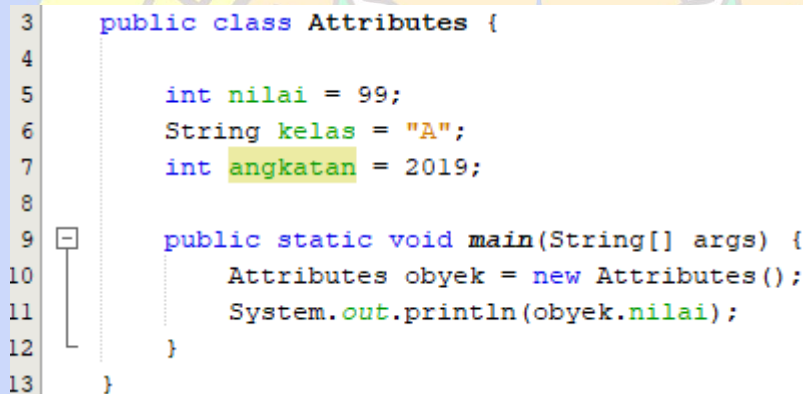

2.2 Class Attributes

Sebuah variabel yang dibuat di dalam kelas biasa disebut dengan atribut kelas (*class attribute*) atau juga bisa kita sebut dengan field. Misalnya kita ingin membuat sebuah kelas baru dengan nama "**Attributes**" dengan tiga (tiga) atribut "nilai", "kelas", "angkatan".

```
public class Attributes {
    int nilai = 99;
    String kelas = "A";
    int angkatan = 2019;
}
```

2.2.1 Accessing Attributes

Untuk mengakses sebuah atribut kelas, pastikan bahwa kita telah membuat atribut yang akan diakses. Adapun caranya adalah dengan menggunakan sintaks dot/titik. Contoh kode program:



```
3 public class Attributes {
4
5     int nilai = 99;
6     String kelas = "A";
7     int angkatan = 2019;
8
9     public static void main(String[] args) {
10         Attributes obyek = new Attributes();
11         System.out.println(obyek.nilai);
12     }
13 }
```

Pada kode di atas, kita membuat sebuah kelas dengan nama "**Attributes**", dan sebuah objek dengan variabel "nilai" yang diinisialisasi dengan nilai 99. Pada baris ke-10, kita membuat sebuah objek dari sebuah kelas "**Attributes**" dengan nama "obyek". Proses ini biasa kita sebut dengan instansiasi. Setelah kita melakukan instansiasi, selanjutnya kita dapat mengakses semua atribut yang dimiliki oleh kelas dengan cara menggunakan variabel *obyek* dan diikuti dengan titik/dot, kemudian nama variabel atribut yang ingin diakses. Contohnya:

```
obyek.nilai ➔ Mengakses atribut nilai
obyek.kelas ➔ Mengakses atribut kelas
obyek.angkatan ➔ Mengakses atribut angkatan
```


2.2.2 Modify Attributes

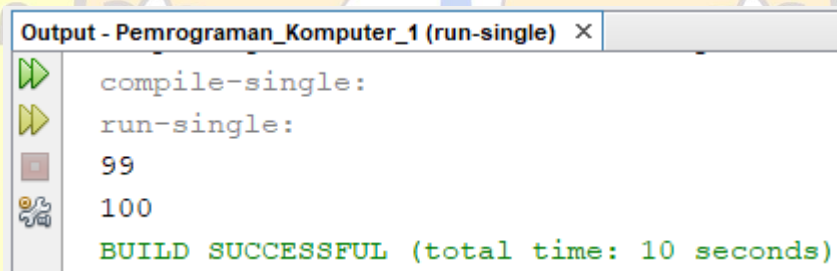
Selain mengakses nilai pada sebuah atribut kelas, kita juga dapat merubah nilai pada atribut tersebut. Cara merubah nilai pada atribut langkahnya sama seperti melakukan inisialisasi nilai pada atribut, yaitu dengan menggunakan operator = dan diikuti dengan nilai baru yang diinginkan.

```

3 public class Attributes {
4
5     int nilai = 99;
6     String kelas = "A";
7     int angkatan = 2019;
8
9     public static void main(String[] args) {
10         Attributes obyek = new Attributes();
11         System.out.println( obyek.nilai );
12         obyek.nilai = 100;
13         System.out.println( obyek.nilai );
14     }
15 }

```

Pada baris ke-12, nilai pada variabel "nilai" yang semula 99 diganti dengan 100. Output jika kode di atas dijalankan adalah seperti berikut:



```

Output - Pemrograman_Komputer_1 (run-single) x
compile-single:
run-single:
99
100
BUILD SUCCESSFUL (total time: 10 seconds)

```

2.3 Methods

Di java, sebuah method atau metode merupakan sekumpulan statement atau pernyataan yang dikelompokkan untuk menjalankan satu atau lebih operasi. Method juga bisa kita kenal dengan fungsi atau subrutin pada bahasa pemrograman tertentu. Sebagai contoh ketika kita memanggil method `println()` pada pernyataan "`System.out.println();`", maka sebenarnya sistem mengeksekusi atau menjalankan beberapa pernyataan untuk menampilkan pesan di konsol.

2.3.1 Creating and Calling a Method

Ketika membuat sebuah metode, kita dapat menentukan apakah metode yang kita buat mengembalikan nilai atau tidak, dan apakah metode memiliki parameter atau tidak. Hal ini dapat disesuaikan dengan kebutuhan.

Sintaks:

```
modifier returnType nameOfMethod (Parameter List) {  
    // method body  
}
```

Sintaks di atas menjelaskan:

- ✓ **modifier** - Mendefinisikan jenis akses metode dan opsional untuk digunakan.
- ✓ **returnType** - Metode dapat mengembalikan nilai.
- ✓ **nameOfMethod** - Nama metode.
- ✓ **Parameter List** - Daftar parameter, yaitu jenis, urutan, dan jumlah parameter dari suatu metode. Sifatnya opsional, metode diperbolehkan tanpa menggunakan parameter.
- ✓ **method body** - berisi statement atau pernyataan yang akan dijalankan oleh method.

Beikut merupakan beberapa contoh dari pembauatan method/metode:

A. Method yang tidak mengembalikan nilai dan tidak menggunakan parameter formal (baris 9-11).

Ciri-ciri:

- ✓ Menggunakan keyword **void**
- ✓ Tidak diperlukan / tidak adanya keyword **return**

Keterangan:

- ✓ **private: access modifier** (method hanya bisa diakses melalui kelas ini saja)
- ✓ **cetakNilai: Nama method**

```

3      public class Attributes {
4
5          int nilai = 99;
6          String kelas = "A";
7          int angkatan = 2019;
8
9      private void cetakNilai() {
10         System.out.println("Nilai: " + nilai);
11     }
12
13     public static void main(String[] args) {
14         Attributes obyek = new Attributes();
15         obyek.cetakNilai();
16         obyek.nilai = 100;
17         obyek.cetakNilai();
18     }
19 }

```

Pada baris ke-15 dan ke-17 merupakan contoh pemanggilan method `cetakNilai` melalui obyek "obyek". Method ini, ketika dipanggil akan menjalankan satu statemen baris perintah untuk menampilkan nilai.

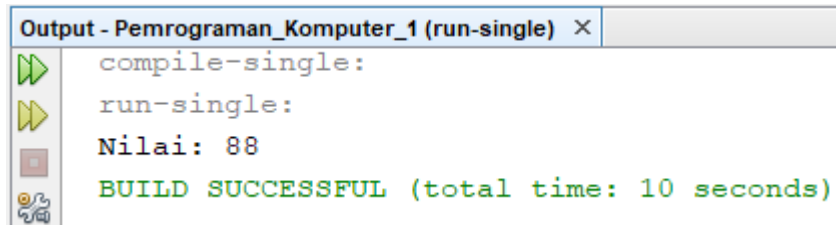
B. Method tidak mengembalikan nilai, memiliki satu parameter (baris ke 13-15)

```

3      public class Attributes {
4
5          int nilai = 99;
6          String kelas = "A";
7          int angkatan = 2019;
8
9      private void cetakNilai() {
10         System.out.println("Nilai: " + nilai);
11     }
12
13     private void ubahNilai(int nilaiBaru) {
14         this.nilai = nilaiBaru;
15     }
16
17     public static void main(String[] args) {
18         Attributes obyek = new Attributes();
19         obyek.ubahNilai(88);
20         obyek.cetakNilai();
21     }
22 }

```

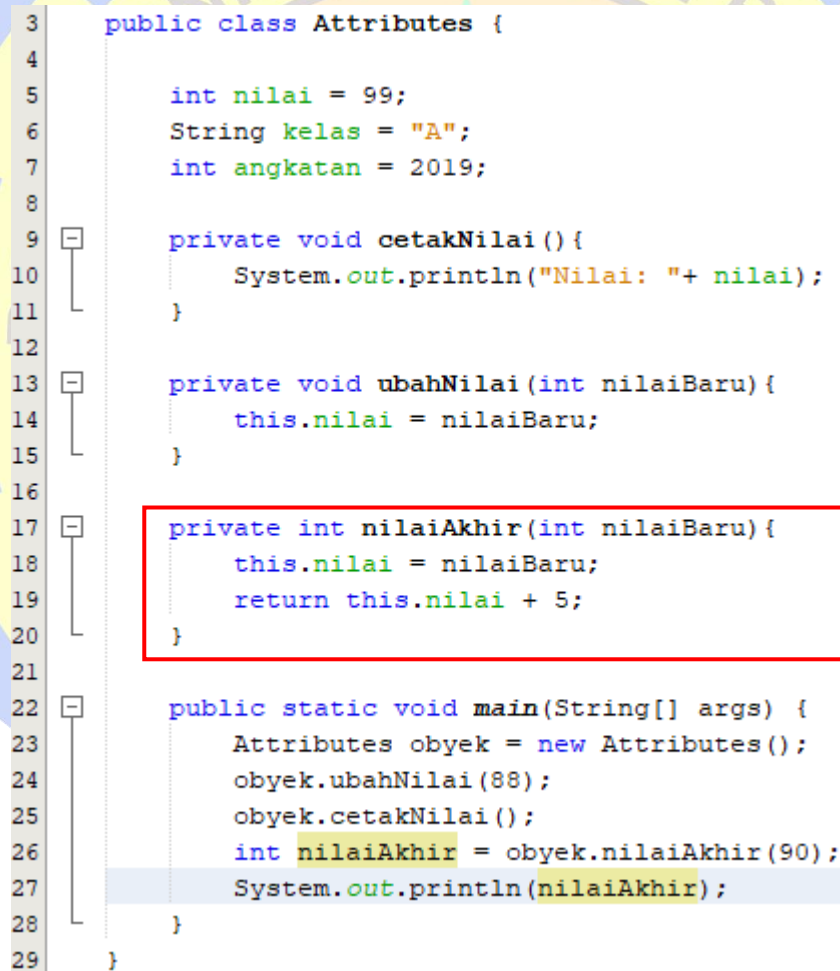
Pada baris kode di atas, baris ke-19 melakukan pemanggilan method “ubahNilai” dengan menyertakan nilai parameter aktual (88), dimana nilai tersebut akan digunakan untuk mengubah nilai pada variabel “nilai”. Kode pada baris ke-20 memanggil method “cetakNilai”. Jika kelas ini dijalankan, maka akan menghasilkan keluaran sebagai berikut:



```

Output - Pemrograman_Komputer_1 (run-single) x
compile-single:
run-single:
Nilai: 88
BUILD SUCCESSFUL (total time: 10 seconds)
  
```

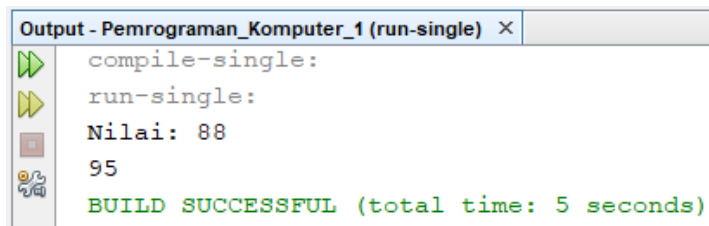
C. Method mengembalikan nilai, menggunakan parameter (baris ke 17-20)



```

3 public class Attributes {
4
5     int nilai = 99;
6     String kelas = "A";
7     int angkatan = 2019;
8
9     private void cetakNilai(){
10         System.out.println("Nilai: " + nilai);
11     }
12
13     private void ubahNilai(int nilaiBaru){
14         this.nilai = nilaiBaru;
15     }
16
17     private int nilaiAkhir(int nilaiBaru){
18         this.nilai = nilaiBaru;
19         return this.nilai + 5;
20     }
21
22     public static void main(String[] args) {
23         Attributes obyek = new Attributes();
24         obyek.ubahNilai(88);
25         obyek.cetakNilai();
26         int nilaiAkhir = obyek.nilaiAkhir(90);
27         System.out.println(nilaiAkhir);
28     }
29 }
  
```

Baris ke-26: mendeklarasikan variabel “nilaiAkhir” dan mengisi nilai dari kembalian method “nilaiAkhir”. Jika program dijalankan, maka keluarannya adalah sebagai berikut:



```

Output - Pemrograman_Komputer_1 (run-single) X
compile-single:
run-single:
Nilai: 88
95
BUILD SUCCESSFUL (total time: 5 seconds)

```

2.3.2 Overloading Method

Metode overloading merupakan method-method yang berada dalam satu kelas dengan nama yang sama, namun memiliki parameter yang berbeda (baik berbeda tipe datanya maupun jumlah parameternya). Contohnya:

```

public class Methods {
    private int uts, uas, tugas;
    private String[] dataNilai;

    public void setNilai(int uts) {
        this.uts = uts;
    }

    public void setNilai(int uts, int uas) {
        this.uts = uts;
        this.uas = uas;
    }

    public void setNilai(int uts, int uas, int tugas) {
        this.uts = uts;
        this.uas = uas;
        this.tugas = tugas;
    }

    public void setNilai(String[] dataNilai) {
        this.dataNilai = dataNilai;
    }
}

```

Pada kode di atas, kelas `Methods` memiliki empat atribut, dan empat *method* yang namanya sama. Namun, parameter yang digunakan berbeda (jumlah dan tipe datanya). Method pertama memiliki satu parameter, method kedua memiliki 2 parameter, method ketiga memiliki tiga parameter, dan method keempat memiliki satu parameter dengan tipe data yang berbeda, yaitu array string.

2.4 Constructor

Konstruktor atau *constructor* merupakan method spesial yang digunakan untuk menginisialisasi objek. Konstruktor dipanggil ketika sebuah objek dari kelas dibuat. Nama konstruktor adalah sama dengan nama kelas, dan secara sintaks mirip dengan method, namun tidak memiliki nilai kembalian eksplisit.

Semua kelas memiliki konstruktor, baik kita mendefinisikannya atau tidak, karena Java secara otomatis menyediakan konstruktor default yang menginisialisasi kelas yang kita buat. Namun, setelah kita menentukan atau membuat konstruktor kita sendiri, maka konstruktor default tidak lagi digunakan.

Contoh:

```

3  public class Konstruktor {
4
5
6  public static void main(String[] args) {
7      Konstruktor kons = new Konstruktor();
8  }
9  }

```

Pada kode di atas (baris ke-7), dilakukan deklarasi objek “kons” dan inisialisasi variabel `kons` dengan menggunakan konstruktor default yang dibuat oleh Java. Bisa kita lihat pada seluruh kode di kelas tersebut, bahwa di sana tidak terdapat konstruktor. Namun demikian, kita tetap bisa membuat/mendefinisikan konstruktor sendiri jika ingin menerapkan beberapa pengaturan pada saat objek kelas diinstansiasi (pada baris ke-6), seperti pada kode di bawah ini:

```

3  public class Konstruktor {
4
5  public Konstruktor() {
6      //constructor body
7  }
8
9  public static void main(String[] args) {
10     Konstruktor kons = new Konstruktor();
11 }
12 }

```

2.5 Modifiers

Di Java, kita dapat mengontrol level akses pada kelas, atribut, method, dan konstruktor. Misalnya: kata kunci `public` adalah pengubah akses (*access modifier*), artinya digunakan untuk mengatur tingkat akses pada kelas, atribut, metode, dan konstruktor.

2.5.1 Access Modifiers

Access Modifier merupakan fitur di Java yang digunakan untuk mengontrol tingkat atau level akses pada kelas, atribut, method, dan konstruktor.

Tabel access modifier untuk kelas

Modifier	Description
<code>public</code>	Kelas dapat diakses oleh kelas lain, baik di package yang sama maupun di package yang berbeda.
<code>default</code>	Kelas hanya dapat diakses oleh kelas dalam paket yang sama. Modifier ini otomatis digunakan ketika kita tidak menentukan modifier pada kelas.

Tabel access modifier untuk atribut, method, dan konstruktor

Modifier	Description
<code>public</code>	Kode dapat diakses oleh semua kelas
<code>private</code>	Kode hanya dapat diakses di dalam kelas yang dideklarasikan
<code>default</code>	Kode hanya dapat diakses dalam paket yang sama. Modifier ini otomatis digunakan ketika kita tidak menentukan modifier.
<code>protected</code>	Kode dapat diakses dalam paket dan subkelas yang sama.

2.5.2 Non-Access Modifiers

Non-Access Modifier tidak mengontrol tingkat atau level akses, tetapi menyediakan fungsionalitas lain.

Tabel access modifier untuk kelas

Modifier	Description
<code>final</code>	Kelas tidak dapat diwarisi oleh kelas lain
<code>abstract</code>	Kelas tidak dapat digunakan untuk membuat objek

Tabel access modifier untuk atribut dan method,

Modifier	Description
<code>final</code>	Atribut dan metode tidak dapat diganti (<i>override</i>) / diubah
<code>static</code>	Atribut dan metode menjadi milik kelas, bukan objek

abstract	Hanya dapat digunakan di kelas abstrak, dan hanya dapat digunakan pada method. Method tersebut tidak memiliki isian atau pernyataan apapun, misalnya <code>abstrak void run();</code> . Isian kode atau pernyataan disediakan oleh subclass
transient	Atribut dan metode dilewati saat membuat serial objek yang memuatnya
synchronized	Metode hanya dapat diakses oleh satu <i>thread</i> dalam satu waktu
volatile	Nilai atribut tidak di-cache pada lokal <i>thread</i> , dan selalu dibaca dari "memori utama"

2.6 Inheritance

Pewarisan atau Inheritance merupakan mekanisme di java dimana satu kelas diijinkan untuk mewarisi fitur (field dan metode) dari kelas lain. Sebelum masuk ke contoh dan implementasi, terdapat beberapa terminologi penting yang perlu diperhatikan, diantaranya:

- A. **Super Class:** Kelas yang fiturnya diwarisi (kelas dasar atau induk).
- B. **Sub Class:** Kelas yang mewarisi kelas lain (kelas turunan, kelas diperpanjang, atau kelas anak). Subclass dapat menambahkan field dan metodenya sendiri selain field dan metodenya super class.
- C. **Reusability:** Inheritance mendukung konsep "reusability", yaitu ketika kita ingin membuat kelas baru dan sudah ada kelas yang menyertakan beberapa kode yang kita inginkan, kita bisa mendapatkan kelas baru kita dari kelas yang sudah ada. Dengan melakukan ini, kita dapat menggunakan kembali field dan metode kelas yang ada.

Kata kunci yang digunakan untuk melakukan inheritance adalah "`extends`"

Sintaks:

```
class derivedClass extends baseClass {
    //methods and fields
}
```

Pada sintaks di atas, **derivedClass** merupakan kelas turunan (kelas yang mewarisi), dan **baseClass** merupakan kelas dasar (kelas yang diwariskan). Dengan demikian, semua method dan field yang dimiliki oleh kelas **baseClass** secara otomatis juga dimiliki oleh kelas **derivedClass**.

Dalam contoh pewarisan di bawah ini, kelas **Sepeda** adalah kelas dasar yang memiliki tiga field yaitu: merek, warna, dan tahunRilis. satu konstruktor (dengan tiga parameter), dan tiga method yaitu: method `aturWarna`, `aturTahunRilis` dan `aturMerek`. Kelas **SepedaGunung** adalah kelas turunan yang memperluas kelas **Sepeda** yang memiliki satu field yaitu: `tinggiDudukan`, satu konstruktor (dengan empat parameter), dan dua method yaitu: `aturTinggiDudukan`, dan `lihatInfoSepeda`. Kelas **Tes** adalah kelas yang digunakan menjalankan program untuk menguji implementasi pewarisan kelas.

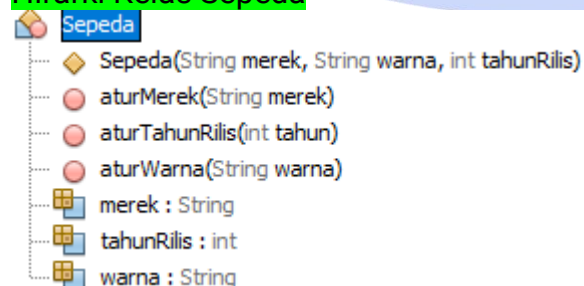
Kode Program: Kelas Sepeda

```

3      public class Sepeda {
4          String merek, warna;
5          int tahunRilis;
6
7      [-] public Sepeda(String merek, String warna, int tahunRilis) {
8          this.merek = merek;
9          this.warna = warna;
10         this.tahunRilis = tahunRilis;
11     }
12
13     [-] public void aturWarna(String warna) {
14         this.warna = warna;
15     }
16
17     [-] public void aturTahunRilis(int tahun) {
18         this.tahunRilis = tahun;
19     }
20
21     [-] public void aturMerek(String merek) {
22         this.merek = merek;
23     }
24 }

```

Hirarki Kelas Sepeda



Kode Program: Kelas SepedaGunung

```

3      public class SepedaGunung extends Sepeda{
4          double diameterRoda;
5
6      [-] public SepedaGunung(String merek, String warna, int tahun, double diameter){
7          super(merek, warna, tahun);
8          this.diameterRoda = diameter;
9      }
10
11     [-] public void aturDiameterRoda(double diameter){
12         this.diameterRoda = diameter;
13     }
14
15     [-] public void lihatInfoSepeda(){
16         System.out.println("====Informasi Sepeda====");
17         System.out.println("Merek: "+merek);
18         System.out.println("Warna: "+warna);
19         System.out.println("Tahun: "+tahunRilis);
20         System.out.println("Diameter Roda: "+diameterRoda);
21         System.out.println("");
22     }
23 }
24

```

Kelas SepedaGunung merupakan kelas turunan dari Sepeda, dengan demikian kelas SepedaGunung secara otomatis mewarisi semua *field* dan *method* yang dimiliki oleh kelas Sepeda yang dapat digunakan tanpa harus membuatnya. Tentunya hal ini dapat menghemat penulisan kode program, karena tidak perlu menulis ulang *field* dan *method* pada kelas turunan.

Kode Program: Kelas Tes

```

3      public class Tes {
4
5      [-] public static void main(String[] args) {
6          SepedaGunung sg = new SepedaGunung("Polygon", "Merah", 2010, 27.5);
7          sg.lihatInfoSepeda();
8          sg.aturWarna("Biru");
9          sg.aturDiameterRoda(26.5);
10         sg.lihatInfoSepeda();
11     }
12 }

```

Output:

```

Output - Pemrograman_Komputer_1 (run-single) x
>> =====Informasi Sepeda=====
>> Merek: Polygon
>> Warna: Merah
>> Tahun: 2010
>> Diameter Roda: 27.5

>> =====Informasi Sepeda=====
>> Merek: Polygon
>> Warna: Biru
>> Tahun: 2010
>> Diameter Roda: 26.5

BUILD SUCCESSFUL (total time: 5 seconds)

```

2.7 Encapsulation

Enkapsulasi atau enkapsulasi merupakan pembungkus data ke dalam satu unit untuk mengikat kode dan data yang dimanipulasi. Enkapsulasi juga dimaksudkan untuk melindungi data dengan mencegah data tidak dapat diakses oleh kode di luar pelindung. Beberapa garis besar, mekanisme enkapsulasi dapat disimpulkan sebagai berikut:

- A. Secara teknis dalam enkapsulasi, sebuah variabel atau data yang ada di dalam sebuah kelas disembunyikan dari kelas lain dan hanya dapat diakses melalui fungsi atau method kelas itu sendiri di mana variabel atau data tersebut dideklarasikan.
- B. Variabel maupun data disembunyikan dengan cara menjadikan variabel atau data menjadi pribadi dengan menambahkan kata kunci `private` pada variabel atau data yang ingin disembunyikan dan menulis metode publik di kelas untuk mengatur dan mendapatkan nilai variabel atau data.

Contoh:

```

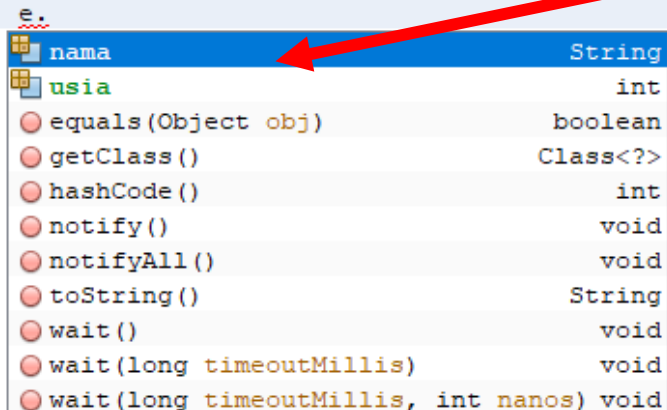
3  public class Encapsulate {
4      String nama;
5      int usia;
6
7  }

public class Tes {

    public static void main(String[] args) {
        Encapsulate e = new Encapsulate();

        e.
    }
}

```



Attribute/Method	Type
nama	String
usia	int
equals(Object obj)	boolean
getClass()	Class<?>
hashCode()	int
notify()	void
notifyAll()	void
toString()	String
wait()	void
wait(long timeoutMillis)	void
wait(long timeoutMillis, int nanos)	void

Pada kode di atas, atribut atau *field* (atau yang biasa kita sebut data member) `nama`, dan `usia` masih dapat diakses oleh kelas lain, hal ini karena tidak diterapkannya enkapsulasi. Untuk menyembunyikannya, kita tambahkan kata kunci `private` pada atribut `nama` dan `usia`. Selanjutnya, untuk mengakses dan memodifikasi data pada atribut tersebut dapat dilakukan dengan membuat method publik dan menambahkan pernyataan untuk memodifikasi nilai pada atribut `nama` dan `usia`.

Data: Di-enkapsulasi

```

3  public class Encapsulate {
4      private String nama;
5      private int usia;
6
7      public String getNama() {
8          return nama;
9      }
10
11     public void setNama(String nama) {
12         this.nama = nama;
13     }
14
15     public int getUsia() {
16         return usia;
17     }
18
19     public void setUsia(int usia) {
20         this.usia = usia;
21     }
22
23 }

```

```

public class Tes {

    public static void main(String[] args) {
        Encapsulate e = new Encapsulate();
        e.
    }
}

```

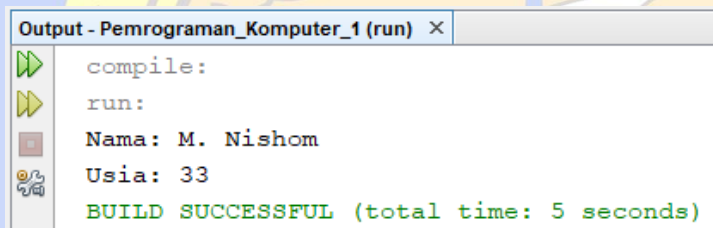
	<code>equals(Object obj)</code>	<code>boolean</code>
	<code>getClass()</code>	<code>Class<?></code>
	<code>getNama()</code>	<code>String</code>
	<code>getUsia()</code>	<code>int</code>
	<code>hashCode()</code>	<code>int</code>
	<code>notify()</code>	<code>void</code>
	<code>notifyAll()</code>	<code>void</code>
	<code>setNama(String nama)</code>	<code>void</code>
	<code>setUsia(int usia)</code>	<code>void</code>
	<code>toString()</code>	<code>String</code>
	<code>wait()</code>	<code>void</code>
	<code>wait(long timeoutMillis)</code>	<code>void</code>
	<code>wait(long timeoutMillis, int nanos)</code>	<code>void</code>

Pada kode di atas, atribut (`nama` dan `usia`) pada kelas `Encapsulate` tidak dapat dilihat, sehingga aman dan tidak dapat diakses oleh kelas lain. Sedangkan jika ingin mengakses dan memanipulasi nilai pada atribut tersebut **hanya** dapat dilakukan dengan method publik yang sudah didefinisikan pada kelas `Encapsulate`. Contohnya sebagai berikut:

```
public class Tes {

    public static void main(String[] args) {
        Encapsulate e = new Encapsulate();
        e.setNama("M. Nishom");
        e.setUsia(33);
        System.out.println("Nama: "+e.getNama());
        System.out.println("Usia: "+ e.getUsia());
    }
}
```

Output:



```
Output - Pemrograman_Komputer_1 (run) x
compile:
run:
Nama: M. Nishom
Usia: 33
BUILD SUCCESSFUL (total time: 5 seconds)
```

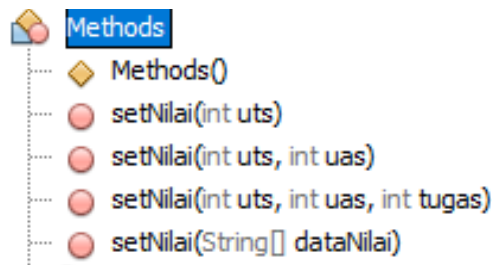
2.8 Polymorphism

Dalam bahasa pemrograman berorientasi objek, banyak kelas yang terkait satu sama lain melalui pewarisan. Hal itulah yang melahirkan adanya prinsip polimorfisme atau *polymorphism*, dimana kelas dapat memiliki banyak “bentuk” method meskipun namanya sama. Bentuk dimaksudkan isi, parameter maupun tipe datanya dapat berbeda-beda.

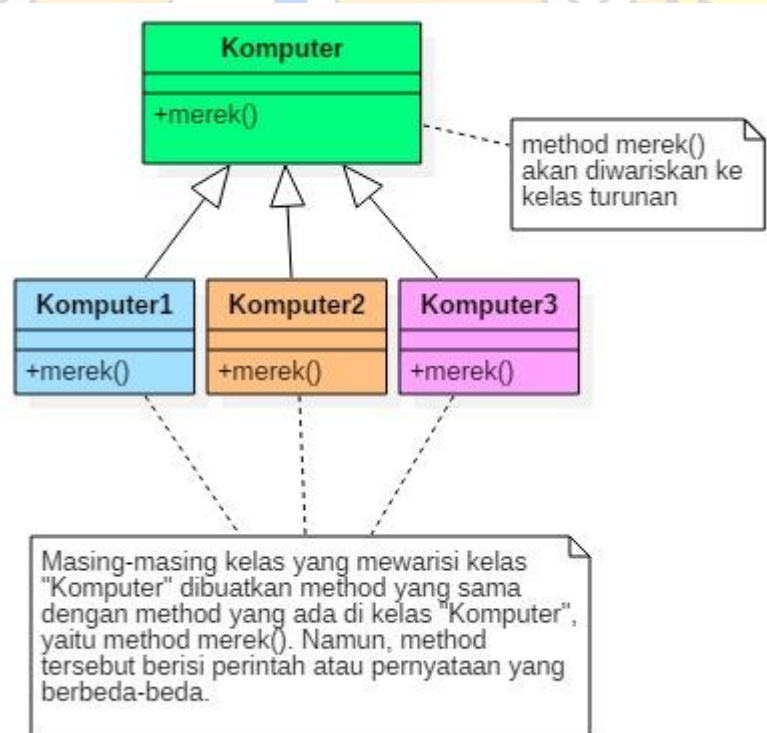
“Berbeda-beda bentuk namun
satu hati
namanya sama”
~ Polymorphism

Polymorphism memiliki dua jenis, yaitu statis dan dinamis. Kedua jenis ini memiliki ciri khasnya masing-masing, yaitu:

A. Static Polymorphism: menggunakan metode overloading



B. Dinamic Polymorphism, menggunakan metode overriding. Metode overriding merupakan metode yang dibuat dalam kelas turunan atau subclass, tetapi namanya sama dengan metode yang dimiliki kelas dasar atau superclass. Namun, meskipun namanya sama tetapi mereka memiliki perintah atau *statement* yang berbeda-beda. Untuk lebih jelasnya, diagram kelas berikut dapat mengilustrasikan bagaimana sebuah overriding.



Kode: Komputer.java

```
public class Komputer {  
  
    public void merek() {  
        System.out.println("Merek Komputer");  
    }  
}
```

Kode: Komputer1.java

```
public class Komputer1 extends Komputer {  
  
    @Override  
    public void merek() {  
        System.out.println("Asus");  
    }  
}
```

Kode: Komputer2.java

```
public class Komputer2 extends Komputer {  
  
    @Override  
    public void merek() {  
        System.out.println("Lenovo");  
    }  
}
```

Kode: Komputer3.java

```
public class Komputer3 extends Komputer {  
  
    @Override  
    public void merek() {  
        System.out.println("Acer");  
    }  
}
```

Method `merek()` pada kode di atas akan mencetak teks yang berbeda-beda pada layar konsol. Kode `@Override` di kelas `Komputer1`, `komputer2`, dan kelas `komputer3` menunjukkan bahwa deklarasi metode di bawahnya dimaksudkan untuk menggantikan deklarasi metode yang sama yang ada di dalam kelas `Komputer` (superclass).

2.9 Abstraction

Abstraksi data (*Data Abstraction*) adalah proses menyembunyikan detail tertentu dan hanya menampilkan informasi penting kepada pengguna. Abstraksi atau *Abstraction* dapat dicapai dengan *abstract class* atau *interface* (yang akan

Anda pelajari lebih lanjut di sub-bab berikutnya). Kata kunci *abstract* adalah non-access modifier, digunakan untuk *class* dan *method*:

- A. Abstract Class: adalah kelas terbatas yang tidak dapat digunakan untuk membuat objek (untuk mengaksesnya, itu harus diwarisi dari kelas lain).
- B. Abstract Method: hanya dapat digunakan dalam *abstract class*, dan tidak memiliki isi. Tubuh atau isi disediakan oleh *subclass* (kelas yang mewarisi *abstract class*).

Kelas abstrak dapat memiliki metode abstrak dan reguler, **Contohnya:**

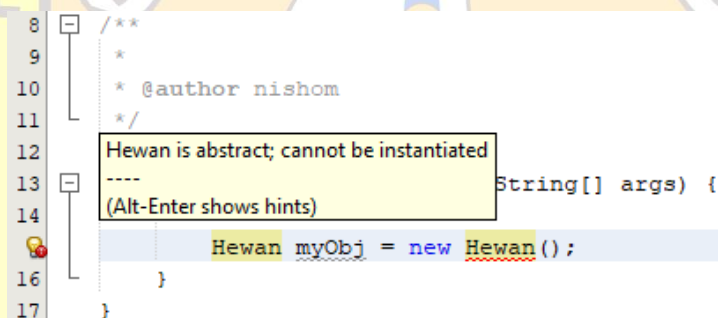
```
abstract class Hewan {

    public abstract void suaraHewan();

    public void tidur() {
        System.out.println("Zzz");
    }

}
```

Dari contoh kode di atas, kita tidak mungkin dapat membuat objek kelas Hewan. Misalkan kita mencoba membuat objek kelas tersebut dengan melakukan instansiasi, maka akan menghasilkan *error* atau kesalahan.



```
8  /**
9   *
10  * @author nishom
11  */
12  Hewan is abstract; cannot be instantiated
13  ----
14  (Alt-Enter shows hints)
15  String[] args) {
16
17  Hewan myObj = new Hewan();
18  }
19  }
```

Untuk mengakses kelas abstrak, maka kelas tersebut harus diwarisi dari kelas lain. Contohnya kita buat kelas baru dengan nama *Kucing* yang mewarisi kelas *Hewan*, dan mengganti isi atau *body* method *suaraHewan()* dengan perintah mencetak suara kucing.

```
7  public class Kucing extends Hewan{
8
9      @Override
10     public void suaraHewan() {
11         System.out.println("Kucing bersuara: Meong");
12     }
13
14 }
```


Untuk melihat penggunaannya, kita buat kelas baru dengan nama Tes, selanjutnya buat objek dari kelas Kucing, dan panggil method-methodnya.

Kode Program:

```

2
3  /**
4   *
5   * @author nishom
6   */
7  public class Tes {
8
9      public static void main(String[] args) {
10         Hewan obj = new Kucing();
11         obj.suaraHewan();
12         obj.tidur();
13     }
14 }

```

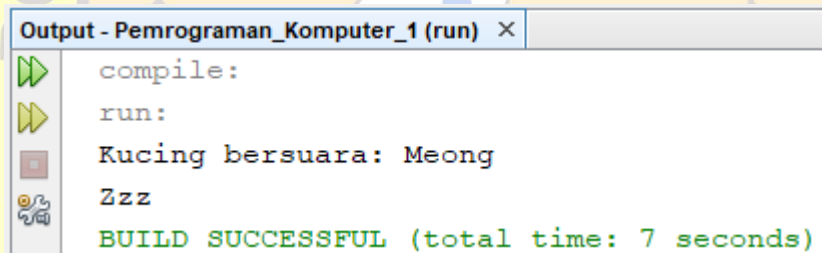
Keterangan:

Hewan → abstract class

obj → nama variabel objek

new Kucing() → perintah untuk melakukan instansiasi objek Kucing

Output:



```

Output - Pemrograman_Komputer_1 (run) X
compile:
run:
Kucing bersuara: Meong
Zzz
BUILD SUCCESSFUL (total time: 7 seconds)

```

2.10 Interfaces

Antarmuka atau *interface* merupakan "*abstract class*" yang digunakan untuk mengelompokkan metode terkait dengan tanpa pernyataan atau kode apapun di dalamnya, Contoh:

```

2
3  /**
4   *
5   * @author nishom
6   */
7  public interface Hewan {
8
9      public void suaraHewan(); // interface method (tidak memiliki isi/body)
10     public void tidur(); // interface method (tidak memiliki isi/body)
11 }

```

Untuk mengakses *interface method* pada kode di atas, *interface Hewan* harus "diimplementasikan" (seperti diwariskan) oleh kelas lain dengan kata kunci *implements* (bukan *extends*). Isi atau *body* dari *interface method* disediakan oleh kelas yang mengimplementasikan *interface Hewan*, contoh:

```

2
3  /**
4   *
5   * @author nishom
6   */
7  // Kucing mengimplementasikan interface Hewan
8  public class Kucing implements Hewan {
9
10     @Override
11     public void suaraHewan() {
12         // Isian metode suaraHewan() disediakan di sini
13         System.out.println("Suara kucing: meong");
14     }
15
16     @Override
17     public void tidur() {
18         // Isian metode tidur() disediakan di sini
19         System.out.println("Zzz");
20     }
21 }

```

Selanjutnya kita buat kelas *Tes* untuk membuat objek dan menjalankan method-method yang disediakan dalam kelas tersebut.

Kode Program:

```

7  public class Tes {
8
9      public static void main(String[] args) {
10         Kucing obj = new Kucing();
11         obj.suaraHewan();
12         obj.tidur();
13     }
14 }

```

Output:

```

Output - Pemrograman_Komputer_1 (run-single) X
>> compile-single:
>> run-single:
Suara kucing: meong
Zzz
BUILD SUCCESSFUL (total time: 7 seconds)

```

2.11 Packages and API

Sebuah paket atau *package* di Java digunakan untuk mengelompokkan kelas yang terkait, atau kalau dalam file explorer biasa kita sebut dengan folder. Paket digunakan untuk menghindari konflik nama *file*, dan untuk menulis kode yang lebih mudah dirawat atau diperbaiki. Paket dibagi menjadi dua kategori, yaitu: Paket Bawaan atau *built-in packages* (paket dari Java API) dan paket yang ditentukan pengguna atau programmer (*User-defined Packages*).

2.11.1 Built-In Packages

Java API adalah pustaka atau *library* kelas yang telah ditulis sebelumnya, yang gratis untuk digunakan, termasuk dalam lingkungan Java Development. Pustaka berisi komponen untuk mengelola input, pemrograman database, dan banyak lagi. Daftar lengkapnya dapat ditemukan di situs web Oracles: <https://docs.oracle.com/javase/8/docs/api/>. Pustaka dibagi menjadi paket dan kelas. Artinya kita dapat mengimpor satu kelas (bersama dengan metode dan atributnya), atau seluruh paket yang berisi semua kelas yang termasuk dalam paket yang ditentukan. Untuk menggunakan kelas atau paket dari pustaka atau API, kita perlu menggunakan kata kunci `import`, contoh:

```
import java.io.File;    // Impor satu kelas/class
import java.io.IOException; // Impor satu kelas/class
import java.util.*;    // Impor seluruh paket/package

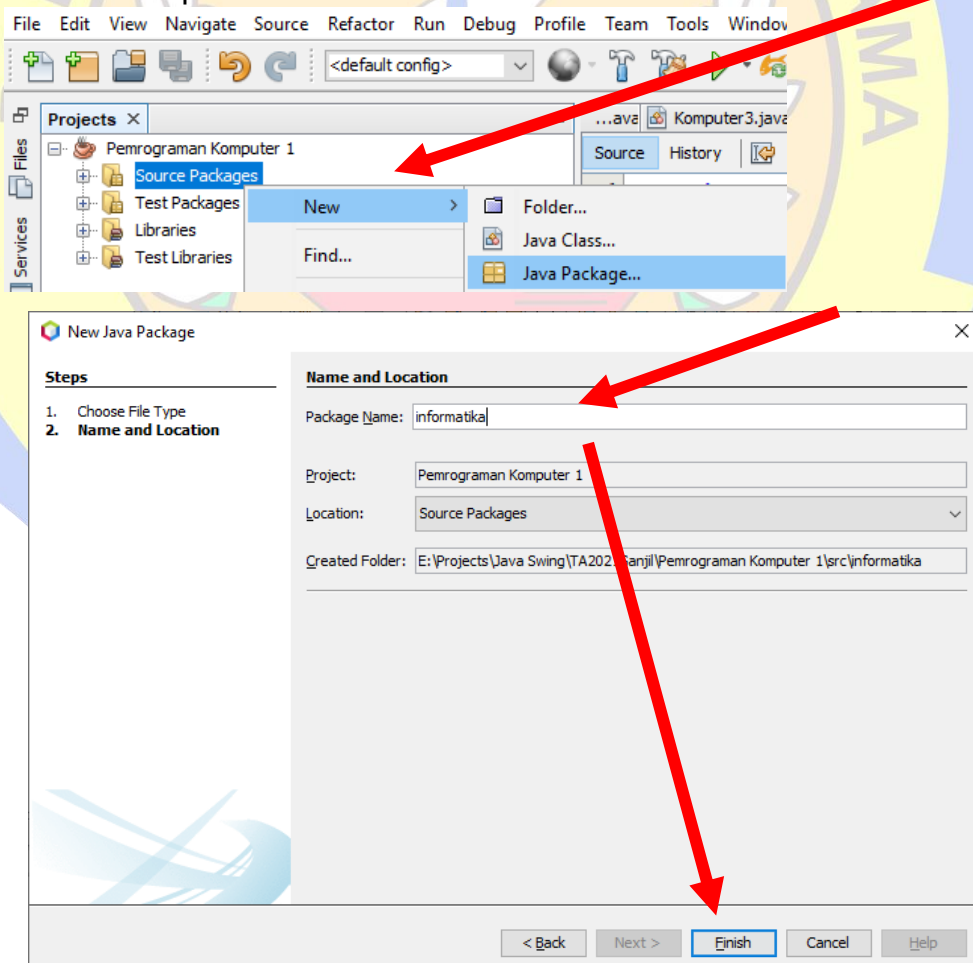
/**
 *
 * @author nishom
 */
public class BuiltIn {
    public static void main(String[] args) {
        try {
            Scanner s = new Scanner(System.in);
            System.out.print("Ketik nama file: ");
            String namaFile = s.next()+".txt";
            File f = new File(namaFile);
            f.createNewFile();
            System.out.println("File "+namaFile+" telah dibuat");
            System.out.println("Lokasi file: "+f.getAbsolutePath());
        } catch (IOException e) {
        }
    }
}
```

Kode `import java.io.File;` digunakan untuk mengimpor kelas `File` untuk digunakan dalam mengelola `file`. Kode `import java.io.IOException;` digunakan untuk mengetahui pengecualian pada kesalahan proses pada saat operasi input/output dijalankan. Sedangkan kode `import java.util.*;` digunakan untuk mengimpor semua `class` maupun `interface` yang ada di dalam paket `java.util`, pada kode di atas contoh kelas dalam paket `java.util` yang dipakai adalah kelas `Scanner`.

2.11.2 User-defined Packages

Untuk membuat paket kita sendiri, kita perlu memahami bahwa Java menggunakan direktori sistem file untuk menyimpannya. Sama seperti folder di komputer kita. Pembuatan paket baru dapat kita tambahkan dengan cara sebagai berikut:

- A. Klik kanan pada built-in package "Source Packages", kemudian pilih "New" → pilih "Java Package"
- B. Isi nama paket → klik tombol "Finish"



2.12 Practice

Pada bab ini, kita akan mempraktikkan pembuatan sebuah proyek sederhana yang mengimplementasikan seluruh konsep pemrograman berorientasi objek. Ikuti langkah-langkah berikut untuk memulai praktikum.

1. Buatlah sebuah project baru dengan nama "Pemrograman Komputer 1"

Ketentuan:

- ✓ Categories : Java With Ant
- ✓ Projects : Java Application
- ✓ Hilangkan tanda centang pada opsi "Create Main Class"

2. Buat package baru dengan nama "**practice.oop**". Pada praktikum ini, semua kelas dan interface akan kita buat dalam paket tersebut.

3. Buat *interface* dengan nama "FlyBehavior", kemudian ubah kode programnya menjadi seperti berikut:

```
public interface FlyBehavior {
    public void fly();
}
```

4. Buat *interface* dengan nama "QuackBehavior", kemudian ubah kode programnya menjadi seperti berikut:

```
public interface QuackBehavior {
    public void quack();
}
```

5. Buat *class* dengan nama "FlyWithWings", kemudian ubah kode programnya menjadi seperti berikut:

```
public class FlyWithWings implements FlyBehavior {
    @Override
    public void fly() {
        System.out.println("I'm flying!!");
    }
}
```

6. Buat *class* dengan nama "FlyNoWay", kemudian ubah kode programnya menjadi seperti berikut:

```
public class FlyNoWay implements FlyBehavior {
    @Override
    public void fly() {
        System.out.println("I can't fly");
    }
}
```

7. Buat *class* dengan nama “Quack”, kemudian ubah kode programnya menjadi seperti berikut:

```
public class Quack implements QuackBehavior {

    @Override
    public void quack() {
        System.out.println("Quack");
    }

}
```

8. Buat *class* dengan nama “Squeak”, kemudian ubah kode programnya menjadi seperti berikut:

```
public class Squeak implements QuackBehavior {

    @Override
    public void quack() {
        System.out.println("Squeak");
    }

}
```

9. Buat *class* dengan nama “MuteQuack”, kemudian ubah kode programnya menjadi seperti berikut:

```
public class MuteQuack implements QuackBehavior {

    @Override
    public void quack() {
        System.out.println("<< Silence >>");
    }

}
```

10. Buat *class* dengan nama “MuteQuack”, kemudian ubah kode programnya menjadi seperti berikut:

```
public abstract class Duck {
    FlyBehavior flyBehavior;
    QuackBehavior quackBehavior;

    public Duck() {
    }

    public void setFlyBehavior(FlyBehavior fb) {
        flyBehavior = fb;
    }
    public void setQuackBehavior(QuackBehavior qb) {
        quackBehavior = qb;
    }
    abstract void display();

    public void performFly() {
        flyBehavior.fly();
    }
    public void performQuack() {
        quackBehavior.quack();
    }
    public void swim() {
        System.out.println("All ducks float, even decoys!");
    }

}
```

11. Buat *class* dengan nama "MallardDuck", kemudian ubah kode programnya menjadi seperti berikut:

```
public class MallardDuck extends Duck {

    public MallardDuck() {

        quackBehavior = new Quack();
        flyBehavior = new FlyWithWings();

    }

    @Override
    public void display() {
        System.out.println("I'm a real Mallard duck");
    }

}
```

12. Buat *class* dengan nama "RedHeadDuck", kemudian ubah kode programnya menjadi seperti berikut:

```
public class RedHeadDuck extends Duck {

    public RedHeadDuck() {
        flyBehavior = new FlyWithWings();
        quackBehavior = new Quack();
    }

    @Override
    public void display() {
        System.out.println("I'm a real Red Headed duck");
    }

}
```

13. Buat *class* dengan nama "RubberDuck", kemudian ubah kode programnya menjadi seperti berikut:

```
public class RubberDuck extends Duck {

    public RubberDuck() {
        flyBehavior = new FlyNoWay();
        //quackBehavior = new Squeak();
        quackBehavior = () -> System.out.println("Squeak");
    }

    public RubberDuck(FlyBehavior flyBehavior, QuackBehavior quackBehavior) {
        this.flyBehavior = flyBehavior;
        this.quackBehavior = quackBehavior;
    }

    @Override
    public void display() {
        System.out.println("I'm a rubber duckie");
    }

}
```

14. Buat *class* dengan nama “DecoyDuck”, kemudian ubah kode programnya menjadi seperti berikut:

```
public class DecoyDuck extends Duck {

    public DecoyDuck() {
        setFlyBehavior(new FlyNoWay());
        setQuackBehavior(new MuteQuack());
    }

    @Override
    public void display() {
        System.out.println("I'm a duck Decoy");
    }
}
```

15. Buat *class* dengan nama “DecoyDuck”, kemudian ubah kode programnya menjadi seperti berikut:

```
public class ModelDuck extends Duck {

    public ModelDuck() {
        flyBehavior = new FlyNoWay();
        quackBehavior = new Quack();
    }

    @Override
    public void display() {
        System.out.println("I'm a model duck");
    }
}
```

16. Buat kelas “FlyRocketPowered”, kemudian ubah kode program menjadi seperti berikut:

```
public class FlyRocketPowered implements FlyBehavior {

    @Override
    public void fly() {
        System.out.println("I'm flying with a rocket");
    }
}
```

17. Buat kelas “DuckSimulator”, kemudian ubah kode program menjadi seperti berikut:

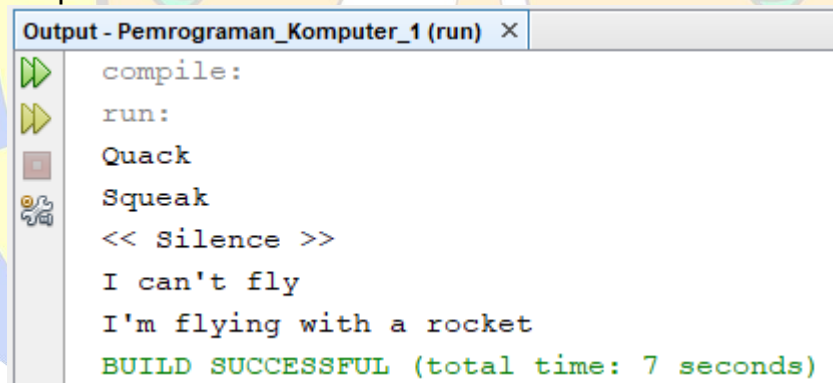

```

1  package practice.oop;
2
3  /**
4   *
5   * @author nishom
6   */
7  public class DuckSimulator {
8
9      public static void main(String[] args) {
10
11          MallardDuck mallard = new MallardDuck();
12          FlyBehavior cantFly = () -> System.out.println("I can't fly");
13          QuackBehavior squeak = () -> System.out.println("Squeak");
14          RubberDuck rubberDuckie = new RubberDuck(cantFly, squeak);
15          DecoyDuck decoy = new DecoyDuck();
16
17          Duck model = new ModelDuck();
18
19          mallard.performQuack();
20          rubberDuckie.performQuack();
21          decoy.performQuack();
22
23          model.performFly();
24          model.setFlyBehavior(new FlyRocketPowered());
25          model.performFly();
26      }
27  }

```

18. Buat kelas "DuckSimulator" sebagai main class, kemudian jalankan project.

Output:



```

Output - Pemrograman_Komputer_1 (run) X
compile:
run:
Quack
Squeak
<< Silence >>
I can't fly
I'm flying with a rocket
BUILD SUCCESSFUL (total time: 7 seconds)

```

2.13 Exercise

Modifikasi kelas `DuckSimulator` dengan menambahkan kelas `Scanner` agar pengguna dapat memilih `Duck` dan mengetahui sifat atau behavior dari masing-masing duck yang dipilih oleh pengguna.