**Pengantar Java Dasar**

Demi menaklukkan medan pemrograman dengan Java, kita perlu memahami terlebih dahulu seperti apakah Java itu. Bagaimana strukturnya? Bagaimana menggunakan fungsi *basic programming* yang ia miliki? Oleh karena itu kita perlu tahu pemrograman dasar Java yang dikemas dalam materi berikut:

1. **Struktur Dasar**  
   Ini adalah materi paling mendasar dari Java yang wajib Anda kuasai.
2. **Tipe Data**  
   Setiap tipe data digunakan untuk menampung konten yang berbeda. Delapan (8) tipe data dalam Java yang akan kita pelajari adalah int, float, char, long, double, boolean, byte dan short.
3. **String**Selain data primitif, *string* merupakan pokok dasar yang perlu Anda kuasai.
4. **Operator**Operator meliputi operasi aritmatika dasar seperti penambahan, pengurangan, perkalian, dan pembagian. Selain aritmatika dasar, terdapat juga operator-operator lain seperti *assignment*,*unary, equality and relational,*dan*condition.*
5. **Fungsi Input Output**  
   Proses *input* dan *output* merupakan hal yang penting agar kita dapat berinteraksi dengan program. Di sini kita akan belajar beberapa komponen yang dapat menerima *input* dari *user* dan komponen yang dapat menampilkan data ke *user*.
6. **Array**Array adalah kumpulan data yang dimasukkan ke dalam satu objek. Di materi ini kita akan belajar tentang pembuatan *array* dan aplikasinya di dalam suatu program.

# Struktur Dasar

## Package

Kita telah menginstal tools yang diperlukan dan membuat program sederhana yang menampilkan '**Hello, World!'** di bagian Pendahuluan. Kini kita akan mempelajari struktur dasar dari program Java. Setiap class di program Java diletakkan dalam sebuah package, yakni mekanisme penempatan/penamaan class agar lebih terstruktur atau modular.

Mekanisme package sangat diperlukan. Salah satunya untuk membedakan class dengan nama yang sama. Misalnya institusi Apache membuat class StringUtils. Begitu juga  dengan institusi Spring yang membuat class StringUtils. Lalu bagaimana kita membedakan keduanya jika kita ingin menggunakan kedua library tersebut secara bersamaan? Karena pada praktiknya hampir setiap aplikasi Java dibuat dengan library dari eksternal. Yup, coding Java sama dengan bermain di dunia open-source.

### Langkah penamaan package:

* Umumnya menggunakan nama domain institusi (dengan penulisan di balik) pembuat program tersebut.
* Sertakan nama program/aplikasi. Ikuti dengan nama modul-modulnya.
* Beri pemisah dengan tanda titik.

Anda tidak perlu bingung, berikut contohnya:

|  |  |
| --- | --- |
| Nama domain institusi | **dicoding.com** |
| Nama aplikasi | **Java Fundamental** |
| Nama modul | **Basic** |
| Maka nama package dapat kita tulis seperti ini | com.dicoding.javafundamental.basic |

Perhatikan tabel di atas, namapackage harus menggunakan **huruf kecil tanpa spasi.** Package akan sinkron dengan directory dari berkas (file) .java (source-code) dan juga hasil compile-nya yaitu file .class.

### Codelab Struktur Dasar

Di bagian pendahuluan Anda sudah membuat program HelloJava dengan sebuah class Main.Java, namun jika dilihat kembali Anda tidak akan melihat informasi package di sana. Dalam kasus ini program HelloJava mengunakan default package, di mana hal tersebut adalah cara yang sangat tidak disarankan. Oleh karena itu mari kita mulai koding dengan membuat kelas Main dalam package.

1. Buka aplikasi IntelliJ.
2. Setelah mengetahui beberapa penjelasan tersebut, sekarang kita pilih **Create New Project.**
3. Pastikan pilih **Java** di panel kiri lalu klik tombol **Next**.
4. Kemudian, masukkanlah **Project name**dengan StrukturDasar. Untuk lokasi proyeknya, Anda bisa memilih lokasi yang diinginkan. Kemudian **Klik** tombol **Finish**.  
   **Catatan**: Jika ada popup-window ***Tip of the Day***, silakan lihat untuk membantu Anda beradaptasi dengan aplikasi IntelliJ.  
   Tutuplah jendela ***Tip of the Day***untuk melanjutkan ke langkah berikutnya.
5. Buat package baru dengan cara sebagai berikut. Di panel proyek sebelah kiri, **klik kanan** directory **src**. Kemudian pilih **New** - **Package**.  
   Kemudian masukkan **package name**-nya dengan com.dicoding.javafundamental.basic.
6. Selanjutnya buat class baru di dalam package yang baru saja Anda buat dengan cara, **klik kanan**di packagetersebut dan pilihlah **New** - **Java Class**.  
   Kemudian masukan nama class tersebut dengan Main dan klik **OK**.
7. Perhatikan posisi dari class  Main yang Anda buat sebelumnya. Class tersebut akan berada di dalam directory yang sama dengan nama package-nya.
8. Tambahkan kode sebagai berikut di class Main.
   1. package com.dicoding.javafundamental.basic;
   3. public class Main {
   4. **public static void main(String[] args) {**
   5. **System.out.println("Hello world!");**
   6. **}**
   7. }
9. Jika Anda perhatikan terdapat kode package yang merupakan lokasi dari kelas Main. Kelas Main berada di dalam package com.dicoding.javafundamental.basic.
10. Lalu jalankanlah class Main.java dengan **klik** salah satu tombol  
    di sebelah kiri panel Editor. Lalu pilih **Run** ‘**Main.main()**’.
11. Hasilnya akan muncul di panel di bawah:

|  |
| --- |
| Hello world! |

Apa yang sudah kita lakukan sebenarnya sama saja dengan bagian Pendahuluan. Bedanya, Anda meletakkan class Main di dalam package com.dicoding.javafundamental.basic. Ayo kita lanjut!

Ketika setiap class diletakkan di dalam sebuah package, bagaimana cara memanggil class tertentu di antara class lain yang berbeda package-nya? Caranya adalah dengan memanggil nama package secara lengkap, kemudian diikuti nama class-nya. Hal ini biasa disebut dengan istilah Fully Qualified Name. Misalnya, fully qualified name dari class  Main yang baru saja kita buat adalah:

1. com.dicoding.javafundamental.basic.Main

## Import

Import digunakan untuk menyederhanakan pemanggilan class yang berbeda package. Alhasil, Anda tak perlu menyebutkan fully qualified name dari class yang ingin digunakan. Dari pengertian ini juga tersirat kita tak perlu juga menyebutkan fully qualified name jika dalam package yang sama.

Khusus untuk class dari package java.lang, ia tidak perlu dipanggil menggunakan fully qualified name. Artinya tidak perlu menggunakan import saat memanggilnya, misalnya class java.lang.System yang kita gunakan untuk print “Hello world!”. Perhatikan bahwa kode di bawah ini:

1. System.out.println("Hello world!");

Pada kode di atas, kita hanya memanggil class dengan nama class System tanpa nama package-nya.

Bagaimana kalau kita import beberapa class dari package yang sama? Kita bisa menggunakan wildcard (simbol \*) untuk menggantikan nama class-nya. Artinya, Java akan otomatis mengenali seluruh class dari package tersebut.

### Codelab Import

OK untuk lebih jelasnya, mari kita langsung koding saja.

1. Buka kembali project **StrukturDasar.**
2. Buatlah kembali package dengan cara **klik kanan** directory **src,**kemudian pilih **New** - **Package.**Masukkanlah nama package-nya dengan com.dicoding.javafundamental.basic.musik.  
   Maka tampilan package saat ini menjadi seperti ini:
3. Selanjutnya buatlah classGitar di dalam packagemusic.  
   Tambahkan kode berikut di class Gitar:
   1. public class Gitar {
   2. **public static void bunyi(){**
   3. **System.out.println("jrenggg..");**
   4. **}**
   5. }
4. Buka kembali classMain, tambahkan kode seperti di bawah ini:
5. Perhatikan **IntelliJ** akan memberi tanda eror berwarna merah karena classGitar tidak dikenali. Untuk itu kita perlu importclassGitar agar dikenali. Pindahkan kursor ke baris kode yang ditandai dengan error tersebut, lalu tekan **alt + Enter(Windows)**atau**option + return(MacOS).**  
   Otomatis **IntelliJ**akan menambahkan baris kode seperti berikut:
   1. **import com.dicoding.javafundamental.basic.musik.Gitar;**
6. Lalu jalankanlah class Main dengan klik tombol run pastikan tidak ada eror.  
   Hasilnya akan jadi seperti ini:

|  |
| --- |
| Hello world! jrenggg.. |

1. Selanjutnya buatlah kembali package baru dengan cara **klik kanan** directory **src,**kemudian pilih **New** - **Package.**Masukkanlah nama package-nya dengan: com.dicoding.javafundamental.basic.kendaraan.  
   Maka struktur package kali ini akan menjadi seperti ini:
2. Selanjutnya buat class baru dengan nama  Mobil, Motor, dan Kereta di dalam package tersebut.  
   Tambahkan baris kode berikut di masing-masing class tersebut.  
   Pertama buka classMobildan ubahlah kode di dalamnya menjadi seperti ini.
   1. public class Mobil {
   2. **public static void jumlahBan(){**
   3. **System.out.println("Ban mobil 4");**
   4. **}**
   5. }

Selanjutnya bukalah classMotordan ubahlah kode di dalamnya:

* 1. public class Motor {
  2. **public static void jumlahBan() {**
  3. **System.out.println("Ban motor 2");**
  4. **}**
  5. }

Terakhir bukalah classKeretadan ubah juga kode di dalamnya menjadi seperti ini:

* 1. public class Kereta {
  2. **public static void jumlahBan() {**
  3. **System.out.println("Ban kereta banyak!");**
  4. **}**
  5. }

1. Bukalah classMain lalu ketik (jangan copy-paste) kode berikut.
   1. **Mobil.jumlahBan();**
2. Ketika kode di atas mulai diketik sebagian, maka IntelliJ akan memberi saran kode (dikenal dengan istilah auto-complete) seperti gambar di bawah. Tekan Enter agar IntelliJ melengkapi baris kode.  
   Dengan bantuan auto complete, Anda tidak perlu melakukan import secara manual.
3. Lengkapi classMaindengan kode sebagai berikut :
   1. Motor.jumlahBan();
   2. Kereta.jumlahBan();
4. Baris import dapat Anda sederhanakan yang awalnya seperti ini:
   1. import com.dicoding.javafundamental.basic.kendaraan.Kereta;
   2. import com.dicoding.javafundamental.basic.kendaraan.Mobil;
   3. import com.dicoding.javafundamental.basic.kendaraan.Motor;
   4. import com.dicoding.javafundamental.basic.musik.Gitar;

Menjadi:

* 1. **import com.dicoding.javafundamental.basic.kendaraan.\*;**
  2. import com.dicoding.javafundamental.basic.musik.Gitar;

Maka kelas Mainmenjadi seperti ini:

* 1. package com.dicoding.javafundamental.basic;
  3. **import com.dicoding.javafundamental.basic.kendaraan.\*;**
  4. **import com.dicoding.javafundamental.basic.musik.Gitar;**
  6. public class Main {
  7. public static void main(String[] args) {
  8. System.out.println("Hello world!");
  10. Gitar.bunyi();
  12. **Mobil.jumlahBan();**
  13. **Motor.jumlahBan();**
  14. **Kereta.jumlahBan();**
  15. }
  16. }

**Catatan:**  
Penggunaan wildcard (\*) kadang tidak disarankan jika class di dalam package tersebut terlalu banyak. Sebabnya, ia memperlama waktu untuk pencarian class tersebut. Untuk itu kita serahkan ke IntelliJ mengatur import yang optimal. Pilih menu **Code** | **Optimize Imports**. Baris import akan kembali ke sebelumnya tanpa menggunakan wildcard (\*).

1. Lalu jalankanclass Main dengan **klik** tombol   
   dan pastikan tidak ada error di dalamnya.
2. Maka hasilnya jadi seperti ini:

|  |
| --- |
| Hello wordl! jrenggg.. Ban mobil 4 Ban motor 2 Ban kereta banyak! |

## Classpath

Classpath adalah mekanisme di Java untuk menemukan class lain. Biasanya class lain tersebut berasal dari library yang berbeda atau bahkan JDK itu sendiri (kita sudah memakai class System). Jika Java tidak bisa menemukan class yang kita panggil melalui kode kita, akan terjadi error ClassNotFoundException atau NoClassDefFoundError. Error ini biasa terjadi bagi pemula Java karena biasanya belum paham konsep classpath atau ada kesalahan pengaturan classpath.

Sekarang tenang ya kalau ketemu error serupa. Kita kan sudah tahu bahwa terduga adalah problem di classpath. Jadi paham juga deh, dari mana kita mesti melakukan troubleshooting.

### Codelab Classpath

1. Buka kembali proyek StrukturDasar. Buat directory baru dengan cara **klik kanan** di proyek StrukturDasar seperti gambar di bawah:  
   Selanjutnya masukkan **directory name**dengan lib.  
   Maka akan jadi seperti ini:
2. Unduh library apache**common-lang**dari tautan berikut:
   * <https://repo1.maven.org/maven2/org/apache/commons/commons-lang3/3.9/commons-lang3-3.9.jar>
3. Salin berkas .jar tersebut ke directory **lib**.  
   Maka akan jadi seperti ini:
4. Klik kanan di commons-lang3-3.9.jar untuk menjadikan file jar menjadi **library,** pilihlah **Add as Library...**.  
   Setelah itu biarkan namanya menjadi default dan tekan **OK**.  
   Maka hasilnya akan jadi seperti ini:  
   Perhatikan sekarang ada tanda panah di kiri sehingga kita bisa browse isi berkas jar tersebut. Jika tanda tersebut tidak ada, maka proses penambahan library yang Anda lakukan gagal.
5. Sekarang berkas jar sudah menjadi library di projectStrukturDasar.
6. Tambahkan baris kode berikut di classMain. Sebaiknya ketik (koding) saja supaya get a grip tentang fitur auto -completeIntelliJ ini.
   * Date today = new Date();
   * System.out.println("Hari ini = " + today);
   * Date tomorrow = DateUtils.addDays(today, 1);
   * System.out.println("Besok = " + tomorrow);

Sehingga kelas Main.java menjadi seperti ini:

* + package com.dicoding.javafundamental.basic;
  + import com.dicoding.javafundamental.basic.kendaraan.\*;
  + import com.dicoding.javafundamental.basic.musik.Gitar;
  + **import org.apache.commons.lang3.time.DateUtils;**
  + import java.util.Date;
  + public class Main {
  + public static void main(String[] args) {
  + System.out.println("Hello world!");
  + Gitar.bunyi();
  + Mobil.jumlahBan();
  + Motor.jumlahBan();
  + Kereta.jumlahBan();
  + **Date today = new Date();**
  + **System.out.println("Hari ini = " + today);**
  + **Date tomorrow = DateUtils.addDays(today, 1);**
  + **System.out.println("Besok = " + tomorrow);**
  + }
  + }

1. Jalankan class Main dengan **klik tombol**, pastikan tidak ada error.  
   Maka hasilnya akan menjadi seperti ini:

|  |
| --- |
| Hello wordl! jrenggg.. Ban mobil 4 Ban motor 2 Ban kereta banyak! Hari ini = Tue Jul 16 18:12:44 WIB 2019 Besok = Wed Jul 17 18:12:44 WIB 2019 |

Praktik yang kita lakukan barusan adalah menambahkan library (file jar) secara manual ke proyek. Cara ini sebenarnya kurang efektif jika dilakukan untuk proyek besar. Bayangkan jika kita menggunakan library A yang bergantung (dependencies) ke library B, C, D, lalu library B bergantung ke library X dan Y. Cukup bikin repot kan? Solusinya, gunakan tools seperti **maven** atau **gradle**. Pembahasan maven dan gradle di luar cakupan dari class ini. Saran kami, bacalah referensi yang ada di bawah karena kedua tools tersebut sudah menjadi standar dalam development Java.

* <http://www.baeldung.com/java-classnotfoundexception-and-noclassdeffounderror>
* [https://maven.apache.org](https://maven.apache.org/)
* [https://gradle.org](https://gradle.org/)
* <https://spring.io/guides/gs/maven>
* <https://spring.io/guides/gs/gradle>

## Pemaketan

Setiap kode Java yang ditulis akan dikompilasi menjadi bytecode (file dengan extension .class) lalu pada akhirnya akan dipaketkan untuk didistribusikan. Bentuk pemaketan yang sering digunakan adalah bentuk **JAR** (Java ARchive), ada juga bentuk lain misal **WAR** (Web ARchive) dan **EAR** (Enterprise ARchive). Dari nama pemaketan ini bisa ditebak sebenarnya hanya berkas archive atau berkas zip. Isinya bisa diintip menggunakan winzip, winrar atau aplikasi sejenis.

 Di dalam hasil pemaketan tersebut ada berkas metadata yang menjelaskan isi berkas JAR. File tersebut adalah manifest.mf yang diletakkan di directory META-INF.

### Codelab Pemaketan

Mari kita coba paketkan project **StrukturDasar** dalam bentuk JAR.

1. Buka kembali IntelliJ. **File** | **Project Structure**|**Artifacts**. Klik tombol **+** lalu pilih **JAR** | **From modules with dependencies**.  
   Pilih **Main Class**, klik tombol   
   atau lalu ketik Main di popup. Kemudia klik **OK** untuk melanjutkan.  
   Maka akan jadi seperti gambar di bawah. Klik **OK** untuk melanjutkan.  
   Hasil akhirnya akan menjadi seperti gambar di bawah. Klik **OK** untuk melanjutkan.
2. Selanjutnya pilihlah menu **Build**dan pilih **Build Artifacts**.  
   Pilihlah **Build** untuk memulai membangun **Artifacts**.  
   Perhatikan, akan terbentuk StrukturDasar.jar di directory artifacts yang isinya sama persis seperti directory production.
3. Perhatikan dan buka berkasMANIFEST.MF yang dibuat otomatis oleh proses **Build**.
   1. Manifest-Version: 1.0
   2. Main-Class: com.dicoding.javafundamental.basic.Main
4. Buka panel **Terminal** di bagian bawah IDE (sesuaikan dengan versi IDE Anda):  
   Kemudian ketikkan**command**atau**perintah** berikut dan lihat hasilnya.
   1. java -jar **[lokasi .jar]**

Untuk lokasi .jar, Anda dapat melihat letak berkas .jar yang sudah Anda generate. Contohnya seperti ini:  
Maka dapat disimpulkan lokasinya adalah **out/artifacts/StrukturDasar\_jar/StrukturDasar.jar**.  
  
Maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Hello wordl! jrenggg.. Ban mobil 4 Ban motor 2 Ban kereta banyak! Hari ini = Tue Jul 16 18:12:44 WIB 2019 Besok = Wed Jul 17 18:12:44 WIB 2019 |

Selesai sudah materi struktur dasar. Sampai ketemu di materi selanjutnya, happy coding!

# Tipe Data

Java merupakan bahasa pemrograman yang strongly typed, maka kita tidak bisa mengabaikan tipe data. Kita harus tahu data seperti apa yang disimpan ke dalam variabel. Selain itu, Java juga bersifat statically typed, yang artinya setiap variabel harus dideklarasikan terlebih dahulu sebelum digunakan.

Setiap bahasa pemrograman memiliki aturan tersendiri dalam menuliskan sebuah variabel. Begitupun pada Java. Setiap variabel harus dideklarasikan terlebih dahulu tipe datanya. Tipe data digunakan untuk menentukan jenis nilai yang akan ditampung pada variabel tersebut. Terdapat 2 (dua) jenis tipe data pada Java, yaitu tipe data primitive/primitif dan reference/referensi. Mari kita pelajari masing-masing komponen-komponen di dalam keduanya.

## Tipe Data Primitif

Tipe data primitif adalah tipe data standar yang tidak diturunkan dari objek manapun. Tipe data primitif telah ditentukan dengan kata kuncinya masing-masing. Terdapat 8 (delapan) tipe data primitif yang Java dukung, antara lain:

1. **Byte**  
   Tipe data integer **8 bit** yang digunakan untuk menampung angka dalam range yang kecil. Nilai minimum dan maksimum dari tipe data ini adalah **-128**, dan **127**. Tipe data byte memiliki nilai default **0**. Ia digunakan untuk menghemat ruang dalam array yang besar, terutama pada bilangan bulat, karena 1 (satu) byte 4 (empat) kali lebih kecil dari tipe data **int**.
   1. byte value = 10;
   2. byte anotherValue = -10;
2. **Short**  
   Merupakan tipe data integer **16 bit** yang digunakan untuk menampung angka dalam range menengah, yaitu antara **-32,768** sampai **32,767**. Sama seperti tipe sebelumnya, nilai default-nya **0**. Ia bisa digunakan untuk menghemat memori seperti tipe data byte namun hanya 2 (dua) kali lebih kecil dari tipe data **int.**
   1. short value = 15000;
   2. short anotherValue = -20000;
3. **Int**  
   Merupakan tipe data integer **32 bit** yang digunakan untuk menyimpan angka dalam range cukup besar, yakni antara **-2,147,483,648** sampai **2,147,483,647** dengan nilai default **0**. Jika kita tidak memperhatikan penggunaan memori, tipe data inilah yang biasanya dipakai.
   1. int value = 150000;
   2. int anotherValue = -200000;
4. **Long**  
   Merupakan tipe data integer yang lebih besar jika dibandingkan dengan tipe data **int.**Ukurannya **64 bit**dan bisa digunakan untuk menyimpan angka dengan range antara **-9,223,372,036,854,775,808** sampai **9,223,372,036,854,775,807**. Tipe data **long**memiliki nilai default **0L**.
   1. long value = 150000L;
   2. long anotherValue = -200000L;
5. **Float**  
   Merupakan sebuah tipe data yang bisa digunakan untuk menampung angka desimal. Nilai default-nya **0.0f**.
   1. float value = 3.5f;
6. **Double**Merupakan sebuah tipe data yang mirip seperti tipe data float, namun memiliki kapasitas yang lebih besar. Nilai default-nya **0.0d**.
   1. double value = 5.0;
7. **Boolean**Merupakan sebuah tipe data yang memiliki 2 (dua) macam nilai, yaitu **true**dan **false**. Nilai default-nya **false**.
   1. boolean value = true;
   2. boolean anotherValue = false;
8. **Char**Merupakan sebuah tipe data yang bisa digunakan untuk menampung karakter. Nilai karakter tersebut dibungkus di dalam tanda ***' '***.
   1. char item = 'A'

## Tipe Data Reference

Tipe data reference merupakan sebuah tipe data yang merujuk ke sebuah objek atau instance dari sebuah class. Salah satu tipe data yang termasuk ke dalam tipe data reference adalah **string**. Tipe data **string** menunjuk ke instance dari class java.lang.String.

Selain itu, kita bisa membuat tipe data sendiri. Sebagai contoh, ketika kita memiliki sebuah kelas User, kita bisa membuat variabel baru dengan tipe data class Usertersebut. Kita perlu membuat instance dari kelas tersebut:

1. User user = new User();

Dengan begitu kita bisa mengakses semua atribut dan fungsi yang ada pada kelas User menggunakan variabel user.

# String

Di dalam bahasa pemrograman Java, string merupakan sebuah tipe **data reference**. Tipe data ini diperlakukan sebagai sebuah obyek. String adalah kumpulan beberapa karakter (char). String sendiri sebenarnya merupakan sebuah class yang terdapat dalam library Java dan digunakan untuk memanipulasi karakter. Contoh sintaks untuk mendeklarasikan sebuah string:

1. String greeting = "Hello World!";
2. System.out.println(greeting);

Jika kode di atas dijalankan, maka konsol akan menampilkan teks HelloWorld**!**

Di dalam kelas Stringterdapat beberapa constructoryang memungkinkan kita untuk memberikan nilai awal untuk string dari sumber yang berbeda. Sebagai contoh, misalnya kita ingin membuat variabel String dari sebuah array character.

1. char[] dicodingChars = { 'd', 'i', 'c', 'o', 'd', 'i', 'n', 'g' };
2. String dicodingString = new String(dicodingChars);
3. System.out.println(dicodingString);

Dengan kode di atas, kita telah membuat sebuah array character yang berisi beberapa karakter.

Kemudian array character tersebut kita gunakan sebagai parameter untuk membuat Stringbaru. Kode tersebut akan menghasilkan teks "**dicoding**".

Perlu diperhatikan bahwa kelas String bersifat final. Alhasil, kita tidak bisa mengubah nilai dari objek yang sudah kita buat. Jika kita ingin mengubah nilai tersebut, kita membutuhkan StringBuffer dan StringBuilder.

## Mengetahui Panjang String

Kelas String memiliki sebuah method untuk mengetahui panjang dari sebuah string, yakni methodlength(). Fungsi tersebut akan mengembalikan/menghasilkan sejumlah karakter dari string, contohnya:

1. String dicoding = "dicoding";
2. int length = dicoding.length();
3. System.out.println(length);

Fungsilength()pada kode di atas akan menghitung jumlah karakter dari variabel dicoding. Jika dijalankan, maka konsol akan menampilkan angka 8.

## Mengambil Karakter Dari Sebuah String

Kita juga bisa mengambil sebuah karakter secara spesifik dari sebuah String dengan menggunakan method charAt(int index) yang sudah tersedia dalam kelas String. Misalnya saat ingin mengambil sebuah karakter dari teks dicoding, kita bisa menggunakan kode berikut:

1. String dicoding = "dicoding";
2. char result = dicoding.charAt(7);
3. System.out.println(result);

Maka jika Anda menjalankan kode di atas, konsol akan menampilkan karakter urutan **ke-8**dari teks tersebut, yaitu "**g**".

Selain method length()dan charAt(int index), masih banyak method lain yang disediakan oleh kelas Stringyang bisa mempermudah kita  memanipulasi sebuah teks. Berikut adalah kumpulan beberapa method yang sering digunakan:

|  |  |  |
| --- | --- | --- |
| **No.** | **Nama** | **Deskripsi** |
| 1 | **length()** | Digunakan untuk mengetahui panjang atau jumlah karakter dalam string. |
| 2 | **charAt(int index)** | Digunakan untuk mengambil sebuah karakter berdasarkan indeks tertentu. |
| 3 | **format(String format, Object… args)** | Digunakan untuk memformat sebuah string. |
| 4 | **substring(int beginIndex)** | Mengembalikan/menghasilkan substring berdasarkan indeks yang diberikan. |
| 5 | **contains(CharSequence s)** | Mengembalikan/menghasilkan nilai **true** atau **false** setelah mencocokkan karakter. |
| 6 | **equals(Object object)** | Memeriksa apakah nilai objek sama dengan nilai string. |
| 7 | **isEmpty()** | Memeriksa apakah sebuah string itu kosong atau tidak. |
| 8 | **concat(String s)** | Mengkonsolidasikan sebuah string. |
| 9 | **replace(char a, char b)** | Mengganti suatu karakter di dalam string. |
| 10 | **indexOf(String a)** | Mengetahui indeks dari sebuah substring. |
| 11 | **toLowerCase()** | Mengubah format stringmenjadi huruf kecil semua. |
| 12 | **toUpperCase()** | Mengubah format stringmenjadi huruf kapital semua. |
| 13 | **trim()** | Menghapus spasi awal dan akhir dari string. |
| 14 | **valueOf(int value)** | Mengkonversi tipe yang diberikan menjadi sebuah string. |
| 15 | **compareTo()** | Membandingkan dua nilai |

# Operator

Kita sudah belajar tentang bagaimana suatu variabel dideklarasikan dan diinisiasi. Selanjutnya kita akan berusaha memahami aksi yang akan diberlakukan terhadap variabel tersebut, misalnya mengubah, menambah, membagi variabel atau aksi lainnya. Untuk itu, kita akan menggunakan operator.

Operator dalam Java merupakan suatu simbol yang digunakan untuk melakukan suatu operasi tertentu (memanipulasi, mengolah) satu atau lebih variabel. Variabel yang dioperasikan disebut sebagai operand. Bahasa Java memiliki banyak operator yang dapat digunakan dan bisa dikelompokkan menjadi beberapa kategori, seperti :

* **Operasi Assignment**
* **Operasi Arithmetic**
* **Operasi Unary**
* **Operasi Equality and Relational**
* **Operasi Conditional**

Selengkapnya, silakan merujuk ke <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>.

## Operasi Assignment

Operasi Assigment adalah salah satu operator yang akan sering ditemui dalam pemrograman Java. Assignment Variable (penempatan/pengalamatan variabel) merupakan operator yang cukup simpel dan diwakilkan dengan simbol operator ‘**=**’. Operator assigment memberikan nilai pada sebuah variabel yang ada di sebelah kiri (left operand) dengan nilai yang ada di sebelah kanan (right operand).

### Codelab Operasi Assignment

Langsung saja kita praktekan melalui kode di bawah ini:

1. Buatlah proyek baru dengan nama **Operator** dengan nama package **com.dicoding.javafundamental.operator** di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama OperatorAssignment:
3. Tambahkan kode berikut di dalam kelas OperatorAssignment:
   1. package com.dicoding.javafundamental.operator;
   3. public class OperatorAssignment {
   5. **public static void main(String[] args) {**
   6. **int value = 2;**
   7. **int anotherValue = 19;**
   8. **System.out.println("Data pada value adalah ->> " + value);**
   9. **System.out.println("Data pada anotherValue adalah ->> " + anotherValue);**
   10. **}**
   11. }
4. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Data pada value adalah ->> 2 Data pada anotherValue adalah ->> 19 |

### Bedah Code Operasi Assignment

Untuk penjelasannya, lihat bagian kode di bawah ini:

1. int value = 2;

Potongan kode di atasvaluemerupakan **left operand.**Angka 2 merupakan sebuah nilai integer (**right operand**). Sesuai dengan penjelasan operasi assignment, nilai value akan menjadi nilai integer 2. Operasi ini bisa juga disebut sebagai copy, karena nilai **left operand**akan mengkopi nilai **right operand**.

## Operasi Aritmatika

Java memiliki banyak operator yang digunakan untuk membantu perhitungan Aritmatika. Operator Aritmatika pada Java antara lain :

|  |  |
| --- | --- |
| **Operator** | **Deskripsi** |
| + | Operator penambahan |
| - | Operator pengurangan |
| \* | Operator pengalian |
| / | Operator pembagian |
| % | Operator sisa hasil |

Pastinya kita sudah familiar dengan simbol-simbol operator di atas pada pelajaran Matematika dasar. Mungkin hanya satu simbol yang terlihat asing dan baru untuk Anda, yakni ‘**%**’ (persen) sebagai operator untuk menghitung sisa hasil bagi atau dapat disebut mod / modulus. Modulus adalah sebuah operasi yang menghasilkan sisa dari pembagian dari suatu bilangan terhadap bilangan lainnnya.

### Codelab Operasi Aritmatika

Langsung saja kita praktekan melalui kode di bawah ini:

1. Bukalah proyek **Operator** dan tambahkan kelas baru didalamnya dengan nama OperatorAritmatika.
2. Tambahkan kode di bawah ini ke dalam kelas OperatorAritmatika:
   1. package com.dicoding.javafundamental.operator;
   3. public class OperatorAritmatika {
   5. **public static void main(String[] args) {**
   6. **System.out.println("Operasi Penjumlahan");**
   7. **int hasilPenjumlahan = 5 + 1;**
   8. **System.out.println("Hasil 5 + 1 = " + hasilPenjumlahan);**
   9. **System.out.println();**
   10. **System.out.println("Operasi Pengurangan");**
   11. **int hasilPengurangan = 4 - 1;**
   12. **System.out.println("Hasil 4 - 1 = " + hasilPengurangan);**
   13. **System.out.println();**
   14. **System.out.println("Operasi Pengalian");**
   15. **int hasilPengalian = 5 \* 5;**
   16. **System.out.println("Hasil 5 \* 5 = " + hasilPengalian);**
   17. **System.out.println();**
   18. **System.out.println("Operasi Pembagian");**
   19. **int hasilPembagian = 20 / 2;**
   20. **System.out.println("Hasil 20 / 2 = " + hasilPembagian);**
   21. **System.out.println();**
   22. **System.out.println("Operasi Habis bagi");**
   23. **int hasilSisa = 8 % 2;**
   24. **System.out.println("Hasil 8 % 2 = " + hasilSisa);**
   25. **int hasilSisaLain = 9 % 2;**
   26. **System.out.println("Hasil 9 % 2 = " + hasilSisaLain);**
   27. **System.out.println();**
   28. **}**
   29. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Operasi Penjumlahan Hasil 5 + 1 = 6  Operasi Pengurangan Hasil 4 - 1 = 3  Operasi Pengalian Hasil 5 \* 5 = 25  Operasi Pembagian Hasil 20 / 2 = 10  Operasi Habis bagi Hasil 8 % 2 = 0 Hasil 9 % 2 = 1 |

### Bedah Code Operasi Aritmatika

Kode di atas merupakan implementasi simbol-simbol Aritmatika yang didukung oleh Java. Seperti pada kode “int hasilPenjumlahan = 5 + 1”, tentunya ini cukup mudah dipahami bagi kita yang sudah mengenal matematika dasar.

Selanjutnya adalah implementasi dari operasi hasil bagi atau yang dipanggil dengan Modulus (mod).

1. int hasilSisa = 8 % 2;
2. int hasilSisaLain = 9 % 2;

Modulus yang berarti habis dibagi, bila dicetak akan menghasilkan keluaran berikut ini:

1. Operasi Habis bagi
2. Hasil 8 % 2 = 0
3. Hasil 9 % 2 = 1

Dari output di atas, dapat disimpulkan bahwa nilai variabel **8** **tidak bersisa** bila dibagi dengan variabel 2. Sementara itu, nilai variabel **9** jika dibagi dengan variabel yang sama (2), **menyisakan sisa** hasil bagi 1.

## Operasi Unary

Merupakan operator yang melakukan berbagai operasi Aritmatik yang hanya melibatkan satu operand. Operasi-operasi tersebut dapat berupa penambahan, pengurangan atau membalik suatu nilai dari bolean. Dalam bahasa pemrograman Java, berikut inilah yang termasuk Operator Unary :

|  |  |
| --- | --- |
| **Operator** | **Deskripsi** |
| + | Operator Unary positif |
| - | Operator Unary negatif |
| ++ | Operator penambahan; peningkatan nilai sebesar 1 point |
| -- | Operator pengurangan,; pengurangan nilai sebesar 1 point |
| ! | Operator komplemet logika, operator; membalikan sebuah nilai bolean |

### Codelab Operasi Unary

Mari kita coba kode di bawah ini untuk melihat bagaimana implementasi Operator di atas.

1. Bukalah proyek **Operator** dan tambahkan kelas baru didalamnya dengan nama OperatorUnary.
2. Tambahkan kode di bawah ini ke dalam kelas OperatorUnary:
   1. package com.dicoding.javafundamental.operator;
   3. public class OperatorUnary {
   5. **public static void main(String[] args) {**
   6. **System.out.println("Operator Unary Plus");**
   7. **int nilaiAwal = 5;**
   8. **int hasil = +nilaiAwal;**
   9. **System.out.println("Hasil +5 = " + hasil);**
   10. **System.out.println();**
   11. **System.out.println("Operator Unary Minus");**
   12. **int nilaiAwal2 = 5;**
   13. **nilaiAwal2 = -nilaiAwal2;**
   14. **System.out.println("Hasil -5 = " + nilaiAwal2);**
   15. **System.out.println();**
   16. **System.out.println("Operator peningkatan nilai sebesar 1 point");**
   17. **int nilaiAwal3 = 5;**
   18. **nilaiAwal3++;**
   19. **System.out.println("Hasil 5++ = " + nilaiAwal3);**
   20. **System.out.println();**
   21. **System.out.println("Operator pengurangan nilai sebesar 1 point");**
   22. **int nilaiAwal4 = 5;**
   23. **nilaiAwal4--;**
   24. **System.out.println("Hasil 5-- = " + nilaiAwal4);**
   25. **System.out.println();**
   26. **System.out.println("Operator komplemen logika");**
   27. **boolean sukses = false; *//Nilai sukses adalah false***
   28. **System.out.println("Hasil !false = " + !sukses);**
   29. **System.out.println();**
   30. **}**
   31. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Operator Unary Plus Hasil +5 = 5  Operator Unary Minus Hasil -5 = -5  Operator peningkatan nilai sebesar 1 point Hasil 5++ = 6  Operator pengurangan nilai sebesar 1 point Hasil 5-- = 4  Operator komplemen logika Hasil !false = true |

### Bedah Code Operasi Unary

#### Operasi Unary Plus

1. int value = +5;

Operator yang menunjukkan bila value yang diberikan operator ini adalah positif. Memang seecara default, bilangan selalu positif.

#### Operasi Unary Minus

1. int value = -5;

Seperti halnya Operatur Plus, Operator Minus bernilai kebalikannya.

#### Operasi Unary Increment

1. int value = 5;
2. value++;

Operator yang memberikan penambahan nilai sebesar 1 point untuk value yang di-increment. Operator ini biasa digunakan untuk pengulangan, yang akan dijelaskan pada materi lainnya.

#### Operasi Unary Decrement

1. int value = 5;
2. value--;

Kebalikan dari Operator di atas, operator decrement memberikan pengurangan nilai sebesar 1 point.

#### Operasi Unary Logical Complement

1. boolean sukses = false; //Nilai sukses adalah false
2. System.out.println("Hasil !false = " + !sukses);

Operator ini akan memberikan nilai kebalikan yang diberikan oleh sebuah boolean seperti pada kode di atas. Awalnya nilai dari sukses bernilai false. Namun ketika dicetak dengan diberikan operator ini, menjadi true.

## Operasi Equality and Relational

Operator Equality and Relational digunakan untuk menentukan dan membandingkan antara 2 variabel (operand) yang menghasilkan nilai True atau False. Berikut Operator Equality and Relational yang ada pada pemrograman Java :

|  |  |
| --- | --- |
| **==** | equal to |
| **!=** | not equal to |
| **>** | greater than |
| **>=** | greater than or equal to |
| **<** | less than |
| **<=** | less than or equal to |

### Codelab Operasi Equality and Relational

Langsung saja kita praktikkan bagaimana menggunakan operator di atas melalui kode di bawah ini:

1. Bukalah proyek **Operator** dan tambahkan kelas baru didalamnya dengan nama OperatorEquality.
2. Tambahkan kode di bawah ini ke dalam kelas OperatorEquality:
   1. package com.dicoding.javafundamental.operator;
   3. public class OperatorEquality {
   5. **public static void main(String[] args) {**
   6. **int value = 5;**
   7. **int anotherValue = 4;**
   8. **boolean result;**
   9. **result = value == anotherValue;**
   10. **System.out.println("Hasil 'value == anotherValue' adalah " + result);**
   11. **System.out.println();**
   12. **System.out.println("Tidak sama dengan..");**
   13. **result = value != anotherValue;**
   14. **System.out.println("Hasil 'value != anotherValue' adalah " + result);**
   15. **System.out.println();**
   16. **System.out.println("Lebih besar dari..");**
   17. **result = value > anotherValue;**
   18. **System.out.println("Hasil 'value > anotherValue' adalah " + result);**
   19. **System.out.println();**
   20. **System.out.println("Sama atau lebih besar dari..");**
   21. **result = value >= anotherValue;**
   22. **System.out.println("Hasil 'value >= anotherValue' adalah " + result);**
   23. **System.out.println();**
   24. **System.out.println("Kurang dari..");**
   25. **result = value < anotherValue;**
   26. **System.out.println("Hasil 'value < anotherValue' adalah " + result);**
   27. **System.out.println();**
   28. **System.out.println("Sama atau kurang dari dengan..");**
   29. **result = value <= anotherValue;**
   30. **System.out.println("Hasil 'result <= anotherValue' adalah " + result);**
   31. **System.out.println();**
   32. **}**
   33. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Hasil 'value == anotherValue' adalah false  Tidak sama dengan.. Hasil 'value != anotherValue' adalah true  Lebih besar dari.. Hasil 'value > anotherValue' adalah true  Sama atau lebih besar dari.. Hasil 'value >= anotherValue' adalah true  Kurang dari.. Hasil 'value < anotherValue' adalah false  Sama atau kurang dari dengan.. Hasil 'value <= anotherValue' adalah false |

#### Bedah Code Equality and Relational

Perhatikan kode berikut:

1. int value = 5;
2. int anotherValue = 4;
4. result = value == anotherValue;
5. result = value != anotherValue;
6. result = value > anotherValue;
7. result = value >= anotherValue;
8. result = value < anotherValue;
9. result = value <= anotherValue;

Operator Unary pada Java terlihat pada cuplikan kode di atas. Output-nya menghasilkan nilai berupa true/false sesuai kondisi nilai yang diberikan.

## Operasi Conditional

Operator Conditional digunakan pada dua atau lebih ekspresi **boolean**. Pada operator ini, nilai dari operator kedua, ketiga dst menjadi pertimbangan hasil dari **true** atau **false**, bila diperlukan. Hal ini tergantung dari operator Condition yang digunakan, yang jenisnya adalah:

|  |  |
| --- | --- |
| **&&** | Conditional-AND |
| **||** | Conditional-OR |

#### Codelab Operasi Conditional

Mari kita praktikkan bagaimana menggunakan Operasi Condition di atas:

1. Bukalah proyek **Operator** dan tambahkan kelas baru didalamnya dengan nama OperatorConditional.
2. Tambahkan kode di bawah ini ke dalam kelas OperatorConditional:
   1. package com.dicoding.javafundamental.operator;
   3. public class OperatorConditional {
   5. public static void main(String[] args) {
   6. **int value = 4;**
   7. **int anotherValue = 5;**
   8. **System.out.println("Conditional AND");**
   9. **boolean result = value == 3 && anotherValue == 5;**
   10. **boolean anotherResult = value != 3 && anotherValue == 5;**
   11. **System.out.println("Hasil Operator AND pada syarat value == 3 dan anotherValue == 5 adalah " + result);**
   12. **System.out.println("Hasil Operator AND pada syarat value != 3 dan anotherValue == 5 adalah " + anotherResult);**
   13. **System.out.println();**
   14. **System.out.println("Conditional OR");**
   15. **result = value == 3 || anotherValue == 5;**
   16. **anotherResult = value != 3 || anotherValue == 5;**
   17. **System.out.println("Hasil Operator OR pada syarat value == 3 dan anotherValue == 5 adalah " + result);**
   18. **System.out.println("Hasil Operator OR pada syarat value != 3 dan anotherValue == 5 adalah " + anotherResult);**
   19. **System.out.println();**
   20. }
   21. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Conditional AND Hasil Operator AND pada syarat value == 3 dan anotherValue == 5 adalah false Hasil Operator AND pada syarat value != 3 dan anotherValue == 5 adalah true  Conditional OR Hasil Operator OR pada syarat value == 3 dan anotherValue == 5 adalah true Hasil Operator OR pada syarat value != 3 dan anotherValue == 5 adalah true |

### Bedah Code Operasi Conditional

#### Conditional AND

1. boolean result = value == 3 && anotherValue == 5;
2. boolean anotherResult = value != 3 && anotherValue == 5;

Pada kode yang menerapkan Operator **AND** di atas ini, terlihat bila ada salah satunya tidak memenuhi syarat yang sudah diberikan, hasilnya adalah nilai **false**.

#### Conditional OR

1. result = value == 3 || anotherValue == 5;
2. anotherResult = value != 3 || anotherValue == 5;

Berkebalikan dengan operator Conditional **AND**, Operator **OR** akan memberikan nilai true apabila salah satunya sudah memenuhi syarat.

# Input Output

Pemrograman Java terdiri dari 3 komponen dasar yaitu **Input**, **Proses** dan **Output**. Dari materi sebelumnya juga kita sudah belajar menggunakan beberapa kode yang merupakan bagian dari **Proses**dan **Output.**Sudahkah Anda menyadarinya? Pada materi ini kita akan mempelajari bagaimana mengambil hasil **Input**hingga bagaimana hasil **Input**diproses dan ditampilkan pada layar.

## Lantas bagaimana Input, Proses dan Output itu?

* **Input** : Nilai yang dimasukkan
* **Proses** : Serangkaian langkah yang dilakukan untuk mengelola input yang diberikan
* **Output** : Menampilkan hasil olah data.

Pada bahasa pemrograman Java, **Basic Input-**nya sudah dibekali beberapa library untuk membantu pengambilan Input berbasis teks. Beberapa di antaranya:

* **Bufferedreader**
* **Scanner**

Sedangkan untuk menampilkan atau **Output**dari program adalah:

* **Print**
* **Println**

Berikut bagaimana masing-masing kelas yang disediakan oleh Bahasa Pemogragaman Java untuk mengambil input dari user.

## Scanner

Scanner merupakan kelas yang menyediakan fungsi-fungsi untuk membaca dan mengambil input dari pengguna. Scanner memiliki kemudahan yang dapat membaca teks, baik yang memiliki tipe **data primitif**maupun **string**.

### Codelab Scanner

Mari kita coba kode di bawah ini untuk implementasi materi **Scanner** ini.

1. Buatlah proyek baru dengan nama **InputOuput** dengan nama package com.dicoding.javafundamental.inputouput di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama InputOutputFunction, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.inputouput;
   3. **import java.util.Scanner;**
   5. public class InputOutputFunction {
   7. **public static void main(String[] args) {**
   8. **Scanner scanner = new Scanner(System.in);**
   9. **System.out.println("Program penjumalahan sangat sederhana");**
   10. **System.out.print("Masukan Angka pertama : ");**
   11. **int value = scanner.nextInt();**
   12. **System.out.print("Masukan Angka kedua : ");**
   13. **int anotherValue = scanner.nextInt();**
   14. **int result = value + anotherValue;**
   15. **System.out.println("Hasilnya adalah : " + result);**
   16. **}**
   17. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Program penjumlahan sangat sederhana Masukan Angka pertama : **10** Masukan Angka kedua : **5** Hasilnya adalah : 15 |

### Bedah Code Scanner

Perhatikan kode di bawah ini:

1. Scanner scanner = new Scanner(System.in);

Kode di atas merupakan inisialisasi awal ketika akan menggunakan Scanner. Di bagian ini kita mencoba untuk menggunakan perintah mengambil input yang diberikan oleh user. Bila program dijalankan, setiap input yang diberikan akan disimpan oleh Scanner.

1. System.out.print("Masukan Angka pertama : ");
2. int value = scanner.nextInt();

Selanjutnya kita akan mengambil data yang telah dimasukkan melalui Scanner. Dengan memanggilScanner.nextInt()setiap **Input** dari keyboard akan diberikan ke value.

Adapun catatan yang harus diperhatikan adalah penggunaan fungsi Scanner. Pengambilan data sangat bergantung pada tipe data yang dimasukan. Ini harus sesuai, misalnya untuk **string**, panggilah Scanner.next(). Jika data yang diinputkan**Integer,** panggilah Scanner.nextInt(). Jika input tidak sesuai, program akan menemui eror.

## BufferedReader

Kelas dalam Java ini merupakan paket dari Java.io. BufferedReader dapat digunakan pada materi ini sebagai Basic Input karena sebenarnya kelas ini tidak hanya digunakan untuk membaca input dari keyboard saja, melainkan juga untuk mendapatkan input dari user. Fungsi terakhir ini adalah fungsi dasar BufferedReader yang sama dengan Scanner**.**

Dalam implementasinyaBufferedReader tidak dapat berjalan sendiri. Untuk mendapatkan input dibutuhkan kelas InputStreamReader.

### Codelab BufferedReader

Pada codelab kali ini, kita akan memanfaatkan BufferdReader untuk mendapatkan input dari user.

1. Bukalah kembali proyek **InputOutput** dan buatlah kelas baru dengan nama UserBufferedReader.
2. Masukkan kode berikut ke dalam kelas UserBufferedReader:
   1. package com.dicoding.javafundamental.inputouput;
   3. **import java.io.BufferedReader;**
   4. **import java.io.IOException;**
   5. **import java.io.InputStreamReader;**
   7. public class UserBufferedReader {
   9. **public static void main(String[] args) {**
   10. **InputStreamReader streamReader = new InputStreamReader(System.in);**
   11. **BufferedReader bufferedReader = new BufferedReader(streamReader);**
   12. **System.out.println("Program penjumlahan sangat sederhana");**
   13. **int value = 0;**
   14. **int anotherValue = 0;**
   15. **try {**
   16. **System.out.print("Masukan Angka pertama : ");**
   17. **value = Integer.parseInt(bufferedReader.readLine());**
   18. **System.out.print("Masukan Angka kedua : ");**
   19. **anotherValue = Integer.parseInt(bufferedReader.readLine());**
   20. **} catch (IOException e) {**
   21. **e.printStackTrace();**
   22. **}**
   23. **int resul = value + anotherValue;**
   24. **System.out.println("Hasilnya adalah : " + resul);**
   25. **}**
   26. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Program penjumlahan sangat sederhana Masukan Angka pertama : **9** Masukan Angka kedua : **2** Hasilnya adalah : 11 |

### Bedah Code BufferedReader

Mari kita kupas bagaimana kode di atas berjalan.

1. InputStreamReader streamReader = new InputStreamReader(System.in);
2. BufferedReader bufferedReader = new BufferedReader(streamReader);

Tidak jauh berbeda dengan cara dari penggunaan Scanner, kita harus menginisialisasi BufferedReader yang dibantu oleh InputStreamReaderagar dapat membaca Input dari Keyboard.

1. try {
2. System.out.print("Masukan Angka pertama : ");
3. value = Integer.parseInt(bufferedReader.readLine());
4. System.out.print("Masukan Angka kedua : ");
5. anotherValue = Integer.parseInt(bufferedReader.readLine());
7. } catch (IOException e) {
8. e.printStackTrace();
9. }

Namun sedikit berbeda dengan implementasi Scanner, tambahkan ioException sebagai penanganan error input pada BufferedReader. Selain itu untuk menyimpan pada sebuah variabel, data yang diberikan oleh BufferedReader harus kita parsing terlebih dahulu.

1. int value = Integer.parseInt(bufferedReader.readLine());

Seperti pada potongan kode diatas, bila ingin menyimpan data pada int maka nilai yang ada pada bufferedReader.readLine()harus dikonversi dahulu ke dalam bentuk integer menggunakan Integer.parseInt***.***

## print dan println

Tentunya kita sudah familiar bagaimana memberikan output dari yang sudah kita praktikkan pada beberapa contoh program sebelumnya. Pada dasarnya kedua fungsi di atas sama-sama memiliki peran untuk menampilkan teks di Console maupun dari IDE yang kita gunakan.

Apa yang berbeda diantara 2 fungsi di atas? Berikut implementasinya:

### Codelab print dan println

Pada codelab kali ini, kita akan memanfaatkan **print** dan **println:**

1. Bukalah kembali proyek **InputOutput** dan buatlah kelas baru dengan nama UserOutput.
2. Masukkan kode berikut ke dalam kelas UserOutput:
   1. package com.dicoding.javafundamental.inputouput;
   3. public class UserOutput {
   5. public static void main(String[] args) {
   6. System.out.println("Dicetak pakai println()");
   7. System.out.println("Ini juga dicetak pakai println()");
   8. System.out.print("Ini dicetak dengan print()");
   9. System.out.print(" dan ini juga dicetak dengan print()");
   10. }
   11. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Dicetak pakai println() Ini juga dicetak pakai println() Ini dicetak dengan print() dan ini juga dicetak dengan print() |

### Bedah Code print dan println

Bila kode di atas kita jalankan, Ouput yang diberikan kurang lebih seperti di bawah ini:

1. Dicetak pakai println()
2. Ini juga dicetak pakai println()
3. Ini dicetak dengan print() dan ini juga dicetak dengan print()

Saat program dijalankan, semua kode yang di-print di atas akan keluar sesuai fungsi yang kita pakai. Kita bedah kode diatas.

1. System.out.println("Dicetak pakai println()");

Penggunaan printlnakan menampilkan teks dan tambahan baris baru.

1. System.out.print("Ini dicetak dengan print()");
2. System.out.print(" dan ini juga dicetak dengan print()");

Sedangkan fungsi printmenampilkan keluaran berupa teks sesuai dengan yang dimasukkan.

# Array

Array adalah obyek yang bisa digunakan untuk menyimpan kumpulan data lebih dari satu dengan tipe  sama. Array memiliki jumlah data yang fixed (tetap).

## Ilustrasi Array

Array bisa diilustrasikan dengan gambar berikut:

Angka di atas menunjukkan indeks dari **array** tersebut, yakni dimulai dari **0** sampai **6**. Setiap kolom nilai bisa disebut sebagai elemen.

## Deklarasi Array

Untuk mendeklarasikan suatu array kita bisa menuliskannya seperti berikut ini.

1. void cobaArray(){
2. **double[] arrA;**
3. **arrA = new double[10];**
4. }

Tanda [] menunjukkan bahwa obyek tersebut adalah sebuah **array**. Kemudian, setelah membuat suatu objek array yang memiliki tipe data **double**, kita harus mendefinisikan length (panjang) dari array tersebut. Untuk mendefinisikan panjangnya, kita harus menuliskan jumlahnya dalam nominal integer di dalam tanda [].

Perlu Anda ketahui juga bahwa hanya terdapat 2 jenis penulisan array yang diperbolehkan di dalam Java, yakni:

1. void cobaArray(){
2. // Cara pertama
3. double**[]** arrA = new double[10];
5. // Cara kedua
6. double arrB**[]** = new double[10];
7. }

Keduanya diperkenankan, akan tetapi cara pertama lebih lazim digunakan. Oleh karena itu pada materi ini kita akan menggunakan cara pertama dalam penulisannya.

## Inisiasi Array

Untuk inisiasi suatu array, ada 2 cara yang bisa digunakan.

Pertama, menggunakan tanda {} (sepasang kurung kurawal atau braces/curly brackets) dan memisahkan nilai antar elemen dengan tanda koma.

### Codelab Inisiasi Array

Selanjutnya kita coba melakukan inisiasi Array agar kalian makin paham.

1. Buatlah proyek baru dengan nama **Array** dengan nama package com.dicoding.javafundamental.array di dalamnya:
2. Buatlah sebuah kelas baru dengan nama InisiasiArray, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.array;
   3. public class InisiasiArray {
   4. **public static void main(String[] args) {**
   5. **int[] arrInt = new int[]{1, 2, 3, 4, 5, 6};**
   6. **System.out.println(arrInt[0]);**
   7. **System.out.println(arrInt[1]);**
   8. **System.out.println(arrInt[2]);**
   9. **System.out.println(arrInt[3]);**
   10. **System.out.println(arrInt[4]);**
   11. **System.out.println(arrInt[5]);**
   12. **}**
   13. }

Cara di atas merupakan salah satu contoh untuk melakukan insialisasi array.

1. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| 1 2 3 4 5 6 |

1. Selanjutnya cara kedua dengan melakukan inisiasi array per elemen. Ubahlah kode di dalam kelas InisiasiArraymenjadi seperti ini:
   1. package com.dicoding.javafundamental.array;
   3. public class InisiasiArray {
   4. public static void main(String[] args) {
   5. **int[] arrInt = new int[6];**
   6. **arrInt[0] = 1;**
   7. **arrInt[1] = 2;**
   8. **arrInt[2] = 3;**
   9. **arrInt[3] = 4;**
   10. **arrInt[4] = 5;**
   11. **arrInt[5] = 6;**
   13. System.out.println(arrInt[0]);
   14. System.out.println(arrInt[1]);
   15. System.out.println(arrInt[2]);
   16. System.out.println(arrInt[3]);
   17. System.out.println(arrInt[4]);
   18. System.out.println(arrInt[5]);
   19. }
   20. }
2. Jalankanlah kembali kode di atas bila sukses, seharusnya Console akan menampilkan **output**seperti ini:

|  |
| --- |
| 1 2 3 4 5 6 |

1. Selain itu, cara pertama juga kita sederhanakan dengan menghilangkan kata new + **tipe data**-nya, menjadi seperti ini:
   1. package com.dicoding.javafundamental.array;
   3. public class InisiasiArray {
   4. public static void main(String[] args) {
   5. **int[] arrInt = {1, 2, 3, 4, 5, 6};**
   7. System.out.println(arrInt[0]);
   8. System.out.println(arrInt[1]);
   9. System.out.println(arrInt[2]);
   10. System.out.println(arrInt[3]);
   11. System.out.println(arrInt[4]);
   12. System.out.println(arrInt[5]);
   13. }
   14. }
2. Jalankanlah kembali kode di atas maka tampilan **output**akan jadi seperti ini:

|  |
| --- |
| 1 2 3 4 5 6 |

### Bedah Kode Kegunaan Array

Lalu apa kegunaan utama dari suatu array? Seperti yang sudah dijelaskan sebelumnya, array berupa kumpulan data. Ini berarti bahwa satu objek array dapat menyimpan beberapa data sekaligus.

Jika dibandingkan dengan variabel primitif maka untuk membuat 6 buah variabel integer kita bisa menuliskannya seperti berikut:

1. int int1 = 1;
2. int int2 = 2;
3. int int3 = 3;
4. int int4 = 3;
5. int int5 = 5;
6. int int6 = 6;

Penulisan kode di atas bisa kita konversi dengan menggunakan sebuah array yang memiliki 6 elemen, seperti ini:

1. int[] arrInt = new int[6];

Bayangkan jika program kita mengharuskan penulisan obyek sebanyak 1 juta? Tentu kita dapat menghemat banyak energi dalam menulis baris kode dengan array!.

Namun sebenarnya proses konversi kode di atas belum selesai karena kita belum memberikan nilai pada setiap elemennya. Mari masuk ke bab selanjutnya.

## Akses Array

Pada latihan sebelumnya Anda sudah mengetahi bagaimana cara mengakses elemen dari suatu array, yaitu dengan menggunakan tanda [] (tanda kurung siku atau square backets) dan indeksnya. Perhatikan kode di bawah ini untuk akses array.

1. int[] arrInt = {1, 2, 3, 4, 5, 6};
3. System.out.println(arrInt[0]);
4. System.out.println(arrInt[1]);
5. System.out.println(arrInt[2]);
6. System.out.println(arrInt[3]);
7. System.out.println(arrInt[4]);
8. System.out.println(arrInt[5]);

Output-nya:

|  |
| --- |
| 1 2 3 4 5 6 |

## Proses Array

Sebelumnya telah dijelaskan mengenai beberapa poin penting mengenai array, tapi yang tidak kalah penting adalah hubungan antara array dengan looping. **Kenapa looping**? Kembali ke pertanyaan bagaimana jika array-nya memiliki 1 juta panjang elemen, maka kita harus menggunakan looping untuk melakukan proses elemennya.

Jika kita ingin menampilkan nilai array yang memiliki panjang 1 juta dengan menggunakan akses, tiap indeksnya akan kurang lebih seperti ini.

1. public class LoopingArray {
2. public static void main(String[] args) {
3. int[] arrInt = new int[]{1, 2, 3, 4, 5, ..., 999998, 999999, 1000000};
4. System.out.println(arrInt[0]);
5. System.out.println(arrInt[1]);
6. System.out.println(arrInt[2]);
7. System.out.println(arrInt[3]);
9. **...**
11. System.out.println(arrInt[999998]);
12. System.out.println(arrInt[999999]);
13. }
14. }

Jika kita konversi kode di atas dengan menggunakan looping, kita bisa mendapatkan panjang array dengan mengakses atribut length. Kodenya akan menjadi seperti ini.

1. public class LoopingArray {
2. public static void main(String[] args) {
3. int[] arrInt = new int[1000000];
5. // Inisiasi dan menampilkan arrInt dari elemen ke 1 sampai 1000000
6. for (int x = 0; x < arrInt.length; x++) {
7. arrInt[x] = x + 1;
8. System.out.println(arrInt[x]);
9. }
10. }
11. }

## Multi Dimensional Array

Seperti dengan namanya “multi dimensional” berarti array bisa memiliki lebih dari 1 dimensi. Pada penjelasan sebelumnya kita baru hanya menggunakan array yang memiliki 1 dimensi. Jika diilustrasikan dengan dimensi maka 1 dimensi array baru hanya menggunakan sumbu x, seperti ini.

Untuk 2 dimensi array kita bisa menambahkan sumbu y, dengan ilustrasi seperti berikut.

Penulisan kode untuk 2 dimensi array adalah seperti ini.

1. void cobaArray2D() {
2. int[][] arrInt = new int[][];
3. }

Perhatikan bahwa ada 2 tanda [] (**bracket**) yaitu [][]. Tanda [] **pertama** adalah dimensi **pertama** dan tanda [] **kedua** adalah dimensi **kedua**. Sama juga dengan dimensi yang lebih dari itu, misalnya **3** dimensi, maka ada **3** tanda [] yaitu [][][].

Kemudian untuk panjang elemen, pada dimensi kedua tiap panjang elemen-nya pun bisa bervariasi. Misalnya kita ingin membuat array 2 dimensi, panjang elemen dimensi 1 nya adalah 2, kemudian panjang elemen pada dimensi keduanya adalah 2 dan 3 secara berurutan. Penasaran? Langsung kita praktikan!

### Codelab Multi Dimensional Array

1. Bukalah kembali proyek **Array** dan buatlah kelas baru dengan nama MultiDimensionalArray.
2. Masukkan kode berikut ke dalam kelas MultiDimensionalArray:
   1. package com.dicoding.javafundamental.array;
   3. public class MultiDimensionalArray {
   4. **public static void main(String[] args) {**
   5. **char[][] arrChar = new char[2][];**
   6. **arrChar[0] = new char[2];**
   7. **arrChar[1] = new char[3];**
   8. **}**
   9. }
3. Selanjutntya tambahkan kode berikut untuk menambahkan elemen di tiap indeksnya:
   1. package com.dicoding.javafundamental.array;
   3. public class MultiDimensionalArray {
   4. public static void main(String[] args) {
   5. char[][] arrChar = new char[2][];
   6. arrChar[0] = new char[2];
   7. arrChar[1] = new char[3];
   9. **// Dimensi 1 yang indeksnya 0 memiliki panjang elemen 2**
   10. **arrChar[0][0] = 'A';**
   11. **arrChar[0][1] = 'B';**
   12. **// Dimensi 1 yang indeksnya 1 memiliki panjang elemen 3**
   13. **arrChar[1][0] = 'C';**
   14. **arrChar[1][1] = 'D';**
   15. **arrChar[1][2] = 'E';**
   16. **// Tampilkan semua data dari dimensi 1 yang memiliki indeks 0**
   17. **System.out.println("Dimensi 1 dengan indeks 0 dan Dimensi 2 dengan indeks 0 : " + arrChar[0][0]);**
   18. **System.out.println("Dimensi 1 dengan indeks 0 dan Dimensi 2 dengan indeks 1 : " + arrChar[0][1]);**
   19. **// Tampilkan semua data dari dimensi 1 yang memiliki indeks 1**
   20. **System.out.println("Dimensi 1 dengan indeks 1 dan Dimensi 2 dengan indeks 0 : " + arrChar[1][0]);**
   21. **System.out.println("Dimensi 1 dengan indeks 1 dan Dimensi 2 dengan indeks 1 : " + arrChar[1][1]);**
   22. **System.out.println("Dimensi 1 dengan indeks 1 dan Dimensi 2 dengan indeks 2 : " + arrChar[1][2]);**
   23. }
   24. }
4. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Dimensi 1 dengan indeks 0 dan Dimensi 2 dengan indeks 0 : A Dimensi 1 dengan indeks 0 dan Dimensi 2 dengan indeks 1 : B Dimensi 1 dengan indeks 1 dan Dimensi 2 dengan indeks 0 : C Dimensi 1 dengan indeks 1 dan Dimensi 2 dengan indeks 1 : D Dimensi 1 dengan indeks 1 dan Dimensi 2 dengan indeks 2 : E |

## IndexOutOfBounds

Terakhir, selama menggunakan array Anda perlu berhati-hati. Jika tidak, Anda akan sering menemui erorindexOutOfBounds. Ini adalah eror yang disebabkan oleh percobaan saat melakukan akses indeks yang tidak dimiliki oleh array. Misalnya kita melakukan akses pada indeks 4 sedangkan array tersebut hanya memiliki 4 elemen. Ingatlah bahwa indeks dimulai dari 0 dan indeks terakhir adalah panjang element - 1. Agar Anda semakin paham, mari kita coba praktikan!

### Codelab IndexOutOfBounds

1. Bukalah kembali proyek **Array** dan buatlah kelas baru dengan nama IndexOutOfBound.
2. Masukkan kode berikut ke dalam kelas IndexOutOfBound:
   1. package com.dicoding.javafundamental.array;
   3. public class IndexOutOfBound {
   4. public static void main(String[] args) {
   5. **int[] arrA = {1, 2, 3, 4};**
   6. **System.out.println(arrA[0]);**
   7. **System.out.println(arrA[1]);**
   8. **System.out.println(arrA[2]);**
   9. **// Akses indeks ke 4**
   10. **System.out.println(arrA[4]);**
   11. }
   12. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| 1 2 3 Exception in thread "main" java.lang.**ArrayIndexOutOfBoundsException**: **Index 4 out of bounds for length 4** at com.dicoding.javafundamental.array.IndexOutOfBound.main(IndexOutOfBound.java:11) |

### Bedah Code IndexOutOfBounds

Pada kode di atas, kita melakukan perulangan dengan mengeluarkan seluruh isi array. Metode ini dapat kita sebut sebagai iterasi. Cara di atas kita tidak perlu menginisalisasi angka awal dan akhir pada sebuah perulangan. Menggunakan cara ini sangat direkomendasikan.

Setelah Anda menguasai dasar dari bahasa Java, lanjut ke bagian selanjutnya yaitu Control Flow. Control flow dalam bahasa Indonesia dapat diartikan sebagai aliran kendali. Maksud sebenarnya dari control flow adalah bagaimana urutan eksekusi perintah di dalam program. Oleh karena itu kita perlu tahu control flow dalam Java yang dikemas dalam materi berikut:

* **Percabangan**Jalannya suatu program pasti akan bergantung pada kondisi yang dipilih. Untuk itu kita memerlukan percabangan yang berfungsi sebagai *control flow statement*atau*statement*yang dapat mengontrol alur jalan suatu kode berdasarkan kondisinya. Contohnya adalah if-else dan switch.
* **Perulangan**Perulangan adalah kode yang bisa kita gunakan untuk mengulang beberapa instruksi kode, seperti *for*, *while*, dan *do-while*.

# Percabangan

Ada ungkapan “Seribu jalan ke Roma” yang bermakna ada beragam opsi jalan yang bisa dipilih untuk menuju ke Roma. Demikian halnya dengan pemrograman. Terdapat berbagai pilihan dalam pemrograman. Kita memilih salah satu diantaranya. Di tengah-tengah, kita akan menemui problem di mana berjalannya suatu program tergantung dari kondisi yang ditemui. Percabangan bisa kita sebut sebagai control flow statement, di mana fungsi utamanya adalah mengontrol alur program.

## If-Then

Percabangan **if-then** adalah control flow statement yang paling dasar di dalam pemrograman Java. Ini sangat simpel dengan artian program akan menjalankan baris kode jika kondisinya adalah **true**.

### Codelab If-Then

Mari kita coba percabangan If-Then.

1. Buatlah proyek baru dengan nama **Percabangan** dengan nama package com.dicoding.javafundamental.percabangan di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama IfThen, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.percabangan;
   3. public class IfThen {
   4. **public static void main(String[] args) {**
   5. **boolean isOn = true;**
   6. **if (isOn) {**
   7. **System.out.println("Menyalakan lampu");**
   8. **}**
   9. **}**
   10. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Menyalakan lampu |

### Bedah Code If-Then

Perhatikan kode berikut:

1. if (isOn) {
2. System.out.println("Menyalakan lampu");
3. }

Kode di atas berarti ketika nilai isOn adalah **true** maka obyek lampu akan memanggil metode nyala(). Ini adalah contoh penggunaan percabangan yang sangat simpel, di mana ketika suatu kondisi nilainya **true** maka kode yang ada di dalam {} (braces) akan bisa dijalankan.

Penggunaan braces adalah optional ketika kode yang dijalankan hanyalah 1 baris. Akan tetapi jika terdapat lebih dari 1 baris, maka kode yang di dalamnya harus ada dalam {}. Misalnya, pada contoh kode menyalakan lampu seperti di atas, kita hilangkan tanda {}.

1. void ganti() {
2. **if (isOn)**
3. System.out.println("Menyalakan lampu");
4. }

Dan ketika ada lebih dari 1 baris kode maka kita harus memasukkannya ke dalam {} (braces) seperti ini.

1. void ganti() {
2. **if (isOn) {**
3. System.out.println("Menyalakan lampu");
4. System.out.println("Menyalakan lampu lain");
5. **}**
6. }

## If-Then-Else

Kita juga bisa menambahkan seleksi kondisi baru yaitu **else**. Ketika semua kondisi di dalam if bernilai **false** maka kode di dalam else akan dijalankan.

### Codelab If-Then-Else

Mari kita coba percabangan If-Then-Else.

1. Bukalah kembali proyek **Percabangan** dan buatlah kelas baru dengan nama IfThenElse.
2. Masukkan kode berikut ke dalam kelas IfThenElse:
   1. package com.dicoding.javafundamental.percabangan;
   3. public class IfThenElse {
   4. **public static void main(String[] args) {**
   5. **boolean isOn = false;**
   6. **if (isOn) {**
   7. **System.out.println("Menyalakan lampu");**
   8. **} else {**
   9. **System.out.println("Kondisi tidak terpenuhi...");**
   10. **}**
   11. **}**
   12. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Kondisi tidak terpenuhi... |

### Bedah Kode If-Then-Else

Perhatikan kode berikut:

1. if (isOn) {
2. System.out.println("Menyalakan lampu");
3. } else {
4. System.out.println("Kondisi tidak terpenuhi...");
5. }

Kode di atas berarti ketika nilai isOn bernilai **false** maka baris kode yang ada di dalam else akan dijalankan. Di sini else berfungsi sebagai secondary option di mana ketika opsi pertama if  gagal, maka opsi kedua else dijalankan.  Sebenarnya else bukan hanya sebagai secondary option saja. Else lebih cocok dikatakan sebagai last option (opsi terakhir).

## If-Then-ElseIf-Then-Else

Di antara if dan else, kita bisa menambahkan **else if.**

### Codelab If-Then-ElseIf-Then-Else

Mari kita coba percabangan If-Then-ElseIf-Then-Else.

1. Bukalah kembali proyek **Percabangan** dan buatlah kelas baru dengan nama IfThenElseIfThenElse.
2. Masukkan kode berikut ke dalam kelas IfThenElseIfThenElse:
   1. package com.dicoding.javafundamental.percabangan;
   3. public class IfThenElseIfThenElse {
   4. **public static void main(String[] args) {**
   5. **int nilaiUjian = 80;**
   6. **char indeksPrestasi;**
   7. **if (nilaiUjian >= 90) {**
   8. **indeksPrestasi = 'A';**
   9. **} else if (nilaiUjian >= 80) {**
   10. **indeksPrestasi = 'B';**
   11. **} else if (nilaiUjian >= 70) {**
   12. **indeksPrestasi = 'C';**
   13. **} else if (nilaiUjian >= 60) {**
   14. **indeksPrestasi = 'D';**
   15. **} else if (nilaiUjian >= 50) {**
   16. **indeksPrestasi = 'E';**
   17. **} else {**
   18. **indeksPrestasi = 'F';**
   19. **}**
   20. **System.out.println("Nilai ujian akhir anda adalah " + indeksPrestasi);**
   21. **}**
   22. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Nilai ujian akhir anda adalah B |

### Bedah Code If-Then-ElseIf-Then-Else

Perhatikan kode berikut ini:

1. if (nilaiUjian >= 90) {
2. indeksPrestasi = 'A';
3. } else if (nilaiUjian >= 80) {
4. indeksPrestasi = 'B';
5. } else if (nilaiUjian >= 70) {
6. indeksPrestasi = 'C';
7. } else if (nilaiUjian >= 60) {
8. indeksPrestasi = 'D';
9. } else if (nilaiUjian >= 50) {
10. indeksPrestasi = 'E';
11. } else {
12. indeksPrestasi = 'F';
13. }

Kode di atas memiliki beberapa cabang kode yang bisa dijalankan. Tergantung dari nilai variabel integer nilaiUjian, kode yang dijalankan akan mengikuti nilainya. Di atas atau sama dengan **90** maka indeks prestasi adalah '**A'**; di atas **80** maka '**B'**; di atas **70** maka **'C'**; di atas **60** maka **'D'**; di atas **50** maka **'E'**; terakhir di bawah **50** maka indeks prestasinya adalah **'F'**.

Bisa disimpulkan bahwa else merupakan pilihan terakhir. Ketika if dan semua else-ifnilainya adalah **false** maka kode di dalam else yang akan dijalankan.

## Switch

Control flow lainnya adalah **switch**, yang menggunakan tipe data primitif seperti int, short, byte, char. Bisa juga dengan menggunakan tipe enumerated, dan string.

### Codelab Switch

Mari kita coba percabangan Switch.

1. Bukalah kembali proyek **Percabangan** dan buatlah kelas baru dengan nama Switch.
2. Masukkan kode berikut ke dalam kelas Switch:
   1. package com.dicoding.javafundamental.percabangan;
   3. public class Switch {
   4. **public static void main(String[] args) {**
   5. **int input = 6;**
   6. **switch (input) {**
   7. **case 1:**
   8. **System.out.println("1");**
   9. **break;**
   10. **case 2:**
   11. **System.out.println("2");**
   12. **break;**
   13. **case 3:**
   14. **System.out.println("3");**
   15. **break;**
   16. **case 4:**
   17. **System.out.println("4");**
   18. **break;**
   19. **case 5:**
   20. **System.out.println("5");**
   21. **break;**
   22. **case 6:**
   23. **System.out.println("6");**
   24. **break;**
   25. **case 7:**
   26. **System.out.println("7");**
   27. **break;**
   28. **case 8:**
   29. **System.out.println("8");**
   30. **break;**
   31. **case 9:**
   32. **System.out.println("9");**
   33. **break;**
   34. **default:**
   35. **break;**
   36. **}**
   37. **}**
   38. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| 6 |

### Codelab Switch

1. switch (input) {
2. case 1:
3. System.out.println("1");
4. break;
5. case 2:
6. System.out.println("2");
7. break;
8. case 3:
9. System.out.println("3");
10. break;
11. case 4:
12. System.out.println("4");
13. break;
14. case 5:
15. System.out.println("5");
16. break;
17. case 6:
18. System.out.println("6");
19. break;
20. case 7:
21. System.out.println("7");
22. break;
23. case 8:
24. System.out.println("8");
25. break;
26. case 9:
27. System.out.println("9");
28. break;
29. default:
30. break;
31. }

Kode di atas berarti variabel input memiliki nilai integer **6**. Ketika variabel input kita masukkan ke dalam switch maka program akan masuk case **6**. Di dalam case **6**, program akan menampilkan string **“6”**, kemudian pemanggilan break akan menghentikan proses switch.

Tidak berbeda jauh dengan**if-then-else** bukan? Lalu pertanyaannya, kenapa harus ada **break**? Bagaimana jika tidak ada **break** ?  Sepertinya kita harus terlebih dahulu menelisik perbedaan dari **if-then** dan **switch**.

### Codelab Switch Tanpa Break

1. Bukalah kembali proyek **Percabangan** dan buatlah kelas baru dengan nama SwitchWithoutBreak.
2. Masukkan kode berikut ke dalam kelas SwitchWithoutBreak:
   1. package com.dicoding.javafundamental.percabangan;
   3. public class SwitchWithoutBreak {
   4. **public static void main(String[] args) {**
   5. **char input = 'C';**
   6. **int output = 0;**
   7. **switch (input) {**
   8. **case 'A':**
   9. **output++;**
   10. **case 'B':**
   11. **output++;**
   12. **case 'C':**
   13. **output++;**
   14. **case 'D':**
   15. **output++;**
   16. **case 'E':**
   17. **output++;**
   18. **default:**
   19. **output++;**
   20. **}**
   21. **System.out.println("Nilai akhirnya adalah " + output);**
   22. **}**
   23. }
3. Jika kode di atas di jalankan, kira-kira bagaimana ya hasil output-nya?  
   **1 atau 4 ?**
4. Jika Anda menjawab dengan **4** maka Anda benar. Jikat tidak percaya, jalankanlah kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Nilai akhirnya adalah 4 |

### Bedah Code Switch Tanpa Break

Bagaimana bisa output yang dihasilkan 4?

Case tanpa break akan menjalankan semua kode setelah case yang terpilih. Jadi dengan mengambil contoh kode di atas, dengan nilai inputan adalah char **'C'**, maka kode program akan masuk ke dalam case **'C'**. Di dalamnya nilai variabel output akan ditambah dengan 1 karena pemanggilan operator ++. Dan karena tidak ada break maka program akan melanjutkan menjalankan case setelahnya yaitu case **'D'**, case **'E'**, dan ***default***. Dan masing-masing di dalam case ada pemanggilan operator ++, sehingga nilai akhir dari output-nya adalah **4**.

Jika masih bingung perhatikan kembali kode di bawah ini yang sudah ditambahkan dengan komen agar lebih jelas.

1. package com.dicoding.javafundamental.percabangan;
3. public class SwitchWithoutBreak {
4. public static void main(String[] args) {
5. char input = 'C';
6. int output = 0;
7. switch (input) {
8. case 'A':
9. output++;
10. case 'B':
11. output++;
12. case 'C':
13. // Program akan menjalankan kode di dalam case ini
14. output++; // output = 1
16. // Karena di akhir case C tidak ada break, maka program akan menjalankan case selanjutnya
17. case 'D':
18. // Kode ini akan dijalankan oleh program
19. output++; // output = 2
20. case 'E':
21. // Kode ini akan dijalankan oleh program
22. output++; // output = 3
23. default:
24. // Case default juga akan dijalankan
25. output++; // output = 4
26. }
27. System.out.println("Nilai akhirnya adalah " + output);
28. }
29. }

Setelah mengenal **if-then** dan **switch**, lalu statement mana yang kita pilih?

Gunakanlah **if-else** ketika kondisinya merupakan range dari suatu nilai atau kondisi,. Sebabnya, karena **if-else** bisa menggunakan operator seperti **>** atau **<**. Sedangkan untuk  **switch**, gunakanlah ketika kondisinya hanya satu pilihan seperti **integer**, **char**, dan **tipe lainnya**.

# Perulangan

Sebagai Developer kita dituntut untuk membuat baris kode menjadi lebih efisien. Misalnya kita melakukan instruksi yang sama lebih dari sekali, seperti contoh di bawah ini:

1. System.out.print("Halo\n");
2. System.out.print("Halo\n");
3. System.out.print("Halo\n");

Output :

1. Halo
2. Halo
3. Halo

Baris kode di atas kita melakukan perintah yang sama sebanyak 3 kali**dengan menulis kode yang sama.** Hal ini sama ilustrasinya dengan mengirimkan pesan melalui email dengan alamat yang berbeda-beda. Tidak efisien, bukan? Memang kurang efisien kalau kita membuat baris kode dengan mengirimkan secara satu per satu. Dalam kasus seperti ini kita tidak harus mengulang baris kode yang sama secara terus-menerus. Lebih tepat, gunakan metode perulangan.

## Perulangan For

Perulangan memungkinkan kita melakukan instruksi secara berulang-ulang secara sederhana, jelas dan ringkas. Ada berbagai cara melakukan perulangan, namun yang paling populer adalah menggunakan **for**. Secara umum, bentuk pernyataan for di dalam Java adalah sebagai berikut:

1. for (**initialization; termination; increment**) {
2. **statement(s)**
3. }

Perhatikan contoh kode berikut:

1. System.out.print("Halo\n");
2. System.out.print("Halo\n");
3. System.out.print("Halo\n");

Dengan menggunakan perulangan, kita bisa menghasilkan output yang sama dari baris kode di atas dengan menggunakan salah satu metode perulangan:

1. for(int x = 0; x < 3; x++) {
2. System.out.print("Halo\n");
3. }

Output:

1. Halo
2. Halo
3. Halo

Sama output-nya, namun caranya terlihat lebih ringkas, bukan ?

### Codelab Perulangan For

Selanjutnya kita coba melakukan perulangan dengan menuliskan angka dari 1 hingga 10 dengan menggunakan for.

1. Buatlah proyek baru dengan nama **Perulangan** dengan nama package com.dicoding.javafundamental.perulangan di dalamnya:
2. Buatlah sebuah kelas baru dengan nama For, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.perulangan;
   3. public class For {
   4. public static void main(String[] args) {
   5. for (int i = 1; i <= 10; i++) {
   6. System.out.println("Angka : " + i);
   7. }
   8. }
   9. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Angka : 1 Angka : 2 Angka : 3 Angka : 4 Angka : 5 Angka : 6 Angka : 7 Angka : 8 Angka : 9 Angka : 10 |

### Bedah Code Perulangan For

Kita membuat variabel dengan tipe data integer yang kita sebut sebagai initialization. Dalam hal ini kita menginisialisasi angka dimulai dari **1**.

1. int i=1;

Pada tahap ini kita menentukan batasan nilai akhir suatu perulangan atau disebut dengan termination. Contoh di atas adalah angka lebih kecil atau sama dengan **10**.

1. i<=10;

Kita menentukan aksi terhadap perulangan, aksi tersebut bisa menaikkan (increment) atau menurunkan (decrement). Apabila kita ingin melakukan aksi increment maka nilai awal (initialization) harus lebih kecil daripada nilai akhir (termination). Begitupun sebaliknya, jika melakukan aksi menurun maka nilai awal (initialization) harus lebih besar daripada nilai akhir (termination). Seperti contoh di atas kita melakukan aksi increment.

1. i++;

Nah, apabila ketiga kondisi di atas sudah terpenuhi, pengulangan dapat tercipta. Bagaimana dengan varian implementasi dengan memanfaatkan **for?**

## Perulangan For Bersarang

**For** dapat menerapkan pengulangan bersarang (nested). Dengan kata lain kita membuat melakukan **for** di dalam **for**.

### Codelab Perulangan For Bersarang

Sebagai contoh kita akan membuat segitiga bintang legendaris dengan memanfaatkan for bersarang.

1. Bukalah kembali proyek **Perulangan** dan buatlah kelas baru dengan nama ForBersarang.
2. Masukkan kode berikut ke dalam kelas ForBersarang:
   1. public class ForBersarang {
   2. **public static void main(String[] args) {**
   3. **int a = 5;**
   4. **for (int i = 0; i <= a; i++) {**
   5. **for (int j = 0; j <= i; j++) {**
   6. **System.out.print("\*");**
   7. **}**
   8. **System.out.println("");**
   9. **}**
   10. **}**
   11. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| \* \*\* \*\*\* \*\*\*\* \*\*\*\*\* \*\*\*\*\*\* |

### Bedah Code Perulangan For Bersarang

Perhatikan kode berikut ini:

1. for (int i = 0; i <= a; i++) {
2. for (int j = 0; j <= i; j++) {
3. System.out.print("\*");
4. }
5. System.out.println("");
6. }

Inilah yang disebut dengan perulangan **for** bersarang. Perulangan ke pertama akan mempengaruhi perulangan kedua, begitu juga ketika ada perulangan ketiga, keempat dan seterusnya, maka akan dipengaruhi oleh perulangan sebelumnya.

## Perulangan While

Perulangan dengan menggunakan while digunakan jika  proses saat kondisi terpenuhi. While akan mengecek kesesuaian dengan kondisi pra perintah.

### Codelab Perulangan While

1. Bukalah kembali proyek **Perulangan** dan buatlah kelas baru dengan nama While.
2. Masukkan kode berikut ke dalam kelas While:
   1. package com.dicoding.javafundamental.perulangan;
   3. public class While {
   4. **public static void main(String[] args) {**
   5. **int value = 1;**
   6. **while (value <= 10) {**
   7. **System.out.print("Angka : " + value);**
   8. **value++;**
   9. **System.out.print("\n");**
   10. **}**
   11. **}**
   12. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Angka : 1 Angka : 2 Angka : 3 Angka : 4 Angka : 5 Angka : 6 Angka : 7 Angka : 8 Angka : 9 Angka : 10 |

### Bedah Code Perulangan While

Kita mendeklarasikan variabel nilai dengan angka 1.

1. int **value** = 1;

Pengecekan suatu kondisi pada variabel. Apabila nilai lebih kecil atau sama dengan 10.

1. while (**value** <= 10)

Melakukan sebuah perintah apabila kondisi terpenuhi.

1. System.out.print("Angka : " + **value**);
2. **value**++;
3. System.out.print("\n");

## Perulangan Do-While

Perulangan yang mempunyai fungsi yang sama dengan While, tetapi pengecekan kondisinya dilakukan di akhir. Pada perulangan ini minimal melaksanakan perintah sekali, kemudian mengecek kondisi.

### Codelab Perulangan Do-While

1. Bukalah kembali proyek **Perulangan** dan buatlah kelas baru dengan nama DoWhile.
2. Masukkan kode berikut ke dalam kelas DoWhile:
   1. package com.dicoding.javafundamental.perulangan;
   3. public class DoWhile {
   4. **public static void main(String[] args) {**
   5. **int value = 1;**
   6. **do {**
   7. **System.out.println("Angka : " + value);**
   8. **value++;**
   9. **} while (value <= 10);**
   10. **}**
   11. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Angka : 1 Angka : 2 Angka : 3 Angka : 4 Angka : 5 Angka : 6 Angka : 7 Angka : 8 Angka : 9 Angka : 10 |

### Bedah Code Perulangan Do-While

Bagaimana bisa menghasilkan output seperti di atas? Mari kita bahas bersama-sama:

1. Kita mendeklarasikan variabel nilai dengan angka 1.
   1. int value = 1;
2. Melakukan sebuah perintah terlebih dahulu.
   1. System.out.println("Angka : " + value);
   2. value++;
3. Pengecekan suatu kondisi terhadap variabel. Apabila terpenuhi maka perintah dilanjutkan.
   1. while (value <= 10)

Nah, kita sudah mencoba macam-macam perulangan pada Java. Penggunaan metode perulangan tergantung pada kasus dan mengaplikasiannya. Bisa jadi salah satu metode, paling cocok diimplementasikan pada kasus tertentu. Metode mana yang paling kamu suka?

# Pengantar Collection Type

Collection adalah kumpulan interface yang digunakan sebagai wadah untuk mengumpulkan beberapa elemen menjadi satu kesatuan. Dari definisi tersebut bisa ditafsirkan Collection mirip dengan Array karena keduanya merepresentasikan struktur data. Java Collections Framework terdiri dari tiga komponen, yaitu:

1. **Interface**: Tipe abstrak dari Collection yang membentuk suatu hierarki.
2. **Implementation**: Tipe konkrit yang mengimplementasikan Interface dari Collection serta dan merupakan suatu struktur data yang bisa langsung digunakan.
3. **Algorithm**: Ini adalah method yang melakukan komputasi berguna, misal sorting atau searching.

Dari komponen pembentuk Java Collections Framework bisa kita lihat salah satu perbedaan Collection dengan Array. Misalnya Array tidak mempunyai method untuk sorting atau jumlah elemen di Array tidak bisa dinamis. Sementara Collection bisa dinamis.

Dapat disimpulkan bahwa Collection adalah struktur data yang lebih kompleks dan lebih canggih dibanding Array. Untuk itu setiap programmer wajib menguasai Java Collections Framework. Sebabnya, kita akan selalu berurusan dengan data saat membuat aplikasi.

Core Collection Interface dapat dirangkum pada gambar di bawah ini.

Gambar diambil dari <https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html>

Dalam materi ini kita hanya akan membahas Java Collections Framework yang paling sering digunakan yaitu **List, Set**, dan **Map**. Sebagian Implementasi dari ketiga interface tersebut bisa kita lihat di tabel di bawah ini.

|  |  |  |
| --- | --- | --- |
| **List** | **Set** | **Map** |
| ArrayList LinkedList | TreeSet HashSet LinkedHashSet | TreeMap HashMap LinkedHashMap |

## Codelab Collection

Untuk dapat melihat hierarki dari Java Collections Framework secara lebih lengkap kita bisa menggunakan fitur **Type Hierarchy**di IntelliJ.

1. Buatlah proyek baru dengan nama **Collection** dengan nama package com.dicoding.javafundamental.collection di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama Main, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.collection;
   3. import java.util.Collection;
   5. public class Main {
   7. public static void main(String[] args) {
   8. Collection collection;
   9. }
   10. }
3. Blok deklarasi interface Collection lalu tekan **ctrl + h** (atau melalui menu **Navigate** - Type **Hierarchy**). Akan muncul panel Hierarchy dari interface Collection.
4. Perhatikan panel Hierarchy. Ada banyak subtype atau turunan dari interface Collection. Fitur **Type Hierarchy** di IntelliJ sangat membantu programmer untuk melihat struktur atau hierarki inheritance dari interface atau class. Selain itu ada fitur **Method Hierarchy**untuk melihat struktur inheritance dari method dan **Call Hierarchy** yang fungsinya untuk melihat di mana saja suatu method dipanggil (invoked).
5. Di panel Hierarchy terlihat interface Collection mempunyai turunan, salah satunya **List** dan **Set**sesuai dengan gambar Core Collection Interface di atas.

## List

List adalah suatu Collection di mana data yang masuk akan disimpan secara teratur. **List mempunyai index** berdasarkan urutan objek yang dimasukkan ke dalam List. Objek di dalam List bisa duplikat artinya objek yang sama bisa dimasukkan beberapa kali ke dalam List dan akan mempunyai index yang berbeda. **Objek dikatakan sama jika dibandingkan menggunakan method equals() menghasilkan nilai true**. Hal ini akan berbeda dengan **Set** yang akan kita bahas berikutnya. Salah satu implementasi dari interface List adalah class **ArrayList**. ArrayList mirip dengan Array hanya saja ArrayList ukurannya tidak tetap atau fleksibel.

### Codelab List

Yuk mari kita langsung coding untuk melihat contoh penggunaan List.

1. Bukalah kembali proyek **Collection.**
2. Buatlah kelas baru dengan nama ListPlanet dan masukkan kode berikut ke dalamnya:
   1. package com.dicoding.javafundamental.collection;
   3. **import java.util.ArrayList;**
   4. **import java.util.Collection;**
   5. **import java.util.List;**
   7. public class ListPlanet {
   9. **public static void main(String[] args) {**
   10. **// deklarasi Array**
   11. **String[] heroes = new String[2];**
   12. **heroes[0] = "riki";**
   13. **heroes[1] = "sven";**
   14. **//heroes[2] = "zeus"; // error ArrayIndexOutOfBoundsException karena ukuran Array tetap**
   15. **// menggunakan ArrayList**
   16. **List<String> planets = new ArrayList<>();**
   17. **planets.add("mercury"); // method add() untuk menambahkan objek ke List**
   18. **planets.add("venus");**
   19. **planets.add("earth");**
   20. **planets.add("mars"); // objek lainnya masih bisa terus ditambahkan ke List**
   21. **System.out.println("List planets awal:");**
   22. **for (int i = 0; i < planets.size(); i++) { // method size() untuk mendapatkan ukuran List**
   23. **// method get() untuk melihat isi List pada index i**
   24. **System.out.println("\t index-"+ i +" = " + planets.get(i));**
   25. **}**
   26. **planets.remove("venus"); // method remove() untuk mengeluarkan objek dari List**
   27. **System.out.println("List planets akhir:");**
   28. **for (int i = 0; i < planets.size(); i++) {**
   29. **System.out.println("\t index-"+ i +" = " + planets.get(i));**
   30. **}**
   31. **}**
   32. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| List planets awal: index-0 = mercury index-1 = venus index-2 = earth index-3 = mars List planets akhir: index-0 = mercury index-1 = earth index-2 = mars |

## Set

Set mirip dengan List hanya saja **objek di dalam Set tidak bisa diduplikasi.** Yang artinya jika objek yang sama dimasukkan beberapa kali ke dalam Set maka Set hanya akan menyimpan objek tersebut satu kali saja. Coba ingat kembali materi Inheritance bagian Overriding dan Overloading. Perbedaan lain antara Set dengan List adalah penyimpanan pada Set tidak teratur.

### Codelab Set

Ayo langsung coding biar lebih asik.

1. Bukalah kembali proyek **Collection.**
2. Buatlah kelas baru dengan nama SetPlanet dan masukkan kode berikut ke dalamnya:
   1. package com.dicoding.javafundamental.collection;
   3. import java.util.\*;
   5. public class SetPlanet {
   7. public static void main(String[] args) {
   8. // menggunakan HashSet
   9. Set<String> planets = new HashSet<>();
   10. planets.add("mercury"); // method add() untuk menambahkan objek ke Set
   11. planets.add("venus");
   12. planets.add("earth");
   13. planets.add("earth"); // menambahkan objek "earth" beberapa kali
   14. planets.add("earth");
   15. planets.add("mars"); // objek bisa terus ditambahkan ke Set
   17. // method size() untuk mendapatkan ukuran Set
   18. System.out.println("Set planets awal: (size = " + planets.size() + ")");
   19. for (String planet : planets) {
   20. System.out.println("\t " + planet);
   21. }
   23. planets.remove("venus"); // method remove() untuk mengeluarkan objek dari Set
   25. System.out.println("Set planets akhir: (size = " + planets.size() + ")");
   26. for (Iterator iterator = planets.iterator(); iterator.hasNext(); ) {
   27. // looping menggunakan Iterator
   28. System.out.println("\t " + iterator.next());
   29. }
   30. }
   31. }
3. Outputnya akan seperti di bawah ini. Perhatikan objek “**earth**” hanya ada satu kali dan perhatikan juga outputnya tidak sesuai dengan urutan ketika kita menambahkan objek ke Set (menggunakan method add()). Ini yang dimaksud penyimpanan pada Set tidak teratur.

|  |
| --- |
| Set planets awal: (size = 4) venus mercury mars earth Set planets akhir: (size = 3) mercury mars earth |

## Map

Map adalah struktur data dalam bentuk pasangan **key-value**. Map sama dengan “associative array “ dalam bahasa PHP. Objek disimpan di Map sebagai value menggunakan key yang harus unik dan harus berupa objek juga. Salah satu implementasi dari interface Map adalah class **HashMap**.

### Codelab Map

Untuk lebih mengenal Map mari kita langsung coding.

1. Bukalah kembali proyek **Collection.**
2. Buatlah kelas baru dengan nama Planet dan masukkan kode berikut ke dalamnya:
   1. package com.dicoding.javafundamental.collection;
   3. class Planet {
   4. **private String name;**
   5. **private double mass;**
   6. **public Planet(String name, double mass) {**
   7. **this.name = name;**
   8. **this.mass = mass;**
   9. **}**
   10. **@Override**
   11. **public String toString() {**
   12. **return "Planet " + name + ", mass: " + mass;**
   13. **}**
   14. }
3. Buatlah kelas MapPlanet dan tambahkan kode berikut di dalamnya:
   1. package com.dicoding.javafundamental.collection;
   3. import java.util.HashMap;
   4. import java.util.Map;
   6. public class MapPlanet{
   8. public static void main(String[] args) {
   9. // menggunakan HashMap
   10. Map<String, Planet> planets = new HashMap();
   11. planets.put("key-1", new Planet("Mercury", 0.06)); // method put() untuk menambahkan objek ke Map
   12. planets.put("key-2", new Planet("Venus", 0.82));
   13. planets.put("key-3", new Planet("Earth", 1.00));
   14. planets.put("key-4", new Planet("Mars", 0.11));
   15. planets.put("key-4", new Planet("Mars-X", 0.11)); // menambahkan ke Map dengan key yang sama
   17. // method size() untuk mendapatkan ukuran Set
   18. System.out.println("Map planets awal: (size = "+ planets.size() +")");
   19. for (String key : planets.keySet()) { // looping key dari Map
   20. // method get() untuk melihat isi Map berdasarkan key
   21. System.out.println("\t " + key + " : " + planets.get(key));
   22. }
   24. planets.remove("key-2"); // method remove() untuk mengeluarkan objek dari Map
   26. System.out.println("Map planets akhir: (size = "+ planets.size() +")");
   27. for (Planet planet : planets.values()) { // looping value dari Map
   28. System.out.println("\t " + planet);
   29. }
   30. }
   31. }
4. Outputnya akan seperti di bawah ini. Perhatikan. Jika menambahkan dengan key yang sama maka value akan diganti. Lebih lanjut, sama halnya pada Set, objek yang dimasukkan ke Map tidak teratur urutannya.

|  |
| --- |
| Map planets awal: (size = 4) key-1 : Planet Mercury, mass: 0.06 key-4 : Planet Mars-X, mass: 0.11 key-3 : Planet Earth, mass: 1.0 key-2 : Planet Venus, mass: 0.82 Map planets akhir: (size = 3) Planet Mercury, mass: 0.06 Planet Mars-X, mass: 0.11 Planet Earth, mass: 1.0 |

Pada dasarnya Collection adalah struktur data. Makin kompleks struktur datanya maka akan dibutuhkan implementasi Collection yang lebih kompleks lagi. Pada materi ini kita hanya membahas Collection yang sudah ada di JDK tanpa menambahkan library apapun. Di luar sana ada banyak library Collection untuk keperluan yang kompleks misal Apache Common Collection dan Google Collection(sekarang menjadi bagian dari Guava).

# Pengantar Object-Oriented Programming

Apa itu OOP? OOP atau (Object-Oriented Programming) adalah suatu paradigma dalam pemrograman yang berbasis pada obyek. OOP terinsiprasi dari model yang telah ada di kehidupan sehari-hari. Contoh saja seperti mobil, mobil merupakan suatu objek. Mobil terbentuk dari beberapa objek yang lebih kecil seperti roda, body, mesin dll. Objek yang terdapat pada mobil saling berhubungan, berinteraksi dan berkomunikasi. Melalui konsep di atas, OOP diciptakan untuk mempermudah pengembangan program dengan cara mengikuti pendekatan model objek yang telah ada dikehidupan sehari-hari. Bagi yang ingin mempelajari bahasa pemrograman Java tentunya harus memahami konsep OOP, karena pemrograman Java didesain dengan menerapkan konsep OOP.

Berikut ini adalah beberapa istilah yang sering digunakan dalam konsep OOP.

1. **Class dan Object**  
   Kelas (terjemahan dari class) dan obyek (terjemahan dari object) adalah hal paling dasar yang harus kita pelajari. Kita akan belajar bagaimana membuat suatu kelas dan menggunakaanya di dalam suatu program.
2. **Method**  
   Suatu metode mendefinisikan perilaku obyek yang dibuat dari kelas. Metode dapat berupa prosedur atau fungsi.
3. **Member**  
   Di materi ini kita akan belajar lebih dalam lagi tentang member (semua yang dideklarasikan) di dalam suatu kelas.
4. **Access Modifier**  
   Di sini kita akan belajar tentang hak akses dari suatu kelas, metode, dan data member. Ada beberapa macam modifier yaitu private, default, protected, dan public.
5. **Encapsulation**  
   Encapsulation adalah metode untuk membundling suatu data dan metode di dalam kelas. Di dalamnya juga akan ada materi tentang bagaimana mengaksesnya dengan memanfaatkan metode di dalam kelas.
6. **Inheritance**  
   Inheritance adalah pewarisan, suatu kelas dapat diturunkan dari kelas lain sehingga mewarisi data dan metode dari kelas lain tersebut. Materi inheritance sangat penting di dalam OOP.
7. **Interface**  
   Selain inheritance, kita juga akan belajar tentang interface. Interface mirip seperti kelas, tetapi hanya berisi method signature (nama, parameter dan return).

# Class

Class merupakan bagian dari turunan obyek yang berfungsi sebagai wadah atau cetakan untuk mendeskripsikan behavior/state yang didukung obyeknya. Behavior adalah tingkah laku yang dapat dilakukan oleh obyek. Behavior adalah kelakuan atau sifat dari obyek,  misalnya lampu bisa dimatikan atau dinyalakan. State adalah kondisi atau keadaan yang bisa terjadi pada obyek misalnya lampu dapat memiliki state nyala atau mati.

Di kehidupan nyata kita mungkin sering menemukan banyak obyek, akan tetapi beberapa di antaranya memiliki kemiripan. Mungkin ada banyak kucing yang ada di dunia ini. Semua sama-sama memiliki bentuk yang sama. Bila diperhatikan ada banyak jenis kucing, mulai dari warna, bentuk hingga ras. Dalam aturan orientasi obyek, dapat dikatakan bahwa jenis, warna dan ras kucing di seluruh belahan dunia ini adalah sebuah kelas **instance** dari obyek yang dikenali sebagai kucing.

1. package com.dicoding.javafundamental.kelas;
3. public class KucingAnggora {
4. // class body
5. }

Dalam aturan penulisan yang dianjurkan pada pemrograman Java, nama kelas diawali dengan huruf besar dan diikuti huruf kecil (contoh: Kucing). Bila suatu kelas terdiri dari banyak kata maka huruf pertama di setiap kata tersebut diberikan huruf besar (contoh: KucingAnggora).

## Constructor

Constructor adalah method yang secara default sudah terbentuk ketika kelas dibuat. Ketika suatu kelas dibuat (instansiasi) maka konstruktor akan terpanggil juga. Constructor harus memiliki nama yang sama dengan nama kelasnya. Lalu apa kegunaan Constructor?

* Memberikan nilai awal terhadap suatu data atau obyek
* Mengalokasikan ruang sebuah kelas dalam obyek

Terdapat dua tipe constructor pada Java:

* **No-argument constructor**, apabila kita tidak mendeklarasikan konstruktor dalam suatu kelas, kompiler Java akan membuat default constructor(tanpa parameter) dan pada body constructor tersebut kosong. Sementara jika kita mendeklarasikan konstruktor tanpa parameter , baik menuliskan kode pada body-nya atau tidak, maka itu bukan default constructor. Itu adalah no-argument constructor.  
  Berikut ini adalah contoh **no-argument constructor**pada suatu kelas.
  1. package com.dicoding.javafundamental.kelas;
  3. public class KucingAnggora {
  5. *//ini adalah contoh no-argument constructor*
  6. **public KucingAnggora() {**
  7. **}**
  8. }
* **Parameterized constructor**, konstruktor yang memiliki parameter untuk menginisiasi atau memberikan nilai awal data dengan nilai yang kita tentukan.  
  Berikut ini adalah contoh **parameterized constructor**pada suatu kelas.
  1. package com.dicoding.javafundamental.kelas;
  3. public class KucingAnggora {
  5. **String namaKucing;**
  6. **int beratBadan;**
  8. *//ini adalah contoh parameterized constructor*
  9. *//dengan memberikan nilai awal data member nama\_kucing dan berat\_badan*
  10. *//dengan parameter yang kita tentukan*
  11. **public KucingAnggora(String nama, int berat) {**
  12. **namaKucing = nama;**
  13. **beratBadan = berat;**
  14. **}**
  15. }

# Object

Dalam paradigma OOP, objek merupakan instance atau wujud nyata dari suatu kelas. Pemahaman mengenai objek tidak kalah pentingnya pada konsep pemrograman berorientasi objek. Salah satu kegunaan objek adalah untuk mengakses berbagai fungsi dan metode pada kelas. Objek bisa dibuat lebih dari satu.

## Menciptakan Objek

Karena objek merupakan instansiasi dari kelas, untuk menciptakan suatu objek kita perlu mendeklarasikan tiap objek yang dibuat dengan mengikutsertakan kelasnya. Pada Java untuk membuat sebuah objek terdapat 3 tahap :

* **Declaration :**mendeklarasikan nama sebuah objek.
* **Instantiation :**memerlukan sebuah perintah new untuk menciptakan objek.
* **Initialization :**inisialisasi dari sebuah objek setelah perintah new.

### Codelab Mencipatakan Objek

Pada contoh berikut kita praktikkan cara menciptakan suatu objek secara sederhana di dalam satu kelas.

1. Buatlah proyek baru dengan nama **Objek** dengan nama package com.dicoding.javafundamental.objek di dalamnya. Kemudian buatlah sebuah kelas baru dengan nama Hewan, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.objek;
   3. public class Hewan {
   4. **public void cetakNama(String nama) {**
   5. **System.out.println("Nama hewan: " + nama);**
   6. **}**
   7. }
2. Setelah itu, buatlah kelas baru dengan nama Main dan masukkan kode berikut di dalamnya:
   1. public class Main {
   2. **public static void main(String[] args) {**
   3. **// menciptakan object dengan nama 'objekHewan'**
   4. **Hewan objekHewan = new Hewan();**
   5. **objekHewan.cetakNama("Elang");**
   6. **}**
   7. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Nama hewan: Elang |

1. Nah, sampai di sini apakah Anda sudah paham ? Jika belum mari kita bahas kodenya secara jelas.

### Bedah Code Menciptakan Objek

Pada kode di atas kita mempunyai sebuah kelas bernama Hewan. Di dalam class tersebut kita mempunyai metode cetakNama yang mempunyai parameter nama. Pada metode cetakNama kita hanya melakukan perintah menampilkan berdasarkan parameter yang dikirim dengan menggunakan tipe data String.

1. public void cetakNama(String nama) {
2. System.out.println("Nama hewan: " + nama);
3. }

Pada kode selanjutnya kita menciptakan suatu objek dengan perintah new. Sedangkan objekHewan merupakan nama dari objek yang kita buat.

1. Hewan objekHewan = new Hewan();

Untuk memanggil sebuah metode di dalam suatu kelas, kita dapat menggunakan (.) (titik).

1. objekNama.cetakNama("Elang");

Sampai di sini, kita telah mengerti bagaimana membuat objek secara sederhana. Tahap selanjutnya kita akan mencoba membuat objek pada sebuah kelas yang lebih kompleks.

### Codelab Menggunakan Objek

Selanjutnya kita akan coba menggunakan objek yang ada di kelas Hewan.

1. Bukalah kembali proyek **Objek**. Kemudian masukkan kode berikut ke dalam kelas Hewan:
   1. package com.dicoding.javafundamental.objek;
   3. public class Hewan {
   4. **String nama;**
   5. **int berat;**
   6. **int jumlahKaki;**
   7. **// Ini merupakan konstruktor dari kelas hewan. konstruktor adalah method yang pertama kali dijalankan ketika object diciptakan.**
   8. **public Hewan(String namaHewan) {**
   9. **nama = namaHewan;**
   10. **}**
   11. **public void beratHewan(int beratHewan) {**
   12. **berat = beratHewan;**
   13. **}**
   14. **public void jumlahKakiHewan(int jumlahKakiHewan) {**
   15. **jumlahKaki = jumlahKakiHewan;**
   16. **}**
   18. public void cetakHewan() {
   19. System.out.println("Nama hewan : " + nama);
   20. **System.out.println("Berat hewan : " + berat + " kg");**
   21. **System.out.println("Jumlah kaki: " + jumlahKaki);**
   22. }
   23. }

Kelas Hewan memliki beberapa atribut, seperti nama, berat dan jumlah kaki. Masing-masing dari atribut tersebut kita berikan parameter dengan membuat method.

1. Ubahlah kode yang ada di dalam kelas Main:
   1. package com.dicoding.javafundamental.objek;
   3. public class Main {
   4. public static void main(String[] args) {
   5. **Hewan Elang = new Hewan("Elang");**
   6. **Hewan Kucing = new Hewan("Kucing");**
   7. **Elang.beratHewan(5);**
   8. **Elang.jumlahKakiHewan(2);**
   9. **Elang.cetakHewan();**
   10. **Kucing.beratHewan(2);**
   11. **Kucing.jumlahKakiHewan(4);**
   12. **Kucing.cetakHewan();**
   13. }
   14. }
2. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Nama hewan : Elang Berat hewan : 5 kg Jumlah kaki: 2 Nama hewan : Kucing Berat hewan : 2 kg Jumlah kaki: 4 |

Kini kita telah memahami cara membuat kelas dan menginstansiasinya pada Java. Ada yang perlu kita tekankan dalam bab ini. Dalam sebuah kelas kita bisa menciptakan beberapa objek. Dengan menggunakan objek kita dapat mengeksekusi baik metode maupun fungsi yang beraneka ragam, yang dipakai dalam sebuah kelas.

# Function

Fungsi pada Java dideklarasikan di dalam sebuah kelas. Fungsi pada Java bertugas untuk mengembalikan nilai. Sedangkan untuk fungsi yang tidak mengembalikan nilai (void) disebut sebagai prosedur. Baik fungsi maupun prosedur di dalam suatu kelas kita sebut sebagai metode. Metode merupakan aspek yang penting di dalam Java. Dalam sebuah kelas bisa terdapat banyak metode sesuai dengan kegunaannya masing-masing.

## Mendeklarasikan Fungsi

Cara membuat sebuah fungsi di dalam kelas adalah sebagai berikut :

1. modifier returnType nameOfFunction(parameters) {
3. }

* **modifier**menunjukkan sifat yang dimiliki pada suatu fungsi seperti public, private, protected.
* **returnType**merupakan nilai balik yang diberikan oleh fungsi. Apabila fungsi tidak memiliki nilai balik maka menggunakan void.
* **nameOfFunction** adalah nama dari sebuah fungsi.
* **parameters** bersifat opsional, Suatu fungsi dapat mempunyai banyak parameters atau pun tidak sama sekali.

## Memanggil Fungsi

Berikut adalah contoh cara mengimplementasikan fungsi secara sederhana di dalam sebuah kelas secara statis. Artinya penggunaan fungsi ini hanya dilakukan pada kelas tersebut tanpa membuat objek.

### Codelab Memanggi Fungsi

Mari kita coba memanggil fungsi di dalam program.

1. Buatlah proyek baru dengan nama **Function** dengan nama package com.dicoding.javafundamental.function di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama CallFunction, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.function;
   3. public class CallFunction {
   5. public static void main(String[] args) {
   6. // memanggil fungsi
   7. **cobaFungsi();**
   8. }
   10. **public static void cobaFungsi() {**
   11. **System.out.println("Ini merupakan bagian dari fungsi");**
   12. **}**
   13. }
3. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Ini merupakan bagian dari fungsi |

## Fungsi dengan nilai balik

Jika contoh kode sebelumnya tanpa  nilai balik, lalu bagaimana untuk fungsi yang mengirimkan/menghasilkan nilai balik? Fungsi yang yang mengirimkan/menghasilkan nilai balik selalu mendeklarasikan tipe data kemudian diakhiri dengan return.

### Codelab Fungsi dengan Nilai Balik

Mari kita coba memanggil fungsi dengan nilai balik di dalam program.

1. Bukalah kembali proyek **Function** dan buatlah kelas baru dengan nama ReturnFunction.
2. Masukkan kode berikut ke dalam kelas ReturnFunction:
   1. package com.dicoding.javafundamental.function;
   3. public class ReturnFunction {
   4. **public static void main(String[] args) {**
   5. **double p = 7;**
   6. **double l = 6.5;**
   7. **double hasil = hitungLuas(p, l); //memanggil fungsi**
   8. **System.out.println("Hasilnya adalah = " + hasil);**
   9. **}**
   10. **//fungsi dengan nilai balik**
   11. **public static double hitungLuas(double panjang, double lebar){**
   12. **double luas = panjang \* lebar;**
   13. **return luas;**
   14. **}**
   15. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Hasilnya adalah = 45.5 |

1. **Catatan** : Penamaan suatu fungsi dapat disesuaikan sesuai kebutuhan. Namun, secara umum penamaan sebuah fungsi diawali dengan huruf kecil dan diikuti dengan huruf besar setelahnya. Hal ini biasa disebut dengan notasi ”punuk unta” seperti contoh:
   1. luas
   2. hitungKeliling
   3. hapusUser
   4. rataNilaiMahasiswa
   5. jumlahKomentarNetizen
   6. jumlahLaguYoungLex

## Fungsi dengan parameter

Fungsi ini bertujuan untuk menghitung luas persegi panjang dengan menggunakan parameter.

### Codelab Fungsi dengan Parameter

Mari kita praktikan agar semakin paham tentang pemanggilan fungsi dengan parameter.

1. Bukalah kembali proyek **Function** dan buatlah kelas baru dengan nama InputFunction.
2. Masukkan kode berikut ke dalam kelas InputFunction:
   1. package com.dicoding.javafundamental.function;
   3. public class InputFunction {
   4. **public static void main(String[] args) {**
   5. **// memanggil fungsi**
   6. **hitungLuas(7, 6.5);**
   7. **}**
   8. **public static void hitungLuas(double panjang, double lebar) {**
   9. **double luas;**
   10. **luas = panjang \* lebar;**
   11. **System.out.println(luas);**
   12. **}**
   13. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| 45.5 |

### Bedah Kode Fungsi dengan Parameter

Perhatikan kode berikut:

1. public static void hitungLuas(double panjang, double lebar) {
2. double luas;
3. luas = panjang \* lebar;
4. System.out.println(luas);
5. }

Pada kode di atas kita membuat fungsi hitungLuas dengan menggunakan paramater serta tidak memberikan nilai balik. Maka dari itu kita menggunakan void. Kemudian fungsi dipanggil dengan memberikan nilai pada masing-masing parameter  serta dipisahkan oleh tanda koma (**,**).

## Method Overloading

Java mengizinkan menggunakan dua atau lebih fungsi dengan nama yang sama dalam satu kelas. Namun, yang membedakan adalah parameternya. Konsep ini disebut dengan method overloading. Sebagai contoh kita membuat fungsi dengan nama yang sama pada kode sebelumnya dengan membedakan tipe datanya menjadi int.

### Codelab Method Overloading

Mari kita praktikan agar semakin paham tentang Method Overloading.

1. Bukalah kembali proyek **Function** dan buatlah kelas baru dengan nama MethodOverloading.
2. Masukkan kode berikut ke dalam kelas MethodOverloading:
   1. package com.dicoding.javafundamental.function;
   3. public class MethodOverloading {
   4. **public static void main(String[] args) {**
   5. **double p = 7;**
   6. **double l = 6.5;**
   7. **double hasil = hitungLuas(p, l);**
   8. **System.out.println("Hasilnya adalah = " + hasil);**
   9. **int pn = 7;**
   10. **int lb = 6;**
   11. **int hsl = hitungLuas(pn, lb);**
   12. **System.out.println("Hasilnya adalah = " + hsl);**
   13. **}**
   14. **public static double hitungLuas(double panjang, double lebar) {**
   15. **double luas = panjang \* lebar;**
   16. **return luas;**
   17. **}**
   18. **public static int hitungLuas(int panjang, int lebar) {**
   19. **int luas = panjang \* lebar;**
   20. **return luas;**
   21. **}**
   22. }
3. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Hasilnya adalah = 45.5 Hasilnya adalah = 42 |

Kini kita telah mengetahui cara mendeklarasikan dan menggunakan sebuah fungsi. Dengan menggunakan fungsi maka kita bisa jauh lebih efisien  dalam  menulis kode . Tidak lagi menulis secara berulang-ulang. Pada tahap selanjutnya kita akan belajar membuat fungsi pada kelas dan memanggilnya dengan menggunakan objek.

# Properti dan Metode

Di dalam OOP  suatu kelas bisa memiliki banyak properti dan banyak metode. Jika dianalogikan dengan kelas hewan, maka hewan memiliki properti antara lain tinggi, berat, dan umur. Hewan juga memiliki metode antara lain untuk jalan, lari, dan makan.

## Properti

Properti atau fields adalah atribut yang menjadi anggota dari suatu kelas. Properti digunakan untuk menyimpan data yang relevan dengan kelas. Seperti yang sudah disebutkan sebelumnya seperti tinggi, berat, dan umur adalah properti dari suatu kelas.

### Codelab Properti

Untuk memudahkan pemahaman materi, mari kita praktikan:

1. Buatlah proyek baru dengan nama **PropertiDanMetode** dengan nama package com.dicoding.javafundamental.propertimetode di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama Hewan, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.propertimetode;
   3. public class Hewan {
   5. **// Properti atau fields**
   6. **double tinggi;**
   7. **double berat;**
   8. **int umur;**
   9. }
3. Kita bisa melakukan inisiasi dari properti dengan menggunakan nilai inisial atau parameter dari konstruktor. Misalnya seperti ini:
   1. package com.dicoding.javafundamental.propertimetode;
   3. public class Hewan{
   5. **// Properti atau fields**
   6. **// Inisiasi dengan nilai inisial**
   7. **double tinggi = 30;**
   8. **double berat = 3;**
   9. **// Inisiasi melalui konstruktor**
   10. **int umur;**
   11. **// Konstruktor dengan parameter**
   12. **Hewan(int umurParam) {**
   13. **umur = umurParam;**
   14. **}**
   15. }
4. Perhatikan parameter umurParam dan umur**.**Pada dasarnya keduanya adalah variabel yang bertipe sama, tapi kedudukannya berbeda. Variabel umurParam adalah **parameter**, sedangkan variabel umuradalah **properti**. Agar lebih praktis maka kita bisa samakan nama variabelnya dan gunakan kode thisuntuk membedakan kedudukannya. Seperti ini misalnya.
   1. package com.dicoding.javafundamental.propertimetode;
   3. public class Hewan {
   4. // Properti atau fields
   6. // Inisiasi dengan nilai inisial
   7. double tinggi = 30;
   8. double berat = 3;
   10. // Inisiasi melalui konstruktor
   11. int umur;
   13. // Konstruktor dengan parameter
   14. Hewan(**int umur**) {
   15. **this.umur = umur;**
   16. }
   17. }

Kode thismereferensikan objek yang sedang digunakan. Pemanggilan thisada di dalam konstruktor kelas Hewan. Artinya nilai thismereferensikan objek Hewan. Kesimpulannya kode this.umur adalah akses ke atribut umur kelas Hewan dalam kelas Hewan.

## Metode

Metode di dalam kelas adalah block statement yang memiliki nama dan bisa dieksekusi dengan memanggilnya. Pemanggilan metode biasa disebut dengan “**invoke**”.

### Codelab Metode

Sebagai contoh, kita gunakan kembali kelas Hewan dengan menambahkan beberapa metode.

1. Bukalah kembali proyek **PropertiDanMetode.**
2. Bukalah kelas Hewan dan masukkan kode berikut ke dalam-nya:
   1. package com.dicoding.javafundamental.propertimetode;
   3. class Hewan{
   4. // Properti atau fields
   6. // Inisiasi dengan nilai inisial
   7. double tinggi = 30;
   8. double berat = 3;
   10. // Inisiasi melalui konstruktor
   11. int umur;
   13. // Konstruktor dengan parameter
   14. Hewan(int umur) {
   15. this.umur = umur;
   16. }
   18. **void lari(){**
   19. **System.out.println("Berlari dengan sangat cepat..");**
   20. **}**
   21. **void jalan(){**
   22. **System.out.println("Berjalan dengan pelan..");**
   23. **}**
   24. **void makan(){**
   25. **System.out.println("Makan dengan menggunakan mulut..");**
   26. **}**
   27. }
3. Buatlah kelas baru dengan nama Main. Kemudian untuk meng-invoke ketiga metode di atas, seperti ini caranya:
   1. package com.dicoding.javafundamental.propertimetode;
   3. public class Main {
   4. public static void main(String[] args) {
   6. **// Kita tambahkan 1 argumen dengan nilai int 2**
   7. **// Nilai tersebut adalah nilai yang digunakan untuk inisiasi variabel umur**
   8. **Hewan kucing = new Hewan(2);**
   9. **kucing.makan();**
   10. **kucing.jalan();**
   11. **kucing.lari();**
   12. }
   13. }
4. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Makan dengan menggunakan mulut.. Berjalan dengan pelan.. Berlari dengan sangat cepat.. |

### Bedah Code Metode

1. void lari(){
2. System.out.println("Berlari dengan sangat cepat..");
3. }
5. void jalan(){
6. System.out.println("Berjalan dengan pelan..");
7. }
9. void makan(){
10. System.out.println("Makan dengan menggunakan mulut..");
11. }

Jika Anda perhatikan, 3 metode di atas menggunakan void sebelum nama metodenya. Ini menandakan bahwa metode tersebut tidak memiliki nilai balik. Apa itu nilai balik? Nilai balik adalah nilai yang dikirim oleh suatu fungsi.

### Codelab Fungsi (Metode dengan Nilai Balik)

Pada dasarnya fungsi adalah metode yang memberikan nilai balik. Nilai balik tersebut bisa kita lihat pada penggunaan tipe data sebagai pengganti dari void, dan kode return. Contohnya seperti ini.

1. Bukalah kembali proyek **PropertiDanMetode.**
2. Bukalah kelas Hewan dan masukkan kode berikut ke dalam-nya:
   1. package com.dicoding.javafundamental.propertimetode;
   3. class Hewan {
   4. // Properti atau fields
   6. // Inisiasi dengan nilai inisial
   7. double tinggi = 30;
   8. double berat = 3;
   10. // Inisiasi melalui konstruktor
   11. int umur;
   13. // Konstruktor dengan parameter
   14. Hewan(int umur) {
   15. this.umur = umur;
   16. }
   18. void lari() {
   19. System.out.println("Berlari dengan sangat cepat..");
   20. }
   22. void jalan() {
   23. System.out.println("Berjalan dengan pelan..");
   24. }
   26. void makan() {
   27. System.out.println("Makan dengan menggunakan mulut..");
   28. }
   30. **double getBerat() {**
   31. **return berat;**
   32. **}**
   33. **double getTinggi() {**
   34. **return tinggi;**
   35. **}**
   36. **int getUmur() {**
   37. **return umur;**
   38. **}**
   39. }
3. Sekarang, mari kita coba invoke fungsi yang telah kita buat. Bukalah kelas Main dan tambahkan kode berikut:
   1. package com.dicoding.javafundamental.propertimetode;
   3. public class Main {
   4. public static void main(String[] args) {
   6. // Kita tambahkan 1 argumen dengan nilai int 2
   7. // Nilai tersebut adalah nilai yang digunakan untuk inisiasi variabel umur
   8. Hewan kucing = new Hewan(2);
   9. kucing.makan();
   10. kucing.jalan();
   11. kucing.lari();
   13. **// Contoh penggunaan fungsi getUmur**
   14. **System.out.println("Umurnya adalah " + kucing.getUmur());**
   15. **// Perhitungan indeks massa tubuh (BMI)**
   16. **// berat(kg) / ( tinggi(m) \* tinggi(m) )**
   17. **double bmi = kucing.getBerat() / ((kucing.getTinggi() \* 0.01) \* (kucing.getTinggi() \* 0.01));**
   18. **System.out.println("Indeks massa tubuhnya adalah " + bmi);**
   19. }
   20. }
4. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Makan dengan menggunakan mulut.. Berjalan dengan pelan.. Berlari dengan sangat cepat.. **Umurnya adalah 2 Indeks massa tubuhnya adalah 33.333333333333336** |

### Bedah Code Fungsi (Metode dengan Nilai Balik)

Fungsi getBerat, getTinggi, dan getUmur masing-masing memiliki tipe data sebelum nama fungsinya. Ia bisa dilihat secara urut double, double, dan int. Tipe data ini menandakan bahwa metode ini akan mengembalikan nilai yang memiliki tipe data tersebut.

Kemudian ada penulisan return, yang wajib dituliskan ketika kita membuat suatu fungsi. Nilai return yang dikembalikan haruslah memiliki tipe data yang sama dengan tipe data yang dideklarasikan. Misalnya getUmur memiliki deklarasi tipe int maka nilai return-nya juga harus memiliki tipe data int.

Perhatikan kode berikut:

1. System.out.println("Umurnya adalah " + kucing.getUmur());
3. double bmi = kucing.getBerat() / ((kucing.getTinggi() \* 0.01) \* (kucing.getTinggi() \* 0.01));
4. System.out.println("Indeks massa tubuhnya adalah " + bmi);

Dari kode di atas ada 2 contoh penggunaan fungsi:

* Pertama, pemanggilan fungsi getUmur() yang langsung ditampilkan ke dalam output. Tipe data yang dikembalikan adalah int.
* Kedua, perhitungan indeks massa tubuh dengan menggunakan fungsi getBerat() dan getTinggi(). Tipe data yang dikembalikan adalah double.

Kedua contoh tersebut, dapat disimpulkan bahwa pemanggilan fungsi akan mengembalikan nilai sesuai dengan tipe data yang dideklarasikan di depan nama fungsi.

Terakhir, tidak ada batasan dalam penulisan nama metode atau fungsi. Akan tetapi jangan lupakan penulisan yang dianjurkan (code conventions). Pertama, nama metode haruslah kata kerja dengan **huruf kecil** (lower case). Dan kalau terdiri lebih dari 1 kata, penulisan dimulai dari kata kerja dengan huruf kecil dan diikuti dengan kata benda yang huruf pertamanya adalah besar/kapital. Ada beberapa contoh yang bisa Anda lihat berikut ini:

1. **l**ari()
2. **j**alan()
3. **s**et**U**mur()
4. **g**et**U**mur()
5. **i**s**E**mpty()
6. **l**empar**B**ola**C**epat()
7. **l**empar**B**ola**L**ambat()

# Access Modifier

Access modifier di dalam Object Oriented Programming (OOP) akan menentukan apakah kelas lain dapat menggunakan field atau meng-invoke methods dari suatu kelas. Ada beberapa macam access modifier yang dapat digunakan yaitu private, default, protected, dan public.

## Private

Access modifier **private** akan membatasi akses hanya di dalam class. Private biasanya digunakan sebagai modifier dari member dan metode suatu class.

### Codelab Access Modifier Private

Mari kita coba access modifier private di dalam program.

1. Buatlah proyek baru dengan nama **AccessModifier** dengan nama package com.dicoding.javafundamental.accessmodifier di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama KelasA, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.accessmodifier;
   3. public class KelasA {
   5. **private int memberA = 5;**
   6. **private int functionA() {**
   7. **return memberA;**
   8. **}**
   9. **int functionB() {**
   10. **// Pemanggilan private member dan private function**
   11. **int hasil = functionA() + memberA;**
   12. **return hasil;**
   13. **}**
   14. }
3. Buatlah kelas Main dan tambahkan kode berikut:
   1. package com.dicoding.javafundamental.accessmodifier;
   3. public class Main {
   4. **public static void main(String[] args) {**
   5. **KelasA kelasA = new KelasA();**
   7. **System.out.println(kelasA.memberA);**
   8. **System.out.println(kelasA.functionA());**
   10. **System.out.println(kelasA.functionB());**
   11. **}**
   12. }

Jika diperhatikan, kode diatas akan terjadi eror karena memberA dan functionA dalam keadaan private, hal tersebut mengatikbatkan tidak bisa diakses dari luar kelas.

1. Maka ubahlah kode di kelas Main menjadi seperti ini:
   1. package com.dicoding.javafundamental.accessmodifier;
   3. public class Main {
   4. **public static void main(String[] args) {**
   5. **KelasA kelasA = new KelasA();**
   6. **System.out.println(kelasA.functionB());**
   7. **}**
   8. }
2. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| 10 |

### Bedah Code Access Modifier Private

Perhatikan kode berikut:

1. public class KelasA {
3. private int memberA = 5;
5. private int functionA() {
6. return memberA;
7. }
9. int functionB() {
10. // Pemanggilan private member dan private function
11. int hasil = functionA() + memberA;
12. return hasil;
13. }
14. }

Dari kode di atas dapat dilihat bahwa ada kelas dengan nama KelasA. Di dalamnya ada 2 contoh kode yang menggunakan modifier **private** yaitu memberA dan functionA. Variabel memberA memiliki tipe data integer dengan nilainya 5. Sementara fungsi functionA memiliki nilai balik integer yang di ambil dari memberA, ini berarti functionA juga akan mengembalikan nilai 5. Kemudian yang perlu diperhatikan adalah pada fungsi functionB. Di dalamnya ada akses ke member **private** dan fungsi **private**. Jika dijalankan maka nilai yang didapatkan adalah **10 (5+5)**. Oleh karena itu, kode di atas merupakan contoh dari modifier private di mana akses ke variabel/fungsi hanya dari kelas tersebut.

## Default

Default modifier berarti penulisan kodenya tanpa atribut modifier. Ini berlaku untuk semua kelas, member, atau fungsi yang kita tuliskan tanpa access modifier.

Modifier default bisa diakses selama masih dalam satu package.

### Codelab Access Modifier Default

Mari kita tambahkan beberapa default modifier baru pada contoh sebelumnya yaitu KelasA.

1. Bukalah kembali proyek **AccessModifier.**
2. Masukkan kode berikut ke dalam kelas kelasA:
   1. public class KelasA {
   3. private int memberA = 5;
   5. **char memberB = 'A';**
   6. **double memberC = 1.5;**
   8. private int functionA() {
   9. return memberA;
   10. }
   12. int functionB() {
   13. // Pemanggilan private member dan private function
   14. int hasil = functionA() + memberA;
   15. return hasil;
   16. }
   17. }

Perhatikan kode di atas. Ada 2 member baru yaitu memberB dan memberC. Keduanya tidak memiliki access modifier maka keduanya adalah termasuk default. Dan lihat juga functionB, fungsi ini juga termasuk default modifier. Terakhir, KelasA juga merupakan default modifier karena tidak memiliki access modifier.

1. Apa bedanya dengan access modifier **private**? Lihatlah contoh pemakaian dari KelasA di kelas Main. Bukalah kelas Main dan tambahkan kode berikut:
   1. package com.dicoding.javafundamental.accessmodifier;
   3. public class Main {
   4. public static void main(String[] args) {
   5. KelasA kelasA = new KelasA();
   7. System.out.println(kelasA.functionB());
   9. **System.out.println(kelasA.memberB);**
   10. **System.out.println(kelasA.memberC);**
   11. }
   12. }
2. Jalankan kode di atas maka hasilnya akan jadi seperti ini:

|  |
| --- |
| 10 A 1.5 |

1. Pemanggilan memberB, memberC, dan functionB seperti di atas tidak akan menimbulkan kompiler eror. Tapi perlu diingat bahwa access modifier default tidak bisa diakses dari luar package. Sebagai contoh kita akan coba modifikasi dengan menambahkan package pada masing-masing kelas. Buatlah 2 package baru di dalamnya dengan nama package1 dan package2.
2. Kemudian pindahkan kelasA dan Main ke dalam package1.
3. Buatlah kelas baru dengan nama Main di dalam package2.
4. Selanjutnya di kelas Main kita  akan mencoba membuat objek KelasA. Masukkan kode berikut di dalam kelas Main.
   1. package com.dicoding.javafundamental.accessmodifier.package2;
   3. **import com.dicoding.javafundamental.accessmodifier.package1.KelasA;**
   5. public class Main {
   6. **public static void main(String[] args) {**
   7. **// Kode ini pasti akan mengalami kompiler error**
   8. **KelasA kelasA = new KelasA();**
   9. **System.out.println(kelasA.functionB());**
   10. **System.out.println(kelasA.memberB);**
   11. **System.out.println(kelasA.memberC);**
   12. **}**
   13. }

Jika diperhatikan kode berikut pasti akan eror:  
Ini terjadi karena functionB, memberB dan memberC tidak dalam keadaan **public**. Anda bisa beri komentar untuk ketiga kode tersebut menjadi seperti ini:

* 1. package com.dicoding.javafundamental.accessmodifier.package2;
  3. import com.dicoding.javafundamental.accessmodifier.package1.KelasA;
  5. public class Main {
  6. public static void main(String[] args) {
  8. // Kode ini pasti akan mengalami kompiler error
  10. KelasA kelasA = new KelasA();
  12. **//**System.out.println(kelasA.functionB());
  14. **//**System.out.println(kelasA.memberB);
  15. **//**System.out.println(kelasA.memberC);
  17. }
  18. }

## Protected

Access modifier **protected** bisa diakses selama masih dalam satu package. Protected memiliki sedikit perbedaan dengan default modifier. Perbedaannya adalah protected bisa diakses dari luar package. Akan tetapi, satu-satunya cara untuk akses dari luar package adalah kelas yang hendak mengakses, merupakan kelas turunannya.

### Codelab Protected

Kita tambahkan access modifier **public** terlebih dahulu pada KelasA agar bisa di akses dari luar package. Kita juga menambahkan metode baru methodC dengan modifiernya sebagai **protected**.

1. Bukalah kembali proyek **AccessModifier.**
2. Tambahkan methodC di dalam kelas kelasA:
   1. package com.dicoding.javafundamental.accessmodifier.package1;
   3. public class KelasA {
   5. private int memberA = 5;
   7. char memberB = 'A';
   8. double memberC = 1.5;
   10. private int functionA() {
   11. return memberA;
   12. }
   14. int functionB() {
   15. // Pemanggilan private member dan private function
   16. int hasil = functionA() + memberA;
   17. return hasil;
   18. }
   20. **protected void methodC(){**
   21. **System.out.println("Percobaan access modifier!!!");**
   22. **}**
   23. }
3. Selanjutnya buatlah kelas baru dengan KelasB yang merupakan turunan dari KelasA di package2.
4. Masukkan kode berikut di dalam KelasB.
   1. package com.dicoding.javafundamental.accessmodifier.package2;
   3. **import com.dicoding.javafundamental.accessmodifier.package1.KelasA;**
   5. public class KelasB extends **KelasA** {
   7. **@Override**
   8. **protected void methodC() {**
   9. **super.methodC();**
   10. **System.out.println("Contoh pemanggilan protected dari package luar");**
   11. **}**
   12. }

Kode di atas adalah contoh kelas turunan yang bisa mengakses metode dengan modifier protected dari induknya.

1. Bukalah Main di **package2** dan tambahkan kode berikut
   1. package com.dicoding.javafundamental.accessmodifier.package2;
   3. import com.dicoding.javafundamental.accessmodifier.package1.KelasA;
   5. public class Main {
   6. public static void main(String[] args) {
   8. // Kode ini pasti akan mengalami kompiler error
   10. KelasA kelasA = new KelasA();
   12. //System.out.println(kelasA.functionB());
   14. //System.out.println(kelasA.memberB);
   15. //System.out.println(kelasA.memberC);
   17. **KelasB kelasB = new KelasB();**
   18. **kelasB.methodC();**
   20. }
   21. }
2. Jalankan kelas Main yang ada di package2, maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Percobaan access modifier!!! Contoh pemanggilan protected dari package luar |

## Public

Access modifier **public** bisa kita sebut sebagai modifier global. Artinya bisa diakses dari manapun bahkan package yang berbeda.

Seperti pada contoh kode sebelumnya, KelasA ditambahkan modifier public. Karena modifiernya public maka bisa diakses dari package lainnya.

Ada 4 access modifier yang masing-masing memiliki batasan akses yang berbeda-beda. Berikut ini tabel perbandingan antara keempatnya.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| Private | Yes | No | No | No |
| Default | Yes | Yes | No | No |
| Protected | Yes | Yes | Yes | No |
| Public | Yes | Yes | Yes | Yes |

Dimulai dari private yang hanya memiliki batasan akses dari dalam kelas. Kemudian default modifier yang memiliki batasan dalam satu package. Protected memiliki akses dari luar package selama kelas tersebut merupakan turunannya. Dan terakhir, public, yang dapat diakses secara global.

### Non Access Modifier

Java juga mengenal modifier lainnya yaitu tipe non access modifier. Ada beberapa non access modifier yaitu static, final, transient,  dan synchronize.

Non access modifier memiliki fungsi yang beragam tergantung dari kebutuhannya. Mari kita bahas satu per satu.

## Static

Yang pertama adalah static. Ia digunakan untuk mendeklarasikan variabel atau metode yang berdiri sendiri tanpa perlu instance dari suatu kelas. Ada dua macam non access modifier static yaitu static variable dan static methods.

### Static Variable

Variabel yang berdiri sendiri tanpa perlu instance dari kelas.

### Codelab Static Variable

Kita akan mempraktikan latihan untuk memperdalam materi satatic variable:

1. Bukalah kembali proyek **AccessModifier.**
2. Buatlah package baru dengan nama package3 dan tambahkan kelas Perhitungan dan Main di dalamnya:
3. Tambahkan kode berikut di dalam kelas Perhitungan:
   1. package com.dicoding.javafundamental.accessmodifier.package3;
   3. public class Perhitungan {
   4. **public static int nilai = 0;**
   5. }
4. Untuk mengakses kelas perhitungan di atas, kita tidak perlu membuat objeknya di dalam kelas Main:
   1. package com.dicoding.javafundamental.accessmodifier.package3;
   3. public class Main {
   4. **public static void main(String[] args) {**
   5. **System.out.println("Nilainya adalah " + Perhitungan.nilai);**
   6. **}**
   7. }
5. Jalankan kelas Main yang ada di package3, maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Nilainya adalah 0 |

1. Dan perlu diketahui juga bahwa **static variable** hanya berjumlah satu, tak  peduli berapapun obyek yang dibuat. Sebagai contoh kita modifikasi kelas Perhitungan dengan menambahkan konstruktor, seperti di bawah ini.
   1. package com.dicoding.javafundamental.accessmodifier.package3;
   3. public class Perhitungan {
   4. public static int nilai = 0;
   6. **Perhitungan() {**
   7. **nilai++;**
   8. **}**
   9. }
2. Kemudian tes pembuatan objek kelas perhitungan sebanyak 5 kali, seperti di bawah ini.
   1. package com.dicoding.javafundamental.accessmodifier.package3;
   3. public class Main {
   4. public static void main(String[] args) {
   5. System.out.println("Nilainya adalah " + Perhitungan.nilai);
   7. **for (int x = 0 ; x < 5; x++){**
   8. **new Perhitungan();**
   9. **}**
   10. **System.out.println("Sampai "+Perhitungan.nilai);**
   11. }
   12. }
3. Jalankan lagi kelas Main yang ada di package3, maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Nilainya adalah 0 Sampai 5 |

### Static Methods

Metode yang berdiri sendiri tanpa perlu instance dari kelas.

### Codelab Static Methods

Mari kita coba:

1. Bukalah kembali proyek **AccessModifier.**
2. Tambahkan kode berikut di dalam kelas Perhitungan:
   1. package com.dicoding.javafundamental.accessmodifier.package3;
   3. public class Perhitungan {
   4. public static int nilai = 0;
   6. **protected static int getNilai(){**
   7. **return nilai;**
   8. **}**
   10. Perhitungan() {
   11. nilai++;
   12. }
   13. }
3. Untuk mengakses kelas perhitungan di atas, kita tidak perlu membuat objeknya di dalam kelas Main:
   1. package com.dicoding.javafundamental.accessmodifier.package3;
   3. public class Main {
   4. public static void main(String[] args) {
   5. System.out.println("Nilainya adalah " + Perhitungan.nilai);
   7. for (int x = 0 ; x < 5; x++){
   8. new Perhitungan();
   9. }
   11. System.out.println("Sampai "+Perhitungan.nilai);
   13. System.out.println("getNilai memiliki nilai " + Perhitungan.getNilai());
   14. }
   15. }
4. Jalankan kelas Main yang ada di package3, maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Nilainya adalah 0 Sampai 5 Getnilai memiliki nilai 5 |

## Final

Non access modifier kedua adalah final. Ada 3 macam penggunaan non access modifier final yaitu final variabel, final methods dan final class.

### Final Variable

Final variabel hanya bisa diinisiasi sekali. Ini menandakan bahwa variabel yang sudah dideklarasikan final tidak dapat diganti dengan objek lainnya.

Perhatikan contoh kode di bawah ini:

1. package com.dicoding.javafundamental.accessmodifier.package4;
3. public class Perhitungan {
4. **final int nilai = 5;**
6. **void ubahNilai(){**
7. **// Kode di bawah akan menampilkan error**
8. **nilai = 10;**
9. **}**
10. }

Kode di atas akan eror karena kita mencoba mengganti variabel nilai dengan objek baru.

Biasanya final digunakan bersamaan dengan static untuk membuat suatu konstanta. Misalnya contoh kode konstanta nilai PI dari lingkaran, seperti di bawah ini.

1. package com.dicoding.javafundamental.accessmodifier.package4;
3. public class Lingkaran {
4. **static final double PI = 3.141;**
5. }

### Final Methods

Metode yang dideklarasikan final maka tidak dapat di override oleh anak kelas. Ini akan berguna jika kita ingin membuat metode yang tidak dapat diubah. Contoh kodenya adalah seperti ini.

1. package com.dicoding.javafundamental.accessmodifier.package4;
3. public class Lingkaran {
4. static final double PI = 3.141;
6. int jari = 7;
8. final double getLuas() {
9. return PI \* (jari \* jari);
10. }
11. }

### Final Class

Kelas yang dideklarasikan sebagai final maka tidak bisa dijadikan sebagai induk kelas.

1. final class Lingkaran {
3. }

## Transient

Transient memiliki hubungan dengan proses serialisasi (serializing). Serialisasi adalah proses konversi suatu objek menjadi byte agar dapat ditransmisikan.

Variabel yang dideklarasikan sebagai transient maka akan tidak dijaga nilainya di dalam proses serialisasi. Contoh penggunaannya adalah seperti ini.

1. class Test implements Serializable {
3. *// Variabel ini tidak akan dijaga nilainya*
4. transient int nilaiA;
6. *// Variabel ini akan dijaga nilainya*
7. double nilaiB;
8. String nilaiC;
10. }

## Synchronized

Synchronized modifier digunakan untuk membatasi akses ke suatu variable/methods yang hanya boleh dilakukan oleh satu thread. Ketika ada 2 thread yang ingin mengakses synchronized variable/methods, maka prosesnya akan dilakukan secara serial (bergantian).

Contoh penggunaanya adalah seperti ini.

1. public synchronized void showData() {
3. }

# Interface

Apa itu interface? Di materi sebelumnya kita sudah mempelajari class.

Pada bahasa pemrograman Java interface sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah interface dapat diimplementasikan oleh kelas lain.

Sebuah kelas dapat mengimplementasikan lebih dari satu interface. Kelas ini akan mendeklarasikan metode pada interface yang dibutuhkan oleh kelas itu sekaligus mendefinisikan isinya pada kode program.

Metode pada interface yang diimplementasikan pada suatu kelas harus sama persis dengan yang ada pada interface tersebut. Property/Field di interface akan menjadi **static final** atau **konstanta**. Method dan field di interface akan**selalu bersifat public**. Perhatikan kata-kata yang diberi huruf tebal.

## Codelab Interface

Untuk memperjelas implementasinya, mari kita mulai saja:

1. Buatlah proyek baru dengan nama **Interface** dengan nama package com.dicoding.javafundamental.oop di dalamnya:
2. Buat **interface** di package yang baru saja Anda buat. **Klik kanan** di packagetersebut. Pilih **New** - **Java Class**. Name: Hewan, Kind: **Interface**, lalu tekan **OK**.
3. Tambahkan baris kode di bawah ini ke interface Hewan.
   1. package com.dicoding.javafundamental.oop;
   3. public interface Hewan {
   4. **public void makan() {**
   5. **System.out.println("Memakan daging, tumbuhan, atau segalanya.");**
   6. **}**
   7. }

IntelliJ memberi tanda eror. Apa yang terjadi? Coba arahkan kursor ke baris yang error (ditandai warna merah) IntelliJ akan memberi informasi “**Interface abstract method cannot have body**”. Yup, ini karena method di interface hanya berupa deklarasi seperti yang sudah dibahas di awal paragraf.  
Ubah method **makan** menjadi deklarasi seperti baris kode di bawah ini.

* 1. package com.dicoding.javafundamental.oop;
  3. public interface Hewan {
  4. **public void makan();**
  5. }

Tambahkan baris kode di bawah ini ke interface Hewan.

* 1. package com.dicoding.javafundamental.oop;
  3. public interface Hewan {
  4. public static final String respirasi = "oksigen";
  6. public void makan();
  7. }

1. Posisi saat ini tidak ada eror yang dilaporkan IntelliJ, tetapi sebagian kode kita menjadi berwarna abu-abu. Coba arahkan kursor ke bagian kode yang berwarna abu-abu. Pertama ke modifier **public** di property atau method yang kita buat. Akan ada informasi “**Modifier ‘public’ is redundant for interface fields..**”. Ini menjelaskan kalimat di paragraf di awal yaitu method dan field akan selalu bersifat **public**.
2. Selain modifier public, kode yang berwana abu-abu adalah keyword **static**. Coba arahkan kursor ke bagian tersebut. Akan ada informasi “**Modifier ‘static’ is redundant for interface fields..**”. Ini menandakan keyword **static** juga tidak diperlukan karena field di interface akan dianggap sebagai **static**.
3. Begitu juga untuk keyword **final** tidak diperlukan. Apabila kursor diarahkan ke bagian keyword final maka akan ada informasi “**Modifier ‘final’ is redundant for interface fields..**”. Ini menjelaskan kalimat di paragraf awal yaitu property/field akan selalu bersifat **static** **final**.
4. Berdasarkan penjelasan sebelumnya maka kita bisa mengubah kode menjadi seperti di bawah ini. Tidak ada eror yang dilaporkan IntelliJ. Perhatikan masih ada kode yang diwarnai abu-abu karena method atau field “... **is never used**”. Hal ini wajar.
   1. package com.dicoding.javafundamental.oop;
   3. public interface Hewan {
   4. **String respirasi = "oksigen";**
   5. **void makan();**
   6. }
5. Ada satu hal lagi yang perlu dibahas dari penjelasan kalimat pada paragraf di awal yaitu field akan menjadi static final atau konstanta. Konstanta di kode Java biasanya ditulis dengan huruf besar semuanya dan diberi pemisah underscore jika lebih dari 1 kata. Hal ini tertuang di java**coding convention**bagian Constants. **Coding convention**sangat diperlukan dalam bahasa pemrograman apapun. Gunanya, mempermudah developer membaca kode yang ditulis oleh orang lain atau bahkan diri sendiri. Sehingga interface Hewan selengkapnya menjadi seperti di bawah ini.
   1. package com.dicoding.javafundamental.oop;
   3. public interface Hewan {
   4. String **RESPIRASI** = "oksigen";
   6. void makan();
   7. }

# Implement

Setelah kita mempelajari cara dan ketentuan dalam membuat interface, berikutnya kita akan mempelajari cara menggunakan interface tersebut. Interface harus dapat digunakan ke kelas lain. Cara untuk mengimplementasi interface yaitu menggunakan keyword **implements** pada kelas yang mengimplementasikannya. Kelas yang mengimplementasi interface (bisa lebih dari 1 interface) harus mendefinisikan isi kode semua deklarasi metode yang ada pada interface tersebut.

## Codelab Implement

Masih ingat kan kalau metode di interface hanya berupa deklarasi? Jika suatu kelas tak mendefinisikan isi kode semua deklarasi metode yang ada di interface, maka kelas tersebut harus menjadi **abstract class**. Untuk lebih jelasnya mari kita langsung coding saja.

1. Buatlah proyek baru dengan nama **InterfaceImplement** dengan nama package com.dicoding.javafundamental.oop di dalamnya:
2. Buat **interface** di package yang baru saja Anda buat. **Klik kanan** di packagetersebut. Pilih **New** - **Java Class**. Name: Hewan, Kind: **Interface**, lalu tekan **OK**.  
   Tambahkan kode berikut di dalamnya:
   1. package com.dicoding.javafundamental.oop;
   3. **public interface Hewan {**
   4. **String respirasi = "oksigen";**
   5. **void makan();**
   6. **}**
3. Buatlah sebuah kelas baru dengan nama Kucing.  
   Kemudian implementasikan dengan kelas Hewan seperti ini:
   1. package com.dicoding.javafundamental.oop;
   3. public class Kucing **implements Hewan {**
   4. }

Perhatikan IntelliJ akan memberi informasi kesalahan ditandai baris kode berwarna merah. Jika kita arahkan kursor ke baris tersebut maka akan ada informasi **“Class ‘Kucing’ must either be declared abstract or implement abstract method ‘makan()’ in ‘Hewan’**”. Ini menjelaskan kalau kita harus memilih untuk menjadikan kelas Kucing sebagai abstract class atau mendefinisikan isi kode semua deklarasi metode dari interface Hewan.

1. Jika kita ingin menjadikan kelas Kucing sebagai abstract class, cukup tambahkan keyword abstract sebelum keyword classseperti berikut.
   1. package com.dicoding.javafundamental.oop;
   3. public **abstract** class Kucing implements Hewan {
   4. }
2. Jika kita ingin mengimplementasikan interface maka kita harus membuat method signature (nama, parameter, return) yang sama disertai isi kode program untuk metodenya. Berikut contoh implementasi metode makan.
   1. package com.dicoding.javafundamental.oop;
   3. public class Kucing implements Hewan {
   4. **@Override**
   5. **public void makan() {**
   7. **}**
   8. }

## Code to Interface not Implementation

Dari latihan sebelumnya kita membuat interfaceHewan. Kemudian kita membuat kelas Kucing yang menerapkan interface tersebut. Maka tiap metode yang berada pada interface harus diimplementasikan. Interface terlihat sebagai suatu **aturan atau kontrak** bagi kelas yang mengimplementasikannya.

Salah satu contoh implementasi interface adalah JDBC API. **Java Database Connectivity (JDBC)** adalah package (java.sql dan javax.sql) yang menyediakan akses ke database. Sedangkan **Application Programming Interface (API)**, dari namanya sudah bisa ditebak adalah kumpulan Interface.

Jadi JDBC API adalah kumpulan interface untuk mengakses database sedangkan implementasinya akan dibuat oleh masing-masing merk database misal MySql, SqlServer, Oracle dan lain-lain. Implementasi tersebut biasa disebut **JDBC Driver**. JDBC Driver akan kita unduh (dalam bentuk jar) tergantung dari database yang digunakan untuk dimasukkan ke classpath.

Dalam coding Java ada istilah **best practice**(**design pattern)** “code to interface not implementation”. Salah satu maksudnya adalah seperti contoh JDBC API di paragraf sebelumnya. Contoh lainnya adalah ketika kita membuat aplikasi yang terdiri dari beberapa layer. Maka lebih baik gunakan interface untuk komunikasi antar layernya.

# Inheritance

Pada dasarnya setiap objek yang berbeda sering memiliki kesamaan atau kemiripan tertentu. Kita ambil contoh kelas hewan. Misalnya, kucing dan anjing memiliki kemiripan sebagai hewan. Kesamaan yang dimiliki kucing dan anjing misalnya sama-sama hewan mamalia. Namun suara atau cara komunikasi kucing dan anjing berbeda. Begitu juga dengan hewan lain, misalnya harimau dan domba keduanya juga hewan mamalia dengan suara serta komunikasi yang berbeda.

Dari contoh di atas bisa dipahami kalau kucing, anjing, harimau, dan domba adalah hewan. Maka dalam bahasa Java kucing, anjing, harimau, dan domba adalah turunan (inheritance) dari hewan. Pada pemrograman berorientasi objek atau OOP, konsep inheritance menjadi salah satu topik yang penting. Suatu objek diwariskan dengan menggunakan keyword extends**.**Pada pemrograman Java setiap objek pada kenyataannya adalah turunan dari classObject. Walaupun tidak secara eksplisit, ini bisa dibuktikan dengan operator instanceof.

Beberapa terminologi yang penting:

* **Super Class** atau **Parent Class**, kelas yang semua fiturnya di wariskan kepada kelas turunannya.
* **Sub Class**atau **Child Class**, kelas turunan yang mewarisi semua fitur dari kelas lain. Sub class dapat menambah field dan metodenya sendiri sebagai tambahan dari kelas yang memberi warisan.
* **Reusability**, yaitu ketika kita ingin membuat kelas baru dan sudah ada kelas yang berisi kode yang kita inginkan, kita bisa menurunkan kelas baru tersebut dari kelas yang sudah ada. Dengan begitu, kita menggunakan kembali field dan metode dari kelas yang telah ada.

## Codelab Inheritance

Untuk lebih detailnya kita coba langsung praktikan pada kode program berikut ini.

1. Buatlah proyek baru dengan nama **Inheritance**dengan nama package com.dicoding.javafundamental.inheritance di dalamnya:
2. Buatlah kelas baru dengan nama Hewan di dalam package tersebut.
3. Tambahkan kode berikut di dalam kelas Hewan:
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Hewan {
   4. **public Hewan() {**
   5. **System.out.println("construct Hewan");**
   6. **}**
   7. }
4. Buatlah kelas Kucing, inherit dari class Hewan menggunakan extends.
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Kucing **extends Hewan** {
   4. **public Kucing() {**
   5. **super(); // akan tetap memanggil constructor dari parent Class**
   6. **System.out.println("construct Kucing");**
   7. **}**
   8. }
5. Selanjutnya buatlah sebuah class baru dengan nama Main, lalu tambahkan kode berikut:
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Main {
   4. **public static void main(String[] args) {**
   5. **Hewan hewan = new Hewan(); // memanggil constructor**
   6. **System.out.println("Apakah hewan IS-A Objek -> " + (hewan instanceof Object));**
   7. **System.out.println("Apakah hewan IS-A Hewan -> " + (hewan instanceof Hewan));**
   8. **System.out.println("Apakah hewan IS-A Kucing -> " + (hewan instanceof Kucing));**
   9. **System.out.println("------------------------");**
   10. **Kucing kucing = new Kucing(); // memanggil constructor**
   11. **System.out.println("Apakah hewan IS-A Objek -> " + (kucing instanceof Object));**
   12. **System.out.println("Apakah kucing IS-A Hewan -> " + (kucing instanceof Hewan));**
   13. **System.out.println("Apakah kucing IS-A Kucing -> " + (kucing instanceof Kucing));**
   14. **}**
   15. }
6. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.  
   Output-nya akan seperti di bawah ini.

|  |
| --- |
| construct Hewan Apakah hewan IS-A Objek -> true Apakah hewan IS-A Hewan -> true Apakah hewan IS-A Kucing -> false ------------------------ construct Hewan construct Kucing Apakah hewan IS-A Objek -> true Apakah kucing IS-A Hewan -> true Apakah kucing IS-A Kucing -> true |

1. Dari output di atas bisa kita simpulkan bahwa seekor kucing adalah (**IS-A**) hewan . Tapi seekor hewan belum tentu kucing dan pastinya hewan adalah instans dari kelas Hewan dan kucing adalah instans dari kelas Kucing yang keduanya adalah (**IS-A**) Object.
2. Perhatikan output berikut:

|  |
| --- |
| ... **construct Hewan construct Kucing** ... |

1. Ketika membuat instans objek kucing, constructor dari class Hewan akan tetap dipanggil walaupun sengaja kita komentari baris super(). Coba jadikan komentar di baris tersebut dan jalankan ulang class Main maka hasilnya akan sama saja.

## Overriding dan Overloading

Setiap kelas yang extends kelas lain akan mewarisi semua field dan metode yang ada di parent class tersebut.

### Codelab Overriding dan Overloading

1. Bukalah kembali proyek **Inheritance.**
2. Masukkan kode berikut ke dalam kelas Hewan:
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Hewan {
   4. public Hewan() {
   5. System.out.println("construct Hewan");
   6. }
   8. **public void makan() {**
   9. **System.out.println("Hewan sedang makan..");**
   10. **}**
   11. }
3. Tambahkan kode berikut di kelas Main:
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Main {
   4. public static void main(String[] args) {
   5. Hewan hewan = new Hewan(); // memanggil constructor
   6. System.out.println("Apakah hewan IS-A Objek -> " + (hewan instanceof Object));
   7. System.out.println("Apakah hewan IS-A Hewan -> " + (hewan instanceof Hewan));
   8. System.out.println("Apakah hewan IS-A Kucing -> " + (hewan instanceof Kucing));
   10. System.out.println("------------------------");
   12. Kucing kucing = new Kucing(); // memanggil constructor
   13. System.out.println("Apakah hewan IS-A Objek -> " + (kucing instanceof Object));
   14. System.out.println("Apakah kucing IS-A Hewan -> " + (kucing instanceof Hewan));
   15. System.out.println("Apakah kucing IS-A Kucing -> " + (kucing instanceof Kucing));
   17. **System.out.println("------------------------");**
   19. **kucing.makan(); *// kucing mewarisi metode makan() dari parent class Hewan***
   20. }
   21. }
4. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| construct Hewan Apakah hewan IS-A Objek -> true Apakah hewan IS-A Hewan -> true Apakah hewan IS-A Kucing -> false ------------------------ construct Hewan construct Kucing Apakah hewan IS-A Objek -> true Apakah kucing IS-A Hewan -> true Apakah kucing IS-A Kucing -> true ------------------------ **Hewan sedang makan..** |

1. Ada kalanya metode yang diwariskan perlu diubah sesuai kebutuhan spesifik dari sub-class tersebut. Perubahan dapat dilakukan dengan 2 cara yaitu **overriding**dan**overloading**. Overriding adalah pembuatan metode baru pada subclass yang sama persis dengan superclassnya, sedangkan overloading adalah pembuatan metode baru pada subclass yang sama dengan method superclass namun parameternya berbeda. Untuk lebih jelasnya, bukalah kelas Hewan.
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Hewan {
   4. public Hewan() {
   5. System.out.println("construct Hewan");
   6. }
   8. public void makan() {  **// base method**
   9. System.out.println("Hewan sedang makan..");
   10. }
   11. }

Selanjutnya bukalah kelas Kucing dan tambahkan metode berikut:

* 1. package com.dicoding.javafundamental.inheritance;
  3. public class Kucing extends Hewan {
  4. public Kucing() {
  5. //super(); // akan tetap memanggil constructor dari parent Class
  6. System.out.println("construct Kucing");
  7. }
  9. **public void makan() { // overriding**
  10. **System.out.println("Kucing sedang makan..");**
  11. **}**
  12. **public void makan(String food) { // overloading**
  13. **System.out.println("Kucing makan " + food);**
  14. **}**
  15. }

1. Selanjutnya bukalah kelas Main dan tambahkan kode berikut:
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Main {
   4. public static void main(String[] args) {
   5. Hewan hewan = new Hewan(); // memanggil constructor
   6. System.out.println("Apakah hewan IS-A Objek -> " + (hewan instanceof Object));
   7. System.out.println("Apakah hewan IS-A Hewan -> " + (hewan instanceof Hewan));
   8. System.out.println("Apakah hewan IS-A Kucing -> " + (hewan instanceof Kucing));
   10. System.out.println("------------------------");
   12. Kucing kucing = new Kucing(); // memanggil constructor
   13. System.out.println("Apakah hewan IS-A Objek -> " + (kucing instanceof Object));
   14. System.out.println("Apakah kucing IS-A Hewan -> " + (kucing instanceof Hewan));
   15. System.out.println("Apakah kucing IS-A Kucing -> " + (kucing instanceof Kucing));
   17. System.out.println("------------------------");
   19. **hewan.makan();**
   20. **kucing.makan();**
   21. **kucing.makan("daging ikan");**
   22. }
   23. }
2. Maka jika dijalankan, akan jadi seperti ini:

|  |
| --- |
| construct Hewan Apakah hewan IS-A Objek -> true Apakah hewan IS-A Hewan -> true Apakah hewan IS-A Kucing -> false ------------------------ construct Hewan construct Kucing Apakah hewan IS-A Objek -> true Apakah kucing IS-A Hewan -> true Apakah kucing IS-A Kucing -> true ------------------------ **Hewan sedang makan.. Kucing sedang makan.. Kucing makan daging ikan** |

1. Seperti dibahas di awal, setiap objek pada kenyataannya adalah **turunan dari class Object.**Maka setiap objek akan otomatis mewarisi method dari class Object. Sebagian metode yang sering digunakan adalah metode toString() dan equals(). Metode toString() bisa kita gunakan untuk merepresentasikan objek dalam bentuk String yang sangat berguna salah satunya untuk debugging. Sedangkan metode equals() digunakan untuk membandingkan antara dua objek, apakah mereka sama. Metode equals() biasanya kita override karena kebutuhan khusus . Misalnya kita mau dua objek Kucing adalah sama jika rasnya sama, meski habitatnya beda. Bukalah kelas Kucingdan tambahkan kode di dalamnya menjadi seperti ini:
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Kucing extends Hewan {
   5. **private String ras;**
   6. **private String habitat;**
   7. **public Kucing(String ras, String habitat) {**
   8. **this.ras = ras;**
   9. **this.habitat = habitat;**
   10. **}**
   11. **@Override**
   12. **public String toString() {**
   13. **return "Kucing ras: " + ras + ", habitat: " + habitat;**
   14. **}**
   15. **@Override**
   16. **public boolean equals(Object obj) {**
   17. **if (obj instanceof Kucing) {**
   18. **Kucing other = (Kucing) obj;**
   19. **return this.ras.equals(other.ras);**
   20. **} else {**
   21. **return false;**
   22. **}**
   23. **}**
   25. public Kucing() {
   26. //super(); // akan tetap memanggil constructor dari parent Class
   27. System.out.println("construct Kucing");
   28. }
   30. public void makan() { // overriding
   31. System.out.println("Kucing sedang makan..");
   32. }
   34. public void makan(String food) { // overloading
   35. System.out.println("Kucing makan " + food);
   36. }
   37. }
2. Tambahkan juga kode berikut di dalam kelas Main.
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Main {
   4. public static void main(String[] args) {
   5. Hewan hewan = new Hewan(); // memanggil constructor
   6. System.out.println("Apakah hewan IS-A Objek -> " + (hewan instanceof Object));
   7. System.out.println("Apakah hewan IS-A Hewan -> " + (hewan instanceof Hewan));
   8. System.out.println("Apakah hewan IS-A Kucing -> " + (hewan instanceof Kucing));
   10. System.out.println("------------------------");
   12. Kucing kucing = new Kucing(); // memanggil constructor
   13. System.out.println("Apakah hewan IS-A Objek -> " + (kucing instanceof Object));
   14. System.out.println("Apakah kucing IS-A Hewan -> " + (kucing instanceof Hewan));
   15. System.out.println("Apakah kucing IS-A Kucing -> " + (kucing instanceof Kucing));
   17. System.out.println("------------------------");
   19. hewan.makan();
   20. kucing.makan();
   21. kucing.makan("daging ikan");
   23. **System.out.println("------------------------");**
   25. **Kucing meow = new Kucing("Ocicat", "tropis");**
   26. **Kucing puss = new Kucing("Ocicat", "subtropis");**
   27. **Kucing popo = new Kucing("Anggora", "subtropis");**
   28. **System.out.println(meow.toString());**
   29. **System.out.println(puss.toString());**
   30. **System.out.println(popo.toString());**
   31. **System.out.println("meow equals puss ? " + meow.equals(puss));**
   32. **System.out.println("meow equals popo ? " + meow.equals(popo));**
   33. }
   34. }
3. Jika dijalankan, outputnya akan seperti di bawah ini:

|  |
| --- |
| construct Hewan Apakah hewan IS-A Objek -> true Apakah hewan IS-A Hewan -> true Apakah hewan IS-A Kucing -> false ------------------------ construct Hewan construct Kucing Apakah hewan IS-A Objek -> true Apakah kucing IS-A Hewan -> true Apakah kucing IS-A Kucing -> true ------------------------ Hewan sedang makan.. Kucing sedang makan.. Kucing makan daging ikan ------------------------ **construct Hewan construct Hewan construct Hewan** **Kucing ras: Ocicat, habitat: tropis Kucing ras: Ocicat, habitat: subtropis Kucing ras: Anggora, habitat: subtropis meow equals puss ? true meow equals popo ? false** |

## Polymorphism

Polymorphism artinya banyak bentuk. Contohnya, kucing adalah seekor Hewan, kucing juga adalah sebuah objek. Artinya kucing bisa dikenali dalam banyak bentuk. Lalu apa maknanya bagi pemrograman khususnya Java?

### Codelab Polymorphism

Kita coba lihat di koding yuk! Kita akan pakai kode yang sebelumnya, agar biar lebih mudah dipahami.

1. Bukalah kembali proyek **Inheritance.**
2. Kita akan melihat kembali kelas **Hewan**,
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Hewan {
   4. public Hewan() {
   5. System.out.println("construct Hewan");
   6. }
   8. public void makan() {  **// base method**
   9. System.out.println("Hewan sedang makan..");
   10. }
   11. }

dan **Kucing**berikut:

* 1. package com.dicoding.javafundamental.inheritance;
  3. public class Kucing extends Hewan {
  5. private String ras;
  6. private String habitat;
  8. public Kucing(String ras, String habitat) {
  9. this.ras = ras;
  10. this.habitat = habitat;
  11. }
  13. @Override
  14. public String toString() {
  15. return "Kucing ras: " + ras + ", habitat: " + habitat;
  16. }
  18. @Override
  19. public boolean equals(Object obj) {
  20. if (obj instanceof Kucing) {
  21. Kucing other = (Kucing) obj;
  22. return this.ras.equals(other.ras);
  23. } else {
  24. return false;
  25. }
  26. }
  28. public Kucing() {
  29. super(); // akan tetap memanggil constructor dari parent Class
  30. System.out.println("construct Kucing");
  31. }
  33. public void makan() { **// overriding**
  34. System.out.println("Kucing sedang makan..");
  35. }
  37. public void makan(String food) { **// overloading**
  38. System.out.println("Kucing makan " + food);
  39. }
  40. }

1. Selanjutnya bukalah kelas Maindan tambahkan kode berikut:
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Main {
   4. public static void main(String[] args) {
   5. Hewan hewan = new Hewan(); // memanggil constructor
   6. System.out.println("Apakah hewan IS-A Objek -> " + (hewan instanceof Object));
   7. System.out.println("Apakah hewan IS-A Hewan -> " + (hewan instanceof Hewan));
   8. System.out.println("Apakah hewan IS-A Kucing -> " + (hewan instanceof Kucing));
   10. System.out.println("------------------------");
   12. Kucing kucing = new Kucing(); // memanggil constructor
   13. System.out.println("Apakah hewan IS-A Objek -> " + (kucing instanceof Object));
   14. System.out.println("Apakah kucing IS-A Hewan -> " + (kucing instanceof Hewan));
   15. System.out.println("Apakah kucing IS-A Kucing -> " + (kucing instanceof Kucing));
   17. System.out.println("------------------------");
   19. hewan.makan();
   20. kucing.makan();
   21. kucing.makan("daging ikan");
   23. System.out.println("------------------------");
   25. Kucing meow = new Kucing("Ocicat", "tropis");
   26. Kucing puss = new Kucing("Ocicat", "subtropis");
   27. Kucing popo = new Kucing("Anggora", "subtropis");
   29. System.out.println(meow.toString());
   30. System.out.println(puss.toString());
   31. System.out.println(popo.toString());
   33. System.out.println("meow equals puss ? " + meow.equals(puss));
   34. System.out.println("meow equals popo ? " + meow.equals(popo));
   36. **System.out.println("------------------------");**
   37. **Object o = new Kucing(); // kucing adalah Object**
   38. **Hewan h = new Kucing(); // kucing adalah Hewan**
   39. **Kucing k = new Kucing(); // kucing adalah Kucing, pastinya :)**
   41. **//o.makan(); -> error, karena method makan() tidak ada di class Object**
   42. **h.makan();**
   43. **k.makan();**
   44. **Object object = k; // bisa langsung diassign**
   45. **Hewan hewanK= k; // bisa langsung diassign**
   46. **//object.makan(); -> error, karena method makan() tidak ada di class Object**
   47. **hewanK.makan();**
   48. **Kucing kucingK = (Kucing) h; // tidak bisa langsung diassign, perlu casting**
   49. **kucingK.makan();**
   51. **Hewan harimau = new Hewan();**
   52. **Kucing anggora = (Kucing) harimau; // compile OK tetapi runtime error ClassCastException**
   53. }
   54. }
2. Coba jalankan kode di atas lalu lihat hasilnya. Dari kode tersebut bisa dilihat instance yang lebih spesifik bisa langsung di-assign ke class yang lebih umum (misal instance kucing ke instance object atau hewan),. Tapi sebaliknya tidak  berlaku.

|  |
| --- |
| construct Hewan Apakah hewan IS-A Objek -> true Apakah hewan IS-A Hewan -> true Apakah hewan IS-A Kucing -> false ------------------------ construct Hewan construct Kucing Apakah hewan IS-A Objek -> true Apakah kucing IS-A Hewan -> true Apakah kucing IS-A Kucing -> true ------------------------ Hewan sedang makan.. Kucing sedang makan.. Kucing makan daging ikan ------------------------ construct Hewan construct Hewan construct Hewan Kucing ras: Ocicat, habitat: tropis Kucing ras: Ocicat, habitat: subtropis Kucing ras: Anggora, habitat: subtropis meow equals puss ? true meow equals popo ? false **------------------------ construct Hewan construct Kucing construct Hewan construct Kucing construct Hewan construct Kucing Kucing sedang makan.. Kucing sedang makan.. Kucing sedang makan.. Kucing sedang makan.. construct Hewan Exception in thread "main" java.lang.ClassCastException: class com.dicoding.javafundamental.inheritance.Hewan cannot be cast to class com.dicoding.javafundamental.inheritance.Kucing (com.dicoding.javafundamental.inheritance.Hewan and com.dicoding.javafundamental.inheritance.Kucing are in unnamed module of loader 'app') at com.dicoding.javafundamental.inheritance.Main.main(Main.java:56)** |

1. Perlu dilakukan casting untuk mengubah instance hewan menjadi instance kucing. Casting sendiri adalah mengubah suatu tipe data menjadi tipe data yang lain. Pada inheritance sendiri, casting dapat dilakukan terhadap kelas yang mempunyai hubungan parent-child. Analoginya kucing adalah pasti hewan tetapi belum tentu hewan adalah kucing. Bisa jadi harimau atau elang atau yang lainnya. bukan?  Sehingga jika casting gagal akan muncul sebagai ClassCastException.
2. Interface juga bisa diterapkan dalam penggunaan polymorphism ini. Buatlah interface dengan nama Mamalia.
3. Lalu kita pasang interface tersebut ke class Kucing.
   1. package com.dicoding.javafundamental.inheritance;
   3. public class Kucing extends Hewan **implements Mamalia**{
   4. ...
   5. }
4. Sehingga kita bisa mendeklarasikan instance kucing menggunakan interface Mamalia.
   1. **Mamlia mamalia = new Kucing();**

Pada praktiknya banyak kode java menggunakan cara deklarasi di atas yaitu menggunakan interface. Coba ingat kembali materi **Code to Interface** di materi Interface.

### Perbedaan Antara Inheritance Dengan Interface

Sekilas penggunaan **Inheritance** dan interface mirip-mirip. Keduanya sama-sama akan menurunkan field/method. Lalu apa bedanya antara I**nheritance** dan interface? Hal ini sering ditanyakan dalam interview programmer Java :) Jawaban yang sering dilontarkan adalah “interface menggunakan **Implements** sedangkan **Inheritance** menggunakan **extends**.”

Sebenarnya, jawaban tersebut **tidak cocok**. Alih-alih, jawaban tersebut cocok untuk pertanyaan bagaimana cara menggunakan interface dan menggunakan **Inheritance**.

Konsep inheritance digunakan untuk **abstraksi** dari yang paling umum ke yang lebih spesifik. Misalnya class Hewan adalah bentuk yang umum, lalu class Kucing adalah turunannya yang lebih spesifik.

Sedangkan interface digunakan sebagai **kontrak atau aturan**. Class yang menerapkan suatu interface wajib override semua method dari interface tersebut. Artinya class tersebut harus mengikuti aturan atau spesifikasi yang ada di interface.

Sampai sini, Anda sudah paham perbedaan **Inheritance** dan **Interface**? Ingat saja kata kuncinya **inheritance adalah abstraksi sedangkan interface adalah kontrak**. Happy coding :)

# Pengantar Common Class

Bahasa pemograman Java dibangun dengan banyak library. Ini akan membantu programmer dalam memecahkan masalah saat proses development. Common class sendiri merupakan kelas yang sering atau umum digunakan dari pustaka standar Java tersebut.

Berikut Common Class yang akan kita pelajari:

1. **Generic**  
   Generics pada Java memperbolehkan tipe (String, Integer, dll, serta user defined) bisa menjadi suatu parameter untuk metode, class dan interface. Generics juga berfungsi untuk type casting dan compile time safety.
2. **Exception**  
   Eror adalah hal yang perlu kita tangani. Nah, di sini  kita bisa menggunakan exception.
3. **Input Output**  
   Di Java sudah tersedia library untuk membaca input dari suatu berkas dan juga menuliskan data ke suatu berkas.
4. **Date Time**  
   Di Java sudah tersedia library yang khusus berurusan dengan waktu.
5. **Casting**  
   Mengubah suatu tipe data atau objek menjadi tipe data atau objek lainnya, akan menjadi mudah dengan menggunakan casting yang sudah tersedia di dalam Java.

# Generics

Generics adalah salah satu fitur yang terdapat di JDK 1.5. Ia sangat ditunggu-tunggu. Salah satunya untuk menyederhanakan penulisan kode dari **type-casting** juga untuk **compile-time type safety**. Generics dalam kode program bisa dikenali dengan **type-parameter**. Contoh penggunaan Generics paling umum adalah Collection.

Sebelumnya di materi Collection kita sudah menggunakan fitur Generics yaitu ketika kita deklarasi **Set<String>**, Set dengan type-parameter <String>. Kita juga deklarasi **Map<String, Planet>**, Map dengan type-parameter <String, Planet>. Tetapi untuk **List** kita deklarasi tanpa type-parameter. Apa perbedaannya? Ayo kita lihat dengan coding!

## Codelab Generics

Mari kita praktikan Codelab berikut agar lebih paham mengenai Generics.

1. Buatlah proyek baru dengan nama **Generics** dengan nama package com.dicoding.javafundamental.generics di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama Planet, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.generics;
   3. class Planet {
   4. **private String name;**
   5. **private double mass;**
   7. **public Planet(String name, double mass) {**
   8. **this.name = name;**
   9. **this.mass = mass;**
   10. **}**
   11. **public void print() {**
   12. **System.out.println("Planet " + name + ", mass: " + mass);**
   13. **}**
   14. }
3. Selanjutnya buatlah kelas Main dan tambahkan kode berikut:
   1. package com.dicoding.javafundamental.generics;
   3. import java.util.ArrayList;
   4. import java.util.List;
   6. public class Main {
   8. public static void main(String[] args) {
   9. **List lo = new ArrayList(); // List tanpa type-parameter**
   10. **lo.add("lo - String 1"); // lo menampung objek String**
   11. **lo.add(new Planet("Mercury", 0.06)); // lo menampung objek Planet**
   12. **List<Planet> lp = new ArrayList(); // List dengan type-parameter Planet**
   13. **lp.add(new Planet("Mercury", 0.06)); // lp menampung objek Planet**
   14. **lp.add("lp - String 1"); // baris ini compile-error, lp tidak diijinkan menampung objek String**
   15. }
   16. }

Dari kode di atas terlihat **List<Planet> lp** tidak diizinkan menampung objek selain Planet. Dalam kasus ini **List<Planet> lp** dilindungi **compile-time type safety** di mana jika ada objek lain yang dimasukkan ke **List<Planet> lp** . Tetapi dengan tipe selain **Planet** , seketika compile error, Artinya deteksilah lebih dahulu sebelum runtime (program dijalankan). Bandingkan dengan **List lo** yang bisa menampung objek String ataupun Planet (bahkan semua jenis objek). Sebagai contoh, kita ingin loop kedua List tersebut untuk memanggil method print dari class Planet.

1. Ubahlah kelas Main menjadi seperti ini:
   1. package com.dicoding.javafundamental.generics;
   3. import java.util.ArrayList;
   4. import java.util.List;
   6. public class Main {
   8. public static void main(String[] args) {
   9. List lo = new ArrayList(); // List tanpa type-parameter
   10. lo.add("lo - String 1"); // lo menampung objek String
   11. lo.add(new Planet("Mercury", 0.06)); // lo menampung objek Planet
   13. **for (Object o : lo) {**
   14. **Planet p = (Planet) o; // perlu type-casting dari Object ke Planet**
   15. **p.print();**
   16. **}**
   18. List<Planet> lp = new ArrayList(); // List dengan type-parameter Planet
   19. lp.add(new Planet("Mercury", 0.06)); // lp menampung objek Planet
   20. **//lp.add("lp - String 1"); // baris ini compile-error, lp tidak diijinkan menampung objek String**
   21. **lp.add(new Planet("Venus", 0.82));**
   22. **for (Planet p : lp) {**
   23. **p.print();**
   24. **}**
   25. }
   26. }
2. Selanjutnya jalankanlah kode di atas pada IDE yang kalian gunakan. Bila sukses, seharusnya Console akan menampilkan **output**seperti ini.

|  |
| --- |
| Exception in thread "main" java.lang.ClassCastException: class java.lang.String cannot be cast to class com.dicoding.javafundamental.generics.Planet (java.lang.String is in module java.base of loader 'bootstrap'; com.dicoding.javafundamental.generics.Planet is in unnamed module of loader 'app') at com.dicoding.javafundamental.generics.Main.main(Main.java:14) |

1. Perhatikan kode untuk loop masing-masing List. Untuk **List lo** perlu dilakukan **type-casting** sedangkan untuk **List<Planet> lp** tidak perlu. Dalam kasus ini terlihat penggunaan Generics membuat kode lebih sederhana seperti yang sudah di bahas di paragraf awal. Dari kode di atas tidak ada compile-error tetapi saat kita jalankan akan terjadi runtime-error. Sebabnya, dalam **List lo** ada objek yang tidak bisa di-cast ke Planet.
2. Suatu saat kode program kita akan menjadi besar, ribuan baris kode! Bayangkan jika kita tidak menggunakan type-parameter saat deklarasi objek Collection. Lalu objek tersebut digunakan di tempat lainnya, misal dikirim sebagai parameter ke suatu method. Bisa saja tanpa sadar kita memasukkan tipe objek yang salah dan baru akan ketahuan ketika program dijalankan. Pusing kan? Nah, gunakan type-parameter. Paling tidak kita akan terbantu karena masalah terdeteksi lebih dini saat compile.

## Wildcards

Perhatikan baris kode di bawah ini.

1. package com.dicoding.javafundamental.generics;
3. import java.util.ArrayList;
4. import java.util.Collection;
5. import java.util.List;
7. public class Wildcard {
8. public static void main(String[] args) {
9. List<String> ls = new ArrayList();
10. ls.add("String1");
11. ls.add("String2");
13. **print(ls); // Apakah baris ini valid?**
15. Collection<Planet> cp = new ArrayList();
16. cp.add(new Planet("Mercury", 0.06));
17. cp.add(new Planet("Venus", 0.82));
19. **print(cp); // Apakah baris ini valid?**
20. }
22. public static void print(Collection<Object> collection) {
23. for (Object o : collection) {
24. System.out.println(o);
25. }
26. }
27. }

Apakah baris yang ditanyakan di atas valid? Coba pindahkan ke IntelliJ. Jawabannya adalah tidak valid. Loh kenapa begitu? Berdasarkan materi Inheritance, bukankah **class String dan class Planet adalah turunan dari class Object**?

Ya benar **tetapi List<String> dan Collection<Planet> bukan turunan (subtype) dari Collection<Object>** atau Collection<Object> bukan supertype dari Collection<Planet> dan List<String>. Lalu apa supertype dari semua tipe Collection? Hal ini dikenal dengan nama **wildcard type**, ditulis dengan syntax **Collection<?>**yang artinya collection of unknown. Kita tulis ulang sebagian kode di atas menggunakan wildcard type.

1. package com.dicoding.javafundamental.generics;
3. import java.util.ArrayList;
4. import java.util.Collection;
5. import java.util.List;
7. public class Wildcard {
8. public static void main(String[] args) {
9. List<String> ls = new ArrayList();
10. ls.add("String1");
11. ls.add("String2");
13. print(ls); // Baris ini valid
15. Collection<Planet> cp = new ArrayList();
16. cp.add(new Planet("Mercury", 0.06));
17. cp.add(new Planet("Venus", 0.82));
19. print(cp); // Baris ini valid
20. }
22. public static void print(**Collection<?>** **collection**) { **// perubahan ada di baris ini**
23. for (Object o : collection) {
24. System.out.println(o);
25. }
26. }
27. }

Sekarang kode kita sebelumnya menjadi **valid**. Perhatikan sekarang method print() menjadi bisa dipanggil dengan tipe Collection yang berbeda atau reuseable.

## Generic Methods

Perhatikan kode di bawah ini.

1. static void arrayToCollection(**Object[] a**, Collection<?> c) {
2. for (**Object o : a**) {
3. c.add(o); ***// baris ini tidak valid***
4. }
5. }

Kode di atas tidak valid karena **Collection<?> c** adalah collection of unknown type dan kita menambahkan tipe Object **o**. Sekarang kita ubah kode kita menjadi sebagai berikut.

1. static **<T>** void arrayToCollection(**T[] a**, Collection<T> c) {
2. for (**T o : a**) {
3. c.add(o); ***// baris ini valid***
4. }
5. }

Kode kita menjadi menarik bukan? Perhatikan penambahan huruf **T**. Kita bisa mengganti dengan huruf apapun, tetapi anjuran coding convention menggunakan huruf T, mengacu ke type. Cara penulisan kode seperti di atas dikenal dengan istilah **Generic Methods**. Lalu kita bisa gunakan method di atas dengan cara seperti di bawah ini.

1. package com.dicoding.javafundamental.generics;
3. import java.util.ArrayList;
4. import java.util.Collection;
6. public class GenericsMethod {
8. private static <T> void arrayToCollection(T[] a, Collection<T> c) {
9. for (T o : a) {
10. c.add(o); // baris ini valid
11. }
12. }
14. public static void main(String[] args) {
15. Object[] oa = new Object[100];
16. Collection<Object> co = new ArrayList();
18. **// T inferred to be Object**
19. arrayToCollection(oa, co);
21. String[] sa = new String[100];
22. Collection<String> cs = new ArrayList();
24. **// T inferred to be String**
25. arrayToCollection(sa, cs);
27. **// T inferred to be Object**
28. arrayToCollection(sa, co);
30. Integer[] ia = new Integer[100];
31. Float[] fa = new Float[100];
32. Number[] na = new Number[100];
33. Collection<Number> cn = new ArrayList();
35. **// T inferred to be Number**
36. arrayToCollection(ia, cn);
38. **// T inferred to be Number**
39. arrayToCollection(fa, cn);
41. **// T inferred to be Number**
42. arrayToCollection(na, cn);
44. **// T inferred to be Object**
45. arrayToCollection(na, co);
47. **// compile-error**
48. //arrayToCollection(na, cs);
49. }
50. }

## Wildcards vs Generic Methods

Setelah kita pelajari wildcards dan generic methods timbul pertanyaan baru. Kapan kita menggunakan wildcards dan kapan menggunakan generic methods? Jawaban sederhananya adalah: ketika balikan (return-type) dari suatu method tidak bergantung kepada tipe parameter (parameter-type). Atau kita hanya ingin memanfaatkan fitur polymorphism untuk tipe parameter method tersebut maka gunakanlah wildcards. Jika ada keterkaitan antara return-type dan parameter-type maka gunakanlah generic methods. Untuk memperjelas maksud kalimat di atas mari kita simak kode di bawah ini.

1. package com.dicoding.javafundamental.generics;
3. import java.util.ArrayList;
4. import java.util.Collection;
6. public class WildcardVSGenericsMethod {
7. **// mengunakan wildcards**
8. static void print(Collection<?> col) {
9. for (Object o : col) {
10. System.out.println(o);
11. }
12. }
14. **// menggunakan generic methods**
15. static <T> Collection arrayToCollection(T[] a) {
16. Collection<T> c = new ArrayList();
17. for (T o : a) {
18. c.add(o);
19. }
20. return c;
21. }
23. public static void main(String[] args) {
24. String[] sa = { "Happy", "Coding" };
25. Collection r = arrayToCollection(sa);
26. print(r);
27. }
28. }

Pada metode print(), kita menggunakan wildcard sebagai parameternya agar dapat bekerja dengan beragam tipe data. Penggunaan wildcard juga dapat kita lakukan saat menuliskan kode metode generic class yang tidak bergantung pada tipe parameternya. Misalnya, List.size atau List.clear. Sementara pada metode arrayToCollection, kita mengembalikan nilai dengan tipe Collection dan untuk tipe parameter kita gunakan T. Tanpa harus menuliskan <String>arrayToCollection(sa). Fitur ini dinamakan type inference, memungkinkan  kita untuk memanggil generic method sebagai metode biasa, tanpa menentukan jenis antara kurung sudut.

# Exception

Exception adalah event (kejadian) yang mengacaukan jalannya suatu program. Worst case scenario ketika suatu program mengalami exception adalah termination. Termination (penutupan) program adalah hal yang harus dihindari. Untuk itu kita harus menangani exception yang terjadi di program, atau yang biasa disebut sebagai handle exception.

Kode yang baik adalah yang terhindar dari segala bentuk kejadian yang menyebabkan efek buruk kepada program. Oleh karena itu mari kita kenali dulu berbagai macam exception yang ada di Java.

Pada dasarnya ada 3 jenis exception berdasarkan kategorinya.

1. **Checked Exception,**adalah exception yang terjadi saat compile time. Di sini programmer harus menambahkan kode untuk meng-handle exception kategori ini. Jika tidak di-handle maka kode yang dituliskan akan error dan tidak akan bisa dikompilasi. Contohnya adalah exception java.io.FileNotFoundException.
2. **Unchecked Exception,**adalah exception yang terjadi saat execution time. Eror ini terjadi dalam lingkup internal dari aplikasi kita, biasanya terjadi karena salah penulisan kode atau penggunaan salah terhadap satu API. Contohnya adalah NullPointerException.
3. **Error,**adalah exception yang diluar kendali user atau programmer. Eror ini terjadi di lingkup eksternal dari aplikasi kita. Ketika exception ini terjadi maka tidak ada yang bisa kita lakukan untuk mengatasinya, contohnya ketika perangkat kerasnya rusak saat kita ingin membaca data.

Dari definisi di atas maka error exception dan unchecked exception termasuk dari exception yang berada pada execution time. Sebabnya, keduanya hanya dialami ketika program sudah berjalan. Perbedaanya adalah unchecked berada di dalam internal program kita, sedangkan error exception berada di eksternal program kita.

Kemudian kode apa yang harus kita tambahkan untuk mengatasi berbagai macam exception tersebut? Kita harus kenal dengan 3 block kode yaitu try, catch, dan finally.

## Try-Catch

Kode yang rawan dengan exception kita masukkan ke dalam block try-catch. Kode yang kita masukkan ke dalam block try-catch biasa disebut sebagai protected code. Kodenya sepert ini.

1. try{
2. *// Protected code*
3. } catch (Exception e){
4. *// Catch block*
5. }

### Codelab Try-Catch

Misalnya, program kita mencoba akses ke filereader. FileReader merupakan komponen yang dapat membaca suatu berkas.

1. Buatlah proyek baru dengan nama **Exception** dengan nama package com.dicoding.javafundamental.exception di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama Main, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.exception;
   3. import java.io.File;
   4. import java.io.FileReader;
   6. public class Main {
   7. **public static void main(String[] args) {**
   8. **String location = "D://namafile.txt";**
   9. **File file = new File(location);**
   10. **FileReader fr = new FileReader(file);**
   11. **}**
   12. }

Jika diperhatikan, kode di atas akan mengalami eror **"Unhandled exception: java.io.FileNotFoundException"**.Ini dikarenakan exception java.io.FileNotFoundException merupakan checked exception. Maka kita perlu memasukkannya ke dalam try-catch agar bisa di -compile.

Lalu, bagaimana jalanya program ketika suatu kode kita masukkan ke dalam **try-catch**? Normalnya, program akan menjalankan kode yang ada di dalam block **try**. Jika tidak ada kesalahan yang terjadi, maka program akan berjalan normal sampai baris kode di dalam try selesai. Dan jika ketika terjadi eror di dalam try, maka program akan keluar dari block try dan menjalankan block catch.

1. Kita gunakan contoh kode yang sebelumnya yaitu membaca berkas dengan menggunakan komponen filereader. Berkas yang akan kita coba baca mempunyai nama latihan.txt. Ubahlah kelas Main menjadi seperti ini:
   1. package com.dicoding.javafundamental.exception;
   3. import java.io.File;
   4. import java.io.FileReader;
   6. public class Main {
   7. public static void main(String[] args) {
   8. **try {**
   9. **// Mencoba membaca berkas latihan.txt**
   10. **File file = new File("D://latihan.txt");**
   11. **FileReader fr = new FileReader(file);**
   12. **// Jika berhasil maka tampilkan pesan**
   13. **System.out.println("Read file berhasil");**
   14. **} catch (Exception e) {**
   15. **// Jika terjadi kesalahan maka tampilkan pesan**
   16. **System.out.println(e.getMessage());**
   17. **}**
   18. }
   19. }
2. Kita asumsikan ada 2 skenario yang bisa terjadi ketika kode di atas dijalankan. **Pertama**, ada berkas latihan.txt di drive d: dan program berhasil membaca berkasnya. **Kedua**, berkas latihan.txt tidak ada di drive d:. Jika dijalankan dengan kedua skenario maka output-nya adalah :

Output skenario berkas ditemukan:

|  |
| --- |
| Read file berhasil |

Output skenario berkas tidak ditemukan:

|  |
| --- |
| D:\latihan.txt (The system cannot find the file specified) |

Dari output di atas maka dapat disimpulkan bahwa kode untuk menampilkan pesan “Read file berhasil” tidak dijalankan karena terjadi exception ketika mencoba membaca berkas latihan.txt. Program akan langsung menuju ke block catch dan menampilkan pesan penyebab dari erornya.

## Multiple catch

Dari kode sebelumnya kita menggunakan Exception untuk menangani semua exception yang terjadi. Sebenarnya, kita bisa memilih tipe exception apa saja yang hanya ingin kita handle. Ini sangat berguna ketika kita ingin meng-handle tiap exception dengan perlakuan yang berbeda.

Lebih lanjut, tidak hanya 1 exception saja yang bisa kita tangani. Kita bisa menambahkan lebih dari 1 block catch. Misalnya seperti ini.

1. try {
3. } catch (ExceptionType name) {
5. } catch (ExceptionType name) {
7. }

Dan setelah Java 7 kita bisa menggunakan 1 baris kode untuk meng-handle multi catch. Seperti ini contohnya.

1. catch (IOException|SQLException ex) {
2. logger.log(ex);
3. throw ex;
4. }

## Finally

Block finally adalah block yang di tambahkan di akhir block try-catch. Finally akan selalu dijalankan setelah try-catch baik terjadi exception atau tidak. Finally bermanfaat ketika kita ingin melakukan cleanup code. Cleanup code di sini maksudnya adalah de-alokasi sumber daya, Artinya semua sumber daya yang dibuka atau digunakan pada blok try seperti koneksi jaringan, database, berkas, stream, dll akan ditutup dengan menjalankan instruksi yang ditulis pada blok finally.

Contohnya adalah seperti ini.

1. package com.dicoding.javafundamental.exception;
3. public class Main {
4. **public static void main(String[] args) {**
5. **int[] a = new int[5];**
6. **try {**
7. **System.out.println("Akses elemen ke 5 :" + a[5]);**
8. **} catch (ArrayIndexOutOfBoundsException e) {**
9. **System.out.println("Exception thrown :" + e);**
10. **} finally {**
11. **a[4] = 10;**
12. **System.out.println("Nilai elemen terakhir: " + a[4]);**
13. **}**
14. **}**
15. }

Output

|  |
| --- |
| Exception thrown :java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5 Nilai elemen terakhir: 10 |

Contoh program di atas adalah contoh akses elemen array yang melebihi dari jumlah indeks array-nya. Program akan masuk ke block catch, kemudian akan menjalankan kode yang ada di block finally, yaitu mengisi nilai indeks ke-4 array dengan 10.

# Input Stream dan Output Stream

Yup kita telah belajar tentang input output sederhana. Kali ini kita belajar akan proses input output yang agak sedikit kompleks dengan menggunakan package java.io. Package java.io berisi komponen yang dapat kita gunakan untuk proses input dan output (I/O) di dalam Java. Input adalah proses untuk membaca dari berkas ke dalam program, sedangkan output adalah proses untuk menuliskan ke berkas.

Sebelum terjun lebih jauh kita harus mengetahui istilah stream terlebih dahulu. Stream adalah proses secara sekuensial elemen data dari waktu ke waktu. Stream bisa diibaratkan seperti barang-barang yang diproses satu per satu pada ban berjalan (conveyor belt).

Kenapa stream? Karena proses input output akan dijalankan secara stream, di mana prosesnya akan berjalan secara sekuensial.

Di Java ada 2 macam stream yaitu inputStream dan outputStream.

1. InputStream, digunakan membaca data dari sumber.
2. OutputStream, digunakan untuk menuliskan data ke suatu destination (target sumber).

Ada beberapa io stream yang akan kita pelajari seperti byte streams, character streams, standar streams, dan lain-lain.

## Byte Streams

Byte streams digunakan untuk proses input output dengan ukuran 8-bit bytes. Contoh komponen byte stream adalah FileInputStream dan FileOutputStream. Ada banyak komponen byte streams lainnya tapi kurang lebih penggunaanya sama.

1. public class Main {
2. public static void main(String[] args) {
3. **FileInputStream in = null;**
4. **FileOutputStream out = null;**
6. **try {**
7. **in = new FileInputStream("latihan\_input.txt");**
8. **out = new FileOutputStream("latihan\_ouput.txt");**
9. **int c;**
11. **while ((c = in.read()) != -1) {**
12. **out.write(c);**
13. **}**
14. **} catch (IOException e) {**
15. **e.printStackTrace();**
16. **} finally {**
17. **try {**
18. **if (in != null) {**
19. **in.close();**
20. **}**
21. **if (out != null) {**
22. **out.close();**
23. **}**
24. **} catch (IOException e) {**
25. **e.printStackTrace();**
26. **}**
27. **}**
28. }
29. }

Di dalam block finally kita tidak boleh lupa untuk menutup stream. Lupa untuk menutup stream akan menyebabkan kebocoran pada resource (resource leaks).

## Character Streams

Java menggunakan Unicode conventions untuk menyimpan data characternya. Character stream digunakan untuk memproses input output dari 16-bit unicode. Ada banyak komponen character streams tapi yang sering digunakan adalah FileReader dan FileWriter. Contoh kodenya seperti ini.

1. package com.dicoding.javafundamental.inputoutput;
3. import java.io.FileReader;
4. import java.io.FileWriter;
5. import java.io.IOException;
7. public class Main {
8. public static void main(String[] args) {
9. **FileReader in = null;**
10. **FileWriter out = null;**
11. **try {**
12. **in = new FileReader("latihan\_input.txt");**
13. **out = new FileWriter("latihan\_ouput.txt");**
14. **int c;**
15. **while ((c = in.read()) != -1) {**
16. **out.write(c);**
17. **}**
18. **} catch (IOException e) {**
19. **e.printStackTrace();**
20. **} finally {**
21. **try {**
22. **if (in != null) {**
23. **in.close();**
24. **}**
25. **if (out != null) {**
26. **out.close();**
27. **}**
28. **} catch (IOException e) {**
29. **e.printStackTrace();**
30. **}**
31. **}**
32. }
33. }

Penggunaan komponen byte streams FileInputStream dan FileOutputStream tidak jauh beda. Sebabnya, di dalam FileReader dan FileWriter terdapat komponen byte streams tersebut. Yang membedakan adalah FileReader dan FileWriter dapat membaca data 2 bytes dalam satu waktu.

Lalu apakah kita harus menggunakan byte streams atau character streams? Byte streams termasuk low-level I/O, jadi gunakanlah ketika ingin memproses data primitive. Ketika data yang ingin kita proses memiliki character dengan Unicode conventions maka sebaiknya gunakan character streams.

## Hierarki Input Output Streams

Hierarki dari kelas input dan output streams bisa dilihat pada gambar di bawah ini.

Bisa dilihat bahwa ada banyak macam kelas streams yang bisa kita gunakan di Java. Tidak perlu belajar semua komponen yang ada. Akan tetapi yang perlu kita pahami adalah perbedaan antara InputStream dan OutputStream.

## File Navigation

Navigation input output tidak kalah penting dengan proses input output. Kita juga harus memahami directories di dalam Java. Directories dalam Java adalah File yang dapat memuat daftar berkas dan direktori. Kita menggunakan objek File untuk membuat directories lalu untuk menampilkan daftar berkas yang tersedia di dalam direktori tertentu.

Untuk navigasi, kita bisa menggunakan objekFile yang bisa menampilkan daftar berkas dan direktori. Dengan menggunakan objek File, kita bisa membuat directory dengan menggunakan fungsi mkdir() atau mkdirs.

1. mkdir(), metode untuk membuat directory. Nilainya true ketika sukses dan false ketika gagal. Gagal bisa disebabkan oleh path directory yang sudah yang ada, atau karena keseluruhan path nya tidak ada.
2. mkdirs(), metode yang digunakan untuk membuat directory dan parent directory-nya.

Misalnya seperti ini untuk membuat suatu directory.

1. package com.dicoding.javafundamental.inputoutput;
3. import java.io.File;
5. public class Main {
6. public static void main(String[] args) {
7. **String dirname = "/java/latihan1";**
8. **File file = new File(dirname);**
9. **// Buat directory**
10. **file.mkdirs();**
11. }
12. }

Karena menggunakan mkdirs, kode di atas akan membuat directory latihan1 dan parent directory java.

Kemudian kita bisa menampilkan list file dari directories dengan memanggil metode list(). Misalnya seperti ini.

1. package com.dicoding.javafundamental.inputoutput;
3. import java.io.File;
5. public class Main {
6. public static void main(String args[]) {
7. **String dirname = "/java/latihan1";**
8. **File file = null;**
9. **String[] paths;**
10. **try {**
11. **// Instansiasi objek File**
12. **file = new File(dirname);**
13. **// Ambil list files dan masukkan ke string paths**
14. **paths = file.list();**
15. **// Tampilkan semua path yang ada**
16. **for (String path : paths) {**
17. **System.out.println(path);**
18. **}**
19. **} catch (Exception e) {**
20. **e.printStackTrace();**
21. **}**
22. }
23. }

Dengan memanfaatkan mkdir dan list, kita bisa membuat program yang dapat melakukan navigasi ke directories di dalam storage.

### DateTime

Java menyediakan beberapa kelas untuk menangani DateTime. Kelas-kelas ini menangani segala yang berhubungan dengan tanggal dan waktu. Pada materi ini kita akan belajar bagaimana menggunakan, menentukan, merubah serta menampilkan dengan format waktu belahan dunia yang diinginkan.

Berikut kelas yang sering digunakan untuk mendukung pengelolaan DateTime

|  |  |
| --- | --- |
| **Kelas** | **Deskripsi** |
| System.currentTimeMillis() | Method yang mengembalikan waktu sekarang dalam millisecond |
| java.util.Date | Kelas yang mewakili tanggal dan waktu. |
| java.util.Calendar | Kelas yang memiliki spesifik method untuk mengkonversi waktu dan tanggal |
| java.text.SimpleDateFormat | Kelas yang membantu menterjemahkan format waktu dalam bentuk String ke format date time serta sebaliknya |

## CurrentTimeMillis

Merupakan method bagian dari kelas System yang menampilkan waktu sekarang dalam satuan milisecond (1000 milisecond = 1 second). CurrentTimeMillis() mengembalikan jumlah waktu mulai dari 1-1-1970 sampai dengan waktu yang sekarang. System.currentTimeMillis() biasanya digunakan untuk mengukur suatu waktu yang dibutuhkan dengan lebih spesifik. Ia biasa juga dipakai untuk tambahan sebagai variabel unik.

### Codelab CurrentTimeMillis

Mari kita coba penerapannya di kode berikut:

1. Buatlah proyek baru dengan nama **Datetime** dengan nama package com.dicoding.javafundamental.datetime di dalamnya:
2. Buatlah sebuah kelas baru di dalamnya dengan nama ExampleSystemCurrentMilis, kemudian tambahkan kode berikut:
   1. package com.dicoding.javafundamental.datetime;
   3. public class ExampleSystemCurrentMilis {
   4. **public static void main(String[] args) {**
   5. **long timeNow = System.currentTimeMillis();**
   6. **System.out.println("Waktu sekarang adalah " + timeNow + " milisecond");**
   7. **}**
   8. }
3. Melalui perintah System.currentTimeMillis();, saat kode diatas dieksekusi,  maka waktu yang didapat saat itu disimpan pada variable timeNow yang bertipe data **long**. Data yang disimpan pada timeNow dapat diolah untuk hal lain atau ditampilkan seperti pada hasil di bawah ini:

|  |
| --- |
| Waktu sekarang adalah 1563792106146 milisecond |

## Date

Merupakan kelas tertua yang menangani Waktu dan Tanggal. Dalam penggunaannya sekarang sudah banyak method yang sudah deprecated karena adanya java.util.Calendar. Kelas Calendar memiliki kemampuan penanganan waktu dan tanggal yang lebih baik. Tetapi jangan kuatir, masih ada beberapa kebutuhan kode yang bisa diselesaikan dengan menggunakan Date.

### Codelab Date

Mari kita praktikan codelab berikut:

1. Bukalah kembali proyek **Datetime.**
2. Buatlah kelas baru dengan nama ExampleDate dan masukkan kode berikut ke dalam kelas tersebut:
   1. package com.dicoding.javafundamental.datetime;
   3. import java.util.Date;
   5. public class ExampleDate {
   6. **public static void main(String[] args) {**
   7. **Date date = new Date();**
   8. **System.out.println("Tanggal sekarang adalah " + date.toString());**
   9. **}**
   10. }

Untuk menghasilkan tanggal dan waktu sekarang java.util.Date kita perlu lakukan inisialisasi lalu panggil dengan method date.toString() untuk menampilkan hasil tanggal sekarang setelah dijalankan programnya.

1. Jalankan tersebut maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Tanggal sekarang adalah Mon Jul 22 17:45:41 WIB 2019 |

## Calendar

Kelas dari grup bagian dari Java.util yang menggantikan fungsi date untuk mengkonversi data tanggal dan waktu secara instan. Kelas Calendar ini memiliki banyak fungsi untuk parsing dan manipulasi tanggal. Berikut hal penting yang perlu diperhatikan tentang mengapa kita menggunakan kelas Calendar ini:

1. Kelas ini menyediakan ruang dan method untuk mengimplementasikan sistem kalendar yang berada di luar dari bagian java util Calendar
2. Kelas ini juga dapat menentukan dan mengukur waktu sesuai yang diinginkan

### Codelab Calendar Menampilkan Waktu Sekarang

1. Bukalah kembali proyek **Datetime.**
2. Buatlah kelas baru dengan nama ExampleCalendardan masukkan kode berikut ke dalam kelas tersebut:
   1. package com.dicoding.javafundamental.datetime;
   3. import java.util.Calendar;
   5. public class ExampleCalendar {
   6. public static void main(String[] args) {
   8. **// Menampilkan waktu sekarang**
   9. **Calendar calendar = Calendar.getInstance();**
   10. **System.out.println("Waktu sekarang adalah " + calendar.getTime());**
   11. }
   12. }
3. Jalankan tersebut maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Waktu sekarang adalah Mon Jul 22 17:48:25 WIB 2019 |

1. Seperti halnya pada Date, inisilasisasi Calendar dipanggil dengan Calendar.getInstance(). Selanjutnya menampilkan waktu yang sekarang dengan memanggil method yang calendar.getTime(). Bila kode di atas dijalankan maka akan tampillah waktu dan tanggal ketika program dieksekusi.

### Codelab Calendar Menampilkan Secara Terpisah

Calendar memiliki banyak sekali method yang mempermudah untuk mengelola waktu dan tanggal. Contoh di bawah ini memperlihatkan bagaimana Anda dapat mengambil waktu / tanggal spesifik yang dinginkan untuk diolah. Misalnya Anda hanya ingin menampilkan tahun, bulan atau hari saja.

1. Bukalah kembali proyek **Datetime.**
2. Buatlah kelas baru dengan nama ExampleSplitCalendardan masukkan kode berikut ke dalam kelas tersebut:
   1. package com.dicoding.javafundamental.datetime;
   3. import java.util.Calendar;
   5. public class ExampleSplitCalendar {
   6. **public static void main(String[] args) {**
   7. **// Menampilkan waktu sekarang**
   8. **Calendar calendar = Calendar.getInstance();**
   9. **System.out.println("Waktu sekarang adalah " + calendar.getTime());**
   10. **// menampilkan spesifik waktu yang diinginkan**
   11. **System.out.println("Tanggal : " + calendar.get(Calendar.DATE));**
   12. **System.out.println("Bulan   : " + calendar.get(Calendar.MONTH));**
   13. **System.out.println("Tahun   : " + calendar.get(Calendar.YEAR));**
   14. **}**
   15. }
3. Jalankan tersebut maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Waktu sekarang adalah Mon Jul 22 17:51:37 WIB 2019 Tanggal : 22 Bulan : 6 Tahun   : 2019 |

1. Calendar memiliki kemampuan untuk menampilkan waktu/tanggal secara spesifik. Method get(Calendar.DATE) milik Calendar membutuhkan parameter untuk penggunaannya. Seluruh field dapat Anda cek di <https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html>.

### Codelab Calendar Menampilkan Waktu Tertentu yang Diinginkan

1. Bukalah kembali proyek **Datetime.**
2. Buatlah kelas baru dengan nama ExampleSpecificCalendardan masukkan kode berikut ke dalam kelas tersebut:
   1. package com.dicoding.javafundamental.datetime;
   3. import java.util.Calendar;
   5. public class ExampleSpecificCalendar {
   6. **public static void main(String[] args) {**
   7. **// Menampilkan waktu sekarang**
   8. **Calendar calendar = Calendar.getInstance();**
   9. **System.out.println("Waktu sekarang adalah " + calendar.getTime());**
   10. **// Menampilkan waktu 15 hari yang lalu**
   11. **calendar.add(Calendar.DATE, -15);**
   12. **System.out.println("15 hari yang lalu: " + calendar.getTime());**
   13. **// Menampilkan waktu 4 bulan yang akan datang**
   14. **calendar.add(Calendar.MONTH, 4);**
   15. **System.out.println("4 bulan kemudian: " + calendar.getTime());**
   16. **// Menampilkan waktu 2 tahun yang akan datang**
   17. **calendar.add(Calendar.YEAR, 2);**
   18. **System.out.println("2 tahun kemudian: " + calendar.getTime());**
   19. **}**
   20. }
3. Jalankan tersebut maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Waktu sekarang adalah Mon Jul 22 17:55:36 WIB 2019 15 hari yang lalu: Sun Jul 07 17:55:36 WIB 2019 4 bulan kemudian: Thu Nov 07 17:55:36 WIB 2019 2 tahun kemudian: Sun Nov 07 17:55:36 WIB 2021 |

1. Salah satu metode yang juga tersedia pada Calendar adalah add() yang memiliki 2 parameter yaitu calendar field dan banyaknya waktu yang ingin ditambahkan.  Seperti pada kode di atas untuk melihat waktu 15 hari yang lalu, kita bisa gunakan calendar.add(Calendar.DATE, -15) di mana -15 adalah 15 hari yang lalu dari waktu saat ini. Begitu juga untuk waktu bulan dan tahun.

## SimpleDateFormat

Dalam menampilkan suatu tanggal dan waktu, biasanya format tampilan di setiap kebutuhan akan berbeda-beda. Kalau hanya untuk sekedar menampilkan waktu dan tanggal , kita dapat menggunakan date atau calendar.

Bagaimana bila menampilkan format yang tidak biasa? Atau Anda ingin parsing data dari sebuah String waktu dan tanggal dengan format yang tidak biasa? Di sini SimpleDateFormat memang berfokus pada memformat dan parsing data tanggal dan waktu serta normalisasi waktu.

### Codelab Menampilkan Waktu Dengan Format yang Ditetapkan.

1. Bukalah kembali proyek **Datetime.**
2. Buatlah kelas baru dengan nama ExampleSimpleDateFormat dan masukkan kode berikut ke dalam kelas tersebut:
   1. package com.dicoding.javafundamental.datetime;
   3. import java.text.SimpleDateFormat;
   4. import java.util.Date;
   6. public class ExampleSimpleDateFormat {
   7. public static void main(String[] args) {
   9. **SimpleDateFormat sdf = new SimpleDateFormat("dd/M/yyyy");**
   10. **String dateFormated = sdf.format(new Date());**
   11. **System.out.println("Format tanggal default : " + new Date());**
   12. **System.out.println("Format tanggal dengan format : " + dateFormated);**
   13. }
   14. }
3. Jalankan tersebut maka hasilnya akan jadi seperti ini:

|  |
| --- |
| Format tanggal default : Mon Jul 22 17:59:24 WIB 2019 Format tanggal dengan format : 22/7/2019 |

1. Saat inisialisasi SimpleDateFormat, kita wajib mengisi parameter pattern. Pattern ini ("dd/M/yyyy") nantinya dijadikan acuan output dari waktu tanggal yang akan diparsing. Kita melakukan inisialisasi atas SimpleDateFormat new SimpleDateFormat("dd/M/yyyy"). Artinya output akan menampilkan (tanggal dalam 2 digit) / (Bulan 1 digit ) / (Tahun 4 digit).

# Casting

Sering kali kita ingin mengubah suatu tipe data tertentu ke bentuk tipe data lain dalam suatu variabel. Perubahan ini disebut dengan istilah casting. Untuk menggunakan casting di dalam Java, cukup deklarasikan tipe data yang baru di dalam statement asignment. Seperti contoh berikut ini :

Pada contoh di atas, kita mencoba mendeklarasikan variabel a sebagai integer. Akan tetapi nilai yang kita berikan merupakan tipe double. Dengan kata lain antara tipe data dan nilai tidak sesuai. Kita bisa memaksa keadaan ini dengan casting. Cara termudah untuk melakukan casting adalah arahkan cursor pada bagian ikon lampu berwarna merah sebelah kiri kemudian pilih cast to ‘int’

Dengan melakukan hal tersebut maka saat kita melakukan perubahan tipe data ke int. di dalam kode, akan muncul seperti berikut :

Apabila kita coba mencetak variabel a , maka output-nya akan menjadi seperti berikut:

Output

|  |
| --- |
| 3 |

Mengapa dapat terjadi demikian? Jawabannya merujuk kepada tipe data yang di-casting. Pada nilai yang sebelumnya, tipe datanya double. Maka saat melakukan casting untuk mengubah tipe data ke bentuk int, nilai tersebut akan berubah  menjadi int.

Kita dapat melakukan casting terhadap beberapa tipe data. Bagaimana jika kita akan melakukan casting dari double ke dalam bentuk string? Dengan cara mendeklarasikan variabel sebagai String dan menggunakan fungsi String.valueOf. Berikut ini contohnya:

1. double a = 3.14;
2. String hasil = String.valueOf(a);
3. System.out.println(hasil);

Output:

|  |
| --- |
| 3.14 |

Perlu diingat bahwa hasil dari output tersebut bukanlah lagi double melainkan sudah menjadi String.

Selain untuk mengubah atau mengkonversikan tipe data menjadi tipe data lain dalam suatu variabel, casting juga dapat digunakan untuk mengkonversi kelas menjadi kelas lain. Untuk menkonversi kelas terdapat dua metode yaitu **Upcasting**dan **Downcasting**. Upcasting dan downcasting terkait erat dengan konsep pewarisan. Sangat umum untuk menggunakan variabel referensi untuk merujuk pada tipe yang lebih spesifik. Dan setiap kali kita melakukan ini, upcasting secara implisit terjadi. Upcasting adalah casting dari subclass ke superclass sementara downcasting adalah casting dari superclass ke subclass.

Contoh penggunaan **upcasting** dan **downcasting** adalah sebagai berikut:

1. public class **Hewan** {
2. public void **makan**() {
3. // ...
4. }
5. }

Kita buat terlebih dahulu kelas Hewan

1. public class **Kucing** extends **Hewan** {
2. public void **makan**() {
3. // ...
4. }
6. public void **meow**() {
7. // ...
8. }
9. }

Selanjutnya buat kelas Kucing yang akan meng-extends kelas Hewan

1. public class Main {
2. public static void main(String[] args) {
3. Kucing anggora = new Kucing();
5. Hewan hewan = anggora; **//secara implisit upcasting**
6. hewan = (Hewan) anggora; **//secara explisit melakukan casting dari subclass ke superclass**
8. Hewan hewan2 = new Kucing();
9. ((Kucing) hewan2).meow(); **//downcast hewan ke kucing, casting dari superclass ke subclass**
10. }
11. }
12. Kucing anggora = new Kucing();

Setelah itu kita buat kelas Main, lalu kita instansiasi objek kucing anggora yang mereferensi variabel tipe kucing.

1. Hewan hewan = anggora; **//secara implisit upcasting**

Kita juga bisa menetapkan referensi anggora menjadi variabel tipe hewan. Karena pada kasus ini kucing adalah subclass sementara hewan adalah superclass. Dan pada saat kita menetapkan referensinya (assign).

1. hewan = (Hewan) anggora; **//secara explisit melakukan casting dari subclass ke superclass**

Pada kode program sebelumnya telah dilakukan upcasting secara implisit. Sementara untuk kode program di atas kita lakukan upcasting secara eksplisit. Kita tidak perlu melakukan upcasting secara eksplisit, karena pada Java, kompiler akan mengetahui bila kelas tersebut ada hubungan inheritance. Apabila kita hendak menggunakan metode yang terdapat pada kelas kucing untuk objek hewan, hal tersebut tidak dapat dilakukan misalnya hewan.meow(). Untuk bisa melakukannya, kita bisa melakukan downcasting.

1. Hewan hewan2 = new Kucing();
2. ((Kucing) hewan2).meow(); **//downcast hewan2 ke kucing, casting dari superclass ke subclass**

Sebelumnya kita telah mereferensikan variabel hewan ke instans kucing. Dan ketika ingin menggunakan metode yang ada pada kelas kucing kita bisa menggunakan downcast seperti pada kode program di atas untuk menggunakan metode meow yang terdapat pada kelas kucing. Downcast dilakukan hanya pada instans kucing.

Selesai sudah materi Java Fundamental. Menguasai bahasa Java adalah salah satu batu pertama yang harus kamu lompati untuk terjun ke dunia "persilatan" pengembangan aplikasi. Pondasi Java yang kuat akan menciptakan aplikasi yang excellent. Kami tunggu aplikasi dahsyat dari Anda.

Selanjutnya? Masih banyak kelas-kelas lainnya yang menarik untuk Anda ikuti.

Bisa dimulai dari kelas Reguler :

* [Belajar Membuat Aplikasi Android Pemula](https://www.dicoding.com/academies/51)

Atau bisa juga melanjutkan ke kelas Picodiploma :

* [Menjadi Android Developer Expert](https://www.dicoding.com/academies/14)
* [Kotlin Android Developer Expert](https://www.dicoding.com/academies/55)
* [Menjadi Game Developer Expert](https://www.dicoding.com/academies/47)
* [Membangun Progressive Web Apps](https://www.dicoding.com/academies/74)

# Studi Kasus

Kita akan membuat beberapa program sederhana untuk melatih dan mengimplementasikan materi dari modul yang telah kita pelajari. Beberapa program yang akan dibuat adalah :

1. **Studi Kasus : Implementasi Pernyataan If**
2. **Studi Kasus : Implementasi Perulangan For, Pernyataan If, dan Function dalam OOP**
3. **Studi Kasus : Implementasi Pernyataan Switch, Operator Aritmatika, dan OOP**

Mari kita ulik satu per satu dari studi kasus ini!

# Studi Kasus : Implementasi Pernyataan If

Program ini cukup sederhana. Kita harus menentukan wujud air apakah beku, cair atau gas. Masukan dari user berupa suhu akan diproses dengan menggunakan **kondisional IF** untuk tiga kasus karena kita kan menentukan tiga wujud air. Untuk tahap awalnya, masih sama dengan modul first java project, yang bisa dilihat di modul [First Java Project](https://www.dicoding.com/academies/60/tutorials/1860/).

1. Untuk studi kasus, kita bisa membuat project baru dan beri nama **SuhuAir**. Selanjutnya buatlah package di dalamnya dengana nama com.dicoding.javafundamental.suhuair.
2. Studi kasus pertama ini cukup mudah. Kita buat kelas baru dan beri nama SuhuAir. Kemudian kita buat static metode main.
   1. public class SuhuAir {
   2. public static void main(String[] args){
   4. }
   5. }
3. Kita tulis beberapa atribut dan kode program awal yang dibutuhkan
   1. package com.dicoding.javafundamental.suhuair;
   3. import java.util.Scanner;
   5. public class SuhuAir {
   6. **public static void main(String[] args){**
   7. **/\* contoh pemakaian IF tiga kasus : wujud air \*/**
   8. **/\*Kamus\*/**
   9. **int T;**
   10. **/\*Program\*/**
   11. **System.out.println("Contoh IF tiga kasus");**
   12. **System.out.print("Suhu (der. C) = ");**
   13. **//scanner untuk masukan suhu air**
   14. **Scanner scanner = new Scanner(System.in);**
   15. **T = scanner.nextInt(); //masukan suhu air dengan tipe int**
   16. **}**
   17. }
4. Lalu kita tambahkan kondisional **if-else**. Untuk kode program secara keseluruhan dapat dilihat berikut ini.
   1. package com.dicoding.javafundamental.suhuair;
   3. import java.util.Scanner;
   5. public class SuhuAir {
   6. public static void main(String[] args) {
   7. /\* contoh pemakaian IF tiga kasus : wujud air \*/
   9. /\*Kamus\*/
   10. int T;
   12. /\*Program\*/
   13. System.out.println("Contoh IF tiga kasus");
   14. System.out.print("Suhu (der. C) = ");
   16. //scanner untuk masukan suhu air
   17. Scanner scanner = new Scanner(System.in);
   18. T = scanner.nextInt(); //masukan suhu air dengan tipe int
   20. **//proses pengecekan dengan if**
   21. **if (T < 0) {**
   22. **System.out.println("Wujud air beku " + T);**
   23. **} else if ((0 <= T) && (T <= 100)) {**
   24. **System.out.println("Wujud air cair " + T);**
   25. **} else if (T > 100) {**
   26. **System.out.println("Wujud air uap/gas " + T);**
   27. **}**
   28. }
   29. }
5. Jika telah menuliskan kode di atas, coba jalankan dengan menekan tombol run. Setelah program di atas dijalankan, maka pada console-nya akan tampil output seperti berikut ini. Kita akan memasukan suhu air. Misalnya kita masukan 20 maka akan keluar hasil bahwa wujud air cair 20. Anda bisa coba masukkan suhu yang lain dan lihat output hasilnya.

|  |
| --- |
| Contoh IF tiga kasus Suhu (der. C) = **20** Wujud air cair 20 |

Kode ini masih bisa dieksplorasi lebih jauh dengan menambahkan fitur-fitur lainnya, misalnya kita buat program konversi suhu. Selamat berkreasi.

# Studi Kasus : Implementasi Perulangan For, Pernyataan If, dan Function dalam OOP

Pada program ini kita akan membuat program yang menghitung jumlah huruf vokal dan konsonan dari masukan user yang bertipe data String. Untuk membuatnya kita bisa menggunakan perulangan. Dengan perulangan kita dapat mengakses karakter yang terdapat pada String melalui setiap indeksnya.

1. Untuk studi kasus, kita bisa membuat project baru dan beri nama **VokalKonsonan**. Selanjutnya buatlah package di dalamnya dengana nama com.dicoding.javafundamental.vokalkonsonan.
2. Studi kasus pertama ini cukup mudah. Kita buat kelas baru dan beri nama VokalKonsonan. Kemudian kita buat static metode main.
   1. public class VokalKonsCount {
   2. public static void main(String[] args){
   4. }
   5. }
3. Di studi kasus yang kedua ini kita akan coba membuat fungsi baru. Buat 2 fungsi statis untuk menghitung huruf konsonan dan vokal dengan nama num\_vokal dan num\_konsonan.
   1. package com.dicoding.javafundamental.vokalkonsonan;
   3. public class VokalKonsonan {
   4. public static void main(String[] args) {
   6. }
   8. **private static int num\_vokal(String word) {**
   9. **int i;**
   10. **int jumlah\_vokal = 0;**
   11. **for (i = 0; i < word.length(); i++) {**
   12. **if (word.charAt(i) == 'a' || word.charAt(i) == 'i' || word.charAt(i) == 'u' || word.charAt(i) == 'e' || word.charAt(i) == 'o') {**
   13. **jumlah\_vokal++;**
   14. **}**
   15. **}**
   16. **return jumlah\_vokal;**
   17. **}**
   18. **private static int num\_konsonan(String word) {**
   19. **int i;**
   20. **int jumlah\_konsonan = 0;**
   21. **for (i = 0; i < word.length(); i++) {**
   22. **if (word.charAt(i) != 'a' && word.charAt(i) != 'i' && word.charAt(i) != 'u' && word.charAt(i) != 'e' && word.charAt(i) != 'o' && word.charAt(i) != ' ') {**
   23. **jumlah\_konsonan++;**
   24. **}**
   25. **}**
   26. **return jumlah\_konsonan;**
   27. **}**
   28. }
4. Tambahkan atribut dan program awal yang dibutuhkan.
   1. package com.dicoding.javafundamental.vokalkonsonan;
   3. import java.util.Scanner;
   5. public class VokalKonsonan {
   7. public static void main(String[] args) {
   8. **/\*Kamus\*/**
   9. **String word = "";**
   10. **int vokal = 0;**
   11. **int konsonan = 0;**
   12. **/\*Program\*/**
   13. **//masukan kalimat**
   14. **System.out.print("Masukan kalimat : ");**
   15. **Scanner scanner = new Scanner(System.in);**
   16. **word = scanner.nextLine();**
   17. }
   19. private static int num\_vokal(String word) {
   20. int i;
   21. int jumlah\_vokal = 0;
   23. for (i = 0; i < word.length(); i++) {
   24. if (word.charAt(i) == 'a' || word.charAt(i) == 'i' || word.charAt(i) == 'u' || word.charAt(i) == 'e' || word.charAt(i) == 'o') {
   25. jumlah\_vokal++;
   26. }
   27. }
   28. return jumlah\_vokal;
   29. }
   31. private static int num\_konsonan(String word) {
   32. int i;
   33. int jumlah\_konsonan = 0;
   35. for (i = 0; i < word.length(); i++) {
   36. if (word.charAt(i) != 'a' && word.charAt(i) != 'i' && word.charAt(i) != 'u' && word.charAt(i) != 'e' && word.charAt(i) != 'o' && word.charAt(i) != ' ') {
   37. jumlah\_konsonan++;
   39. }
   40. }
   41. return jumlah\_konsonan;
   42. }
   43. }
5. Silakan lihat kode program main secara keseluruhan berikut ini. Untuk mendapat nilai balik atau nilai yang dikirimnya dari metode yang telah dibuat, jangan lupa untuk panggil fungsi tersebut. Lalu simpan nilai baliknya.
   1. package com.dicoding.javafundamental.vokalkonsonan;
   3. import java.util.Scanner;
   5. public class VokalKonsonan {
   6. public static void main(String[] args) {
   7. /\*Kamus\*/
   8. String word = "";
   9. int vokal = 0;
   10. int konsonan = 0;
   12. /\*Program\*/
   13. //masukan kalimat
   14. System.out.print("Masukan kalimat : ");
   15. Scanner scanner = new Scanner(System.in);
   16. word = scanner.nextLine();
   18. //memanggil prosedur yang mengembalikan nilai int
   19. vokal = num\_vokal(word);
   20. konsonan = num\_konsonan(word);
   22. System.out.println("Jumlah huruf vokal : " +vokal);
   23. System.out.println("Jumlah huruf konsonan : " +konsonan);
   24. }
   26. private static int num\_vokal(String word) {
   27. int i;
   28. int jumlah\_vokal = 0;
   30. for (i = 0; i < word.length(); i++) {
   31. if (word.charAt(i) == 'a' || word.charAt(i) == 'i' || word.charAt(i) == 'u' || word.charAt(i) == 'e' || word.charAt(i) == 'o') {
   32. jumlah\_vokal++;
   33. }
   34. }
   35. return jumlah\_vokal;
   36. }
   38. private static int num\_konsonan(String word) {
   39. int i;
   40. int jumlah\_konsonan = 0;
   42. for (i = 0; i < word.length(); i++) {
   43. if (word.charAt(i) != 'a' && word.charAt(i) != 'i' && word.charAt(i) != 'u' && word.charAt(i) != 'e' && word.charAt(i) != 'o' && word.charAt(i) != ' ') {
   44. jumlah\_konsonan++;
   45. }
   46. }
   47. return jumlah\_konsonan;
   48. }
   49. }
6. Kemudian jalankan program tersebut. Setelah menjalankan program tersebut, masukkan suatu kalimat pada console untuk menghitung huruf vokal dan konsonannya. Misalnya, masukan kalimat "**dicoding**" lalu tekan **Enter** maka akan muncul jumlah huruf **vokal : 3** dan jumlah huruf **konsonan : 5**. Cobalah mengetikkan kalimat yang lain dan lihat outputnya.

|  |
| --- |
| Masukan kalimat : **dicoding** Jumlah huruf vokal : 3 Jumlah huruf konsonan : 5 |

Kalau masih ingin bereksplorasi, kita bisa mencoba menggunakan bilangan ASCI saat melakukan pengecekan apakan vokal/konsonan. Sehingga saat memasukan kalimat dengan huruf kapital atau non-kapital, tetap terhitung. Selamat berkreasi.

# Studi Kasus : Implementasi Pernyataan Switch, Operator Aritmatika, dan OOP

Program yang terakhir adalah menghitung luas dan keliling dari bangun datar. Kali ini kita akan mencoba menggunakan konsep atau paradigma OOP agar lebih mudah dalam proses pemrogramannya. Anda  membuat interface bangun datar agar dapat mengmplementasikannya pada setiap kelas bangun datar yang akan dibuat.

1. Untuk studi kasus, kita bisa membuat project baru dan beri nama **BangunDatar**. Selanjutnya buatlah package di dalamnya dengana nama com.dicoding.javafundamental.bangundatar.
2. Studi kasus pertama ini cukup mudah. Kita buat kelas **interface baru** dan beri nama BangunDatar. Kemudian tambahkanlah kode berikut:
   1. package com.dicoding.javafundamental.bangundatar;
   3. public interface BangunDatar {
   4. **public double luas(int s1, int s2);**
   5. **public double keliling(int s1, int s2);**
   6. }
3. Buatlah kelas baru dengan nama **PersegiPanjang**. Kemudian implementasikan interface BangunDatardengan mendeklarasikan setiap fungsi yang ada pada interface tersebut. Karena ini adalah implements maka tambahkan juga  statement isi fungsinya.
   1. package com.dicoding.javafundamental.bangundatar;
   3. public class PersegiPanjang **implements** **BangunDatar** {
   5. **PersegiPanjang() {**
   6. **}**
   7. **@Override**
   8. **public double luas(int s1, int s2) {**
   9. **return (s1 \* s2);**
   10. **}**
   11. **@Override**
   12. **public double keliling(int s1, int s2) {**
   13. **return (2 \* (s1 + s2));**
   14. **}**
   15. }
4. Buatlah kelas yang lain dengan nama SegitigaSiku**. I**mplementasikan juga interface-nya
   1. package com.dicoding.javafundamental.bangundatar;
   3. public class SegitigaSiku **implements BangunDatar** {
   4. **SegitigaSiku() {**
   5. **}**
   6. **@Override**
   7. **public double luas(int s1, int s2) {**
   8. **return (0.5 \* s1 \* s2);**
   9. **}**
   10. **@Override**
   11. **public double keliling(int s1, int s2) {**
   12. **return (s1 + s2 + Math.sqrt((s1 \* s1) + (s2 \* s2)));**
   13. **}**
   14. }
5. Buatlah kelas Lingkaran, kemudian implementasikan interface kembali. Pada kelas Lingkaranini, kita juga mencoba untuk menerapkan metode overloadkarena rumus menghitung luas dan keliling pada lingkaran cukup dengan satu parameter saja yaitu jari-jari.
   1. package com.dicoding.javafundamental.bangundatar;
   3. public class Lingkaran**implements BangunDatar**{
   4. **private double pi = 3.1415;**
   5. **Lingkaran() {**
   6. **}**
   7. **@Override**
   8. **public double luas(int s1, int s2) {**
   9. **return 0;**
   10. **}**
   11. **@Override**
   12. **public double keliling(int s1, int s2) {**
   13. **return 0;**
   14. **}**
   15. **public double luas(int r) { //metode overload**
   16. **return (pi \* r \* r);**
   17. **}**
   18. **public double keliling(int r) { //methode overload**
   19. **return (2 \* pi \* r);**
   20. **}**
   21. }
6. Terakhir buat kelas Mainuntuk menjalankan program. Pada kelas ini kita instansiasi kelas setiap bangun datar.
   1. package com.dicoding.javafundamental.bangundatar;
   3. public class Main {
   4. public static void main(String[] args) {
   5. **/\*Kamus\*/**
   6. **PersegiPanjang opersegi;**
   7. **SegitigaSiku osegitiga;**
   8. **Lingkaran olingkaran;**
   9. **int s1, s2, r;**
   10. **opersegi = new PersegiPanjang();**
   11. **osegitiga = new SegitigaSiku();**
   12. **olingkaran = new Lingkaran();**
   13. }
   14. }
7. Kita buat metode mainMenu. Metode ini digunakan untuk menampilkan pilihan menu.
   1. package com.dicoding.javafundamental.bangundatar;
   3. import java.util.Scanner;
   5. public class Main {
   7. **private static char pilihan;**
   8. **private static Scanner scanner = new Scanner(System.in);**
   10. public static void main(String[] args) {
   11. /\*Kamus\*/
   12. PersegiPanjang opersegi;
   13. SegitigaSiku osegitiga;
   14. Lingkaran olingkaran;
   15. int s1, s2, r;
   17. opersegi = new PersegiPanjang();
   18. osegitiga = new SegitigaSiku();
   19. olingkaran = new Lingkaran();
   20. }
   22. **private static void mainMenu(){**
   23. **//pilihan menu**
   24. **System.out.println("====================");**
   25. **System.out.println("Hitung Luas dan Keliling :");**
   26. **System.out.println("1. Persegi Panjang");**
   27. **System.out.println("2. Segitiga Siku");**
   28. **System.out.println("3. Lingkaran");**
   29. **System.out.print("Masukan Menu : ");**
   30. **pilihan = scanner.next().charAt(0);**
   31. **}**
   32. }
8. Lengkapi kode di dalam main menjadi seperti ini:
   1. package com.dicoding.javafundamental.bangundatar;
   3. import java.util.Scanner;
   5. public class Main {
   7. private static char pilihan;
   8. private static Scanner scanner = new Scanner(System.in);
   10. public static void main(String[] args) {
   11. /\*Kamus\*/
   12. PersegiPanjang opersegi;
   13. SegitigaSiku osegitiga;
   14. Lingkaran olingkaran;
   15. int s1, s2, r;
   17. opersegi = new PersegiPanjang();
   18. osegitiga = new SegitigaSiku();
   19. olingkaran = new Lingkaran();
   21. **/\*Program\*/**
   22. **//panggil metode**
   23. **mainMenu();**
   24. **//kondisional untuk pilihan menu yang dimasukan**
   25. **switch(pilihan){**
   26. **case '1':{**
   27. **System.out.print("Masukan panjang persegi: ");**
   28. **s1 = scanner.nextInt();**
   29. **System.out.print("Masukan lebar persegi: ");**
   30. **s2 = scanner.nextInt();**
   31. **System.out.println("Luas persegi panjang: " + opersegi.luas(s1,s2));**
   32. **System.out.println("Keliling persegi panjang: " + opersegi.keliling(s1,s2));**
   33. **break;**
   34. **}**
   35. **case '2':{**
   36. **System.out.print("Masukan alas segitiga siku: ");**
   37. **s1 = scanner.nextInt();**
   38. **System.out.print("Masukan tinggi segitiga siku: ");**
   39. **s2 = scanner.nextInt();**
   40. **System.out.println("Luas segitiga: " + osegitiga.luas(s1,s2));**
   41. **System.out.println("Keliling segitiga: " + osegitiga.keliling(s1,s2));**
   42. **break;**
   43. **}**
   44. **case '3':{**
   45. **System.out.print("Masukan jari-jari lingkaran: ");**
   46. **r = scanner.nextInt();**
   47. **System.out.println("Luas lingkaran: " + olingkaran.luas(r));**
   48. **System.out.println("Keliling lingkaran: " + olingkaran.keliling(r));**
   49. **break;**
   50. **}**
   51. **default :{**
   52. **System.out.println("tidak ada pilihan");**
   53. **break;**
   54. **}**
   55. **}**
   56. }
   58. private static void mainMenu(){
   59. //pilihan menu
   60. System.out.println("====================");
   61. System.out.println("Hitung Luas dan Keliling :");
   62. System.out.println("1. Persegi Panjang");
   63. System.out.println("2. Segitiga Siku");
   64. System.out.println("3. Lingkaran");
   65. System.out.print("Masukan Menu : ");
   66. pilihan = scanner.next().charAt(0);
   67. }
   68. }
9. Untuk menjalankan program yang telah dibuat, klik tombol **run**. Setelah menjalankan program tersebut, console akan menampilkan seperti berikut ini. Pertama akan muncul pilihan menu bangun datar yang akan kita hitung luas dan kelilingnya. Untuk mencoba, bisa pilih antara **1-3**. Misalnya Anda bisa pilih menu **1**, yaitu menghitung luas dan keliling persegi panjang. Maka Anda akan diminta untuk memasukkan **panjang** dan **lebar**. Setelah itu, akan keluar hasil perhitungan dari **luas** dan **keliling**persegi panjang. Outputnya akan jadi seperti ini:

|  |
| --- |
| ==================== Hitung Luas dan Keliling : 1. Persegi Panjang 2. Segitiga Siku 3. Lingkaran Masukan Menu : 1 Masukan panjang persegi: 10 Masukan lebar persegi: 20 Luas persegi panjang: 200.0 |

Untuk mencoba memilih menu lain, jalankan lagi programnya dan pilih menghitung luas dan keliling pada bangun datar yang lain. Selain itu Anda juga bisa membuat bangun datar yang lain, misalnya trapesium, layang-layang, belah ketupat, dll. Untuk tambahan menghitung luas dan kelilingnya. Selamat mencoba dan berkreasi.