



Modul Kuliah

# PROGRAMASI BERORIENTASI OBJEK

Abas Setiawan, Suprayogi, Nurul Anisa Sri Winarsih



UNIVERSITAS *for a better future*  
**DIAN NUSWANTORO**

PERGURUAN TINGGI TERAKREDITASI A

## KATA PENGANTAR

Pemrograman dengan paradigma berorientasi objek telah menjadi bagian penting pada pengembangan perangkat lunak. Banyak aplikasi atau *software* yang diprogram dengan paradigma ini. Pada jurusan Teknik Informatika S1 Universitas Dian Nuswantoro terdapat mata kuliah yang membahas tentang paradigma pemrograman berorientasi objek. Untuk menunjang mata kuliah tersebut diperlukan suatu diktat atau modul untuk memudahkan mahasiswa.

Modul Pemrograman Berorientasi Object ini merupakan diktat kuliah bagi mahasiswa yang tidak hanya menyajikan konsep-konsep dasar paradigma memprogram berorientasi objek tetapi juga memberikan solusi praktis aplikatif. Disamping itu, mahasiswa juga diperkenalkan pembuatan *Class Diagram*, *Graphical User Interface*, *Design Pattern*, dan integrasi antara basis data dan aplikasi yang dibuat. Bahasa Pemrograman pada modul ini keseluruhan menggunakan JAVA. Namun demikian, materi-materi yang bersifat konsep utama paradigma pemrograman berorientasi objek dapat diprogram tidak hanya dengan bahasa pemrograman JAVA. Sebagian besar materi diambil dari buku Liang, D.J., 2015, Introduction to JAVA Programming Comprehensive Version Tenth Edition, Pearson.

Penulis juga mengucapkan terima kasih kepada pihak-pihak terkait yang mendukung diadakannya modul perkuliahan ini. Serta penulis berharap modul ini dapat digunakan dengan sebaik-baiknya. Modul ini bukan pengganti perkuliahan melainkan pendukung perkuliahan atau untuk melengkapi kuliah yang diberikan.

Penulis menyadari bahwa masih terdapat banyak kesalahan dari modul ini dan mungkin belum bisa memuaskan banyak pihak. Penulis akan sangat mengapresiasi masukan kritik dan saran tentang modul ini karena hal tersebut akan dapat digunakan sebagai acuan perbaikan dimasa yang akan datang.

Semarang, 1 Januari 2020

Penulis

## DAFTAR ISI

KATA PENGANTAR .....	i
DAFTAR ISI.....	iii
BAB 1 PENGENALAN PEMROGRAMAN BERORIENTASI OBJEK .....	1
1.1 Pendahuluan.....	1
1.2 Berpikir Objek .....	3
1.3 UML Class Diagram .....	3
1.4 Kesimpulan .....	4
1.5 Latihan Soal .....	4
BAB 2 DASAR PEMROGRAMAN DENGAN JAVA .....	6
2.1 Sejarah Bahasa Pemrograman.....	6
2.2 Karakteristik dan Kelebihan Java .....	7
2.3 Instalasi Java dan IDE.....	8
2.4 Struktur Anatomi dan Sintak Program Java.....	12
2.5 Kompilasi dan Menjalankan Program .....	13
2.6 Input Output Console.....	16
2.7 <i>Identifier</i> , Variabel, dan Assignment.....	18
2.8 Tipe data dan Operasi .....	19
2.9 Seleksi Pengambilan Keputusan .....	22
2.10 Perulangan.....	28
2.11 Kesimpulan .....	32
2.12 Latihan Soal .....	32
2.13 Praktikum.....	32

BAB 3 CLASS DAN OBJEK DALAM PBO .....	34
3.1 Mendefinisikan Class dari Objek.....	34
3.2 Property dan Method.....	34
3.2.1 Property .....	34
3.2.2 Method .....	34
3.3 Overloading Method .....	40
3.4 Konstruktor dan Destruktor .....	42
3.5 Static Variable dan Constant.....	43
3.6 Visibility Modifiers.....	47
3.7 Kesimpulan .....	49
3.8 Kuis dan Latihan Soal .....	50
3.9 Praktikum.....	50
BAB 4 ENKAPSULASI .....	53
4.1 Konsep Enkapsulasi .....	53
4.2 Setter-Getter Method .....	54
4.3 Kesimpulan .....	59
4.4 Kuis dan Latihan Soal .....	59
4.5 Praktikum.....	59
BAB 5 INTERAKSI ANTAR OBJEK .....	63
5.1 Keterkaitan antar Class .....	63
5.2 Asosiasi .....	63
5.3 Agregasi dan Komposisi .....	65
5.4 Studi Kasus .....	66
5.5 Kesimpulan .....	75

5.6 Kuis dan Latihan Soal .....	75
5.7 Praktikum.....	76
BAB 6 ARRAY DAN COLLECTION.....	77
6.1 Konsep Array .....	77
6.2 Deklarasi dan Pembuatan Array .....	78
6.3 Mengolah data dengan Array.....	81
6.4 Array Multidimensi.....	85
6.5 Array dari Objek .....	86
6.6 Array Dinamis (List Collection) .....	88
6.7 Kesimpulan .....	92
6.8 Kuis dan Latihan Soal.....	92
6.9 Praktikum.....	92
BAB 7 PEWARISAN .....	93
7.1 Konsep Pewarisan .....	93
7.2 Superclasses dan Subclasses .....	94
7.3 Keyword Super .....	101
7.4 Overriding Method.....	101
7.5 Overriding vs Overloading .....	101
7.6 Kesimpulan .....	102
7.7 Kuis dan Latihan Soal.....	102
7.8 Praktikum.....	103
BAB 8 POLYMORPHISM.....	104
8.1 Konsep Polymorphism.....	104
8.2 Dynamic Binding .....	106

8.3 Kesimpulan .....	107
8.4 Kuis dan Latihan Soal.....	108
8.5 Praktikum.....	108
<b>BAB 9 ABSTRACT CLASS DAN INTERFACE .....</b>	<b>109</b>
9.1 Abstract Class .....	109
9.2 Interface .....	111
9.3 Implementasi Kombinasi Abstract Class dan Interface .....	114
9.4 Kesimpulan .....	116
9.5 Kuis dan Latihan Soal.....	116
9.6 Praktikum.....	117
<b>BAB 10 EXCEPTION, PEMROGRAMAN GENERIK, DAN DESIGN PATTERN.....</b>	<b>118</b>
10.1 Exception .....	118
10.2 Mekanisme Try-Catch .....	119
10.3 Pemrograman Generik .....	121
10.4 Design Pattern.....	123
10.5 Singleton Pattern.....	123
10.6 Model-View-Controller Pattern .....	125
9.5 Kuis dan Latihan Soal.....	127
<b>BAB 11 PEMROGRAMAN JAVA GUI DENGAN JAVA FX .....</b>	<b>128</b>
11.1 Java FX .....	128
11.2 Panes, Groups, UI Controls, dan Shapes di Java FX .....	131
11.3 Property Binding .....	132
11.4 Properties dan Method Node .....	134
11.5 Event-Driven Programming.....	146

11.6 Events dan Event Sources .....	148
11.7 Inner Class .....	149
11.8 Handler berupa Anonymous Inner-Class.....	152
11.9 Lambda Expressions .....	153
11.10 Studi kasus aplikasi GUI menghitung Hutang.....	155
11.11 JavaFX UI Controls .....	159
11.11.1 Label .....	160
11.11.2 Button.....	163
11.11.3 CheckBox.....	165
11.11.4 RadioButton .....	168
11.11.5 TextField.....	170
11.11.6 TextArea .....	172
11.11.7 ComboBox .....	175
11.11.8 ListView.....	177
11.11.9 ScrollBar .....	180
11.11.20 Slider.....	182
BAB 12 KONSEP DATABASE DAN DBMS .....	184
12.1 Model data relasional .....	185
12.2 SQL.....	186
12.3 Mengakses dan memanipulasi database.....	187
BAB 13 APLIKASI CRUD DENGAN JDBC .....	190
13.1 Membuat aplikasi database menggunakan JDBC.....	190
13.2 Koneksi ke database.....	191
13.3 Membuat statement.....	191

13.4 Memproses ResultSet.....	192
13.5 Contoh Aplikasi CRUD .....	192
13.6 SQLLite.....	221
BAB 14 GUI DAN MVC CRUD MENGGUNAKAN NETBEANS .....	233
14.1 Persiapan menggunakan Netbeans.....	233
14.2 Java Form.....	234
14.3 MVC CRUD menggunakan Netbeans .....	237
14.4 Deploy Aplikasi .....	248
14.5 Kesimpulan .....	249
14.6 Kuis dan Latihan Soal.....,	250
14.7 Praktikum.....	250

# BAB 1 PENGENALAN PEMROGRAMAN BERORIENTASI OBJEK

## 1.1 Pendahuluan

Pada dasarnya komputer merupakan alat untuk menyimpan dan memproses data. Komputer terdiri dari perangkat keras (*hardware*) dan perangkat lunak (*software*). Perangkat keras merupakan perangkat yang terlihat secara fisik yang merupakan elemen penting komputer. Perangkat keras tidak dapat bekerja tanpa adanya perangkat lunak. Perangkat lunak menyediakan instruksi-instruksi yang digunakan untuk mengendalikan perangkat keras dan membuatnya melakukan sesuatu tugas (penyelesaian masalah) tertentu. Perangkat lunak dikembangkan dengan cara di program. Jadi bagaimana cara untuk membuat program (melakukan pemrograman)? Kembali pada perangkat lunak, pada dasarnya program juga demikian, berisi instruksi-instruksi yang diberikan kepada komputer atau perangkat komputer tentang tugas apa yang akan/dapat dilakukan.

Meskipun demikian, komputer tidak dapat secara langsung menerima instruksi-instruksi yang diberikan oleh programmer. Programmer merupakan istilah bagi seseorang yang membuat program komputer. Komputer hanya sebuah mesin dan mesin hanya mengetahui angka 0 (*on*) dan 1 (*off*). Seorang programmer tidak dapat membuat instruksi-instruksi pemrograman dengan deretan angka 0 dan 1. Programmer memerlukan alat atau *tool* yang dapat digunakan sebagai “penerjemah” antara bahasa manusia dengan bahasa mesin. Alat tersebut dikenal dengan istilah bahasa pemrograman. Jadi selain membutuhkan instruksi-instruksi yang diberikan pada komputer, programmer juga membutuhkan bahasa pemrograman.

Setiap programmer memiliki pendekatan atau cara yang berbeda untuk menuliskan instruksi-instruksi tersebut. Secara umum, terdapat empat pendekatan pemrograman yaitu, prosedural/imperatif, deklaratif, fungsional, dan berorientasi objek. Pendekatan prosedural merupakan pendekatan pemrograman tradisional yang dikembangkan dengan cara membuat urutan-urutan instruksi diikuti dengan manipulasi data untuk suatu tujuan atau tugas tertentu. Pendekatan deklaratif berfokus pada pendeskripsi masalah yang akan diselesaikan dibandingkan dengan membuat urutan atau langkah-langkah yang akan dilakukan. Pada pendekatan fungsional, program dilihat sebagai entitas yang menerima input dan menghasilkan output. Program dibagi-bagi pada unit kecil yang disebut dengan fungsi dan

fungsi tersebut dapat digunakan pada fungsi yang lainnya. Pendekatan berorientasi objek memungkinkan program sebagai set unit yang disebut dengan istilah objek. Setiap objek memiliki kemampuan untuk melakukan suatu aksi dan saling terkait antar satu objek dengan yang lainnya untuk mencapai tujuan tertentu.

Fokus yang akan dibahas secara mendalam pada Buku ini adalah pendekatan berorientasi objek. Pendekatan ini dapat disebut dengan istilah “Pemrograman Berorientasi Objek”. Pendekatan ini merupakan pendekatan yang paling modern jika dibandingkan dengan pendekatan yang lainnya. Kenapa bisa dikatakan modern? Kenapa programmer harus menggunakan pendekatan ini? Program merupakan representasi dari sistem yang dibangun oleh developer (sekumpulan organisasi atau perusahaan yang berisi programmer, analis sistem, dll) atau programmer. Semakin besar sistem yang dibangun maka program akan menjadi semakin kompleks. Berikut adalah alasan kenapa programmer membutuhkan pemrograman berorientasi objek:

- Memiliki kemampuan pemeliharaan program yang cepat dan efisien (tanpa merombak sistem) dibandingkan dengan paradigma pemrograman yang lain,
- Mempercepat waktu pengembangan,
- Mendukung kerja sama tim saat memprogram,
- Mudah menerjemahkan dari model bisnis dalam model pemrograman,
- Paradigma pemrograman selain berorientasi objek akan bekerja dengan baik pada sistem dengan lingkup yang terisolasi, tetapi tidak jika sistem terintegrasikan,
- Mendukung sistem operasi modern dan dapat dengan mudah beradaptasi dengan program pihak ketiga (*third-party program*),
- Kemampuan untuk dapat membuat *Graphical User Interface* (GUI) yang mudah dimengerti.

Jadi apa definisi dari pemrograman berorientasi objek? Pemrograman berorientasi objek merupakan pendekatan pengembangan perangkat lunak/program yang memiliki struktur berbasis objek yang dapat berinteraksi satu sama lain untuk memenuhi suatu tujuan. Sebagian besar bahasa pemrograman modern saat ini mendukung pendekatan berbasis objek ini. Akan

tetapi, pada buku ini, akan digunakan bahasa pemrograman Java untuk melakukan pemrograman berorientasi objek.

## 1.2 Berpikir Objek

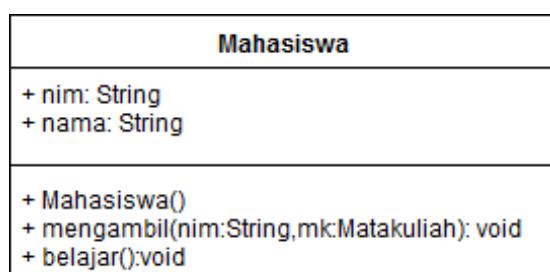
Pada pemrograman prosedural kita dapat mengatakan suatu cara menyelesaikan masalah berdasarkan oleh aksi (*action-driven*) dan data berada terpisah dari aksi-aksi yang ada. Didalam pendekatan berorientasi objek, data dan aksi berada dalam satu tempat yang disebut dengan objek. Misalnya kita dapat membayangkan sebuah sistem besar yang akan dibangun yaitu Sistem Informasi Akademik. Didalam sistem tersebut terdapat entitas yang merepresentasikan bagian kecil dari sistem yang terlibat dalam sistem yang besar itu, katakanlah objek Mahasiswa. Pada objek Mahasiswa terdapat attribut data nama, nomor induk mahasiswa, jenis kelamin, dll. Selain itu, objek Mahasiswa juga memiliki aksi seperti mengambil mata kuliah, melihat data diri, dll.

Ketika melihat aksi yang ada dalam objek Mahasiswa tersebut, dapat dilihat bahwa untuk melakukan aksi mengambil mata kuliah diperlukan setidaknya data nim untuk membedakan antara mahasiswa satu dengan yang lain. Disamping itu, terdapat objek lain yang terkait yaitu Mata Kuliah. Didalam objek Mata Kuliah terdapat attribut data seperti nama mata kuliah, jadwal, dan ruang. Dari contoh yang sudah diceritakan sebelumnya dapat dilihat bahwa sistem terbagi dalam sistem yang lebih kecil dengan direpresentasikan oleh objek-objek yang saling terkait satu sama lain. Dengan demikian, cara berpikir seperti ini dapat digunakan sebagai acuan untuk memulai memprogram dengan pendekatan berorientasi objek.

## 1.3 UML Class Diagram

UML merupakan singkatan dari *Unified Modelling Language* yang merupakan bahasa spesifikasi yang terstukturisasi untuk pemodelan objek. UML ditemukan pada tahun 1990-an oleh James Rumbaugh, Grady Booch, dan Ivar Jacobson. UML juga dapat dikatakan sebagai jembatan dari kode program kedalam model bisnis. Didalam UML terdapat bagian kecil yang dikenal dengan istilah “*Class Diagram*” yang dapat membuat representasi visual dari *class* objek saat membuat program yang berbasis objek. *Class* merupakan bentuk rancangan program dari sebuah objek. Jadi objek merupakan representasi langsung dari *class*. *Class* akan dibahas lebih lanjut di BAB 3. Pada buku ini kita akan fokus kepada bagian *class diagram* untuk merepresentasikan objek-objek yang diprogram.

Satu *class diagram* berbentuk kotak dengan tiga bagian yang ditumpuk seperti pada Gambar 1.1. Bagian tumpukan pertama yang paling atas berisi nama *class*. Bagian tumpukan kedua atau tengah merupakan bagian untuk data atau properti. Bagian tumpukan ketiga atau paling bawah berisi method-method atau fungsi-fungsi. Properti dan method merupakan anggota didalam struktur anatomi *class*. Properti merupakan atribut data yang dimiliki oleh suatu *class* atau dapat dikatakan sebagai variabel. Method merupakan bentuk tingkah laku atau aksi yang dimiliki oleh suatu *class*. Seperti halnya pada pemrograman fungsional, method juga mirip dengan fungsi yang dapat memiliki nilai pengembalian ataupun tidak memiliki nilai pengembalian. Properti dan method akan dijelaskan lebih lanjut pada BAB 3.



Gambar 1.1 Contoh *class diagram* Mahasiswa

## 1.4 Kesimpulan

Pada BAB ini memberikan penjelasan umum tentang paradigma pemrograman berorientasi objek. Dimana ketika akan membangun program, seorang programmer akan berpikir semua entitas yang terkait sebagai suatu objek-objek yang saling terkait. Selain itu, terdapat UML *class diagram* yang dapat menerjemahkan dari bahasa pemrograman menjadi bahasa bisnis. Pada BAB 2, akan dibahas cara memprogram dengan bahasa pemrograman Java dimulai dari pemrogramman dasar. Pemrograman berorientasi objek dengan bahasa pemrograman java akan mulai dibahas dari BAB 3.

## 1.5 Latihan Soal

Jawablah pertanyaan-pertanyaan dibawah ini dengan tepat!

1. Apa perbedaan antara pendekatan pemrograman prosedural dengan pendekatan berorientasi objek?
2. Mengapa programmer membutuhkan pemrograman berorientasi objek?
3. Apa kaitannya antara *class* dengan objek?

4. Siapa yang menemukan UML?
5. Mengapa dalam memprogram berorientasi objek juga dibutuhkan UML *class diagram*?

## BAB 2 DASAR PEMROGRAMAN DENGAN JAVA

### 2.1 Sejarah Bahasa Pemrograman

Komputer tidak dapat mengerti dengan mudah instruksi yang dimasukkan oleh manusia. Komputer butuh alat untuk menerjemahkan bahasa manusia kedalam bahasa “mesin”. Istilah “mesin” digunakan karena pada dasarnya komputer adalah mesin yang dapat menerima instruksi, memproses, dan mengeluarkan keluaran output kedalam perangkat lain yang terhubung (monitor). Alat yang digunakan untuk memudahkan komputer memahami instruksi yang dimasukkan adalah bahasa pemrograman. Ada banyak sekali bahasa pemrograman yang dikembangkan untuk mempermudah manusia dalam memprogram di komputer. Komputer memiliki bahasa dasar (*native*) yang dikenal dengan bahasa mesin. Bahasa mesin terdiri dari rangkaian urutan angka biner. Berikut adalah contoh penulisan bahasa mesin:

```
10111010101110101011110111110111101
```

Dapat dibayangkan ketika programmer harus memprogram dengan bahasa mesin, hal tersebut sangatlah tidak menyenangkan. Program yang ditulis akan sulit untuk dibaca dan dimodifikasi. Dengan alasan tersebut muncul bahasa pemrograman assembly. Assembly memungkinkan programmer untuk memasukkan instruksi dengan kata-kata singkat atau tidak lagi rangkaian angka biner. Kata-kata singkat tadi juga dikenal dengan istilah “*mnemonic*”. Berikut adalah contoh kode yang berisi instruksi sederhana dengan bahasa assembly:

```
add 2, 3, result
```

Menulis dengan bahasa assembly lebih mudah dibandingkan dengan bahasa mesin. Meskipun demikian, bahasa ini merupakan bahasa yang paling dekat dengan bahasa mesin. Untuk memprogram dengan assembly programmer harus sudah memahami cara kerja CPU secara terperinci. Karena kedekatan bahasa assembly dengan mesin sehingga bahasa ini termasuk dalam kategori bahasa tingkat-rendah.

Pada tahun 1950-an, generasi baru pemrograman muncul dengan kategori bahasa tingkat-tinggi. Bahasa tingkat-tinggi mendukung penulisan program dengan bahasa Inggris dan mudah untuk digunakan. Selain itu, bahasa tingkat-tinggi ini dapat dijalankan di beberapa jenis mesin (tidak hanya pada satu mesin tertentu). Instruksi-instruksi perintah yang

dimasukkan dengan bahasa tingkat-tinggi ini dikenal dengan istilah *statement*. Berikut adalah contoh *statement* dengan bahasa tingkat-tinggi untuk menghitung luas lingkaran:

```
area = 7 * 7 * 3.14;
```

Terdapat banyak sekali bahasa pemrograman yang masuk dalam kategori bahasa tingkat-tinggi. Tabel 2.1 menunjukkan beberapa bahasa pemrograman yang populer saat ini yang termasuk dalam kategori bahasa tingkat-tinggi.

Tabel 2.1 Bahasa Tingkat-Tinggi yang Populer

Bahasa	Deskripsi singkat
C	Merupakan bahasa pemrograman yang dikembangkan dari laboratorium Bell. Berisi kombinasi dari bahasa assembly dengan bahasa yang sudah dikembangkan kedalam bahasa tingkat-tinggi yang mudah dipahami.
C++	Bahasa pemrograman yang mendukung pemrograman berorientasi objek yang dikembangkan dari bahasa C.
C#	Bahasa pemrograman yang mendukung pemrograman berorientasi objek yang dikembangkan oleh Microsoft.
Java	Bahasa pemrograman yang mendukung pemrograman berorientasi objek yang dikembangkan oleh Sun Microsystems yang saat ini bagian dari Oracle.
Javascript	Bahasa pemrograman berbasis web yang dikembangkan oleh Netscape.
Python	Bahasa pemrograman yang dapat berguna untuk apa saja ( <i>genera purpose language</i> ). Sangat efektif untuk menulis program-program singkat.
Visual Basic	Bahasa pemrograman yang dikembangkan oleh Microsoft dan sebagian besar penerapannya digunakan dalam produk-produk aplikasi dari Microsoft.

## 2.2 Karakteristik dan Kelebihan Java

Java merupakan bahasa pemrograman yang sangat *powerfull* yang dapat dikembangkan pada berbagai platform seperti, komputer *desktop*, perangkat *mobile*, dan server (*web*). Pada buku ini, akan digunakan bahasa pemrograman java untuk membuat program berorientasi objek. Java dikembangkan oleh tim dari Sun Microsystems yang dipimpin oleh James Gosling. Pada

tahun 2010, perusahaan Sun Microsystem sudah dibeli oleh Oracle. Bahasa java atau dahulu dikenal dengan Oak di desain mulai dari tahun 1991 untuk bahsa pemrograman pada alat-alat elektronik.

Nama java baru muncul pada tahun 1995 dimana fokusnya adalah mengembangkan aplikasi berbasis web. Java muncul dan menjadi populer dikalangan programmer karena kemudahannya dan dapat berjalan dibanyak platform. Bahkan saat ini, java digunakan sebagai dasar program aplikasi dengan platform Android. Platform Android merupakan platform yang paling populer dan besar dikalangan perangkat *mobile*. Java memiliki beberapa kelebihan diantaranya adalah sebagai berikut:

- Bahasa pemrograman yang sederhana,
- Mendukung pemrograman berorientasi objek,
- Mendukung pemrograman yang terdistribusi,
- Mudah untuk ditafsirkan (*interpreted*).
- Bahasa pemrograman yang kuat yang berarti bahwa *compiler* dapat menanggapi kesalahan program dengan baik.
- Bahasa pemrograman yang memiliki mekanisme keamanan kusus.
- Memiliki arsitektur yang netral atau tulis sekali dijalankan dimana saja.
- Portabel atau dapat digunakan untuk mengembangkan aplikasi pada berbagai platform.
- Memiliki performa tinggi.
- Mendungkung pemrograman banyak *thread* atau *multithreaded*.
- Bahasa pemrograman yang dinamis (dapat langsung ter-*compile*).

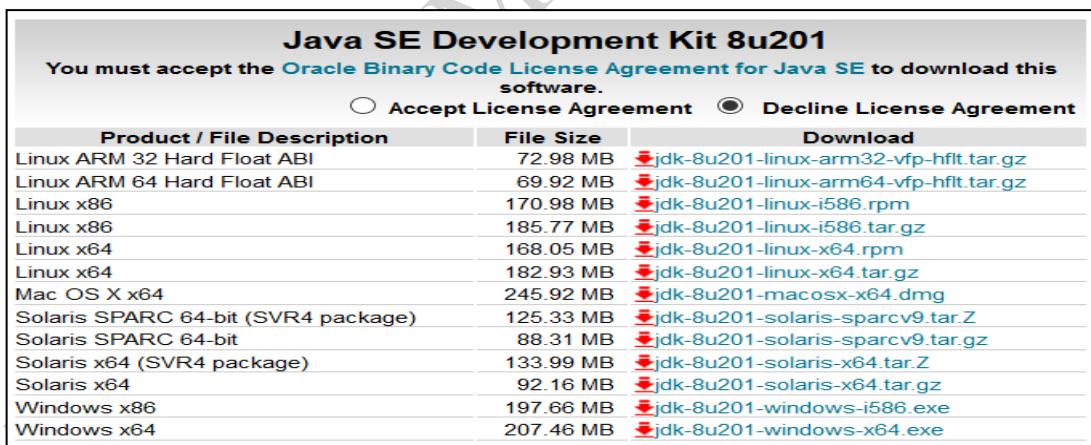
### 2.3 Instalasi Java dan IDE

Bahasa pemrograman memiliki kebijakan masing-masing dalam aturan tata-bahasanya atau *syntax*. Untuk dapat memulai memprogram dengan java dibutuhkan Java *Application Programming Interface* (API) atau dikenal dengan istilah Java *Library*. Didalam Java API berisi beberapa *class* dan *interface* yang dapat digunakan untuk mengembangkan aplikasi

dengan Java. Java API dapat diakses dengan mengunduh aplikasi JDK (*Java Development Toolkit*). Java datang dengan tiga edisi, yaitu:

- Java Stikitard Edition (Java SE) untuk mengembangkan aplikasi disisi klien (*client side*). Aplikasi-aplikasi di edisi ini dapat berjalan pada aplikasi *desktop*.
- Java Enterprise Edition (Java EE) untuk mengembangkan aplikasi disisi server (*server side*).
- Java Micro Edition (Java ME) untuk mengembangkan aplikasi pada perangkat *mobile*.

Buku ini menggunakan edisi Java SE untuk memperkenalkan pemrograman java dengan berorientasi objek. Terdapat beberapa versi Java SE, tetapi pada buku ini kita akan menggunakan Java SE 8 yang dirilis oleh Oracle. Java SE sudah termasuk didalam aplikasi JDK 1.8. JDK 1.8 atau JDK 8 dapat diunduh di <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>. Ketika mengunduh perhatikan bahwa kita harus menyetujui persetujuan lisensi yang harus diterima. Selain itu juga harus diperhatikan sistem operasi yang digunakan. Gambar 2.1 memperlihatkan halaman unduhan ketika akan mengunduh JDK 8u201.

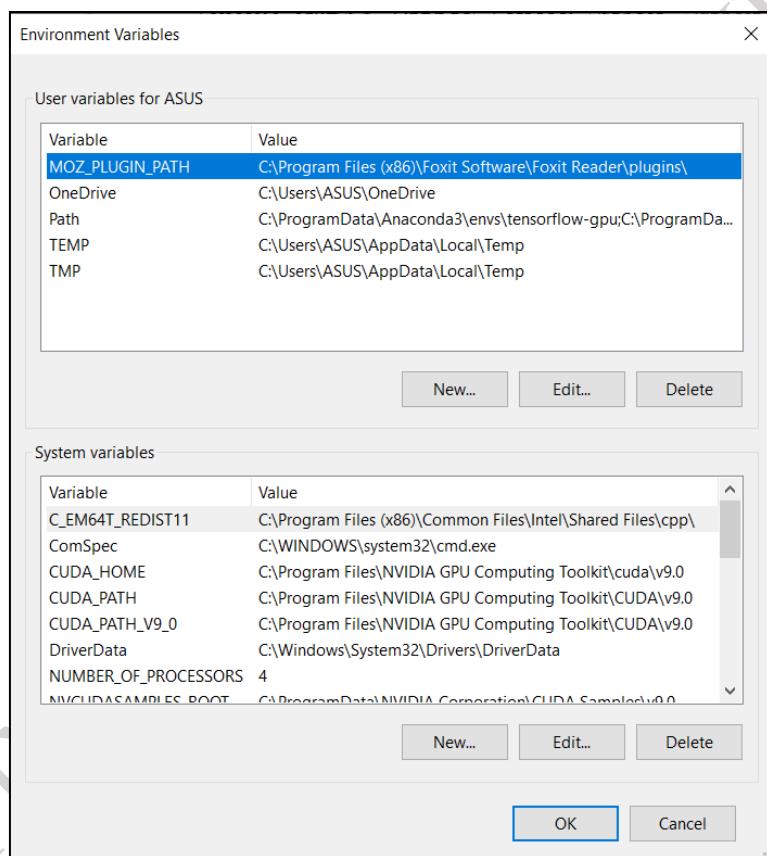


Gambar 2.1 Halaman unduhan JDK 8u201

JDK berisi beberapa set program yang terpisah. Setiap set program tersebut memiliki fungsi masing-masing seperti memanggil beberapa isntruksi yang sudah ditentukan melalui *command line*, kompilasi, menjalankan program, dan pengetesan program Java. Program untuk menjalankan aplikasi Java dikenal dengan nama JRE (*Java Runtime Environment*). Intinya JDK digunakan untuk kompilasi dan menjalankan program Java. Meskipun demikian,

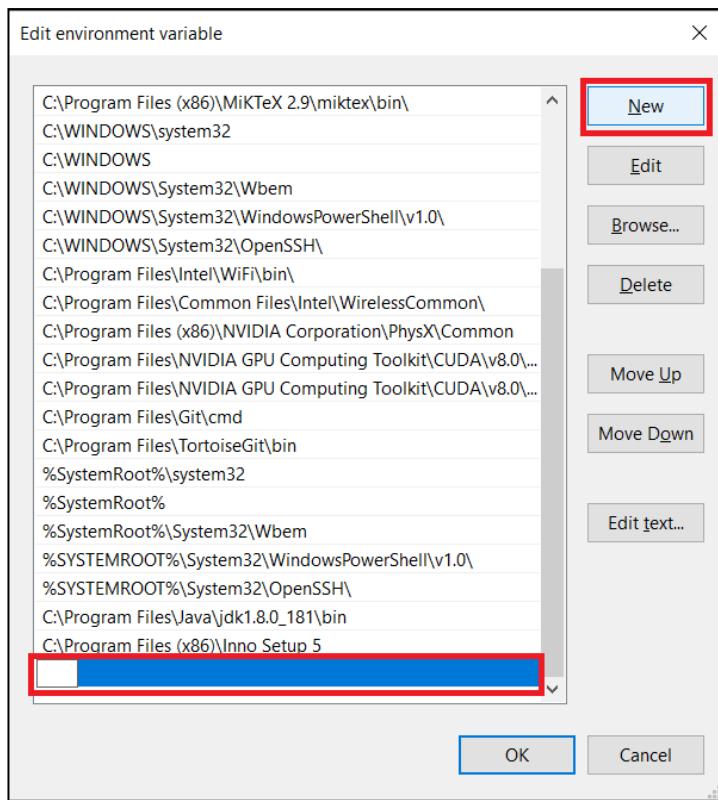
kita dapat menggunakan aplikasi selain JDK seperti, Netbeans dan Eclipse. Untuk dapat menjalankan JDK tentu saja JDK harus terinstal terlebih dahulu. Setelah itu kita harus mendaftarkan perintah untuk kompilasi dan menjalankan program pada sistem operasi. Berikut adalah cara untuk mendaftarkan perintah tersebut pada sistem operasi Windows 10.

1. Klik logo Windows 10 cari “*Edit the system environment variable*” atau cari “Control Panel” → System and Security → System → Advanced System Settings (di tab Advanced klik tombol “*Environment Variables..*”). Gambar 2.2 memperlihatkan halaman *environment variables*.



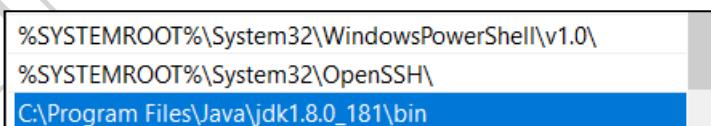
Gambar 2.2 Halaman *environment variables*

2. Fokus pada bagian *System variables*, tarik kebawah dan cari *variable* dengan nama “path” kemudian klik tombol Edit (yang bawah/bagian *system variable*). Perhatikan bahwa **JANGAN menghapus variable atau environment variables didalam path karena itu dapat menyebabkan kerusakan pada sistem operasi!** Gambar 2.3 memperlihatkan halaman *edit environment variable* didalam path.



Gambar 2.3 Halaman edit environment variable pada path

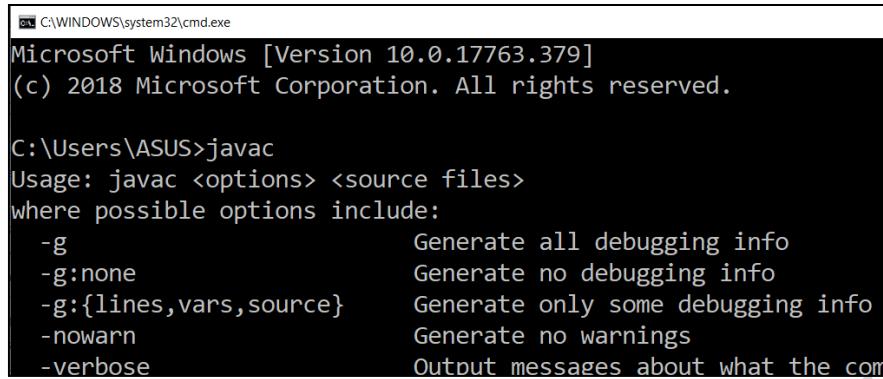
Setelah itu, seperti pada Gambar 2.3 klik pada tombol New maka akan ada tambahan isian kosong dibagian bawah. Klik tombol Browse dan cari alamat direktori JDK yang sudah di instal sebelumnya dan masuk pada folder bin. Contohnya: C:\Program Files\Java\jdk1.8.0\_181\bin. Gambar 2.4 menunjukkan penambahan direktori JDK pada *environment variable* yang sudah benar.



Gambar 2.4 Penambahan direktori JDK pada *environment variable*

Jika sudah seperti pada Gambar 2.4 langkah berikutnya adalah klik tombol “OK” dan keluar dari halaman *environment variable*.

3. Buka *command prompt* pada windows atau ketik cmd pada pencarian di Windows 10. Ketikkan “javac”, jika muncul keluaran pada *command prompt* seperti Gambar 2.5, berarti instalasi berhasil.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17763.379]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ASUS>javac
Usage: javac <options> <source files>
where possible options include:
    -g                         Generate all debugging info
    -g:none                     Generate no debugging info
    -g:{lines,vars,source}       Generate only some debugging info
    -nowarn                     Generate no warnings
    -verbose                    Output messages about what the com
```

Gambar 2.5 Instalasi JDK yang berhasil

Untuk software aplikasi Netbeans atau Eclipse menyediakan IDE (*Integrated Development Environment*) untuk mengembangkan aplikasi berbasis Java dengan cepat. Misalnya pada Netbeans, menyediakan *Graphical User Interface* (GUI) yang dapat membantu programmer (*drag-and-drop*) tanpa harus memprogram semua komponen GUI yang ada. Netbeans dapat di unduh di halaman <https://netbeans.org/downloads/8.2/> dan dapat di instalasi dengan mudah dengan mengikuti petunjuk (*next-next* saja). Pada buku ini hanya digunakan Netbeans dengan Java SE.

## 2.4 Struktur Anatomi dan Sintak Program Java

Pemrograman di Java setidaknya membutuhkan satu *class* dan satu *main method*. Programmer diperbolehkan untuk membuat *class* baru dengan nama yang diinginkan oleh programmer itu sendiri. Namun demikian terdapat aturan sintak seperti berikut:

```
public class <Nama Class> { }
```

Nama *class* disini hanya boleh dimulai oleh huruf (biasanya programmer menggunakan stikir dengan dimulai dari huruf besar) dan tidak boleh ada spasi. Boleh ada angka dan karakter khusus seperti “\_” tetapi tetap diawali dengan huruf. Berikut adalah contoh penamaan *class* yang benar:

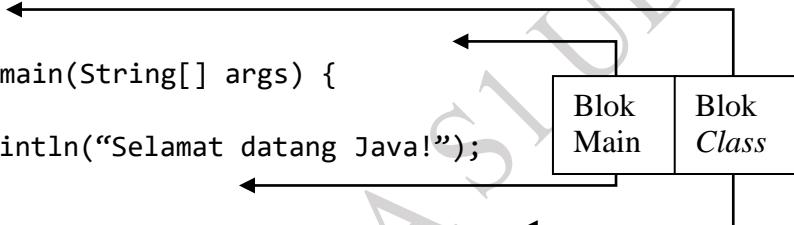
```
public class Hello { }
```

Selain itu, diperlukan *main method* sebagai acuan pertama kali program akan mengeksekusi instruksi yang kita berikan atau *statement*. Pembuatan *main method* yang benar adalah sebagai berikut:

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

Perhatikan bahwa kurung kurawal buka “{“ dan kurung kurawal tutup “}” merupakan bentuk blok dari beberapa komponen program. Komponen program dapat berisi satu atau banyak *statement*. Misalnya pada sebuah class memiliki blok komponen program yaitu data dan method. Demikian halnya dengan method yang memiliki blok komponen program berisi beberapa *statement*. Blok juga dapat bersarang atau *nested*, artinya didalam satu blok bisa berisi blok-blok lain. Begitu juga dengan blok-blok lain tersebut bisa berisi beberapa blok lainnya. Berikut adalah contoh program utuh yang berisi satu class bernama Hello dan satu main method:

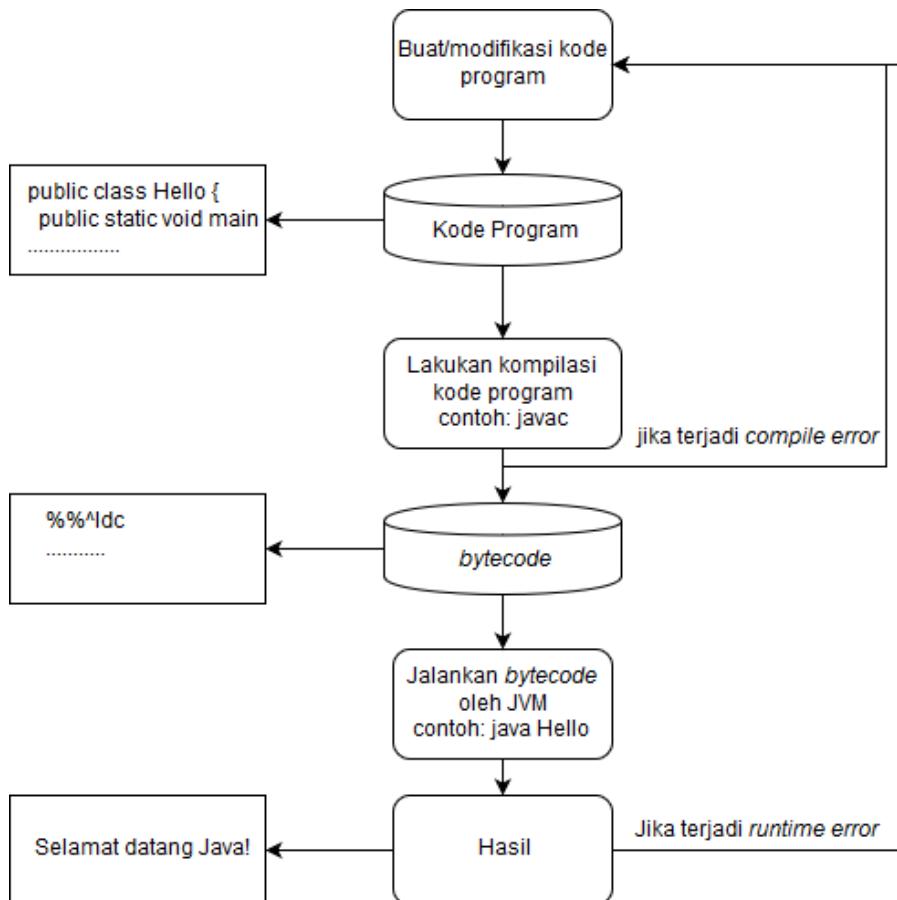
```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Selamat datang Java!");  
    }  
}
```



Pada saat memprogram Java, *statement* diakhiri dengan karakter “;”. Semua yang ditulis di Java mengikuti aturan penulisan yang sensitif atau dikenal dengan istilah *case-sensitive*. Misalnya ketika menuliskan “main” diganti dengan “Main”, hal ini akan menyebabkan kesalahan sintak. Kesalahan sintaksis akan muncul ketika programmer tidak mematuhi aturan sintak.

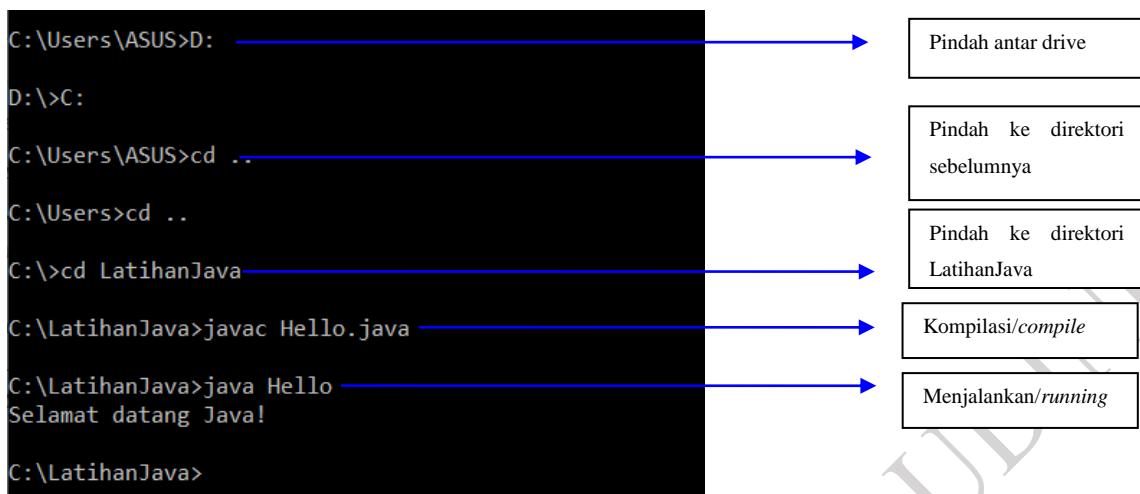
## 2.5 Kompilasi dan Menjalankan Program

Ketika akan membuat program java pertama kali pastikan bahwa program java disimpan dengan ekstensi file “.java”. Setelah menyimpan file dengan ekstensi tersebut, dilanjutkan dengan proses kompilasi yang akan menghasilkan file dengan ekstensi file “.class”. Dari file hasil kompilasi tersebut akan dijalankan oleh *Java Virtual Machine* (JVM). Proses kompilasi dan menjalankan tersebut akan diulang terus menerus selama memprogram dengan Java. Apa yang terjadi sebetulnya dan apa itu JVM? Jadi file .class pada dasarnya merupakan bahasa yang mirip (tidak sama persis) dengan bahasa mesin atau dapat disebut dengan *bytecode* yang dapat di jalankan pada platform apapun selama memiliki JVM. Java memiliki JVM yang dapat menerjemahkan *bytecode* pada apapun platform dan/atau sistem operasi dari pada menggunakan mesin fisik. Gambar 2.6 merupakan proses pengembangan program Java dengan studi kasus class Hello pada sub bab sebelumnya.



Gambar 2.6 Siklus kompilasi dan menjalankan program Java.

Ketika programmer membuat program java untuk pertama kali menggunakan editor (notepad misalnya), programmer akan menyimpan pada direktori tertentu didalam harddisk. Misalnya programmer menyiapkan dengan nama “LatihanJava” di drive C dan kemudian akan menyimpan file Hello.java didalam folder tersebut. Maka mekanisme untuk melakukan kompilasi dan menjalankan aplikasi dilakukan melalui *command prompt* windows seperti pada Gambar 2.7 (perhatikan bahwa jdk sudah di instal seperti pada BAB 2.3). Program java dapat ditulis menggunakan editor teks sederhana, seperti notepad misalnya. Jika notepad terasa monoton dan kurang bawarna programmer bisa mengunduh dan menginstal notepad++ (<https://notepad-plus-plus.org/download/v7.6.4.html>) atau visual studio code (<https://code.visualstudio.com/download>).



Gambar 2.7 Proses kompilasi dan menjalankan program Java.

Dalam pemrograman dengan Java, terdapat tiga jenis kesalahan program atau *error* yang umum, yaitu *syntax error*, *runtime error*, dan *logic error*. *Syntax error* atau *compile error* merupakan kesalahan yang sering sekali dilakukan oleh programmer pemula. Kesalahan ini dikarenakan penulisan program yang tidak tepat atau tidak sesuai dengan sintak pemrograman yang seharusnya. Sebagai contoh, programmer lupa menambahkan titik-koma “;” di akhir *statement*. Kesalahan tersebut akan muncul ketika kita meng-kompile (saat menjalankan javac). Bentuk kesalahannya adalah sebagai berikut:

```
Hello.java:3:error: ';' expected
```

Kesalahan tersebut diatas terjadi pada Hello.java baris ke 3 dengan keterangan kesalahan “dibutuhkan titik-koma”. *Syntax error* merupakan bentuk kesalahan program yang sangat mudah untuk dideteksi.

Kesalahan program yang kedua adalah *runtime error*. *Runtime error* akan mengakibatkan program berhenti secara tidak normal. Artinya program berhasil di kompile tetapi saat dijalankan program akan menjadi berhenti secara tiba-tiba. Contoh dari error ini adalah pembagian suatu nilai dengan nilai 0. Maka akan muncul kesalahan yang dikenal dengan “*Exception*” saat ketika menjalankan program. Kesalahan program seperti ini muncul karena saat program dijalankan, lingkungan pemrograman Java (sistem Java) tidak mungkin dapat menangani masalah tersebut.

Kesalahan jenis ketiga adalah *logic error*. *Logic error* atau dapat disebut dengan istilah “*bug*” merupakan jenis kesalahan yang cukup sulit dideteksi. Program berhasil di kompile dan di

jalankan, tetapi terdapat kesalahan yang dilakukan program saat program tersebut berjalan (program tidak berjalan semestinya). Contohnya, ketika kita membuat program untuk mendeteksi suhu mendidih dan normal pada air. User mencoba untuk memasukkan angka 110 untuk melihat suhunya normal atau mendidih. Suhu mendidih seharusnya diatas 100 derajat celcius. Tetapi output yang dihasilkan program adalah suhu normal. Untuk menangani masalah ini, programmer harus melakukan proses *debugging*. *Debugging* merupakan proses untuk mencari *bug* program dengan mencari pada setiap baris program yang menyebabkan *bug* tersebut muncul.

## 2.6 Input Output Console

Console merupakan istilah jaman dulu yang memiliki arti memasukkan teks dan menampilkan dilayar. Console input artinya kode program yang digunakan untuk memasukkan apa yang diketikkan oleh keyboard. Console output artinya kode program yang digunakan untuk menampilkan yang diketikkan tadi di layar monitor. Seperti program Hello.java yang sebelumnya sudah dijelaskan, disana terdapat kode statement untuk console output yaitu `System.out.println`. Secara umum Java memiliki dua console output yaitu `System.out.println` dan `System.out.print`. Keduanya memberikan output teks atau dikenal dengan tipe data String. Perbedaanya adalah output teks yang dihasilkan oleh `println` dimulai dari baris yang baru sedangkan pada `print` tidak demikian. Kode 2.1 merupakan contoh kode program yang menerapkan console output `println` dan `print`.

### Kode 2.1 Hello.java

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.print("Nama saya Budi!");  
4         System.out.println("Selamat datang Java!");  
5     }  
6 }
```

Console input pada Java sedikit memerlukan mekanisme khusus. Java memiliki `System.out` untuk mendefinisikan output dan `System.in` untuk mendefinisikan input. Untuk melakukan console input tidak bisa langsung memanggil `System.in`, melainkan

dengan menggunakan Scanner *class*. Scanner digunakan untuk membuat objek dari pembacaan input oleh System.in seperti kode berikut:

```
Scanner input = new Scanner(System.in);
```

Scanner merupakan *class* yang didefinisikan oleh java.util.Scanner sehingga dapat dipanggil dengan menambahkan import java.util.Scanner; terlebih dahulu. Variabel *input* merupakan Objek dari scanner yang terbentuk dari System.in. Untuk membuat *input* menjadi suatu tipe data yang kita ingginkan kita harus lakukan pemanggilan method khusus. Misalnya, ketika kita ingin tipe data *input* menjadi double maka kita panggil method nextDouble() seperti pada kode berikut:

```
double beratbadan = input.nextDouble();
```

Variabel beratbadan dapat menerima apapun masukan user dengan tipe angka pecahan atau tipe data double. Kode 2.2 menjelaskan tentang penggunaan console input dan output untuk perhitungan luas persegi.

#### Kode 2.2 HitungLuasPersegi.java

```
1 import java.util.Scanner;
2 public class c {
3     public static void main(String[] args) {
4         // Buat objek Scanner
5         Scanner input = new Scanner(System.in);
6         // Kabari user untuk memasukkan angka sisi persegi
7         System.out.print ("Masukkan sisi persegi: ");
8         double sisi = input.nextDouble();
9         // hitung luas
10        double luas = sisi * sisi;
11        // tampilkan hasil
12        System.out.print ("Luas persegi: "+luas);
13    }
14 }
```

Simpan kode 2.1 dengan nama HitungLuasPersegi.java dan lakukan kompilasi. Setelah itu jalankan program dan coba ketikkan 12 lalu tekan enter. Hasil yang akan ditampilkan program yaitu Luas persegi: 144.

## 2.7 Identifier, Variabel, dan Assignment

*Identifier* merupakan nama-nama yang dibuat sendiri untuk mengidentifikasi variabel, method, atau *class*. Perhatikan bahwa penamaan identifier di Java adalah *case-sensitive*. Semua *identifier* di Java harus mengikuti kaidah-kaidah aturan sebagai berikut:

- Terdiri dari urutan karakter yang terdiri dari huruf, angka, garis bawah “\_”, dan penkita dollar “\$”.
- *Identifier* harus dimulai dari huruf, garis bawah, atau penkita dollar. Tidak boleh dimulai dengan angka.
- *Identifier* tidak boleh memuat kata-kata yang sudah disiapkan oleh Java. Misalnya menamai variabel dengan *class*.
- Bisa terdiri dari satu atau lebih karakter.

Variabel pada Java digunakan untuk merepresentasikan suatu nilai yang mungkin akan berubah-ubah didalam program. Nilai tersebut mengikuti suatu tipe data tertentu atau objek tertentu. Perlu mekanisme pengenalan variabel yang disebut dengan deklarasi variabel. Deklarasi variabel meminta kompiler untuk mengalokasikan ruang memori yang benar berdasarkan kepada tipe data/objek dari variabel tersebut. Deklarasi variabel dapat dilakukan dengan cara berikut:

```
<tipe data> <nama variabel>

int x;

double rata2;
```

Kita dapat memberikan nilai awal dari sebuah variabel dengan cara inisialisasi variabel. Berikut adalah cara inisialisasi variabel:

```
int y = 10;

double panjang;

panjang = 9.5;
```

Setelah deklarasi variabel, kita dapat membuat *assignment statement*. *Assignment* dilakukan varibel dengan cara memerintah (*assign*) suatu nilai yang ditujukan ke variabel tersebut. *Assignment statement* dapat digunakan sebagai ekspresi. Ekspresi mewakili komputasi perhitungan yang melibatkan nilai, variabel, dan operator untuk mengevaluasi suatu nilai tertentu. Jadi aturan sintak untuk *assigment* adalah sebagai berikut:

```
variabel = ekspresi
```

*Assignment* diwakili oleh simbol sama dengan “=”. Berikut adalah contoh assignment kepada nilai langsung, varibel lain, dan ekspresi aritmatika:

```
double x = 10;  
double sisi = x;  
double luas = sisi * sisi;
```

Perhatikan bahwa yang di-*assign* berada pada sisi sebelah kanan dan tidak boleh terbalik. Selain itu kita juga bisa melakukan *assignment* pada banyak variabel sekaligus dalam satu *statement*. Berikut adalah contoh dari *assignment* pada banyak variabel:

```
i = j = k = 100;
```

sama dengan,

```
i = 100;  
j = 100;  
k = 100;
```

Programmer juga dapat melakukan *assignment* pada *statement* atau *method*. Berikut adalah contoh penerapannya pada console output:

```
System.out.print(a = 10);
```

Sama dengan,

```
a = 10;  
System.out.print(a);
```

## 2.8 Tipe data dan Operasi

Pada deklarasi variabel, variabel harus memiliki tipe data sebagai bentuk representasi dari nilai variabel tersebut. Java memiliki beberapa tipe data numerik, karakter, dan boolean. Selain itu, setiap tipe data memiliki jangkauan nilai yang berbeda-beda. Compiler akan

mengalokasikan memori dari variabel sesuai dengan tipe data-nya. Tabel 2.2 memperlihatkan tipe data numerik yang didukung lengkap dengan jangkauan nilai dan ukuran penyimpanannya.

Tabel 2.2 Tipe data numerik di Java

<b>Nama</b>	<b>Jangkauan nilai</b>	<b>Ukuran penyimpanan</b>
int	- $2^{31}$ s/d $2^{31}-1$	32-bit signed (4 byte)
short	- $2^{15}$ s/d $2^{15}-1$	16-bit signed (2 byte)
long	- $2^{63}$ s/d $2^{63}-1$	64-bit signed (8 byte)
float	Negatif : -3.4028235E + 38 s/d -1.4E -45  Positif: 1.4E -45 s/d 3.4028235E+38	32-bit IEEE 754 (4 byte)
double	Negatif : -1.7976931348623157E+308 s/d -4.9E -324  Positif: 4.9E -324 s/d 1.7976931348623157E+308	64-bit IEEE 754 (8 byte)
byte	- $2^7$ s/d $2^7-1$	8-bit signed

Tipe data yang lain (selain numerik) yaitu, karakter diwakili dengan **char**, urutan karakter diwakili dengan **String**, dan logika boolean dengan **boolean**. Tipe data **char** memiliki ukuran 1 byte. Tipe data **String** memiliki ukuran banyaknya karakter dikalikan dengan 1 byte. Tipe data **boolean** memiliki ukuran 1 byte. Kembali ke tipe data numerik, untuk membaca dari keyboard (atau melakukan *console input*), programmer akan menggunakan Scanner. Pada Kode 2.1, kita telah membuat scanner dan menerapkan method **nextDouble()** untuk mendapatkan input dengan tipe data **double**. Untuk input yang lain disajikan pada Tabel 2.2.

Tabel 2.3 Method untuk objek scanner

<b>Method</b>	<b>Deskripsi</b>
nextInt()	Membaca bilangan bulat dari tipe data int
nextShort()	Membaca bilangan bulat dari tipe data short
nextLong()	Membaca bilangan bulat dari tipe data long
nextFloat()	Membaca bilangan riil dari tipe data float
nextDouble ()	Membaca bilangan riil dari tipe data double
nextByte()	Membaca bilangan bulat dari tipe data byte

Tipe data numerik memiliki operasi dasar aritmatika, sehingga membutuhkan operator dasar seperti: penjumlahan “+”, pengurangan “-”, pembagian “/”, perkalian “\*”, dan sisa bagi “%”. Istilah operan merupakan nilai yang dioperasikan oleh operator. Ketika kedua operan bertipe bilangan bulat (integer) dan kemudian akan dilakukan operasi pembagian, jika hasilnya bilangan riil maka angka dibelakang koma akan dihilangkan. Misalnya 5 / 2 akan menghasilkan nilai 2 bukan 2,5. Agar hasilnya menjadi 2,5 kedua operan harus diganti dengan tipe bilangan riil (misalnya float). Kode 2.2 merupakan contoh penggunaan operasi pada variabel yang telah ditentukan.

#### Kode 2.3 CetakWaktu.java

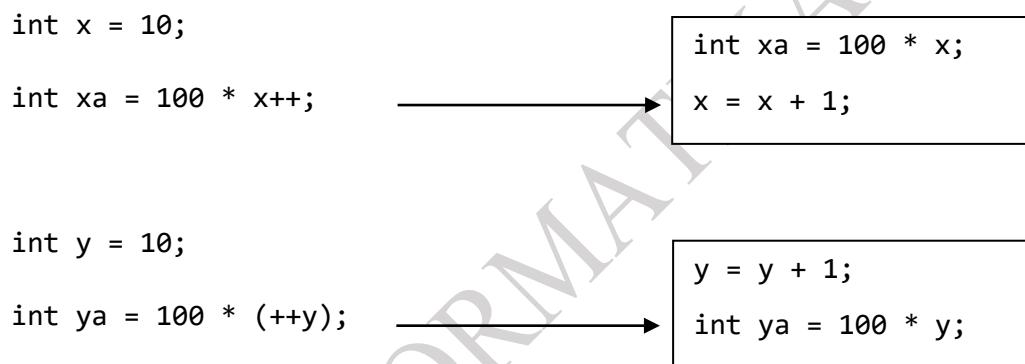
```

1 import java.util.Scanner;
2 public class CetakWaktu {
3     public static void main(String[] args) {
4         Scanner input = new Scanner(System.in);
5         System.out.print ("Masukkan waktu dalam detik: ");
6         int detik = input.nextInt();
7         int menit = detik / 60;
8         int sisadetik = detik % 60;
9         System.out.print (detik+ " detik adalah "+menit+
" menit "+sisadetik+" detik.");

```

10	}
11	}

Terdapat operator khusus yang digunakan untuk menaikkan 1 nilai variabel atau *increment* dan menurunkan 1 nilai variabel atau *decrement*. Misalnya terdapat variabel `int x = 2;` dan kemudian terdapat operasi increment `x++;`. Operasi *increment* tersebut sama dengan `x = x + 1;`. Begitu juga dengan operator *decrement*, misalnya: `x--;` sama dengan `x = x - 1;`. Meskipun demikian kita dapat menuliskan `++` atau `--` sebelum variabelnya, contoh `++x;` atau `--x;`. Dengan demikian *increment* atau *decrement* yang dilakukan setelah variabel disebut dengan *postincrement* atau *postdecrement*. Kemudian untuk *increment* atau *decrement* yang dilakukan sebelum variabel disebut dengan *preincrement* atau *predecrement*. Berikut adalah perbedaan keduanya.



Jika nilai `xa` dan `ya` dioutputkan, mereka memiliki hasil yang berbeda, yaitu `xa = 1000` dan `ya = 1100`.

Programmer dapat melakukan konversi bilangan bulat (integer) ke bilangan riil (float atau double) atau sebaliknya dengan cara *explicit casting*. Misalnya, terdapat variabel `double x = 90.5;` kita dapat melakukan *assignment* ke variabel yang bertipe integer dengan cara `int y = (int) x;` sehingga `y` akan memiliki nilai 90 bukan 90,5.

## 2.9 Seleksi Pengambilan Keputusan

Program dapat memilih *statement* yang mana yang akan di eksekusi berdasarkan suatu kondisi. Misalnya, programmer membuat program untuk menentukan air mendidih atau air yang tidak mendidih. Terdapat variabel suhu air yang dimasukkan oleh user. Untuk dapat

mengoutputkan air mendidih atau air tidak mendidih berdasarkan nilai suhu yang dimasukkan oleh user, program harus memiliki mekanisme untuk memilih atau mengambil keputusan. Lalu bagaimana mekanismenya? Java menyediakan *selection statement* yang dapat membiarkan kita untuk membuat pilihan aksi dengan berbagai alternatif jawaban. Berikut adalah contoh kode *selection statement*:

```
double suhuair = 205.1;  
if(suhuair > 100)  
    System.out.println("mendidih");  
else  
    System.out.println("tidak mendidih");
```

Perhatikan pada kode sebelumnya boleh untuk tidak menggunakan blok kurung-kurawal jika hanya ada satu *statement* didalamnya. *Selection statement* menggunakan kondisi dari ekspresi boolean (*true* atau *false*). Ekspresi boolean biasanya berkaitan dengan operator relasional. Tabel 2.3 merupakan operator relasional dengan disertai simbol.

Tabel 2.4 Operator relasional

Simbol Java	Simbol Matematika	Keterangan
<code>==</code>	<code>=</code>	Sama dengan
<code>!=</code>	<code>≠</code>	Tidak sama dengan
<code>&gt;</code>	<code>&gt;</code>	Lebih dari
<code>&gt;=</code>	<code>≥</code>	Lebih dari sama dengan
<code>&lt;</code>	<code>&lt;</code>	Kurang dari
<code>&lt;=</code>	<code>≤</code>	Kurang dari sama dengan

Setiap penerapan dari ekspresi boolean akan menghasilkan nilai *true* atau *false*. Misalnya, pada kode sebelumnya `if (suhuair > 100)` jika menggunakan output console, akan mengembalikan nilai *true*.

```
double suhuair = 205.1;  
System.out.println(suhuair > 100); //true  
System.out.println(suhuair <= 100); //false
```

Secara umum kerangka untuk pengambilan keputusan dengan *statement* IF terdiri dari beberapa jenis. Jenis pertama adalah *one-way* IF, *two-way* IF, *multi-way* IF, dan *nested* IF. Pada jenis *one-way* IF, *statement* didalam IF akan dieksekusi jika dan hanya jika kodisinya bernilai *true*. Berikut adalah kerangka sintak dari *one-way* IF:

```
if (ekspresi boolean)
{
    statement;
}
```

Pada jenis *two-way* IF, *statement* didalam IF akan dieksekusi jika kodisinya bernilai *true* jika tidak maka akan mengeksekusi *statement* didalam ELSE. Berikut adalah kerangka sintak dari *two-way* IF:

```
if (ekspresi boolean)
{
    Statement untuk kasus true;
}
else
{
    Statement untuk kasus false;
}
```

Jenis lebih lanjut untuk IF adalah *multi-way* IF. *Multi-way* IF memiliki banyak alternatif pilihan kondisi (lebih dari satu) dengan masing-masing *statement* didalamnya. Berikut adalah kerangka sintak dari *multi-way* IF:

```
if (ekspresi boolean kasus 1)
{
    Statement untuk kasus 1 bernilai true;
}
else if (ekspresi boolean kasus 2)
{
    Statement untuk kasus 2 bernilai true;
}
else
```

```

{
    Statement untuk kasus selain yang disebutkan sebelumnya;
}

```

Jenis IF yang lain adalah *nested* IF. *Nested* IF memungkinkan programmer untuk dapat membuat statement IF yang lain berada didalam statement IF (bersarang). Dengan demikian, IF yang ada didalam statement IF disebut dengan inner IF dan IF yang diluar adalah outer IF. Tidak ada batasan inner IF pada praktiknya di Java. Berikut adalah kerangka sintak dari *nested* IF:

```

if (ekspresi boolean kasus 1)
{
    if (ekspresi boolean kasus 2)
        Statement untuk kasus 1 dan 2 bernilai true;
}
else
{
    Statement untuk kasus selain yang disebutkan sebelumnya;
}

```

Kode 2.3 merupakan contoh penerapan semua jenis *statement* IF. Perhatikan bahwa pada Kode 2.3, user diminta untuk memasukkan nilai. Setiap nilai yang dimasukkan oleh user akan ditanggapi oleh program dengan membuat output A atau B atau C atau D. Saat user memasukkan nilai 100 maka statement IF pertama bernilai true dan kemudian dilanjutkan pada inner IF. Pada inner IF juga bernilai true sehingga program akan menghasilkan output A. Jika user memberikan input 99, output yang dihasilkan adalah B karena pada inner IF bernilai false sehingga yang dieksekusi adalah bagian ELSE didalam outer IF. Bagaimana jika user memberikan input 10 dan 50? Apa hasil outputnya?

#### Kode 2.4 TestIF.java

1	import java.util.Scanner;
2	public class TestIF{
3	public static void main(String[] apa)
4	{
5	Scanner input = new Scanner(System.in);

```

6      System.out.print("Masukkan nilai: ");
7      int angka = input.nextInt();
8
9      if(angka > 60)
10     {
11         if(angka%2 == 0)
12         {
13             System.out.println("A");
14         }
15         else
16         {
17             System.out.println("B");
18         }
19     else if(angka > 30)
20     {
21         System.out.println("C");
22     }
23     else
24     {
25         System.out.println("D");
26     }
27 }
28 }
```

Terkadang programmer menginginkan suatu eksekusi program yang dihasilkan dari kombinasi lebih dari satu kondisi. Misalnya, saat programmer akan membuat konversi nilai huruf dari nilai angka yang didapatkan oleh mahasiswa. Mahasiswa akan mendapatkan nilai B jika memiliki nilai pada kisaran angka 70 sampai 84. Bagaimana caranya memasukkan kondisi nilai angka minimal 70 sampai dengan 84 pada program? Kita dapat melakukan konversi kondisi tersebut dengan operasi logika. Misalnya, nilai angka disimbolkan x sehingga terdapat dua kondisi  $x \geq 70$  dan  $x < 85$ . Kata “dan” disini disebut dengan operasi

logika. Pada bahasa pemrograman Java, terdapat beberapa operasi logika yang disajikan pada Tabel 2.4.

Tabel 2.5 Operator logika

Simbol Java	Nama	Deskripsi
&&	Dan	Logika Konjungsi
	Atau	Logika Disjungsi
!	Tidak	Logika Negasi
^	Eksklusif Atau (XOR)	Logika Eksklusif

Contoh penerapan operator logika di Java disajikan pada Kode 2.3. Kode 2.3 merupakan penyelesaian masalah untuk konversi nilai angka kedalam nilai huruf dengan aturan: A didapatkan dari nilai angka lebih dari 85, B didapatkan dari nilai angka 70-84, C didapatkan dari nilai angka 60-69, D didapatkan dari nilai angka 50-59, dan E didapatkan dari nilai angka kurang dari 50.

**Kode 2.5 TestNilaiHuruf.java**

```
1 import java.util.Scanner;
2 public class TestNilaiHuruf{
3     public static void main(String[] apa)
4     {
5         Scanner input = new Scanner(System.in);
6         System.out.print("Masukkan nilai angka: ");
7         int angka = input.nextInt();
8         if(angka > 85)
9         {
10             System.out.println("A");
11         }
12         else if(angka >= 70 && angka <= 84)
13         {
```

```

13         System.out.println("B");
14     }
15     else if(angka >= 60 && angka <= 69)
16     {
17         System.out.println("C");
18     }
19     else if(angka >= 50 && angka <= 59)
20     {
21         System.out.println("D");
22     }
23     else
24     {
25         System.out.println("E");
26     }
27 }
28 }
```

## 2.10 Perulangan

Perulangan merupakan cara menuliskan *statement* secara berulang-ulang. Hal ini sangat memudahkan programmer, misalnya ketika ingin memberikan output “aku bisa” selama 1000 kali. Apakah programmer akan menuliskan console output selama 1000 kali?

```

System.out.println("aku bisa");
System.out.println("aku bisa");
System.out.println("aku bisa");
.....
System.out.println("aku bisa");
```

Tentu hal tersebut akan membutuhkan tenaga dan waktu yang besar. Java menyediakan mekanisme untuk menulis *statement* sekali (atau dalam kasus lain bisa beberapa kali tetapi tidak banyak) dalam satu metode perulangan. Secara sederhana bentuk dari *statement* perulangan tersebut adalah sebagai berikut:

```

int count = 0;
while(count < 1000)
```

```
{  
    System.out.println("aku bisa");  
    count++;  
}
```

Variabel `count` di berikan nilai awal (inisialisasi) dengan nilai 0. Kemudian kode `while` akan melakukan pengecekan apakah `count < 1000` bernilai `true`. Jika benar bernilai `true`, maka program akan mengeksekusi *statement console output* “aku bisa” dan akan melakukan pembaharuan nilai `count` dengan ditambah satu (*increment*). Sehingga, *console output* “aku bisa” akan terus dieksekusi berulang-ulang sampai nilai `count` menjadi 1001 dan kondisi pengecekan didalam `while` menjadi `false`.

Di java bentuk perulangan tidak hanya dapat dilakukan dengan blok *statement while*, tetapi ada `do-while`, dan `for`. Seperti contoh sebelumnya, `while` mengeksekusi *statement* secara berulang-ulang selama kondisi didalam `while` terpenuhi atau bernilai `true`. Sintak dari perulangan dengan `while` di Java adalah sebagai berikut:

```
while (kondisi didalam perulangan)  
{  
    Statement s;  
    Tambahan statement untuk mengendalikan perulangan; (opsional)  
}
```

Variasi dari perulangan `while` adalah `do-while`. Berikut adalah sintak dari perulangan dengan `do-while` di Java:

```
do {  
    Statement s;  
} while (kondisi didalam perulangan)
```

Pada perulangan `do-while`, didalam blok `do` merupakan isi utama dari badan perulangan. Dimana didalam `do` tersebut bisa disisi beberapa statement. Kode 2.4 merupakan contoh penerapan `do-while` untuk membuat user memasukkan apapun angka kecuali 0 untuk menghitung nilai penjumlahan dari yang sudah dihitung sebelumnya.

### Kode 2.6 TestPenjumlahanDoWhile.java

```
1 import java.util.Scanner;
2 public class TestPenjumlahanDoWhile {
3     public static void main(String[] apa)
4     {
5         int angka;
6         int sum = 0;
7         Scanner input = new Scanner(System.in);
8         do {
9             System.out.println("Masukkan angka kecuali 0");
10            data = input.nextInt();
11            sum += data;
12        } while(data != 0);
13        System.out.println("Hasil Penjumlahannya adalah "+sum);
14    }
15 }
```

Perulangan terakhir dapat dilakukan dengan mekanisme **for**. Perulangan dengan **for** merupakan cara yang dapat dipahami dengan mudah oleh programmer pemula. Pada perulangan sebelumnya yaitu dengan **while** atau **do-while**, terkadang programmer lupa untuk mengendalikan variabel yang akan diulang-ulang dan akhirnya akan berakhiran pada perulangan tidak terhingga. Secara sederhana perulangan **for** dapat dibandingkan dengan perulangan **while** seperti berikut:

```
For (i = nilaiAwal; i < nilaiAkhir; i++){
    //perulangan
    ...
}
```

```
i = nilaiAwal
while(i < nilaiAkhir) {
    //perulangan
    ...
    i++;
}
```

Gambar 2.8 Perulangan

Secara umum, sintak dari perulangan **for** adalah sebagai berikut:

```
for (aksi-awal; kondisi didalam perulangan;  
aksi setelah setiap iterasi) {  
Statement s;  
}
```

Perulangan juga dapat dibuat bersarang atau dikenal dengan istilah *nested loop*. Program akan mengeksekusi lebih dahulu pada perulangan terdalam (*inner loop*) baru perulangan yang luar (*outer loop*). Kode 2.5 merupakan contoh penggunaan perulangan bersarang pada tabel perkalian.

#### Kode 2.7 TabelPerkalian.java

```
1 public class TabelPerkalian {  
2     public static void main(String[] apa)  
3     {  
4         // Judul dan heading tabel  
5         System.out.println(" Multiplication Table");  
6         // Angka judul  
7         System.out.print(" ");  
8         for (int j = 1; j <= 9; j++)  
9             System.out.print(" " + j);  
10        System.out.println("\n -----");  
11        // isi tabel  
12        for (int i = 1; i <= 9; i++) {  
13            System.out.print(i + " | ");  
14            for (int j = 1; j <= 9; j++) {  
15                // tampilkan perkalian  
16                System.out.printf("%4d", i * j);  
17            }  
18            System.out.println();  
19        }  
20    }
```

## **2.11 Kesimpulan**

BAB ini merupakan BAB untuk mengulas kembali materi-materi tentang pemrograman dasar dengan bahasa pemrograman Java. Materi-materi dasar pemrograman tersebut meliputi input/output console, identified, variabel, assignment, tipe data, operasi, logika pengambilan keputusan, dan perulangan. Dengan pengantar ini, diharapkan pada BAB-BAB berikutnya akan lebih mudah untuk memahami sintak dalam memprogram dengan pendekatan berorientasi objek pada bahasa pemrograman Java.

## **2.12 Latihan Soal**

Jawablah pertanyaan-pertanyaan dibawah ini dengan tepat!

1. Sebutkan kelebihan bahasa pemrograman Java?
2. Sebutkan apa saja tipe data primitif yang didukung oleh Java?
3. Sebutkan operator relasional di Java?
4. Sebutkan dan jelaskan jenis-jenis *statement IF* yang didukung oleh Java?
5. Sebutkan apa saja mekanisme perulangan yang didukung oleh Java?

## **2.13 Praktikum**

Lakukan praktik dibawah ini sesuai dengan permintaan input dan output yang telah diberikan dengan bahasa pemrograman Java!

### 1. Kasus 1: Ganjil-Genap

Terdapat kasus plat nomer ganjil genap yang diterapkan di Kota XXX. Dari plat nomor tersebut, ganjil genap ditentukan oleh angkanya saja, tidak dengan huruf. Buat program untuk mengotputkan lajur yang dipilih, jika genap lajur kiri, jika ganjil lajur kanan.

Contoh Input/Output:

Input #1: 1234

Output #1: Lajur kiri

Input #2: 1231

Output #2: Lajur kanan

## 2. Kasus 2: Palindrom

Palindrom merupakan suatu rangkaian kata yang sama ketika dibaca baik dari depan ataupun dari belakang. Misalnya, “katak”, “malam”, ”makam” merupakan contoh dari kata palindrom. Pada praktikum ini, programmer diminta untuk mengecek apakah kata yang di masukkan merupakan palindrom atau bukan.

Contoh Input/Output:

Input #1: makan

Output #1: makan bukan palindrom

Input #2: malam

Output #2: malam merupakan palindrom

## 3. Kasus 3: Segitiga sama siku

Gambarkan segitiga sama siku dengan rangkaian karakter ‘\*’. Alas dari segitiga sama siku tersebut ditentukan oleh angka yang dimasukkan oleh user.

Contoh Input/Output:

Input #1: 3

Output #1:     \*  
              \*\*  
              \*\*\*

Input #2: 4

Output #2:     \*  
              \*\*  
              \*\*\*  
              \*\*\*\*

## BAB 3 CLASS DAN OBJEK DALAM PBO

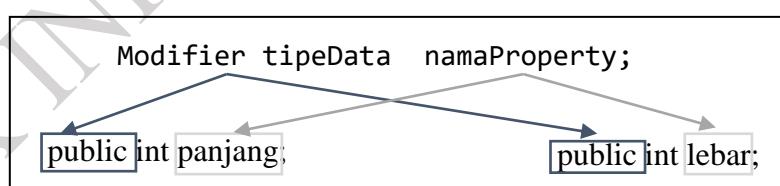
### 3.1 Mendefinisikan Class dari Objek

Pemrograman berbasis objek (PBO) adalah pemrograman yang menggunakan konsep objek. Objek sendiri memiliki representasi entitas di dunia nyata dapat diidentifikasi secara jelas. Contoh seorang mahasiswa, TV, persegi panjang, lingkaran, dan bahkan pembayaran dapat dilihat sebagai objek. Objek memiliki identitas, status, dan perilaku yang unik. Class adalah *blueprint*, cetak biru, atau kontrak yang menentukan apa yang akan menjadi bidang data dan metode objek. Objek adalah turunan dari kelas. Hubungan antar class dan objek dapat dianalogikan dengan hubungan antara resep bakso dan bakso. Artinya Kita dapat membuat bakso sebanyak yang Kita inginkan dari satu resep. Di dalam class terdapat property, method, dan konstruktor.

### 3.2 Property dan Method

#### 3.2.1 Property

Property atau keadaan objek (sering juga disebut atribut atau variabel) diwakili oleh bidang data dengan nilai saat itu. Contohnya adalah objek mahasiswa yang memiliki property NIM, nama, jenis kelamin, dan lain-lain. Objek persegi panjang memiliki bidang data panjang dan lebar. Panjang dan lebar tersebut merupakan property yang menjadi ciri sebuah persegi panjang. Cara penulisan dalam kode program sebagai berikut:



Gambar 3.1 Cara mendefinisikan property

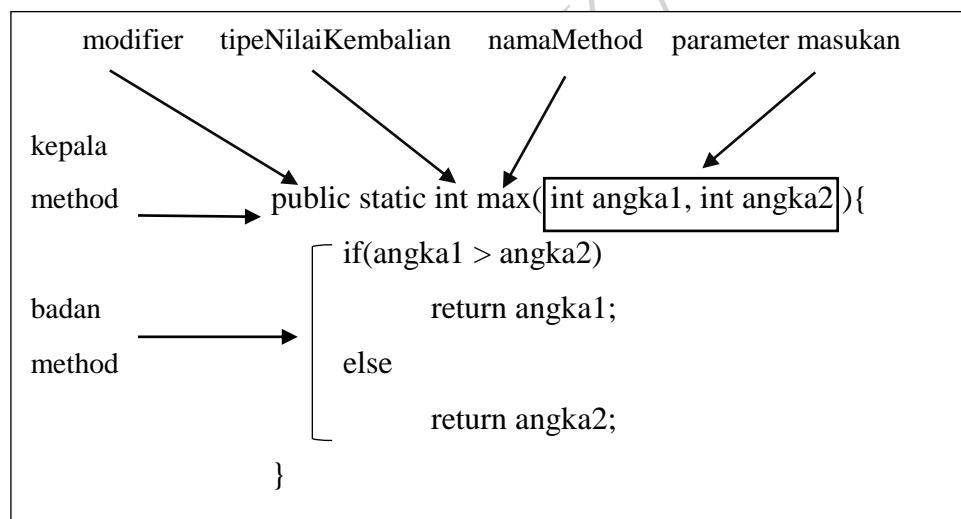
#### 3.2.2 Method

Method atau bisa disebut sebagai fungsi di pemrograman fungsional merupakan perilaku dari suatu objek. Misalnya kita dapat mendefinisikan method bernama `getLuas()` dan `getKeliling()` untuk objek persegi panjang. Sebuah objek persegi panjang dapat memanggil `getLuas()` untuk mengembalikan nilai luas dan `getKeliling()` untuk

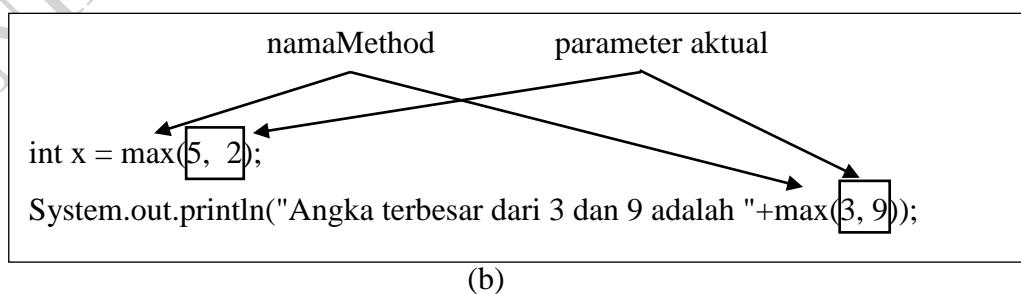
mengembalikan nilai keliling. Selain itu, kelas menyediakan method khusus, yang dikenal sebagai konstruktor, yang dipanggil untuk membuat objek baru. Konstruktor dirancang untuk melakukan tindakan inisialisasi, seperti menginisialisasi bidang data objek. Penjelasan tentang konstruktor yang lebih lengkap terdapat pada Bab 3.4. Cara penulisan method yang sederhana dalam kode program sebagai berikut:

```
modifier tipeNilaiKembalian namaMethod(parameter masukan){  
    //badan method  
}
```

Kepala method mendefinisikan modifier, tipe nilai kembalian, nama method, dan parameter masukan. Badan method berisi kode program untuk melaksanakan suatu tujuan. Modifier yang digunakan pada gambar 3.2 adalah public static. Modifier selengkapnya akan dijelaskan pada bab 3.6.



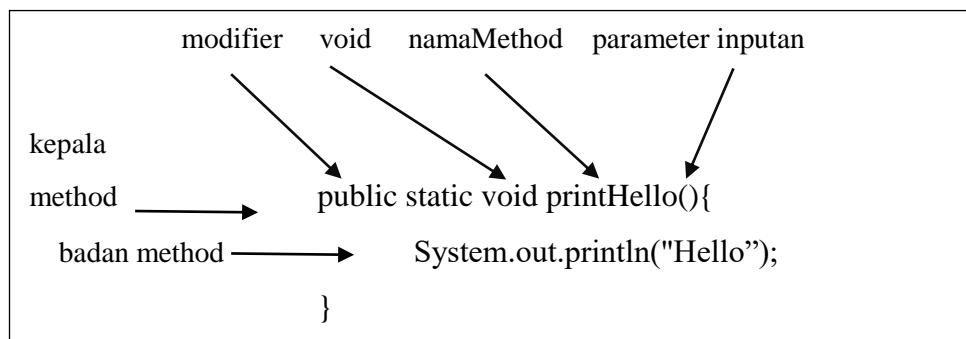
(a)



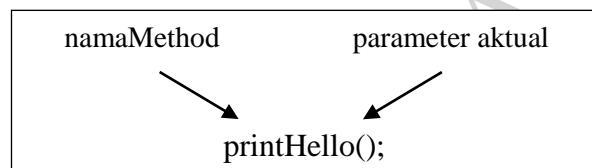
(b)

Gambar 3.2 Cara mendefinisikan (a) dan memanggil (b) method dengan nilai kembalian

Method bisa saja memiliki atau tidak memiliki nilai kembalian. Gambar 3.2 merupakan method yang dapat mengembalikan suatu nilai. Sedangkan method yang tidak memiliki nilai kembalian menggunakan kata kunci `void` seperti pada Gambar 3.3.



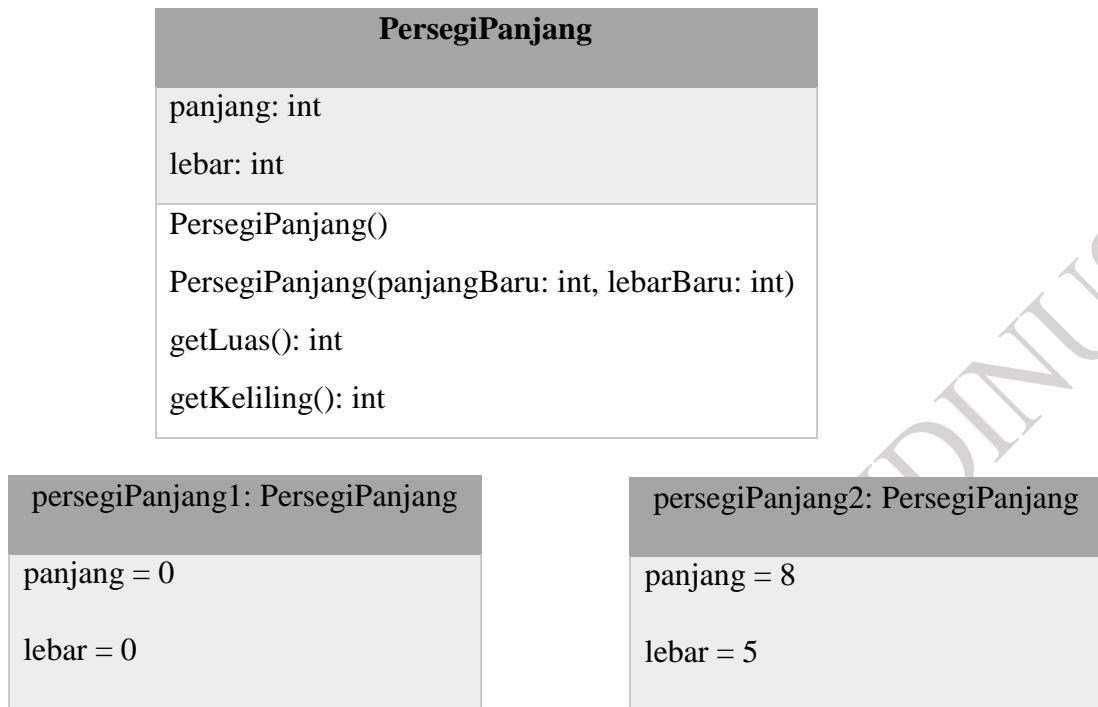
(a)



(b)

Gambar 3.3 Cara mendefinisikan (a) dan memanggil (b) method tanpa nilai kembalian

Parameter masukan pada method bekerja seperti *placeholder*. Ketika method tersebut dipanggil, Kita dapat memasukkan suatu nilai ke dalam parameter. Nilai tersebut dikenal dengan istilah parameter aktual. Jumlah parameter masukan sangat beragam sesuai kebutuhan, bahkan yang tidak mencantumkan parameter masukan juga tidak apa-apa. Kembali pada PersegiPanjang, Gambar 3.3 adalah contoh UML class diagram dimana PersegiPanjang memiliki 2 objek. Class PersegiPanjang sendiri memiliki 2 property bertipe data integer yaitu panjang dan lebar. Memiliki 2 konstruktor dan 2 method.



Gambar 3.3 Class dan objek dalam UML class diagram

Implementasi dari UML Gambar 3.3 tersebut disajikan pada Kode 3.1 dan Kode 3.2.

#### Kode 3.1 PersegiPanjang.java

```

1 public class PersegiPanjang {
2     //panjang dan lebar dari persegi panjang
3     int panjang;
4     int lebar; ] ← Property
5
6     //konstruktor dari objek PersegiPanjang
7     PersegiPanjang(){
8
9 }
10
11    //konstruktor dari objek PersegiPanjang
12    PersegiPanjang(int panjangBaru, int lebarBaru){
13        panjang = panjangBaru;
14        lebar = lebarBaru;
15    }

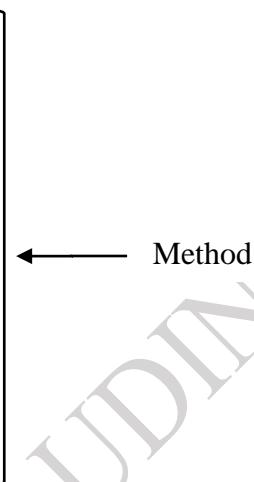
```

Annotations in the code:

- A bracket on line 4 spans from "int lebar;" to the closing brace of the class definition, labeled "Property".
- A large bracket on the right side of the code spans from the opening brace of the constructor in line 7 to the closing brace of the constructor in line 15, labeled "Konstruktor".

```

15
16     //mengembalikan nilai luas persegi panjang
17     int getLuas(){
18         return panjang*lebar;
19     }
20
21     //mengembalikan nilai keliling persegi panjang
22     int getKeliling(){
23         return 2*(panjang+lebar);
24     }
25 }
```



### Kode 3.2 PersegiPanjangDemo.java

```

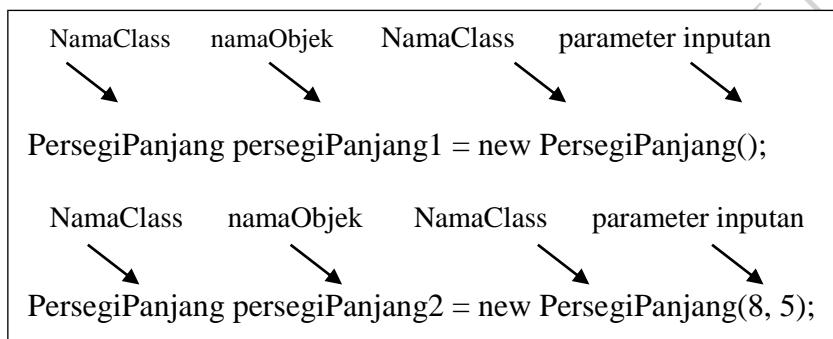
1 public class PersegiPanjangDemo {
2     /* Main Method */
3     public static void main(String[] args){
4         //membuat objek peregi panjang dengan panjang=0 dan lebar=0
5         PersegiPanjang persegiPanjang1 = new PersegiPanjang();
6         //memanggil variabel dari class
7         //memanggil method dari class
8         System.out.println("Luas persegi panjang 1 =" +
9             persegiPanjang1.panjang+" * "+persegiPanjang1.lebar+" =
"+persegiPanjang1.getLuas());
10
11         //membuat objek peregi panjang dengan panjang=8 dan lebar=5
12         PersegiPanjang persegiPanjang2 = new PersegiPanjang(8, 5);
13         System.out.println("Luas persegi panjang 2 = "+
14             persegiPanjang2.panjang+" * "+persegiPanjang2.lebar+
" = "+persegiPanjang2.getLuas());
15     }
16 }
```

**Output:**

```
Luas persegi panjang 1 = 0 * 0 = 0  
Luas persegi panjang 2 = 8 * 5 = 40
```

Berdasarkan Kode 3.1 terdapat *template* atau cetak biru persegi panjang yang didalamnya terdiri dari property, konstruktor, dan method. Sedangkan untuk membuat objek terdapat pada file PersegiPanjangDemo.java. Sintaks penulisan objek sebagai berikut:

```
NamaClass namaObjek = new NamaClass(parameter masukan);
```



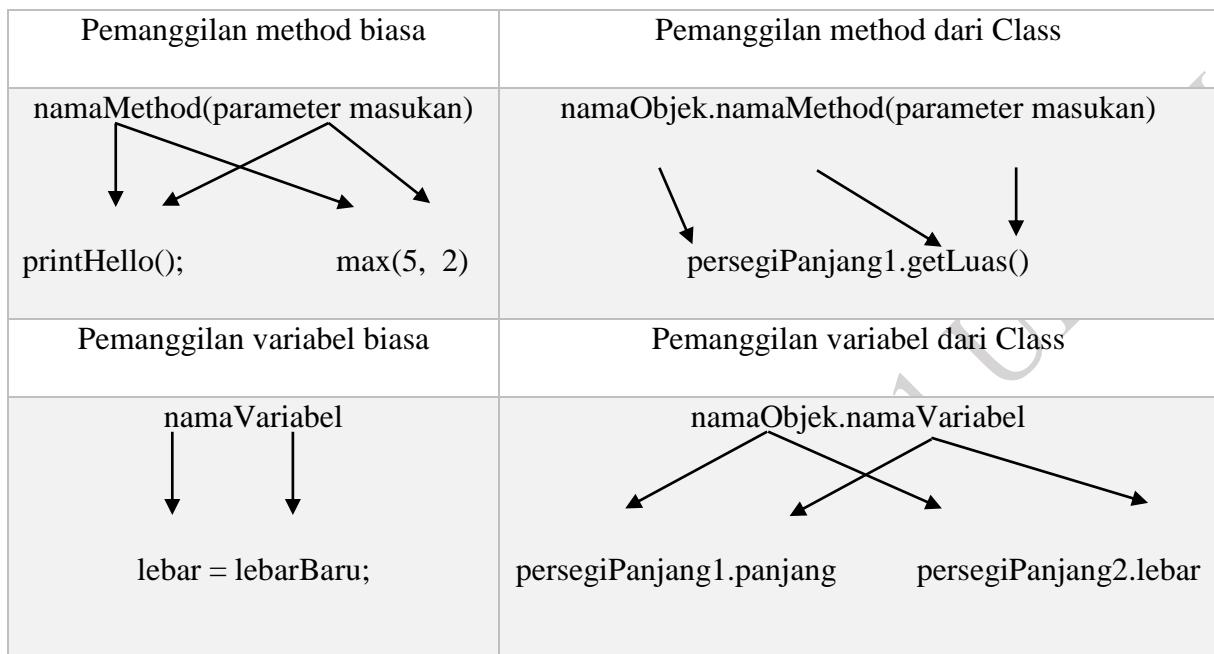
Gambar 3.4 Sintak penulisan objek

Saat objek pertama kali diinisialisasi, otomatis memanggil method konstruktor. Terdapat 2 konstruktor di class Persegi, konstruktor pertama tidak memiliki parameter masukan dan konstruktor ke-2 memiliki 2 parameter masukan. Memiliki nama method yang sama namun parameter yang berbeda disebut dengan overloading method. Penjelasan tentang overloading method yang lebih lengkap terdapat pada Bab 3.3.

Pada Gambar 3.4, menunjukkan bahwa objek persegiPanjang1 memanggil konstruktor pertama (`PersegiPanjang.java` baris ke-7). Konstruktor tersebut tidak melakukan tindakan apa pun, jadi property panjang dan lebar memiliki nilai awal yaitu 0. Sedangkan persegiPanjang2 memanggil konstruktor kedua. Konstruktor kedua ini menginisialisasi property panjang dengan nilai masukan panjangBaru dan menginisialisasi property lebar dengan nilai masukan lebarBaru.

Gambar 3.4 dan Kode 3.1 sama-sama menunjukkan pemanggilan method, namun dengan cara berbeda. Gambar 3.4 memanggil method biasa sedangkan kode 3.1 memanggil method dari class. Jika method biasa hanya menggunakan sintak `namaMethod(parameter masukan)`. Jika method dari class menggunakan sintak `namaObjek.namaMethod(parameter masukan)`.

masukan). Begitu pula cara memanggil variabel dari class menggunakan sintak `namaObjek.namaVariabel` sedangkan variabel biasa menggunakan sintak `namaVariabel`. Lihat Gambar 3.5 untuk mendapatkan informasi yang lebih jelas.



Gambar 3.5 Sintak pemanggilan method dan variabel dari class

### 3.3 Overloading Method

Overloading method adalah method-method yang memiliki nama yang sama namun berbeda-beda parameter masukannya. Dengan ini, akan membuat program lebih jelas dan lebih mudah dibaca. Tidak hanya dapat diterapkan pada konstruktor, overloading method juga dapat diterapkan pada method biasa. Contoh pada method `max` yang berfungsi untuk mengembalikan nilai terbesar.

```
public static int max(int angka1, int angka2){
    if(angka1 > angka2)
        return angka1;
    else
        return angka2;
}
```

Kode program method `max` diatas bekerja hanya untuk tipe data integer dengan 2 parameter masukan. Namun bagaimana jika ingin membuat method `max` yang bekerja untuk tipe data double atau lebih dari 2 parameter masukan? Solusinya adalah membuat method dengan

nama yang sama namun berbeda parameter masukannya. Berikut contoh kode program yang terdiri dari 3 method dengan nama yang sama namun dengan beberapa variasi parameter masukan.

### Kode 3.3 MethodOverloadingDemo.java

```
1 public class MethodOverloadingDemo {  
2     /* Main Method */  
3     public static void main(String[] args){  
4         /* Memanggil method max dengan parameter masukan integer */  
5         System.out.println("Angka terbesar dari 3 dan 9 adalah  
" +max(3, 9));  
6  
7         /* Memanggil method max dengan parameter masukan double */  
8         System.out.println("Angka terbesar dari 4.25 dan 4.0 adalah  
" +max(4.25, 4.0));  
9  
10        /* Memanggil method max dengan parameter masukan 3 double */  
11        System.out.println("Angka terbesar dari 4.25, 4.0, dan 5.6  
adalah " +max(4.25, 4.0, 5.6));  
12    }  
13  
14    // Mengembalikan nilai terbesar dari 2 parameter masukan integer  
15    public static int max(int angka1, int angka2){  
16        if(angka1 > angka2)  
17            return angka1;  
18        else  
19            return angka2;  
20    }  
21  
22    // Mengembalikan nilai terbesar dari 2 parameter masukan double  
23    public static double max(double angka1, double angka2){  
24        if(angka1 > angka2)  
25            return angka1;
```

```

25     else
26         return angka2;
27     }
28
29     // Mengembalikan nilai terbesar dari 3 parameter masukan double
30     public static double max(double angka1, double angka2, double
31     angka3){
32         return max(max(angka1, angka2), angka3);
33     }

```

Ketika memanggil `max(3, 9)`, method `max` yang digunakan adalah method `max` yang memiliki 2 parameter masukan integer. Ketika memanggil `max(4.25, 4.0)`, method `max` yang digunakan adalah method `max` yang memiliki 2 parameter masukan double. Ketika memanggil `max(4.25, 4.0, 5.6)`, method `max` yang digunakan adalah method `max` yang memiliki 3 parameter masukan double.

Misal, jika kita memanggil `max(2, 3.0)` apakah bisa dijalankan? Jawabannya adalah bisa. Ketika memanggil `max(2, 3.0)`, method `max` yang digunakan adalah method `max` yang memiliki 2 parameter masukan double. Masukan 2 akan otomatis berubah menjadi nilai double dan memenuhi method ini.

Kenapa method `max(double, double)` tidak digunakan untuk `max(3, 9)`, sedangkan baik `max(double, double)` dan `max(int, int)` dapat digunakan? Karena compiler Java menemukan bahwa method yang cocok untuk `max(3, 9)` adalah `max(int, int)` dibanding `max(double, double)`.

### 3.4 Konstruktor dan Destruktor

Konstruktor merupakan method khusus yang dirancang dan dipanggil untuk melakukan tindakan inisialisasi, seperti menginisialisasi bidang data objek. Dalam satu class, memungkinkan untuk memiliki lebih dari satu konstruktor namun memiliki parameter yang berbeda-beda yang disebut dengan overloading method. Sintak penulisan konstruktor sebagai berikut:

<pre>modifier NamaClass(parameter masukan){     //badan konstruktor }</pre>	
<pre>PersegiPanjang(){ }</pre>	<pre>PersegiPanjang(int panjangBaru, int lebarBaru){     panjang = panjangBaru;     lebar = lebarBaru; }</pre>

Gambar 3.6 Sintak penulisan konstruktor

Destruktor merupakan method khusus yang akan dipanggil saat objek dihapus dari memori. Pada bahasa pemrograman Java yang sekarang kita gunakan, tidak memiliki method destructor karna Java memiliki *garbage collector* untuk mengelola memori. *Garbage collector* dapat menghapus objek yang tidak terpakai dengan otomatis.

### 3.5 Static Variable dan Constant

Property panjang dan lebar dalam class `persegiPanjang` dikenal sebagai variabel instance. Variabel instance terkait dengan instance spesifik dari kelas, variabel tersebut tidak dibagikan di antara objek dari kelas yang sama. Misalnya, pada objek di bawah ini:

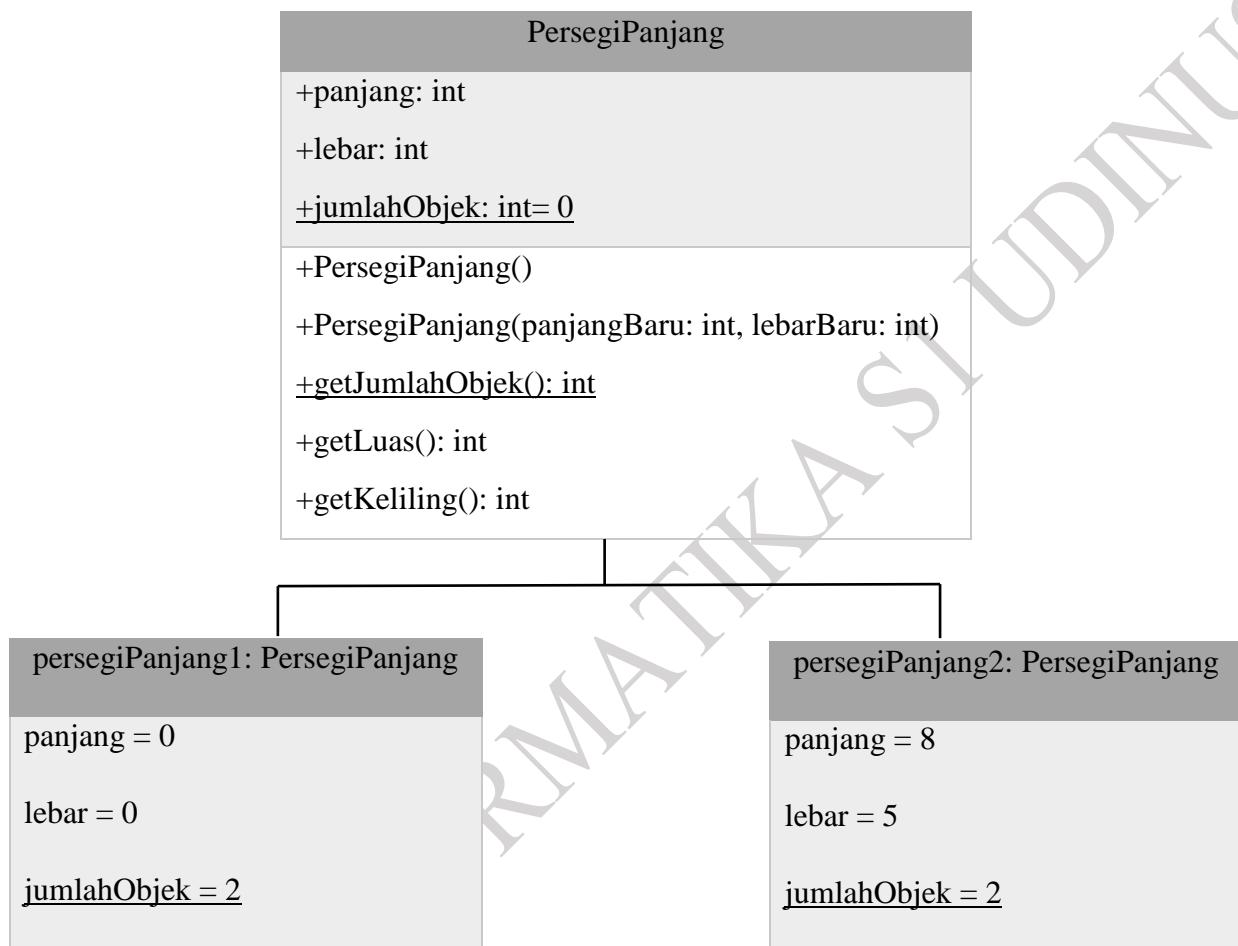
```
PersegiPanjang persegiPanjang1 = new PersegiPanjang();
PersegiPanjang persegiPanjang2 = new PersegiPanjang(8, 5);
```

Panjang dan lebar pada `persegiPanjang1` tidak tergantung pada panjang dan lebar di `persegiPanjang2` dan disimpan di lokasi memori yang berbeda. Perubahan yang dilakukan pada panjang atau lebar `persegiPanjang1` tidak memengaruhi panjang atau lebar `persegiPanjang2`, dan sebaliknya.

Jika Kita ingin semua variable instance dapat berbagi data, gunakan variabel static atau dikenal juga sebagai variabel class. Variabel static menyimpan nilai untuk variabel di lokasi memori umum. Karena lokasi umum ini, jika satu objek mengubah nilai variabel static, semua objek dari kelas yang sama akan terpengaruh. Java mendukung variabel static dan juga method static. Method static dapat dipanggil tanpa membuat turunan dari kelas.

Mari kita modifikasi class `PersegiPanjang` dengan menambahkan variabel-variabel `jumlahObjek` untuk menghitung jumlah objek `PersegiPanjang` yang dibuat. Ketika objek pertama class ini dibuat, `jumlahObjek` adalah 1. Ketika objek kedua dibuat, `jumlahObjek` menjadi 2. UML dari class `PersegiPanjang` yang baru ditunjukkan pada Gambar 3.7. Class

PersegiPanjang mendefinisikan variabel panjang dan lebar sebagai variabel instance dan metode `getLuas()`, `getKeliling()` sebagai method instance sedangkan variabel `jumlahObjek` sebagai variabel static dan method `getJumlahObjek` sebagai method static. (Perhatikan variabel dan metode statis digarisbawahi dalam diagram kelas UML.)



Setelah 2 object PersegiPanjang dibuat, maka jumlahObjek = 2

Gambar 3.7 UML class diagram dengan variabel dan method static

Kode 3.4 merupakan implementasi dari Class `PersegiPanjang` setelah ditambahkan variabel static dan method static. Kode 3.5 merupakan kode untuk pengetesan Class `PersegiPanjang` yang telah dimodifikasi sebelumnya.

#### Kode 3.4 PersegiPanjang.Java

1	public class PersegiPanjang {
2	//panjang dan lebar dari persegi panjang

```

3  public int panjang;
4  public int lebar;
5  public static int jumlahObjek = 0; ← Variabel static
6
7  //konstruktor dari objek PersegiPanjang
8  public PersegiPanjang(){
9      jumlahObjek++; ← Menambah nilai 1 pada
10 }
11
12 //konstruktor dari objek PersegiPanjang
13 public PersegiPanjang(int panjangBaru, int lebarBaru){
14     panjang = panjangBaru;
15     lebar = lebarBaru; ← Menambah nilai 1 pada
16     jumlahObjek++; ← variabel jumlahObjek
17 }
18
19 //mengembalikan nilai getJumlahObjek
20 public static int getJumlahObjek(){
21     return jumlahObjek; } ← Method Static
22
23 //mengembalikan nilai luas persegi panjang
24 public int getLuas(){
25     return panjang*lebar;
26 }
27
28 //mengembalikan nilai keliling persegi panjang
29 public int getKeliling(){
30     return 2*(panjang+lebar);
31 }
32 }
```

### Kode 3.5 PersegiPanjangDemo.java

```
1 public class PersegiPanjangDemo {  
2     /* Main Method */  
3     public static void main(String[] args){  
4         System.out.println("Sebelum membuat objek");  
5         //memanggil variabel static  
6         System.out.println("Jumlah objek = "+  
7             PersegiPanjang.jumlahObjek);  
8  
9         //Membuat objek persegiPanjang1  
10        PersegiPanjang persegiPanjang1 = new PersegiPanjang();  
11        System.out.println("\nSetelah membuat objek persegiPanjang1");  
12  
13        //memanggil variabel instance  
14        System.out.println("Panjang = "+persegiPanjang1.panjang+  
15        "\nLebar = "+persegiPanjang1.lebar);  
16  
17        //memanggil method static  
18        System.out.println("Jumlah objek = "+  
19            PersegiPanjang.getJumlahObjek());  
20  
21        //Membuat objek persegiPanjang2  
22        PersegiPanjang persegiPanjang2 = new PersegiPanjang(8, 5);  
23        System.out.println("\nSetelah membuat objek persegiPanjang1");  
24        //memanggil variabel instance  
25        System.out.println("Panjang = "+ persegiPanjang2.panjang +  
26        "\nLebar="+persegiPanjang2.lebar);  
27  
28        //memanggil variabel static  
29        System.out.println("Jumlah objek = "+  
30            PersegiPanjang.jumlahObjek);  
31    }
```

27 }

### Output:

Sebelum membuat objek

Jumlah objek = 0

Setelah membuat objek persegiPanjang1

Panjang = 0

Lebar = 0

Jumlah objek = 1

Setelah membuat objek persegiPanjang2

Panjang = 8

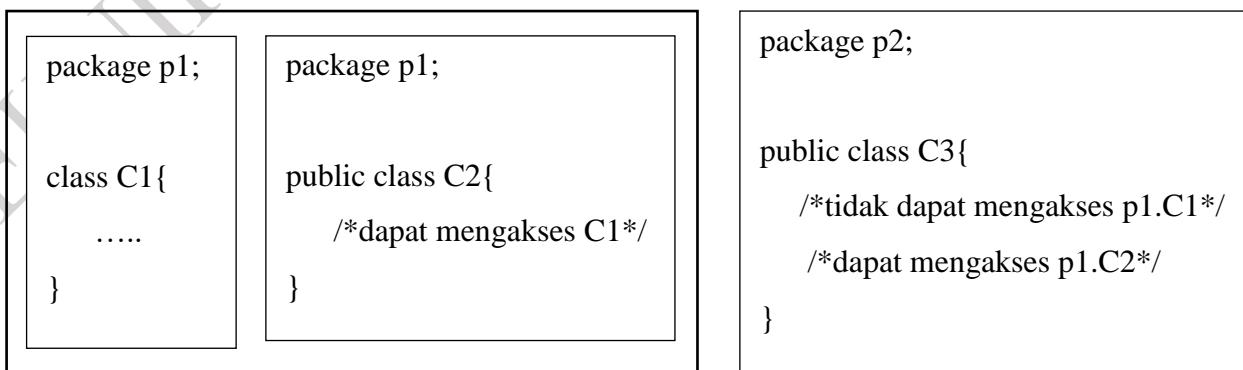
Lebar = 5

Jumlah objek = 2

Berdasarkan Kode 3.4 dan Kode 3.5, variabel instance (panjang dan lebar) milik instance dan memiliki penyimpanan memori yang tidak tergantung satu sama lain. Variabel static (jumlahObjek) dibagikan oleh semua instance dari kelas yang sama.

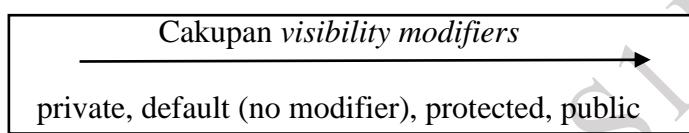
### 3.6 Visibility Modifiers

*Visibility modifiers* dapat digunakan untuk menentukan visibilitas kelas dan anggotanya. Kita dapat menggunakan *visibility modifier* **public** untuk class, method, dan property untuk menyatakan bahwa mereka dapat diakses dari kelas lain. Jika tidak ada *visibility modifiers* yang digunakan, maka *visibility modifiers* nya **default** yang berarti class, method, dan property dapat diakses oleh kelas apa pun dalam *package* yang sama, dikenal sebagai *package-private* atau *package-access*. Gambaran modifier **public** dan **default** untuk package terdapat pada gambar 3.8.



Gambar 3.8 Gambaran modifier public dan default untuk package

Selain *visibility modifier* **public** dan **default**, Java menyediakan *visibility modifier* **private** dan **protected** untuk anggota kelas. Modifier **private** membuat method dan property hanya dapat diakses dari dalam kelasnya sendiri. Untuk modifier **protected**, Kita dapat mengakses bidang data yang dilindungi atau metode dalam superclass dari subclassnya. Modifier **protected** mengizinkan subclass untuk mengakses property atau metode yang didefinisikan dalam superclass, tetapi tidak mengizinkan non subclass dalam package yang berbeda untuk mengakses itu. Materi superclass dan subclass akan dibahas selengkapnya pada Bab 7. Skema visibility modifier meningkat dalam urutan di bawah ini, sesuai Gambar 3.9.



Gambar 3.9 Cakupan *visibility modifier*

Dari gambar 3.9 menjelaskan bahwa cakupan paling kecil adalah **private**, dimana modifier **private** hanya dapat diakses pada class yang sama. Selanjutnya adalah modifier **default**, dimana modifier **default** hanya dapat diakses pada class dan package yang sama. Disusul modifier **protected**, dimana modifier **protected** dapat diakses pada class dan package yang sama serta subclass baik di package yang berbeda atau package yang sama. Terakhir, cakupan paling besar adalah **public**, dimana modifier **public** dapat diakses pada class dan package yang sama maupun berbeda serta subclass baik di package yang berbeda atau package yang sama. Ringkasan cakupan visibilitas keempat modifier dapat dilihat pada Tabel 3.1

Tabel 3.1 Visibilitas property dan method

Modifier pada anggota class	Akses dari class yang sama	Akses dari package yang sama	Akses dari subclass di luar package	Akses dari luar package
Public (+)	v	v	v	v
Protected (#)	v	v	v	-
Default/ no modifier	v	v	-	-
Private (-)	v	-	-	-

### 3.7 Kesimpulan

1. Class adalah *template*, cetak biru, atau kontrak yang menentukan apa yang akan menjadi bidang data dan metode objek.
2. Objek adalah turunan dari kelas. Kita dapat menggunakan kata kunci **new** untuk membuat objek baru dan operator titik(.) untuk mengakses property dan method dari objek tersebut.
3. Overloading method adalah method-method yang memiliki nama yang sama namun berbeda-beda parameter masukannya.
4. Konstruktor merupakan method khusus yang dirancang dan dipanggil untuk melakukan tindakan inisialisasi. Destruktor merupakan method khusus yang akan dipanggil saat objek dihapus dari memori. Bahasa pemrograman Java tidak memiliki method destructor karna Java memiliki *garbage collector* untuk mengelola memori.
5. Terdapat 2 macam property/ method antara lain:
  - a. Property/ method instance terkait dengan instance spesifik dari kelas, property/ method tersebut tidak dibagikan di antara objek dari kelas yang sama. Cara pemanggilannya yaitu NamaObjek.property atau NamaObjek.namaMethod.
  - b. Property/ method statis dapat digunakan bersama oleh semua instance dari kelas yang sama. Cara pemanggilannya menggunakan NamaClass.property atau NamaClass.namaMethod.

6. *Visibility modifiers* dapat digunakan untuk menentukan visibilitas kelas dan anggotanya. Terdapat 4 variasi *visibility modifiers* antara lain:
  - a. *Visibility modifier public* untuk class, method, dan property untuk menyatakan bahwa mereka dapat diakses dari kelas lain.
  - b. Jika tidak ada *visibility modifiers* yang digunakan, maka *visibility modifiers* nya **default (no modifier)** yang berarti class, method, dan property dapat diakses oleh kelas apa pun dalam *package* yang sama.
  - c. Modifier **private** membuat method dan property hanya dapat diakses dari dalam kelasnya sendiri.
  - d. Modifier **protected** mengizinkan subclass untuk mengakses property atau metode yang didefinisikan dalam superclass, tetapi tidak mengizinkan non subclass dalam package yang berbeda untuk mengakses itu.

### 3.8 Kuis dan Latihan Soal

1. *Template*, cetak biru, atau kontrak yang menentukan apa yang akan menjadi bidang data dan metode objek disebut ...
2. Jelaskan apa perbedaan property dan method!
3. Mengapa diperlukan overloading method?
4. Apakah pada pemrograman Java terdapat destructor? Jelaskan penyebabnya!
5. Property/ method yang dapat digunakan bersama oleh semua instance dari kelas yang sama disebut ...
6. Jelaskan masing-masing perbedaan dari ke-4 *visibility modifier*!

### 3.9 Praktikum

1. Buatlah program class Lingkaran yang merepresentasikan template lingkaran. Di dalam class tersebut terdapat:
  - a. Property jari\_jari dengan tipe data double
  - b. Property phi dengan tipe data double bernilai 3.14
  - c. Property static jumlahLingkaran dengan tipe data integer bernilai 0
  - d. Konstruktor pertama tanpa ada masukan dan tidak melakukan apa-apa
  - e. Konstruktor kedua memiliki 1 masukan yaitu jari\_jari\_baru bertipe data double
  - f. Method getLuas, mengembalikan nilai luas lingkaran

- g. Method getKeliling, mengembalikan nilai keliling lingkaran
- h. Method static getJumlahLingkaran, mengembalikan nilai jumlah lingkaran yang dibuat

Gunakan modifier default untuk semua property dan method. Buat juga 3 objek, objek pertama tanpa inisialisasi jari\_jari, objek kedua dengan inisialisasi jari\_jari dari property, dan objek ketiga dengan inisialisasi jari\_jari dari konstruktor. Gambar pula UML class diagramnya.

2. Buatlah program class Piramida yang merepresentasikan bentuk piramida. Kreasikan konten property, method, dan objek sesuai keinginan Anda.
3. Buatlah program class Mahasiswa berdasarkan UML class diagram di bawah ini

Mahasiswa
<pre>+nim: string +nama: string +alamat: string +ipk: double</pre>
<pre>+Mahasiswa() +Mahasiswa(nim: string) +Mahasiswa(nimBaru: string, namaBaru: string, alamatBaru: string, ipkBaru: string) +predikat(ipk: double): string +cetak(): void</pre>

Penjelasan UML class diagram Mahasiswa

#### Atribut

- nim bertipe data string dengan modifier public
- nama bertipe data string dengan modifier public
- alamat bertipe data string dengan modifier public
- ipk bertipe data double dengan modifier public

#### Konstruktor

- Mahasiswa()
- Tidak melakukan apapun
- Mahasiswa(nim: string)
  - Beri nilai property nim milik class diisi dengan nilai nim masukan method

- Mahasiswa(nimBaru: string, namaBaru: string, alamatBaru: string, ipkBaru: string)
  - Beri nilai property nim milik class diisi dengan nilai nimBaru masukan method
  - Beri nilai property nama milik class diisi dengan nilai namaBaru masukan method
  - Beri alamat property nama milik class diisi dengan nilai alamatBaru masukan method
  - Beri nilai property ipk milik class diisi dengan nilai ipkBaru masukan method

### Method

- predikat(ipk: double): String
  - Jika ipk diantara 2.0 sampai 2.75, kembalikan string “Memuaskan”
  - Jika ipk diantara 2.76 sampai 3.5, kembalikan string ‘Sangat memuaskan’
  - Jika ipk diantara 3.51 sampai 4.0, kembalikan string “Dengan pujian”
  - Jika tidak, kembalikan string “-“
- cetak(): void
  - Menampilkan nim, nama, alamat, ipk dan predikat dalam susunan vertical

#### Contoh output:

```
Nama    = Dilan
Alamat = Bandung
NIM    = 1
IPK    = 3.51
Predikat = Dengan Pujian
```

## BAB 4 ENKAPSULASI

### 4.1 Konsep Enkapsulasi

Konsep enkapsulasi adalah membuat property menjadi modifier **private** sehingga dapat melindungi data dan membuat kelas mudah dikelola. Encapsulasi ini memungkinkan programmer untuk aturan membuat kode program seperti: *read-only* atau *write-only*. Sebagai contoh, ketika programmer Google membuat API Google Map, tidak mungkin Google memberikan akses untuk satelit secara langsung kepada programmer *third-party* (programmer biasa yang menggunakan API Google Map untuk mengakses peta). Hal tersebut dapat dilindungi dengan membuat beberapa akses terbatas pada method atau property sehingga programmer *third-party* hanya dapat membaca atau mengambil nilainya saja tanpa bisa merubah nilainya. Lebih sederhananya, perhatikan pemberian nilai pada property panjang dan lebar pada class PersegiPanjangDemo sebelumnya.

```
persegiPanjang2.panjang = 6;  
persegiPanjang2.lebar = 7;
```

Pemberian nilai tersebut dilakukan secara langsung ke property. Cara seperti itu merupakan contoh yang tidak bagus karena dua alasan, yaitu:

1. Data dapat dirusak. Misalnya, property jumlahObjek berfungsi untuk menampung nilai jumlah objek yang dibuat, tetapi bisa saja diset dengan manual misalnya,

```
Lingkaran.jumlahObjek = 10;
```

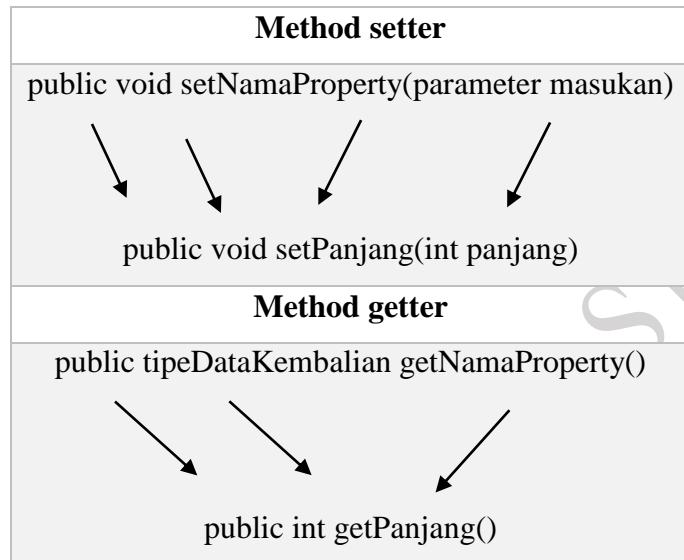
2. Class menjadi sulit dipertahankan dan rentan terhadap bug. Misalkan kita ingin memodifikasi class PersegiPanjang untuk memastikan bahwa nilai panjang tidak negatif setelah yang lain program sudah menggunakan class tersebut. Kita harus mengubah tidak hanya class Circle tetapi juga program yang menggunakan klien mungkin telah memodifikasi nilai panjang secara langsung misalnya

```
persegiPanjang2.panjang = -6;
```

Untuk mencegah modifikasi property secara langsung, Kita harus mendeklarasikan property dengan modifier **private**. Cara tersebut dikenal sebagai enkapsulasi. Property dengan modifier **private** tidak dapat diakses oleh objek dari luar class. Namun, klien sering mengambil dan memodifikasi property. Untuk membuat sebuah property **private** dapat diakses, Kita perlu menyediakan method **setter** dan **getter**.

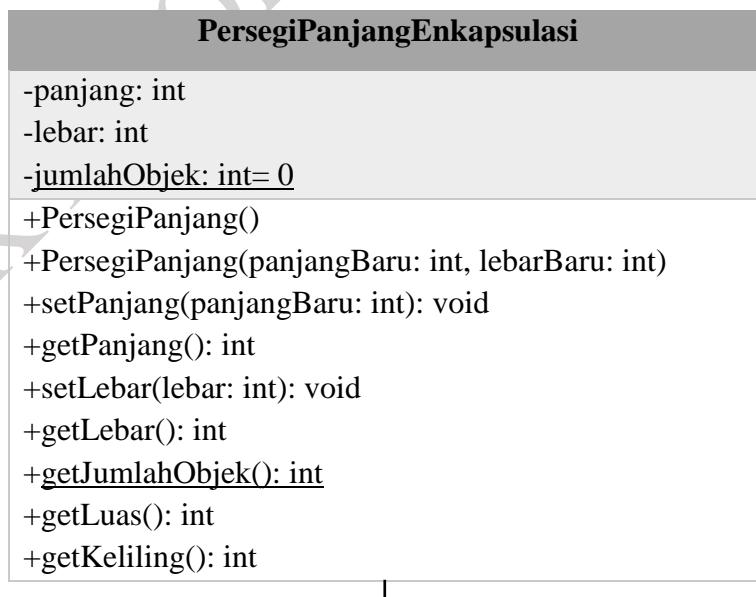
## 4.2 Setter-Getter Method

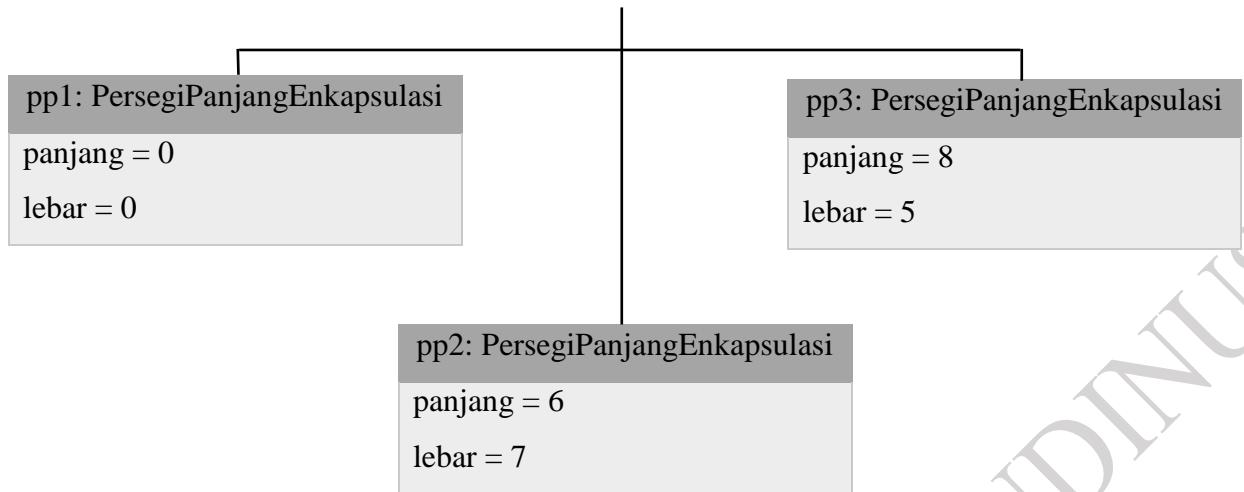
Method **setter** berfungsi untuk memperbarui nilai dari property **private** sedangkan method **getter** berfungsi untuk mengambil nilai dari property **private**. Method **setter** juga dikenal dengan nama *mutator* dan method **getter** dikenal dengan nama *accesor*. Kedua method tersebut memiliki sintak sebagai berikut:



Gambar 4.1 Sintak method setter dan getter

Mari kita buat class PersegiPanjang biasa pada Gambar 3.7 pada Bab sebelumnya ke dalam bentuk enkapsulasi dengan dilengkapi method setter dan getter.





Gambar 4.2 Class dan Objek PersegiPanjangEnkapsulasi dalam UML Class Diagram

Pada gambar UML class diagram di atas, terdapat beberapa perbedaan. Pertama pada property, lambang + diganti dengan – agar modifier **public** berganti dengan **private**. Kedua bertambahnya 4 method, yaitu method **setter** dan **getter** untuk panjang serta method **setter** dan **getter** untuk lebar. Dari Gambar 4.2 kita bisa membuat kode program seperti pada Kode 4.1 dan pengetesannya pada Kode 4.2.

#### Kode 4.1 PersegiPanjangEnkapsulasi.java

```

1  public class PersegiPanjangEnkapsulasi {
2      //panjang, lebar, dan jumlahObjek dengan modifier private
3      private int panjang;
4      private int lebar;
5      private int jumlahObjek = 0;
6
7      //konstruktor dari objek PersegiPanjang
8      public PersegiPanjangEnkapsulasi(){
9          jumlahObjek++;
10     }
11     //konstruktor dari objek PersegiPanjang
12     public PersegiPanjangEnkapsulasi(int panjangBaru, int
13         lebarBaru){
14         panjang = panjangBaru;
15         lebar = lebarBaru;
16     }
17
18     public int getPanjang() {
19         return panjang;
20     }
21
22     public void setPanjang(int panjang) {
23         this.panjang = panjang;
24     }
25
26     public int getLebar() {
27         return lebar;
28     }
29
30     public void setLebar(int lebar) {
31         this.lebar = lebar;
32     }
33
34     public int getJumlahObjek() {
35         return jumlahObjek;
36     }
37
38     public void setJumlahObjek(int jumlahObjek) {
39         this.jumlahObjek = jumlahObjek;
40     }
41
42     public String toString() {
43         return "PersegiPanjangEnkapsulasi [panjang=" + panjang +
44             ", lebar=" + lebar + ", jumlahObjek=" + jumlahObjek + "]";
45     }
46
47 }

```

Property private

```

15     jumlahObjek++;
16 }
17 //method setter untuk property panjang
18 public void setPanjang(int panjangBaru){
19     panjang = panjangBaru;
20 }
21 //method getter untuk property panjang
22 public int getPanjang(){
23     return panjang;
24 }
25 //method setter untuk property lebar
26 public void setLebar(int lebar){
27     //gunakan kata kunci this karena nama masukan parameter
28     //lebar sama dengan nama property lebar
29     this.lebar = lebar;
30 }
31 //method getter untuk property lebar
32 public int getLebar(){
33     return lebar;
34 }
35 //method getter untuk mengembalikan nilai getJumlahObjek
36 public int getJumlahObjek(){
37     return jumlahObjek;
38 }
39 //method getter untuk mengembalikan nilai luas persegi panjang
40 public int getLuas(){
41     return panjang*lebar;
42 }
43 //method getter untuk mengembalikan nilai keliling
44 public int getKeliling(){
45     return 2*(panjang+lebar);
46 }

```

47	}
----	---

#### Kode 4.2 PersegiPanjangEnkapsulasiDemo.java

```
1 public class PersegiPanjangEnkapsulasiDemo {  
2     /* Main Method */  
3  
4     public static void main(String[] args){  
5         System.out.println("Sebelum membuat objek");  
6         System.out.println("Jumlah objek = "+  
7             PersegiPanjangEnkapsulasi.getJumlahObjek());  
8  
9         //membuat objek persegi panjang dengan panjang=0 dan lebar=0  
10        PersegiPanjangEnkapsulasi pp1 = new  
11            PersegiPanjangEnkapsulasi();  
12            System.out.println("Luas persegi panjang 1 = "+  
13                pp1.getPanjang()+" * "+pp1.getLebar()+" = "+pp1.getLuas());  
14            System.out.println("Jumlah objek = "+  
15                PersegiPanjangEnkapsulasi.getJumlahObjek());  
16  
17         //membuat objek persegi panjang dengan panjang=6 dan lebar=7  
18        PersegiPanjangEnkapsulasi pp2 = new  
19            PersegiPanjangEnkapsulasi();  
20            pp2.setPanjang(6);  
21            pp2.setLebar(7);  
22            System.out.println("Luas persegi panjang 2 = "+  
23                pp2.getPanjang()+" * "+pp2.getLebar()+" = "+pp2.getLuas());  
24            System.out.println("Jumlah objek = "+  
25                PersegiPanjangEnkapsulasi.getJumlahObjek());  
26  
27         //membuat objek persegi panjang dengan panjang=8 dan lebar=5  
28        PersegiPanjangEnkapsulasi pp3 = new
```

```

27     PersegiPanjangEnkapsulasi(8, 5);
28     System.out.println("Luas persegi panjang 3 = "+
29         pp3.getPanjang()+" * "+pp3.getLebar()+" = "+pp3.getLuas());
30     System.out.println("Jumlah objek = "+
31         PersegiPanjangEnkapsulasi.getJumlahObjek());
32 }
33 }
```

**Output:**

Sebelum membuat objek

Jumlah objek = 0

Luas persegi panjang 1 = 0 \* 0 = 0

Jumlah objek = 1

Luas persegi panjang 2 = 6 \* 7 = 42

Jumlah objek = 2

Luas persegi panjang 3 = 8 \* 5 = 40

Jumlah objek = 3

Berdasarkan UML class diagram Gambar 4.2, simbol ‘-‘ ditulis dengan modifier **private** yang dapat dilihat diawal Kode 4.1 PersegiPanjangEnkapsulasi.java. Method **setter** untuk property **panjang** terdapat pada method **setPanjang** yang berfungsi untuk memberi nilai property **panjang** diisi dengan nilai masukan **panjangBaru** milik method. Method **getter** pada **panjang** terdapat pada method **getPanjang** hanya berfungsi untuk mengembalikan nilai dari property **panjang**.

Masih di dalam file PersegiPanjangEnkapsulasi.java, method **setter** untuk property **lebar** terdapat method **setLebar** berfungsi untuk memberi nilai property **lebar** milik class diisi dengan nilai masukan method. Karena nama masukan **lebar** sama dengan nama property **lebar**, maka harus menggunakan kata kunci **this**. Sedangkan method **getter** pada **lebar** terdapat pada baris 37 – 39. Didalam method **getLebar** hanya berfungsi untuk mengembalikan nilai dari property **lebar**. Method **getJumlahObjek**, **getLuas**, dan **getKeliling** bisa disebut dengan method **getter** karena hanya berfungsi untuk mengembalikan nilai.

File PersegiPanjangEnkapsulasiDemo.java berisi tentang deklarasi 3 objek antara lain objek pp1, pp2, dan pp3. Cara memanggil method **setter** yaitu pp2.setPanjang(6);. Sedangkan sebelumnya, cara memberi nilai pada property seperti ini persegiPanjang2.panjang = 6;. Cara memanggil method **getter** yaitu pp2.getPanjang(). Sedangkan kode sebelumnya (Kode 3.4), cara mengambil nilai pada property seperti ini persegiPanjang2.panjang. Pada Bab ini kita mempelajari bahwa penggunaan method **setter** dan **getter** dapat melindungi data dan membuat kelas mudah dikelola.

### 4.3 Kesimpulan

1. Konsep enkapsulasi adalah membuat property menjadi memiliki akses dengan batasan khusus sehingga dapat melindungi data dan membuat kelas mudah dikelola.
2. Property dengan modifier **private** tidak dapat diakses oleh objek dari luar class. Untuk membuat sebuah property **private** dapat diakses, kita perlu menyediakan mekanisme method **setter** dan **getter**.
3. Method **setter** berfungsi untuk memperbarui nilai dari property **private** sedangkan method **getter** berfungsi untuk mengambil nilai dari property **private**.

### 4.4 Kuis dan Latihan Soal

1. Jelaskan konsep enkapsulasi!
2. Sebutkan alasan kenapa kita harus menggunakan enkapsulasi!
3. Jelaskan perbedaan method setter dan getter!

### 4.5 Praktikum

1. Buatlah program class LingkaranEnkapsulasi yang merupakan modifikasi dari class Lingkaran. Class LingkaranEnkapsulasi merepresentasikan template Lingkaran yang menerapkan konsep enkapsulasi. Di dalam class tersebut terdapat:
  - a. Property jari\_jari dengan tipe data double
  - b. Property phi dengan tipe data double bernilai 3.14
  - c. Property static jumlahLingkaran dengan tipe data integer bernilai 0
  - d. Konstruktor pertama tanpa ada masukan dan tidak melakukan apa-apa
  - e. Konstruktor kedua memiliki 1 masukan yaitu jari\_jari\_baru bertipe data double

- f. Method setJari memiliki 1 masukan yaitu jari\_jari bertipe data double yang memperbarui nilai dari property jari\_jari
- g. Method getJari, mengembalikan nilai jari\_jari
- h. Method getLuas, mengembalikan nilai luas lingkaran
- i. Method getKeliling, mengembalikan nilai keliling lingkaran
- j. Method static getJumlahLingkaran, mengembalikan nilai jumlah lingkaran yang dibuat

Gunakan modifier **private** untuk semua property serta modifier **public** untuk semua konstruktor dan method. Buat juga 3 objek, objek pertama tanpa inisialisasi jari\_jari, objek kedua dengan inisialisasi jari\_jari dari method **setter** setJari, dan objek ketiga dengan inisialisasi jari\_jari dari konstuktor. Gambar pula UML class diagramnya.

2. Buatlah program class PiramidaEnkapsulasi yang merepresentasikan bentuk piramida yang merepresentasikan template Piramida dan menerapkan konsep enkasulasi. Kreasikan konten property, method, dan objek sesuai keinginan Anda.
3. Buatlah program class MahasiswaEnkapsulasi berdasarkan UML class diagram di bawah ini

MahasiswaEnkapsulasi
-nim: string
-nama: string
-alamat: string
-ipk: double
+Mahasiswa()
+Mahasiswa(nim: string)
+Mahasiswa(nimBaru: string, namaBaru: string, alamatBaru: string, ipkBaru: string)
+setNim(nim: string): void
+getNim(): string
+setNama(nama: string): void
+getNama(): string
+setAlamat(alamat: string): void
+getAlamat (): string

```
+setIpk(ipk: double): void  
+getIpk(): double  
+predikat(ipk: double): Sstring  
+cetak(): void
```

Penjelasan UML class diagram MahasiswaEnkapsulasi

### Atribut

- nim bertipe data string dengan modifier private
- nama bertipe data string dengan modifier private
- alamat bertipe data string dengan modifier private
- ipk bertipe data double dengan modifier private

### Konstruktor

- Mahasiswa()  
Tidak melakukan apapun
- Mahasiswa(nim: string)
  - Beri nilai property nim milik class diisi dengan nilai nim masukan method
- Mahasiswa(nimBaru: string, namaBaru: string, alamatBaru: string, ipkBaru: string)
  - Beri nilai property nim milik class diisi dengan nilai nimBaru masukan method
  - Beri nilai property nama milik class diisi dengan nilai namaBaru masukan method
  - Beri alamat property nama milik class diisi dengan nilai alamatBaru masukan method
  - Beri nilai property ipk milik class diisi dengan nilai ipkBaru masukan method

### Method

- setNim(nim: string): void
  - memberi nilai property nim milik class diisi dengan nilai masukan method
- +getNim(): string
  - mengembalikan nilai dari property nim
- +setNama(nama: string): void

- memberi nilai property nama milik class diisi dengan nilai masukan method
- `+getNama(): string`  
mengembalikan nilai dari property nama
  - `+setAlamat(alamat: string): void`  
memberi nilai property alamat milik class diisi dengan nilai masukan method
  - `+getAlamat (): string`  
mengembalikan nilai dari property alamat
  - `+setIpk(ipk: double): void`  
memberi nilai property ipk milik class diisi dengan nilai masukan method
  - `+getIpk(): double`  
mengembalikan nilai dari property ipk
  - `predikat(ipk: double): String`  
Jika ipk diantara 2.0 sampai 2.75, kembalikan string “Memuaskan”  
Jika ipk diantara 2.76 sampai 3.5, kembalikan string ‘Sangat memuaskan’  
Jika ipk diantara 3.51 sampai 4.0, kembalikan string “Dengan pujian”  
Jika tidak, kembalikan string “-“
  - `cetak(): void`
    - Menampilkan nim, nama, alamat, ipk dan predikat dalam susunan vertical

Contoh output:

```
Nama    = Dilan
Alamat = Bandung
NIM    = 1
IPK   = 3.51
Predikat = Dengan Pujian
```

## BAB 5 INTERAKSI ANTAR OBJEK

### 5.1 Keterkaitan antar Class

Kita perlu mengetahui keterkaitan antar class untuk merancang class. Keterkaitan umum antar class yaitu asosiasi, agregrasi, komposisi dan pewarisan. Pada bab 5 ini akan membahas keterkaitan antar class tentang asosiasi, agregrasi, dan komposisi, sedangkan pewarisan akan dibahas pada bab 7.

### 5.2 Asosiasi

Asosiasi adalah hubungan umum yang menggambarkan keterkaitan antara 2 class. Sebagai contoh, seorang penonton yang menonton film adalah keterkaitan class Penonton dan class Film. Anggota rumah produksi, seorang sutradara membuat film adalah keterkaitan class RumahProduksi dan class Film. Asosiasi dapat digambarkan dalam UML class diagram pada gambar 5.1.



Gambar 5.1 UML asosiasi pada class Penonton, Film, dan RumahProduksi

Gambar 5.1 menunjukkan bahwa penonton dapat menonton beberapa film. Anggota rumah produksi (sutradara) dapat membuat beberapa film. Film mungkin memiliki 1 sampai 300 penonton dan film dibuat oleh 1 atau 2 anggota rumah produksi (sutradara).

Sebuah asosiasi digambar oleh garis solid antar class. Label menunjukkan keterkaitan antar class dan segitiga hitam menunjukkan arah hubungan. Pada gambar 5.1 memiliki label menonton dan membuat. Simbol segitiga hitam ► menunjukkan bahwa penonton dapat menonton beberapa film. Setiap class yang terlibat mungkin memiliki nama peran yang menggambarkan peran yang dimainkannya dalam hubungan. Pada gambar 5.1, sutradara adalah peran dari RumahProduksi.

Setiap class yang terlibat dalam hubungan asosiasi dapat menentukan *multiplicity* yang ditempatkan disamping class dekat garis solid. *Multiplicity* berfungsi untuk menentukan berapa banyak objek class yang terlibat dalam hubungan di UML. *Multiplicity* simbol \* berarti objek yang tidak terbatas dan interval m...n berarti objek yang berjumlah diantara

angka m dan n. Pada gambar 5.1, penonton dapat menonton film sebanyak apapun dan setiap film dapat ditonton antara 1 sampai 300 orang dalam satu kali penayangan. Setiap film diproduksi oleh 1 atau 2 anggota rumah produksi, dan sutradara dapat membuat film sebanyak apapun.

Dalam bahasa pemrograman Java, kita dapat menerapkan keterkaitan antar class menggunakan property dan method. Gambar 5.2 menggambarkan implementasi dari gambar 5.1. Keterkaitan penonton menonton film dapat diimplementasikan menggunakan method nontonFilm di class Penonton dan method tambahPenonton di class Film. Keterkaitan rumah produksi membuat film diimplementasikan menggunakan method buatFilm di class RumahProduksi dan method setRumahProduksi di class Film. Class penonton dapat menggunakan listFilm untuk menyimpan film-film apa saja yang ditonton, class RumahProduksi dapat menggunakan listFilm untuk menyimpan film-film yang dibuat, dan class Film dapat menggunakan listPenonton untuk menyimpan para penonton yang menonton dan property rumahProduksi untuk menyimpan informasi pembuat film.

```
public class Penonton{  
    private Film[] listFilm;  
  
    public void nontonFilm(Film f){  
        ...  
    }  
}
```

```
public class RumahProduksi{  
    private Film[] listFilm;  
  
    public void buatFilm(Film f){  
        ...  
    }  
}
```

```
public class Film{  
    private Penonton[] listPenonton;  
    private RumahProduksi[] listRumahProduksi;  
  
    public void tambahPenonton(Penonton p){  
        ...  
    }  
  
    public void setRumahProduksi(RumahProduksi rp){  
        ...  
    }  
}
```

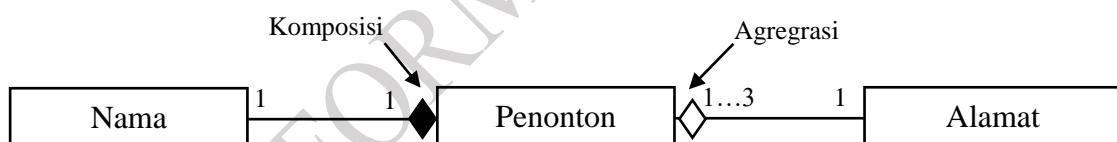
Gambar 5.2 Asosiasi yang diimplementasikan menggunakan property dan method di class

### 5.3 Agregasi dan Komposisi

Agregasi adalah bentuk khusus dari asosiasi yang mewakili keterkaitan antara dua objek. Pemilik objek disebut *aggregating object* dan kelasnya disebut *aggregating class*. Objek subjek disebut *aggregated object*, dan kelasnya disebut *aggregated class*. Kita dapat menyebut agregasi antara dua objek sebagai komposisi jika keberadaan *aggregated object* tergantung pada *aggregating object*. Maka, *aggregated object* tidak dapat tercipta dengan sendirinya.

Contoh "seorang penonton memiliki nama" adalah komposisi hubungan antara class Penonton dan class Nama karena nama tergantung pada penonton, sedangkan "seorang penonton memiliki alamat" adalah hubungan agregasi antara class Penonton dan class Alamat karena alamat bisa ada dengan sendirinya. Komposisi menyiratkan kepemilikan. Satu objek memiliki objek lain. Ketika objek pemilik dihancurkan, objek yang terkait hancur juga.

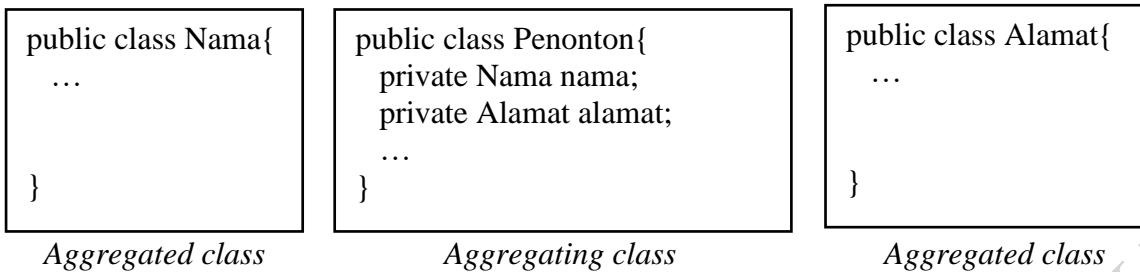
Lihat gambar 5.3 yang berisikan UML, simbol belah ketupan hitam menyatakan bahwa *aggregating class* (Penonton) memiliki hubungan komposisi dengan *aggregated class* (Nama). Sedangkan simbol belah ketupan putih yang melekat pada *aggregating class* (Penonton) memiliki hubungan agregasi dengan *aggregated class* (Alamat).



Gambar 5.3 Contoh UML dengan keterkaitan Agregrasi dan Komposisi

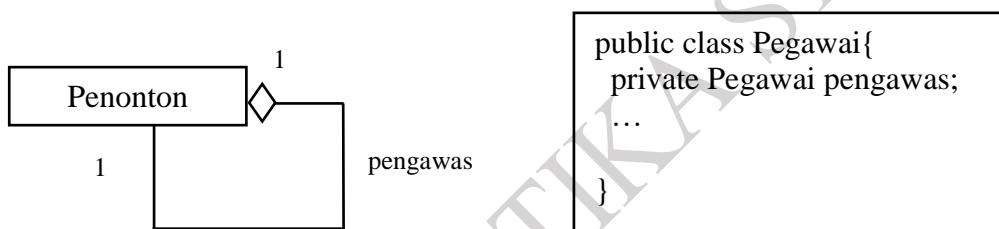
Pada gambar 5.3, setiap penonton hanya memiliki satu *multiplicity* alamat dan masing-masing alamat bisa dibagikan oleh hingga 3 penonton. Setiap penonton memiliki satu nama, dan nama tersebut unik untuk setiap penonton.

Keterkaitan agregasi biasanya direpresentasikan sebagai property dalam *aggregating class*. Contohnya pada gambar 5.4 yang menunjukkan relasi "seorang penonton memiliki nama" dan "seorang penonton memiliki alamat", terlihat dari variabel property nama dan alamat di kelas Siswa.



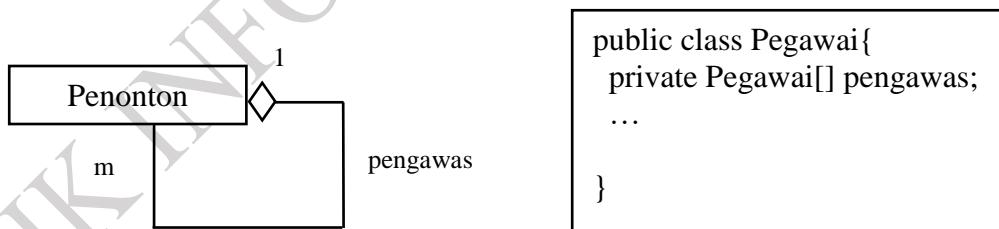
Gambar 5.4 Agregasi dan Komposisi yang diimplementasi menggunakan property

Agregasi bisa muncul di antara objek-objek dari kelas yang sama. Misalnya, seorang pegawai mungkin memiliki pengawas. Gambar 5.5 menjelaskan hubungan "seorang pegawai memiliki pengawas". Seorang pengawas dapat direpresentasikan sebagai property di kelas Pegawai.



Gambar 5.5 Agregasi Seorang Pegawai Memiliki Pengawas

Jika seorang pegawai dapat memiliki beberapa pengawas, maka kita perlu menambahkan array, lihat gambar 5.6. Array tersebut berguna untuk pengawas produksi, pengawas kualitas, dan lain-lain.

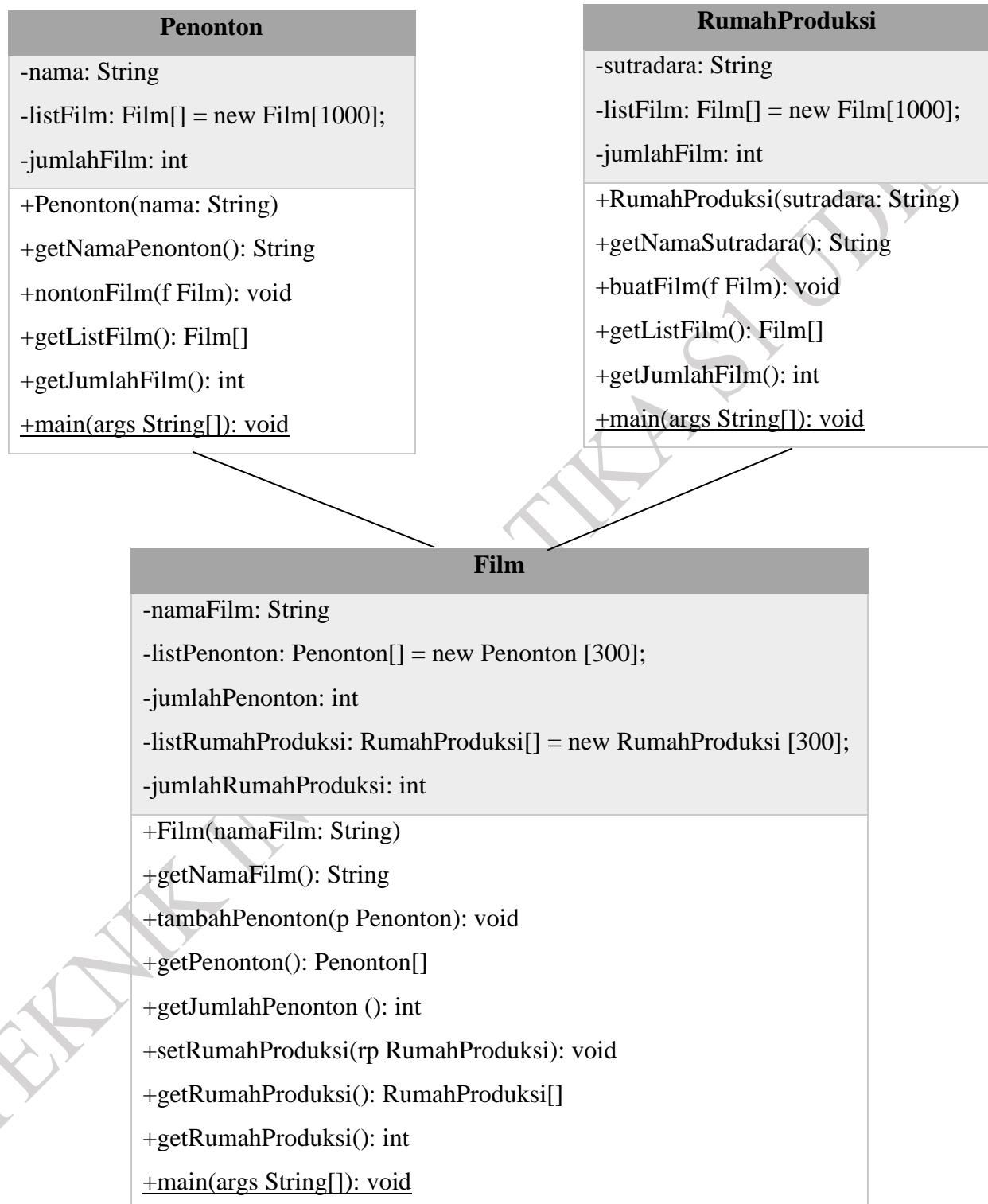


Gambar 5.6 Agregasi Seorang Pegawai Memiliki Banyak Pengawas

## 5.4 Studi Kasus

Agar lebih memahami materi, mari kita implementasi class Penonton, Film, dan RumahProduksi. UML class diagram lengkap terdapat pada gambar 5.7 yang merupakan perluasan dari UML pada gambar 5.2. Kode sumber 5.1 hanya menampilkan class Film secara utuh, sedangkan class Penonton dan class RumahProduksi bisa dilengkapi secara

mandiri sebagai tugas praktikum. Kode sumber yang digunakan menerapkan konsep array of object, kita dapat memahami praktiknya pada bab ini dan akan belajar tentang teorinya pada bab 6.5.



Gambar 5.7 UML class diagram ClassPenonton, Film, dan RumahProduksi Lengkap

**Kode 5.1 Penonton.java**

```
1 public class Penonton {  
2     //property  
3     private String nama;  
4     /*Sebagai tugas praktikum 1, konversi property dibawah ini  
5 menggunakan kode sumber  
6     -listFilm: Film[] = new Film[1000];  
7         //Array untuk menampung Film  
8     -jumlahFilm: int  
9         //Jumlah dari Film yang ditampung (default = 0)  
10    sampai property ini*/  
11  
12     //konstruktor  
13     public Penonton(String nama){  
14         this.nama = nama;  
15     }  
16  
17     //method  
18     public String getNamaPenonton(){  
19         return nama;  
20     }  
21  
22     /*Buat dan isi method-method dibawah ini  
23     +nontonFilm(f Film): void  
24         //Tambah film yang ditonton  
25     +getListFilm(): Film[]  
26         //Kembalikan list film yang ditonton  
27     +getJumlahFilm(): int  
28         //Kembalikan jumlah film yang ditonton  
29     sampai method ini*/  
30  
31     public static void main(String[] args) {
```

```
32     Penonton p1 = new Penonton("Abas");
33     Penonton p2 = new Penonton("Yogi");
34     Penonton p3 = new Penonton("Anisa");
35
36     p1.nontonFilm(new Film("Dilan 1991"));
37     p1.nontonFilm(new Film("Avenger: Endgame"));
38
39     p2.nontonFilm(new Film("Dilan 1991"));
40     p2.nontonFilm(new Film("Avenger: Endgame"));
41
42     p3.nontonFilm(new Film("Dilan 1991"));
43
44     System.out.println("Jumlah film yang ditonton
"+p1.getNamaPenonton()+" adalah "+p1.getJumlahFilm());
45     Film[] f1 = p1.getListFilm();
46     System.out.print("|| ");
47     for(int i=0; i<p1.getJumlahFilm(); i++){
48         System.out.print(f1[i].getNamaFilm()+" || ");
49     }
50
51     System.out.println("\n");
52
53     System.out.println("Jumlah film yang ditonton
"+p2.getNamaPenonton()+" adalah "+p2.getJumlahFilm());
54     Film[] f2 = p2.getListFilm();
55     System.out.print("|| ");
56     for(int i=0; i<p2.getJumlahFilm(); i++){
57         System.out.print(f2[i].getNamaFilm()+" || ");
58     }
59
60     System.out.println("\n");
61
```

```

62     System.out.println("Jumlah film yang ditonton
63         "+p3.getNamaPenonton()+" adalah "+p3.getJumlahFilm());
64         Film[ ] f3 = p3.getListFilm();
65         System.out.print("|| ");
66         for(int i=0; i<p3.getJumlahFilm(); i++){
67             System.out.print(f3[i].getNamaFilm()+" || ");
68         }
69     }

```

**Output:**

Jumlah film yang ditonton Abas adalah 2

|| Dilan 1991 || Avenger: Endgame ||

Jumlah film yang ditonton Yogi adalah 2

|| Dilan 1991 || Avenger: Endgame ||

Jumlah film yang ditonton Anisa adalah 1

|| Dilan 1991 ||

**Kode 5.2 Film.java**

```

1  public class Film {
2      //property
3      private String namaFilm;
4      private Penonton[] listPenonton = new Penonton[300];
5      private int jumlahPenonton;
6      private RumahProduksi[] listRumahProduksi = new RumahProduksi[2];
7      private int jumlahRumahProduksi;
8
9      //konstruktor
10     public Film(String namaFilm){
11         this.namaFilm = namaFilm;
12     }

```

```
13    }
14
15    //method
16    public String getNamaFilm(){
17        return namaFilm;
18    }
19    public void tambahPenonton(Penonton p){
20        listPenonton[jumlahPenonton] = p;
21        jumlahPenonton++;
22    }
23    public Penonton[] getPenonton(){
24        return listPenonton;
25    }
26    public int getJumlahPenonton(){
27        return jumlahPenonton;
28    }
29    public void setRumahProduksi(RumahProduksi rp){
30        listRumahProduksi[jumlahRumahProduksi] = rp;
31        jumlahRumahProduksi++;
32    }
33    public RumahProduksi[] getRumahProduksi(){
34        return listRumahProduksi;
35    }
36    public int getJumlahRumahProduksi(){
37        return jumlahRumahProduksi;
38    }
39    public static void main(String[] args) {
40        Film film1 = new Film("Dilan 1991");
41        Film film2 = new Film("Avenger: Endgame");
42
43        film1.setRumahProduksi(new RumahProduksi("Pidi Baiq"));
44        film1.setRumahProduksi(new RumahProduksi("Fajar Bustomi"));
```

```

45
46     film2.setRumahProduksi(new RumahProduksi("Anthony Russo"));
47     film2.setRumahProduksi(new RumahProduksi("Joe Russo"));
48
49     film1.tambahPenonton(new Penonton("Abas"));
50     film1.tambahPenonton(new Penonton("Yogi"));
51     film1.tambahPenonton(new Penonton("Anisa"));
52
53     film2.tambahPenonton(new Penonton("Abas"));
54     film2.tambahPenonton(new Penonton("Yogi"));
55
56     System.out.println("==> Box Office 2019 ==<");
57     System.out.println("Film 1 - "+film1.getNamaFilm());
58     System.out.print("Sutradara = ");
59     RumahProduksi[] rp1 = film1.getRumahProduksi();
60     for(int i=0; i<film1.getJumlahRumahProduksi(); i++){
61         System.out.print(rp1[i].getNamaSutradara()+" , ");
62     }
63     System.out.println("");
64     System.out.println("Jumlah penonton film
"+film1.getNamaFilm()+" adalah "+film1.getJumlahPenonton());
65     Penonton[] p1 = film1.getPenonton();
66     System.out.print("|| ");
67     for(int i=0; i<film1.getJumlahPenonton(); i++){
68         System.out.print(p1[i].getNamaPenonton()+" || ");
69     }
70     System.out.println("\n\n");
71     System.out.println("Film 2 - "+film2.getNamaFilm());
72     System.out.print("Sutradara = ");
73     RumahProduksi[] rp2 = film2.getRumahProduksi();
74     for(int i=0; i<film2.getJumlahRumahProduksi(); i++){
75         System.out.print(rp2[i].getNamaSutradara()+" , ");

```

```

76    }
77    System.out.println("");
78    System.out.println("Jumlah penonton film
79 "+film2.getNamaFilm()+" adalah "+film2.getJumlahPenonton());
80    Penonton[] p2 = film2.getPenonton();
81    System.out.print("|| ");
82    for(int i=0; i<film2.getJumlahPenonton(); i++){
83        System.out.print(p2[i].getNamaPenonton()+" || ");
84    }
85    System.out.println("");
86 }

```

**Output:**

==== Box Office 2019 ===

Film 1 - Dilan 1991

Sutradara = Pidi Baiq, Fajar Bustomi,

Jumlah penonton film Dilan 1991 adalah 3

|| Abas || Yogi || Anisa ||

Film 2 - Avenger: Endgame

Sutradara = Anthony Russo, Joe Russo,

Jumlah penonton film Avenger: Endgame adalah 2

|| Abas || Yogi ||

**Kode 5.3 RumahProduksi.java**

```

1 public class RumahProduksi{
2     //property
3     private String sutradara;
4     /*Sebagai tugas praktikum 1, konversi property dibawah ini
menggunakan kode sumber
5     -listFilm: Film[] = new Film[1000];
6         //Array untuk menampung Film

```

```

7   -jumlahFilm: int
8       //Jumlah dari Film yang ditampung (default = 0)
9   sampai property ini*/
10
11  //konstruktor
12  public RumahProduksi(String sutradara){
13      this.sutradara = sutradara;
14  }
15
16  //method
17  public String getNamaSutradara(){
18      return sutradara;
19  }
20  /*Buat dan isi method-method dibawah ini
21  +buatFilm(f Film): void
22      //Tambah film yang dibuat
23  +getListFilm(): Film[]
24      //Kembalikan list film yang ditonton
25  +getJumlahFilm(): int
26      //Kembalikan jumlah film yang ditonton
27  sampai method ini*/
28
29  public static void main(String[] args) {
30      RumahProduksi rp1 = new RumahProduksi("Pidi Baiq");
31      RumahProduksi rp2 = new RumahProduksi("Fajar Bustomi");
32      RumahProduksi rp3 = new RumahProduksi("Anthony Russo");
33      RumahProduksi rp4 = new RumahProduksi("Joe Russo");
34
35      rp1.buatFilm(new Film("Dilan 1990"));
36      rp1.buatFilm(new Film("Dilan 1991"));
37      rp2.buatFilm(new Film("Dilan 1990"));
38      rp2.buatFilm(new Film("Dilan 1991"));

```

```

38     rp3.buatFilm(new Film("Avenger: Endgame"));
39     rp4.buatFilm(new Film("Avenger: Endgame"));
40
41     System.out.println("Jumlah film yang dibuat
42 "+rp1.getNamaSutradara()+" adalah "+rp1.getJumlahFilm());
43     Film[] f1 = rp1.getListFilm();
44     System.out.print("|| ");
45     for(int i=0; i<rp1.getJumlahFilm(); i++){
46         System.out.print(f1[i].getNamaFilm()+" || ");
47     }
48 }
49 }
```

#### **Output:**

Jumlah film yang dibuat Pidi Baiq adalah 2

|| Dilan 1990 || Dilan 1991 ||

Kode sumber 5.1 Class Penonton, Film, dan RumahProduksi

## **5.5 Kesimpulan**

1. Keterkaitan umum antar class yaitu asosiasi, agregasi, komposisi dan pewarisan
2. Asosiasi adalah hubungan umum yang menggambarkan keterkaitan antara 2 class.
3. Agregasi adalah bentuk khusus dari asosiasi yang mewakili keterkaitan antara dua objek.
4. Pemilik objek disebut *aggregating object* dan kelasnya disebut *aggregating class*.  
Objek subjek disebut *aggregated object*, dan kelasnya disebut *aggregated class*.
5. Kita bisa menyebut keteraitan komposisi jika keberadaan *aggregated object* tergantung pada *aggregating object*.

## **5.6 Kuis dan Latihan Soal**

1. Jelaskan perbedaan antara keterkaitan asosiasi, agregasi, dan komposisi!
2. Disebut apakah keterkaitan yang menyiratkan kepemilikan?
3. Keterkaitan yang memungkinkan muncul di antara objek-objek dari kelas yang sama adalah ...

## **5.7 Praktikum**

1. Lengkapi kode sumber 5.1 pada class Penonton
2. Lengkapi kode sumber 5.1 pada class RumahProduksi

Lihat pada class Film sebagai contoh acuan!

## BAB 6 ARRAY DAN COLLECTION

### 6.1 Konsep Array

Biasanya seorang programmer membutuhkan suatu cara untuk menyimpan data dengan jumlah data yang besar. Misalnya, setelah programmer berhasil menyimpan sejumlah data yang besar tadi akan digunakan untuk mengurutkan setiap data yang disimpan. Hal ini akan membutuhkan banyak variabel untuk menampung setiap nilai yang ada. Jika data tersebut berjumlah kurang dari 10 mungkin tidak akan menjadi masalah apabila membuat sebanyak 10 variabel. Akan tetapi, jika jumlah data yang disimpan tersebut ratusan atau ribuan atau bahkan lebih, tentu programmer tidak akan mungkin menuliskan variabel sebanyak itu. Jadi bagaimana programmer akan menangani masalah ini?

Dibutuhkan suatu cara yang efisien agar programmer tidak perlu menuliskan banyak variabel untuk menampung data. Beberapa bahasa pemrograman tingkat tinggi (seperti Java) sudah menyediakan struktur data yang disebut dengan istilah Array. Array memungkinkan programmer untuk menyimpan suatu koleksi elemen nilai yang serupa secara terurut dengan tipe data yang sama. Sebagai contoh, programmer membutuhkan penyimpanan nilai-nilai integer sebanyak 6 elemen nilai integer. Maka programmer hanya cukup membuat satu variabel array untuk menampung seluruh koleksi dari setiap elemen nilai integer yang ada. Setiap elemen nilai yang ada didalam Array diakses dengan suatu indeks. Indeks tersebut merepresentasikan urutan data yang ada didalam array. Pada bahasa pemrograman Java indeks dimulai dari nilai 0 (karena di bahasa pemrograman lain indeks kadang dimulai dari nilai 1). Berikut adalah ilustrasi array `ListNilai`:

Tabel 6.1 Ilustrasi array `ListNilai`

Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ListNilai	34	5	23	23	2	12	123	1	28	4	6	76	45	64	66

Pada contoh tersebut kita memiliki variabel array dengan nama `ListNilai` dengan tipe integer. Misal dari `ListNilai` tersebut elemen pertama memiliki nilai 34 dan berada pada indeks ke-0. Kita dapat menyederhanakan pemanggilan nilai pada indeks ke-0 tersebut dengan cara `ListNilai[0]=34`. Sehingga jika dilihat elemen variabel `ListNilai` akan menjadi

`ListNilai[0], ListNilai[1], ... , ListNilai[14]`. Perhatikan bahwa jumlah seluruh elemen array pada variabel `ListNilai` adalah 15, bukan 14. Hal itu dikarenakan indeks array dimulai dari 0 bukan 1.

## 6.2 Deklarasi dan Pembuatan Array

Deklarasi variabel array berbeda dengan deklarasi variabel primitif biasa. Deklarasi suatu variabel dari array tidak mengalokasikan ruang memori secara langsung, namun hanya menyiapkan lokasi penyimpanan data didalam memory sebagai referensi. Jika suatu variabel tidak memuat referensi dari array maka variabel tersebut akan bernilai null. Berikut adalah cara untuk mendeklarasikan variabel array.

```
tipeElement[] arrRefVar;
```

atau,

```
tipeElement arrRefVar[];
```

Suatu `tipeElement` bisa jadi apapun termasuk tipe data primitif atau objek. Pada contoh sintak kedua yaitu `tipeElement arrRefVar[];` menggunakan gaya penulisan sintak pada C/C++, untuk pemograman dengan Java menuliskan sintak array dengan cara tersebut tidak disarankan. Implementasi kode dari deklarasi array dengan nama variabel `ListNilai` dengan referensi array dari elemen bertipe data `int` adalah sebagai berikut:

```
int[] ListNilai;
```

Kita tidak dapat mengisi nilai element array sampai array tersebut selesai diciptakan. Setelah mendeklarasikan array, untuk menciptakan array kita harus menggunakan operator `new` dan melakukan assignment kepada tipe data yang direferensi pada variabel array tersebut. Berikut adalah cara untuk menciptakan array pada variabel `ListNilai` yang sudah dideklarasikan sebelumnya:

```
ListNilai = new int[15];
```

Atau kita dapat secara langsung mendeklarasikan dan menciptakan array, seperti berikut:

```
int[] ListNilai = new int[15];
```

Untuk men-assign suatu nilai didalam array tentu kita harus melakukan dengan tipe data yang sesuai dengan yang di referensi oleh array tersebut. Pada variabel `ListNilai` hanya memperbolehkan nilai yang bertipe integer saja. Sintak untuk men-assign suatu nilai kedalam array adalah sebagai berikut:

```
arrRefVar[indeks] = nilai;
```

Sebagai contoh dari `ListNilai` yang sebelumnya sudah kita buat, adalah sebagai berikut:

```
ListNilai[0] = 34;  
ListNilai[1] = 5;  
ListNilai[2] = 23;  
ListNilai[3] = 23;  
ListNilai[4] = 2;  
ListNilai[5] = 12;  
ListNilai[6] = 123;  
ListNilai[7] = 1;  
ListNilai[8] = 28;  
ListNilai[9] = 4;  
ListNilai[10] = 6;  
ListNilai[11] = 76;  
ListNilai[12] = 45;  
ListNilai[13] = 64;  
ListNilai[14] = 66;
```

Pada dasarnya sewaktu array dibuat, programmer mendefinisikan nilai kapasitas dari array tersebut, misalnya pada variabel `ListNilai` memiliki kapasitas array sejumlah 15 elemen. Jumlah atau kapasitas dari elemen tersebut dapat diakses dengan memanggil `length` dari nama variabel array yang dideklarasikan sebelumnya, seperti `ListNilai.length`. Sehingga kita dapat mengisi elemen array ke 14 dengan cara `ListNilai[ListNilai.length-1] = 66;` dari pada `ListNilai[14] = 66;`. Kenapa harus `ListNilai.length-1`? Karena maksimal indeks yang dimiliki `ListNilai` adalah 14 bukan 15 yang merupakan jumlah seluruh element array.

Bagaimana jika programmer tidak men-assign nilai dari variabel array yang sudah dibuat? Secara otomatis, ketika array diciptakan setiap elemen dari array tersebut akan diberi nilai awal. Untuk tipe data numerik nilai awal adalah 0, karakter adalah \u0000, dan false untuk boolean.

Inisialisasi suatu array dapat dilakukan dengan dua cara. Cara pertama memasukkan nilai-nilai elemen array saat array diciptakan secara langsung. Cara kedua adalah memasukkan nilai-nilai setelah array diciptakan. Cara pertama adalah sebagai berikut:

```
double[] arrNilai = {23.2, 1.2, 4.2};
```

Cara yang kedua adalah sebagai berikut:

```
double[] arrNilai = new double[3];
arrNilai[0] = 23.2;
arrNilai[1] = 1.2;
arrNilai[2] = 4.2;
```

Perhatikan bahwa untuk cara pertama tidak bisa dipisahkan tanpa pembuatan array (dengan keyword new). Berikut adalah cara yang SALAH ketika melakukan inisialisasi array di Java:

```
double[] arrNilai;
arrNilai[0] = 23.2;
arrNilai[1] = 1.2;
arrNilai[2] = 4.2;
```

Cara kedua memberikan leluasa kepada programmer untuk dapat menginisialisasi setiap elemen array. Namun demikian, jika jumlah kapasitas array yang diciptakan besar, maka tidak mungkin akan dilakukan di isi nilai satu per satu. Maka dari itu, biasanya programmer akan menginisialisasi dengan bantuan perulangan. Berikut adalah Kode 7.1 merupakan contoh mengisi array dan men-outputkan array dengan mekanisme perulangan.

#### Kode 6.1 ArrayInputOutputDemo.java

```
1 public class ArrayInputOutputDemo {
2     /* Main Method */
3     public static void main(String[] args){
4         // deklarasi dan ciptakan array
5         double[] arrNilai = new arrNilai[10];
6         // inputkan nilai pada setiap elemen array
7         java.util.Scanner input = new java.util.Scanner(System.in);
8         System.out.print("Enter " + arrNilai.length + " values: ");
9         for (int i = 0; i < arrNilai.length; i++)
10    }
```

```

11         arrNilai[i] = input.nextDouble();
12         // outputkan setiap elemen array yang sudah diinputkan
13         for (int i = 0; i < arrNilai.length; i++) {
14             System.out.print(arrNilai[i] + " ");
15         }
16     }
17 }
```

Kode 7.1 dapat dimodifikasi jika user tidak menginginkan untuk menginputkan nilai secara manual melainkan dengan nilai random dari 0 sampai 10 misalnya. Maka mekanisme input nilai pada setiap elemen array dapat diganti seperti berikut:

```

for (int i = 0; i < arrNilai.length; i++) {
    arrNilai [i] = Math.random() * 10;
}
```

Java juga memiliki mekanisme perulangan khusus yang disebut dengan foreach. Programmer dapat menggunakan perulangan ini untuk menelusuri array tanpa harus mengakses indeksnya. Secara umum sintaks untuk membuat perulangan foreach ini adalah sebagai berikut:

```

for(tipedataElemen elemen: variabelreferensiarray) {
    //proses setiap elemen array
}
```

Kita akan mencoba untuk memodifikasi bagian output dari Kode 7.1 yang tadinya dengan perulangan for menjadi perulangan foreach. Berikut adalah kode output yang sudah dirubah:

```

for(double e: arrNilai) {
    System.out.print(e+" ");
}
```

### 6.3 Mengolah data dengan Array

Array berisi data-data yang mungkin akan diolah sedemikian rupa sehingga dapat memberikan output keluaran seperti yang diinginkan. Array juga dapat di salin / copy dengan cara membuat dua buah variabel dengan tipe data array dan memanfaatkan operator ‘=’. Pastikan bahwa kedua variabel array tersebut bertipe sama.

```
arr1 = arr2;
```

Berikut adalah contoh implementasi menyalin nilai setiap member array dari suatu variabel ke variabel lain:

```
int[] arrSumber = {1,2,3,4,5};  
int[] arrTarget = new int[arrSumber.length];  
for(int i=0;i<arrSumber.length;i++)  
{  
    arrTarget[i] = arrSumber[i];  
}
```

Cara yang lain adalah dengan menggunakan `System.arraycopy`. `System.arraycopy` memiliki empat buah parameter yaitu variabel sumber yang akan di salin, posisi awal indeks dari variabel sumber, variabel target, posisi awal indeks dari variabel target, dan banyaknya member array yang akan di salin. Berikut adalah implementasi kodennya:

```
System.arraycopy(arrSumber, 0, arrTarget, 0, arrSumber.length);
```

Array juga dapat digunakan sebagai referensi didalam parameter suatu method dan mengembalikan nilai dari suatu method. Misalnya kita akan membuat method untuk menyalin array dengan versi sendiri:

```
public static int[] copyMyArray(int[] arrSumber)  
{  
    int[] arrTarget = new int[arrSumber.length];  
    for(int i=0;i<arrSumber.length;i++)  
    {  
        arrTarget[i] = arrSumber[i];  
    }  
    return arrTarget;  
}
```

Untuk memanggil method tersebut dapat dilakukan didalam method main seperti berikut:

```
int[] arrS = {23,42,2,1};  
int[] arrT = copyMyArray(arrS);  
for(int i=0;i<arrT.length;i++)  
    System.out.println(arrT[i]);
```

Java mendukung banyaknya argumen variabel dengan tipe data yang sama yang diteruskan kedalam suatu method yang dapat dianggap sebagai array. Hal tersebut dapat juga disebut dengan list dengan argument yang panjang. Kode 7.2 merupakan contoh implementasi list dengan argumen yang panjang yang digunakan pada pemanggilan dan parameter suatu method.

#### Kode 6.2 ArgumentVariabelDemo.java

```
1 public class ArgumentVariabelDemo {  
2     public static void main(String[] args) {  
3         cetakMaks(34, 3, 3, 2, 56.5);  
4         cetakMaks(new double[]{1, 2, 3});  
5     }  
6     public static void cetakMaks(double... numbers) {  
7         if (numbers.length == 0) {  
8             System.out.println("tidak ada argumen.");  
9             return;  
10        }  
11        double result = numbers[0];  
12        for (int i = 1; i < numbers.length; i++)  
13            if (numbers[i] > result)  
14                result = numbers[i];  
15        System.out.println("Nilai maksimal: " + result);  
16    }  
17}
```

Programmer juga dapat mencari element didalam array. Pada dasarnya array menyimpan data-data yang mungkin akan dicari oleh user. Dalam pencarian data didalam array dapat menggunakan algoritma pencarian seperti: pencarian linier/sekuelas, pencarian biner, dan lain sebagainya. Berikut adalah contoh method yang berisi algoritma pencarian linier didalam array.

```

public static int linierSearch(int[] arr, int yangdicari)
{
    for(int i=0;i<arr.length;i++)
        if(yangdicari == arr[i])
            return i;
    return -1;
}

```

Elemen array dapat juga diurutkan dengan suatu algoritma pengurutan. Seperti halnya algoritma pencarian, banyak sekali algoritma pengurutan yang dapat digunakan untuk mengurutkan setiap elemen array, seperti: pengurutan sekuensial, buble sort, merge sort, dan lain sebagainya. Berikut adalah contoh method dengan pengurutan sekuensial:

```

public static void selectionSort(double[] list) {
    for (int i = 0; i < list.length - 1; i++) {
        // cari nilai minimal dari list[i..list.length-1]
        double min = list[i];
        int minIndex = i;

        for (int j = i + 1; j < list.length; j++) {
            if (min > list[j]) {
                min = list[j];
                minIndex = j;
            }
        }

        // tukar list[i] dengan list[minIndex]
        if (minIndex != i) {
            list[minIndex] = list[i];
            list[i] = min;
        }
    }
}

```

Jika programmer tidak ingin susah-susah untuk membuat method pencarian atau pengurutan sendiri, Java menyediakan class **Arrays**. Class **Arrays** memungkinkan kita untuk memanggil

beberapa method seperti pencarian atau pengurutan. Untuk memanfaatkan method-method atau operasi yang ada didalam class `Arrays` harus di tambahkan terlebih dahulu `import java.util.Arrays;`. Berikut adalah contoh operasi sorting dan searching dengan class `Arrays`:

```
double[] num = {1.2, 31, 2.65, 10};  
Arrays.sort(num);  
Arrays.binarySearch(num, 1.2);
```

## 6.4 Array Multidimensi

Biasanya data secara fisik direpresentasikan dengan bentuk tabel. Sedangkan yang Array hanya berisi sebaris atau secarik data. Java memungkinkan untuk merepresentasikan data Array seperti halnya suatu tabel yaitu dengan Array 2D (dimensi). Misalnya ada tabel seperti berikut:

Tabel 6.2 Tabel merepresentasikan Array 2D

Nama	Nilai Matematika	Nilai Fisika
Hanum	92	80
Eko	75.6	98.4
Heri	67	70

Dari data nilai tersebut, programmer dapat membuat array 2D-nya dengan cara seperti berikut:

```
double[][] nilai = {  
    {92, 80},  
    {75.6, 98.4},  
    {67, 70}};
```

Perhatikan bahwa sintak `[][]` merupakan deklarasi untuk array 2D. Kode sebelumnya dapat dibuat seperti berikut:

```
double[][] nilai = new double [3][2];  
nilai[0][0] = 92; nilai[0][1] = 80; nilai[1][0] = 75.6;
```

```
nilai[1][1] = 98.4; nilai[2][0] = 67; nilai[2][1] = 70;
```

Array 2D seringkali juga dapat digunakan untuk pengganti “matriks”. Untuk dapat menggunakan array 2D biasanya programmer tetap memanfaatkan perulangan. Berikut adalah kode untuk deklarasi dan inisialisasi array 2D dengan diberikan nilai random dan kemudian dicetak pada console output:

```
//deklarasi  
int[][] matriks = new int[10][10];  
//inisialisasi dan pemberian nilai random 0-99  
for(int baris=0;baris<matriks.length;baris++)  
    for(int kolom=0;kolom<matriks[baris].length;kolom++)  
        matriks[baris][kolom] = (int)(Math.random() * 100);  
  
//cetak pada console output  
for(int baris=0;baris<matriks.length;baris++) {  
    for(int kolom=0;kolom<matriks[baris].length;kolom++) {  
        System.out.print(matriks[baris][kolom]+” ”);  
    }  
    System.out.println();  
}
```

## 6.5 Array dari Objek

Kita juga dapat membuat array dari objek. Sebagai contoh, sebelumnya terdapat class MahasiswaIndividu. Kemudian dari class MahasiswaIndividu tersebut akan dibuat array dari MahasiswaIndividu dan akan digunakan untuk mengakses member class MahasiswaIndividu. Kode 7.3 merupakan class MahasiswaIndividu dan Kode 7.4 merupakan class TestMahasiswaIndividu yang mengimplementasikan array dari objek MahasiswaIndividu.

<b>Kode 6.3 MahasiswaIndividu.java</b>
<pre>1 public class MahasiswaIndividu 2 { 3     private String nim; 4     public void setNim(String nim) 5     { 6         this.nim = nim;</pre>

```
7     }
8     public String getNim()
9     {
10        return this.nim;
11    }
12 }
```

#### Kode 6.4 TestMahasiswaIndividu.java

```
1 public class TestMahasiswaIndividu
2 {
3     public static void main(String[] args)
4     {
5         //deklarasi list dari objek
6         MahasiswaIndividu[] listMhs = new MahasiswaIndividu[10];
7
8         //instance setiap member list
9         for(int i=0;i<10;i++)
10            listMhs[i] = new MahasiswaIndividu();
11
12         //akses setter getter method dari setiap member
13         for(int i=0;i<10;i++)
14            listMhs[i].setNim("A11.2020."+ (i+1));
15
16         for(int i=0;i<10;i++)
17            System.out.println(listMhs[i].getNim());
18     }
19 }
```

Kode 7.4 akan menghasilkan output sebagai berikut:

A11.2020.1

A11.2020.2

A11.2020.3

A11.2020.4

A11.2020.5

A11.2020.6

A11.2020.7

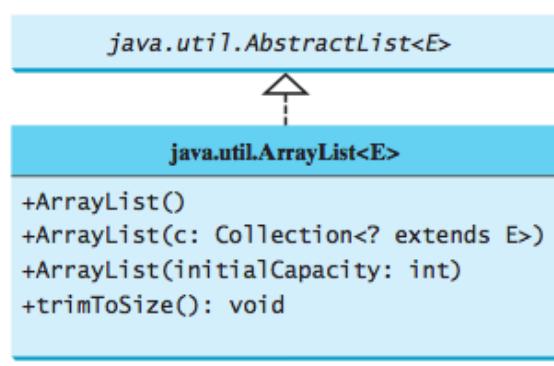
A11.2020.8

A11.2020.9

A11.2020.10

## 6.6 Array Dinamis (List Collection)

Berbeda dengan array yang sebelumnya dijelaskan, array dinamis merupakan bentuk array yang memungkinkan programmer untuk dapat membuat list dengan mengatur setiap member list sesuka hati. Di Java terdapat banyak cara untuk membuat array dinamis tanpa harus membuat array dinamis manual, seperti, list, linked list, queue, ataupun stack. Terdapat class ArrayList di Java. ArrayList merupakan kelas turunan dari List yang memiliki ukuran list yang bisa bertambah (dinamis/growable). Class ini salah satu bagian dari Generic Programming di Java. Generic Programming adalah suatu teknik pemrograman dari pembentukan variable atau method atau class yang dapat digunakan untuk semua jenis tipe data. ArrayList termasuk dalam java framework collection.



Gambar 6.1 Class Diagram ArrayList

Didalam ArrayList terdapat beberapa method yang seringkali digunakan. Berikut adalah beberapa method tersebut:

- add(obj) → untuk menambahkan member list
- remove(obj) → untuk menghapus/menghilangkan member list
- get(index) → mendapatkan akses memberlist ke-index

Kode 7.5 adalah Kode TestMahasiswaIndividu yang isinya telah dirubah menggunakan ArrayList.

#### **Kode 6.5 TestMahasiswaIndividu.java**

```

1 import java.util.ArrayList;
2 public class TestMahasiswaIndividu
3 {
4     public static void main(String[] args)
5     {
6         //deklarasi list dari objek
7         ArrayList<MahasiswaIndividu> listMhs = new
8         ArrayList<MahasiswaIndividu>();
9
10        //instance setiap member list dengan menambahkan kedalam
11        list
12        for(int i=0;i<10;i++)
13            listMhs.add(new MahasiswaIndividu());
14
15        //akses setter getter method dari setiap member
16        for(int i=0;i<10;i++)
17            listMhs.get(i).setNim("A11.2020."+ (i+1));
18
19        for(int i=0;i<10;i++)
20            System.out.println(listMhs.get(i).getNim());
}

```

Selain ArrayList juga terdapat Vector. Turunan dan implementasi yang sama dengan ArrayList. Setiap vektor mencoba untuk mengoptimalkan manajemen penyimpanan dengan mempertahankan kapasitasnya. Termasuk juga kedalam Java Collection. Lebih banyak Method dari Vector jika dibandingkan dengan ArrayList. Vector tensinkronisasi sedangkan ArrayList tidak. Tensinkronisasi maksudnya "Thread-Safe" dari semua keuntungan method yang sedang disinkronkan (melalui kata kunci synchronized). Berikut adalah method-method yang ada pada Vector:

- addElement(element) → Menambah element
- capacity() → Mengembalikan kapasitas vektor
- clone() → Mengembalikan objek yang di copy dari vector
- contains(element) → mencari element vector
- copyInto(element[]) → copy element pada array tertentu
- elementAt(index) → ambil elemen dari index yang ditentukan
- insertElementAt(element , index) → tambahkan element dengan indexnya
- isEmpty() → cek apakah vector Kosong atau tidak
- remove(index) → hapus element berdasarkan index
- size() → ukuran vektor
- set(index, element) → isiakan elemen vector pada suatu index yang diingkan

Kode 7.6 merupakan implementasi dari penggunaan Vector.

<b>Kode 6.6 TestVector.java</b>
---------------------------------

1	import java.util.Vector;
2	public class TestVector
3	{
4	public static void main(String[] args)
5	{
6	// initial size is 3, increment is 2
7	Vector v = new Vector(3, 2);

```

8      System.out.println("Initial size: " + v.size());
9      System.out.println("Initial capacity: " + v.capacity());
10
11     v.addElement(new Integer(1));
12     v.addElement(new Integer(2));
13     v.addElement(new Integer(3));
14     v.addElement(new Integer(4));
15
16     System.out.println("Capacity after four additions: " +
17     v.capacity());
18
19     v.addElement(new Double(5.45));
20
21     System.out.println("Current capacity: " + v.capacity());
22
23     v.addElement(new Double(6.08));
24     v.addElement(new Integer(7));
25
26     System.out.println("Current capacity: " + v.capacity());
27
28     v.addElement(new Float(9.4));
29     v.addElement(new Integer(10));
30
31     System.out.println("Current capacity: " + v.capacity());
32     v.addElement(new Integer(11));
33     v.addElement(new Integer(12));
34
35     System.out.println("First element: " +
36     (Integer)v.firstElement());
37
38     System.out.println("Last element: " +
39     (Integer)v.lastElement());
40
41
42     if(v.contains(new Integer(3)))
43
44         System.out.println("Vector contains 3.");
45
46     }
47 }
```

## **6.7 Kesimpulan**

Array sangat membantu programmer untuk membuat list dari serangkaian data. Terdapat dua jenis array yaitu array statis dan dinamis. Array Statis merupakan array yang sudah memiliki ukuran yang ditentukan dari awal. Sedangkan array dinamis merupakan array yang dapat memiliki ukuran yang dapat disesuaikan (*growable*). Pada dasarnya struktur data akan lebih teratur dan rapi jika menggunakan array. Pada java sudah disediakan beberapa objek array dinamis seperti ArrayList dan Vector.

## **6.8 Kuis dan Latihan Soal**

1. Jelaskan apa yang dimaksud dengan array?
2. Apa perbedaan array statis dan dinamis?
3. Apakah bisa membuat list dari objek dengan Vector? Jika bisa buatkan contoh programnya!

## **6.9 Praktikum**

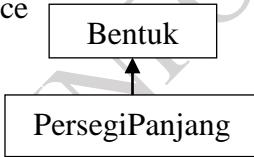
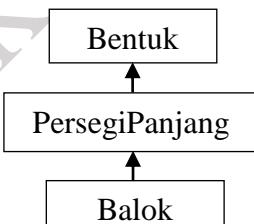
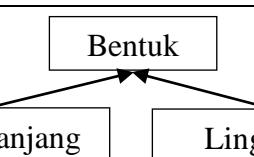
Buat class Mobil dan kemudian buat implementasi list dari Mobil dengan menggunakan ArrayList!

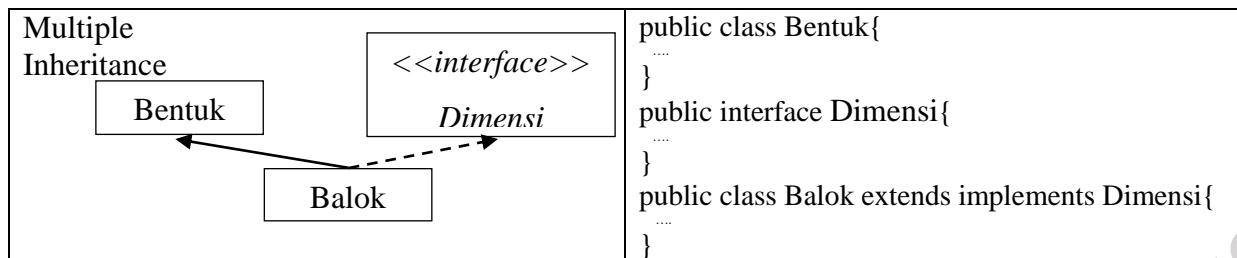
# BAB 7 PEWARISAN

## 7.1 Konsep Pewarisan

Pewarisan/ inheritance memungkinkan kita untuk mendefinisikan class baru dari class yang sudah ada. Konsep ini sangat penting dan bermanfaat untuk menghindari redundansi, membuat sistem mudah dipahami dan mudah dirawat. Misalnya terdapat class PersegiPanjang, Persegi, Lingkaran, dan Segitiga. Class-class tersebut memiliki banyak fitur/ method umum yang sama seperti luas, keliling, volume dan lain-lain. Agar tidak membuat method umum secara berulang-ulang, kita perlu menggunakan konsep pewarisan. Meletakkan method umum ke dalam superclass dan method tersebut dapat digunakan oleh subclass (class PersegiPanjang, Persegi, Lingkaran, dan Segitiga). Pewarisan ini disebut hierarchical inheritance (1 superclass dan banyak subclass), terdapat jenis pewarisan yang lain seperti single inheritance (1 superclass dan 1 subclass), multi level inheritance (1 superclass, banyak subclass, dan banyak subclass lagi), dan multiple inheritance (2 superclass atau lebih dan 1 subclass), jenis ini bisa dilakukan dengan interface. Jenis-jenis pewarisan dapat dilihat pada tabel 7.1.

Tabel 7.1. Jenis-jenis pewarisan

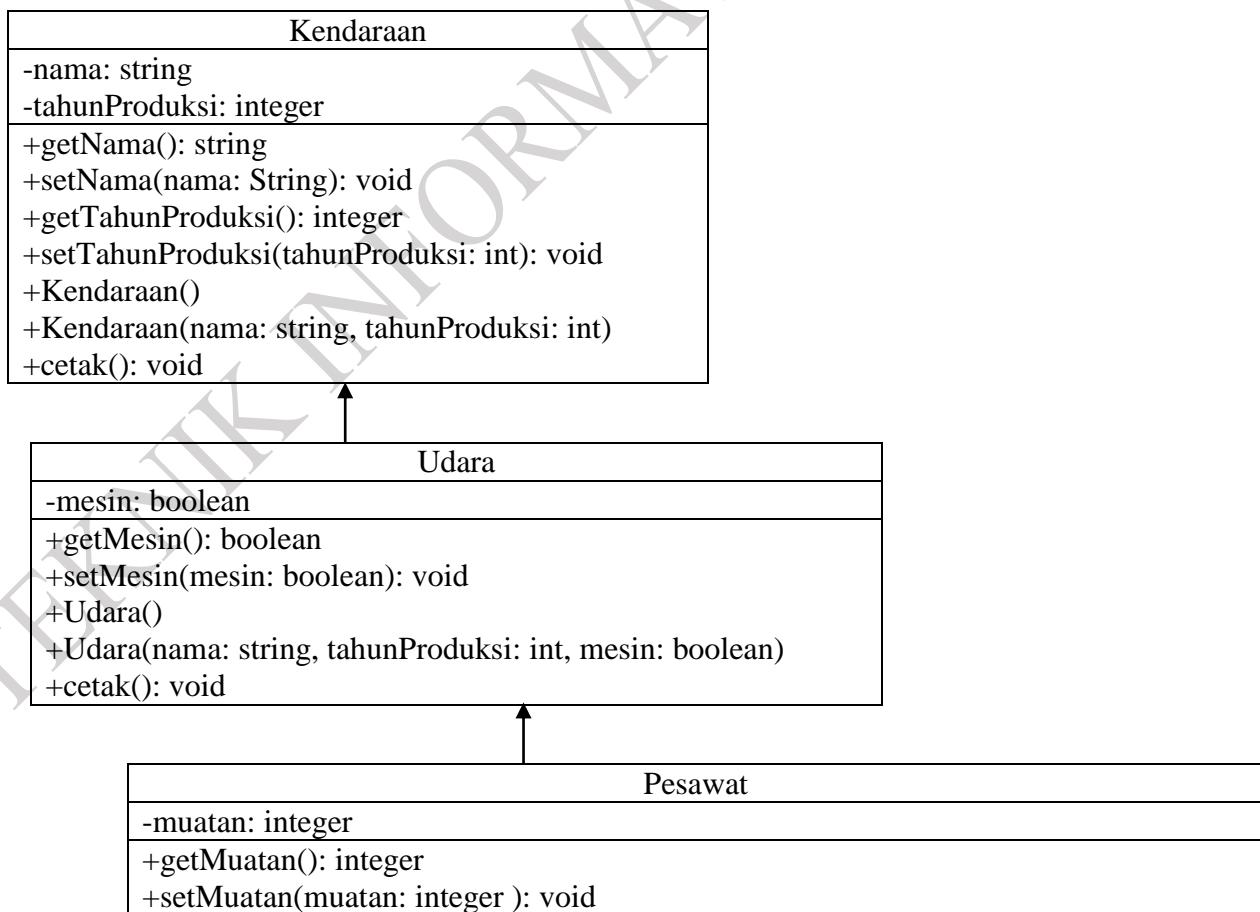
Struktur	Contoh Kode Sumber
Single Inheritance 	public class Bentuk{ ... } public class PersegiPanjang extends Bentuk{ ... }
Multi Level Inheritance 	public class Bentuk{ ... } public class PersegiPanjang extends Bentuk{ ... } public class Balok extends PersegiPanjang{ ... }
Hierarchical Inheritance 	public class Bentuk{ ... } public class PersegiPanjang extends Bentuk{ ... } public class Lingkaran extends Bentuk{ ... }



## 7.2 Superclasses dan Subclasses

Superclass dapat juga disebut dengan parent class atau base class, dan subclass disebut dengan child class, extended class, atau derived class. Superclass berisi method dan property umum. Misal kita akan membuat superclass Bentuk, dengan property warna dan isi. Sedangkan kita memiliki subclass PersegiPanjang, Persegi, Lingkaran, dan Segitiga. Property yang dimiliki superclass Bentuk dapat digunakan pula oleh masing-masing subclassnya. Karena kita sudah memiliki class PersegiPanjang dan Lingkaran pada bab-bab sebelumnya, kita akan mengkonversi mandiri jadi class pewarisan di bagian praktikum.

Kali ini, mari kita menggunakan contoh lain dengan jenis multi level inheritance. Class Kendaraan sebagai superclass, class Udara sebagai subclass, dan class Pesawat sebagai subclass lainnya. Berikut gambar 7.1 yang berisi tentang UML Pewarisan Kendaraan.



```

+Pesawat ()
+Pesawat (nama: string, tahunProduksi: int, mesin: boolean, muatan: int)
+reset(nama: string, tahunProduksi; int, mesin: boolean, muatan: int): void
+kategori(muatan: int): String
+cetak(): void

```

Gambar 7.1. UML Pewarisan Kendaraan

Class Kendaraan memiliki property dan method umum, seperti contoh property nama dan tahunProduksi, serta method-method pendukung. Karena class Kendaraan merupakan superclass, maka semua property dan method tersebut dapat digunakan pada class Udara. Class Udara juga merupakan superclass dari class Pesawat, jadi class Pesawat bisa mengakses semua property dan method milik class Udara maupun class Kendaraan. Gunakan katakunci extends untuk menunjukkan bahwa kelas tersebut merupakan pewarisan.

```

Subclass   Superclass
    ↘       ↙
public class Udara extends Kendaraan{ }

```

Berikut kode sumber 7.1 yang menjelaskan pewarisan kendaraan berdasarkan UML gambar 7.1.

#### Kode 7.1 Kendaraan.java

```

1  public class Kendaraan {
2
3      private String nama;
4
5      private int tahunProduksi;
6
7
8      public String getNama(){
9          return nama;
10     }
11
12     public void setNama(String nama){
13         this.nama = nama;
14     }
15
16     public int getTahunProduksi(){
17         return tahunProduksi;
18     }
19
20     public void setTahunProduksi(int tahunProduksi){
21         this.tahunProduksi = tahunProduksi;
22     }
23
24 }
```

```

16    }
17    public Kendaraan(){ //overloading method, ingat.. konstruktor juga
18      merupakan methhod
19    }
20    public Kendaraan(String nama, int tahunProduksi){ //overloading
21      method
22        setNama(nama);
23        setTahunProduksi(tahunProduksi);
24      }
25    public void cetak(){//overriding method
26      System.out.println("Nama\t\t= "+nama);
27      System.out.println("Tahun Produksi = "+tahunProduksi);
28    }

```

### Kode 7.2 Udara.java

```

1  public class Udara extends Kendaraan{
2    private boolean mesin;
3
4    public boolean getMesin(){
5      return mesin;
6    }
7    public void setMesin(boolean mesin){
8      this.mesin = mesin;
9    }
10   public Udara(){//overloading method
11
12   }
13   public Udara(String nama, int tahunProduksi, boolean mesin){
14     //overloading method

```

```

15     super(nama, tahunProduksi); //memanggil konstruktor Kendaraan
16     setMesin(mesin);
17 }
18 public void cetak(){ //overriding method
19     super.cetak(); //memanggil method cetak dari Kendaraan
20     System.out.print("Mesin\t\t= ");
21     if(mesin==true)
22         System.out.println("Jet");
23     else
24         System.out.println("Baling-baling");
25 }
26 }
```

### Kode 7.3 Pesawat.java

```

1 public class Pesawat extends Udara{
2     private int muatan;
3
4     public int getMuatan(){
5         return muatan;
6     }
7     public void setMuatan(int muatan){
8         this.muatan = muatan;
9     }
10    public Pesawat(){//overloading method
11    }
12
13    public Pesawat(String nama, int tahunProduksi, boolean mesin, int
14    muatan){ //overloading method
15        super(nama, tahunProduksi, mesin); //memanggil konstruktor
16        Udara
17        setMuatan(muatan);
```

```

17    }
18    public void reset(String nama, int tahunProduksi, boolean mesin,
19    int muatan){
20        setNama(nama); //memanggil method setNama dari Kendaraan
21        setTahunProduksi(tahunProduksi); //memanggil method
22        setTahunProduksi dari Kendaraan
23        setMesin(mesin); //memanggil method setMesin dari Udara
24        setMuatan(muatan);
25    }
26    public String kategori(int muatan){
27        if(muatan<=100)
28            return "Mini";
29        else if(muatan<=200)
30            return "Sedang";
31        else
32            return "Besar";
33    }
34    public void cetak(){ //overriding method
35        super.cetak(); //memanggil method cetak dari Udara
36        System.out.println("Muatan \t\t= "+muatan+" orang");
37        System.out.println("Kategori\t= "+kategori(muatan));
38    }
39}

```

#### Kode 7.4 PesawatDemo.java

```

1  public class PesawatDemo {
2      public static int min2(int a, int b){
3          if(a<b)
4              return a;
5          else
6              return b;
7      }
8  }

```

```

8      }
9      public static boolean isNamaSama(Pesawat p1, Pesawat p2){
10
11     if(p1.getNama().toLowerCase().equals(p2.getNama().toLowerCase()))
12         return true;
13     else
14         return false;
15 }
16     public static void main(String[] args){
17         Pesawat[] p = new Pesawat[3];
18         p[0] = new Pesawat();
19         p[1] = new Pesawat("Garuda Boeing 777", 1990, true, 305);
20         p[2] = new Pesawat("Citilink Airbus A320", 1988, true, 150);
21         p[0].reset("Wings Air ATR 72-500", 1988, false, 72);
22
23         p[0].cetak(); System.out.println("");
24         p[1].cetak(); System.out.println("");
25         p[2].cetak(); System.out.println("");
26
27         System.out.print("Nama pesawat pertama dan kedua ");
28         if(isNamaSama(p[0], p[1]))
29             System.out.println("sama");
30         else
31             System.out.println("tidak sama");
32
33         System.out.print("");
34         if(p[0].getMuatan()<min2(p[1].getMuatan(), p[2].getMuatan()))
35             System.out.println("Muatan paling sedikit =
36             "+p[0].getNama());
37             else if(p[1].getMuatan()<min2(p[0].getMuatan(),
p[2].getMuatan()))
38                 System.out.println("Muatan paling sedikit =

```

```

38     "+p[1].getNama());
39         else
40             System.out.println("Muatan paling sedikit =
41             "+p[2].getNama());
42         }
43     }

```

**Output:**

Nama = Wings Air ATR 72-500

Tahun Produksi = 1988

Mesin = Baling-baling

Muatan = 72 orang

Kategori = Mini

Nama = Garuda Boeing 777

Tahun Produksi = 1990

Mesin = Jet

Muatan = 305 orang

Kategori = Besar

Nama = Citilink Airbus A320

Tahun Produksi = 1988

Mesin = Jet

Muatan = 150 orang

Kategori = Sedang

Nama pesawat pertama dan kedua tidak sama

Muatan paling sedikit = Wings Air ATR 72-500

### **7.3 Keyword Super**

Keyword super digunakan untuk merujuk langsung property/ method/ konstruktor pada superclass. Contoh masing-masing penerapan sebagai berikut:

- Keyword super untuk property, contohnya super.nama berarti kita merujuk property nama dari class Kendaraan
- Keyword super untuk method, contohnya pada kode sumber 7.1 Udara.java baris ke-17. Menggunakan super.cetak(); berarti memanggil method cetak dari Kendaraan. Karena pada class Udara juga memiliki method cetak, maka kita wajib menggunakan super jika ingin memanggil method cetak milik superclass
- Keyword super untuk konstruktor, contohnya pada kode sumber 7.1 Udara.java baris ke-13. Menggunakan super(nama, tahunProduksi); berarti memanggil konstruktor Kendaraan yang memiliki parameter inputan nama dan tahunProduksi.

### **7.4 Overriding Method**

Overriding method adalah method yang memiliki nama dan parameter inputan yang sama. Itu berarti overriding menyediakan implementasi baru untuk suatu method dalam subclass. Contohnya pada kode sumber 7.1 untuk method cetak(). Method cetak pada Kendaraan.java terletak di baris ke 23, pada Udara.java terletak di baris 16, dan pada Pesawat.java terletak di baris ke 31. Jika kita ingin memanggil method cetak milik superclass, kita harus menggunakan keyword super. Kalau tidak, maka method cetak merujuk pada diri sendiri.

### **7.5 Overriding vs Overloading**

Overriding method adalah method yang memiliki nama dan parameter inputan yang sama. Sedangkan overloading method adalah method-method yang memiliki nama yang sama namun berbeda-beda parameter inputannya, kita sudah membahasnya tuntas pada BAB 3.3. Untuk melihat perbedaan overriding dan overloading dengan lebih jelas, mari kita lihat tabel 7.1 di bawah ini.

Tabel 7.1. Overriding vs Overloading

Overriding	Overloading
<pre>public class Kendaraan {     public void cetak(){//overriding method         ...     } }  public class Udara extends Kendaraan {     public void cetak(){//overriding method         ...     } }</pre>	<pre>public class Kendaraan {     public Kendaraan(){//overloading method, ingat.. konstruktor juga merupakan method         ...     }      public Kendaraan(String nama, int tahunProduksi){ //overloading method         ...     } }</pre>

## 7.6 Kesimpulan

Pewarisan/ inheritance memungkinkan kita untuk mendefinisikan class baru dari class yang sudah ada. Konsep ini sangat penting dan bermanfaat untuk menghindari redundansi, membuat sistem mudah dipahami dan mudah dirawat. Superclass dapat juga disebut dengan parent class atau base class berisi property atau method umum yang akan diturunkan untuk subclass. Subclass disebut dengan child class, extended class, atau derived class, dapat menggunakan property atau method dari superclass. Syarat untuk melakukan pewarisan, harus menggunakan keyword extends. Keyword super digunakan untuk merujuk langsung property/ method/ konstruktor pada superclass. Overriding method adalah method yang memiliki nama dan parameter inputan yang sama. Itu berarti overriding menyediakan implementasi baru untuk suatu method dalam subclass.

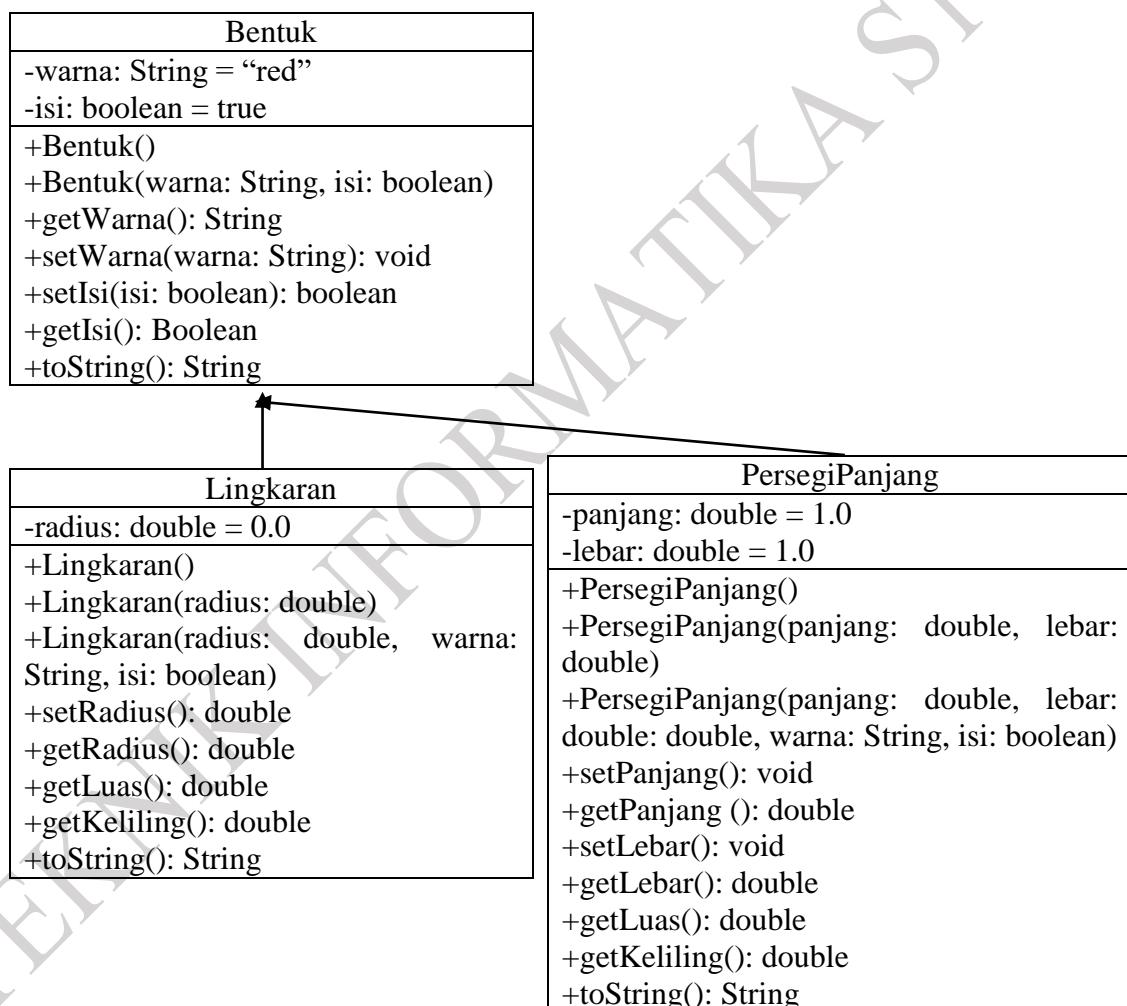
## 7.7 Kuis dan Latihan Soal

1. Apa manfaat dari pewarisan?
2. Apakah benar, kita bisa menggunakan semua property atau method dari superclass?

3. Apa keyword yang digunakan untuk melakukan pewarisan?
4. Apa keyword yang digunakan untuk merujuk langsung property/ method/ konstruktor pada superclass?
5. Jelaskan perbedaan overloading dan overriding!

## 7.8 Praktikum

1. Buatlah implementasi kode sumber dari UML class diagram di bawah ini!
2. Buatlah UML Class Diagram dan kode sumber yang menerapkan konsep pewarisan dengan tema bebas! Minimal 3 class.



# BAB 8 POLYMORPHISM

## 8.1 Konsep Polymorphism

Tiga pilar dasar pada PBO yaitu Enkapsulasi, Inheritance, dan Polymorphism. Kita sudah mempelajari Enkapsulasi dan Inheritance. Sebelum ke polymorphism, ada dua istilah yang harus dimengerti terlebih dahulu:

- Class yang didefinisikan oleh sub-class disebut dengan subtype
- Class yang didefinisikan oleh super-class disebut dengan supertype

Sebagai contoh ObjekGeometri merupakan supertype dan Kotak merupakan subtype sebagaimana yang sudah dipahami seperti pada konsep inheritance. Secara sederhana Polymorphism berarti memiliki arti banyak bentuk. Dalam pemrograman berorientasi objek, polymorphism berarti suatu variable dari supertype dapat mereferensi objek subtype.

Lalu apa yang dimaksud dengan mereferensi? Hubungan inheritance memungkinkan subclass untuk mewarisi fitur dari superclass-nya dengan tambahan fitur baru. Subclass adalah spesialisasi superclass-nya. Setiap contoh dari subclass juga merupakan instance dari superclass, tetapi tidak sebaliknya. Misalnya setiap kotak adalah ObjekGeometri, tetapi tidak setiap ObjekGeometri adalah kotak. Karena itu, kita bisa selalu meneruskan instance dari subclass ke parameter dari jenis superclassnya.

**Kode 8.1 ObjekGeometri.java**

```
1 public class ObjekGeometri
2 {
3     private String color = "red";
4
5     public void setColor(String color)
6     {
7         this.color = color;
8     }
9     public String getColor()
10    {
11        return this.color;
```

```
12    }
13 }
```

#### Kode 8.2 Lingkaran.java

```
1 public class Lingkaran extends ObjekGeometri
2 {
3
4 }
```

#### Kode 8.3 Kotak.java

```
1 public class Kotak extends ObjekGeometri
2 {
3
4 }
```

#### Kode 8.4 TestPolymorphism.java

```
1 public class TestPolymorphism
2 {
3     public static void main(String[] args)
4     {
5         Kotak persegi = new Kotak();
6         persegi.setColor("Blue");
7         ObjekGeometri bentukSem = new ObjekGeometri();
8         bentukSem.setColor("Black");
9
10        displayObject(persegi);
11        displayObject(new Lingkaran());
12        displayObject(bentukSem);
13    }
14    public static void displayObject(ObjekGeometri obj)
15    {
```

```

16     System.out.println(obj.getColor());
17 }
18 }
```

### Output

Blue  
Red  
Black

Pada Kode 8.4 terdapat static method untuk mendisplay warna objek yaitu `displayObject`. Pada mehthod tersebut terdapat parameter masukan `ObjekGeometri` yang berarti super-type dari `Lingkaran` dan `Kotak`. Secara sederhana polymorphism berarti variabel super-type dapat merujuk ke objek sub-type.

## 8.2 Dynamic Binding

Dynamic Binding merupakan perluasan dari konsep polymorphism yang tidak hanya super-type dan sub-type juga, tetapi terkadang super-type juga merupakan sub-type dari super-type yang lain. Sehingga jika akan membuat method dengan dynamic binding berarti method yang dapat diimplementasi pada beberapa kelas sepanjang rantai keturunan. Method yang akan dipanggil secara dinamis dan terikat saat runtime. Kode 8.5 adalah contoh penerapan Dynamic Binding.

### Kode 8.5 DynamicBindingDemo.java

```

1 public class DynamicBindingDemo {
2     public static void main(String[] args) {
3         m( new GraduateStudent());
4         m(new Student());
5         m(new Person());
6         m(new Object());
7     }
8
9     public static void m(Object x) {
10        System.out.println(x.toString());
```

```

11    }
12 }
13 class GraduateStudent extends Student { }
14
15 class Student extends Person {
16     @Override
17     public String toString() {
18         return "Student" ;
19     }
20 }
21 class Person extends Object {
22     @Override
23     public String toString() {
24         return "Person";
25     }
26 }

```

**Output:**

Student  
 Student  
 Person  
[Java.lang.Object@15db9742](#)

### 8.3 Kesimpulan

Salah satu konsep penting dalam pemrograman berorientasi objek adalah polymorphism. Dengan polymorphism objek akan dianggap memiliki banyak bentuk sehingga hal ini akan memudahkan programmer dalam membuat referensi pemanggilan objek dari satu method. Perluasan dari teknik polymorphism adalah Dynamic Binding. Dynamic Binding memungkinkan programmer untuk melakukan polymorphism disepanjang hierarki turunan dari suatu objek-objek dari sistem yang dikembangkan.

## **8.4 Kuis dan Latihan Soal**

1. Jelaskan apa yang dimaksud dengan polymorphism?
2. Mengapa programmer membutuhkan polymorphism?
3. Apakah polymorphism selalu terkait dengan inheritance?

## **8.5 Praktikum**

Buat class Kendaraan, Mobil, Truk, SepedaMotor, Bus, MobilBerat. Perhatikan bahwa Kendaraan merupakan super-class dari Mobil, SepedaMotor, dan MobilBerat. Kemudian, MobilBerat merupakan super-class dari Truk dan Bus. Buat method yang dapat mengimplementasikan Dynamic Binding! Member class (property atau method) pada setiap Class boleh dibuat sesuka hati.

# BAB 9 ABSTRACT CLASS DAN INTERFACE

## 9.1 Abstract Class

Pada konsep pewarisan, super-class biasanya memiliki struktur yang umum dan sub-class lebih spesifik. Seharusnya pada super-class memiliki fitur-fitur umum yang akan diwariskan kepada sub-class. Bisa jadi, programmer membutuhkan super-class yang hanya menyajikan fitur-fitur umum yang harus dijelaskan lebih spesifik didalam sub-class tanpa harus membuat objek dari super-class tersebut. Suatu class yang hanya menyajikan fitur-fitur umum dan tidak dapat dibuat objek disebut dengan class abstract. Kita dapat menyebut class abstract ini sebagai class yang akan digunakan sebagai acuan pada desain sub-class yang akan diwarisi oleh abstract class ini. Suatu abstract class juga dapat memiliki abstract method. Abstract method harus di implementasikan secara konkret didalam sub-class. Misalnya pada Kode 9.1 mencoba untuk membuat abstract class untuk Hewan dan Kode 9.2 membuat sub-class Ayam.

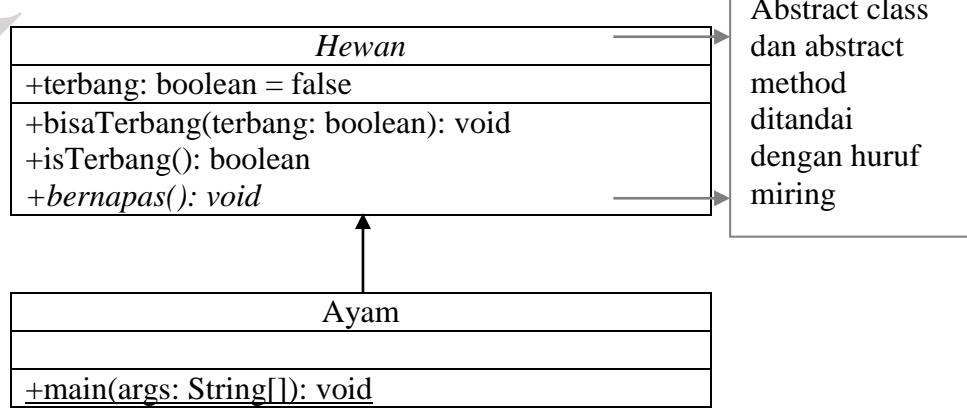
**Kode 9.1 Hewan.java**

```
1 public abstract class Hewan
2 {
3     public boolean terbang = false;
4
5     public void bisaTerbang(boolean terbang)
6     {
7         this.terbang = terbang;
8     }
9
10    public boolean isTerbang()
11    {
12        return this.terbang;
13    }
14
15    public abstract void bernapas();
16 }
```

### Kode 9.2 Ayam.java

```
1 public class Ayam extends Hewan
2 {
3     public void bernapas()
4     {
5         System.out.println("aku bernapas");
6     }
7
8     public static void main(String[] args)
9     {
10        Ayam kutuk = new Ayam();
11        kutuk.bisaTerbang(true);
12        System.out.println("Apakah bisa terbang:");
13        kutuk.bernapas();
14    }
15 }
```

Setiap sub-class yang menerapkan abstract class sebagai super-class harus dengan menambahkan kata kunci “extends” seperti halnya menerapkan pewarisan biasa. Perhatikan bahwa jika didalam abstract class memiliki abstract method, abstract method tersebut harus dijelaskan secara konkret di dalam sub-class seperti pada method `bernapas()`. Kita juga tidak dapat membuat objek `Hewan` karena merupakan abstract class.



Gambar 9.1 UML class diagram abstract class Hewan

Gambar 9.1 menjelaskan tentang UML class diagram abstract `Hewan` yang mewariskan atribut dan methodnya ke class `Ayam`. Untuk menandakan jika class `Hewan` termasuk class abstract, maka harus ditulis dengan huruf miring, begitu pula untuk method dan atribut. Method `bernapas()` pada class `Ayam` tidak perlu ditulis kembali, karena default harus dijelaskan secara konkret dalam artian wajib dikoding di kelas `Ayam`.

## 9.2 Interface

Mirip dengan abstract, interface hanya berfokus pada konstruksi operasi umum pada suatu object. Kita dapat menganggap interface ini sebagai kerangka dari suatu class. Secara struktur interface bukan merupakan class. Sehingga dalam membuat kode program interface tidak lagi menggunakan kata “`class`”. Kode 9.3 merupakan contoh interface `MahlukHidup`.

**Kode 9.3 MahlukHidup.java**

```
1 public interface MahlukHidup  
2 {  
3     public abstract void bernapas();  
4     public boolean isHidup();  
5 }
```

Interface hanya berisi abstract method baik yang menggunakan kata `abstract` maupun tidak. Disini ditandai dengan definisi method yang langsung ditutup dengan tanda titik koma bukan kurung kurawal buka dan tutup. Interface dapat diterapkan kepada class biasa maupun abstract class. Untuk mengimplementasikan interface dibutuhkan tambahan sintak `implements` seperti Kode 9.4 `Hewan.java` yang telah dimodifikasi ini.

**Kode 9.4 Hewan.java**

```
1 public abstract class Hewan implements MahlukHidup  
2 {  
3     public boolean terbang = false;  
4  
5     public void bisaTerbang(boolean terbang)  
6     {  
7         this.terbang = terbang;
```

```

8    }
9
10   public boolean isTerbang()
11  {
12      return this.terbang;
13  }
14
15  public abstract void bernapas();
16
17  public boolean isHidup()
18  {
19      return true;
20  }
21 }
```

Semua method yang sudah didefinisikan didalam interface **WAJIB** di implementasikan didalam class yang sudah menerapkan interface tersebut (kecuali jika diterapkan pada abstract method). Kode 9.5 merupakan class Ayam yang sudah dimodifikasi dengan memanggil method `isHidup()`.

#### Kode 9.5 Ayam.java

```

1  public class Ayam extends Hewan
2  {
3      public void bernapas()
4      {
5          System.out.println("aku bernapas");
6      }
7
8      public static void main(String[] args)
9      {
10         Ayam kutuk = new Ayam();
11         kutuk.bisaTerbang(true);
```

```

12     System.out.println("Apakah bisa terbang:
13         "+kutuk.isTerbang());
14     kutuk.bernapas();
15     System.out.println("Apakah benda hidup:
16         "+kutuk.isHidup());
    }
}

```

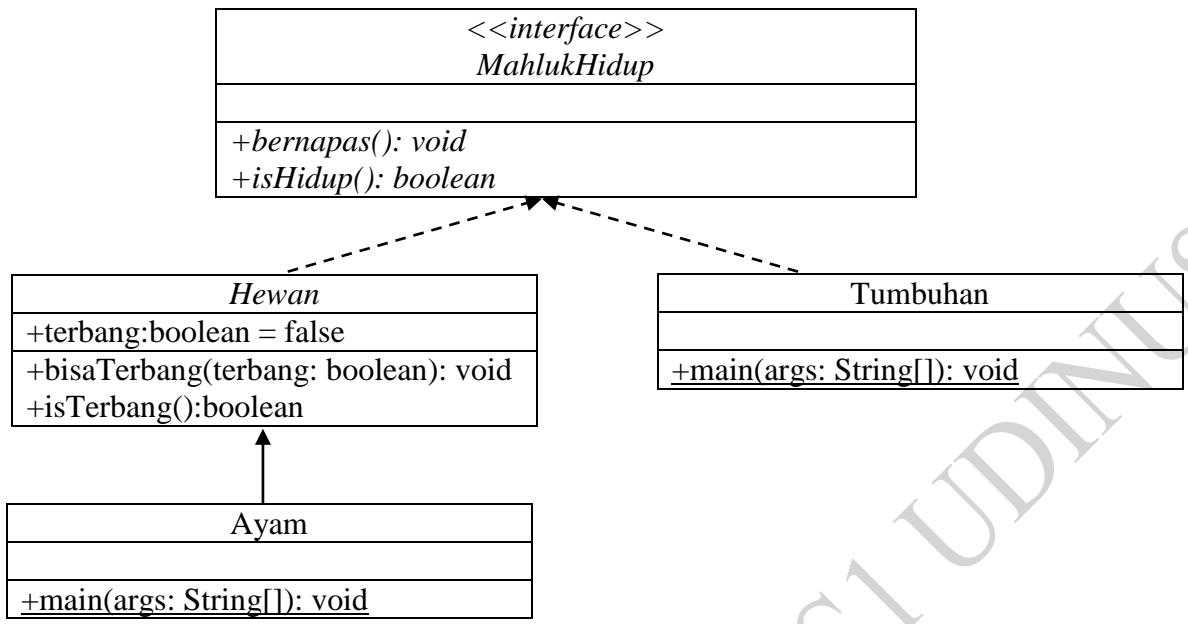
Pada contoh kode sebelumnya interface diterapkan pada abstract method. Bagaimana jika diterapkan pada class biasa. Kode 9.6 merupakan contoh penerapan interface MahlukHidup pada class biasa.

#### Kode 9.6 Tumbuhan.java

```

1 public class Tumbuhan implements MahlukHidup
2 {
3     public void bernapas()
4     {
5         System.out.println("aku tidak bernapas");
6     }
7     public boolean isHidup()
8     {
9         return true;
10    }
11    public static void main(String[] args)
12    {
13        Tumbuhan cemara = new Tumbuhan();
14        cemara.bernapas();
15        System.out.println("Apakah benda hidup:
16            "+cemara.isHidup());
    }
}

```



Gambar 9.2 UML class diagram interface class *MahlukHidup*

Gambar 9.2 menjelaskan tentang UML class diagram interface *MahlukHidup*. Class interface dapat diwariskan dengan keyword implements, dalam UML class diagram digambarkan dengan garis panah putus-putus. Class interface dituliskan dengan huruf miring dan dijelaskan lagi dengan keterangan kata `<<interface>>`. Secara implisit atribut di class interface bersifat public static dan final. Sedangkan untuk method, secara implisit bersifat public dan abstrak sehingga ditulis dengan huruf miring. Method `bernafas()` dan `isHidup()` pada class *Hewan* dan *Tumbuhan* tidak perlu ditulis kembali, karena method tersebut bersifat abstract dan wajib dikoding.

### 9.3 Implementasi Kombinasi Abstract Class dan Interface

Dalam mendesain class diagram, terkadang desainer membuat desain yang mengharuskan programmer untuk menerapkan abstract class dan interface secara langsung pada class biasa. Apakah itu memungkinkan? Tentu saja bisa. Misalnya kita membuat abstract class *Dedaunan* seperti pada Kode 9.7.

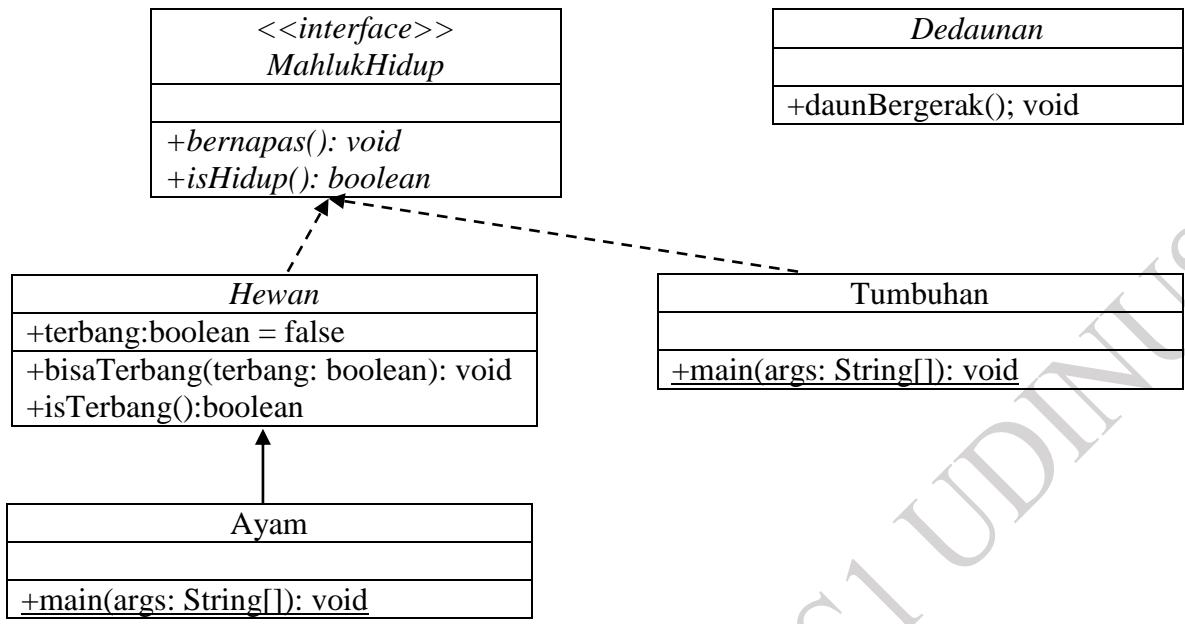
Kode 9.7 Dedaunan.java	
1	public abstract class Dedaunan
2	{
3	public void daunBergerak()

4	{
5	System.out.println("daun melambai-lambai");
6	}
7	}

Kita akan memodifikasi Kode 9.6 tidak hanya menerapkan interface MahlukHidup, tetapi juga menerapkan abstract class Dedaunan. Kode 9.8 merupakan class Tumbuhan yang telah dimodifikasi. UML class diagram dapat dilihat pada GAMBAR 9.

#### Kode 9.8 Tumbuhan.java

1	public class Tumbuhan <b>extends</b> Dedaunan <b>implements</b> MahlukHidup
2	{
3	public void bernapas()
4	{
5	System.out.println("aku tidak bernapas");
6	}
7	
8	public boolean isHidup()
9	{
10	return true;
11	}
12	
13	public static void main(String[] args)
14	{
15	Tumbuhan cemara = new Tumbuhan();
16	cemara.bernapas();
17	System.out.println("Apakah benda hidup:
18	"+cemara.isHidup());
19	cemara.daunBergerak();
20	}



Gambar 9.3 UML class diagram kombinasi abstract interface

## 9.4 Kesimpulan

Seorang desainer sistem, dalam mendesain suatu class terkadang membutuhkan bentuk class yang abstrak atau kerangka dari sebuah class sebelum diimplementasikan. Hal ini digunakan untuk memudahkan programmer dalam menerjemahkan “bahasa bisnis” kedalam kode program. Abstract class merupakan class yang hanya menyajikan fitur-fitur umum dan tidak dapat dibuat objek. Sedangkan interface merupakan bentuk kerangka suatu class yang juga berisi fitur-fitur umum yang berbentuk abstrak.

## 9.5 Kuis dan Latihan Soal

1. Jelaskan apa perbedaan abstrak dan interface?
2. Yang mana dari gambar berikut yang benar?

(a)

```

class A {
    abstract void unfinishedO {
    }
}

```

(b)

```

public class abstract A {
    abstract void unfinishedO;
}

```

(c)

```

class A {
    abstract void unfinishedO;
}

```

(d)

```

abstract class A {
    protected void unfinishedO;
}

```

(e)

```

abstract class A {
    abstract void unfinishedO;
}

```

(f)

```

abstract class A {
    abstract int unfinishedO;
}

```

3. Dapatkah suatu abstract class mengimplementasikan interface?

## **9.6 Praktikum**

Perusahaan “Harpindo Kita” membutuhkan sebuah sistem informasi jual beli mobil. Sistem mampu mendaftarkan biodata karyawan dan pembeli. Pembeli dibagi dalam 2 jenis yaitu: pembeli individu dan pembeli borongan. Mobil yang dijual ada tiga tipe yaitu: Mobil sedan, Mobil minibus, dan mobil bus. Karyawan dapat menambahkan harga setiap mobil yang ada. Pembeli hanya dapat membeli satu tipe mobil saja (berarti dalam hal ini berarti dapat membeli mobil banyak dengan tipe yang sama). Karyawan dapat melihat data pembeli yang sudah membeli mobil. Karyawan mendapatkan keuntungan 10% dari total jumlah harga terbayar oleh setiap pembeli dalam setiap waktu.

### **PETUNJUK:**

1. Minimal terdapat 7 kelas (terserah boleh subclass, superclass, abstract class, atau interface)
2. Method dan property bebas yang sesuai dengan ilustrasi kasus diatas.
3. Keterangan setiap property dan method yang ada pada class diagram dijelaskan beserta potongan kode programnya.
4. Boleh dikembangkan. Dan beri keterangan pengembangan dari kasus diatas.

# BAB 10 EXCEPTION, PEMROGRAMAN GENERIK, DAN DESIGN PATTERN

## 10.1 Exception

Pada dasarnya ada tiga jenis error didalam pemrograman dengan Java, yaitu: syntax error, logic error, dan runtime error. Syntax error merupakan kesalahan yang terjadi saat programmer salah menuliskan kode program. Misalnya, saat programmer kurang memberikan titik koma di akhir dari statement. Biasanya error ini akan muncul saat program di-compile. Logic error berkaitan dengan kesalahan yang dilakukan programmer saat programmer membuat kode sehingga akan menghasilkan output kode yang tidak sesuai dengan yang diinginkan. Kesalahan logic error seringkali disebut sebagai bug/kutu. Untuk menangani bug dilakukan proses debugging. Proses debugging adalah mencari satu persatu baris perbaris kode yang mana yang membuat output yang dihasilkan tidak sesuai.

Runtime error adalah kesalahan program yang terjadi saat program sudah berjalan dimana jika lingkungan pemrograman mendeteksi operasi yang tidak mungkin dapat ditangani. Contoh nyata dari error runtime ini adalah misalnya saat kita menjalankan aplikasi berbasis java yang membutuhkan koneksi internet, tiba-tiba internet mati dan menyebabkan aplikasi tadi berhenti/terminate secara mendadak. Error runtime disebabkan oleh exception. Exception adalah objek yang mewakili error atau kondisi yang mencegah eksekusi dari proses yang tidak berjalan normal.

Ada banyak sekali objek atau tipe exception, misalnya: input mismatch exception, arithmetic exception, null pointer exception, index of out bound exception, illegal argument exception, dll. Berikut adalah contoh kode yang berpotensi menghasilkan input mismatch exception:

```
import java.util.Scanner;
public class TestException
{
    public static void main(String[] args)
    {
        // Buat objek Scanner
        Scanner input = new Scanner(System.in);
        // Kabari user untuk memasukkan angka sisi persegi
```

```

        System.out.print("Masukkan sisi persegi: ");
        double sisi = input.nextDouble();
        System.out.print("Sisi: "+sisi);
    }
}

```

Perhatikan bahwa output dari program tersebut adalah “Masukkan sisi persegi: ”. Jika user memberikan inputan berupa string misalnya “as”. Maka akan muncul error exception Exception in thread "main" java.util.InputMismatchException“. Kita sebagai programmer terkadang tidak memerhatikan user dalam hal inputan sederhana ini. Tetapi hal ini bisa saja terjadi. Maka dari itu perlu penanganan exception.

## 10.2 Mekanisme Try-Catch

Menangani exception pada dasarnya merupakan cara untuk mencegah terjadinya runtime errors. Terdapat mekanisme untuk menangani exception yaitu dengan Try-Catch. Try-Catch memiliki dua blok yaitu blok kode untuk try dan blok kode untuk catch. Pada blok Try programmer akan mencoba menuliskan kode yang mungkin dapat menyebabkan exception. Ketika terjadi exception saat aplikasi berjalan (runtime) blok Catch akan menangkap pesan error exception yang terjadi dan akan mengeksekusi yang ditulis oleh programmer pada blok tersebut. Misalnya ketika user memberikan inputan string padahal semestinya user memberikan input integer. Program tidak akan langsung terminate (berhenti) tetapi akan memunculkan pesan kesalahan input. Pesan kesalahan tersebut berasal dari blok catch. Kode 10.1 merupakan contoh implementasi dari permisalan sebelumnya.

<b>Kode 10.1 AngkaError.java</b>
----------------------------------

1	import java.util.Scanner;
2	public class AngkaError
3	{
4	public static void main(String[] args)
5	{
6	// Buat objek Scanner
7	System.out.print("Masukkan sembarang angka: ");
8	Scanner input = new Scanner(System.in);
9	int angka;

```
10  
11     try{  
12         angka = input.nextInt();  
13         System.out.print("Angka yang dimasukkan: "+angka);  
14     }  
15     catch(Exception e)  
16     {  
17         System.out.print("Inputan salah!");  
18     }  
19 }  
20 }
```

**Output:**

Masukkan sembarang angka: 12

Angka yang dimasukkan: 12

Suatu program yang dapat mendeteksi error dapat dibuat dari instance dari tipe exception yang sesuai dan lemparkan (throw it). Sebagai contoh, terdapat program meneruskan nilai dari suatu method dan argument harus berupa argument non-negative, tetapi malah argument negative. Program dapat dibuat dari instance `IllegalArgumentException` dan kemudian di `throw` seperti berikut:

```
IllegalArgumentException ex = new IllegalArgumentException("Argumen salah!");
```

```
Throw ex;
```

atau

```
throw new IllegalArgumentException("Argumen salah!");
```

Block catch dapat lebih dari satu dan terdapat tambahan blok untuk `finally`. Jika ingin beberapa kode dieksekusi terlepas dari apakah exception tersebut terjadi atau tertangkap kita dapat menggunakan `finally`. Finally juga dapat digunakan tanpa catch. Dari Kode 10.1 kita dapat menambahkan `finally` setelah block catch.

```

finally
{
    System.out.print("Terima kasih.");
}

```

### 10.3 Pemrograman Generik

Secara sederhana, pemrograman generik dalam hal ini Generic Type/ Tipe Generik merupakan class atau interface generik yang memperbolehkan semua jenis tipe data sebagai suatu parameter. Sebelumnya sudah di implementasi pada ArrayList dan Vector dengan simbol “<...>”. Berikut adalah contoh sederhana dari perbandingan class biasa dengan class generik:

Tabel 10.1 Perbandingan class biasa dengan generik

Biasa	Generik
<pre> public class Box {     private int object;     public void set(int object) {         this.object = object;     }     public int get() {         return object;     } } </pre>	<pre> public class Box&lt;T&gt; {     private T t;     public void set(T t) {         this.t = t;     }     public T get() {         return t;     } } </pre>

Pada versi non-generic clas Box jika seandainya instance dari Box adalah Kotak dan Kotak.set(“50”), maka akan menghasilkan error. Error disebabkan hanya boleh diset oleh nilai integer saja. Pada versi generic dapat di set oleh berbagai tipe data, baik primitive maupun tipe data bentukan, dan bahkan array/list. Format class generic:

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

Untuk instance dari class box versi generik, berikut caranya:

```
Box<Integer> integerBox = new Box<Integer>();
```

Meskipun tipe data atau objek yang digenerik-kan dapat diberi nama dengan bebas, beberapa programmer memberikan standar sebagai berikut:

- E - Elemen (sering digunakan di Java Collections Framework)
- K - Kunci

- N – Angka (*number*)
- T – Tipe
- V – Nilai (*value*)
- S,U,V, dll

Timbul pertanyaan, dalam satu class terkadang programmer menginginkan tidak hanya satu variabel atau properti yang akan di-generik-kan. Kode 10.2 memberikan contoh satu class memiliki dua tipe generik.

#### Kode 10.2 OrderedPair.java

```

1  public class OrderedPair<K, V> {
2
3      private K key;
4
5      private String value;
6
7      public OrderedPair(K key, V value) {
8
9          this.key = key;
10         this.value = (String)value;
11     }
12
13     public K getKey() {
14
15         return key;
16     }
17
18     public String getValue()
19     {
20
21         if(this.key == "aaa")
22             return this.value;
23         else
24             return "salah";
25     }
26
27
28     public static void main(String[] args)
29     {
30
31         OrderedPair<String, String> p2 = new OrderedPair<String,
32 String>("aaa", "2");
33
34         System.out.println(p2.getValue());
35     }
36
37
38 }
```

24	}
25	}

**Output:**

2

Contoh lain dari generik tidak hanya pada class tetapi juga dapat diterapkan pada method secara langsung. Berikut adalah contoh method generic untuk menampilkan elemen array:

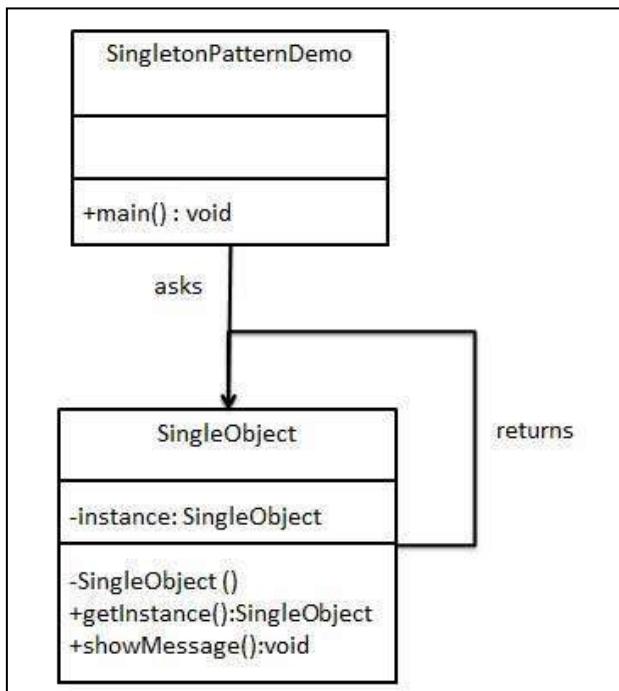
```
public static < E > void printArray( E[] inputArray ) {  
    // Display array elements  
    for(E element : inputArray) {  
        System.out.printf(element+” ”);  
    }  
    System.out.println();  
}
```

## 10.4 Design Pattern

Design pattern merupakan istilah untuk merepresentasikan praktik penggunaan terbaik yang digunakan oleh programmer/pengembang software yang berpengalaman dalam pemrograman berorientasi objek. Menyediakan solusi untuk permasalahan umum dari programmer/pengembang software selama mengembangkan software. Solusi ini diperoleh dari trial & error yang dilakukan oleh programmer/pengembang perangkat lunak yang berpengalaman.

## 10.5 Singleton Pattern

Merupakan Design Pattern paling sederhana yang memungkinkan untuk dapat diprogram dengan Java. Pola ini melibatkan satu Class tunggal yang bertanggung jawab untuk membuat sebuah objek tunggal yang akan tercipta. Class ini menyediakan cara untuk mengakses objeknya yang secara langsung dapat diakses tanpa harus melakukan instance objek dari Class. Class diagram untuk class singleton ini seperti pada Gambar 10.1.



Gambar 10.1 Class Diagram Singleton

Kode 10.3 merupakan contoh implementasi singleton patten untuk class SingleObject.

#### Kode 10.3 SingleObject.java

```

1  public class SingleObject {
2
3      //buat static instance objek ini
4      private static SingleObject instance = new SingleObject();
5
6      //buat konstruktor secara private sehingga
7      //tidak akan bisa di instance dengan cara biasa
8      private SingleObject(){}
9
10     //dapatkan instance
11     public static SingleObject getInstance(){
12         return instance;
13     }
14
15     //tampilkan pesan
16     public void showMessage(){

```

```
17     System.out.println("Hello World!");
18 }
19
20 public static void main(String[] args) {
21     SingleObject object = SingleObject.getInstance();
22     object.showMessage();
23 }
24 }
```

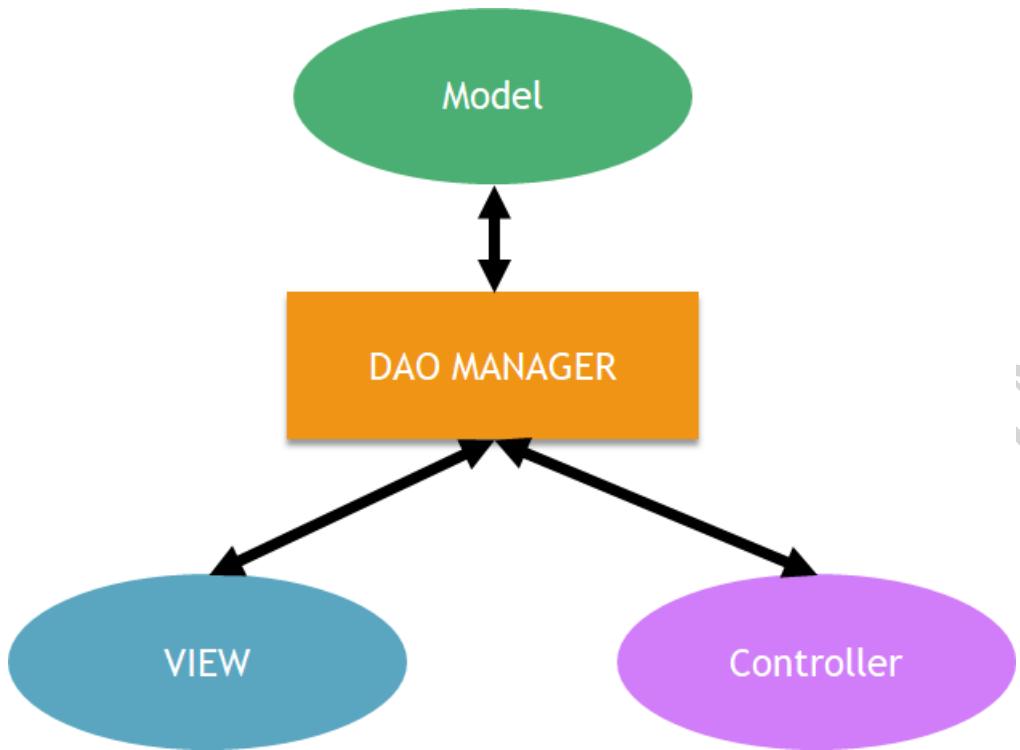
Pada main method tidak ada keyword untuk melakukan instance atau “new”. Objek langsung dibuat dan langsung dapat digunakan ketika memanggil method getInstance(). Dua baris kode didalam main dapat digantikan dengan kode dibawah ini:

```
SingleObject.getInstance().showMessage();
```

Eksekusi dari kode tersebut akan menghasilkan output yang sama dengan dua baris kode sebelumnya.

## 10.6 Model-View-Controller Pattern

Model-View-Controller atau MVC adalah sebuah metode untuk membuat sebuah aplikasi dengan memisahkan data (Model) dari tampilan (View) dan cara bagaimana memprosesnya (Controller). Dalam implementasinya kebanyakan framework dalam aplikasi website adalah berbasis arsitektur MVC. MVC memisahkan pengembangan aplikasi berdasarkan komponen utama yang membangun sebuah aplikasi seperti manipulasi data, antarmuka pengguna, dan bagian yang menjadi kontrol dalam sebuah aplikasi.



Gambar 10.2 Model View Controller (MVC)

Pola MVC memiliki layer yang disebut dengan Model yang merepresentasikan data yang digunakan oleh aplikasi sebagaimana proses bisnis yang diasosiasikan terhadapnya. Dengan memilahnya sebagai bagian terpisah, seperti penampungan data, persistence, serta proses manipulasi, terpisah dari bagian lain aplikasi. Terdapat beberapa kelebihan dalam pendekatan ini. Pertama, membuat detail dari data dan operasinya dapat ditempatkan pada area yang ditentukan (Model) dibanding tersebar dalam keseluruhan lingkup aplikasi. Hal ini memberikan keuntungan dalam proses pemeliharaan aplikasi. Kedua, dengan pemisahan total antara data dengan implementasi interface, komponen model dapat digunakan kembali oleh aplikasi lain yang memiliki kegunaan yang hampir sama.

Layer view mengandung keseluruhan detail dari implementasi user interface. Disini, komponen grafis menyediakan representasi proses internal aplikasi dan menuntun alur interaksi user terhadap aplikasi. Tidak ada layer lain yang berinteraksi dengan pengguna, hanya View. Penggunaan layer View memiliki beberapa kelebihan: memudahkan pengabungan divisi desain dalam development team. Divisi desain dapat berkonsentrasi pada style, look and feel, dan sebagainya, dalam aplikasi tanpa harus memperhatikan lebih pada detail yang lain. Dengan memiliki layer View yang terpisah memungkinkan ketersediaan multiple interface dalam aplikasi. Jika inti dari aplikasi terletak pada bagian lain (dalam

Model), multiple interfaces dapat dibuat (Swing, Web, Console), secara keseluruhan memiliki tampilan yang berbeda namun mengeksekusi komponen Model sesuai fungsionalitas yang diharapkan.

Terakhir, arsitektur MVC memiliki layer Controller. Layer ini menyediakan detail alur program dan transisi layer, dan juga bertanggungjawab akan penampungan events yang dibuat oleh user dari View dan melakukan update terhadap komponen Model menggunakan data yang dimasukkan oleh user. Kelebihan dalam penggunaan layer Controller secara terpisah adalah dengan menggunakan komponen terpisah untuk menampung detail dari transisi layer, komponen view dapat didesain tanpa harus memperhatikan bagian lain secara berlebih. Hal ini memudahkan team pengembang multiple interface bekerja secara terpisah dari yang lain secara simultan. Interaksi antar komponen View terabstraksi dalam Controller.

Data Access Object (DAO) merupakan sebuah object yang menyediakan sebuah abstract interface terhadap beberapa database atau mekanisme persistence, menyediakan beberapa operasi tertentu tanpa mengekspos detail database. Penerapan konsep ini sering disebut dengan separation of concern dimana setiap kode dipisahkan berdasarkan fungsinya sehingga kode diatasnya hanya perlu mengetahui secara abstrak cara mengakses data tanpa perlu mengetahui bagaimana akses ke sumber data diimplementasikan. DAO sering dikaitkan dengan Java EE dan akses ke relational database melalui JDBC API, karena memang DAO berasal dari pedoman praktek Sun Microsystem. Kebanyakan peggunaan DAO adalah satu objek DAO untuk satu objek entity.

## 9.5 Kuis dan Latihan Soal

1. Jelaskan bagaimana cara menangani exception?
2. Sebutkan macam-macam exception di Java (10 saja)?
3. Apakah class generik bisa diwariskan?
4. Apakah design pattern singleton dapat diwariskan?

# BAB 11 PEMROGRAMAN JAVA GUI DENGAN JAVA FX

## 11.1 Java FX

JavaFX adalah framework untuk membangun program-program Java berbasis *Graphical User Interface* (GUI). Ketika Java diperkenalkan, *class* GUI dipaket dalam sebuah *library* yang dikenal sebagai AbstractWindows Toolkit (AWT). AWT baik digunakan untuk mengembangkan user interface grafis sederhana, namun tidak untuk mengembangkan proyek GUI yang komprehensif. Selain itu, AWT rentan terhadap *bug* khususnya permasalahan *platform*. Komponen user interface AWT digantikan oleh *library* yang lebih baik, fleksibel, dan yang dikenal sebagai swing. Komponen swing dilukis secara langsung pada kanvas menggunakan kode Java. Komponen swing lebih sedikit bergantung pada platform target, dan menggunakan lebih sedikit sumber daya GUI asli. Swing dirancang untuk mengembangkan aplikasi GUI desktop. Saat ini penggunaan swing digantikan oleh platform GUI yang benar-benar baru yang dikenal sebagai JavaFX.

JavaFX menggabungkan teknologi GUI modern, JavaFX menyediakan dukungan *multitouch* untuk perangkat yang mendukung sentuhan seperti tablet dan ponsel pintar. JavaFX memiliki dukungan animasi 2D, 3D, dan pemutaran video serta audio. Dengan menggunakan perangkat lunak pihak ketiga, kita dapat mengembangkan program JavaFX untuk digunakan pada perangkat yang menjalankan OS atau Android. JavaFX adalah tool untuk mengembangkan aplikasi GUI lintas platform yang kaya pada komputer desktop dan pada perangkat genggam. Kode 11.1 merupakan contoh program sederhana untuk JavaFX.

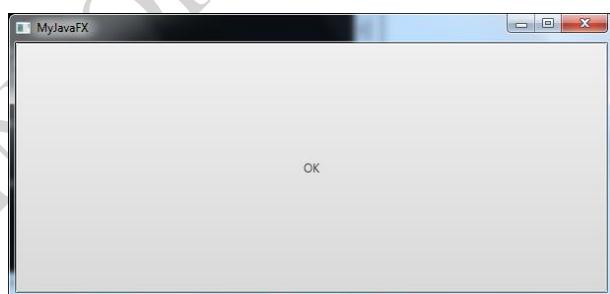
**Kode 11.1 MyJavaFX.java**

```
1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.control.Button;
4 import javafx.stage.Stage;
5 public class MyJavaFX extends Application {
6     @Override // Override method start method pada class Application
7     public void start(Stage primaryStage) {
8         // membuat scene dan menempatkan suatu button pada scene
9         Button btOK = new Button("OK");
```

```

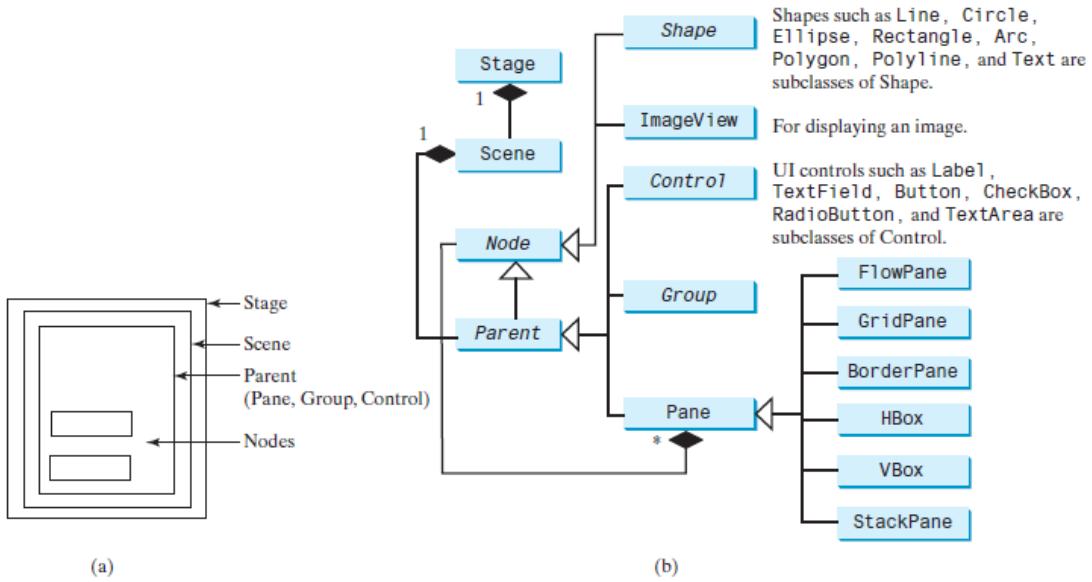
10    Scene scene = new Scene(btOK, 600, 250);
11    primaryStage.setTitle("MyJavaFX"); // Set judul stage
12    // menempatkan scene pada stage
13    primaryStage.setScene(scene);
14    primaryStage.show(); // Display stage
15 }
16 /* main method hanya diperlukan
17 untuk IDE yg tdk support JavaFX */
18 public static void main(String[] args) {
19     Application.launch(args);
20 }
21 }
```

Hasil dari Kode 11.1 ketika dijalankan disajikan pada Gambar 11.1. *Method start()* merupakan method yang dibaca pertamakali untuk semua aplikasi *javaFX*, *Method launch* adalah method statis yang didefinisikan dalam class aplikasi untuk menjalankan aplikasi *JavaFX* yang berdiri sendiri. method Main tidak diperlukan jika program dijalankan dari *command Line*, namun diperlukan untuk menjalankan program *JavaFX* dari IDE yang tidak support dengan *JavaFX*. Ketika menjalankan aplikasi *JavaFX* tanpa method main, JVM secara otomatis memanggil method *launch* untuk menjalankan aplikasi.



Gambar 11.1 Hasil tampilan kode MyJavaFX.java

Class *main* meng-*override* *method start* yang didefinisikan dalam *javafx.application*. Setelah application *JavaFX* diluncurkan, JVM membangun instance class menggunakan *constructor* no-arg dan memanggil method awal. *Method start* biasanya menempatkan kontrol UI dalam sebuah *scene* dan menampilkan *scene* dalam *stage*, seperti yang ditunjukkan pada Gambar 2



Gambar 11.2 (a) Pane dan Group digunakan untuk menempatkan Nodes. (b) Node bisa berupa shape,image view,UI control,group, dan pane.

Contoh pembuatan tombol Button dilakukan dengan `Button btOK = new Button("OK");`. Untuk membuat objek Button dan menempatkannya di dalam objek Scene dilakukan dengan `Scene scene = new Scene(btOK, 600, 250);`. Objek Scene dapat dibuat menggunakan constructor Scene (node, lebar, tinggi) seperti berikut: `Scene scene = new Scene(btOK, 600, 250);`. Membuat `scene` dengan menentukan lebar dan tinggi dan menempatkan suatu `node` (Button) pada `Scene`. Objek `Stage` merupakan suatu `window`. Objek `Stage` dengan nama primary stage secara otomatis dibuat oleh JVM ketika aplikasi dijalankan. Kode `primaryStage.setScene(scene);` baris ini mengatur `scene` ke primary stage `primaryStage.show();` dan ini menampilkan primary stage.

JavaFX memberi nama *class Stage* dan *Scene* menggunakan analogi dari sebuah drama. *stage* dianggap sebagai platform untuk mendukung *Scene*, dan *nodes* sebagai aktor untuk tampil di *Scene*. Kita bisa membuat lebih dari satu stage seperti Kode 11.2.

### Kode 11.2 MultipleStageDemo.java

```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.control.Button;
```

```

4 import javafx.stage.Stage;
5 public class MultipleStageDemo extends Application {
6     @Override
7     public void start(Stage primaryStage) {
8         Button btOK = new Button("OK");
9         Scene scene = new Scene(btOK, 600, 250);
10        primaryStage.setTitle("MyJavaFX");
11        primaryStage.setScene(scene);
12        primaryStage.show();
13        Stage stage = new Stage(); // membuat stage baru
14        stage.setTitle("Second Stage"); // Set judul stage
15        // Set scene dengan sebuah button di dalam stage
16        stage.setScene(new Scene(new Button("New Stage"),
17            200, 250));
18        stage.setResizable(false);
19        stage.show();
20    }
21 }
```

## 11.2 Panes, Groups, UI Controls, dan Shapes di Java FX

Pane, Groups, UI controls, dan shapes adalah bagian dari *node* seperti pada Gambar 11.2 (a). Pane secara otomatis meletakkan *node* di lokasi dengan ukuran yang diinginkan. *Node* adalah komponen visual seperti shape, image view, UI control, group, atau pane. Shape dapat berupa text, line, circle, ellipse, rectangle, arc, polygon, polyline, dll. Kontrol-UI berupa label, button, check box, radio button, text field, text area, and dll. Group merupakan *container* yg mengelompokkan koleksi *node*. Kode 11.3 memperlihatkan penggunaan Button pada JavaFx.

### Kode 11.3 ButtonInPane.java

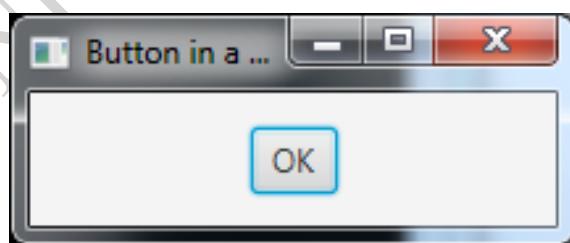
```

1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.control.Button;
4 import javafx.stage.Stage;
```

```

5 import javafx.scene.layout.StackPane;
6
7 public class ButtonInPane extends Application {
8     @Override
9     public void start(Stage primaryStage) {
10         StackPane pane = new StackPane();
11         pane.getChildren().add(new Button("OK"));
12         Scene scene = new Scene(pane, 200, 50);
13         primaryStage.setTitle("Button in a pane");
14         primaryStage.setScene(scene);
15         primaryStage.show();
16     }
17 }
```

Kode ButtonInPane.java membuat sebuah panel pane(stackPane) kemudian menambahkan button sebagai child dari pane. Method `getChildren()` mengembalikan instans dari `javafx.collections.ObservableList` dimana `ObservableList` mirip dengan `ArrayList` yang digunakan untuk menyimpan koleksi elemen. pemanggilan method `add` menambahkan elemen button ke dalam list, `StackPane` menempatkan *node* ditengah-tengah *pane* bertumpukan dengan elemen *node* lainnya. Hasil dari kode tersebut disajikan pada Gambar 11.3.



Gambar 11.3 Hasil Kode ButtonInPane.java

### 11.3 Property Binding

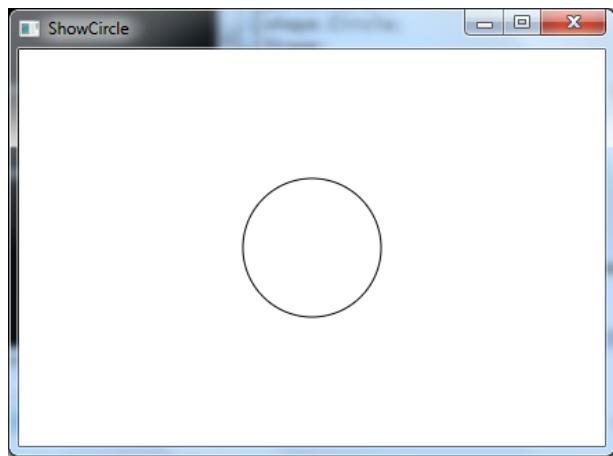
JavaFX memperkenalkan konsep yang disebut property binding yang memungkinkan obyek target terikat dengan obyek sumber. Jika nilai dalam obyek sumber berubah maka obyek target juga secara otomatis berubah. Obyek target disebut sebagai obyek binding atau properti binding, dan obyek sumber disebut obyek bindable atau obyek observable.

Untuk menampilkan circle agar selalu berada ditengah window meskipun ukuran window berubah, koordinat x dan y dari circle perlu diatur ulang ke tengah panel. Hal ini dapat dilakukan dengan binding centerX dengan lebar pane 2 dan centerY dengan tinggi pane 2. Implementasi tersebut diberikan pada Kode 11.4.

**Kode 11.4 ShowCircleCentered.java**

```
1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.Pane;
4 import javafx.scene.paint.Color;
5 import javafx.scene.shape.Circle;
6 import javafx.stage.Stage;
7 public class ShowCircleCentered extends Application {
8     @Override
9     public void start(Stage primaryStage) {
10         Pane pane = new Pane();
11
12         Circle circle = new Circle();
13
14         circle.centerXProperty().bind(pane.widthProperty().divide(2));
15
16         circle.centerYProperty().bind(pane.heightProperty().divide(2));
17
18         circle.setRadius(50);
19         circle.setStroke(Color.BLACK);
20         circle.setFill(Color.WHITE);
21         pane.getChildren().add(circle);
22
23         Scene scene = new Scene(pane, 200, 200);
24         primaryStage.setTitle("ShowCircle");
25         primaryStage.setScene(scene);
26         primaryStage.show();
27     }
28 }
```

Hasil dari Kode ShowCircleCentered.java diperlihatkan pada Gambar 11.4.



Gambar 11.4 Hasil output ShowCircleCentered.java

## 11.4 Properties dan Method Node

Class node mendefinisikan beberapa properti dan method yang secara umum berlaku untuk semua node yaitu properti style dan rotate. Properti style pada JavaFX mirip dengan cascading style sheets (CSS) yang digunakan untuk mengatur style dari elemen HTML pada halaman web, properti style dalam JavaFX disebut CSS JavaFX dan didefinisikan menggunakan prefik -fx- .

Contoh:

```
circle.setStyle("-fx-stroke: black; -fx-fill: red;")
```

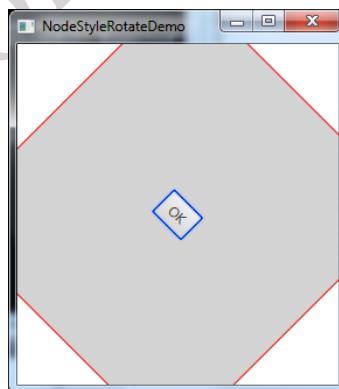
Property rotate dapat mengatur sudut perputaran suatu node terhadap titik pusatnya, nilai positif membuat node berputar searah jarum jam dan nilai negatif sebaliknya. Implementasi dari rotate tersebut dilakukan pada Kode 11.5. Hasil dari Kode 11.5 disajikan pada Gambar 11.5.

### Kode 11.5 NodeStyleRotateDemo.java

```
1 import javafx.application.Application;  
2 import javafx.scene.Scene;  
3 import javafx.scene.control.Button;  
4 import javafx.stage.Stage;
```

```

5 import javafx.scene.layout.StackPane;
6
7 public class NodeStyleRotateDemo extends Application {
8     @Override
9     public void start(Stage primaryStage) {
10         StackPane pane = new StackPane();
11         Button btOK = new Button("OK");
12         btOK.setStyle("-fx-border-color: blue;");
13         pane.getChildren().add(btOK);
14         pane.setRotate(45);
15         pane.setStyle("-fx-border-color: red; -fx-background-
color: lightgray;");
16         Scene scene = new Scene(pane, 200, 250);
17         primaryStage.setTitle("NodeStyleRotateDemo");
18         primaryStage.setScene(scene);
19         primaryStage.show();
20     }
21 }
```



Gambar 11.5 Hasil dari Kode NodeStyleRotateDemo.java

Class Color digunakan untuk menentukan warna, JavaFX mendefinisikan class Paint dalam menggambar suatu node. Instan color dapat dibuat menggunakan constructor berikut ini:

```
public Color(double r, double g, double b, double opacity);
```

Dimana r, g, dan b merupakan komponen warna red,green,blue dengan nilai dengan range 0.0 (gelap) sampai dengan 1.0 (terang). Nilai opacity menentukan transparansi warna dengan nilai berada pada range 0.0 (*completely transparent*) sampai dengan 1.0 (*completely opaque*), hal ini dikenal sebagai model *RGBA (Red Green Blue Alpha)*. Berikut contoh penerapannya:

```
Color color = new Color(0.25, 0.14, 0.333, 0.51);
```

Untuk pembuatan font tulisan dengan tipe atau format tertentu. Intansiasi suatu font dapat menggunakan *constructor*-nya atau menggunakan method static. Berikut adalah Contoh pembuatan font dengan format tertentu:

```
Font font1 = new Font("SansSerif", 16);
```

```
Font font2 = Font.font("Times New Roman", FontWeight.BOLD,  
FontPosture.ITALIC, 12);
```

Kode 11.6 merupakan contoh penerapan pengaturan jenis tulisan atau font pada sebuah label.

#### Kode 11.6 FontDemo.java

```
1 import javafx.application.Application;  
2 import javafx.scene.Scene;  
3 import javafx.scene.layout.*;  
4 import javafx.scene.paint.Color;  
5 import javafx.scene.shape.Circle;  
6 import javafx.scene.text.*;  
7 import javafx.scene.control.*;  
8 import javafx.stage.Stage;  
9  
10 public class FontDemo extends Application {  
11     @Override  
12     public void start(Stage primaryStage) {  
13         Pane pane = new StackPane();  
14         Circle circle = new Circle();  
15         circle.setRadius(50);  
16         circle.setStroke(Color.BLACK);  
17         circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));  
18         pane.getChildren().add(circle);  
19     }  
20 }
```

```

18
19     Label label = new Label("JavaFX");
20     label.setFont(Font.font("Times New Roman",
21                           FontWeight.BOLD, FontPosture.ITALIC, 20));
22     pane.getChildren().add(label);
23
24     Scene scene = new Scene(pane);
25     primaryStage.setTitle("FontDemo");
26     primaryStage.setScene(scene);
27     primaryStage.show();
28 }
29 }
```

Berikutnya adalah penambahan citra atau gambar dengan Java FX yang diwakilkan oleh class `Image` dan Class `ImageView`. Class `Image` mewakili suatu citra grafis, sedangkan class `ImageView` digunakan untuk menampilkan suatu citra. Class `javafx.scene.image.Image` mewakili suatu image grafis dan digunakan untuk menampilkan image dari suatu namaFile atau suatu URL Berikut adalah contoh dari membuat suatu objek `Image` dari suatu gambar gif:

```

new Image("image/us.gif")
new Image("http://liveexample.pearsoncmg.com/book/image/us.gif")
```

Kode `javafx.scene.image.ImageView` merupakan suatu node yang digunakan untuk menampilkan image, sebuah `ImageView` dapat dibuat dari obyek `Image`, misalnya:

```

Image image = new Image("image/us.gif");
ImageView imageView = new ImageView(image);
```

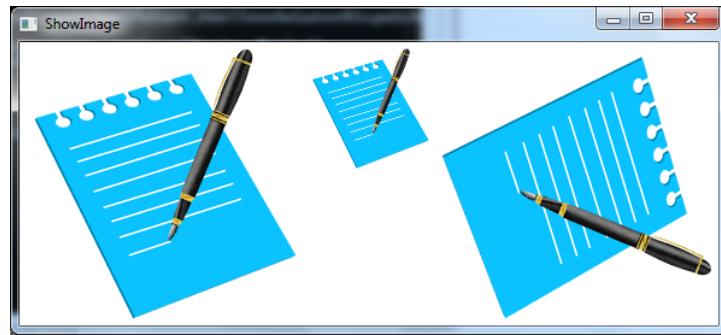
Sebagai alternatif suatu `ImageView` dapat juga dibuat langsung dari suatu file ataupun URL, seperti:

```
ImageView imageView = new ImageView("image/us.gif");
```

Kode 11.7 merupakan implementasi full dari `Image` dan `ImageView`. Gambar 11.6 merupakan hasil output dari Kode 11.7.

**Kode 11.7 ShowImage.java**

```
1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.layout.HBox;
4 import javafx.scene.layout.Pane;
5 import javafx.geometry.Insets;
6 import javafx.stage.Stage;
7 import javafx.scene.image.Image;
8 import javafx.scene.image.ImageView;
9
10 public class ShowImage extends Application {
11     @Override
12     public void start(Stage primaryStage) {
13         Pane pane = new HBox(10);
14         pane.setPadding(new Insets(5, 5, 5, 5));
15         Image image = new Image("image/srt30.png");
16         pane.getChildren().add(new ImageView(image));
17         ImageView imageView2 = new ImageView(image);
18         imageView2.setFitHeight(100);
19         imageView2.setFitWidth(100);
20         pane.getChildren().add(imageView2);
21         ImageView imageView3 = new ImageView(image);
22         imageView3.setRotate(90);
23         pane.getChildren().add(imageView3);
24         Scene scene = new Scene(pane);
25         primaryStage.setTitle("ShowImage");
26         primaryStage.setScene(scene);
27         primaryStage.show();
28     }
29 }
```



Gambar 11.6 Hasil output ShowImage.java

JavaFX menyediakan banyak jenis Pane yg secara otomatis menempatkan node pada lokasi dan ukuran yang diinginkan. Pane dan Group merupakan container yang mengatur penempatan suatu node. Class group digunakan untuk mengelompokkan node dan menjalankan transformasi dan mengatur skala dalam group. Obyek Pane dan UI control dapat dirubah ukurannya, namun obyek group,shape,dan text tidak dapat dirubah ukurannya. JavaFX menyediakan beberapa jenis pane untuk mengorganisir node dalam suatu container yang di sajikan pada Tabel 11.1.

Tabel 11.1 Jenis-jenis Pane

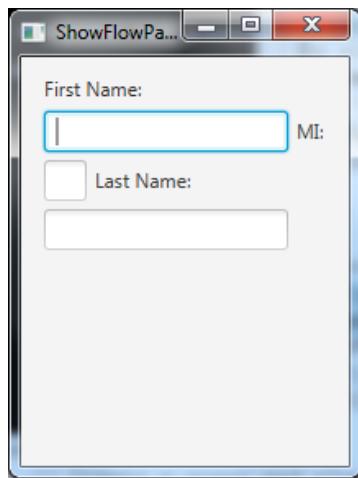
Class	Deskripsi
Pane	Base Class yaitu berupa layout pane, berisi method getChildren untuk mengembalikan list node yang ada dalam suatu pane.
StackPane	Menempatkan node di atas node lainnya pada tengah panel
FlowPane	Menempatkan node perbaris secara horisontal atau perkolom secara vertikal
GridPane	Menempatkan node di dalam cell dalam grid
BorderPane	Menempatkan node pada area top,right,bottom, left, dan center.
HBox	Menempatkan node dalam satu baris
VBox	Menempatkan node dalam satu kolom

Menempatkan node secara perbaris dalam susunan horisontal dari kiri ke kanan, atau secara perkolom dengan susunan vertikal dari atas ke bawah, dalam susunan sesuai urutan suatu

node ditambahkan. Setelah satu baris/kolom di isikan, selanjutnya berpindah ke baris/kolom baru. Untuk menentukan arah susunan penempatan node baik secara horizontal atau vertical digunakan konstanta: `Orientation.HORIZONTAL` atau `Orientation.VERTICAL`. Kode 11.8 merupakan kode untuk menampilkan Pane dan Gambar 11.7 merupakan hasil output dari Kode 11.8.

#### Kode 11.8 ShowFlowPane.java

```
1 import javafx.application.Application;
2 import javafx.geometry.Insets;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Label;
5 import javafx.scene.control.TextField;
6 import javafx.scene.layout.FlowPane;
7 import javafx.stage.Stage;
8 public class ShowFlowPane extends Application {
9     @Override
10    public void start(Stage primaryStage) {
11        FlowPane pane = new FlowPane();
12        pane.setPadding(new Insets(11, 12, 13, 14));
13        pane.setHgap(5);
14        pane.setVgap(5);
15        pane.getChildren().addAll(new Label("First Name:"),
16            new TextField(),new Label("MI:"));
17        TextField tfMi = new TextField();
18        tfMi.setPrefColumnCount(1);
19        pane.getChildren().addAll(tfMi,
20            new Label("Last Name:"),new TextField());
21        Scene scene = new Scene(pane, 200, 250);
22        primaryStage.setTitle("ShowFlowPane");
23        primaryStage.setScene(scene);
24        primaryStage.show();
25    }
}
```



Gambar 11.7 Hasil output ShowFlowPane.java

GridPane menyusun node dalam bentuk susunan grid/matriks, suatu node ditempatkan pada suatu cell.

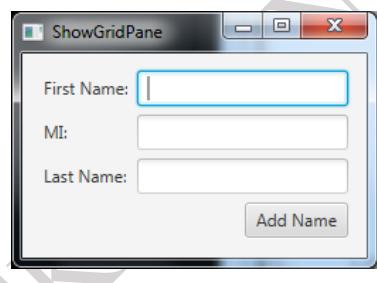
#### Kode 11.8 ShowGridPane.java

```
1 import javafx.application.Application;
2 import javafx.geometry.HPos;
3 import javafx.geometry.Insets;
4 import javafx.geometry.Pos;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Button;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TextField;
9 import javafx.scene.layout.GridPane;
10 import javafx.stage.Stage;
11
12 public class ShowGridPane extends Application {
13     @Override
14     public void start(Stage primaryStage) {
15
16         GridPane pane = new GridPane();
17         pane.setAlignment(Pos.CENTER);
```

```

18     pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
19     pane.setHgap(5.5);
20     pane.setVgap(5.5);
21
22     pane.add(new Label("First Name:"), 0, 0);
23     pane.add(new TextField(), 1, 0);
24     pane.add(new Label("MI:"), 0, 1);
25     pane.add(new TextField(), 1, 1);
26     pane.add(new Label("Last Name:"), 0, 2);
27     pane.add(new TextField(), 1, 2);
28     Button btAdd = new Button("Add Name");
29     pane.add(btAdd, 1, 3);
30     GridPane.setHalignment(btAdd, HPos.RIGHT);
31
32     Scene scene = new Scene(pane);
33     primaryStage.setTitle("ShowGridPane");
34     primaryStage.setScene(scene);
35     primaryStage.show();
36 }
37 }
```

## Output



### 11.4.1 BorderPane

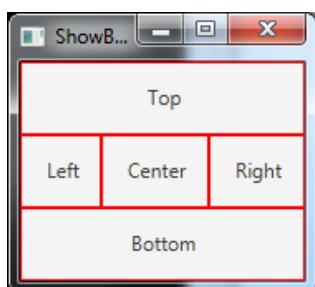
BorderPane menempatkan node pada lima area yaitu: top,right,bottom, left, dan center dengan menggunakan method: setTop(node), setBottom(node), setLeft(node), setRight (node) , dan setCenter(node).

**Kode 11.9 ShowBorderPane.java**

```
1 import javafx.application.Application;
2 import javafx.geometry.Insets;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Label;
5 import javafx.scene.layout.BorderPane;
6 import javafx.scene.layout.StackPane;
7 import javafx.stage.Stage;
8
9 public class ShowBorderPane extends Application {
10
11     @Override
12
13     public void start(Stage primaryStage) {
14
15         BorderPane pane = new BorderPane();
16
17         pane.setTop(new CustomPane("Top"));
18
19         pane.setRight(new CustomPane("Right"));
20
21         pane.setBottom(new CustomPane("Bottom"));
22
23         pane.setLeft(new CustomPane("Left"));
24
25         pane.setCenter(new CustomPane("Center"));
26
27         Scene scene = new Scene(pane);
28
29         primaryStage.setTitle("ShowBorderPane");
30
31         primaryStage.setScene(scene);
32
33         primaryStage.show();
34
35     }
36
37     class CustomPane extends StackPane {
38
39         public CustomPane(String title) {
40
41             getChildren().add(new Label(title));
42
43             setStyle("-fx-border-color: red");
44
45             setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
46
47         }
48
49     }
50 }
```

```
32 }
```

### Output



### 11.4.2 HBox dan VBox

HBox Menempatkan node dalam susunan satu baris, sedangkan VBox menempatkan node dalam susunan satu kolom.

Kode 11.10 BoxVBox.java

```
1 import javafx.application.Application;
2 import javafx.geometry.Insets;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.control.Label;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.HBox;
8 import javafx.scene.layout.VBox;
9 import javafx.stage.Stage;
10 import javafx.scene.image.Image;
11 import javafx.scene.image.ImageView;
12
13 public class ShowHBoxVBox extends Application {
14     @Override
15     public void start(Stage primaryStage) {
16
17         BorderPane pane = new BorderPane();
18         pane.setTop(getHBox());
```

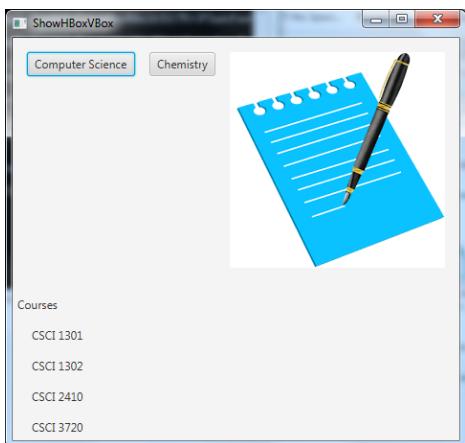
```

19         pane.setLeft(getVBox());
20
21         Scene scene = new Scene(pane);
22
23         primaryStage.setTitle("ShowHBoxVBox");
24         primaryStage.setScene(scene);
25         primaryStage.show();
26     }
27
28     private HBox getHBox() {
29
30         HBox hBox = new HBox(15);
31
32         hBox.setPadding(new Insets(15, 15, 15, 15));
33         hBox.setStyle("-fx-background-color: gold");
34         hBox.getChildren().add(new Button("Computer Science"));
35         hBox.getChildren().add(new Button("Chemistry"));
36
37         ImageView imageView = new ImageView(new
38             Image("image/srt30.png"));
39
40         hBox.getChildren().add(imageView);
41
42         return hBox;
43     }
44
45     private VBox getVBox() {
46
47         VBox vBox = new VBox(15);
48
49         vBox.setPadding(new Insets(15, 5, 5, 5));
50         vBox.getChildren().add(new Label("Courses"));
51
52         Label[] courses = {new Label("CSCI 1301"), new
53             Label("CSCI 1302"),
54             new Label("CSCI 2410"), new Label("CSCI 3720")};
55
56
57         for (Label course: courses) {
58
59             vBox.setMargin(course, new Insets(0, 0, 0, 15));
60
61             vBox.getChildren().add(course);
62         }
63
64         return vBox;
65     }

```

```
49 }  
50 }
```

## Output

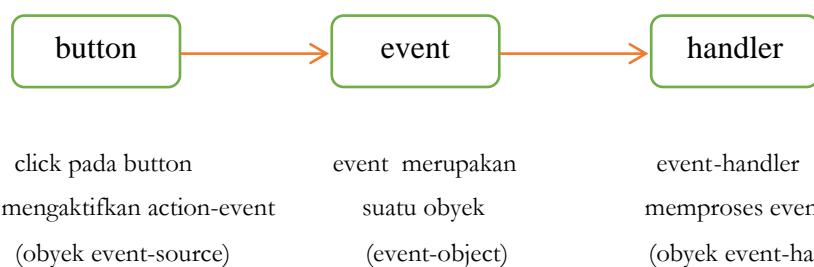


## 11.5 Event-Driven Programming

Tidak semua obyek bisa ditangani menjadi event-action. Untuk bisa menjadi pengendali dari suatu event-action ada dua persyaratan harus dipenuhi:

1. Obyek harus merupakan instan dari interface `EventHandler<T extends Event>`.
2. Obyek Event Handler harus diregistrasikan dengan obyek event source menggunakan method `source.setOnAction(handler)`.

Untuk merespon sebuah button click, maka diperlukan kode untuk memproses aksi button click tersebut. button merupakan obyek event-source yang merupakan tempat action berasal. Diperlukan pembuatan obyek yang mampu menangani event-action pada button. Obyek ini disebut event handler, seperti yang ditunjukkan pada Gambar berikut:



Gambar 11.8 Pemrograman event-driven

Interface EventHandler<ActionEvent> berisi method handle(ActionEvent) utk memproses action-event. Class handler yang dibuat harus melakukan override method ini untuk merespon suatu event.

#### HandleEvent.java

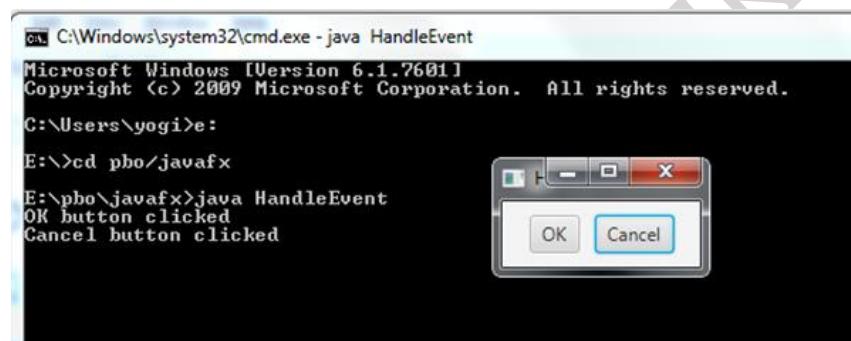
```
1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.layout.HBox;
6 import javafx.stage.Stage;
7 import javafx.event.ActionEvent;
8 import javafx.event.EventHandler;
9 public class HandleEvent extends Application {
10     @Override
11     public void start(Stage primaryStage) {
12         HBox pane = new HBox(10);
13         pane.setAlignment(Pos.CENTER);
14         Button btOK = new Button("OK");
15         Button btCancel = new Button("Cancel");
16         OKHandlerClass handler1 = new OKHandlerClass();
17         btOK.setOnAction(handler1);
18         CancelHandlerClass handler2 = new CancelHandlerClass();
19         btCancel.setOnAction(handler2);
20         pane.getChildren().addAll(btOK, btCancel);
21
22         Scene scene = new Scene(pane);
23         primaryStage.setTitle("HandleEvent");
24         primaryStage.setScene(scene);
25         primaryStage.show();
26     }
27 }
28 }
```

```

29 class OKHandlerClass implements EventHandler<ActionEvent> {
30     @Override
31     public void handle(ActionEvent e) {
32         System.out.println("OK button clicked");
33     }
34 }
35
36 class CancelHandlerClass implements EventHandler<ActionEvent> {
37     @Override
38     public void handle(ActionEvent e) {
39         System.out.println("Cancel button clicked");
40     }
41 }

```

### Output



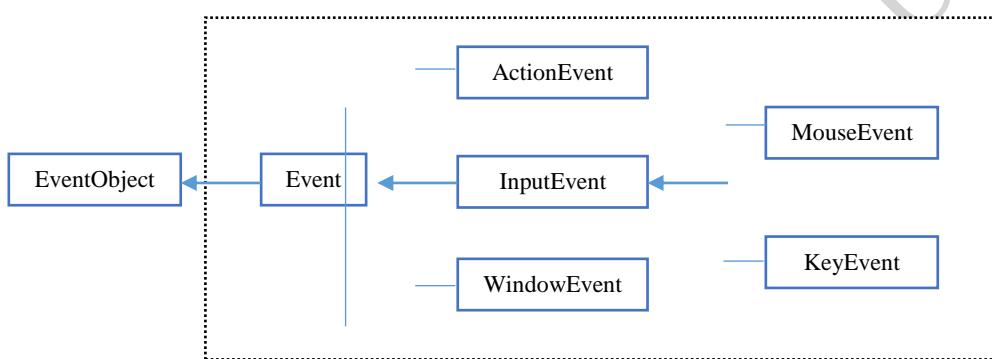
Dua class handler yang didefinisikan pada listing code diatas dan masing--masing class handler meng-implementasikan EventHandler<ActionEvent> untuk memproses ActionEvent. Obyek handler1 merupakan instan dari class OKHandler,yang diregistrasikan untuk button btOK. Ketika button OK diclick method handle(ActionEvent) dalam class OKHandler dipanggil untuk memproses event.

## 11.6 Events dan Event Sources

Event adalah suatu obyek yang dibuat dari suatu event-source , membangkitkan event artinya membuat suatu event dan mendelegasikan handler untuk menghandle event tersebut.

Program GUI java berinteraksi dengan user dan event mengendalikan(drive) eksekusinya proses seperti ini disebut pemrograman event-driven. Event dapat didefinisikan sebagai sinyal ke program bahwa sesuatu telah terjadi. event dibangkitkan oleh aksi-aksi user seperti pergerakan mouse, click mouse, penekanan tombol. program dapat memilih utk merespon atau mengabaikan event.

Komponen yang menciptakan suatu event dan menjalankannya disebut event-source/source-object/source-component, sebagai contoh button merupakan source-object dari suatu action-event button-click. Event merupakan instan dari suatu class event dan java.util.EventObject adalah class root dari susunan hirarki hubungan class-class event sebagai berikut:



Gambar 11.9 obyek event pada JavaFX yang merupakan obyek dari class javafx.event.Event

## 11.7 Inner Class

Inner class/nested class merupakan suatu class yang didefinisikan dalam lingkup class lainnya, inner classes berguna untuk mendefinisikan class-class handler. Pada umumnya inner class hanya digunakan oleh outer class-nya.

<b>Kode 11.11 Luar.java</b>	<b>Kode 11.12 TestInnerClass.java</b>
<pre> 1 public class Luar { 2     int varLuar; 3     void printLuar(){ 4         System.out.println("cetak 5             luar"); 6     } 7     class Dalam { 8 9     } 10 }</pre>	<pre> 1 public class TestInnerClass{ 2 3     public static void 4     main(String[] x){ 5         //cara1 membuat obyek 6         Luar l = new Luar(); 7         Luar.Dalam d = l.new 8         Dalam(); 9     } 10 }</pre>

7	int varDalam;	6	l.printLuar();
8	void printDalam(){	7	d.printDalam();
9	System.out.println("cetak	8	//cara2 membuat obyek
	dalam");	9	Luar.Dalam dlm = new
10	}	10	Luar().new Dalam();
11	}	11	dlm.printDalam();
12	}	12	}
		13	}

### Output kode 11.12

```
C:\Windows\system32\cmd.exe
E:\pbo\javafx>java TestInnerClass
cetak luar
cetak dalam
cetak luar
cetak dalam
E:\pbo\javafx>
```

Contoh menggunakan inner class:

### Kode 11.13 HandleEvent2.java

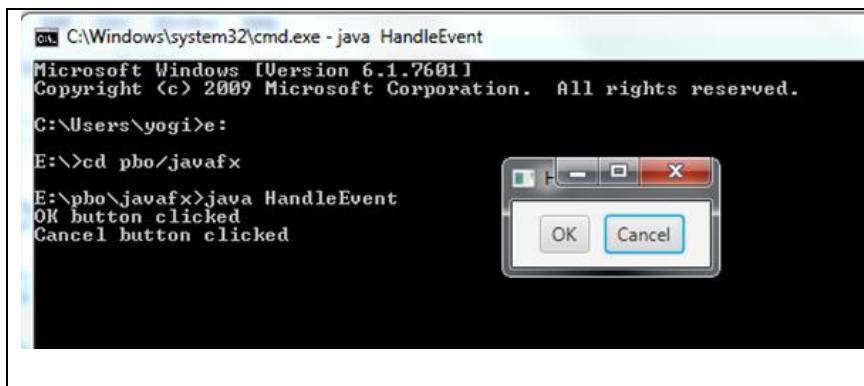
```
1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.layout.HBox;
6 import javafx.stage.Stage;
7 import javafx.event.ActionEvent;
8 import javafx.event.EventHandler;
9 public class HandleEvent2 extends Application {
10     public void start(Stage primaryStage) {
11         HBox pane = new HBox(10);
12         pane.setAlignment(Pos.CENTER);
13         Button btOK = new Button("OK");
14         Button btCancel = new Button("Cancel");
```

```

15         //OKHandlerClass handler1 = new OKHandlerClass();
16
17         //btOK.setOnAction(handler1);
18
19         //membuat obyek handler
20         btOK.setOnAction(new OKHandlerClass());
21
22         //CancelHandlerClass      handler2      =      new
23         CancelHandlerClass();
24
25         //btCancel.setOnAction(handler2);
26
27         //membuat obyek handler
28         btCancel.setOnAction(new CancelHandlerClass());
29
30         pane.getChildren().addAll(btOK, btCancel);
31
32         Scene scene = new Scene(pane);
33
34         primaryStage.setTitle("HandleEvent");
35
36         primaryStage.setScene(scene);
37
38         primaryStage.show();
39
40     }
41
42     //inner classs
43     class OKHandlerClass implements EventHandler<ActionEvent> {
44
45         @Override
46
47         public void handle(ActionEvent e) {
48
49             System.out.println("OK button clicked");
50
51         }
52
53     }
54
55     //inner classs
56     class CancelHandlerClass implements EventHandler<ActionEvent> {
57
58         @Override
59
60         public void handle(ActionEvent e) {
61
62             System.out.println("Cancel button clicked");
63
64         }
65
66     }
67
68 }

```

**Output:**



## 11.8 Handler berupa Anonymous Inner-Class

Anonymous inner class merupakan suatu inner class tanpa nama,dimana dalam mendefinisikan suatu inner class sekaligus juga membuat instannya.

Contoh menggunakan anonymous inner class:

**Kode 11. 14 HandleEvent3.java**

```
1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.layout.HBox;
6 import javafx.stage.Stage;
7 import javafx.event.ActionEvent;
8 import javafx.event.EventHandler;
9 public class HandleEvent3 extends Application {
10     @Override
11     public void start(Stage primaryStage) {
12         HBox pane = new HBox(10);
13         pane.setAlignment(Pos.CENTER);
14         Button btOK = new Button("OK");
15         Button btCancel = new Button("Cancel");
16         //membuat obyek handler
17         //btOK.setOnAction(new OKHandlerClass());
```

```

18         //menggunakan Anonymous inner class
19         btOK.setOnAction(new
20             EventHandler<ActionEvent>(){
21                 public void handle(ActionEvent e) {
22                     System.out.println("OK button clicked");
23                 }
24             });
25             //membuat obyek handler
26             //btCancel.setOnAction(new CancelHandlerClass());
27             //menggunakan Anonymous inner class
28             btCancel.setOnAction(new EventHandler<ActionEvent>(){
29                 public void handle(ActionEvent e) {
30                     System.out.println("Cancel button clicked");
31                 }
32             });
33             pane.getChildren().addAll(btOK, btCancel);
34             Scene scene = new Scene(pane);
35             primaryStage.setTitle("HandleEvent");
36             primaryStage.setScene(scene);
37             primaryStage.show();
38     }

```

**Output:**



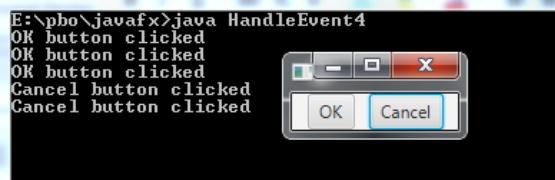
## 11.9 Lambda Expressions

Lambda expressions digunakan untuk menyederhanakan penggunaan code dalam event-handling.

### Kode 11. 15 HandleEvent4.java

```
1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.layout.HBox;
6 import javafx.stage.Stage;
7 import javafx.event.ActionEvent;
8 import javafx.event.EventHandler;
9 public class HandleEvent4 extends Application {
10     @Override
11     public void start(Stage primaryStage) {
12         HBox pane = new HBox(10);
13         pane.setAlignment(Pos.CENTER);
14         Button btOK = new Button("OK");
15         Button btCancel = new Button("Cancel");
16         //membuat obyek handler
17         btOK.setOnAction((ActionEvent e) -> {
18             System.out.println("OK button clicked");
19         });
20         //membuat obyek handler
21         //membuat obyek handler
22         btCancel.setOnAction((ActionEvent e) -> {
23             System.out.println("Cancel button clicked");
24         });
25         pane.getChildren().addAll(btOK, btCancel);
26         Scene scene = new Scene(pane);
27         primaryStage.setTitle("HandleEvent");
28         primaryStage.setScene(scene);
29         primaryStage.show();
30     }
31 }
```

## Output



## 11.10 Studi kasus aplikasi GUI menghitung Hutang

Berikut adalah contoh kode program GUI sederhana untuk membuat kalkulator hutang.

### Kode 11.16 Hutang.java

```
1  public class Hutang {  
2      private double bungaPerTahun;  
3      private int jmlTahun;  
4      private double jmlPinjaman;  
5      private java.util.Date tglPeminjaman;  
6      public Hutang() {  
7          this(2.5, 1, 1000);  
8      }  
9      public Hutang(double bungaPerTahun, int jmlTahun, double  
10     jmlPinjaman)  
11     {  
12         this.bungaPerTahun = bungaPerTahun;  
13         this.jmlTahun = jmlTahun;  
14         this.jmlPinjaman = jmlPinjaman;  
15         tglPeminjaman = new java.util.Date();  
16     }  
17  
18     public double getBungaPerTahun() {  
19         return bungaPerTahun;  
20     }  
21  
22     public void setBungaPerTahun(double bungaPerTahun) {
```

```

22         this.bungaPerTahun = bungaPerTahun;
23     }
24
25     public int getJmlTahun() {
26         return jmlTahun;
27     }
28
29     public void setJmlTahun(int jmlTahun) {
30         this.jmlTahun = jmlTahun;
31     }
32
33     public double getJmlPinjaman() {
34         return jmlPinjaman;
35     }
36
37     public void setJmlPinjaman(double jmlPinjaman) {
38         this.jmlPinjaman = jmlPinjaman;
39     }
40
41     public double getBayarPerBulan() {
42         double bungaPerBulan = bungaPerTahun / 1200;
43         double BayarPerBulan = jmlPinjaman * bungaPerBulan/
44             (1 - (1 / Math.pow(1 + bungaPerBulan, jmlTahun * 12)));
45         return BayarPerBulan;
46     }
47
48     public double getTotalBayar() {
49         double totalBayar = getBayarPerBulan() * jmlTahun * 12;
50         return totalBayar;
51     }
52     public java.util.Date getTglPeminjaman() {
53         return tglPeminjaman;

```

53	}
54	}

**Kode 11. 17 HitungHutang.java**

```
1 import javafx.application.Application;
2 import javafx.geometry.Pos;
3 import javafx.geometry.HPos;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Button;
6 import javafx.scene.control.Label;
7 import javafx.scene.control.TextField;
8 import javafx.scene.layout.GridPane;
9 import javafx.stage.Stage;
10
11 public class HitungHutang extends Application {
12     private TextField tfBungaPerTahun = new TextField();
13     private TextField tfJmlTahun = new TextField();
14     private TextField tfJmlHutang = new TextField();
15     private TextField tfBayarPerBulan = new TextField();
16     private TextField tfTotalBayar = new TextField();
17     private Button btHitung = new Button("Hitung");
18
19     @Override
20     public void start(Stage primaryStage) {
21         // membuat UI
22         GridPane gridPane = new GridPane();
23         gridPane.setHgap(5);
24         gridPane.setVgap(5);
25         gridPane.add(new Label("Bunga Tahunan:"), 0, 0);
26         gridPane.add(tfBungaPerTahun, 1, 0);
27         gridPane.add(new Label("Jml Tahun:"), 0, 1);
28         gridPane.add(tfJmlTahun, 1, 1);
29         gridPane.add(new Label("Jml Hutang:"), 0, 2);
```

```

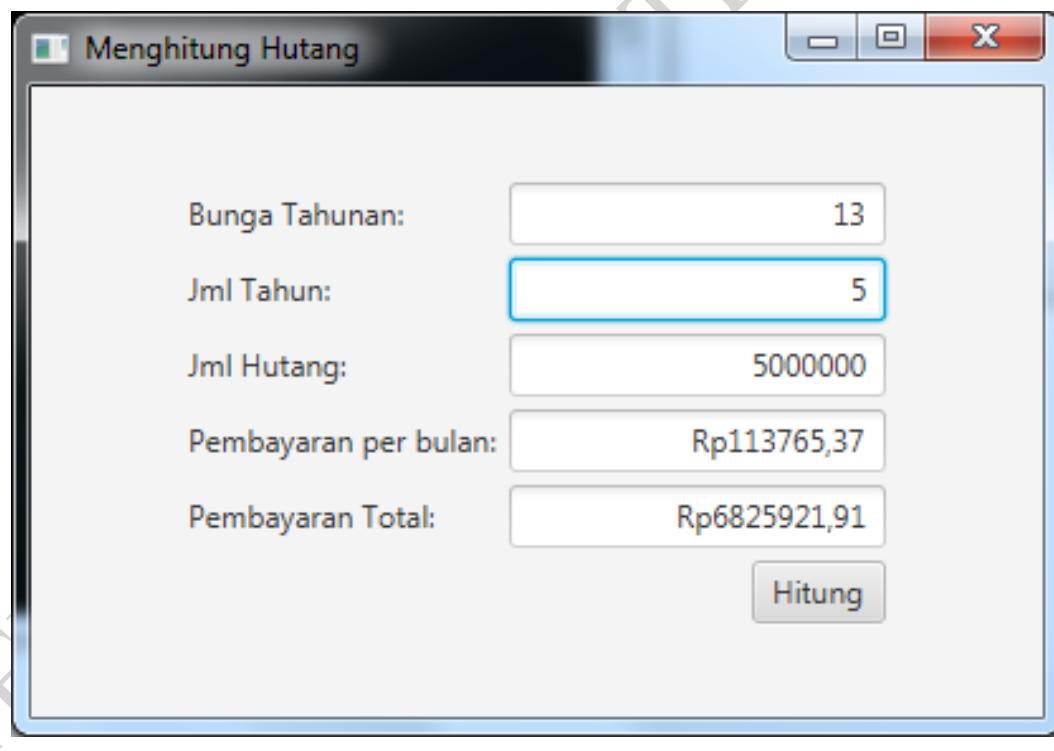
30     gridPane.add(tfJmlHutang, 1, 2);
31     gridPane.add(new Label("Pembayaran per bulan:"), 0, 3);
32     gridPane.add(tfBayarPerBulan, 1, 3);
33     gridPane.add(new Label("Pembayaran Total:"), 0, 4);
34     gridPane.add(tfTotalBayar, 1, 4);
35     gridPane.add(btHitung, 1, 5);
36
37     // Set properti UI
38     gridPane.setAlignment(Pos.CENTER);
39     tfBungaPerTahun.setAlignment(Pos.BOTTOM_RIGHT);
40     tfJmlTahun.setAlignment(Pos.BOTTOM_RIGHT);
41     tfJmlHutang.setAlignment(Pos.BOTTOM_RIGHT);
42     tfBayarPerBulan.setAlignment(Pos.BOTTOM_RIGHT);
43     tfTotalBayar.setAlignment(Pos.BOTTOM_RIGHT);
44     tfBayarPerBulan.setEditable(false);
45     tfTotalBayar.setEditable(false);
46     GridPane.setHalignment(btHitung, HPos.RIGHT);
47
48     // Process events
49     btHitung.setOnAction(e -> hitungPembayaranHutang());
50
51     // buat scene dan tempatkan dlm stage
52     Scene scene = new Scene(gridPane, 400, 250);
53     primaryStage.setTitle("Menghitung Hutang");
54     primaryStage.setScene(scene);
55     primaryStage.show();
56
57     private void hitungPembayaranHutang() {
58         // ambil nilai dari text fields
59         double bunga =

```

Double.parseDouble(tfBungaPerTahun.getText());

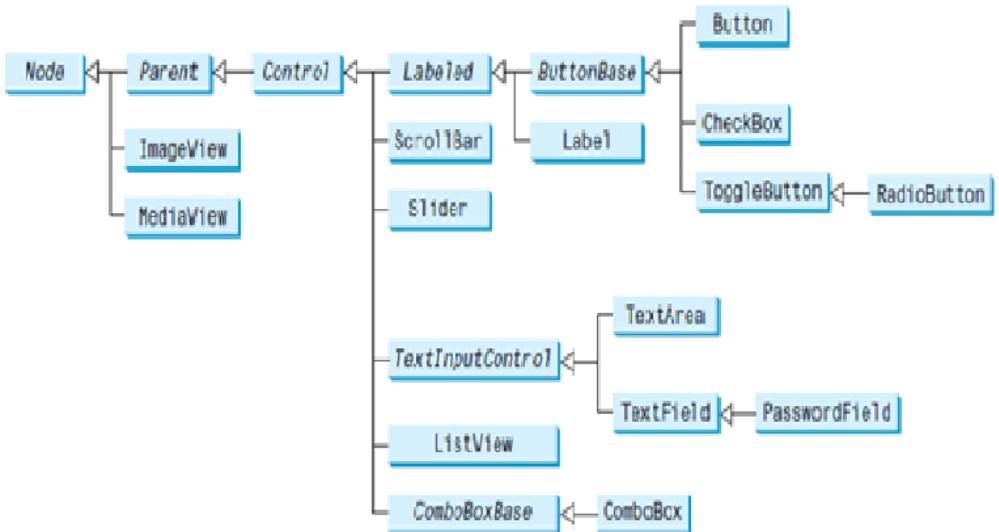
```
60     int year = Integer.parseInt(tfJmlTahun.getText());
61     double loanAmount =
62         Double.parseDouble(tfJmlHutang.getText());
63
64     // membuat obyek hutang
65     Hutang hutang = new Hutang(bunga, year, loanAmount);
66
67     // Display pembayaran bulanan dan pembayaran total
68     tfBayarPerBulan.setText(String.format("Rp%.2f", hutang.
69         getBayarPerBulan()));
70     tfTotalBayar.setText(String.format("Rp%.2f", hutang.
71         getTotalBayar()));
72 }
```

### Output



## 11.11 JavaFX UI Controls

Diagram kontrol dari keseluruhan JavaFX UI disajikan pada Gambar 11.10.



Gambar 11.10 UI controls yang sering digunakan dalam membuat user-interface

### 11.11.1 Label

Label adalah area tampilan untuk teks pendek ataupun node. Label sering digunakan untuk memberi keterangan teks (memberi label) terhadap suatu UI kontrol.

Membuat label:

```

Label lb1 = new Label("US\n50 States", us);
lb1.setStyle("-fx-border-color: green; -fx-border-width: 2");
lb1.setContentDisplay(ContentDisplay.BOTTOM);
lb1.setTextFill(Color.RED);
  
```

Tabel 11.2 Konstruktor dan deskripsi Label

Constructor	Deskripsi
Label()	Membuat label kosong
Label(text: String)	Membuat label dengan suatu teks
Label(text: String, graphic: Node)	Membuat label dengan teks dan gambar

Tabel 11.3 Properti dan deskripsi Label

Properti	Deskripsi
Alignment : <Pos>	Menentukan <i>alignment</i> dari text dan node yang diberi label
contentDisplay : <ContentDisplay>	Menentukan posisi node relatif thd teks dengan menggunakan constanta ContentDisplay TOP, BOTTOM, LEFT, dan RIGHT dalam.
Graphic: <Node>	Menentukan gamabar pada label
graphicTextGap: Double	Jarak/Gap antara gambar dan teks
textFill: <Paint>	<paint> yang digunakan untuk mengisi teks
text: String	Teks pada label
underLine: Boolean	Menentukan apakah underline atau tidak
wrapText: Boolean	Menentukan apakah teks dilakukan <i>wrap</i> jika panjang teks melebihi lebar label.

#### Kode 11.18 LabelWithGraphic.java

```

1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.scene.Scene;
4 import javafx.scene.control.ContentDisplay;
5 import javafx.scene.control.Label;
6 import javafx.scene.image.Image;
7 import javafx.scene.image.ImageView;
8 import javafx.scene.layout.HBox;
9 import javafx.scene.layout.StackPane;
10 import javafx.scene.paint.Color;
11 import javafx.scene.shape.Circle;
12 import javafx.scene.shape.Rectangle;
13 import javafx.scene.shape.Ellipse;

```

```
14 public class LabelWithGraphic extends Application {  
15     @Override  
16     public void start(Stage primaryStage) {  
17         ImageView ina = new ImageView(new  
18             Image("image/benderaIna.jpeg"));  
19         ina.setPreserveRatio(true);  
20         ina.setFitWidth(250);  
21         //membuat label dengan teks dan gambar  
22         Label lb1 = new Label("INA\nIndonesia", ina);  
23         lb1.setStyle("-fx-border-color: green; -fx-border-width:  
24             2");  
25         lb1.setContentDisplay(ContentDisplay.BOTTOM);  
26         lb1.setTextFill(Color.RED);  
27         Label lb2 = new Label("Circle", new Circle(50, 50, 25));  
28         lb2.setContentDisplay(ContentDisplay.TOP);  
29         lb2.setTextFill(Color.ORANGE);  
30         Label lb3 = new Label("Rectangle", new Rectangle(10, 10,  
31             50, 25));  
32         lb3.setContentDisplay(ContentDisplay.RIGHT);  
33         Label lb4 = new Label("Ellipse", new Ellipse(50, 50, 50,  
34             25));  
35         lb4.setContentDisplay(ContentDisplay.LEFT);  
36         Ellipse ellipse = new Ellipse(50, 50, 50, 25);  
37         ellipse.setStroke(Color.GREEN);  
38         ellipse.setFill(Color.WHITE);  
39         StackPane stackPane = new StackPane();  
        stackPane.getChildren().addAll(ellipse, new  
Label("JavaFX"));  
        Label lb5 = new Label("pane dalam sebuah label",  
stackPane);  
        lb5.setContentDisplay(ContentDisplay.BOTTOM);  
        HBox pane = new HBox(20);
```

```

40     pane.getChildren().addAll(lb1, lb2, lb3, lb4, lb5);

41

42     Scene scene = new Scene(pane, 750, 300);
43     primaryStage.setTitle("LabelWithGraphic");
44     primaryStage.setScene(scene);
45     primaryStage.show();
46 }

```

### Output



### 11.11.2 Button

Button adalah suatu tombol yang membangkitkan event ketika diclick. JavaFX menyediakan regular buttons, toggle buttons, check box buttons, dan radio buttons.

Tabel 11.4 Konstruktor dan deskripsi Button

Constructor	Deskripsi
Button()	Membuat button kosong
Button(text: String)	Membuat button dengan suatu teks
Button(text: String, graphic: Node)	Membuat button dengan teks dan gambar

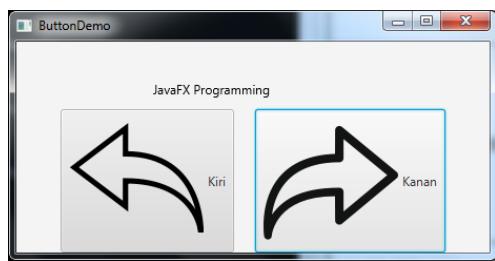
**Kode 11.19 ButtonDemo.java**

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.geometry.Pos;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Button;
6 import javafx.scene.image.ImageView;
7 import javafx.scene.layout.BorderPane;
8 import javafx.scene.layout.HBox;
9 import javafx.scene.layout.Pane;
10 import javafx.scene.text.Text;
11
12 public class ButtonDemo extends Application {
13     protected Text text = new Text(50, 50, "JavaFX Programming");
14     protected BorderPane getPane() {
15         HBox paneForButtons = new HBox(20);
16         Button btLeft = new Button("Kiri", new
17             ImageView("image/panahKiri.png"));
18         Button btRight = new Button("Kanan", new
19             ImageView("image/panahKanan.png"));
20         paneForButtons.getChildren().addAll(btLeft, btRight);
21         paneForButtons.setAlignment(Pos.CENTER);
22         paneForButtons.setStyle("-fx-border-color: green");
23
24         BorderPane pane = new BorderPane();
25         pane.setBottom(paneForButtons);
26
27         Pane paneForText = new Pane();
28         paneForText.getChildren().add(text);
29         pane.setCenter(paneForText);
30         btLeft.setOnAction(e->text.setX(text.getX() - 10));
31         btRight.setOnAction(e -> text.setX(text.getX() + 10));
```

```

30         return pane;
31     }
32     @Override
33     public void start(Stage primaryStage) {
34         Scene scene = new Scene(getPane(), 450, 200);
35         primaryStage.setTitle("ButtonDemo");
36         primaryStage.setScene(scene);
37         primaryStage.show();
38     }
39 }
```

### Output



### 11.11.3 CheckBox

CheckBox digunakan user untuk menentukan pilihan mirip dengan button sebuah checkbox memiliki properti turunan yaitu onAction, text, graphis, dan lain-lain.

Tabel 11.5 Konstruktor dan deskripsi CheckBox

Constructor	Deskripsi
CheckBox()	Membuat checkbox kosong
CheckBox(text: String)	Membuat checkbox dengan suatu teks

Contoh checkbox:

```

CheckBox chkUS = new CheckBox("US");
chkUS.setGraphic(new ImageView("image/usIcon.gif"));
chkUS.setTextFill(Color.GREEN);
chkUS.setContentDisplay(ContentDisplay.LEFT);
```

```
chkUS.setStyle("-fx-border-color: black");
chkUS.setSelected(true);
chkUS.setPadding(new Insets(5, 5, 5, 5));
```

#### Kode 11.20 CheckBoxDemo.java

```
1 import javafx.event.ActionEvent;
2 import javafx.event.EventHandler;
3 import javafx.geometry.Insets;
4 import javafx.scene.control.CheckBox;
5 import javafx.scene.layout.BorderPane;
6 import javafx.scene.layout.VBox;
7 import javafx.scene.text.Font;
8 import javafx.scene.text.FontPosture;
9 import javafx.scene.text.FontWeight;
10 public class CheckBoxDemo extends ButtonDemo {
11     @Override
12     protected BorderPane getPane() {
13         BorderPane pane = super.getPane();
14         Font fontBoldItalic = Font.font("Times New Roman",
15             FontWeight.BOLD, FontPosture.ITALIC, 24);
16         Font fontBold = Font.font("Times New Roman",
17             FontWeight.BOLD, FontPosture.REGULAR, 24);
18         Font fontItalic = Font.font("Times New Roman",
19             FontWeight.NORMAL, FontPosture.ITALIC, 24);
20         Font fontNormal = Font.font("Times New Roman",
21             FontWeight.NORMAL, FontPosture.REGULAR, 24);
22         text.setFont(fontNormal);
23
24         VBox paneForCheckboxes = new VBox(20);
25         paneForCheckboxes.setPadding(new Insets(5, 5, 5, 5));
26         paneForCheckboxes.setStyle("-fx-border-color: green");
27         CheckBox chkBold = new CheckBox("Bold");
```

```

28         CheckBox chkItalic = new CheckBox("Italic");
29
30         paneForCheckBoxes.getChildren().addAll(chkBold,
31             chkItalic);
32
33         pane.setRight(paneForCheckBoxes);
34
35         EventHandler<ActionEvent> handler = e -> {
36
37             if (chkBold.isSelected() && chkItalic.isSelected()) {
38                 text.setFont(fontBoldItalic);
39             }
40             else if (chkBold.isSelected()) {
41                 text.setFont(fontBold);
42             }
43             else if (chkItalic.isSelected()) {
44                 text.setFont(fontItalic);
45             }
46             else {
47                 text.setFont(fontNormal);
48             }
49         };
50
51         chkBold.setOnAction(handler);
52         chkItalic.setOnAction(handler);
53
54         return pane;
55     }
56
57     public static void main(String[] args) {
58
59         launch(args);
60     }
61 }
```

### Output



#### 11.11.4 RadioButton

Radio button dikenal juga sebagai option buttons, digunakan untuk memilih single item dari sekelompok item yang ada.

Tabel 11.6 Konstruktor dan deskripsi RadioButton

Constructor	Deskripsi
RadioButton()	Membuat RadioButton kosong
RadioButton(text: String)	Membuat RadioButton dengan suatu teks

```
RadioButton rbUS = new RadioButton("US");
rbUS.setGraphic(new ImageView("image/usIcon.gif"));
rbUS.setTextFill(Color.GREEN);
rbUS.setContentDisplay(ContentDisplay.LEFT);
rbUS.setStyle("-fx-border-color: black");
rbUS.setSelected(true);
rbUS.setPadding(new Insets(5, 5, 5, 5));
```

mengelompokkan radio button

```
ToggleGroup group = new ToggleGroup();
rbRed.setToggleGroup(group);
rbGreen.setToggleGroup(group);
rbBlue.setToggleGroup(group);
```

#### Kode 11.21 RadioButtonDemo.java

```
1 import javafx.geometry.Insets;
2 import javafx.scene.control.RadioButton;
3 import javafx.scene.control.ToggleGroup;
4 import javafx.scene.layout.BorderPane;
5 import javafx.scene.layout.VBox;
6 import javafx.scene.paint.Color;
```

```
7 public class RadioButtonDemo extends CheckBoxDemo {  
8     @Override  
9     protected BorderPane getPane() {  
10         BorderPane pane = super.getPane();  
11         VBox paneForRadioButtons = new VBox(20);  
12         paneForRadioButtons.setPadding(new Insets(5, 5, 5, 5));  
13         paneForRadioButtons.setStyle  
14             ("-fx-border-width: 2px; -fx-border-color: green");  
15  
16         RadioButton rbRed = new RadioButton("Red");  
17         RadioButton rbGreen = new RadioButton("Green");  
18         RadioButton rbBlue = new RadioButton("Blue");  
19         paneForRadioButtons.getChildren().addAll(rbRed, rbGreen,  
rbBlue);  
20         pane.setLeft(paneForRadioButtons);  
21  
22         ToggleGroup group = new ToggleGroup();  
23         rbRed.setToggleGroup(group);  
24         rbGreen.setToggleGroup(group);  
25         rbBlue.setToggleGroup(group);  
26  
27         rbRed.setOnAction(e -> {  
28             if (rbRed.isSelected()) {  
29                 text.setFill(Color.RED);  
30             }  
31         });  
32  
33         rbGreen.setOnAction(e -> {  
34             if (rbGreen.isSelected()) {  
35                 text.setFill(Color.GREEN);  
36             }  
37         });
```

```

38
39         rbBlue.setOnAction(e -> {
40             if (rbBlue.isSelected()) {
41                 text.setFill(Color.BLUE);
42             }
43         });
44         return pane;
45     }
46
47     public static void main(String[] args) {
48         launch(args);
49     }
50 }
```

### Output



### 11.11.5 TextField

Text field digunakan untuk memasukkan/menampilkan teks. Contoh penggunaan textfield:

```
TextField tfMessage = new TextField("T-Storm");
tfMessage.setEditable(false);
tfMessage.setStyle("-fx-text-fill: red");
tfMessage.setFont(Font.font("Times", 20));
tfMessage.setAlignment(Pos.BASELINE_RIGHT);
```

### Kode 11.22 TextFieldDemo.java

```

1 import javafx.geometry.Insets;
2 import javafx.geometry.Pos;
3 import javafx.scene.control.Label;
```

```

4 import javafx.scene.control.TextField;
5 import javafx.scene.layout.BorderPane;
6 public class TextFieldDemo extends RadioButtonDemo {
7     @Override
8     protected BorderPane getPane() {
9         BorderPane pane = super.getPane();
10
11         BorderPane paneForTextField = new BorderPane();
12         paneForTextField.setPadding(new Insets(5, 5, 5, 5));
13         paneForTextField.setStyle("-fx-border-color: green");
14         paneForTextField.setLeft(new Label("Enter a new message:");
15
16         TextField tf = new TextField();
17         tf.setAlignment(Pos.BOTTOM_RIGHT);
18         paneForTextField.setCenter(tf);
19         pane.setTop(paneForTextField);
20
21         tf.setOnAction(e -> text.setText(tf.getText()));
22
23         return pane;
24     }
25     public static void main(String[] args) {
26         launch(args);
27     }
28 }
```

## Output



## 11.11.6 TextArea

TextArea digunakan untuk mengisikan teks panjang multiline. Contoh:

```
TextArea taNote = new TextArea("This is a text area");
taNote.setPrefColumnCount(20);
taNote.setPrefRowCount(5);
taNote.setWrapText(true);
taNote.setStyle("-fx-text-fill: red");
taNote.setFont(Font.font("Times", 20));
```

**Kode 11.23 DescriptionPane.java**

```
1 import javafx.geometry.Insets;
2 import javafx.scene.control.Label;
3 import javafx.scene.control.ContentDisplay;
4 import javafx.scene.control.ScrollPane;
5 import javafx.scene.control.TextArea;
6 import javafx.scene.image.ImageView;
7 import javafx.scene.layout.BorderPane;
8 import javafx.scene.text.Font;
9 public class DescriptionPane extends BorderPane {
10     /** Label utk menampilkan image dan title */
11     private Label lblImageTitle = new Label();
12     /** Text area utk menampilkan text */
13     private TextArea taDescription = new TextArea();
14     public DescriptionPane() {
15         // tengahkan icon dan teks dan menempatkan teks dibawah
16         // icon
17         lblImageTitle.setContentDisplay(ContentDisplay.TOP);
18         lblImageTitle.setPrefSize(200, 100);
19         // Set font pada label dan text field
```

```
20         lblImageTitle.setFont(new Font("SansSerif", 16));
21         taDescription.setFont(new Font("Serif", 14));
22
23         taDescription.setWrapText(true);
24         taDescription.setEditable(false);
25
26         // buat scroll pane utk text area
27         ScrollPane scrollPane = new ScrollPane(taDescription);
28
29         // tempatkan label dan scroll pane dalam border pane
30         setLeft(lblImageTitle);
31         setCenter(scrollPane);
32         setPadding(new Insets(5, 5, 5, 5));
33     }
34
35     /** Set title */
36     public void setTitle(String title) {
37         lblImageTitle.setText(title);
38     }
39
40     /** Set image view */
41     public void setImageView(ImageView icon) {
42         lblImageTitle.setGraphic(icon);
43         //ina.setPreserveRatio(true);
44         //ina.setFitWidth(250);
45     }
46
47     /** Set text description */
48     public void setDescription(String text) {
49         taDescription.setText(text);
50     }
51 }
```

**Kode 11.24 TextAreaDemo.java**

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.scene.Scene;
4 import javafx.scene.image.ImageView;
5
6 public class TextAreaDemo extends Application {
7     @Override
8     public void start(Stage primaryStage) {
9         // membuat description pane
10        DescriptionPane descriptionPane = new DescriptionPane();
11        // Set title, text, and image in the description pane
12        descriptionPane.setTitle("Indonesia");
13        String description = "Bendera Merah Putih ... ";
14        descriptionPane.setImageView(new
15            ImageView("image/benderaIna2.png"));
16        descriptionPane.setDescription(description);
17        // buat scene dan letakkan pada stage
18        Scene scene = new Scene(descriptionPane, 500, 300);
19        primaryStage.setTitle("TextAreaDemo");
20        primaryStage.setScene(scene);
21        primaryStage.show();
22    }
23 }
```

**Ouput**

### 11.11.7 ComboBox

Combo box atau daftar pilihan atau drop-down list berisi daftar item yang dapat dipilih oleh user. Contoh:

```
ComboBox<String> cbo = new ComboBox<>();  
cbo.getItems().addAll("Item 1", "Item 2", "Item 3", "Item 4");  
cbo.setStyle("-fx-color: red");  
cbo.setValue("Item 1");
```

**Kode 11.25 ComboBoxDemo.java**

```
1 import javafx.application.Application;  
2 import javafx.stage.Stage;  
3 import javafx.collections.FXCollections;  
4 import javafx.collections.ObservableList;  
5 import javafx.scene.Scene;  
6 import javafx.scene.control.ComboBox;  
7 import javafx.scene.control.Label;  
8 import javafx.scene.image.ImageView;  
9 import javafx.scene.layout.BorderPane;  
10  
11 public class ComboBoxDemo extends Application {  
12     // deklarasi array Strings utk judul bendera  
13     private String[] flagTitles = {"Indonesia", "Kamboja", "Malasya",  
14         "Philipina", "Singapur", "Thailand", "Vietnam"};  
15     // deklarasi array ImageView utk bendera 9 negara  
16     private ImageView[] flagImage = {new  
17         ImageView("image/benderaIna2.png"),  
18         new ImageView("image/benderaKamb.jpg"),  
19         new ImageView("image/benderaMalasya.png"),  
20         new ImageView("image/benderaphil.jpg"),  
21         new ImageView("image/benderaSing.jpg"),  
22         new ImageView("image/benderathai.jpg"),  
23         new ImageView("image/benderaViet.jpg")  
24 }
```

```

22    };
23
24    private String[] flagDescription = new String[7];
25    private DescriptionPane descriptionPane = new DescriptionPane();
26    // buat combo box utk memilih negara
27    private ComboBox<String> cbo = new ComboBox<>();
28
29    @Override
30    public void start(Stage primaryStage) {
31        // Set text description
32        flagDescription[0] = "Bendera Indonesia ... ";
33        flagDescription[1] = "Description for Kamboja ... ";
34        flagDescription[2] = "Description for Malasya ... ";
35        flagDescription[3] = "Description for Philipina ... ";
36        flagDescription[4] = "Description for singapura ... ";
37        flagDescription[5] = "Description for Thailand ... ";
38        flagDescription[6] = "Description for Vietnam ... ";
39
40        // Set the first country (Ina) for display
41        setDisplay(0);
42
43        pane
44            BorderPane pane = new BorderPane();
45
46            BorderPane paneForComboBox = new BorderPane();
47            paneForComboBox.setLeft(new Label("pilih negara: "));
48            paneForComboBox.setCenter(cbo);
49            pane.setTop(paneForComboBox);
50            cbo.setPrefWidth(400);
51            cbo.setValue("Indonesia");
52
53            ObservableList<String> items =
54            FXCollections.observableArrayList(flagTitles);

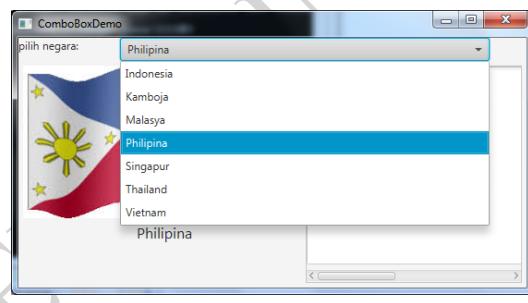
```

```

51         cbo.getItems().addAll(items);
52         pane.setCenter(descriptionPane);
53
54         // tampilkan negara terpilih
55         cbo.setOnAction(e ->
56             setDisplay(items.indexOf(cbo.getValue())));
57
58         // buat scene dan letakkan pada stage
59         Scene scene = new Scene(pane, 550, 270);
60         primaryStage.setTitle("ComboBoxDemo");
61         primaryStage.setScene(scene);
62         primaryStage.show();
63     }
64
65     /** Set display informasi pada description pane */
66     public void setDisplay(int index) {
67         descriptionPane.setTitle(flagTitles[index]);
68         descriptionPane.setImageView(flagImage[index]);
69         descriptionPane.setDescription(flagDescription[index]);
70     }

```

### Output



### 11.11.8 ListView

ListView merupakan kontrol yang pada dasarnya sama dengan combobox namun membolehkan user untuk memilih sebuah item ataupun beberapa item. Contoh:

```

ObservableList<String> items = FXCollections.observableArrayList("Item 1", "Item 2", "Item
3", "Item 4", "Item 5", "Item 6");
ListView<String> lv = new ListView<>(items);
lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
// menambahkan suatu listener pada property untuk menangani perubahan nilai pada properti:
lv.getSelectionModel().selectedItemProperty().addListener(ov -> {
    System.out.println("Selected indices: " + lv.getSelectionModel().getSelectedIndices());
    System.out.println("Selected items: " + lv.getSelectionModel().getSelectedItems());
});

```

### Kode 11.26 ListViewDemo.java

```

1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.collections.FXCollections;
4 import javafx.scene.Scene;
5 import javafx.scene.control.ListView;
6 import javafx.scene.control.ScrollPane;
7 import javafx.scene.control.SelectionMode;
8 import javafx.scene.image.ImageView;
9 import javafx.scene.layout.BorderPane;
10 import javafx.scene.layout.FlowPane;
11
12 public class ListViewDemo extends Application {
13
14     private String[] flagTitles = {"Indonesia", "Kamboja",
15     "Malasya", "Philipina", "Singapur", "Thailand", "Vietnam"};
16
17     private ImageView [] flagImage = {
18         new ImageView("image/benderaIna2.png"),
19         new ImageView("image/benderaKamb.jpg"),
20         new ImageView("image/benderaMalasya.png"),
21         new ImageView("image/benderaphil.jpg"),
22     }
23
24     @Override
25     public void start(Stage stage) {
26         BorderPane root = new BorderPane();
27         Scene scene = new Scene(root, 400, 300);
28         stage.setScene(scene);
29         stage.show();
30     }
31
32     public static void main(String[] args) {
33         Application.launch(ListViewDemo.class, args);
34     }
35 }

```

```

21     new ImageView("image/benderaSing.jpg"),
22     new ImageView("image/benderathai.jpg"),
23     new ImageView("image/benderaViet.jpg")
24   };
25
26   @Override
27   public void start(Stage primaryStage) {
28     ListView<String> lv = new ListView<>
29       (FXCollections.observableArrayList(flagTitles));
30     lv.setPrefSize(400, 400);
31     lv.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
32     // buat pane utk image views
33     FlowPane imagePane = new FlowPane(10, 10);
34     BorderPane pane = new BorderPane();
35     pane.setLeft(new ScrollPane(lv));
36     pane.setCenter(imagePane);
37     lv.getSelectionModel().selectedItemProperty().addListener(
38       ov -> {
39       imagePane.getChildren().clear();
40       for (Integer i:
41         lv.getSelectionModel().getSelectedIndices()) {
42         imagePane.getChildren().add(flagImage[i]);
43       }
44     });
45     Scene scene = new Scene(pane, 850, 370);
46     primaryStage.setTitle("ListViewDemo");
47     primaryStage.setScene(scene);
48     primaryStage.show();
49   }

```

### Output



### 11.11.9 ScrollBar

ScrollBar merupakan control yang digunakan untuk memilih suatu range nilai.

contoh:

```
ScrollBar sb = new ScrollBar();
sb.valueProperty().addListener(ov -> {
    System.out.println("old value: " + oldVal);
    System.out.println("new value: " + newVal);
});
```

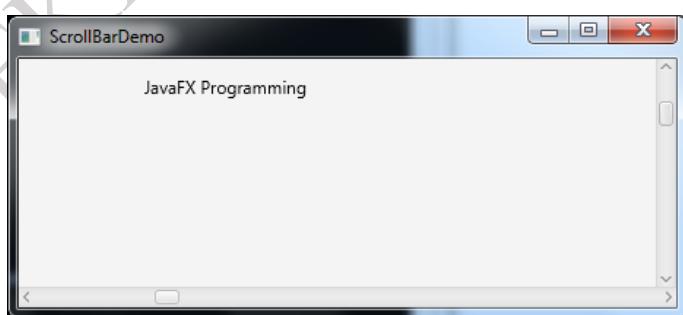
#### Kode 11.27 ScrollBarDemo.java

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.geometry.Orientation;
4 import javafx.scene.Scene;
5 import javafx.scene.control.ScrollBar;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.Pane;
8 import javafx.scene.text.Text;
9 public class ScrollBarDemo extends Application {
10     @Override
11     public void start(Stage primaryStage) {
12         Text text = new Text(20, 20, "JavaFX Programming");
13     }
}
```

```

14     ScrollBar sbHorizontal = new ScrollBar();
15     ScrollBar sbVertical = new ScrollBar();
16     sbVertical.setOrientation(Orientation.VERTICAL);
17
18     Pane paneForText = new Pane();
19     paneForText.getChildren().add(text);
20
21     BorderPane pane = new BorderPane();
22     pane.setCenter(paneForText);
23     pane.setBottom(sbHorizontal);
24     pane.setRight(sbVertical);
25     // Listener utk horizontal scroll bar value change
26     sbHorizontal.valueProperty().addListener(ov ->
27         text.setX(sbHorizontal.getValue() * paneForText.getWidth() /
28             sbHorizontal.getMax()));
29     // Listener utk vertical scroll bar value change
30     sbVertical.valueProperty().addListener(ov ->
31         text.setY(sbVertical.getValue() * paneForText.getHeight() /
32             sbVertical.getMax()));
33     Scene scene = new Scene(pane, 450, 170);
34     primaryStage.setTitle("ScrollBarDemo");
35     primaryStage.setScene(scene);
36     primaryStage.show();
37 }
38 }
```

### Output



### 11.11.20 Slider

Slider sama dengan ScrollBar, namun Slider memiliki lebih banyak properties dan memiliki lebih banyak variasi tampilan. Contoh:

```
Slider slHorizontal = new Slider();
slHorizontal.setShowTickLabels(true);
slHorizontal.setShowTickMarks(true);
```

**Kode 11. 28 SliderDemo.java**

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.geometry.Orientation;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Slider;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.Pane;
8 import javafx.scene.text.Text;
9
10 public class SliderDemo extends Application {
11     @Override
12     public void start(Stage primaryStage) {
13         Text text = new Text(20, 20, "JavaFX Programming");
14         Slider slHorizontal = new Slider();
15         slHorizontal.setShowTickLabels(true);
16         slHorizontal.setShowTickMarks(true);
17
18         Slider slVertical = new Slider();
19         slVertical.setOrientation(Orientation.VERTICAL);
20         slVertical.setShowTickLabels(true);
21         slVertical.setShowTickMarks(true);
```

```

21     slVertical.setValue(100);
22
23     // buat sebuah teks dlm pane
24     Pane paneForText = new Pane();
25     paneForText.getChildren().add(text);
26
27     // buat border pane utk text dan scroll bars
28     BorderPane pane = new BorderPane();
29     pane.setCenter(paneForText);
30     pane.setBottom(slHorizontal);
31     pane.setRight(slVertical);
32
33     slHorizontal.valueProperty().addListener(ov ->
34         text.setX(slHorizontal.getValue() * paneForText.getWidth() / slHorizontal.getMax());
35
36         slVertical.valueProperty().addListener(ov ->
37             text.setY((slVertical.getMax() - slVertical.getValue()) * paneForText.getHeight() / slVertical.getMax()));
38
39     // buat scene dan letakkan pada stage
40     Scene scene = new Scene(pane, 450, 170);
41     primaryStage.setTitle("SliderDemo");
42     primaryStage.setScene(scene);
43     primaryStage.show();
44 }

```

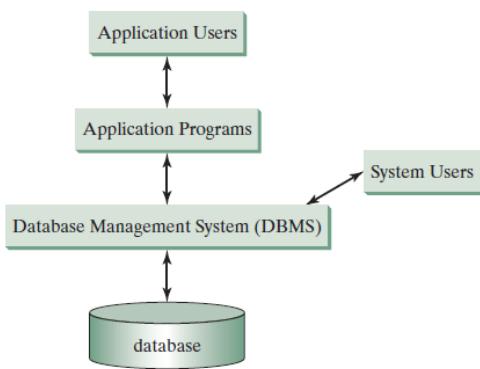
## Output



## BAB 12 KONSEP DATABASE DAN DBMS

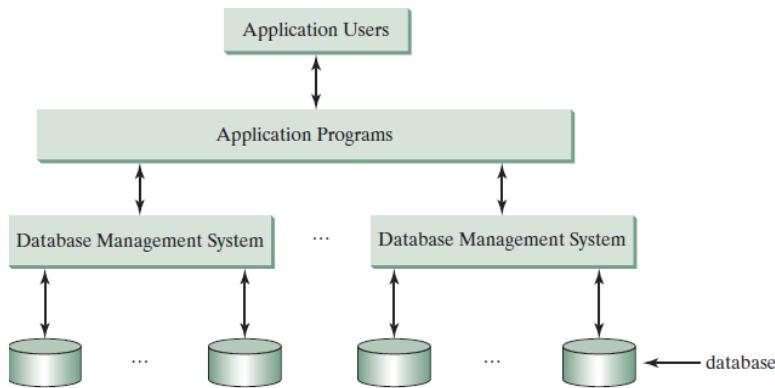
Sistem database sudah digunakan di banyak tempat, sebagai contoh kartu rencana studi mahasiswa, nilai mahasiswa, pembayaran mahasiswa di simpan di dalam suatu database. Ketika belanja online pemesanan barang yang kita lakukan di simpan di dalam database. Sistem database tidak hanya menyimpan data saja, namun juga menyediakan akses terhadap data dalam melakukan update, manipulasi, dan analisis terhadap data.

Sistem database terdiri dari data, software yang melakukan penyimpanan dan manajemen data dalam suatu database, dan program aplikasi untuk menyajikan data dan membuat user dapat berinteraksi menggunakan sistem database.



Gambar 21.1 Sistem database terdiri dari data, DBMS dan program aplikasi.

Database merupakan tempat penyimpanan data(gudang data) yang menghasilkan informasi. Contoh Sistem database diantaranya adalah MySQL, Oracle, IBM's DB2, Informix, MS-SQL Server, SQLite, Sybase. Ketika kita membeli suatu sistem database dari vendor, pada dasarnya kita membeli software DBMS. DBMS dibuat untuk digunakan oleh programmer profesional dan tidak ditujukan untuk pemakai biasa. namun pemakai dapat menggunakan suatu program aplikasi yang dibuat di atas DBMS, sehingga program aplikasi merupakan interface yang menghubungkan antara pemakai biasa dan sistem database. Bentuk program aplikasi dapat berupa aplikasi GUI desktop atau aplikasi web.



Gambar 12.2 Program aplikasi dapat mengkases banyak DBMS

## 12.1 Model data relasional

Saat ini sistem database yang banyak digunakan adalah sistem database relational, yang berbasis pada model data relasional, yang memiliki tiga komponen yaitu struktur,integritas,dan bahasa. struktur mewakili bentuk dari data,integritas menentukan batasan-batasan pada data,dan bahasa menyediakan akses dan manipulasi thd data.

Relation/Table Name	Columns/Attributes				
Course Table	courseId	subjectId	courseNumber	title	numOfCredits
Tuples/ Rows	11111	CSCI	1301	Introduction to Java I	4
	11112	CSCI	1302	Introduction to Java II	3
	11113	CSCI	3720	Database Systems	3
	11114	CSCI	4750	Rapid Java Application	3
	11115	MATH	2750	Calculus I	5
	11116	MATH	3750	Calculus II	5
	11117	EDUC	1111	Reading	3
	11118	ITEC	1344	Database Administration	3

Gambar 12.3 Suatu table memiliki nama table, nama kolom dan baris

Student Table									
ssn	firstName	mi	lastName	phone	birthDate	street	zipCode	deptID	
444111110	Jacob	R	Smith	9129219434	1985-04-09	99	Kingston Street	31435	BIOC
444111111	John	K	Stevenson	9129219434	null	100	Main Street	31411	BIOC
444111112	George	K	Smith	9129213454	1974-10-10	1200	Abercorn St.	31419	CS
444111113	Frank	E	Jones	9125919434	1970-09-09	100	Main Street	31411	BIOC
444111114	Jean	K	Smith	9129219434	1970-02-09	100	Main Street	31411	CHEM
444111115	Josh	R	Woo	7075989434	1970-02-09	555	Franklin St.	31411	CHEM
444111116	Josh	R	Smith	9129219434	1973-02-09	100	Main Street	31411	BIOC
444111117	Joy	P	Kennedy	9129229434	1974-03-19	103	Bay Street	31412	CS
444111118	Toni	R	Peterson	9129229434	1964-04-29	103	Bay Street	31412	MATH
444111119	Patrick	R	Stoneman	9129229434	1969-04-29	101	Washington St.	31435	MATH
444111120	Rick	R	Carter	9125919434	1986-04-09	19	West Ford St.	31411	BIOC

Gambar 12.4 tabel student menyimpan informasi tentang student

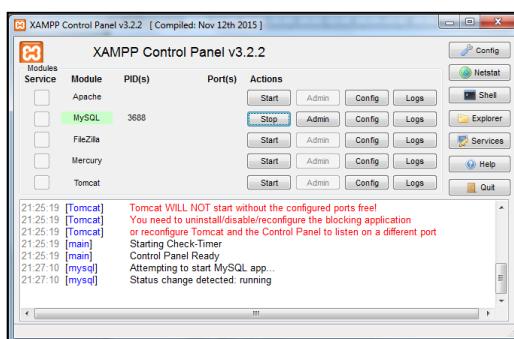
Enrollment Table				
	ssn	courseId	dateRegistered	grade
444111110	11111	2004-03-19	A	
444111110	11112	2004-03-19	B	
444111110	11113	2004-03-19	C	
444111111	11111	2004-03-19	D	
444111111	11112	2004-03-19	F	
444111111	11113	2004-03-19	A	
444111112	11114	2004-03-19	B	
444111112	11115	2004-03-19	C	
444111112	11116	2004-03-19	D	
444111113	11111	2004-03-19	A	
444111113	11113	2004-03-19	A	
444111114	11115	2004-03-19	B	
444111115	11115	2004-03-19	F	
444111115	11116	2004-03-19	F	
444111116	11111	2004-03-19	D	
444111117	11111	2004-03-19	D	
444111118	11111	2004-03-19	A	
444111118	11112	2004-03-19	D	
444111118	11113	2004-03-19	B	

Gambar 12.5 Tabel Enrollment menyimpan informasi tentang Enrollment dari student.

Integritas (integrity constraint) merupakan kondisi legal pada suatu tabel yang harus dipenuhi, dimana ada tiga jenis constraint yaitu: domain constraint, primary key constraint, foreign key constraint.

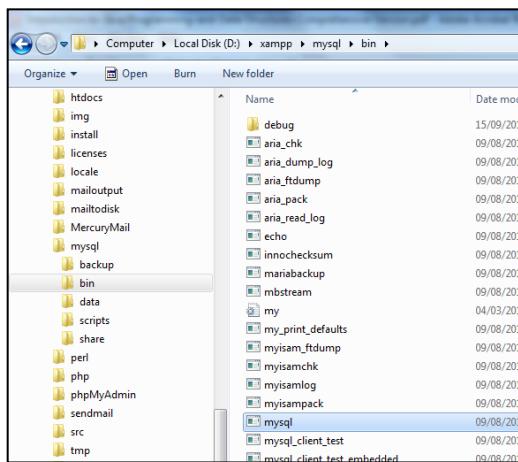
## 12.2 SQL

SQL merupakan bahasa untuk mengakses sistem database relasional. program aplikasi harus menggunakan SQL untuk dapat mengakses database. create, drop, table, retrieve, modifikasi data. MySQL adalah salah satu software DBMS, untuk memasang MySQL pada komputer kita , kita dapat menggunakan XAMPP , yang merupakan software aplikasi gratis untuk aplikasi database mysql, web server, dan mail server.



Gambar 12.6 Tampilan aplikasi XAMPP

Lokasi file mysql pada folder xampp



Gambar 12.7 Folder xampp

### 12.3 Mengakses dan memanipulasi database

Menggunakan commandtext untuk mengakses mysql, untuk login ke dalam database menggunakan perintah:

D:\xampp\mysql\bin\MySQL -root -p

(password dikosongkan kemudian enter)

```
D:\>cd D:\xampp\mysql\bin
D:\xampp\mysql\bin>mysql -uroot -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.1.26-MariaDB mariadb.org binary distribution
Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]>
```

Gambar 12.8 Command windows untuk mengakses database MySQL

Berikut beberapa perintah sql untuk akses dan manipulasi database :

show databases; untuk melihat daftar database,

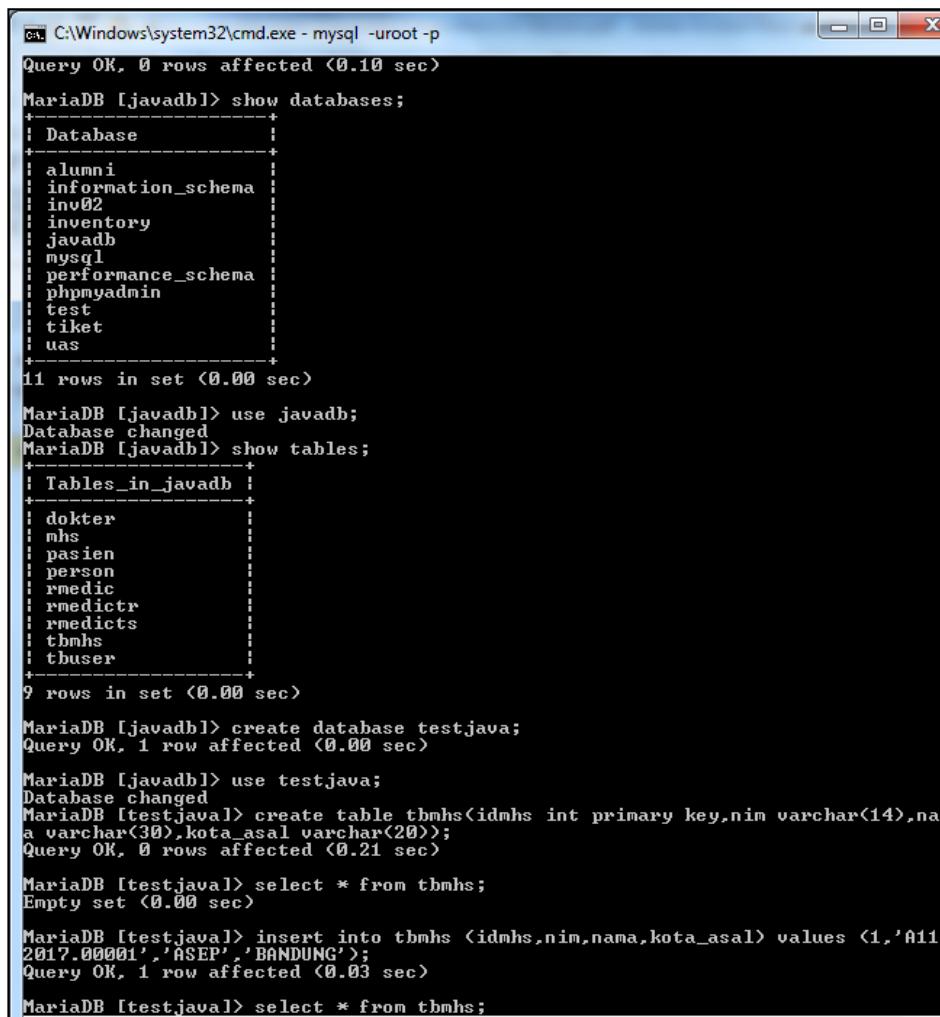
use dbjava; untuk masuk ke dalam database "dbjava"

show tables; untuk melihat daftar table,

select \* from tbmhs; untuk melihat isi dari tabel "tbmhs".

insert into tbmhs (idmhs, nim, nama, kota\_asal) values (1, 'A11.2017.00001', 'BUDI', 'SEMARANG'); untuk mengisikan record baru ke dalam tabel tbmhs.

Eksekusi query menggunakan command text:



```
C:\Windows\system32\cmd.exe - mysql -uroot -p
Query OK, 0 rows affected (0.10 sec)
MariaDB [javadb]> show databases;
+-----+
| Database |
+-----+
| alumni   |
| information_schema |
| inv02    |
| inventory |
| javadb   |
| mysql    |
| performance_schema |
| phpmyadmin |
| test     |
| tiket   |
| uas      |
+-----+
11 rows in set (0.00 sec)

MariaDB [javadb]> use javadb;
Database changed
MariaDB [javadb]> show tables;
+-----+
| Tables_in_javadb |
+-----+
| dokter  |
| mhs    |
| pasien  |
| person  |
| rmedic  |
| rmedictr |
| rmedicts |
| tbmhs   |
| tbuser  |
+-----+
9 rows in set (0.00 sec)

MariaDB [javadb]> create database testjava;
Query OK, 1 row affected (0.00 sec)

MariaDB [javadb]> use testjava;
Database changed
MariaDB [testjava]> create table tbmhs(idmhs int primary key,nim varchar<14>,na
a varchar<30>,kota_asal varchar<20>);
Query OK, 0 rows affected (0.21 sec)

MariaDB [testjava]> select * from tbmhs;
Empty set (0.00 sec)

MariaDB [testjava]> insert into tbmhs (idmhs,nim,nama,kota_asal) values (1,'A11
2017.00001','ASEP','BANDUNG');
Query OK, 1 row affected (0.03 sec)

MariaDB [testjava]> select * from tbmhs;
```

Gambar 12.9 Eksekusi query menggunakan command text

Menggunakan aplikasi GUI untuk mengakses MySQL

SQLYog merupakan program aplikasi mode GUI untuk me-manage database mysql dimana untuk memanage database akan lebih mudah menggunakan aplikasi ini dibandingkan dengan command text.

Mengeksekusi query pada SQLYog.

The screenshot shows the SQLyog Free MySQL GUI interface. On the left, the database tree shows various databases like 'root@localhost', 'alumni', 'dbjava', etc. The main area has two tabs: 'Query' and 'Query'. The 'Query' tab contains the following SQL code:

```
1 create user 'nina'@'localhost'%' identified by '123';
2 grant all privileges on *.* to 'nina'@'localhost' identified by '123'
3
4 create database dbjava;
5 show databases;
6 use dbjava;
7 create table tbmhs (
8     idmhs int primary key,
9     nim varchar(14),
10    nama varchar(30),
11    kota_asal varchar(20)
12 );
13 select * from tbmhs;
14 insert into tbmhs (idmhs,nim,nama,kota_asal)
15 values (1,'A11.2017.00001','AMIR','SEMARANG');
16 insert into tbmhs (idmhs,nim,nama,kota_asal)
17 values (2,'A11.2017.00002','TARI','SOLO');
18 select * from tbmhs;
19
```

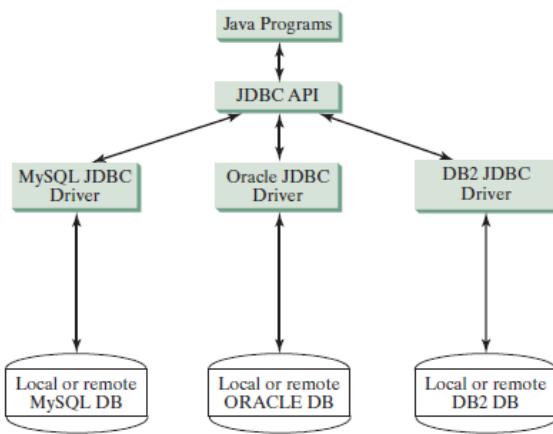
The bottom part of the interface shows the 'Result' tab with the following data:

	idmhs	nim	nama	kota_asal
1	A11.2017.00001	AMIR	SEMARANG	
2	A11.2017.00002	TARI	SOLO	

Gambar 12.10 eksekusi query membuat user,membuat database,membuat table,mengisi data pada tabel,dan menampilkan isi tabel.

## BAB 13 APLIKASI CRUD DENGAN JDBC

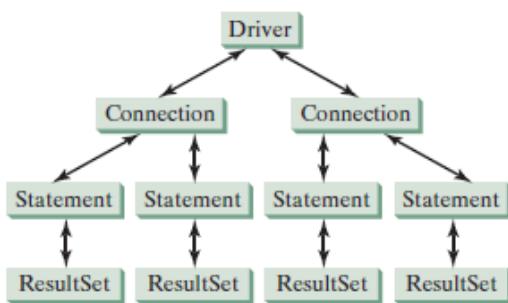
JDBC merupakan API java untuk membuat aplikasi database. Dengan menggunakan API JDBC suatu aplikasi dapat mengeksekusi query SQL, menampilkan hasil, dan menyajikan data ke dalam bentuk yang user-friendly.



Gambar 13.1 Program Java yang mengakses dan memanipulasi database menggunakan JDBC Driver

### 13.1 Membuat aplikasi database menggunakan JDBC

Kunci utama membuat aplikasi database adalah Connection, Statement, dan Resultset.



Gambar 13.2 Class-class JDBC membuat program dapat terkoneksi dengan database, mengirim statement SQL dan memproses result/hasil.

## 13.2 Koneksi ke database

Untuk membuat koneksi ke database menggunakan method `getConnection` pada class `Connnection`.

`Connection cn=DriverManager.getConnection(url,user,password);` Pada method `getConnection` harus ditentukan url yang berupa nama jdbc drivier, lokasi server,dan nama database serta user dan password.

*Contoh:*

```
Connection cn =  
DriverManager.getConnection("jdbc:mysql://localhost/dbjava","nina","123");
```

## 13.3 Membuat statement

Statement digunakan untuk memanipulasi data dengan membuat perintah *SQL*, kemudian melakukan eksekusi. Ada dua interface *statement* yaitu, *Statement* dan *PreparedStatement*. Interface *Statement* digunakan untuk eksekusi terhadap statement sql yang bersifat static, sementara *Preparestatement* merupakan extends dari *Statement*, yang digunakan untuk eksekusi statement sql secara dinamis(precompile) yaitu dengan parameter(atau tanpa parameter), penggunaan *PrepareStatement* lebih efisien utk eksekusi yg berulang.

Contoh perintah sql menggunakan interface Statetement.

```
Statement st = cn.createStatement();  
  
String sql = "insert into tbmhs(idmhs, nim, nama, kota_asal) values (3,  
'A11.2017.00003', 'BUDI', 'SALATIGA')";  
st.executeUpdate(sql);
```

Contoh perintah sql menggunakan Preparedstatatement.

```
Connection cn=MySQLDB.getConnection();  
  
String sql="insert into tbmhs(idmhs,nim,nama,kota_asal) values (?,?,?,?,?)";  
PreparedStatement pst = cn.prepareStatement(sql);  
pst.setInt(1,7);  
pst.setString(2,"A11.2018.90001");
```

```

pst.setString(3, "UMAR");
pst.setString(4, "SEMARANG");
pst.executeUpdate();
System.out.println("Insert berhasil ...");

```

### 13.4 Memproses ResultSet

Resultset menampung tabel yang sedang diakses, dimana posisi awal baris berada di posisi null. Sedangkan untuk berpindah ke baris berikutnya menggunakan method next. Contoh memproses resulset:

```

cn=DriverManager.getConnection("jdbc:mysql://localhost/dbjava","nina","123");
System.out.println("Koneksi Berhasil");
//membuat statement
Statement st = cn.createStatement();
String sql="select * from tbmhs";
ResultSet rs = st.executeQuery(sql);
rs.beforeFirst();
while (rs.next()){
    System.out.println(rs.getString(1));
}

```

### 13.5 Contoh Aplikasi CRUD

Contoh program koneksi

<b>Kode 13.1 MySQLDB.java</b>
-------------------------------

1	import java.sql.*;
2	public class MySQLDB {
3	static Connection cn ;
4	public static Connection getConnection(){
5	try{
6	cn =
	DriverManager.getConnection("jdbc:mysql://localhost/dbjava", "nina",

```

1      "123");
2      System.out.println("Koneksi berhasil");
3      }catch(SQLException se){
4          se.printStackTrace();
5      }
6      return cn;
7  }
8
9  public static void main(String[] args){
10     MySQLDB.getConnection();
11 }
12 }
13
14 }
```

### Output

```

C:\Windows\system32\cmd.exe
E:\pbo\jdbc\modul>javac MySQLDB.java
E:\pbo\jdbc\modul>java MySQLDB
Koneksi berhasil
E:\pbo\jdbc\modul>
```

Contoh program dengan statement insert,update,delete

### Kode 13.2 MySQLDB.java

```

1 import java.sql.*;
2 public class MySQLDB {
3     static Connection cn ;
4     public static Connection getConnection(){
5         if (cn==null){
6             try{
7                 cn =
8                     DriverManager.getConnection("jdbc:mysql://localhost/dbjava", "nina",
9                     "123");
10            System.out.println("Koneksi berhasil");
11        }catch(SQLException se){
12            se.printStackTrace(); }}
```

```
11    }
12        return cn;
13    }
14 }
```

### Kode 13.3 TestDB.java

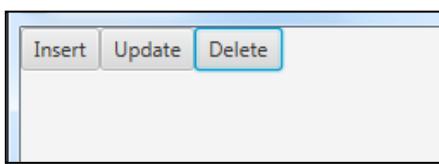
```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.scene.Scene;
4 import javafx.scene.layout.HBox;
5 import javafx.scene.control.Button;
6 import java.sql.*;
7 public class TestDB extends Application {
8     @Override
9     public void start(Stage s){
10         HBox hb= new HBox();
11         Button btInsert = new Button("Insert");
12         Button btUpdate = new Button("Update");
13         Button btDel = new Button("Delete");
14         btInsert.setOnAction(e->{
15             try{
16                 Connection cn=MySQLDB.getConnection();
17                 Statement st = cn.createStatement();
18                 String sql="insert into tbmhs(idmhs, nim, nama,
19 kota_asal) values ('6','A11.2017.10111','ARI','KUDUS')";
20                 st.executeUpdate(sql);
21                 System.out.println("Insert berhasil ...");
22             }catch(SQLException se){
23                 se.printStackTrace();
24             }
25         });
26         btUpdate.setOnAction(e->{
```

```

27     try{
28
29         Connection cn=MySQLDB.getConnection();
30
31         Statement st = cn.createStatement();
32
33         String sql="Update tbmhs set nama='HERU' where
34
35         idmhs=1";
36
37         st.executeUpdate(sql);
38
39         System.out.println("UPDATE berhasil ...");
40
41     }catch(SQLException se){
42
43         se.printStackTrace();  }
44
45     });
46
47     hb.getChildren().addAll(btInsert,btUpdate,btDel);
48
49     Scene sc = new Scene(hb,300,200);
50
51     s.setTitle("Test Database");
52
53     s.setScene(sc);
54
55     s.show();
56
57 }

```

### Output



Contoh program dengan PreparedStatement insert,update,delete

**Kode 13.4 TestDB2.java**

```
1 import javafx.application.Application;
2 import javafx.stage.Stage;
3 import javafx.scene.Scene;
4 import javafx.scene.layout.HBox;
5 import javafx.scene.control.Button;
6 import java.sql.*;
7 public class TestDB2 extends Application {
8     @Override
9     public void start(Stage s){
10         HBox hb= new HBox();
11         Button btInsert = new Button("Insert");
12         Button btUpdate = new Button("Update");
13         Button btDel = new Button("Delete");
14         btInsert.setOnAction(e->{
15             try{
16                 Connection cn=MySQLDB.getConnection();
17                 String sql="insert into tbmhs(idmhs,nim,nama,kota_asal)
values (?,?,?,?,?)";
18                 PreparedStatement pst = cn.prepareStatement(sql);
19                 pst.setInt(1,7);
20                 pst.setString(2,"A11.2018.90001");
21                 pst.setString(3,"UMAR");
22                 pst.setString(4,"SEMARANG");
23                 pst.executeUpdate()
24                 System.out.println("Insert berhasil ...");
25             }catch(SQLException se){
26                 se.printStackTrace();
27             }
}
```

```

28 });
29 btUpdate.setOnAction(e->{
30     try{
31         Connection cn=MySQLDB.getConnection();
32         String sql="Update tbmhs set nama='HERU' where idmhs=?";
33         PreparedStatement pst = cn.prepareStatement(sql);
34         pst.setInt(1,1);
35         pst.executeUpdate();
36         System.out.println("Update berhasil ...");
37     }catch(SQLException se){
38         se.printStackTrace();
39     }
40 });
41 btDel.setOnAction(e->{
42     try{
43         Connection cn=MySQLDB.getConnection();
44         String sql="delete from tbmhs where idmhs=?";
45         PreparedStatement pst = cn.prepareStatement(sql);
46
47         pst.setInt(1,7);
48         pst.executeUpdate();
49         System.out.println("Delete berhasil ...");
50     }catch(SQLException se){
51         se.printStackTrace();
52     }
53 });
54 hb.getChildren().addAll(btInsert,btUpdate,btDel);
55 Scene sc = new Scene(hb,300,200);
56 sc.setTitle("Test Database");
57 sc.setScene(sc);
58 sc.show();
59 }

```

Memproses ResultSet dengan menampilkan di tabelview

**Kode 13. 5 MHS.java**

```
1 public class MHS {  
2     private int idmhs;  
3     private String nim;  
4     private String nama;  
5     private String kota_asal;  
6  
7     @Override  
8     public String toString() {  
9         return "MHSM{" + "idmhs=" + idmhs + ", nim=" + nim + ", nama=" +  
+ nama + ", kota_asal=" + kota_asal + '}';  
10    }  
11    public MHS(){  
12    }  
13    public int getIdmhs(){  
14        return idmhs;  
15    }  
16    public void setIdmhs(int idmhs) {  
17        this.idmhs = idmhs;  
18    }  
19    public String getNim() {  
20        return nim;  
21    }  
22    public void setNim(String nim) {  
23        this.nim = nim;  
24    }  
25}
```

```
26     public String getNama() {  
27         return nama;  
28     }  
29  
30     public void setNama(String nama) {  
31         this.nama = nama;  
32     }  
33  
34     public String getKota_asal() {  
35         return kota_asal;  
36     }  
37  
38     public void setKota_asal(String kota_asal) {  
39         this.kota_asal = kota_asal;  
40     }  
41 }
```

#### Kode 13.6 TestDB3.java

```
1 import javafx.application.Application;  
2 import javafx.stage.Stage;  
3 import javafx.scene.Scene;  
4 import javafx.scene.layout.HBox;  
5 import javafx.scene.control.TableView;  
6 import java.sql.*;  
7 import javafx.collections.FXCollections;  
8 import javafx.collections.ObservableList;  
9 import javafx.geometry.Pos;  
10 import javafx.scene.control.TableColumn;  
11 import javafx.scene.control.cell.PropertyValueFact  
12  
13 public class TestDB3 extends Application {  
14     @Override  
15     public void start(Stage s){
```

```

16    Scene sc = new Scene(getHBTabel(),600,400);
17    s.setTitle("Test Database");
18    s.setScene(sc);
19    s.show();
20 }
21 public HBox getHBTabel(){
22     HBox hb= new HBox();
23     TableView<MHS> tbMHS = new TableView();
24     hb.getChildren().add(tbMHS);
25     TableColumn<MHS,Integer> colIdmhs= new
26     TableColumn("ID");
27     TableColumn<MHS,String> colNim=new
28     TableColumn("NIM");
29     TableColumn<MHS,String> colNama=new
30     TableColumn("Nama");
31     TableColumn<MHS,String> colKota_asal=new
32     TableColumn("Kota Asal");
33
34     colIdmhs.setCellValueFactory(new
35     PropertyValueFactory<>("idmhs"));
36     colNim.setCellValueFactory(new
37     PropertyValueFactory<>("nim"));
38     colNama.setCellValueFactory(new
39     PropertyValueFactory<>("nama"));
40     colKota_asal.setCellValueFactory(new
41     PropertyValueFactory<>("kota_asal"));
42
43     colIdmhs.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
44
45     colNim.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
46
47     colNama.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));

```

```

39 colKota_asal.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
40 tbMHS.getColumns().addAll(colIdmhs,colNim,colNama,colKota_asal);
41 tbMHS.setItems(getAllMhs());
42 return hb;
43 }
44 public ObservableList<MHS> getAllMhs(){
45 ObservableList<MHS> lmhs= FXCollections.observableArrayList();
46 try {
47     Statement st=MySQLDB.getConnection().createStatement();
48     String sql="select * from tbmhs";
49     ResultSet rs=st.executeQuery(sql);
50     rs.beforeFirst();
51     while(rs.next()){
52         MHS m = new MHS();
53         m.setIdmhs(Integer.parseInt(rs.getString(1)));
54         m.setNim(rs.getString(2));
55         m.setNama(rs.getString(3));
56         m.setKota_asal(rs.getString(4));
57         lmhs.add(m);
58         System.out.println(m);
59     }
60 } catch (SQLException ex) {
61     ex.printStackTrace();
62 return lmhs;
63 }
64 }
```

ID	NIM	Nama	Kota Asal
1	A11.2017.00001	HERU	SEMARANG
2	A11.2017.00002	TARI	SOLO
3	A11.2017.00003	BUDI	SALATIGA
4	A11.2017.00003	AGUS	SALATIGA
5	A11.2017.00003	AGUS	SALATIGA
6	A11.2017.10111	ARI	KUDUS

Output

## Metadata

interface Connection membuat koneksi terhadap database, koneksi yang membuat statement sql dapat dieksekusi dan dapat menghasilkan data *resultset*. Connnection juga menyediakan akses informasi meta data dari sebuah database dimana informasi metadata tersebut menjelaskan kapabilitas dari suatu database diantaranya dukungan terhadap bentuk SQL,store procedure dll.

Untuk membuat obyek dari Metadata sebuah database digunakan method getMetaData pada obyek koneksi.

```
DatabaseMetaData dbMD = conn.getMetaData();
```

Contoh

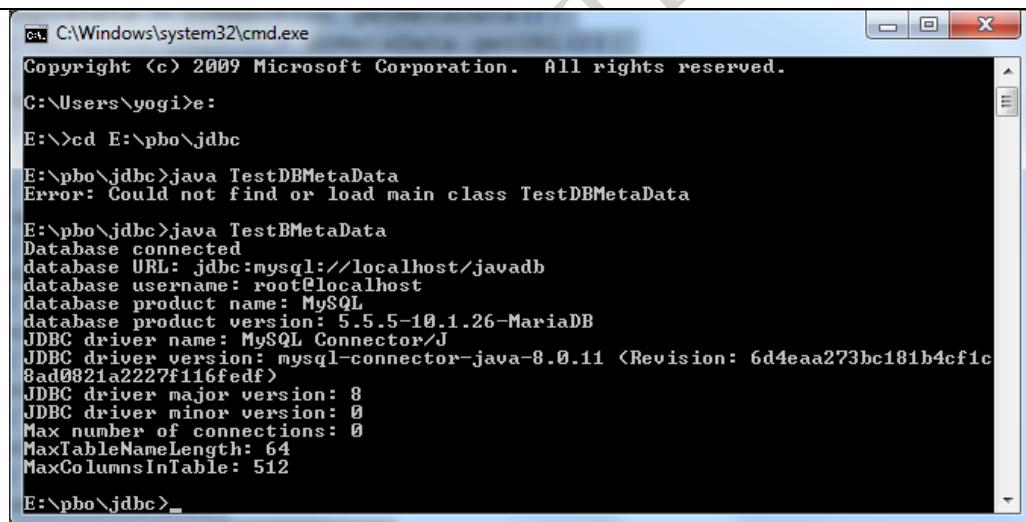
### Kode 13.7 TestBMetaData

```
1 import java.sql.*;  
2 public class TestBMetaData {  
3     public static void main(String[] args) throws SQLException,  
4         ClassNotFoundException {  
5         Connection connection =  
6             DriverManager.getConnection("jdbc:mysql://localhost/javadb", "root",  
7                 "");  
8         System.out.println("Database connected");  
9         DatabaseMetaData dbMetaData = connection.getMetaData();  
10        System.out.println("database URL: " + dbMetaData.getURL());  
11        System.out.println("database username: " +  
12            dbMetaData.getUserName());  
13        System.out.println("database product name: " +  
14            dbMetaData.getDatabaseProductName());  
15        System.out.println("database product version: " +  
16            dbMetaData.getDatabaseProductVersion());  
17        System.out.println("JDBC driver name: " +  
18            dbMetaData.getDriverName());
```

```

12     System.out.println("JDBC driver version: " +
13         dbMetaData.getDriverVersion());
14     System.out.println("JDBC driver major version: " +
15         dbMetaData.getDriverMajorVersion());
16     System.out.println("JDBC driver minor version: " +
17         dbMetaData.getDriverMinorVersion());
18     System.out.println("Max number of connections: " +
19         dbMetaData.getMaxConnections());
20     System.out.println("MaxTableNameLength: " +
21         dbMetaData.getMaxTableNameLength());
22     System.out.println("MaxColumnsInTable: " +
23         dbMetaData.getMaxColumnsInTable());
24     connection.close();
25 }

```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command line shows the user navigating to the directory 'E:\pbo\jdbc' and running the Java command 'java TestDBMetaData'. The output of the program is displayed in the window, providing detailed JDBC metadata information:

```

Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\yogi>e:
E:>>cd E:\pbo\jdbc

E:\pbo\jdbc>java TestDBMetaData
Error: Could not find or load main class TestDBMetaData

E:\pbo\jdbc>java TestBMetaData
Database connected
database URL: jdbc:mysql://localhost/javadb
database username: root@localhost
database product name: MySQL
database product version: 5.5.5-10.1.26-MariaDB
JDBC driver name: MySQL Connector/J
JDBC driver version: mysql-connector-java-8.0.11 <Revision: 6d4eaa273bc181b4cf1c
8ad0821a2227f116fedf>
JDBC driver major version: 8
JDBC driver minor version: 0
Max number of connections: 0
MaxTableNameLength: 64
MaxColumnsInTable: 512

E:\pbo\jdbc>_

```

Kode 13.8 merupakan contoh penggunaan MetaData.

#### Kode 13.8 TestRSMetaData.java

1	import java.sql.*;
2	

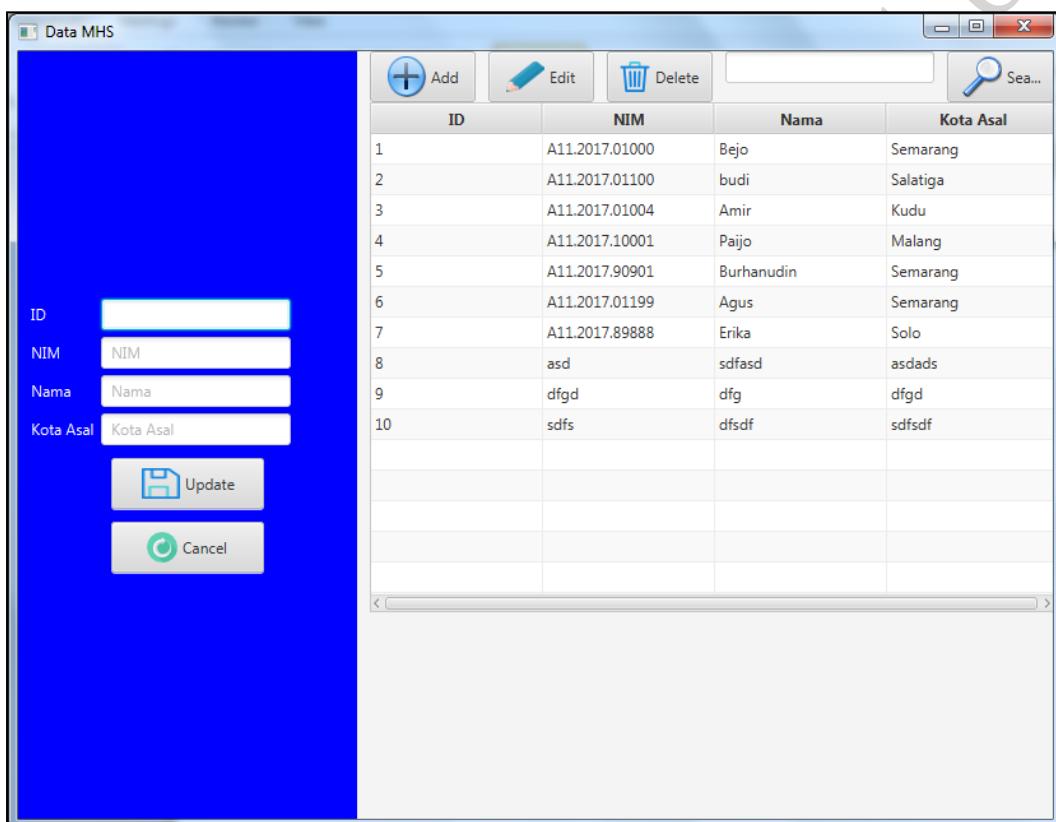
```

3  public class TestRSMetaData {
4      public static void main(String[] args) throws SQLException,
5          ClassNotFoundException {
6          // Connect to a database
7          Connection connection =
8              DriverManager.getConnection("jdbc:mysql://localhost/dbjava",
9                  "root", "");
10         System.out.println("Database connected");
11         // Create a statement
12         Statement statement = connection.createStatement();
13         // Execute a statement
14         ResultSet resultSet = statement.executeQuery("select *
from tbmhs");
15
16         ResultSetMetaData rsMetaData =
17             resultSet.getMetaData();
18         for (int i = 1; i <= rsMetaData.getColumnCount(); i++)
19             System.out.printf("%-12s\t", rsMetaData.getColumnName(i));
20         System.out.println();
21         // Iterate through the result and print the students'
22         // column
23         while (resultSet.next()) {
24             for (int i = 1; i <= rsMetaData.getColumnCount();
25                 i++)
26                 System.out.printf("%-12s\t",
27                     resultSet.getObject(i));
28             System.out.println();
29         }
30         // Close the connection
31         connection.close();
32     }
33 }
```

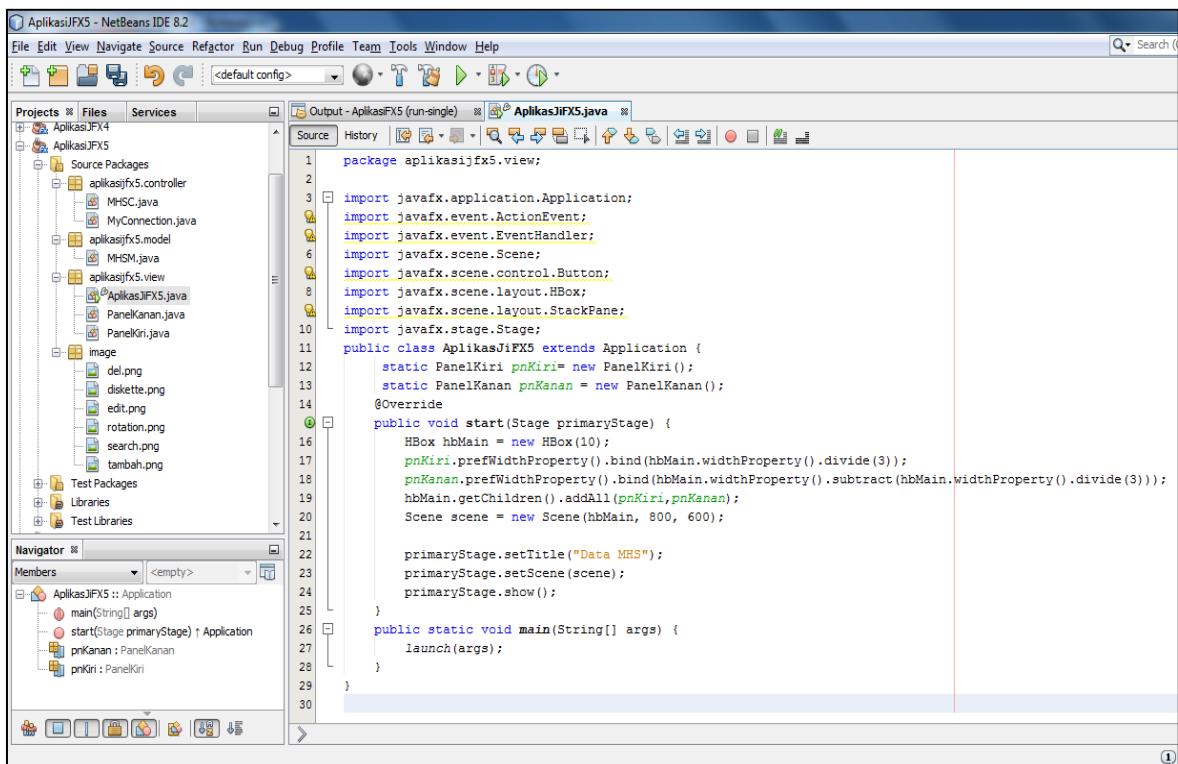
## Output

```
E:\pbo\jdbc\modul>javac TestRSMetaData.java
E:\pbo\jdbc\modul>java TestRSMetaData
Database connected
idmhs      nim          nama        kota_asal
1          A11.2017.00001 HERU        MEDAN
2          A11.2017.00002 TARI        SOLO
3          A11.2017.00003 BUDI        SALATIGA
4          A11.2017.00003 AGUS        SALATIGA
5          A11.2017.00003 AGUS        SALATIGA
6          A11.2016.77771 HERI        MEDAN
7          A11.2017.00007 SLAMET      JEPARA
E:\pbo\jdbc\modul>
```

Contoh Aplikasi menggunakan desain pattern MVC:



Gambar 13.3 Hasil program



Gambar 13.9 Project terdiri dari package controller, model dan view.

Source code lengkap

#### Package view

#### Kode 13.9 AplikasiJiFX5.java

```
1 package aplikasijfx5.view;
2
3 import javafx.application.Application;
4 import javafx.event.ActionEvent;
5 import javafx.event.EventHandler;
6 import javafx.scene.Scene;
7 import javafx.scene.control.Button;
8 import javafx.scene.layout.HBox;
9 import javafx.scene.layout.StackPane;
10 import javafx.stage.Stage;
11
12 public class AplikasiJiFX5 extends Application {
13     static PanelKiri pnKiri= new PanelKiri();
14     static PanelKanan pnKanan = new PanelKanan();
```

```

14     @Override
15     public void start(Stage primaryStage) {
16         HBox hbMain = new HBox(10);
17
18         pnKiri.prefWidthProperty().bind(hbMain.widthProperty().divide(3));
19         pnKanan.prefWidthProperty().bind(hbMain.
20             widthProperty().subtract(hbMain.widthProperty().divide(3)));
21
22         hbMain.getChildren().addAll(pnKiri,pnKanan);
23
24         Scene scene = new Scene(hbMain, 800, 600);
25
26         primaryStage.setTitle("Data MHS");
27         primaryStage.setScene(scene);
28         primaryStage.show();
29     }
30
31     public static void main(String[] args) {
32         launch(args);
33     }
34 }
```

### Kode 13.10 PanelKanan.java

```

1 package aplikasijfx5.view;
2
3 import aplikasijfx5.controller.MHSC;
4 import aplikasijfx5.model.MHSM;
5 import java.util.Optional;
6 import javafx.geometry.Pos;
7 import javafx.scene.control.Alert;
8 import javafx.scene.control.Alert.AlertType;
9 import javafx.scene.control.Button;
10 import javafx.scene.control.ButtonType;
11 import javafx.scene.control.TableColumn;
12 import javafx.scene.control.TableView;
13 import javafx.scene.control.TextField;
```

```
14 import javafx.scene.control.cell.PropertyValueFactory;
15 import javafx.scene.image.Image;
16 import javafx.scene.image.ImageView;
17 import javafx.scene.layout.HBox;
18 import javafx.scene.layout.VBox;
19
20 public class PanelKanan extends VBox{
21     HBox hbSearch = new HBox(10);
22     TextField tfSearch = new TextField();
23     Button btSearch,btAdd,btEdit,btDelete ;
24     TableView<MHSM> tbMHS = new TableView();
25     MHSC mhsc = new MHSC();
26     public PanelKanan(){
27         Image imgAdd = new Image("image/tambah.png");
28         Image imgEdit = new Image("image/edit.png");
29         Image imgDel = new Image("image/del.png");
30         Image imgSearch = new Image("image/search.png");
31
32         ImageView ivAdd = new ImageView(imgAdd);
33         ivAdd.setFitWidth(70);
34         ivAdd.setFitWidth(30);
35         ImageView ivEdit = new ImageView(imgEdit);
36         ivEdit.setFitWidth(70);
37         ivEdit.setFitWidth(30);
38         ImageView ivDel = new ImageView(imgDel);
39         ImageView ivSearch = new ImageView(imgSearch);
40         btSearch = new Button("Search",ivSearch);
41         btAdd = new Button("Add",ivAdd);
42         btEdit = new Button("Edit",ivEdit);
43         btDelete = new Button("Delete",ivDel);
44         btAdd.setPrefWidth(120);
45         btAdd.setPrefHeight(40);
```

```

46     btEdit.setPrefWidth(120);
47     btEdit.setPrefHeight(40);
48     btDelete.setPrefWidth(120);
49     btDelete.setPrefHeight(40);
50     btSearch.setPrefWidth(120);
51     btSearch.setPrefHeight(40);
52     tfSearch.setPrefWidth(200);
53     TableColumn<MHSM, Integer> colIdmhs= new TableColumn("ID");
54     TableColumn<MHSM, String> colNim=new TableColumn("NIM");
55     TableColumn<MHSM, String> colNama=new TableColumn("Nama");
56     TableColumn<MHSM, String> colKota_asal=new TableColumn("Kota
Asal");
57
58         colIdmhs.setCellValueFactory(new
PropertyValueFactory<>("idmhs"));
59         colNim.setCellValueFactory(new
PropertyValueFactory<>("nim"));
60         colNama.setCellValueFactory(new
PropertyValueFactory<>("nama"));
61         colKota_asal.setCellValueFactory(new
PropertyValueFactory<>("kota_asal"));
62     colIdmhs.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
63     colNim.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
64     colNama.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
65     colKota_asal.prefWidthProperty().bind(tbMHS.widthProperty()).
divide(4));
66
67
68     tbMHS.getColumns().addAll(colIdmhs,colNim,colNama,colKota_asal);
69     tbMHS.setItems(mhsc.getAllMhs());

```

```

70         //hbSearch.setAlignment(Pos.CENTER);
71
72     hbSearch.getChildren().addAll(btAdd,btEdit,btDelete,tfSearch,
73     btSearch);
74
75     this.getChildren().addAll(hbSearch,tbMHS);
76
77     btAdd.setOnMouseClicked(e->{
78
79         AplikasJiFX5.pnKiri.aktif(true);
80
81         AplikasJiFX5.pnKiri.blAdd=true;
82
83         AplikasJiFX5.pnKiri.tfID.requestFocus();
84
85     });
86
87     btSearch.setOnMouseClicked(e->{
88
89
90     this.tbMHS.setItems(mhsc.searchMHS(tfSearch.getText()));
91
92     });
93
94
95
96
97
98

```

```

99         AplikasJiFX5.pnKiri.tfID.setEditable(false);
100        AplikasJiFX5.pnKiri.blAdd=false;
101        AplikasJiFX5.pnKiri.tfNama.requestFocus();
102    });
103    btDelete.setOnMouseClicked(e->{
104        MHSM m= tbMHS.getSelectionModel().getSelectedItem();
105        //
106        Alert alert = new Alert(AlertType.CONFIRMATION);
107        alert.setTitle("Hapus Data");
108        alert.setHeaderText("Yakin data dihapus?");
109        alert.setContentText("data "+m.getNama()+" dihapus");
110        Optional<ButtonType> option = alert.showAndWait();
111
112        if (option.get() == ButtonType.OK){
113            mhsc.delete(String.valueOf(m.getIdmhs()));
114            this.tbMHS.setItems(mhsc.getAllMhs());
115        }
116        //
117    });
118 }
119 }
```

### Kode 13.11 PanelKiri.java

```

1 package aplikasijfx5.view;
2
3 import aplikasijfx5.controller.MHSC;
4 import aplikasijfx5.model.MHSM;
5 import javafx.geometry.Insets;
6 import javafx.geometry.Pos;
7 import javafx.scene.control.Button;
8 import javafx.scene.control.Label;
9 import javafx.scene.control.TextField;
10 import javafx.scene.image.Image;
```

```
11 import javafx.scene.image.ImageView;
12 import javafx.scene.layout.GridPane;
13 import javafx.scene.layout.VBox;
14
15 public class PanelKiri extends VBox {
16     Label lbID = new Label("ID");
17     Label lbNIM = new Label("NIM");
18     Label lbNama = new Label("Nama");
19     Label lbKota = new Label("Kota Asal");
20     TextField tfID=new TextField();
21     TextField tfNIM=new TextField();
22     TextField tfNama=new TextField();
23     TextField tfKota=new TextField();
24     Button btUpdate ;
25     Button btCancel;
26     int tempId;
27     boolean blAdd;
28     MHSC mhsc = new MHSC();
29
30     public PanelKiri(){
31         GridPane gp = new GridPane();
32         gp.add(lbID,0,0);
33         gp.add(lbNIM,0,1);
34         gp.add(lbNama,0,2);
35         gp.add(lbKota,0,3);
36         gp.add(tfID,1,0);
37         gp.add(tfNIM,1,1);
38         gp.add(tfNama,1,2);
39         gp.add(tfKota,1,3);
40         gp.setVgap(5);gp.setHgap(5);
41         tfID.setPromptText("ID MHS");
42         tfNIM.setPromptText("NIM");
```

```

43     tfNama.setPromptText("Nama");
44     tfKota.setPromptText("Kota Asal");
45     lbID.setStyle("-fx-text-fill:white;");
46     lbNIM.setStyle("-fx-text-fill:white;");
47     lbNama.setStyle("-fx-text-fill:white;");
48     lbKota.setStyle("-fx-text-fill:white;");
49     Image imgUpd = new Image("image/diskette.png");
50     Image imgCancel = new Image("image/rotation.png");
51     ImageView ivUpd = new ImageView(imgUpd);
52     ImageView ivCancel = new ImageView(imgCancel);
53     ivUpd.setFitWidth(70);
54     ivUpd.setFitWidth(30);

55
56     btUpdate = new Button("Update",ivUpd);
57     btUpdate.setPrefWidth(120);
58     btUpdate.setPrefHeight(40);
59     btCancel = new Button("Cancel",ivCancel);
60     btCancel.setPrefWidth(120);
61     btCancel.setPrefHeight(40);
62     //btUpdate.setStyle("-fx-background-color:green;-fx-text-
fill:white;-fx-font-size:14;");
63     //btCancel.setPrefWidth(100);
64     //btCancel.setStyle("-fx-background-color:green;-fx-text-
fill:white;-fx-font-size:14;");

65
66     tfID.setEditable(false);
67     this.getChildren().addAll(gp,btUpdate,btCancel);
68     this.setSpacing(10);
69     this.setPadding(new Insets(10));
70     this.setStyle("-fx-background-color:blue;");
71     this.setAlignment(Pos.CENTER);
72     aktif(false);

```

```

73         btCancel.setOnMouseClicked(e -> {
74             kosong();
75             aktif(false);
76         });
77         btUpdate.setOnMouseClicked(e -> {
78             if (!tfID.getText().equals("")){
79                 MHSM m = new MHSM();
80                 m.setIdmhs(Integer.parseInt(tfID.getText()) );
81                 m.setNim(tfNIM.getText());
82                 m.setNama(tfNama.getText());
83                 m.setKota_asal(tfKota.getText());
84                 if (blAdd==true)
85                     mhsc.insert(m);
86                 else{
87                     m.setIdmhs(tempId);
88                     mhsc.update(m);
89                 }
90             AplikasiJiFX5.pnKanan.tbMHS.setItems(mhsc.getAllMhs());
91             kosong();
92             aktif(false);
93         });
94     });
95     });
96 }
97     public void kosong(){
98         tfID.setText("");
99         tfNIM.setText("");
100        tfNama.setText("");
101        tfKota.setText("");
102    }
103    public void aktif(boolean bl){

```

```

104         tfID.setEditable(bl);
105         tfNIM.setEditable(bl);
106         tfNama.setEditable(bl);
107         tfKota.setEditable(bl);
108     }
109 }

```

#### Package model

#### **Kode 13.12 MHSM.java**

```

1 package aplikasijfx5.model;
2
3 public class MHSM {
4     private int idmhs;
5     private String nim;
6     private String nama;
7     private String kota_asal;
8
9     @Override
10    public String toString() {
11        return "MHSM{" + "idmhs=" + idmhs + ", nim=" + nim + ",
12        nama=" + nama + ", kota_asal=" + kota_asal + '}';
13    }
14    public MHSM(){
15    }
16    public int getIdmhs() {
17        return idmhs;
18    }
19    public void setIdmhs(int idmhs) {
20        this.idmhs = idmhs;
21    }
22    public String getNim() {
23        return nim;
24    }

```

```

24     public void setNim(String nim) {
25         this.nim = nim;
26     }
27     public String getNama() {
28         return nama;
29     }
30     public void setNama(String nama) {
31         this.nama = nama;
32     }
33     public String getKota_asal() {
34         return kota_asal;
35     }
36     public void setKota_asal(String kota_asal) {
37         this.kota_asal = kota_asal;
38     }
39 }
```

#### Package controller

#### Kode 13.13 MHSC.java

```

1 package aplikasijfx5.controller;
2
3 import aplikasijfx5.model.MHSM;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6 import java.sql.Statement;
7 import java.util.ArrayList;
8 import java.util.logging.Level;
9 import java.util.logging.Logger;
10 import javafx.collections.FXCollections;
11 import javafx.collections.ObservableList;
12
13 public class MHSC {
14     Statement st;
```

```

15     public void insert(MHSM m){
16         try {
17             st=MyConnection.getConnection().createStatement();
18             String sql="insert into tbmhs
19             (idmhs,nim,nama,kota_asal) values('"+m.getIdmhs()+"','"+m.
20             getNim()+"','"+m.getNama()+"','"+m.getKota_asal()+"') ";
21             st.executeUpdate(sql);
22         } catch (SQLException ex) {
23             Logger.getLogger(MHSC.class.getName()).log(Level.SEVERE, null, ex);
24         }
25     }
26     public void update(MHSM m){
27         try {
28             st=MyConnection.getConnection().createStatement();
29             String sql="update tbmhs set Nim =
30             '"+m.getNim()+"',nama='"+m.getNama()+"',kota_asal='"+
31             m.getKota_asal()+"' where idmhs='"+m.getIdmhs()+"'";
32             st.executeUpdate(sql);
33         } catch (SQLException ex) {
34             Logger.getLogger(MHSC.class.getName()).log(Level.SEVERE,
35             null, ex);
36         }
37     }
38     public void delete(String idMhs){
39         try {
40             st=MyConnection.getConnection().createStatement();
41             String sql = "delete from tbmhs where idMhs='"+
42             idMhs+"'";
43             st.executeUpdate(sql);
44         } catch (SQLException ex) {
45

```

```

40    Logger.getLogger(MHSC.class.getName()).log(Level.SEVERE,
41        null, ex);
42    }
43    }
44    public ObservableList<MHSM> getAllMhs(){
45        ObservableList<MHSM> lmhs=
46            FXCollections.observableArrayList();
47        try {
48            st=MyConnection.getConnection().createStatement();
49            String sql="select * from tbmhs";
50            ResultSet rs=st.executeQuery(sql);
51            rs.beforeFirst();
52            while(rs.next()){
53                MHSM m = new MHSM();
54                m.setIdmhs(Integer.parseInt(rs.getString(1)));
55                m.setNim(rs.getString(2));
56                m.setNama(rs.getString(3));
57                m.setKota_asal(rs.getString(4));
58                lmhs.add(m);
59                System.out.println(m);
60            }
61        } catch (SQLException ex) {
62            Logger.getLogger(MHSC.class.getName()).log(Level.SEVERE,
63                null, ex);
64        }
65        return lmhs;
66    }
67    public ObservableList<MHSM> searchMHS(String nama){
68        ObservableList<MHSM> lmhs=
69            FXCollections.observableArrayList();
70        try {

```

```

67         st=MyConnection.getConnection().createStatement();
68
69         String sql="select * from tbmhs where nama
70         like '%"+nama+"%'";
71
72         ResultSet rs=st.executeQuery(sql);
73
74         rs.beforeFirst();
75
76         while(rs.next()){
77
78             MHSM m = new MHSM();
79
80             m.setIdmhs(Integer.parseInt(rs.getString(1)));
81
82             m.setNim(rs.getString(2));
83
84             m.setNama(rs.getString(3));
85
86             m.setKota_asal(rs.getString(4));
87
88             lmhs.add(m);
89
90             System.out.println(m);
91
92         }
93
94     } catch (SQLException ex) {
95
96         Logger.getLogger(MHSC.class.getName()).log(Level.SEVERE,
97         null, ex);
98
99     }
100
101     return lmhs;
102 }
103 }
```

#### Kode 13.14 MyConnection.java

```

1 package aplikasijfx5.controller;
2
3 import java.sql.*;
4 import java.util.logging.Level;
5 import java.util.logging.Logger;
6
7
8 public class MyConnection {
9     private static Connection con;
10    private MyConnection(){
11    }
12 }
```

```
10     public static Connection getConnection(){
11         if (con==null){
12             try{
13                 con =
14                     DriverManager.getConnection("jdbc:mysql://localhost/javadb","root",
15                     "");  
16                     //System.out.println("Koneksi berhasil...");  
17             }catch(SQLException se){  
18                 se.printStackTrace();  
19             }  
20             return con;  
21         }  
22         public static void disconnect(){  
23             if (con!=null)  
24                 con=null;  
25         }
```

## 13.6 SQLLite

SQLite adalah database-engine SQL transaksional yang berdiri sendiri. dan merupakan database yang dalam penggunaannya tidak memerlukan konfigurasi sehingga berbeda dengan database lain .

SQLite tidak memerlukan proses server atau sistem yang terpisah untuk beroperasi (serverless).

SQLite tidak memerlukan konfigurasi dalam penggunaannya, yang artinya tidak diperlukan pengaturan atau administrasi terhadap database. Database SQLite lengkap disimpan dalam sebuah file disk yang memiliki format lintas platform. SQLite sangat kecil dan ringan, dengan ukuran kurang dari 400KB atau kurang dari 250KB dengan fitur opsional dihilangkan.

SQLite mandiri, yang berarti tidak ada ketergantungan eksternal dengan aplikasi dan sistem lainnya.

Transaksi SQLite sepenuhnya sesuai standar ACID(Atomicity, Consistency, Isolation, Durability), yang memungkinkan akses yang aman dari berbagai proses atau threads. SQLite mendukung sebagian besar fitur bahasa query dalam standar SQL92 (SQL2).SQLite ditulis dalam ANSI-C dan menyediakan API yang sederhana dan mudah digunakan.SQLite tersedia di UNIX (Linux, Mac OS-X, Android, iOS) dan Windows (Win32, WinCE, WinRT).

Tabel 13.1 Beberapa fitur yang tidak didukung dari SQL92 di SQLite

No	Fitur yg tidak didukung	Keterangan
1	RIGHT OUTER JOIN	Hanya mendukung LEFT OUTER JOIN
2	FULL OUTER JOIN	Hanya LEFT OUTER JOIN yg didukung
3	ALTER TABLE	RENAME TABLE dan ADD COLUMN pada the ALTER TABLE didukung  DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT tidak didukung.
4	Trigger	FOR EACH ROW triggers didukung  Tapi FOR EACH STATEMENT triggers tidak didukung

5	View	VIEW dalam SQLite bersifat read-only  Tidak bisa menggunakan statemntn DELETE, INSERT, or UPDATE pada view.
6	GRANT and REVOKE	Satu-satunya izin akses yang dapat diterapkan adalah izin akses file normal dari sistem operasi yang mendasarinya.

#### Kode 13.15 Contoh koneksi sekaligus membuat database baru “testdb”

```

1 Connection c ;
2 Statement stmt;
3 try {
4     Class.forName("org.sqlite.JDBC");
5     c = DriverManager.getConnection("jdbc:sqlite:test.db");
6     System.out.println("Opened database successfully");
7 } catch ( ClassNotFoundException | SQLException e ) {
8     System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
9 }
```

#### Kode 13.16 Contoh membuat tabel baru pada SQLLite

```

1 Connection c ;
2 Statement stmt;
3 try {
4     Class.forName("org.sqlite.JDBC");
5     c = DriverManager.getConnection("jdbc:sqlite:test.db");
6     System.out.println("Opened database successfully");
7     stmt = c.createStatement();
8     String sql = "CREATE TABLE tbmhs " +
                  "(idmhs INT PRIMARY KEY NOT NULL," +
                  " nim TEXT NOT NULL, " +

```

```

    " nama TEXT NOT NULL, " +
    " kota_asalCHAR(50))";

9     stmt.executeUpdate(sql);
10    stmt.close();
11    c.close();
12
13 } catch ( ClassNotFoundException | SQLException e ) {
14     System.err.println( e.getClass().getName() + ": " +
15     e.getMessage() );
}

```

### Kode 13.17 Contoh menambahkan data pada tabel

```

1 Connection c = null;
2
3 Statement stmt = null;
4
5 try {
6
6     Class.forName("org.sqlite.JDBC");
7
8     c = DriverManager.getConnection("jdbc:sqlite:test.db");
9
10    System.out.println("Opened database successfully");
11
12    stmt = c.createStatement();
13
14    String sql = "insert into tbmhs(idmhs,nim,nama,kota_asal)
values (1,'A11.2017.00001','AGUS','SMG') ";
15
16    stmt.executeUpdate(sql);
17
18    sql = "insert into tbmhs(idmhs,nim,nama,kota_asal) values
(2,'A11.2017.00002','BUDI','SOLO') ";
19
20    stmt.executeUpdate(sql);
21
22    sql = "insert into tbmhs(idmhs,nim,nama,kota_asal) values
(3,'A11.2017.00003','WATI','KUDUS') ";
23
24    stmt.executeUpdate(sql);
25
26
27    stmt.close();
28
29    c.commit();

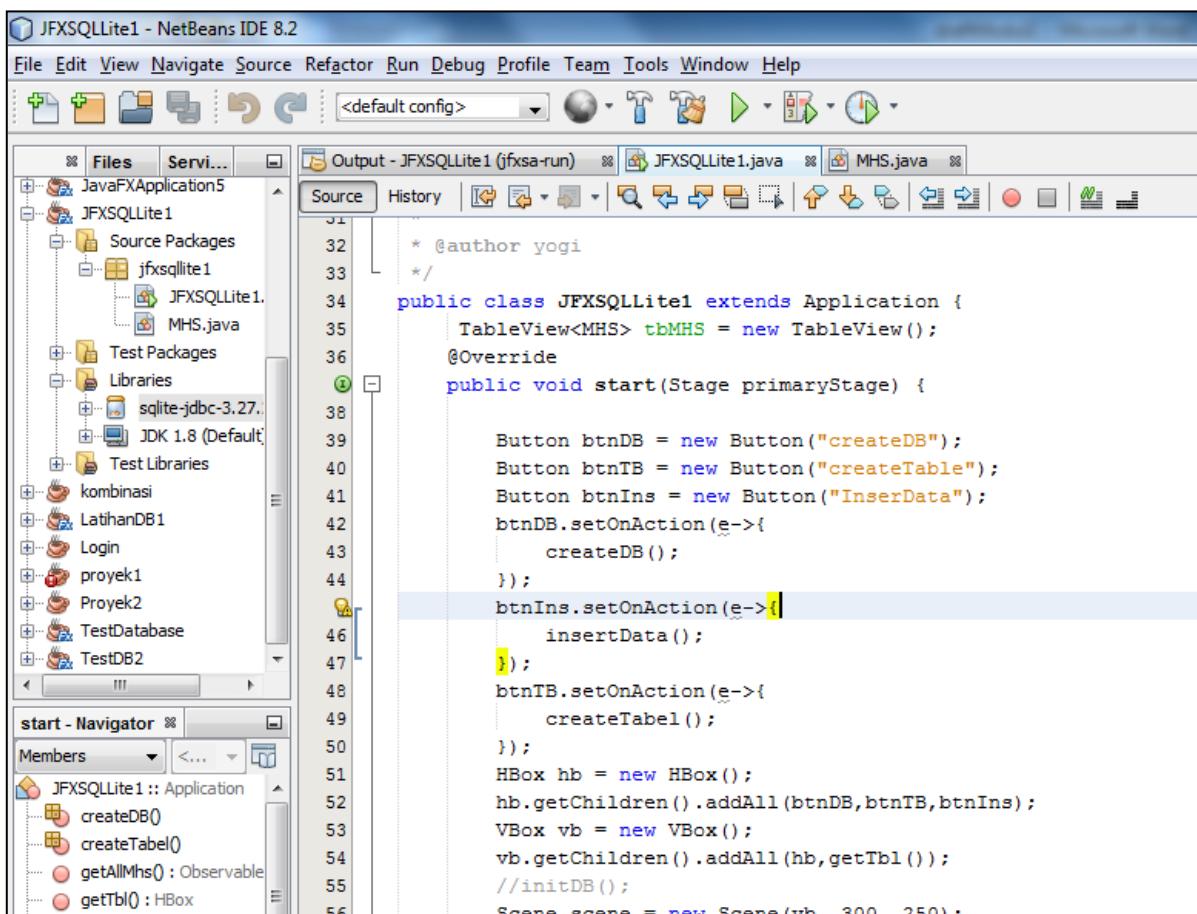
```

```
18     c.close();
19 } catch ( Exception e ) {
20     System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
}
```

#### Kode 13.18 Contoh memproses resultSet

```
1 Connection c ;
2 Statement st;
3 ResultSet rs;
4 try {
5     c = DriverManager.getConnection("jdbc:sqlite:test.db");
6     String sql="select * from tbmhs";
7     st = c.createStatement();
8     rs=st.executeQuery(sql);
9     String idmhs,nim,nama,kota_asal;
10    while(rs.next()){
11        idmhs= rs.getString(1);
12        nim= rs.getString(2);
13        nama= rs.getString(3);
14        kota-asal= rs.getString(4);
15        System.out.println(idmhs+" "+nim+" "+nama);
16    }
17 } catch (Exception ex) {
18     System.err.println(ex);
19 }
```

Contoh Aplikasi Lengkap dengan database SQLLite:



Gambar 13.4 Menambahkan driver sqlite-jdbc-3.27.2.1.jar pada library NetBeanIDE

#### Kode 13.19 JFXSQLLite1.java

```
1 package jfxsqlite1;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.sql.Statement;
8 import java.util.logging.Level;
9 import java.util.logging.Logger;
10 import javafx.application.Application;
11 import javafx.collections.FXCollections;
12 import javafx.collections.ObservableList;
```

```
13 import javafx.event.ActionEvent;
14 import javafx.event.EventHandler;
15 import javafx.scene.Scene;
16 import javafx.scene.control.Button;
17 import javafx.scene.control.TableColumn;
18 import javafx.scene.control.TableView;
19 import javafx.scene.control.cell.PropertyValueFactory;
20 import javafx.scene.layout.HBox;
21 import javafx.scene.layout.StackPane;
22 import javafx.scene.layout.VBox;
23 import javafx.stage.Stage;
24
25 public class JFXSQLLite1 extends Application {
26     TableView<MHS> tbMHS = new TableView();
27     @Override
28     public void start(Stage primaryStage) {
29
30         Button btnDB = new Button("createDB");
31         Button btnTB = new Button("createTable");
32         Button btnIns = new Button("InserData");
33         btnDB.setOnAction(e->{
34             createDB();
35         });
36         btnIns.setOnAction(e->{
37             insertData();
38         });
39         btnTB.setOnAction(e->{
40             createTabel();
41         });
42         HBox hb = new HBox();
43         hb.getChildren().addAll(btnDB,btnTB,btnIns);
44         VBox vb = new VBox();
```

```

45         vb.getChildren().addAll(hb,getTbl());
46
47         Scene scene = new Scene(vb, 300, 250);
48
49         primaryStage.setTitle("Hello World!");
50         primaryStage.setScene(scene);
51         primaryStage.show();
52     }
53
54     public static void main(String[] args) {
55         launch(args);
56     }
57
58     void createDB(){
59         Connection c ;
60         Statement stmt;
61         try {
62             Class.forName("org.sqlite.JDBC");
63             c = DriverManager.getConnection("jdbc:sqlite:test.db");
64             System.out.println("Opened database successfully");
65             c.close();
66
67         } catch ( ClassNotFoundException | SQLException e ) {
68             System.err.println( e.getClass().getName() + ": " +
69             e.getMessage() );
70             //System.exit(0);
71         }
72         System.out.println("Opened database successfully");
73     }
74
75     void createTabel(){
76         Connection c = null;
77         Statement stmt = null;
78         try {

```

```

76     Class.forName("org.sqlite.JDBC");
77
78     c = DriverManager.getConnection("jdbc:sqlite:test.db");
79
80     System.out.println("Opened database successfully");
81
82     stmt = c.createStatement();
83
84     String sql = "CREATE TABLE tbmhs " +
85                 "(idmhs INT PRIMARY KEY NOT NULL," +
86                  " nim TEXT, " +
87                  " nama TEXT, " +
88                  " kota_asal TEXT )";
89
90     stmt.executeUpdate(sql);
91
92     stmt.close();
93
94     c.close();
95
96
97 } catch ( Exception e ) {
98
99     System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
100
101    //System.exit(0);
102
103 }
104
105 System.out.println("Create table successfully");
106
107 }
108
109
110 void insertData(){
111
112     Connection c = null;
113
114     Statement stmt = null;
115
116     try {
117
118         Class.forName("org.sqlite.JDBC");
119
120         c = DriverManager.getConnection("jdbc:sqlite:test.db");
121
122         System.out.println("Opened database successfully");
123
124         stmt = c.createStatement();
125
126         String sql = "insert into tbmhs(idmhs,nim,nama,kota_asal)
127 values (1,'A11.2017.00001','AGUS','SEMARANG') ";
128
129         stmt.executeUpdate(sql);

```

```

102         sql = "insert into tbmhs(idmhs,nim,nama,kota_asal) values
103             (2,'A11.2017.00002','BUDI','SEMARANG') ";
104             stmt.executeUpdate(sql);
105             sql = "insert into tbmhs(idmhs,nim,nama,kota_asal) values
106             (3,'A11.2017.00003','WATI','SEMARANG') ";
107             stmt.executeUpdate(sql);
108             stmt.close();
109             c.commit();
110             c.close();
111
112     } catch ( Exception e ) {
113         System.err.println( e.getClass().getName() + ": " +
114 e.getMessage() );
115     }
116
117     System.out.println("insert data successfully");
118 }
119
120 @SuppressWarnings("CallToPrintStackTrace")
121 public ObservableList<MHS> getAllMhs(){
122     Connection c ;
123     Statement st;
124     ResultSet rs;
125     ObservableList<MHS> lmhs=
126 FXCollections.observableArrayList();
127     try {
128         c = DriverManager.getConnection("jdbc:sqlite:test.db");
129         String sql="select * from tbmhs";
130         st = c.createStatement();
131         rs=st.executeQuery(sql);
132         while(rs.next()){
133             MHS m = new MHS();
134             m.setIdmhs(rs.getInt("idmhs"));
135             m.setNim(rs.getString("nim"));
136             m.setNama(rs.getString("nama"));
137             m.setKotaAsal(rs.getString("kota_asal"));
138             lmhs.add(m);
139         }
140     }
141     catch (Exception e) {
142         e.printStackTrace();
143     }
144     return lmhs;
145 }
```

```

130         m.setIdmhs(Integer.parseInt(rs.getString(1)));
131         m.setNim(rs.getString(2));
132         m.setNama(rs.getString(3));
133         m.setKota_asal(rs.getString(4));
134         lmhs.add(m);
135         System.out.println(m);
136     }
137     } catch (Exception ex) {
138         System.err.println(ex);
139     }
140     return lmhs;
141 }
142 public HBox getTbl(){
143     HBox hb = new HBox();
144
145     TableColumn<MHS, Integer> colIdmhs= new TableColumn("ID");
146     TableColumn<MHS, String> colNim=new TableColumn("NIM");
147     TableColumn<MHS, String> colNama=new TableColumn("Nama");
148     TableColumn<MHS, String> colKota_asal=new TableColumn("Kota
Asal");
149
150     colIdmhs.setCellValueFactory(new
PropertyValueFactory<>("idmhs"));
151     colNim.setCellValueFactory(new
PropertyValueFactory<>("nim"));
152     colNama.setCellValueFactory(new
PropertyValueFactory<>("nama"));
153     colKota_asal.setCellValueFactory(new
PropertyValueFactory<>("kota_asal"));
154     colIdmhs.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
155     colNim.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));

```

```

157 colNama.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
158
159 colKota_asal.prefWidthProperty().bind(tbMHS.widthProperty().divide(4));
160
161 tbMHS.getColumns().addAll(colIdmhs,colNim,colNama,colKota_asal);
162 tbMHS.setItems(getAllMhs());
163 hb.getChildren().add(tbMHS);
164 return hb;
165 }
166 }

```

### Kode 13.20 MHS.java

```

1 package jfxsqlite1;
2 public class MHS {
3     private int idmhs;
4     private String nim;
5     private String nama;
6     private String kota_asal;
7
8     @Override
9     public String toString() {
10         return "MHSM{" + "idmhs=" + idmhs + ", nim=" + nim + ",
11             nama=" + nama + ", kota_asal=" + kota_asal + '}';
12     }
13     public MHS(){
14     }
15     public int getIdmhs() {
16         return idmhs;
17     }
18     public void setIdmhs(int idmhs) {
19         this.idmhs = idmhs;
20     }
21 }

```

```

19 }
20     public String getNim() {
21         return nim;
22     }
23     public void setNim(String nim) {
24         this.nim = nim;
25     }
26     public String getNama() {
27         return nama;
28     }
29     public void setNama(String nama) {
30         this.nama = nama;
31     }
32     public String getKota_asal() {
33         return kota_asal;
34     }
35     public void setKota_asal(String kota_asal) {
36         this.kota_asal = kota_asal;
37     }
38 }
```

### Output

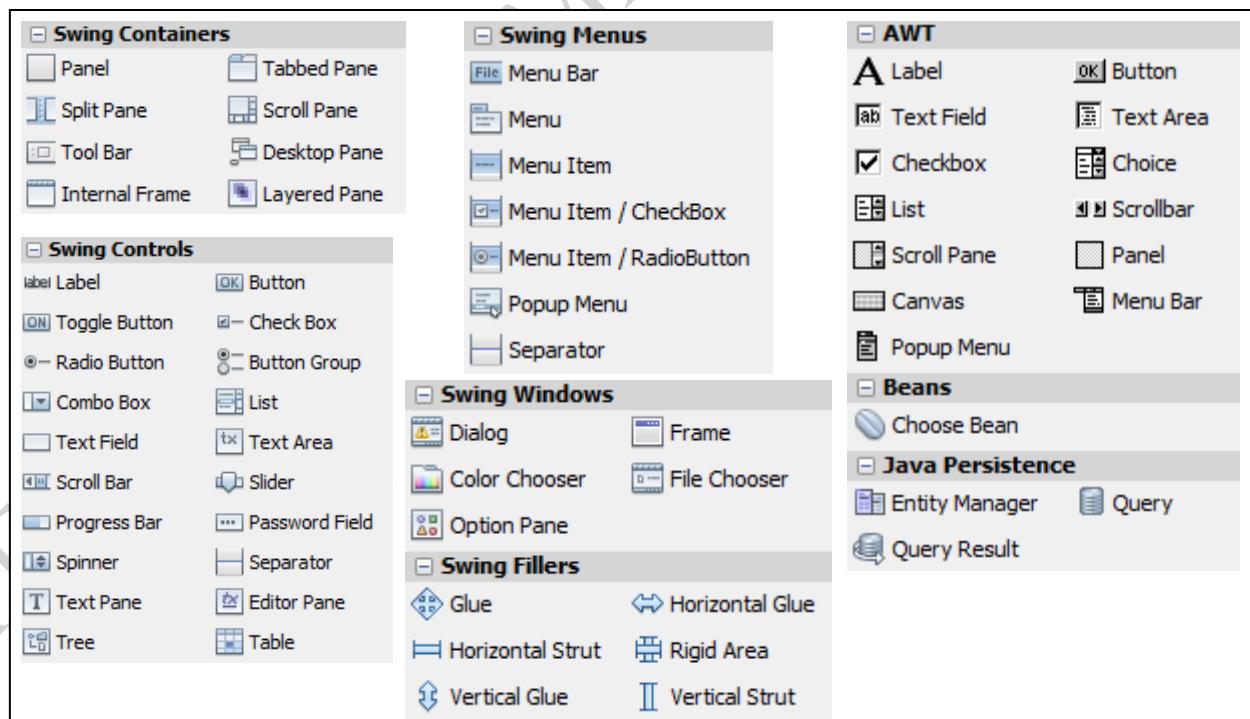
ID	NIM	Nama	Kota Asal
1	A11.2017.00001	AGUS	SEMARANG
2	A11.2017.00002	BUDI	SEMARANG
3	A11.2017.00003	WATI	SEMARANG

## BAB 14 GUI DAN MVC CRUD MENGGUNAKAN NETBEANS

Pada bab-bab sebelumnya, untuk membuat program Java kita menggunakan Notepad. Selain itu kita juga bisa menggunakan text editor yang lain seperti Wordpad, Sublime, Visual Studio Code, atau yang lain. Untuk compile dan run program menggunakan command window. Namun, pada bab ini kita bisa menggunakan Java development tools berupa Integrated Development Environment (IDE) seperti Netbeans atau Eclipse. IDE tersebut dapat digunakan untuk membuat, mengedit, compile, run, dan debugging pemrograman Java dengan cepat dengan satu tampilan. Sehingga kita dapat meningkatkan produktivitas kita dalam pemrograman. Berikut cara membuat pemrograman Graphical User Interface (GUI) Java menggunakan IDE Netbeans.

### 14.1 Persiapan menggunakan Netbeans

Hal yang pertama diperlukan adalah instalasi, kunjungi website ini Download Netbeans <https://netbeans.org/downloads/8.0.2/> pilih untuk Desktop dan Install. Komponen-komponen GUI menggunakan Java Swing yang terdiri dari Swing Containers, Swing Controls, Swing Menus,dan Swing Windows. Di dalam setiap komponen swing terdapat berbagai macam bentuk seperti pada gambar 14.1.

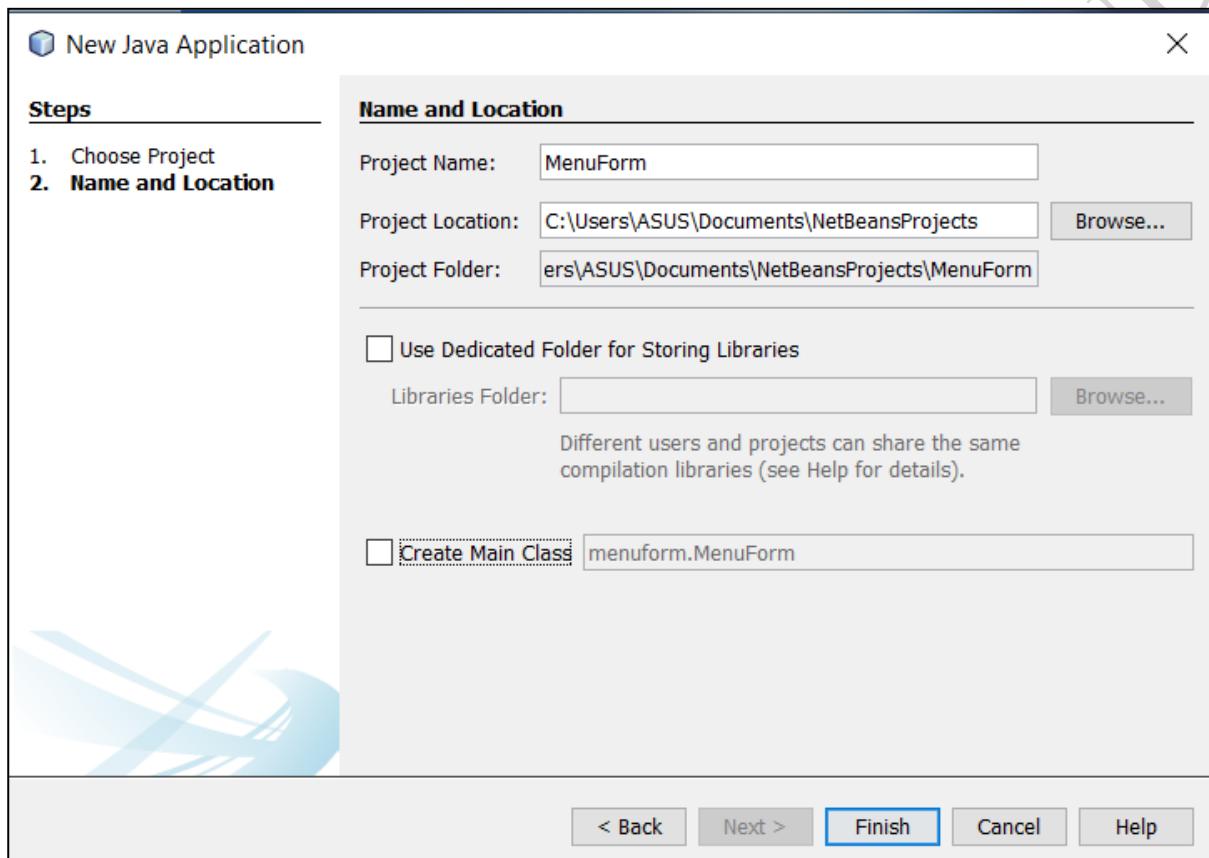


Gambar 14.1 Komponen dari Java Swing

## 14.2 Java Form

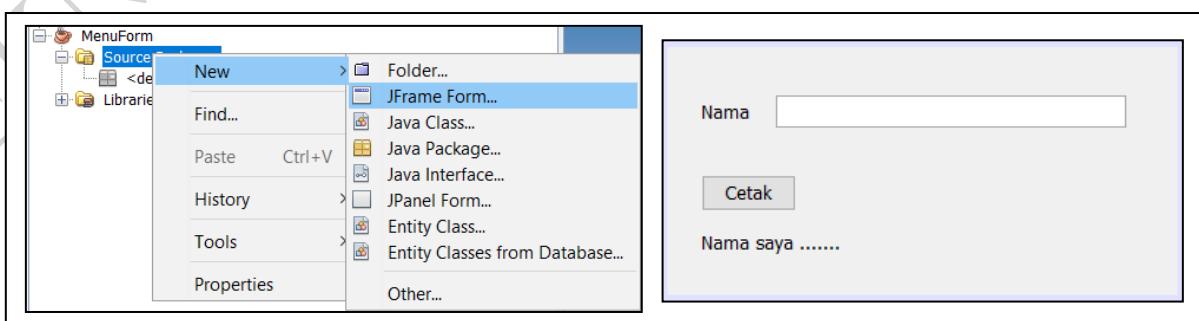
Setelah membuka Netbeans, mari kita membuat Project untuk program kita. Ikuti langkah-langkah berikut dengan seksama:

1. Klik File – New Project – pilih Java – Java Application – Next – ketik judul project dengan “**MenuForm**” – terakhir klik Finish.



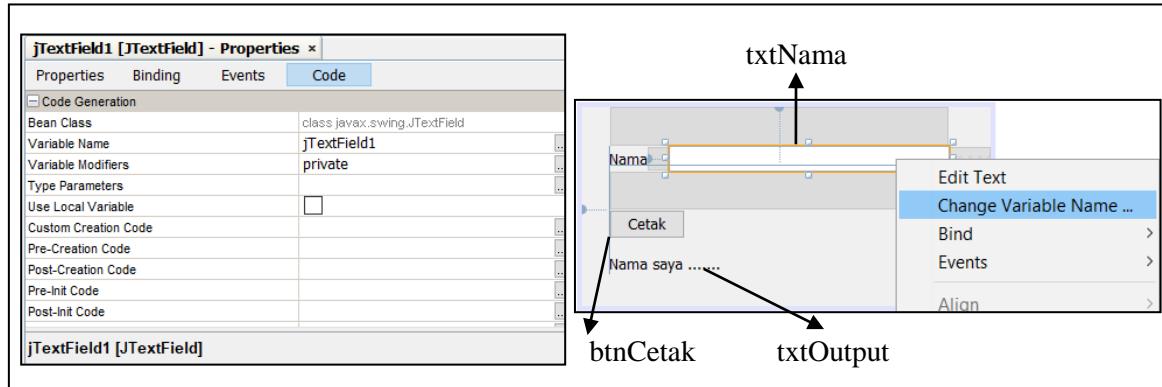
Gambar 14.2 Membuat project **MenuForm**

2. Buat JFrame Form dengan nama “**MainMenuItem**”. Tambahkan komponen Button, Label, dan textfield. F2 untuk mengganti keterangan text pada Button/Label/Textfield.



Gambar 14.3 Membuat JFrame Form **MainMenuItem**

3. Rubah nama variabel pada textfield. Nama variabel digunakan sebagai perwakilan dari komponen UI tersebut untuk dapat di program. Cara 1: klik pada textfield → lihat property. Cara 2: klik kanan pada textfield → Change Variable Name. Beri nama variabel textfield dengan nama “**txtNama**”. Selain itu untuk label output “Nama saya ...” ganti nama variabel menjadi “**txtOutput**”. Button cetak ganti variabelnya menjadi “**btnCetak**”.



Gambar 14.4 Rubah nama variabel

4. Netbeans menyediakan Event-Listener Otomatis dengan cara double klik pada btnCetak. Tambahkan kode seperti pada gambar berikut:

```
private void btnCetakActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    txtOutput.setText(txtNama.getText());
}
```

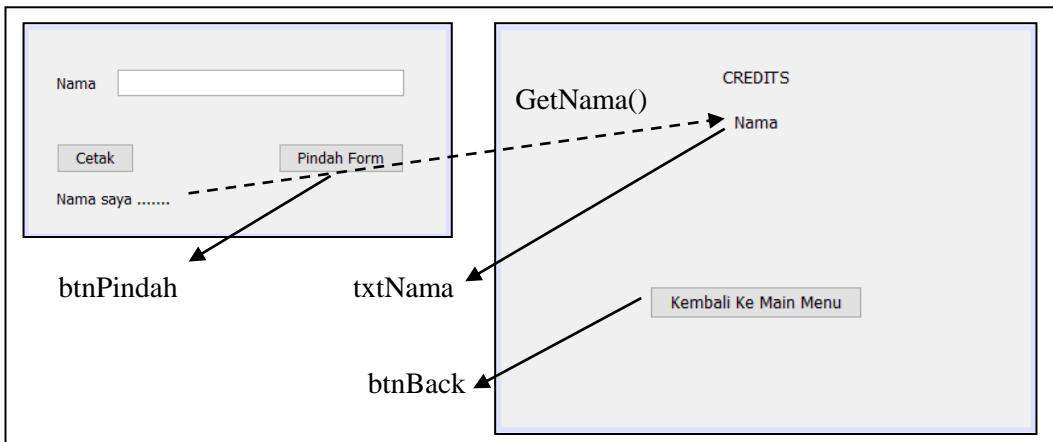
Gambar 14.5 Event-Listener Otomatis

5. Run, jadikan MainMenuForm sebagai Main Form.



Gambar 14.6 Hasil output

6. Tambahkan 1 button dengan nama variabel **btnPindah** dimana 1 button tersebut digunakan sebagai navigasi untuk berpindah form. Setelah itu buat Form baru Jframe Form dengan nama “**CreditsForm**”. Di dalamnya terdapat **txtNama** dan **btnBack**.



Gambar 14.7 Tambah button **txtPindah** dan JFrame **CreditsForm**

7. Mari kita kirimkan teks antar form. Buat Method di dalam form **MainMenuForm** dengan nama **GetNama()**.

```

public String GetNama()
{
    return txtOutput.getText();
}

private void btnPindahActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    CreditsForm formCred = new CreditsForm(txtOutput.getText());
    formCred.setVisible(true);
    this.dispose();
}

```

Gambar 14.8 Buat method **GetNama()** da isi **btnPindah** di form **MainMenuForm**

8. Buka **CreditsForm**, tambahkan konstruktor baru dengan parameter **txtNama**. Buat variabel property untuk menampung **txtOutput** dari Form **MainMenuForm**.

```

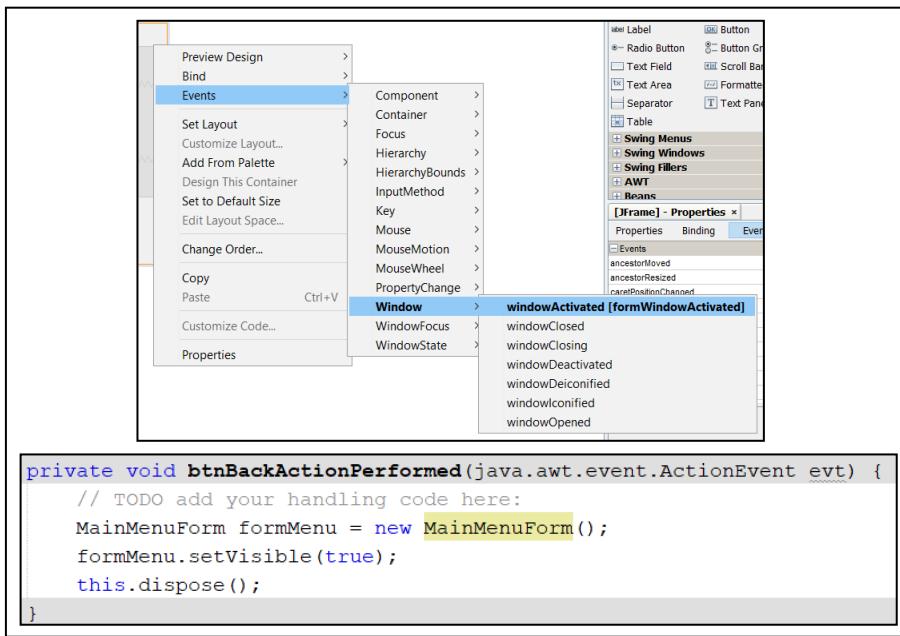
String txtNamaDariFormLain;
public CreditsForm() {
    initComponents();
}

public CreditsForm(String txtNama) {
    initComponents();
    txtNamaDariFormLain = txtNama;
}

```

Gambar 14.9 Buat konstruktor baru di Form **CreditsForm**

9. Masih di dalam Form **CreditsForm**, klik kanan pada Form dan tambahkan event seperti pada gambar 14.10. Tambahkan didalam event listener windowActivated kode: **txtNama.setText(txtNamaDariFormLain)**. Tambahkan event pada **btnBack** (double klik pada **btnBack**) seperti pada gambar 14.10. Run dan lihat apa yang terjadi!

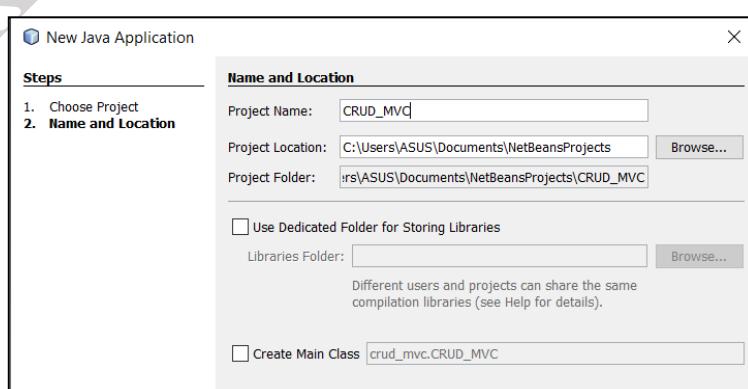


Gambar 14.10 Edit beberapa tempat di Form CreditsForm

### 14.3 MVC CRUD menggunakan Netbeans

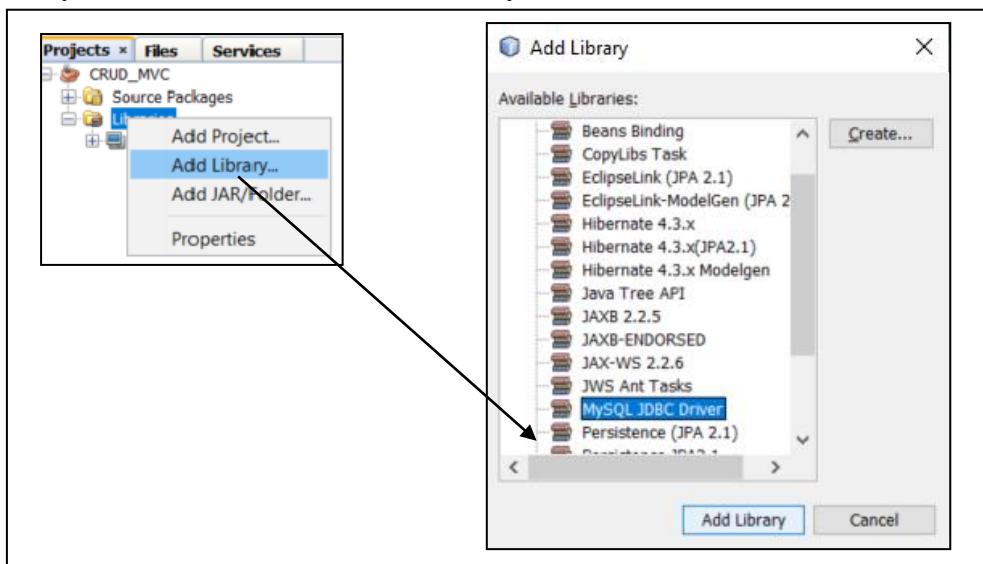
Pada bab 10.6 terdapat konsep tentang Model-View-Controller (MVC). Dimana MVC adalah sebuah metode untuk membuat sebuah aplikasi dengan memisahkan data (Model) dari tampilan (View) dan cara bagaimana memprosesnya (Controller). Mari kita implementasikan bersama.

1. Persiapan database dengan mendownload dan install xampp <https://www.apachefriends.org/index.html>. Aktifkan Apache local server dan mysql. Buat database dengan nama **db\_crud**. Buat tabel database dengan nama **tblmahasiswa** yang berisi **id int(11)**, **nim varchar(20)**, **nama varchar(30)**, **jk varchar(15)**, dan **alamat text**.
2. Buat Project dengan nama “**CRUD\_MVC**”. Uncheck create Main Class.



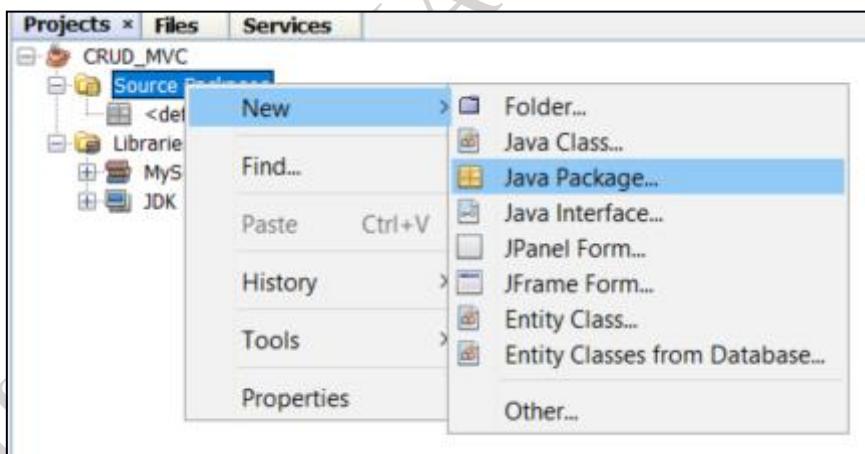
Gambar 14.11 Buat Project dengan nama **CRUD\_MVC**

3. Tambahkan **Library Mysql** dengan cara klik kanan pada Libraries → Pilih Add Library  
 → Cari MySQL JDBC Driver → Add Library



Gambar 14.12 Tambahkan **Library Mysql**

4. Siapkan Package/ Folder untuk MVC. Klik kanan pada Source Packages → New → Java Packages. Buat package dengan nama “mvc.Controller”, “mvc.DAO”, “mvc.DAOInterface”, “mvc.Koneksi”, “mvc.Model”, dan “mvc.View”.



Gambar 14.13 Menyiapkan package

5. Buat Class Koneksi didalam package mvc.Koneksi (Klik kanan pada mvc.Koneksi → new → Java Class

```

package mvc.Koneksi;
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
import java.sql.Connection;
import java.sql.SQLException;

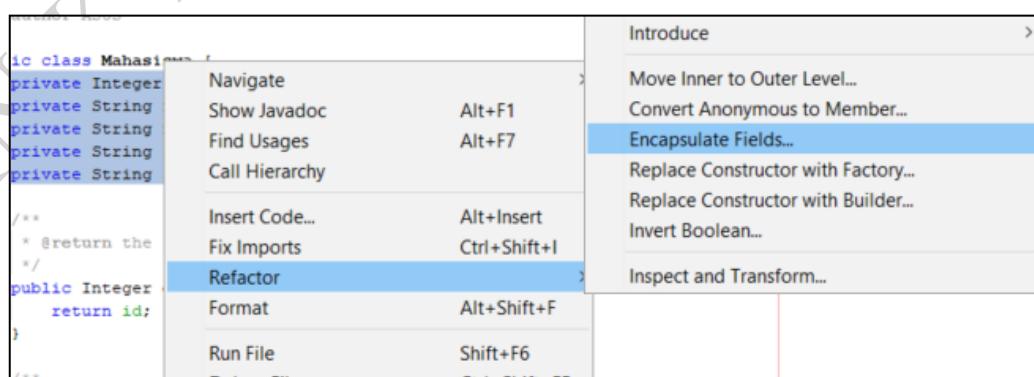
/**
 *
 * @author ASUS
 */
public class Koneksi {
    static Connection con;

    public static Connection connection() {
        if (con == null) {
            MysqlDataSource data = new MysqlDataSource();
            data.setDatabaseName("db_crud");
            data.setUser("root");
            data.setPassword("");
            try {
                con = data.getConnection();
            } catch (SQLException ex) {
                System.out.println("tidak koneksi");
            }
        }
        return con;
    }
}

```

Gambar 14.14 Class Koneksi

- Buat Class Mahasiswa didalam package mvc.Model. Buatlah property private Integer id, private String nim, private String nama, private String jk, dan private String alamat. Setelah itu, blok semua kode diatas → klik kanan → Refactor → Encapsulate Fields → Refactor. Maka akan membuat Getter Setter otomatis.



Gambar 14.15 Getter Setter otomatis

7. Buat Class **TabelModelMahasiswa.java** didalam package **mvc.Model**.

```
package mvc.Model;
import java.util.List;
import javax.swing.table.AbstractTableModel;
/*
 *
 * @author ASUS
 */
public class TabelModelMahasiswa extends AbstractTableModel{
    List<Mahasiswa> lb;

    public TabelModelMahasiswa(List<Mahasiswa> lb) {
        this.lb = lb;
    }

    @Override
    public int getColumnCount() {
        return 5;
    }

    public int getRowCount() {
        return lb.size();
    }
}
```

```
public String getColumnName(int column) {
    switch (column) {
        case 0:
            return "ID";
        case 1:
            return "Nim";
        case 2:
            return "Nama";
        case 3:
            return "Kelamin";
        case 4:
            return "Alamat";
        default:
            return null;
    }
}

@Override
public Object getValueAt(int row, int column) {
    switch (column) {
        case 0:
            return lb.get(row).getId();
        case 1:
            return lb.get(row).getNim();
        case 2:
            return lb.get(row).getNama();
        case 3:
            return lb.get(row).getJk();
        case 4:
            return lb.get(row).getAlamat();
        default:
            return null;
    }
}
```

Gambar 14.16 Class **TabelModelMahasiswa.java**

8. Buat Class interface **IMahasiswa.java** didalam package **mvc.DAOInterface**.

```
package mvc.DAOInterface;

import java.util.List;
import mvc.Model.Mahasiswa;

/*
 *
 * @author ASUS
 */
public interface IMahasiswa {
    public void insert(Mahasiswa b);
    public void update(Mahasiswa b);
    public void delete(int id);
    public List<Mahasiswa> getAll();
    public List<Mahasiswa> getCarinama(String nama);
}
```

Gambar 14.17 Class interface **IMahasiswa.java**

9. Buat Class **DAOMahasiswa.java** didalam package **mvc.DAO** yang berisi CRUD.

```
import mvc.Koneksi.Koneksi;
import mvc.Model.Mahasiswa;
import mvc.DAOInterface.IMahasiswa;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author ASUS
 */
public class DAOMahasiswa implements IMahasiswa{
    Connection connection;
    final String insert = "INSERT INTO tblmahasiswa (nim, nama,jk, alamat) VALUES (?, ?, ?,?)";
    final String update = "UPDATE tblmahasiswa set nim=?, nama=?, jk=?, alamat=? where id=? ;";
    final String delete = "DELETE FROM tblmahasiswa where id=? ;";
    final String select = "SELECT * FROM tblmahasiswa;";
    final String carinama = "SELECT * FROM tblmahasiswa where nama like ?";

    public DAOMahasiswa() {
        connection = Koneksi.connection();
    }

    public void insert(Mahasiswa b) {
        PreparedStatement statement = null;
        try {
            statement = connection.prepareStatement(insert);
            statement.setString(1, b.getNim());
            statement.setString(2, b.getNama());
            statement.setString(3, b.getJk());
            statement.setString(4, b.getAlamat());
            statement.executeUpdate();
            ResultSet rs = statement.getGeneratedKeys();
            while (rs.next()) {
                b.setId(rs.getInt(1));
            }
        } catch (SQLException ex) {
            System.out.println("Berhasil Input");
        } finally {
            try {
                statement.close();
            } catch (SQLException ex) {
                System.out.println("Gagal Input");
            }
        }
    }
}
```

```

public void update(Mahasiswa b) {
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(update);
        statement.setString(1, b.getNim());
        statement.setString(2, b.getNama());
        statement.setString(3, b.getJk());
        statement.setString(4, b.getAlamat());
        statement.setInt(5, b.getId());
        statement.executeUpdate();

    } catch (SQLException ex) {
        System.out.println("Berhasil Update");
    } finally {
        try {
            statement.close();
        } catch (SQLException ex) {
            System.out.println("Gagal Input");
        }
    }
}

public void delete(int id) {
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(delete);

        statement.setInt(1, id);
        statement.executeUpdate();

    } catch (SQLException ex) {
        System.out.println("Berhasil Delete");
    } finally {
        try {
            statement.close();
        } catch (SQLException ex) {
            System.out.println("Gagal Update");
        }
    }
}

```

```

public List<Mahasiswa> getAll() {
    List<Mahasiswa> lb = null;
    try {
        lb = new ArrayList<Mahasiswa>();
        Statement st = connection.createStatement();
        ResultSet rs = st.executeQuery(select);
        while (rs.next()) {
            Mahasiswa b = new Mahasiswa();
            b.setId(rs.getInt("id"));
            b.setNim(rs.getString("nim"));
            b.setNama(rs.getString("nama"));
            b.setJk(rs.getString("jk"));
            b.setAlamat(rs.getString("alamat"));
            lb.add(b);
        }
    } catch (SQLException ex) {
        Logger.getLogger(DAOMahasiswa.class.getName()).log(Level.SEVERE, null, ex);
    }
    return lb;
}

public List<Mahasiswa> getCarinama(String nama) {
    List<Mahasiswa> lb = null;
    try {
        lb = new ArrayList<Mahasiswa>();
        PreparedStatement st = connection.prepareStatement(carinama);
        st.setString(1, "%" + nama + "%");
        ResultSet rs = st.executeQuery();
        while (rs.next()) {
            Mahasiswa b = new Mahasiswa();
            b.setId(rs.getInt("id"));
            b.setNim(rs.getString("nim"));
            b.setNama(rs.getString("nama"));
            b.setJk(rs.getString("jk"));
            b.setAlamat(rs.getString("alamat"));
            lb.add(b);
        }
    } catch (SQLException ex) {
        Logger.getLogger(DAOMahasiswa.class.getName()).log(Level.SEVERE, null, ex);
    }
    return lb;
}

```

Gambar 14.18 Class **DAOMahasiswa.java**

10. Buat Form dengan JFrame Form dialam package **mvc.View** dengan nama **“FormMahasiswa”**. Khusus untuk jkel combo box → properties → Model diganti L,P.

No	Name	Rename
1	JtextField1	txtID
2	JtextField 2	txtNim
3	JtextField 3	txtNama
4	JtextField 4	txtAlamat
5	JtextField 5	txtCariNama
6	JcomboBox1	setJk
7	Jtable	tabelData
8	Jbutton1	buttonInsert
9	Jbutton2	buttonUpdate
10	Jbutton3	buttonDelete
11	Jbutton4	buttonReset
12	Jbutton5	buttonCariNama

Gambar 14.19 GUI dan aturan nama variabel pada **FormMahasiswa**

- Tambahkan kode sebagai properti:
  - ControllerMahasiswa cbt;
- Tambahkan kode didalam Konstruktor FormMahasiswa setelah pemanggilan initComponents():
  - cbt = new ControllerMahasiswa(this);
  - cbt.isiTabel();
- Tambahkan kode Simpan dengan double klik button Simpan:
  - cbt.insert();
  - cbt.isiTabel();

- cbt.reset();
- Tambahkan kode Ubah dengan double klik button Ubah:
  - cbt.update();
  - cbt.isiTable();
  - cbt.reset();
- Tambahkan kode Hapus dengan double klik button Hapus:
  - cbt.delete();
  - cbt.isiTable();
  - cbt.reset();
- Tambahkan kode Batal dengan double klik button Batal:
  - cbt.reset();
- Tambahkan kode Cari dengan double klik button Cari :
  - cbt.carinama();
- Tambahkan beberapa method berikut , jangan lupa import juga

```

public JTextField getTxtID() {
    return txtID;
}
public JTextField getTxtNim() {
    return txtNim;
}
public JTextField getTxtNama() {
    return txtNama;
}
public JComboBox getTxtJk() {
    return setJk;
}
public JTextField getTxtAlamat() {
    return txtAlamat;
}
public JTable getTabelData() {
    return tabelData;
}
public JButton getButtonInsert() {
    return buttonInsert;
}
public JButton getButtonUpdate() {
    return buttonUpdate;
}
public JButton getButtonDelete() {
    return buttonDelete;
}

public JButton getButtonReset(){
    return buttonReset;
}
public JButton getButtonCari(){
    return buttonCariNama;
}
public JTextField getTxtCariNama(){
    return txtCariNama;
}
  
```

Gambar 14.20 Tambahan beberapa method di View

11. Buat Class **ControllerMahasiswa.java** didalam package **mvc.Controller**

```
package mvc.Controller;

import mvc.DAO.DAOMahasiswa;
import mvc.DAOInterface.IMahasiswa;
import mvc.Model.Mahasiswa;
import mvc.Model.TableModelMahasiswa;
import mvc.View.FormMahasiswa;
import java.util.List;
import javax.swing.JOptionPane;
/**
 *
 * @author ASUS
 */
public class ControllerMahasiswa {
    FormMahasiswa frame;
    IMahasiswa implMahasiswa;
    List<Mahasiswa> lb;

    public ControllerMahasiswa(FormMahasiswa frame) {
        this.frame = frame;
        implMahasiswa = new DAOMahasiswa();
        lb = implMahasiswa.getAll();
    }

    //mengosongkan field
    public void reset() {
        frame.getTxtID().setText("");
        frame.getTxtNim().setText("");
        frame.getTxtNama().setText("");
        frame.getTxtJk().setSelectedItem("");
        frame.getTxtAlamat().setText("");
    }

    //menampilkan data ke dalam tabel
    public void isiTable() {
        lb = implMahasiswa.getAll();
        TableModelMahasiswa tmb = new TableModelMahasiswa(lb);
        frame.getTabelData().setModel(tmb);
    }
}
```

```

//merupakan fungsi untuk menampilkan data yang dipilih dari tabel
public void isiField(int row) {
    frame.getTxtID().setText(lb.get(row).getId().toString());
    frame.getTxtNim().setText(lb.get(row).getNim());
    frame.getTxtNama().setText(lb.get(row).getNama());
    frame.getTxtJk().setSelectedItem(lb.get(row).getJk());
    frame.getTxtAlamat().setText(lb.get(row).getAlamat());
}

//merupakan fungsi untuk insert data berdasarkan inputan user dari textfield di frame
public void insert() {
    if (!frame.getTxtNim().getText().trim().isEmpty() & !frame.getTxtNama().getText().trim().isEmpty()) {
        Mahasiswa b = new Mahasiswa();
        b.setNim(frame.getTxtNim().getText());
        b.setNama(frame.getTxtNama().getText());
        b.setJk(frame.getTxtJk().getSelectedItem().toString());
        b.setAlamat(frame.getTxtAlamat().getText());
        implMahasiswa.insert(b);
        JOptionPane.showMessageDialog(null, "Simpan Data sukses");
    } else {
        JOptionPane.showMessageDialog(frame, "Data Tidak Boleh Kosong");
    }
}

//berfungsi untuk update data berdasarkan inputan user dari textfield di frame
public void update() {
if (!frame.getTxtID().getText().trim().isEmpty()) {
    Mahasiswa b = new Mahasiswa();
    b.setNim(frame.getTxtNim().getText());
    b.setNama(frame.getTxtNama().getText());
    b.setJk(frame.getTxtJk().getSelectedItem().toString());
    b.setAlamat(frame.getTxtAlamat().getText());
    b.setId(Integer.parseInt(frame.getTxtID().getText()));
    implMahasiswa.update(b);
    JOptionPane.showMessageDialog(null, "Update Data sukses");
} else {
    JOptionPane.showMessageDialog(frame, "Pilih data yang akan di ubah");
}
}

//berfungsi menghapus data yang dipilih
public void delete() {
    if (!frame.getTxtID().getText().trim().isEmpty()) {
        int id = Integer.parseInt(frame.getTxtID().getText());
        implMahasiswa.delete(id);
        JOptionPane.showMessageDialog(null, "Hapus Data sukses");
    } else {
        JOptionPane.showMessageDialog(frame, "Pilih data yang akan di hapus");
    }
}

public void isiTableCariNama() {
    lb = implMahasiswa.getCariNama(frame.getTxtCariNama().getText());
    TableModelMahasiswa tmb = new TableModelMahasiswa(lb);
    frame.getTabelData().setModel(tmb);
}

```

```

public void carinama() {
    if (!frame.getTxtCariNama().getText().trim().isEmpty()) {
        implMahasiswa.getCariNama(frame.getTxtCariNama().getText());
        isiTableCariNama();
    } else {
        JOptionPane.showMessageDialog(frame, "SILAHKAN PILIH DATA");
    }
}

```

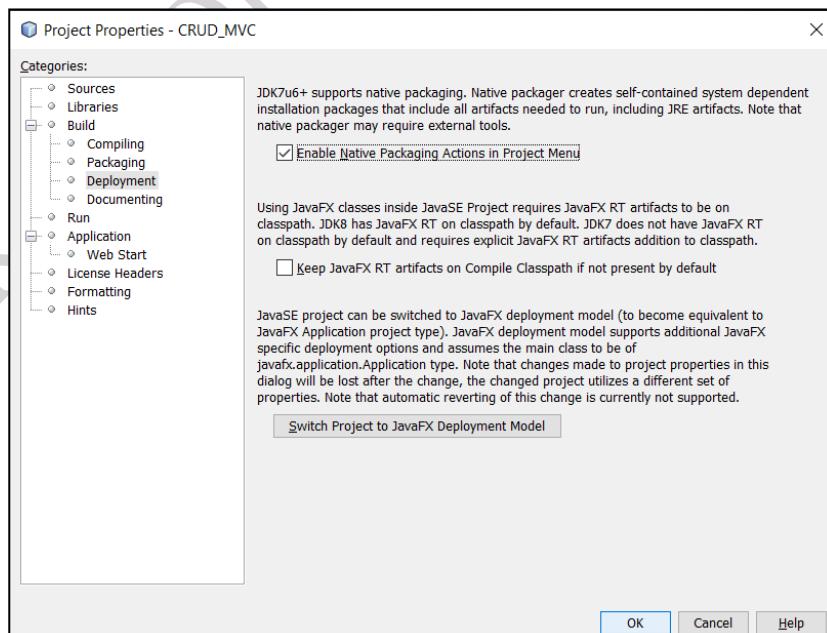
Gambar 14.21 Class ControllerMahasiswa.java

- Run di **FormMahasiswa** sebagai Main Class.

#### 14.4 Deploy Aplikasi

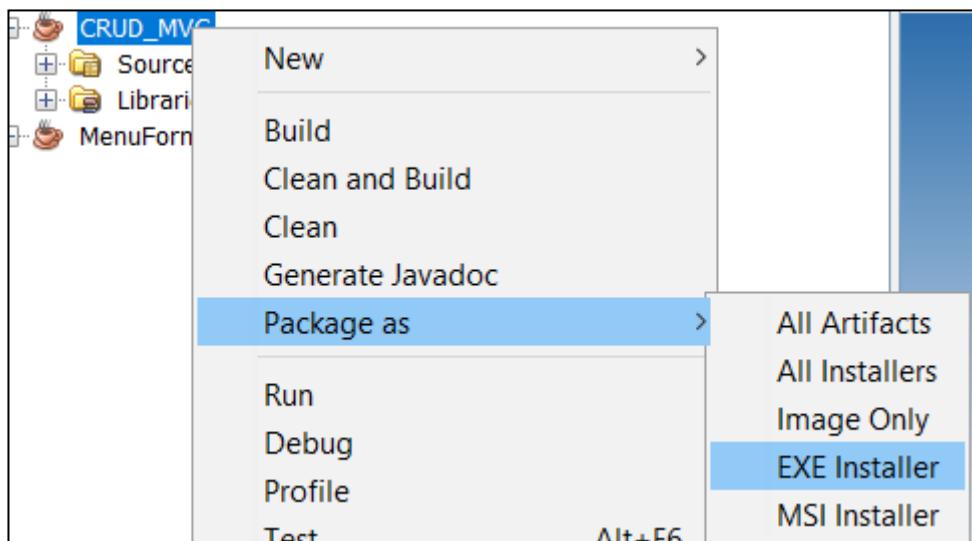
Deploy adalah proses untuk menjadikan program bisa digunakan user. Mari ikuti langkah-langkah berikut ini:

- Install **Inno Setup** dengan mendownload di <http://www.jrsoftware.org/> Pilih versi yang terbaru dan bukan Beta. Install lalu tambahkan ke Path, System Environment variabel: C:\Program Files (x86)\Inno Setup 5 (langkah ini seperti setting javac). Cek di cmd: cmd.exe /c cmd.exe /k iscc.exe
- Aktifkan Native Packaging pada Project.. Klik kanan pada project misal: CRUD\_MVC → Pilih dan klik Properties. Cari Deployment → centang enable native pakaging → oke



Gambar 14.22 Proses deployment

3. Klik kanan pada project → Package as → EXE Installer. Tunggu beberapa saat karena prosesnya cukup lama.



Gambar 14.23 Packaging (exe)

4. Installer (.exe) saved to:

C:\Users\ASUS\Documents\NetBeansProjects\CRUD\_MVC\dist\bundles

NetBeansProjects > CRUD_MVC > dist > bundles		
Name	Date modified	Type
CRUD_MVC-1.0	22/02/2019 09.12	Application

Gambar 14.24 Tempat exe tersimpan

## 14.5 Kesimpulan

Kita bisa menggunakan Java development tools berupa Integrated Development Environment (IDE) seperti Netbeans atau Eclipse untuk membuat, mengedit, compile, run, dan debugging pemrograman Java dengan cepat dengan satu tampilan. Sehingga kita dapat meningkatkan produktivitas kita dalam pemrograman. Komponen-komponen GUI menggunakan Java Swing yang terdiri dari Swing Containers, Swing Controls, Swing Menus, dan Swing Windows.

## **14.6 Kuis dan Latihan Soal**

1. Jelaskan alasan kenapa lebih cepat membuat program Java menggunakan IDE daripada menggunakan text editor?
2. Komponen Java Netbenas apa yang digunakan untuk membuat GUI?
3. Sebutkan 3 isi dari Swing Menus!

## **14.7 Praktikum**

Terdapat suatu event yang diadakan oleh pemerintah yaitu tentang Pendaftaran Duta Coding. Pemerintah membutuhkan sebuah sistem yang dapat mempermudah User untuk mendaftar ketika sudah berada ditempat. Sistem tersebut harus dapat memberikan setidaknya informasi Nomer KTP atau NIK, Nama, Jenis Kelamin, Tempat Tinggal, Usia, dan Alasan kenapa mengikuti acara tersebut. User hanya bisa menginputkan informasi tersebut. Admin dari pegawai pemerintah dapat melihat data User, kemudian dapat melakukan edit dan delete. Perhatikan bahwa user HANYA bisa menginputkan informasi.

**Petunjuk: Ada dua form. Satu untuk user dan satu untuk admin. Keduanya terpisah. Meskipun demikian hanya menggunakan 1 tabel database.**

**Buat aplikasi CRUD.nya yang sudah terhubung dengan database!**

## **DAFTAR PUSTAKA**

Liang, D.J., 2015, Introduction to JAVA Programming Comprehensive Version Tenth Edition, Pearson.

[https://netbeans.org/projects\\_tags/documentation](https://netbeans.org/projects_tags/documentation)

<https://www.tutorialspoint.com/java/index.htm>

[https://www.tutorialspoint.com/design\\_pattern/](https://www.tutorialspoint.com/design_pattern/)

Pratama, M.R., 2007, Implementasi MVC Dengan DAO Pada Java Desktop Application, url:

<https://ilmukomputer.org/wp-content/uploads/2012/06/Mudafiq-Implementasi-MVC.DAO-Java-Desktop.pdf>