MODUL PRAKTIKUM DASAR PENGEMBANGAN PERANGKAT LUNAK

Kode Matakuliah CCJ123

Oleh :

Kundang K Juman

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS ESA UNGGUL

Kata Pengantar

Dalam penyusunan modul praktikum ini, tidak sedikit hambatan yang penulis hadapi. Namun penulis menyadari bahwa kelancaran dalam penyusunan materi ini tidak lain berkat bantuan, dorongan, rekan sejawat dan para Mahasiswa, sehingga kendala-kendala yang penulis hadapi teratasi. Adapun tentang modul praktikum ini telah saya usahakan semaksimal mungkin dan tentunya dengan bantuan berbagai pihak, sehingga dapat memperlancar pembuatan modul ini. Untuk itu saya tidak lupa menyampaikan bayak terima kasih kepada semua pihak yang telah membantu dalam pembuatan modul ini.

Namun tidak lepas dari semua itu, saya menyadari sepenuhnya bahwa ada kekurangan baik dari segi penyusun bahasanya maupun segi lainnya. Oleh karena itu dengan lapang dada dan tangan terbuka saya membuka selebar-lebarnya bagi pembaca yang ingin memberi saran dan kritik kepada saya sehingga dapat memperbaiki modul ini..

Akhirnya penyusun mengharapkan semoga dari modul ini dan pemanfaatannya dapat diambil hikmah dan manfaatnya sehingga dapat memberikan inpirasi terhadap pembaca.

Jakarta 10 April 2018

Penyusun

Kundang K Juman

Konsep Dasar Pengembangan Perangkat Lunak

Konsep dasar rekayasa perangkat lunak mempunyai dua hal pokok yaitu perangkat lunak (software) dan komponen perekayasa. Menurut IEEE definisi perangkat lunak (software) merupakan program komputer, prosedur, data dan semua dokumentasi yang berhubungan operasi pada sistem komputer. jadi bisa disimpulkan bahwa software merupakan kumpulan dari object membentuk konfigurasi yang didalamnya termasuk program, dokumen, dan data. Sedangkan Perekayasa software bertugas mengembangkan produk perangkat lunak, yang secara produk dapat dikategorikan menjadi 2 tipe yaitu : a. Produk generik Sistem stand-alone, produk shrink-wrapped b. Produk pesanan Produk custemisasi, terdapat proses interaksi antara pemesan dan pembuat. Rekayasa perangkat lunak dapat didefinisikan sebagai disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan. Rekayasa perangkat lunak tidak hanya berhubungan dengan proses teknis dari pengembangan perangkat lunak tetapi juga mencakup kegiatan manajemen proyek perangkat lunak dan pengembangan alat bantu, metode dan teori untuk mendukung produksi perangkat lunak. Secara umum rekayasa perangkat lunak memakai pendekatan yang sistematis dan terorganisir dengan menggunakan metode tertentu.

Proses dan Metode Perangkat Lunak

Proses perangkat lunak merupakan serangkaian kegiatan dan hasil hasil relevannya yang menghasilkan perangkat lunak. Kegiatan ini sebagian besar dilakukan oleh perekayasa perangkat lunak. Terdapat empat kegiatan proses dasar, meliputi : 1. Spesifikasi perangkat lunak. Fungsionalitas perangkat lunak dan batasan kemampuan operasinya harus didefinisikan. 2. Pengembangan perangkat lunak. Pengembangan terhadap produk perangkat lunak yang memenuhi spesifikasi perangkat lunak. 3. Validasi perangkat lunak. Perangkat lunak harus divalidasi untuk menjamin bahwa perangkau lunak melakukan apa yang diinginkan oleh user. 4. Evolusi rangkat lunak. Perangkat lunak harus berkembang untuk memenuhi kebutuhan pelanggan yang berubah ubah. Metode rekayasa perangkat lunak

merupakan pendekatan terstruktur terhadap pengembangan perangkat lunak yang bertujuan memfasilitasi produksi perangkat lunak kualitas tinggi dengan cara yang efektif dalam hal biaya. Terdapat beberapa metode yang bisa digunaka seperti metode berorientasi fungsi, metode berorientasi objek dan metode pendekatan gabungan yang sekarang lebih dikenal dengan istilah UML (unified modeling language). 3. Evolusi Perkembangan Software

Era Pertama (1950 – 1960) : Batch Orientation Suatu orientasi di mana proses dilakukan setelah data dikumpulkan dalam satuan waktu tertentu, atau proses dilakukan setelah data terkumpul, lawan dari batch adalah ONLINE atau Interactive Process. Keuntungan dari Interactive adalah mendapatkan data yang selalu up to date. Limmited distribution Suatu penyebaran software yang terbatas pada perusahaan-perusahaan tertentu. Custom software Software yang dikembangkan berdaasarkan perusahaan-perusahaan tertentu. Era Kedua (1960 – 1970) : Multi user Suatu sistem di mana satu komputer digunakan oleh beberapa user pada saat yang sama. Real Time Suatu sistem yang dapat mengumpulkan, menganalisa dan mentransformasikan data dari berbagai sumber, mengontrol proses dan menghasilkan output dalam mili second. Database Perkembangan yang pesat dari alat penyimpan data yang OnLine menyebabkan muncul generasi pertama. DBMS (DataBase Management System). Product Software Adalah software yang dikembangkan untuk dijual kepada masyarakat luas.

Era Ketiga (1980 – 1990) : Distributed system Suatu sistem yang tidak hanya dipusatkan pada komputer induk (Host computer), daerah atau bidang lainnya, yang juga memiliki komputer yang ukurannya lebih kecil dari komputer induk. Lawan dari distributed system adalah Centralized System. Embedded Intelegence Suatu product yang diberi tambahan “Intellegence” dan biasanya ditambahkan mikroprocessor yang mutakhir. Contohnya adalah automobil, robot, peralatan diagnostic serum darah. Low Cost Hardware harga hardware yang semakin rendah, ini dimungkinkan karena munculnya Personal Computer. Consummer Inpact Adanya perkembangan komputer yang murah menyebabkan banyaknya software yang dikembangkan, software ini memberi dampak yang besar terhadap masyarakat. Era Keempat (1990 – 2000) : Expert system

Suatu penerapan A.I. (Artificial Intellegence) pada bidang-bidang tertentu, misalnya bidang kedokteran, komunikasi, dll. AI Machine Suatu mesin yang dapat meniru kerja dari sebagian otak manusia. Misalnya mesin robot, komputer catur. Parallel Architecture Arsitektur komputer yang memungkinkan proses kerja LAN paralel, yang dimungkinkan adanya prosesor berbeda dalam satu komputer.

4. Karakteristik dan Atribut Perangkat Lunak Karakteristik perangkat lunak : a. Software merupakan elemen sistem logik dan bukan elemen sistem fisik seperti hardware. b. Elemen itu tidak aus, tetapi bisa rusak. c. Elemen software itu

direkayasa atau dikembangkan dan bukan dibuat di pabrik seperti hardware d. Software itu tidak bisa dirakit. Atribut perangkat lunak : a. Dapat dipelihara : Perangkat lunak dapat ditulis sedemikian rupa sehingga perangkat lunak dapat berubah seiring dengan perubahan kebutuhan pelanggan. b. Dapat diandalkan : Perangkat lunak mempunyai serangkaian karakteristik, termasuk keandalan, keamanan dan keselamatan. c. Efisien : Perangkat lunak tidak boleh menggunakan sumber daya sistem seperti siklus memori dan prosesor. d. Kemampupakaian : Perangkat lunak harus dapat dipakai, memiliki interface user yang bagus dan dokumentasi yang mencukupi. 5. Tanggung Jawab Profesional dan Etika Rekayasa perangkat lunak jelas dibatasi oleh hukum lokal, nasional dan internasional. Perekayasa perangkat lunak harus memiliki tanggung jawab etis dan moral jika ingin dihormati sebagai profesional. Terdapat beberapa standar dan kode etik yang harus

dipertimbangkan, yaitu: a. Konfidensialitas Harus menghormati konfidensialitas atasan dan kliennya walaupun tidak ada persetujuan yang ditanda tangani secara formal. b. Kompetensi Tidak boleh menyalahi tingkat kompetensinya (melebihi atau menyimpang) c. Hak Properti Intelektual Menyadari terhadap hukum yang mengatur penggunaan properti intelektual, seperti paten, hak cipta dan lain sebagainya. d. Penyalahgunaan Komputer Tidak boleh dengan sengaja menyalah gunakan komputer yang nantinya berakibat merugikan orang lain, seperti penyebaran virus, penyadapan dan lain sebagainya. 6. Model Proses Perangkat Lunak Model proses perangkat lunak merupakan deskripsi yang disederhanakan dari proses perangkat lunak yang di presentasikan dengan sudut pandang tertentu. Model, sesuai sifatnya merupakan penyederhanaan, sehingga model proses perangkat lunak merupakan abstraksi dari proses sebenarnya yang dideskripsikan. Model proses juga bisa mencakup kegiatan yang merupakan bagian dari proses perangkat lunak, produk perangkat lunak dan peran orang yang

terlibat pada rekayasa perangkat lunak. Ada beberapa contoh jenis model proses perangkat lunak, antara lain: a. Model aliran kerja (work flow) Model ini memandang proses dari urutan dan prosedur kerja (input, output dan ketergantungannya). b. Model aliran data (data flow) Model ini merepresentasikan proses sebagai satu set kegiatan yang masing masing melakukan transformasi data. c. Model peran/aksi Model ini merepresentasikan peran orang yang terlibat pada proses perangkat lunak dan kegiatan yang menjadi tanggung jawabnya dalam penyelesaian sebuah sistem. Life Cycle Life-cycle sebuah perangkat lunak mencakup semua kegiatan yang yang perlu dilakukan untuk mendefinisikan, mengembangkan, menguji, mengantarkan, mengoperasikan, dan memelihara produk perangkat lunak. Beberapa model yang akan dibahas adalah : model fase (phased model), model biaya (cost model), model prototipe (prototype model), dan model berurutan (successive model).

a. Model Fase Model ini membagi life cycle ke dalam sederetan kegiatan (fase). Setiap fase membutuhkan informasi masukan, proses, dan produk yang terdefinisi dengan baik. Deretan fase tersebut adalah : analisa, perancangan,

implementasi, pengujian, dan pemeliharaan. Berikut ini model fase dasar yang dinyatakan sebagai waterfall chart :

Daftar Simbol pada *Flowchart*

Simbol	Keterangan
	*Terminator* Menggambarkan kegiatan awal atau akhir dari suatu proses.
	*Proses* Menggambarkan suatu proses.
	*Data* Menggambarkan kegiatan masukan atau keluaran yang dihasilkan.
	*Decision* Menggambarkan suatu keputusan atau tindakan yang harus diambil pada kondisi tertentu.
	*Predefine Proses* Menggambarkan proses-proses yang masih bisa dijabarkan dalam algoritma.
	*Line Connector* Menghubungkan suatu simbol dengan simbol lain pada modul yang sama.
	*On-page Reference* Menghubungkan suatu simbol dengan simbol yang lainnya pada halaman yang sama.
	*Off-page Referance* Menghubungkan suatu simbol dengan simbol yang lainnya pada halaman yang berbeda.

Contoh Program Sederhana Penulisan C, C++ dan Java

Ada banyak compiler C, antara lain Turbo C/C++ buatan Borland C, Microsoft C dan Microsoft C++, yang dibuat oleh Microsoft Co. Bahasa pemrograman C , C++ dan java mempunyai banyak persamaan, seperti contoh program menghitung total dua buah bilangan (misal 5 dan 2), kemudian mencetak total tersebut, yang ditulis dalam Bahasa C, C+

Bahasa pemrograman

Bahasa atau dalam bahasa inggris language adalah suatu sistim untuk berkomunikasi. Bahasa tertulis menggunakan simbol (yaitu huruf) untuk membentuk kata. Dalam ilmu komputer,bahasa manusia disebut bahasa alamiah (natural

languages), dimana komputer tidak bisa memahaminya, sehingga diperlukan suatu bahasa komputer. Komputer mengerjakan transformasi data berdasarkan kumpulan perintah - program - yang telah dibuat oleh pemrogram. Kumpulan perintah ini harus dimengerti oleh komputer, berstruktur tertentu (syntax) dan bermakna. Bahasa pemrograman merupakan notasi untuk memberikan secara tepat program komputer Bahasa pemprograman Java pada awalnya dibuat oleh James Gosling pada tahun 1995 sebagai bagian dari Sun Microsystem Java Platform. Sintaks Java banyak diturunkan dari C dan C++ tetapi lebih sederhana, ketat dan umumnya mempunyai akses ke OS yang lebih terbatas. Hal ini karena Java ditujukan sebagai bahasa pemprograman yang cukup sederhana untuk dipelajari dan mudah dibaca.Aplikasi Java ditulis sebagai file berekstensi .java yang dicomplai menjadi .class .file .class ini adalah bytecode yang bisa jalankan di semua Java Virtual Machine, tidak peduli apapun OS-nya ataupun arsitektur processornya. Java adalah bahasa yang ditujukan untuk semua kebutuhan, concurent, berbasis class, object oriented serta didesain agar tidak tergantung terhadap lingkungan dimana aplikasi dijalankan (OS dan processor

Modul Pertemua 1 Java :

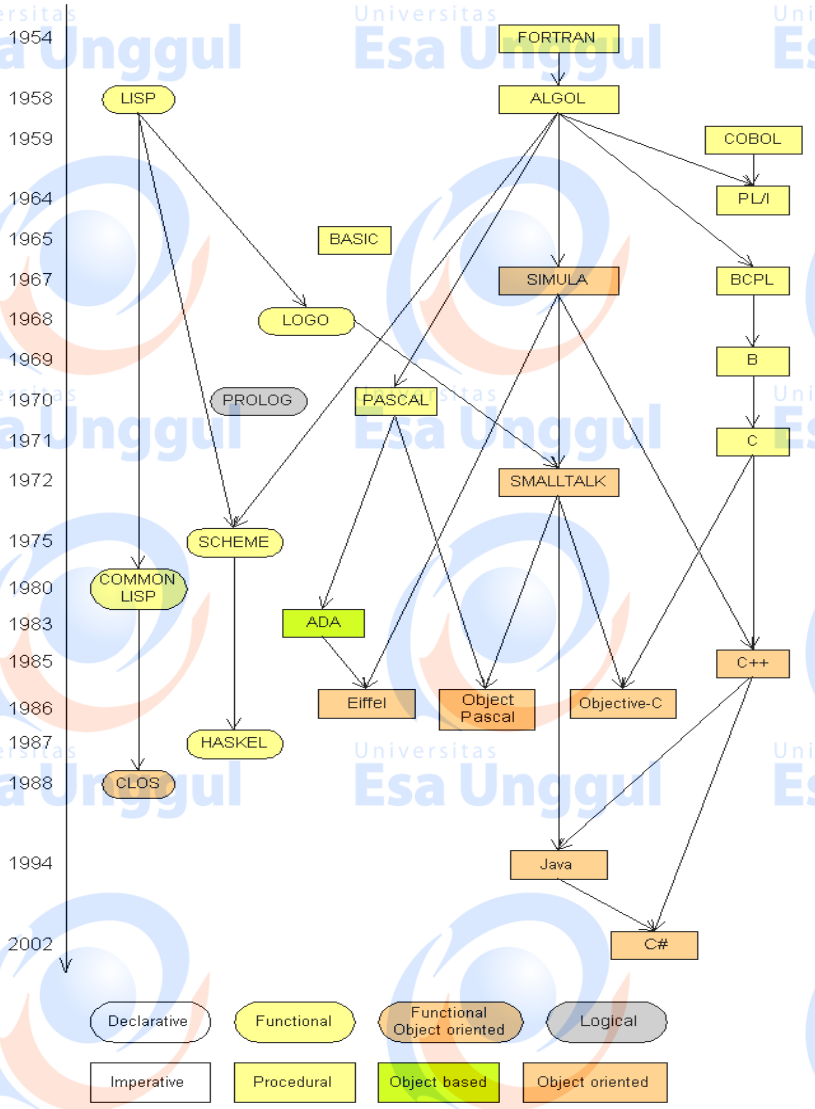
Introduction to Java Programming

What is computer science?

- Computer Science
  - The study of theoretical foundations of information and computation and their implementation and application in computer systems. -- Wikipedia
  - Many subfields
    - Graphics, Computer Vision
    - Artificial Intelligence
    - Scientific Computing
    - Robotics
    - Databases, Data Mining
    - Computational Linguistics, Natural Language Processing ...
  - Computer Engineering
  - Overlap with CS and EE; emphasizes hardware

- **program**: A set of instructions to be carried out by a computer.
- **Programming languages**

- **program execution**: The act of carrying out the instructions contained in a program.
- **programming language**: A systematic set of rules used to describe computations in a format that is editable by humans.
  - This textbook teaches programming in a language named Java.
- Some influential ones:
- FORTRAN
- science / engineering
- COBOL
- business data
- LISP
- logic and AI
- BASIC
- a simple language



Basic Java programs with println statements

Compile/run a program

1. *Write* it.
   - **code** or **source code**: The set of instructions in a program.
   - *Compile* it.
   - **compile**: Translate a program from one language to another.
   - **byte code**: The Java compiler converts your code into a format named *byte code* that runs on many computer types.
   - *Run* (execute) it.
   - **output**: The messages printed to the user by a program.

A Java program

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
    System.out.println();
    System.out.println("This program produces");
    System.out.println("four lines of output");
  }
}
```

- Its output:

```
Hello, world!

This program produces
four lines of output
```

console: Text box into which the program's output is printed

Structure of a Java program

```
public class name {
  public static void main(String[] args) {
    statement;
    statement;
    ...
    statement;
  }
}
```

- Every executable Java program consists of a **class**,
  - that contains a **method** named main,
    - that contains the **statements** (commands) to be execute

System.out.println :

- A statement that prints a line of output on the console.
  - pronounced "print-linn"
  - sometimes called a "println statement" for short
- Two ways to use System.out.println :
  - System.out.println("**text**");

    Prints the given message as output.
  - System.out.println();

    Prints a blank line of output.

Names and identifiers :

- You must give your program a name.

public class **GangstaRap** {

  - Naming convention: capitalize each word (e.g. MyClassName)
  - Your program's file must match exactly (GangstaRap.java)
    - includes capitalization (Java is "case-sensitive")
  - **identifier**: A name given to an item in your program.
  - must start with a letter or _ or $
  - subsequent characters can be any of those or a number
    - legal: _myName TheCure ANSWER_IS_42 $bling$
    - illegal: me+u 49ers side-swipe Ph.D's

Syntax error example

```
1  public class Hello {
2     pooblic static void main(String[] args) {
3        System.owt.println("Hello, world!")_
4     }
5  }
```

- Compiler output:

```
Hello.java:2: <identifier> expected
   pooblic static void main(String[] args) {
      ^
Hello.java:3: ';' expected
}
^
2 errors
```

  - The compiler shows the line number where it found the error.
  - The error messages can be tough to understand!

Strings

- **string**: A sequence of characters to be printed.
  - Starts and ends with a " quote " character.
    - The quotes do not appear in the output.
  - Examples:

    "hello"

    "This is a string. It's very long!"
- Restrictions:
  - May not span multiple lines.

"This is not
a legal String."

- May not contain a " character.

"This is not a "legal" String either."

Escape sequences :

- **escape sequence: A special sequence of characters used to represent certain special characters in a string.**

\t tab character
\n new line character
\" quotation mark character
\\ backslash character

- **Example:**
  System.out.println("\\hello\nhow\tare \"you\"?\\\\");
- **Output:**
  \hello
  how are "you"?

Program version

```
// Suzy Student, CSE 138, Spring 2094
// Prints several figures, with methods for structure and redundancy.
public class Figures3 {
  public static void main(String[] args) {
    egg();
    teaCup();
    stopSign();
    hat();
  }

  // Draws the top half of an an egg figure.
  public static void eggTop() {
    System.out.println("  ______");
    System.out.println(" /      \\");
    System.out.println("/       \\");
  }

  // Draws the bottom half of an egg figure.
  public static void eggBottom() {
    System.out.println("\\       /");
    System.out.println(" \\_____/");
  }

  // Draws a complete egg figure.
  public static void egg() {
    eggTop();
    eggBottom();
    System.out.println();
  }
```

```
// Draws a teacup figure.
   public static void teaCup() {
      eggBottom();
      line();
      System.out.println();
   }

   // Draws a stop sign figure.
   public static void stopSign() {
      eggTop();
      System.out.println("|  STOP  |");
      eggBottom();
      System.out.println();
   }

   // Draws a figure that looks sort of like a hat.
   public static void hat() {
      eggTop();
      line();
   }
   // Draws a line of dashes.
   public static void line() {
      System.out.println("+--------+");
   }
}
```

Soal latihan 1 :

```
import java.awt.Graphics;

public class HelloWorldApplet extends java.applet.Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Apa Kabar Dunia?", 5, 25);
    }
}
```

Dan ini sebuah contoh lain, yaitu applet sederhana untuk menampilkan teks di applet. Sebutlah file ini bernama `HelloWorldApplet.java`:

```
import java.awt.Graphics;
```

Modul 2 :

Classes

A programming problem

- **Given a file of cities' (x, y) coordinates, which begins with the number of cities:**

```
6
50 20
90 60
10 72
74 98
5 136
150 91
```

- **Write a program to draw the cities on a DrawingPanel, then drop a "bomb" that turns all cities red that are within a given radius:**

```
Blast site x? 100
Blast site y? 100
Blast radius? 75
Kaboom!
```

A bad solution

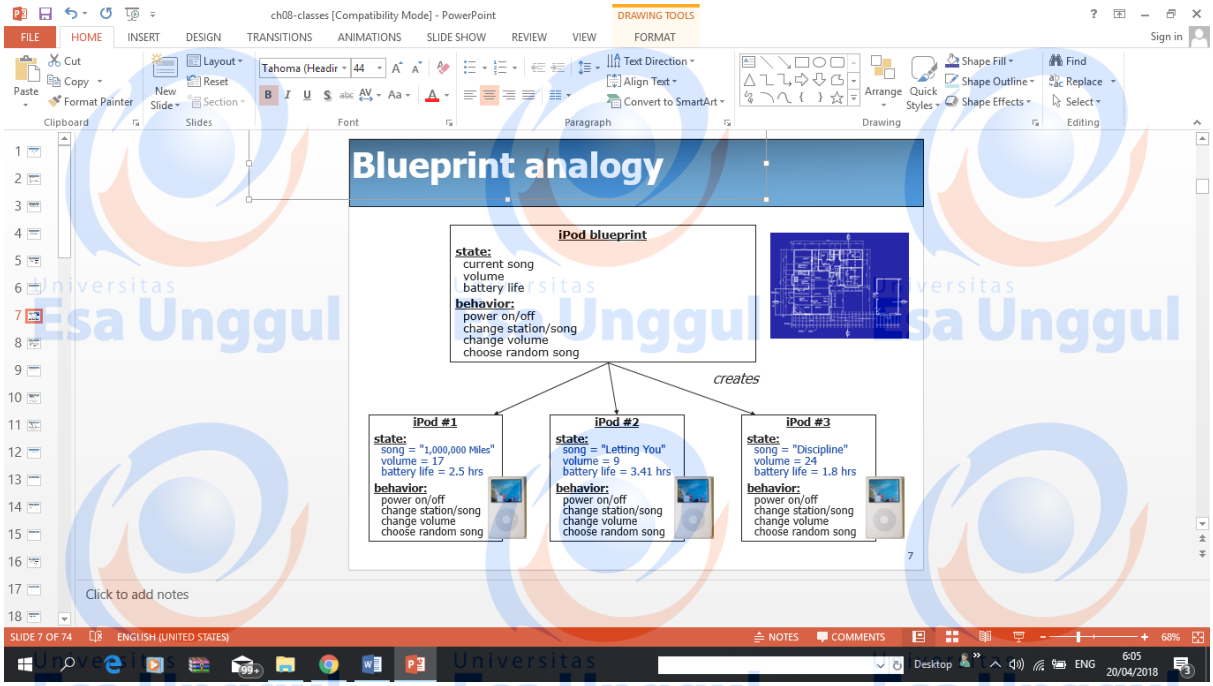
```
Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];
for (int i = 0; i < cityCount; i++) {
  xCoords[i] = input.nextInt();   // read each city
  yCoords[i] = input.nextInt();
}
...
```

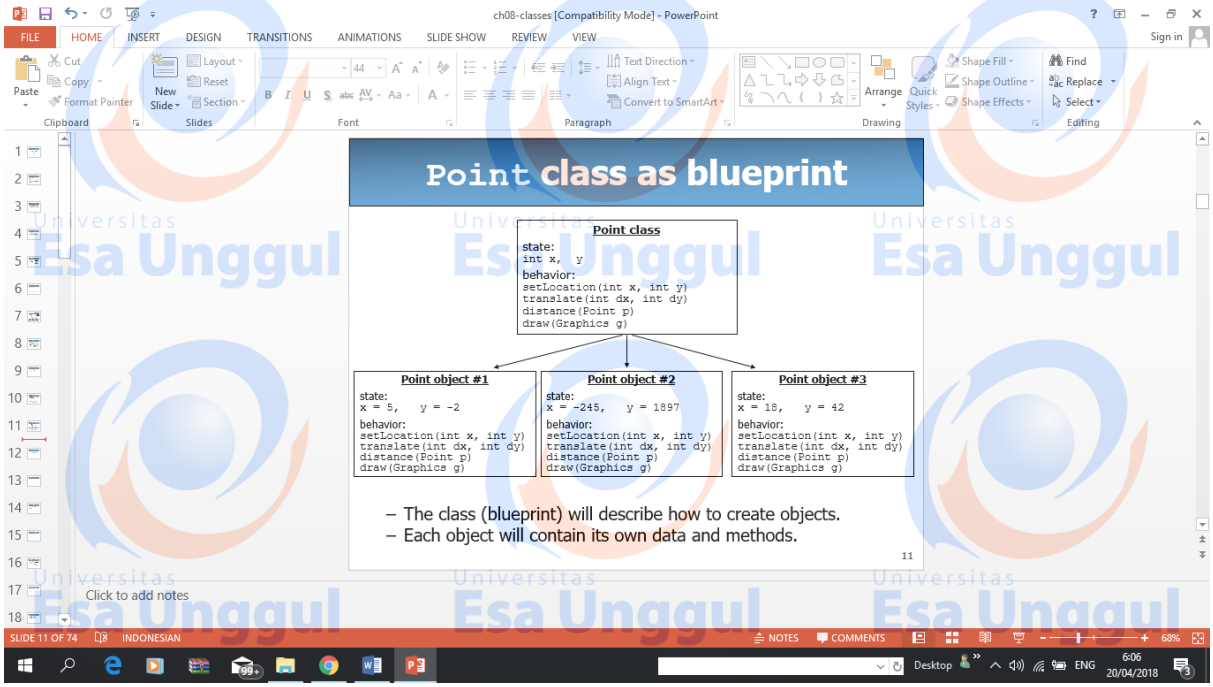
  - **parallel arrays: 2+ arrays with related data at same indexes.**
    - **Considered poor style.**

Classes and objects

- **class: A program entity that represents either:**
  1. **A program / module, or**
  2. **A template for a new type of objects.**
  - **The DrawingPanel class is a template for creating DrawingPanel objects.**
- **object: An entity that combines state and behavior.**
  - **object-oriented programming (OOP): Programs that perform their behavior as interactions between objects.**

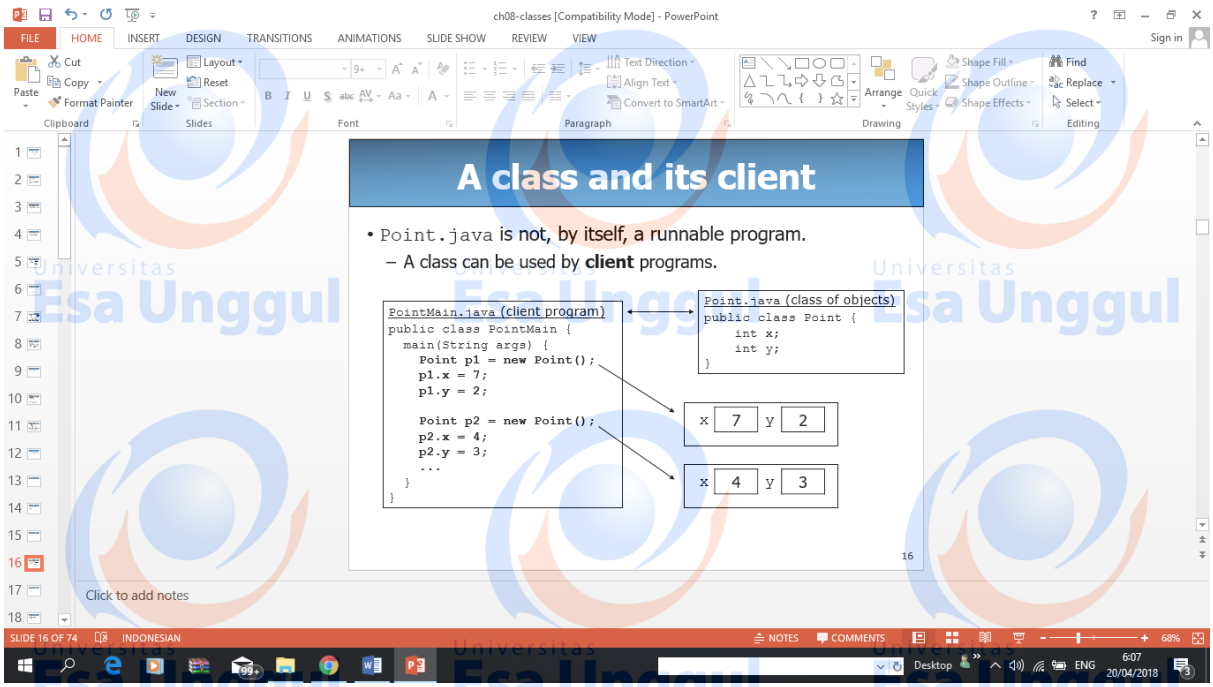
- **class: A program entity that represents either:**
  1. **A program / module, or**
  2. **A template for a new type of objects.**
  - **The DrawingPanel class is a template for creating DrawingPanel objects.**
- **object: An entity that combines state and behavior.**
  - **object-oriented programming (OOP): Programs that perform their behavior as interactions between objects.**


ch08-classes [Compatibility Mode] - PowerPoint
Blueprint analogy
iPod blueprint
state:
current song
volume
battery life
behavior:
power on/off
change station/song
change volume
choose random song
creates
iPod #1
state:
song = "1,000,000 Miles"
volume = 17
battery life = 2.5 hrs
behavior:
power on/off
change station/song
change volume
choose random song
iPod #2
state:
song = "Letting You"
volume = 9
battery life = 3.41 hrs
behavior:
power on/off
change station/song
change volume
choose random song
iPod #3
state:
song = "Discipline"
volume = 24
battery life = 1.8 hrs
behavior:
power on/off
change station/song
change volume
choose random song
Click to add notes


ch08-classes [Compatibility Mode] - PowerPoint
Point class as blueprint
Point class
state:
int x, y
behavior:
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
Point object #1
state:
x = 5, y = -2
behavior:
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
Point object #2
state:
x = -245, y = 1897
behavior:
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
Point object #3
state:
x = 18, y = 42
behavior:
setLocation(int x, int y)
translate(int dx, int dy)
distance(Point p)
draw(Graphics g)
– The class (blueprint) will describe how to create objects.
– Each object will contain its own data and methods.
Click to add notes

- Other classes can access/modify an object's fields.
  - access: **variable.field**
  - modify: **variable.field = value;**
  - Example:

```
Point p1 = new Point();
Point p2 = new Point();
System.out.println("the x-coord is " + p1.x);   // access
p2.y = 13;                                      // modify
```



Point sample :

```
public class PointMain {
   public static void main(String[] args) {
      // create two Point objects
      Point p1 = new Point();
      p1.y = 2;
      Point p2 = new Point();
      p2.x = 4;
      System.out.println(p1.x + ", " + p1.y);   // 0, 2
      // move p2 and then print it
      p2.x += 2;
      p2.y++;
      System.out.println(p2.x + ", " + p2.y);   // 6, 1
   }
}
```

- Exercise: Modify the Bomb program to use Point objects.

```
public class PointMain {
```

```
  public static void main(String[] args) {
    // create two Point objects
    Point p1 = new Point();
    p1.y = 2;
    Point p2 = new Point();
    p2.x = 4;
    System.out.println(p1.x + ", " + p1.y);   // 0, 2
    // move p2 and then print it
    p2.x += 2;
    p2.y++;
    System.out.println(p2.x + ", " + p2.y);   // 6, 1
  }
}
```

- Exercise: Modify the Bomb program to use Point objects.

Latihan 2

Dalam **Listing 1**, pernyataan `System.out.println("Apa Kabar Dunia?");` berarti carilah objek `out` dalam kelas `System` kemudian panggil metode `println` dari objek `out` dengan parameter berupa string `"Apa Kabar Dunia?"`. Sedang dalam **Listing 2**, pernyataan `g.drawString("Apa Kabar Dunia?", 5, 25);` berarti carilah objek `g` kemudian panggil metode `drawString` pada objek `g` dengan parameter `"Apa Kabar Dunia?", 5, 25);`.

Eksekusi

Setelah selesai membahas sintaks dasar Java dalam kedua listing, selanjutnya kita akan mencoba mengeksekusi kedua program ini. Untuk program pertama yang berupa aplikasi biasa, kita tinggal mengetikkan perintah `java HelloWorld` pada prompt dan pesan `Apa Kabar Dunia?` akan tampil di layar (atau mungkin di tempat lain, bergantung sistem operasi Anda). Sedangkan untuk applet kita mesti membuat sebuah file HTML sebagai pembungkus—atau pemanggilnya. Berikut diberikan contoh file HTML untuk membungkus applet yang kita buat.

```
<HTML>
  <HEAD>
    <TITLE>Coba Applet</TITLE>
  </HEAD>
  <BODY>
    <APPLET CODE="HelloWorldApplet.class" WIDTH=150
HEIGHT=25>
    </APPLET>
  </BODY>
</HTML>
```

Beri nama `helloword.html` dan simpanlah di direktori yang sama dengan lokasi file-file `.java` dan `.class` sebelumnya. Untuk mengeksekusi applet kita cukup membuka file HTML

tersebut di browser yang Java-enabled atau mengetikkan perintah `appletviewer namafile.html` di prompt.

MODUL 3. ArrayList

Cara Membuat Array di Java

Cara membuat array kosong:

```
// cara pertama
String[] nama;

// cara kedua
String nama[];

// cara ketiga dengan kata kunci new
String[] nama = new String[5];
```

Parhatikan:

- Kita menggunakan kurung siku `[]` untuk membuat array;
- Kurung siku bisa diletakkan setelah tipe data atau nama array;
- Angka `5` dalam kurung artinya batas atau ukuran array-nya.

Array yang kosong siap diisi dengan data. Pastikan mengisinya dengan data yang sesuai dengan tipe datanya.

Kita bisa mengisinya seperti ini:

```
nama[0] = "Linda";
nama[1] = "Santi";
nama[2] = "Susan";
nama[3] = "Mila";
nama[4] = "Ayu";
```

Atau kalau tidak mau repot, kita bisa membuat array dan langsung mengisinya.

```
String[] nama = {"Linda", "Santi", "Susan", "Mila", "Ayu"};
```

Mengambil Data dari Array

Seperti yang sudah kita ketahui, array memiliki indeks untuk memudahkan kita mengakses datanya.

Karena itu, kita bisa mengambil datanya dengan cara seperti ini:

```
// membuat array
String[] nama = {"Linda", "Santi", "Susan", "Mila", "Ayu"};

// mengambil data array
System.out.println(teman[2]);
```

Kira-kira apa hasil outputnya?

Yep! benar sekali, hasil outputnya adalah:

```
Susan
```

Karena `Susan` terletak di indeks ke-2.

Menggunakan Perulangan

Mengambil data satu per satu dari array mungkin cukup melelahkan, karena kita harus mengtik ulang nama array-nya dengan indeks yang berbeda.

Contoh:

```
System.out.println(teman[0]);
System.out.println(teman[1]);
System.out.println(teman[2]);
System.out.println(teman[3]);
```

Bagaimana kalau data array-nya sampai 1000, maka kita harus mengetik kode sebanyak seribu kali.

Karena itu, disinilah peran perulangan.

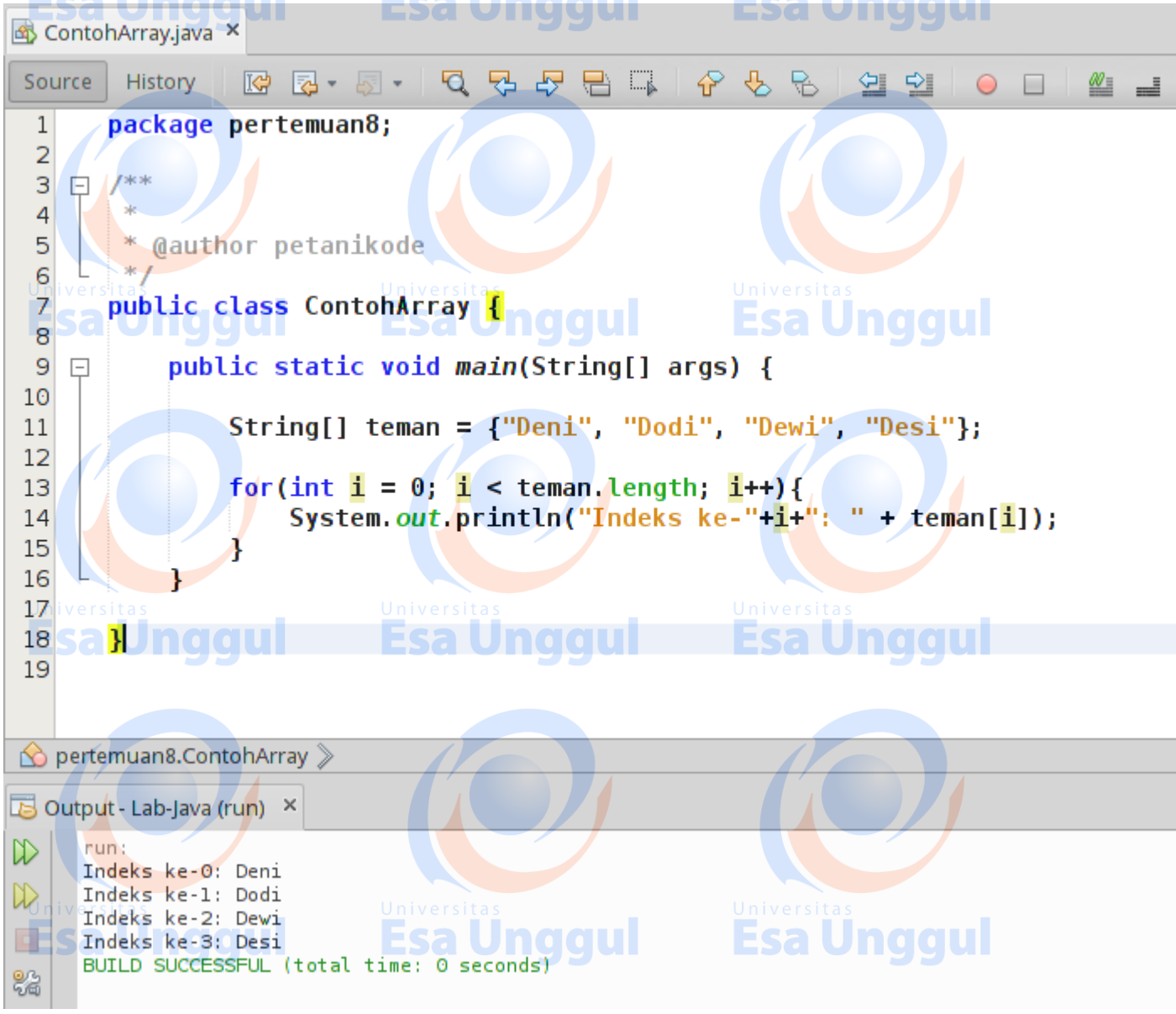
ArrayList as parameter :

- Example:

```
// Removes all plural words from the given list.
public static void removePlural(ArrayList<String> list) {
   for (int i = 0; i < list.size(); i++) {
      String str = list.get(i);
      if (str.endsWith("s")) {
         list.remove(i);
         i--;
      }
   }
}
```

- You can also return a list:

public static ArrayList<**Type**> **methodName**(**params**)



Perhatikan:

Di sana kita menggunakan atribut `length` untuk mengambil panjang array-nya.

Jadi, perulangan akan dilakukan sebanyak isi array-nya.

Sekarang Mari Kita Latihan

Silahkan buat class bernama `Buah`, kemudian ikuti kode berikut:

```
import java.util.Scanner;

public class Buah {
    public static void main(String[] args) {

        // membuat array buah-buahan
        String[] buah = new String[5];

        // membuat scanner
        Scanner scan = new Scanner(System.in);

        // mengisi data ke array
        for( int i = 0; i < buah.length; i++ ){
            System.out.print("Buah ke-" + i + ": ");
            buah[i] = scan.nextLine();
        }

        System.out.println("--------------------------");

        // menampilkan semua isi array
        for( String b : buah ){
            System.out.println(b);
        }

    }
}
```

Hasil outputnya:

```
run:
Buah ke-0: Apel
Buah ke-1: Anggur
Buah ke-2: Manggis
Buah ke-3: Jeruk
Buah ke-4: Semangka
--------------------------
Apel
Anggur
Manggis
Jeruk
Semangka
BUILD SUCCESSFUL (total time: 11 seconds)
```

Perhatikan:

Di sana kita menggunakan perulangan *foreach* untuk menampilkan isi array.

Seperti yang sudah kita pelajari pada materi Perulangan di Java, perulangan ini dapat kita gunakan untuk menampilkan isi array.

Praktek : Latihan Perulangan dan Array pada Java

Buatlah program untuk menginputkan beberapa buah nilai, dan mencari total nilai serta rata rata dari nilai yang di masukkan.

Jawaban kasus yaitu :

Nama file rata2.java

```
1  import java.io.* ;
2  public class rata2{
3      public static void main(String Arg[]) throws IOException{
4
5          int nilai[ ] ;
6          nilai = new int[20] ;
7          double rata = 0, total = 0 ;
8
9          String str  ;
10         int i, jdata ;
11
12         BufferedReader Get=new BufferedReader(new
   InputStreamReader(System.in)) ;
13
14         System.out.println("Menghitung Nilai Rata Rata") ;
15
16         System.out.print("Jumlah Data :")  ; str = Get.readLine() ;
17         jdata = Integer.parseInt(str) ;
18
19         for (i = 0; i<jdata ; i++){
           System.out.print("Nilai ke : "+ (i+1) + " : ") ;
```

```
20                  str = Get.readLine() ;
21                  nilai[i] = Integer.parseInt(str) ;
22              }
23
24              rata = 0 ; total = 0 ;
25              System.out.println("Menghitung Nilai Rata Rata") ;
26
27              for (i = 0; i<jdata ; i++){
28                  System.out.println("Nilai ke : "+ (i+1) + " : "+nilai[i])
    ;
29                  total = total + nilai[i] ;
30              }
31
32          rata = total / jdata ;
33              System.out.println("Total Nilai : "+ total) ;
34              System.out.println("Nilai Rata-Rata : "+ rata) ;
35
36      }
    }
37
38
```

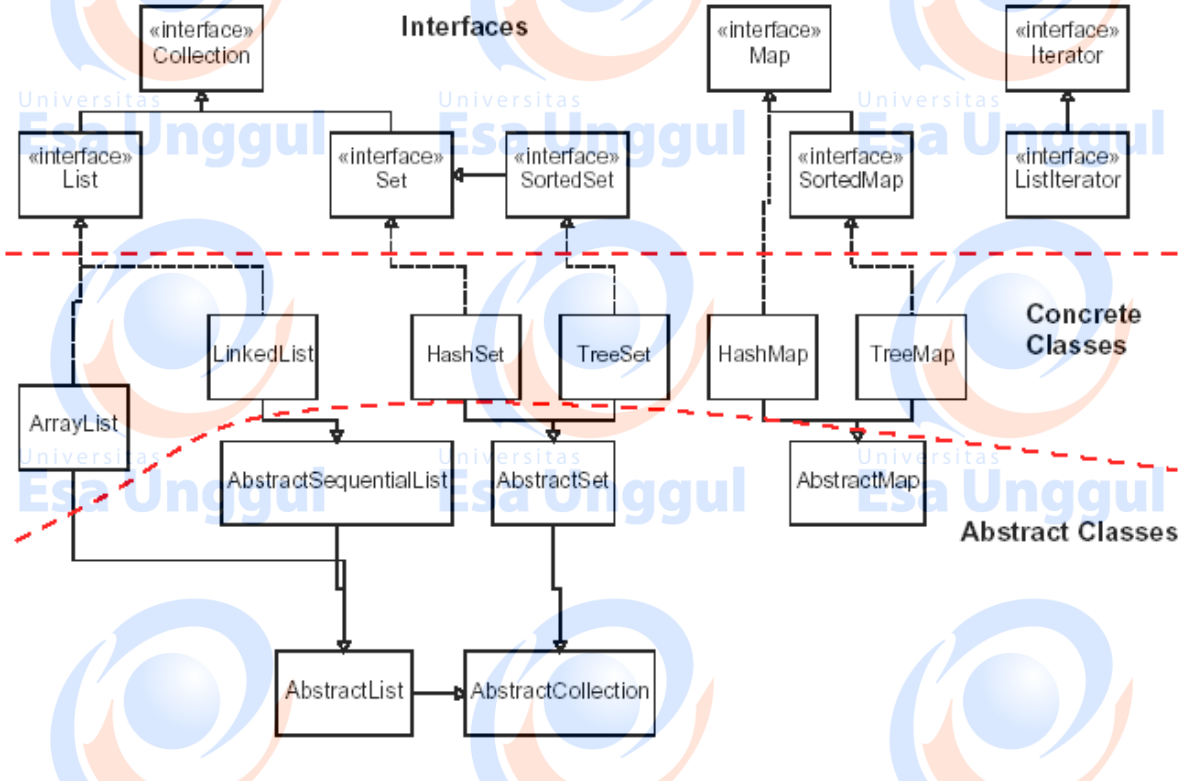
Hasil Program :

```
C:\WINDOWS\system32\cmd.exe

C:\java>java rata2
Menghitung Nilai Rata Rata
Jumlah Data :10
Nilai ke : 1 : 70
Nilai ke : 2 : 50
Nilai ke : 3 : 60
Nilai ke : 4 : 78
Nilai ke : 5 : 90
Nilai ke : 6 : 40
Nilai ke : 7 : 50
Nilai ke : 8 : 30
Nilai ke : 9 : 50
Nilai ke : 10 : 67
Menghitung Nilai Rata Rata
Nilai ke : 1 : 70
Nilai ke : 2 : 50
Nilai ke : 3 : 60
Nilai ke : 4 : 78
Nilai ke : 5 : 90
Nilai ke : 6 : 40
Nilai ke : 7 : 50
Nilai ke : 8 : 30
Nilai ke : 9 : 50
Nilai ke : 10 : 67
Total Nilai : 585.0
Nilai Rata-Rata : 58.5
```

MODUL 4

Java collections framework



SET Methode

```
List<String> list = new ArrayList<String>();
...
Set<Integer> set = new TreeSet<Integer>();     // empty
Set<String> set2 = new HashSet<String>(list);
```

- can construct an empty set, or one based on a given collection

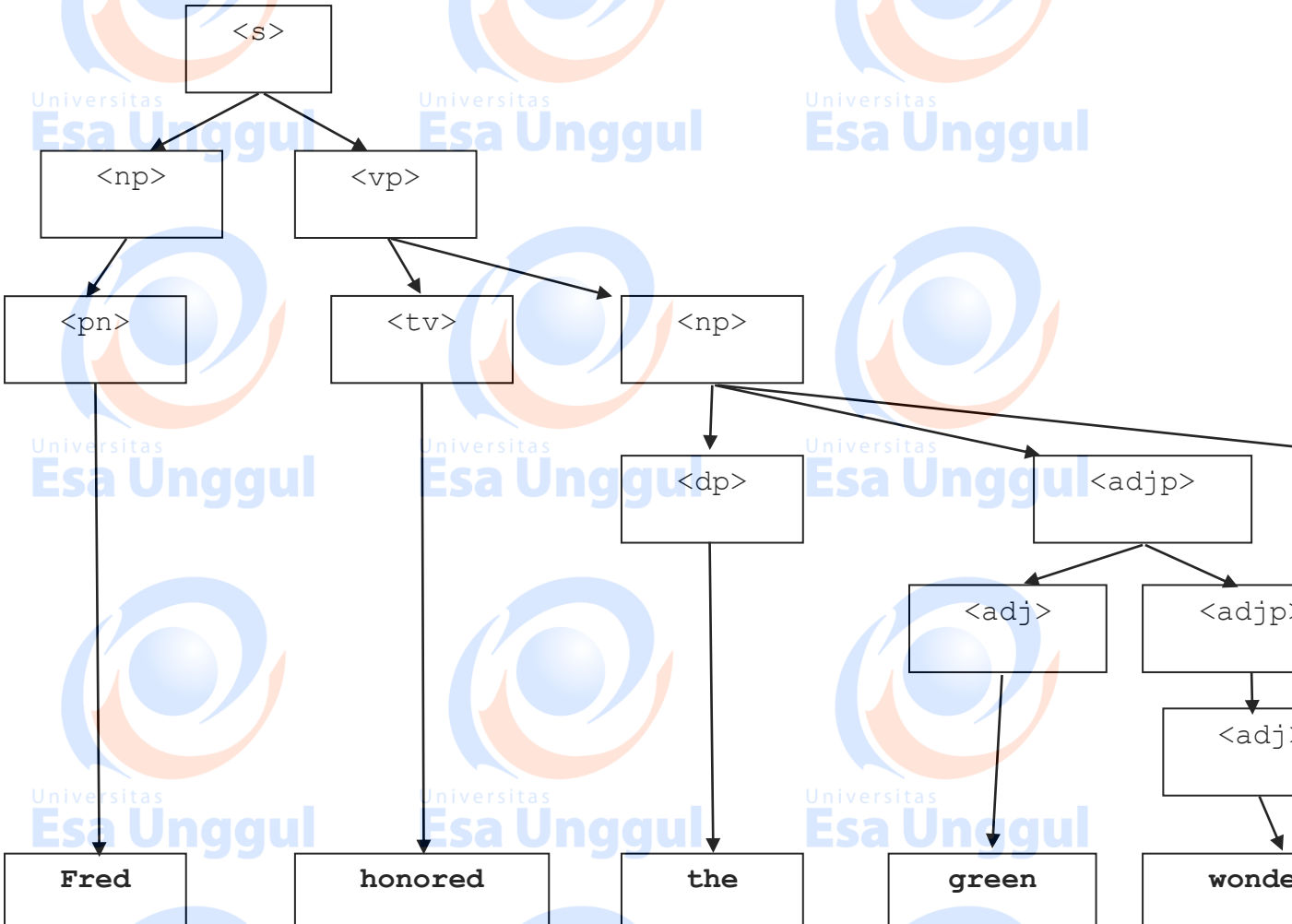
```
List<String> list = new ArrayList<String>();
...
Set<Integer> set = new TreeSet<Integer>();
Set<String> set2 = new HashSet<String>(list);
```

- can construct an empty set, or one based on a given co

Exercise solution :

```
// read file into a map of [word --> number of occurrences]
Map<String, Integer> wordCount = new HashMap<String, Integer>();
Scanner input = new Scanner(new File("mobydick.txt"));
while (input.hasNext()) {
   String word = input.next();
   if (wordCount.containsKey(word)) {
      // seen this word before; increase count by 1
      int count = wordCount.get(word);
      wordCount.put(word, count + 1);
```

Sentence generation



Modul 5

Recursion

- Consider the following method to print a line of * characters:

```
// Prints a line containing the given number of stars.
// Precondition: n >= 0
public static void printStars(int n) {
  for (int i = 0; i < n; i++) {
    System.out.print("*");
  }
  System.out.println();  // end the line of output
}
```

- Write a recursive version of this method (that calls itself).

- Solve the problem without using any loops.
- Hint: Your solution should print just one star at a time.

- Condensing the recursive cases into a single case:

```
public static void printStars(int n) {
  if (n == 1) {
    // base case; just print one star
    System.out.println("*");
  } else {
    // recursive case; print one more star
    System.out.print("*");
    printStars(n - 1);
  }
}
```

Perkenalan Rekursi (Recursion)

Assalamu'alaikum.
Pagi semua.. Postingan kali ini akan membahas salah satu pemrograman yang dinamakan REKURSI.

Rekursi adalah konsep dalam pengeksekusian program, dimana suatu fungsi memanggil dirinya sendiri.
Normalnya, dalam suatu fungsi yang memanggil fungsi, biasanya fungsi yang dipanggil adalah fungsi lain selain dirinya.
Contohnya dalam pemrograman C (Karena bahasa C adalah bahasa pemrograman tingkat tengah yang menjadi dasar berpikir programmer), suatu fungsi main() memanggil fungsi hitung_penjumlahan(2, 3) (Selama fungsi hitung_penjumlahan() itu sendiri didefinsikan)
Tapi ada kalanya, suatu fungsi memanggil dirinya sendiri. Hal inilah yang dinamakan rekursi / rekursif.

Konsep rekursi umumnya digunakan untuk memindai / memetakan / melakukan pengecekan terhadap sesuatu.
Contoh lainnya adalah dalam pengecekan binary tree apakah pre-order atau post-order

Lalu kenapa harus menggunakan rekursi??
Sebenarnya itu pertanyaan yang salah. Pertanyaan yang benar adalah, kapan kita menggunakan rekursi??
Jadi, penggunaan rekursi biasanya diterapkan pada masalah-masalah dimana kita memerlukan data-data lama sebagai parameter, tanpa perlu disimpan dengan kompleks.
Rekursi sendiri identik dengan backtrack, dimana backtrack adalah penelusuran kembali data-data lama untuk diolah lagi.

Dalam sebuah fungsi rekursi, diharuskan minimal ada 2 case.
Yang pertama adalah base case, dimana base case ini akan mengembalikan nilai yang sifatnya pasti suatu nilai.
Contoh base case : `if(i == 0) return i;`
Yang kedua adalah recursion case, dimana case ini memanggil fungsi yang merupakan dirinya sendiri.
Contoh recursion case :

```
factorial(int a){
    if.....
    else return factorial(a-1);
}
```

atau

```
factorial(int a){
    if.....
```

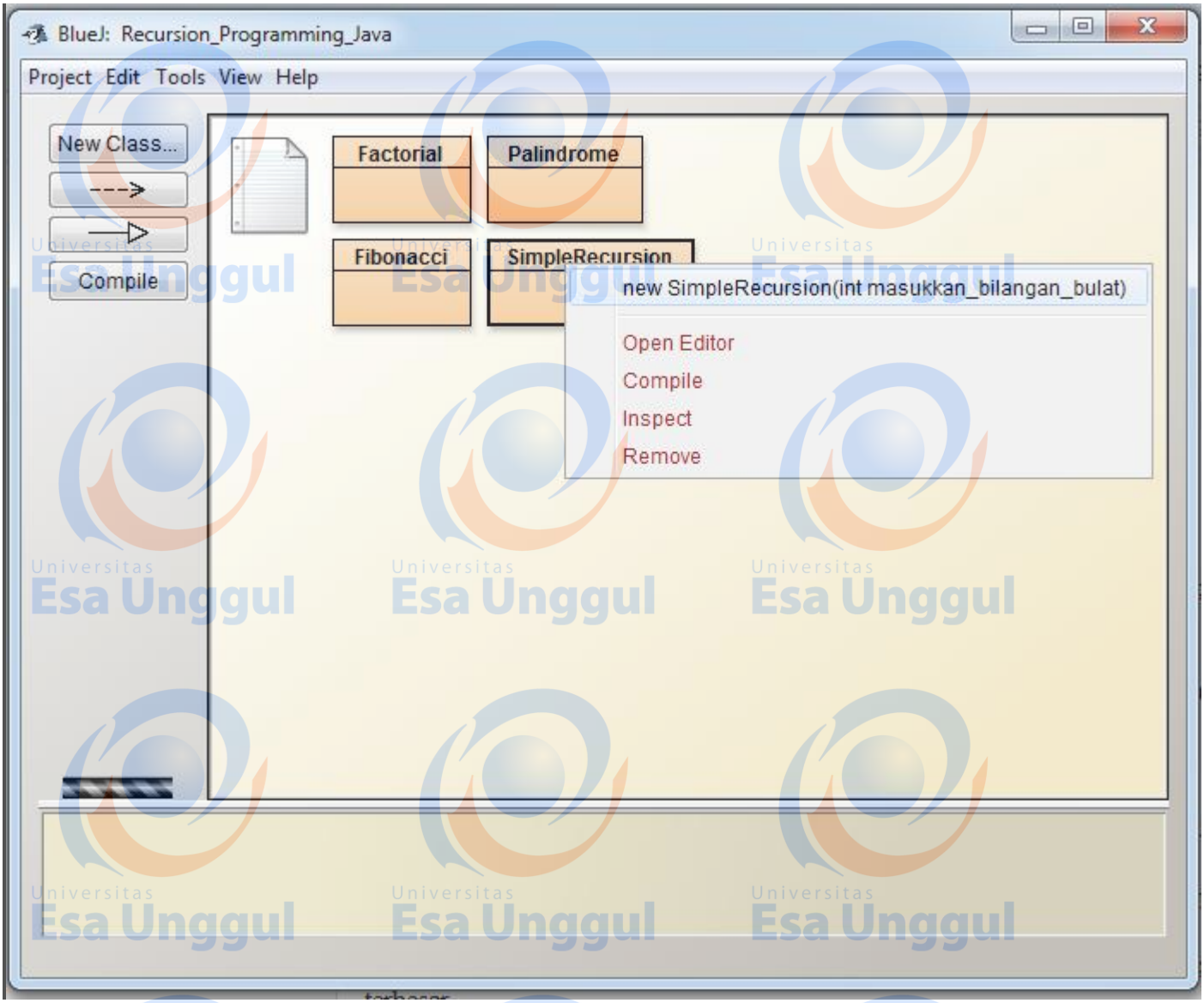
```
    return factorial(a-1);
}
```

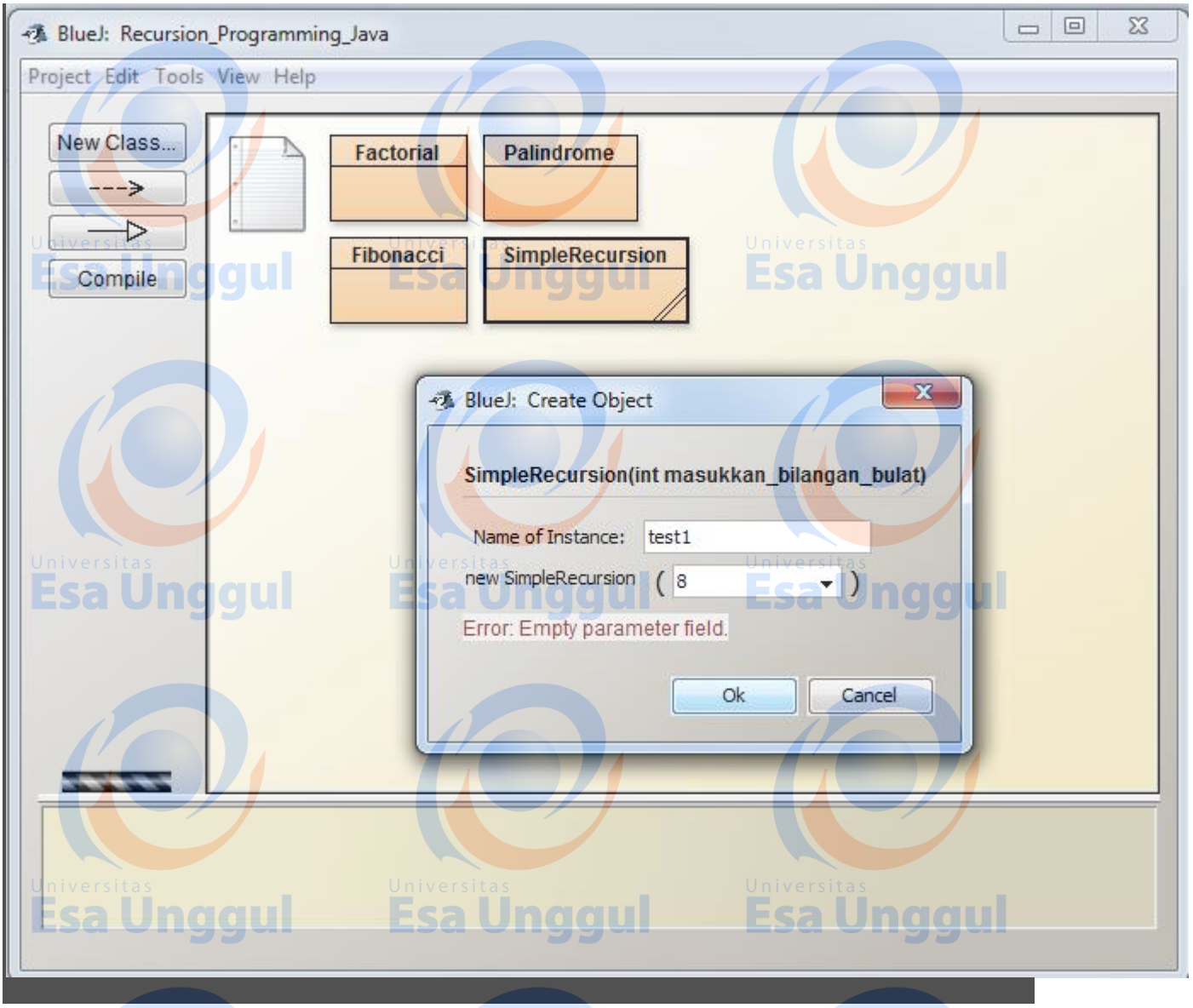
Untuk contoh penggunaan rekursi dalam pemrograman Java, bisa dilihat pada source code dibawah.

```
public class SimpleRecursion
{
    public SimpleRecursion(int masukkan_bilangan_bulat){
        int i = masukkan_bilangan_bulat;
        System.out.println("Dengan inisialisasi integer = " + i);
        System.out.print(recursion(i));
    }

    public int recursion(int a){
        if(a == 0) return a;
        if(a < 0){
            System.out.print(a + "<");
            return recursion(a+1);
        }
        System.out.print(a + ">");
        return recursion(a-1);
    }
}
```

Kemudian hasil output atau visualisasi program tersebut adalah seperti pada gambar dibawah.


BlueJ: Recursion_Programming_Java
Project Edit Tools View Help
New Class...
Compile
Factorial
Palindrome
Fibonacci
SimpleRecursion
new SimpleRecursion(int masukkan_bilangan_bulat)
Open Editor
Compile
Inspect
Remove



Modul 6 :

Searching and Sorting

Simple Search or Linear Search

The simplest way to search for an element is to iterate through the entire array and compare each value of that array with the target element. When a match is found, we may break out of the loop if we are sure that there exist no duplicate values. However, if duplicate values do exist in the array, we might have to continue searching even when a match is found. Since we search for the elements in a linear order from left to right (by convention, you may also search from the end of the array to the beginning), this algorithm is named as linear search. Following is the code snippet for linear search.

```
int[] a = { 3, 34, 5,91, 100}; // an array not containing duplicates
int target = 91; // the element to be searched
for( int i=0; i<a.length; i++) {
   if(a[i]==target) {
      System.out.println ( "Element found at index "+i);
      break; // break should be omitted if the array contains duplicates
   }
}
```

Before we move onto the next algorithm, we will look at what efficiency means in the context of algorithms. There are two things that have to be taken care of when we write algorithms. The first is the execution time and the second is the memory requirement. Execution time is determined by the numbers of statements that are excited by the algorithm while memory requirement is determined by the additional variables that we use. In the above program, we have used only one variable, target, which falls under memory requirement. However, the execution time cannot be specified directly as such even if we consider every statement to take the same time to execute. The reason is that, in this particular algorithm, if the target is in the beginning of the array, then this search would require only a single comparison which is known as the best case. However if the target is located at the end of the array, the number of comparisons required would be equal to the length of the array. This is the worst case. The average execution time would occur when the target is located in the middle of the array. So, one thing that we can conclude is that the efficiency of algorithms depends on the input data too. Here arises the need for a standardised comparison of efficiencies. And one solution is the Big O notation.

The big O notation is gives us a relation between the number of data items contained in the array and the number of comparisons required. It takes the worst case into account. It determines how hard an algorithm has to work to obtain the result. In this particular linear search algorithm, if the number of data items are n, then the number of compressions required are also n. This is the order of the algorithm. Hence, linear search is said to be an algorithm of order n.

There is a better way to arrive on this result by using the formal definition of big O. A function f(n) is said to have an order g(n) written as O(f(n))=g(n) [O represents order] if and only if, there exists an N and a c such that for every n>N, the following condition is true: 0 < f(n) < c* g(n). What this definition has basically done is to put an upper bound on the performance of the algorithm and take the worst case scenario.

Let us understand it in the context of linear search. We should first develop the function f(n). Assume that each of the statements takes the same time to execute. So the execution time is proportional to the number of statements executed which will be equal to 2*n or n or 2*n+1 depending on what you wish to regard as a statement. The important thing here is that the power of n is 1 and not 2 or 3. This basically depends on the loop which is executed n times (the size of the array). In the above definition, if the

f(n) is substituted, you would get g(n) as n by taking appropriate values of c and N. This might not be much clear for the present moment. But for now, assume that order of an algorithm is the number of times the for loop is executed for the array size, n. In this particular case, when the array contains n elements, the for loop has to execute n times (considering the worst case scenario) and hence the order of linear search is n.

Binary Search

Binary search is an efficient algorithm which can be used to search in a sorted array. Note that the array has to be sorted in either ascending or descending order for the algorithm to worth. Initially, the range of the array to be searched begins at the first element of the array and extends up to the last element. This range reduces by half in every iteration. We locate the middle element of the array and compare it with the target. If the target equals the middle element, our search is completed, otherwise the range has to be adjusted accordingly. For now, assume that the array is sorted in ascending order. If the middle element is smaller than the target, then the target cannot be found in the left half of the array as each of those elements would also be smaller than the target. Hence, we can narrow down our search to the right half of the array. On the other hand, if the middle element is larger than the target, we narrow our search to the left auld of the array. Clearly, the range of elements to be searched has been reduced by half. We now perform the same operation of finding the middle element of the new range and reduce the range accordingly. In the second iteration, the range of elements to be searched reduces to one fourth of the original array length. Similarly, with the third iteration, the range of elements reduce to one eighth. Given below is diagrammatic representation of this algorithm

Method name	**Description**
binarySearch(array, value)	**returns the index of the given value in a found)**
binarySearch(array, minIndex, maxIndex, value)	**returns index of given value in a *sorted* *Imax* - 1 (< 0 if not found)**
copyOf(array, length)	**returns a new resized copy of an array**
equals(array1, array2)	**returns true if the two arrays contain sa order**
fill(array, value)	**sets every element to the given value**
sort(array)	**arranges the elements into sorted orde**
toString(array)	**returns a string representing the array,**

Using binarySearch

```
statement1;
statement2;
statement3;
for (int i = 1; i <= N; i++) {
    statement4;
}
for (int i = 1; i <= N; i++) {
    statement5;
    statement6;
    statement7;
}
```

http://www.buildingjavaprograms.com/supplements4.shtml

Chapter : 1,8,10,13

Modul : 7 :

Stacks and queues

- **efficiency: A measure of the use of computing resources by code.**
  - **can be relative to speed (time), memory (space), etc.**
  - **most commonly refers to run time**
  - **Assume the following:**
  - **Any single Java statement takes the same amount of time to run.**
  - **A method call's runtime is measured by the total of the statements inside the method's body.**
  -

- Excersice
- A loop's runtime, if the loop repeats N times, is N times the runtime of the statements in its body.

- Modify our exam score program so that it reads the exam scores into a queue and prints the queue.
  - Next, filter out any exams where the student got a score of 100.
  - Then perform your previous code of reversing and printing the remaining students.
    - What if we want to further process the exams after printing?

Postfix algorithm :

- The algorithm: Use a stack
  - When you see an operand, push it onto the stack.
  - When you see an operator:
    - pop the last two operands off of the stack.
    - apply the operator to them.
    - push the result onto the stack.
  - When you're done, the one remaining stack element is the result.

"5 2 4 * + 7 -"

```
// Evaluates the given prefix expression and ret
// Precondition: string represents a legal postf
public static int postfixEvaluate(String express
    Stack<Integer> s = new Stack<Integer>();
    Scanner input = new Scanner(expression);
    while (input.hasNext()) {
        if (input.hasNextInt()) {     // an opera
            s.push(input.nextInt());
        } else {                      // an opera
            String operator = input.next();
            int operand2 = s.pop();
            int operand1 = s.pop();
            if (operator.equals("+")) {
                s.push(operand1 + operand2);
            } else if (operator.equals("-")) {
                s.push(operand1 - operand2);
            } else if (operator.equals("*")) {
                s.push(operand1 * operand2);
            } else {
                s.push(operand1 / operand2);
```

Modul 8

Implementing a Collection Class:
ArrayIntList

Exercise

- Write a program that reads a file (of unknown size) full of integers and prints the integers in the reverse order to how they occurred in the file. Consider example file data.txt:

  17

  932085

  -32053278

  100

  3

  - When run with this file, your program's output would be:

  3

  100

  -32053278

  932085

  17

Cara Membuat Array di Java

Cara membuat array kosong:

```
// cara pertama
String[] nama;

// cara kedua
String nama[];

// cara ketiga dengan kata kunci new
String[] nama = new String[5];
```

Parhatikan:

- Kita menggunakan kurung siku [] untuk membuat array;
- Kurung siku bisa diletakkan setelah tipe data atau nama array;

- Angka 5 dalam kurung artinya batas atau ukuran array-nya.

Array yang kosong siap diisi dengan data. Pastikan mengisinya dengan data yang sesuai dengan tipe datanya.

Kita bisa mengisinya seperti ini:

```
nama[0] = "Linda";
nama[1] = "Santi";
nama[2] = "Susan";
nama[3] = "Mila";
nama[4] = "Ayu";
```

Atau kalau tidak mau repot, kita bisa membuat array dan langsung mengisinya.

```
String[] nama = {"Linda", "Santi", "Susan", "Mila", "Ayu"};
```

Mengambil Data dari Array

Seperti yang sudah kita ketahui, array memiliki indeks untuk memudahkan kita mengakses datanya.

Karena itu, kita bisa mengambil datanya dengan cara seperti ini:

```
// membuat array
String[] nama = {"Linda", "Santi", "Susan", "Mila", "Ayu"};

// mengambil data array
System.out.println(teman[2]);
```

Kira-kira apa hasil outputnya?

Yep! benar sekali, hasil outputnya adalah:

```
Susan
```

Karena Susan terletak di indeks ke-2.

Menggunakan Perulangan

Mengambil data satu per satu dari array mungkin cukup melelahkan, karena kita harus mengtik ulang nama array-nya dengan indeks yang berbeda.

Contoh:

```
System.out.println(teman[0]);
System.out.println(teman[1]);
System.out.println(teman[2]);
System.out.println(teman[3]);
```

Bagaimana kalau data array-nya sampai 1000, maka kita harus mengetik kode sebanyak seribu kali.

Karena itu, disinilah peran perulangan.

Perhatikan:
Di sana kita menggunakan atribut `length` untuk mengambil panjang array-nya.

Jadi, perulangan akan dilakukan sebanyak isi array-nya.

Linked Lists

- **reference semantics**: Behavior where variables actually store the address of an object in memory.
  - When one reference variable is assigned to another, the object is *not* copied; both variables refer to the *same object*.

```
int[] a1 = {4, 5, 2, 12, 14, 14, 9};
int[] a2 = a1;    // refers to same array as a1
a2[0] = 7;
System.out.println(a1[0]);   // 7
```

Pengertian Linked list :

- sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, yang setiap elemennya terdiri dari dua bagian
- struktur berupa rangkaian elemen saling berkait dimana setiap elemen dihubungkan elemen lain melalui pointer. Pointer adalah alamat elemen. Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori.

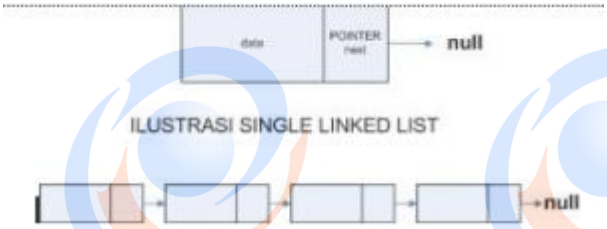
Link list adalah desain tempat penyimpanan data yang terdiri dari *node-node* (simpul-simpul) yang saling terhubung.
Link list dapat diilustrasikan seperti kereta api, dimana kereta api terdiri dari gerbong-gerbong yang saling terhubung yang dapat mengangkut penumpang. Gerbong disini setara dengan *node* dalam link list yang berfungsi untuk menyimpan data.
Jika kita menyimpan data 3, 5 dan 7 dalam array, maka ilustrasi tempat penyimpanannya sbb:
Dengan 1 nama, array bisa menyimpan data yg bertipe sama. Dimana setiap data mempunyai indeks.

Sedangkan jika data tersebut disimpan dalam link list, maka ilustrasi tempat penyimpanannya sbb:

Figure 1

Singly Linked List :
~ Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya dan juga memiliki field yang berisi data.

~ Akhir linked list ditandai dengan node terakhir akan menunjuk ke *null* yang akan digunakan sebagai kondisi berhenti saat pembacaan linked list.



Singly Linked List Non Circular

Doubly Linked List :

~ Linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu: 1 field pointer yang menunjuk ke pointer berikutnya, 1 field pointer yang menunjuk ke pointer sebelumnya dan field yang berisi data dari node tersebut.

~ Pointer next dan prev-nya menunjuk ke *null*.

Singly Circular Linked List :

~ Single Linked List yang pointer next-nya menunjuk ke dirinya sendiri, jika terdiri dari beberapa node maka pointer terakhirnya akan menunjuk ke pointer terdepannya.

Double Circular Linked List :

~ Double Linked List yang pointer next dan prev-nya menunjuk ke dirinya sendiri secara circular.

Link list tidak mempunyai indeks seperti array. Kita hanya bisa memberi nama *node*. Akan tetapi, tidak semua *node* dalam link list mempunyai nama. Sebaiknya kita memberi nama untuk *node* yang pertama (misal namanya **head**), dan *node* yang terakhir (misal namanya **tail**). Tujuannya untuk memudahkan operasi link list dari depan atau belakang, misal nambah data atau menghapus data.

Langkah yang pertama, kita harus mendefinisikan apa itu *node*. Dalam Java, sebaiknya pendefinisian *node* ini dibuat dalam sebuah *class*, misal:

```
class Node{
    public int data;        // data
    public Node next;       // penunjuk
    // --------------------------------------
    public Node(int d){     // konstruktor
        data = d;           // mengisi data Node
        next = null;
    }
}  // end class Node
```

Kemudian kita buat design link list dalam class yang lain, misal:

```
class LinkList{
    private Node head, tail;
    // --------------------------------------
    public LinkList(){   // konstruktor
        head = null;     // inisialisasi
    }
    // method lainnya di bawah
```

Operasi-operasi yang bisa dilakukan dalam link list yaitu:

1. Tambah data *(insert)*
2. Edit data *(edit)*
3. Hapus data *(delete)*
4. Pengurutan data *(sorting)*

5. Pencarian data *(searching)*

Tambah Depan

Untuk tambah data dari depan, caranya:

```
public void tambahDepan(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        baru.next = head;
        head = baru;
    }
}
```

Tambah Belakang

Untuk tambah data dari belakang, caranya:

```
public void tambahBelakang(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        tail.next = baru;
        tail = baru;
    }
}
```

Hapus Depan

Untuk menghapus data dari depan, caranya:

```
public Node hapusDepan(){
    if(head!=null){
        Node temp = head;
        head = head.next;
        temp.next = null;
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Hapus Belakang

Untuk menghapus data dari belakang, caranya:

```
public Node hapusBelakang(){
    if(head!=null){
        Node bantu, temp;
        if(head.next==null){
            temp = head;
            head = tail = null;
        }
        else{
            bantu = head;
            while(bantu.next!=tail){
                bantu = bantu.next;
            }
            temp = tail;
            tail = bantu;
            tail.next = null;
        }
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Pengertian Linked list :

- sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, yang setiap elemennya terdiri dari dua bagian
- struktur berupa rangkaian elemen saling berkait dimana setiap elemen dihubungkan elemen lain melalui pointer. Pointer adalah alamat elemen. Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori.

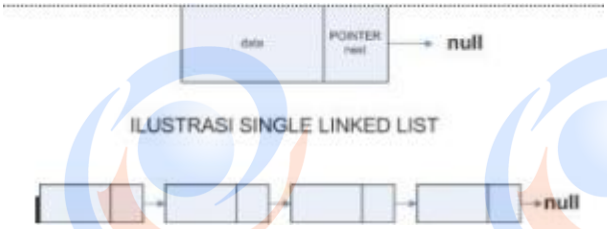
Link list adalah desain tempat penyimpanan data yang terdiri dari *node-node* (simpul-simpul) yang saling terhubung.
Link list dapat diilustrasikan seperti kereta api, dimana kereta api terdiri dari gerbong-gerbong yang saling terhubung yang dapat mengangkut penumpang. Gerbong disini setara dengan *node* dalam link list yang berfungsi untuk menyimpan data.
Jika kita menyimpan data 3, 5 dan 7 dalam array, maka ilustrasi tempat penyimpanannya sbb:
Dengan 1 nama, array bisa menyimpan data yg bertipe sama. Dimana setiap data mempunyai indeks.

Sedangkan jika data tersebut disimpan dalam link list, maka ilustrasi tempat penyimpanannya sbb:

Figure 1

Singly Linked List :
~ Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya dan juga memiliki field yang berisi data.

~ Akhir linked list ditandai dengan node terakhir akan menunjuk ke *null* yang akan digunakan sebagai kondisi berhenti saat pembacaan linked list.



Singly Linked List Non Circular

Doubly Linked List :

~ Linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu: 1 field pointer yang menunjuk ke pointer berikutnya, 1 field pointer yang menunjuk ke pointer sebelumnya dan field yang berisi data dari node tersebut.

~ Pointer next dan prev-nya menunjuk ke *null*.

Singly Circular Linked List :

~ Single Linked List yang pointer next-nya menunjuk ke dirinya sendiri, jika terdiri dari beberapa node maka pointer terakhirnya akan menunjuk ke pointer terdepannya.

Double Circular Linked List :

~ Double Linked List yang pointer next dan prev-nya menunjuk ke dirinya sendiri secara circular.

Link list tidak mempunyai indeks seperti array. Kita hanya bisa memberi nama *node*. Akan tetapi, tidak semua *node* dalam link list mempunyai nama. Sebaiknya kita memberi nama untuk *node* yang pertama (misal namanya **head**), dan *node* yang terakhir (misal namanya **tail**). Tujuannya untuk memudahkan operasi link list dari depan atau belakang, misal nambah data atau menghapus data.

Langkah yang pertama, kita harus mendefinisikan apa itu *node*. Dalam Java, sebaiknya pendefinisian *node* ini dibuat dalam sebuah *class*, misal:

```
class Node{
    public int data;        // data
    public Node next;       // penunjuk
    // --------------------------------------
    public Node(int d){     // konstruktor
        data = d;           // mengisi data Node
        next = null;
    }
} // end class Node
```

Kemudian kita buat design link list dalam class yang lain, misal:

```
class LinkList{
    private Node head, tail;
    // --------------------------------------
    public LinkList(){   // konstruktor
        head = null;     // inisialisasi
    }
    // method lainnya di bawah
```

Operasi-operasi yang bisa dilakukan dalam link list yaitu:

1. Tambah data *(insert)*
2. Edit data *(edit)*
3. Hapus data *(delete)*
4. Pengurutan data *(sorting)*

5. Pencarian data *(searching)*

Tambah Depan

Untuk tambah data dari depan, caranya:

```
public void tambahDepan(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        baru.next = head;
        head = baru;
    }
}
```

Tambah Belakang

Untuk tambah data dari belakang, caranya:

```
public void tambahBelakang(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        tail.next = baru;
        tail = baru;
    }
}
```

Hapus Depan

Untuk menghapus data dari depan, caranya:

```
public Node hapusDepan(){
    if(head!=null){
        Node temp = head;
        head = head.next;
        temp.next = null;
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Hapus Belakang

Untuk menghapus data dari belakang, caranya:

```
public Node hapusBelakang(){
    if(head!=null){
        Node bantu, temp;
        if(head.next==null){
            temp = head;
            head = tail = null;
        }
        else{
            bantu = head;
            while(bantu.next!=tail){
                bantu = bantu.next;
            }
            temp = tail;
            tail = bantu;
            tail.next = null;
        }
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Pengertian Linked list :

- sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, yang setiap elemennya terdiri dari dua bagian
- struktur berupa rangkaian elemen saling berkait dimana setiap elemen dihubungkan elemen lain melalui pointer. Pointer adalah alamat elemen. Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori.

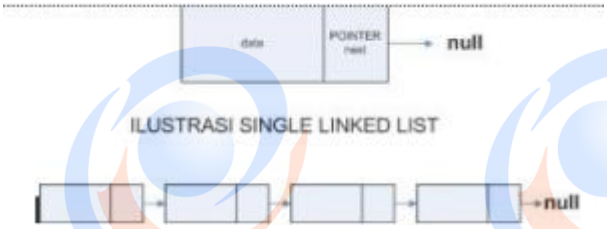
Link list adalah desain tempat penyimpanan data yang terdiri dari *node-node* (simpul-simpul) yang saling terhubung.
Link list dapat diilustrasikan seperti kereta api, dimana kereta api terdiri dari gerbong-gerbong yang saling terhubung yang dapat mengangkut penumpang. Gerbong disini setara dengan *node* dalam link list yang berfungsi untuk menyimpan data.
Jika kita menyimpan data 3, 5 dan 7 dalam array, maka ilustrasi tempat penyimpanannya sbb:
Dengan 1 nama, array bisa menyimpan data yg bertipe sama. Dimana setiap data mempunyai indeks.

Sedangkan jika data tersebut disimpan dalam link list, maka ilustrasi tempat penyimpanannya sbb:

Figure 1

Singly Linked List :
~ Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya dan juga memiliki field yang berisi data.

~ Akhir linked list ditandai dengan node terakhir akan menunjuk ke *null* yang akan digunakan sebagai kondisi berhenti saat pembacaan linked list.



Singly Linked List Non Circular

Doubly Linked List :

~ Linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu: 1 field pointer yang menunjuk ke pointer berikutnya, 1 field pointer yang menunjuk ke pointer sebelumnya dan field yang berisi data dari node tersebut.

~ Pointer next dan prev-nya menunjuk ke *null*.

Singly Circular Linked List :

~ Single Linked List yang pointer next-nya menunjuk ke dirinya sendiri, jika terdiri dari beberapa node maka pointer terakhirnya akan menunjuk ke pointer terdepannya.

Double Circular Linked List :

~ Double Linked List yang pointer next dan prev-nya menunjuk ke dirinya sendiri secara circular.

Link list tidak mempunyai indeks seperti array. Kita hanya bisa memberi nama *node*. Akan tetapi, tidak semua *node* dalam link list mempunyai nama. Sebaiknya kita memberi nama untuk *node* yang pertama (misal namanya **head**), dan *node* yang terakhir (misal namanya **tail**). Tujuannya untuk memudahkan operasi link list dari depan atau belakang, misal nambah data atau menghapus data.

Langkah yang pertama, kita harus mendefinisikan apa itu *node*. Dalam Java, sebaiknya pendefinisian *node* ini dibuat dalam sebuah *class*, misal:

```
class Node{
    public int data;        // data
    public Node next;       // penunjuk
    // ----------------------------------------
    public Node(int d){     // konstruktor
        data = d;           // mengisi data Node
        next = null;
    }
} // end class Node
```

Kemudian kita buat design link list dalam class yang lain, misal:

```
class LinkList{
    private Node head, tail;
    // ----------------------------------------
    public LinkList(){   // konstruktor
        head = null;     // inisialisasi
    }
    // method lainnya di bawah
```

Operasi-operasi yang bisa dilakukan dalam link list yaitu:

1. Tambah data *(insert)*
2. Edit data *(edit)*
3. Hapus data *(delete)*
4. Pengurutan data *(sorting)*

5. Pencarian data *(searching)*

Tambah Depan

Untuk tambah data dari depan, caranya:

```
public void tambahDepan(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        baru.next = head;
        head = baru;
    }
}
```

Tambah Belakang

Untuk tambah data dari belakang, caranya:

```
public void tambahBelakang(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        tail.next = baru;
        tail = baru;
    }
}
```

Hapus Depan

Untuk menghapus data dari depan, caranya:

```
public Node hapusDepan(){
    if(head!=null){
        Node temp = head;
        head = head.next;
        temp.next = null;
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Hapus Belakang

Untuk menghapus data dari belakang, caranya:

```
public Node hapusBelakang(){
    if(head!=null){
        Node bantu, temp;
        if(head.next==null){
            temp = head;
            head = tail = null;
        }
        else{
            bantu = head;
            while(bantu.next!=tail){
                bantu = bantu.next;
            }
            temp = tail;
            tail = bantu;
            tail.next = null;
        }
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Pengertian Linked list :

- sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, yang setiap elemennya terdiri dari dua bagian
- struktur berupa rangkaian elemen saling berkait dimana setiap elemen dihubungkan elemen lain melalui pointer. Pointer adalah alamat elemen. Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori.

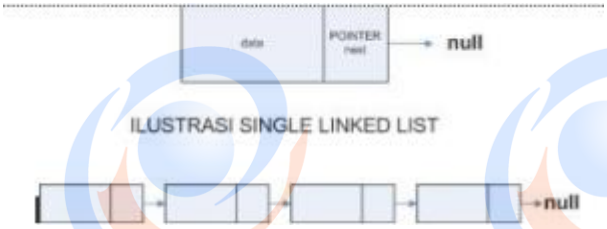
Link list adalah desain tempat penyimpanan data yang terdiri dari *node-node* (simpul-simpul) yang saling terhubung.
Link list dapat diilustrasikan seperti kereta api, dimana kereta api terdiri dari gerbong-gerbong yang saling terhubung yang dapat mengangkut penumpang. Gerbong disini setara dengan *node* dalam link list yang berfungsi untuk menyimpan data.
Jika kita menyimpan data 3, 5 dan 7 dalam array, maka ilustrasi tempat penyimpanannya sbb:
Dengan 1 nama, array bisa menyimpan data yg bertipe sama. Dimana setiap data mempunyai indeks.

Sedangkan jika data tersebut disimpan dalam link list, maka ilustrasi tempat penyimpanannya sbb:

Figure 1

Singly Linked List :
~ Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya dan juga memiliki field yang berisi data.

~ Akhir linked list ditandai dengan node terakhir akan menunjuk ke *null* yang akan digunakan sebagai kondisi berhenti saat pembacaan linked list.



Singly Linked List Non Circular

Doubly Linked List :

~ Linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu: 1 field pointer yang menunjuk ke pointer berikutnya, 1 field pointer yang menunjuk ke pointer sebelumnya dan field yang berisi data dari node tersebut.

~ Pointer next dan prev-nya menunjuk ke *null*.

Singly Circular Linked List :

~ Single Linked List yang pointer next-nya menunjuk ke dirinya sendiri, jika terdiri dari beberapa node maka pointer terakhirnya akan menunjuk ke pointer terdepannya.

Double Circular Linked List :

~ Double Linked List yang pointer next dan prev-nya menunjuk ke dirinya sendiri secara circular.

Link list tidak mempunyai indeks seperti array. Kita hanya bisa memberi nama *node*. Akan tetapi, tidak semua *node* dalam link list mempunyai nama. Sebaiknya kita memberi nama untuk *node* yang pertama (misal namanya **head**), dan *node* yang terakhir (misal namanya **tail**). Tujuannya untuk memudahkan operasi link list dari depan atau belakang, misal nambah data atau menghapus data.

Langkah yang pertama, kita harus mendefinisikan apa itu *node*. Dalam Java, sebaiknya pendefinisian *node* ini dibuat dalam sebuah *class*, misal:

```
class Node{
    public int data;        // data
    public Node next;       // penunjuk
    // ----------------------------------------
    public Node(int d){     // konstruktor
        data = d;           // mengisi data Node
        next = null;
    }
} // end class Node
```

Kemudian kita buat design link list dalam class yang lain, misal:

```
class LinkList{
    private Node head, tail;
    // ----------------------------------------
    public LinkList(){   // konstruktor
        head = null;     // inisialisasi
    }
    // method lainnya di bawah
```

Operasi-operasi yang bisa dilakukan dalam link list yaitu:

1. Tambah data *(insert)*
2. Edit data *(edit)*
3. Hapus data *(delete)*
4. Pengurutan data *(sorting)*

5. Pencarian data *(searching)*

Tambah Depan

Untuk tambah data dari depan, caranya:

```
public void tambahDepan(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        baru.next = head;
        head = baru;
    }
}
```

Tambah Belakang

Untuk tambah data dari belakang, caranya:

```
public void tambahBelakang(int d){
    Node baru = new Node(d);
    if(head==null){
        head = baru;
        tail = baru;
    }
    else{
        tail.next = baru;
        tail = baru;
    }
}
```

Hapus Depan

Untuk menghapus data dari depan, caranya:

```
public Node hapusDepan(){
    if(head!=null){
        Node temp = head;
        head = head.next;
        temp.next = null;
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Hapus Belakang

Untuk menghapus data dari belakang, caranya:

```
public Node hapusBelakang(){
    if(head!=null){
        Node bantu, temp;
        if(head.next==null){
            temp = head;
            head = tail = null;
        }
        else{
            bantu = head;
            while(bantu.next!=tail){
                bantu = bantu.next;
            }
            temp = tail;
            tail = bantu;
            tail.next = null;
        }
        return temp;
    }
    else{
        System.out.println("List kosong.");
        return null;
    }
}
```

Pengertian Linked list :

- sekumpulan elemen bertipe sama, yang mempunyai keterurutan tertentu, yang setiap elemennya terdiri dari dua bagian
- struktur berupa rangkaian elemen saling berkait dimana setiap elemen dihubungkan elemen lain melalui pointer. Pointer adalah alamat elemen. Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walau tidak bersebelahan secara fisik di memori.

Link list adalah desain tempat penyimpanan data yang terdiri dari *node-node* (simpul-simpul) yang saling terhubung.
Link list dapat diilustrasikan seperti kereta api, dimana kereta api terdiri dari gerbong-gerbong yang saling terhubung yang dapat mengangkut penumpang. Gerbong disini setara dengan *node* dalam link list yang berfungsi untuk menyimpan data.
Jika kita menyimpan data 3, 5 dan 7 dalam array, maka ilustrasi tempat penyimpanannya sbb:
Dengan 1 nama, array bisa menyimpan data yg bertipe sama. Dimana setiap data mempunyai indeks.

Sedangkan jika data tersebut disimpan dalam link list, maka ilustrasi tempat penyimpanannya sbb:

Figure 1

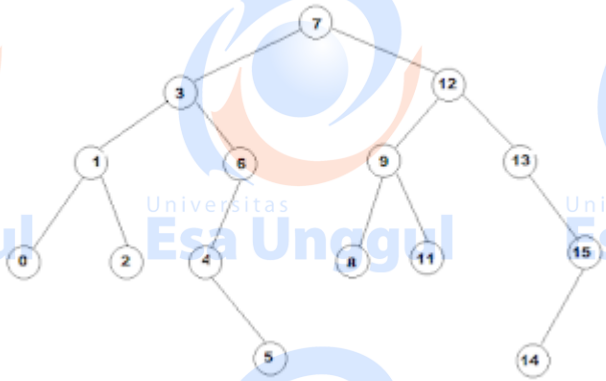
MODUL : 10

Binary Trees

Contoh Coding Program Binary Tree pada Java

Struktur Binary Tree adalah sebuah prohon struktur data dimana setiap simpul memiliki paling banyak dua anak. Secara khusus anak (child) nya dinamakan kiri dan kanan. Anak (child) kiri selalu bernilai lebih kecil daripada nilai induk (parent) nya, sedangkan anak (child) kanan selalu bernilai lebih besar daripada induk (parent) nya.

Berikut ilustrasi dari Struktur Binary Tree :



Berikut adalah coding dari program binary tree :

```
package binary_tree;
class Node{
    Node left, right, parent;
    int data;
}
public class Binary_Tree {
// FUNCTIONS & GLOBAL VARIABLES
    static Node root;

    static void insert(int data){
```

```
        // buat node baru
        Node new_node = new Node();
        new_node.data = data;
        if(root == null){ // tree kosong
            root = new_node;
        }else{
            Node position = root;
            boolean found = false;
            while(!found){
                if(new_node.data < position.data){
                    if(position.left == null){
                        found = true;
                        new_node.parent = position;
                        position.left = new_node;
                    }else{
                        position = position.left;
                    }
                }else{
                    if(position.right == null){
                        found = true;
                        new_node.parent = position;
                        position.right = new_node;
                    }else{
                        position = position.right;
                    }
                }
            }
        }
    }
```

```
    static void view(){
        node_view(root, "");
    }
    static void node_view(Node node, String spaces){
        if(node == null){
            System.out.println(spaces + "EMPTY");
        }else{
            System.out.println(spaces + node.data);
            node_view(node.left, spaces+"    ");
            node_view(node.right, spaces + "    ");
        }
    }

    static void postfix (Node localroot){
        if(localroot != null){
            System.out.print("(");
            postfix(localroot.left);
            postfix(localroot.right);
            System.out.print(" "+localroot.data+" ");
            System.out.print(")");
        }
    }

    static void delete(int deleted){
        boolean found = false;
        Node x = root;
        while(x != null){
            if(x.data == deleted){
```

```
                found = true;
                break;
            }else if(x.left != null  && deleted < x.data){
                x = x.left;
            }else if(x.right != null && deleted > x.data){
                x = x.right;
            }else{
                found = false;
                break;
            }
        }
        if(!found){
            // do nothing, node not found
        }else{
            boolean is_root = x.parent == null;
            boolean is_right = !is_root && x ==
x.parent.right;
            boolean is_left = !is_root && x == x.parent.left;
            // jika tidak punya anak
            if(x.left == null && x.right == null){
                if(is_root){ //  tdk punya anak & adalah root
                    root = null;
                }else{ // tdk punya anak & bukan root
                    if(is_left){ // tdk punya anak & adalah
anak kiri
                        x.parent.left = null;
                    }else if(is_right){ // tdk punya anak &
adalah anak kanan
                        x.parent.right = null;
```

```
                }
                x.parent = null; // putuskan hubungan
dengan parent
            }
        }else if(x.left != null && x.right == null){ //
hanya punya anak kiri
            if(is_root){
                root = x.left;
                root.parent.left = null;
                root.parent = null;
            }else{
                if(is_left){
                    x.parent.left = x.left;
                }else if(is_right){
                    x.parent.right = x.left;
                }
                x.left.parent = x.parent;
                x.parent = null;
                x.left = null;
            }
        }else if(x.left == null && x.right != null){ //
hanya punya anak kanan
            if(is_root){ // root
                root = x.right;
                root.parent.right = null;
                root.parent = null;
            }else{ // bukan root
                if(is_left){
                    x.parent.left = x.right;
```

```
                }else if(is_right){
                    x.parent.right = x.right;
                }
                x.right.parent = x.parent;
                x.parent = null;
                x.right = null;
            }
        }else{ // punya 2 anak
            Node replacement = x.right; // kanan sekali
            while(replacement.left != null){ // kiri
sekiri-kirinya
                replacement = replacement.left;
            }
            if(replacement.right != null){ // kalau
replacement punya anak kanan
                replacement.parent.left =
replacement.right;
                replacement.right.parent =
replacement.parent;
            }else{ // kalau replacement tidak punya anak
                replacement.parent.left = null;
            }
            // replace x
            if(is_root){
                replacement.parent = null;
                root = replacement;
            }else if(is_left){
                replacement.parent = x.parent;
                x.parent.left = replacement;
```

```
            }else if(is_right){
                replacement.parent = x.parent;
                x.parent.right = replacement;
            }
            replacement.left = x.left;
            replacement.right = x.right;
            if(replacement.left != null){
                replacement.left.parent = replacement;
            }
            if(replacement.right != null){
                replacement.right.parent = replacement;
            }
            // hapus x dari tree
            x.parent = null;
            x.left = null;
            x.right = null;
        }
    }
}

public static void main(String[] args) {
    insert(5);
    insert(7);
    insert(6);
    insert(8);
    insert(2);
    insert(4);
    insert(1);
    view();
```

```
        delete(5);
        view();
        postfix(root);
    }
}
```

Ya itulah coding, script dari contoh program binary tree pada java

MODUL 11

Advanced Data Structures:
Hashing and Heaps

- **abstract data type (ADT)**: A specification of a collection of data and the operations that can be performed on it.
  - Describes *what* a collection does, not *how* it does it.
  - Java's collection framework describes ADTs with interfaces:
  - Collection, Deque, List, Map, Queue, Set, SortedMap
  - An ADT can be implemented in multiple ways by classes:
  - ArrayList and LinkedList implement List
  - HashSet and TreeSet implement Set
  - LinkedList , ArrayDeque, etc. implement Queue
- **abstract data type (ADT)**: A specification of a collection of data and the operations that can be performed on it.
  - Describes *what* a collection does, not *how* it does it.
  - Java's collection framework describes ADTs with interfaces:
  - Collection, Deque, List, Map, Queue, Set, SortedMap
  - An ADT can be implemented in multiple ways by classes:
  - ArrayList and LinkedList implement List

- HashSet and TreeSet implement Set
- LinkedList , ArrayDeque, etc. implement Queue

HASHING (Java Programming)

Posted on 14 December 2012 by rasyid13

Hash tables salah satu fasilitas dari method yang sudah disediakan oleh java untuk memudahkan pelabelan suatu array.

Analogi yang tepat dari hash tables sendiri seperti pembacaan kamus.

Ketika memanggil kata tertentu akan keluar hasil balik dari kata kunci tersebut.

Contoh : kamus inggris Indonesia berikut . kita butuh memasukkan key nya untuk memanggil kata yang dimaksud.
Book maka secara otomatis key tersebut akan memanggil nilainya yakni buku.

Jadi seperti halnya array, hash tables hanya mempermudah penamaan array tertentu yang ingin dipanggil dengan kata kunci (key) tertentu.

Pada kata tertentu memiliki hashcode, atau nilai tertentu pada setiap String yang dimiliki nya. Disini tidak dijelaskan bagaimana computer menerjemahkan String kedalam Hash codes tersebut. Namun yang akan dipelajari ialah bagaimana mengkonversi key dalam String tertentu menjadi HashTables. Hash Tables adalah table dari key (kata kunci) yang sudah dibuat dengan index tertentu. Index tertentu tersebut terbentuk dari key. Nah untuk merubah String tersebut kedalam index agar dapat dimengerti didalam hashtables, ialah dengan langkah-langkah sebagai berikut.

1. Awalnya kita menterjemahkan String tersebut kedalam bentuk nilai yakni Hash codesnya. Dengan method >>>> object.hashCode()

   Maka kita akan mendapatkan hashcodes dari String tersebut,

   Contoh : String s = "rasyid";

   Printf(s.hashCode());

   Output : -938116944

1. Setelah itu diperlukan cara untuk mengkonversinya, yakni dengan melakukan modulus. Kenapa harus dengan modulus. Karena dengan modulus terhadap nilai kapasitas (kapasitas dari table yang akan dibuat), maka nilai yang akan dihasilkan tentunya ialah nilai antara 2 hingga N-1 dari table tersebut. Contoh array dengan kapasitas 11. Tentu nya memiliki index antara 0 hingga 10. Maka dari itu, nilai dari suatu bilangan jika dimodulus dengan kapasitas tersebut akan menghasilkan nilai modulus antara 0 hingga 10. Contoh 20 % 11, maka keluarannya ialah 9.

2. Nilai hashcodes pada suatu key (string) jelas berbeda-beda, sehingga mungkin saja nilai hashcodes nya bernilai negative. Suatu nilai negative jika dimodulus akan menghasilkan nilai negative. Maka dari itu kita juga butuh mengkonversi bilangan hashcodes nya menjadi bilangan bulat positif.
Misalkan "jack" menghasilkan angka biner 101111 (*ket : nilai 1 pada kiri merupakan penanda bahwa angka tersebut negative, jika 0 berarti postif).

Cara yang tepat ialah melakukan oprasi Boolean dan dengan angka biner 011111…

Nah angka 01111… (takhingga), jika dikonversi angka menghasilkan bilangan bulat yang begitu panjang, namun dalam bentuk hexadesimalnya ialah 0x7FFFFFFF.

Maka : 101111 & 0x7FFFFFFF maka hasilnya ialah 001111.

1. Nah setelah itu kita sudah mendapatkan nilai dari indexnya.
Contoh :

Misalkan kita ingin membuat hash table dengan jumlah 11

```
int kapasitas = 11;

String s = "rasyid";

int j = (s.hashCode() & 0x7FFFFFFF )% kapasitas ;

Printf(j);
```

Output nya : 77

Contoh Javanya :

```
import java.util.HashMap;
import java.util.Map;
public class TestHashTable {
private static final int MASK = 0x7FFFFFFF;
   private static final int CAPACITY = 101;
   public static void main(String[] args) {
      Map map1 = new HashMap();
      map1.put("Tor", "gate");
      map1.put("Rad", "wheel");
      map1.put("Tag", "day");
      map1.put("Uhr", "clock");
      map1.put("Hut", "hat");
      map1.put("Ohr", "ear");
      System.out.println("map1=" + map1);
      printHashCode("rasyid");
      printHashCode("Pin");
```

```
        printHash("rasyid");
        printHash("Hut");
    }
    private static void printHash(String word) {
        System.out.println("hash(" + word + ") = " + hash(word));
    }
    private static int hash(Object object) {
        return (object.hashCode() & MASK) % CAPACITY;
    }
    private static void printHashCode(String word) {
        System.out.printf("%s: %s%n", word, word.hashCode());
    }
}
```

Ketika key tersebut digunakan pada index tertentu, pada kasus tertentu hash yang kita buat memiliki nilai yang sudah dimiliki hash lain. Sehingga pada hashtable tersebut program kesulitan untuk mencerna mana key yang memiliki index tersebut, misalkan kata "tug" dan kata "raw" memiliki nilai index yang sama. Maka kita perlu solusi untuk menyelsaikan ini.

Solusi :

1. Linear probes.

Linear quadratic probes ialah solusi dengan cara mencari kan nilai index yang kosong metode linear. Metode linear yang dimaksud ialah mencari satu-satu index yang belum terisi dari index ke 0. Probes ialah banyak perulangan yang terjadi selama mencari index yang kosong tersebut. Sama halnya dengan metode pencarian Sequential Search. Metode ini juga memiliki notasi big O sebesar *O(n)*. Kelebihannya metode ini selalu menemukan index yang kosong atau belum terisi. Kekurangannya metode ini jauh lebih lambat.

1. Quadratic Probes.

Linear quadratic probes ialah solusi dengan cara mencari kan nilai index yang kosong metode kuadrat. Metode kuadrat yang dimaksud ialah mencari index yang belum terisi dari index ke 0, dengan menambahkan nilai index yang sama tersebut dengan angka kelipatan kuadrat dimulai dari 1. Metode ini juga memiliki notasi big O sebesar *O(log n)*. Kelebihannya metode ini jauh lebih cepat. Kekurangannya metode ini mungkin saja tidak menemukan nilai dari index selanjutnya, sehingga perulangannya bisa mencapai tak hingga.

Berikut contohnya :

```
public class TestHash {

    private static final int MASK = 0x7FFFFFFF; // 2^32-1

    private static final int CAPACITY = 11;

    private static int size = 0;

    private static boolean[] used = new boolean[CAPACITY];
```

```
private static final int MASK2 = 0x7FFFFFFF; // 2^32-1

private static final int CAPACITY2 = 11;

private static int size2 = 0;

private static boolean[] used2 = new boolean[CAPACITY2];
public static void main(String[] args) {

    System.out.println("LINEAR");

    printHash1("Rad");

    printHash1("Uhr");

    printHash1("Ohr");

    printHash1("Tor");

    printHash1("Hut");

    printHash1("Tag");

    printHash1("Eis");
    printHash1("Ast");

    printHash1("Zug");

    printHash1("Hof");

    printHash1("Mal");
      System.out.println("QUADRATIC");

      printHash2("Rad");

    printHash2("Uhr");

    printHash2("Ohr");

    printHash2("Tor");
```

```
    printHash2("Hut");

    printHash2("Tag");

    printHash2("Eis");

    printHash2("Ast");

    printHash2("Zug");

    printHash2("Hof");

    printHash2("Mal");

}

private static void printHash1(String word) {

    System.out.printf("hash(%s) = %d, load = %d%%%n",

            word, hash(word), 100 * size / CAPACITY);

}

private static int hash(Object object) {

    ++size;

    int h = (object.hashCode() & MASK) % CAPACITY;

    while (used[h]) {

        System.out.printf("%d, ", h);

        h = (h + 1) % CAPACITY;

    }

    used[h] = true;

    return h;

}
```

```
    private static void printHash2(String word) {
        System.out.printf("hash(%s) = %d, load = %d%%%n",
            word, hash2(word), 100 * size2 / CAPACITY2);
    }
    private static int hash2(Object object) {
        ++size2;
        int h = (object.hashCode() & MASK2) % CAPACITY2;
        if (used2[h]) {
            int h0 = h;
            int jump = 1;
            while (used2[h]) {
                System.out.printf("%d, ", h);
                h = (h0 + jump * jump) % CAPACITY2; // squared increment
                ++jump;
            }
        }
        used2[h] = true;
        return h;
    }
}
```

Output :

```
run:
```

LINEAR

hash(Rad) = 3, load = 9%

hash(Uhr) = 4, load = 18%

hash(Ohr) = 2, load = 27%

hash(Tor) = 8, load = 36%

hash(Hut) = 5, load = 45%

3, 4, 5, hash(Tag) = 6, load = 54%

5, 6, hash(Eis) = 7, load = 63%

3, 4, 5, 6, 7, 8, hash(Ast) = 9, load = 72%

9, hash(Zug) = 10, load = 81%

3, 4, 5, 6, 7, 8, 9, 10, hash(Hof) = 0, load = 90%

2, 3, 4, 5, 6, 7, 8, 9, 10, 0, hash(Mal) = 1, load = 100%

QUADRATIC

hash(Rad) = 3, load = 9%

hash(Uhr) = 4, load = 18%

hash(Ohr) = 2, load = 27%

hash(Tor) = 8, load = 36%

hash(Hut) = 5, load = 45%

3, 4, hash(Tag) = 7, load = 54%

5, hash(Eis) = 6, load = 63%

3, 4, 7, hash(Ast) = 1, load = 72%

hash(Zug) = 9, load = 81%

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -8

3, 4, 7, 1, 8, 6, 6, 8, 1, 7, 4, 3, 4, 7, 1, 8, 6, 6, 8, 1, 7, 4, 3, 4, 7, 1, 7, 4, 3, 4, 7, 1, 8, 6, 6, 8, 1, ...

at TestHash.hash2(TestHash.java:73)

at TestHash.printHash2(TestHash.java:63)

at TestHash.main(TestHash.java:42)

Java Result: 1

BUILD SUCCESSFUL (total time: 0 seconds)

Ketarangan : Output asli dari kuadratic tersebut ialah tak dingga pada saat meng-input Hof, maka keluaranya bisa menjadi baris angka sepanjang 25 hal word.. hhehehe..

Referensi :

1.http://www.buildingjavaprograms.com/supplements4.shtml

2.Buku Pemrograman Java Andi Yogyakarta (2015)