

Struktur Data dan Algoritma

Review Java & OOP

Suryana Setiawan, Ruli Manurung & Ade Azurat
(acknowledgments: Denny)

Fasilkom UI



Tujuan kuliah

- Mengulang dan mengingatkan kembali beberapa konsep penting dalam *Object Oriented Programming* dan *Java*.



Outline

- Primitive Type
- Operator & Expression
- Flow Control (Decision & Iteration)
- References
- Exception Handling
- OOP



Java: *strongly* typed language

- Variable adalah sebuah tempat atau lokasi dalam *memory* untuk menyimpan sebuah nilai.
 - **variable**
 - **adj.**
 - **1** *Math.* having no fixed value
 - **n.**
 - **2** *Math., Physics* a) a part of a mathematical expression that may assume any value in a specific, related set of values b) a symbol for such a part: opposed to constant
- dalam Java, setiap variable harus diasosiasikan dengan sebuah tipe yang telah ditentukan. (*strongly typed languages*)
- dengan tipe, compiler dapat membatasi nilai yang dapat disimpan dalam variable tersebut. Contoh:
 - `int grade;`
 - `grade = 85; // ok`
 - `grade = 'A'; // not ok, compiler error`



Tipe data dalam Java

- **Primitive** types: tipe boolean, char, numerik
- **Reference** types: class (termasuk String, Integer), array.
- Apa perbedaan antara primitive and reference type?

tipe	isi	nilai default	besar
boolean	true, false	false	1 bit
char	unicode char	\u0000	16 bit
byte	signed integer	0	8 bit
short	signed integer	0	16 bit
int	signed integer	0	32 bit
long	signed integer	0	64 bit
float	IEEE 754/float	0.0	32 bit



Operator

- Apakah *operator*?
- Berdasarkan jumlah dari *operands*:
 - Unary
 - Binary
- Berdasarkan operasi yang dilakukannya:
 - Arithmetic
 - Logical



Operator & expression

- Apakah itu *expression*?
- Apakah output dari potongan program berikut?

```
int ii = 1;
int jj = 2;
double a = (ii + 1) / (jj + 1);
double b = (double) (ii + 1) / (jj + 1);
double c = (ii + 1) / (double) (jj + 1);
double d = (ii + 1.0) / (jj + 1.0);
System.out.println (a + " " + b);
System.out.println (c + " " + d);
```



Operator & expression

- Apakah output dari potongan program berikut?

```
int width = 10;  
int a = 3 / 4 * width;  
System.out.println (a);  
int b = width * 3 / 4;  
System.out.println (b);  
int c = width / 4 * 3;  
System.out.println (c);
```

Perlu diingat:

- Tipe data dari sebuah ekspresi bergantung pada operator dan tipe data dari *operands*.
- Urutan evaluasi sebuah ekspresi bergantung pada tingkat *precedence* operator. Jika dua buah operators memiliki tingkat precedence yang sama, lihat pada *associativity* operatornya:
 - LR: Left to Right
 - RL: Right to Left



Flow control

- Branch (cabang)
 - Digunakan untuk memilih perintah mana yang akan dijalankan berdasarkan kondisi tertentu.
- Loop (pengulangan)
 - Suatu kelompok program dijalankan secara berulang. Kondisi berhenti menyatakan berapa kali pengulangan dijalankan.



Flow control: *if*

```
if (condition)  
{  
    statement;  
}  
next statement
```

```
if (condition)  
{  
    statement1  
}  
else  
{  
    statement2  
}  
next statement
```

■ Kesalahan umum:

```
if (x == 0) ;  
    System.out.println ("x is zero");
```

```
if (x > 0)  
    System.out.println ("x = ");  
    Svstem.out.println (x);
```



Flow control: **switch**

- Perintah **if-else-if** yang membandingkan nilai sebuah konstant atau bilangan bulat dapat digantikan dengan perintah **switch**.

```
int digit;  
String digitName;  
switch (digit) {  
    case 0: digitName = "zero"; break;  
    case 1: digitName = "one"; break;  
    case 2: digitName = "two"; break;  
    case 3: digitName = "three"; break;  
    case 4: digitName = "four"; break;  
    case 5: digitName = "five"; break;  
    case 6: digitName = "six"; break;  
    case 7: digitName = "seven"; break;  
    case 8: digitName = "eight"; break;  
    case 9: digitName = "nine"; break;  
    default: digitName = ""; break;  
}
```



Flow control: ?

testExpr ? yesExpr : noExpr

- Berguna untuk menyederhanakan perintah *if-then-else* sederhana

- Contoh:

```
if (x > 100) {  
    y = x * 0.9;  
} else {  
    y = x;  
}
```

- Setara dengan:

```
y = (x > 100) ? (x * 0.9) : x;
```



Flow Control: **while**

```
while (condition){  
    statement;  
}  
/* Post condition: ~condition */
```

- Selama nilai ekspresi **condition** true, maka **statement** akan dijalankan, kemudian ekspresi **condition** akan dievaluasi lagi.
- Saat nilai expression **condition** false, pengulangan berhenti. Perintah **statement** tidak lagi dijalankan.
- Ingat: **condition** dievaluasi lebih dulu!



Flow Control: **do-while**

```
do {  
    statement;  
    ...  
} while (condition);  
/* Post condition: ~condition */
```

- **statement** dijalankan lebih dulu, kemudian **condition** dievaluasi.
 - Jika kondisi **true** $\Rightarrow$ **statement** dijalankan lagi.
 - Jika kondisi **false** $\Rightarrow$ loop berhenti.
- **Minimal statement** dijalankan sekali.



Flow control: **for**

```
for (initialization; condition; update) {  
    statement;  
}
```

- setara dengan perintah while berikut:

```
initialization;  
while (condition) {  
    statement;  
    update;  
}
```

- **for** digunakan bila kita sudah tahu berapa banyak pengulangan yang akan dilakukan.



Flow Control: **break**

- Kegunaan: keluar dari loop terdalam
- Bukan kebiasaan programming yang baik (*poor design*)!
- Contoh:

```
while (...) {  
    ...  
    if (something) {  
        break;  
    }  
    ...  
}
```



Flow Control: *continue*

- Kegunaan: menjalankan pengulangan selanjutnya tanpa menjalankan *sis*a dalam blok pengulangan.
- Juga bukan kebiasaan programming yang baik!
- Contoh:

```
for (int ii = 1; ii <= 100; ii++) {  
    if (ii % 10 == 0) {  
        continue;  
    }  
    System.out.println (i) ;  
}
```

- Bisakah anda tulis ulang tanpa *continue*?



Flow Control: latihan

- Tulislah sebuah program yang mencetak semua pasangan *positive integer* (**a**, **b**) di mana:
 - **a < b < 1000**, dan
 - **$(a^2 + b^2 + 1) / (ab)$** adalah integer.



Object oriented programming

■ Class

- Kombinasi dari data dan operasi (metode) dengan *access modifier*.
- Satu class membentuk sebuah tipe data

■ Object

- *instance* dari *class*
- tiap object memiliki:
 - *state* $\Rightarrow$ variable instances / fields
 - *behaviour* $\Rightarrow$ methods



Class

- Class memiliki:
 - Detail implementasi (***blue print***) dari sebuah *object yang akan dibuat* $\Rightarrow$ bayangkan sebuah *object factory*.
 - Class juga menyimpan ***static methods*** dan ***static variables*** dari object yang akan dibuat.
- Class menyatakan detail implementasi *object*
 - Seluruh *method* dan *fields*
 - code untuk method yang akan menentukan perilaku dari object.
- Berdasarkan definisi class $\Rightarrow$ object dibuat.



Object

- Membuat object dengan operator **new**
 - Mengalokasikan ruang memory untuk object baru, dan mengembalikan ***alamat*** dari object dalam memory.
 - Panggil ***constructor*** yang sesuai.
Contoh: **new Integer (20) ;**
- Mendeklarasikan ***reference variable***
 - **ClassName varName ;**
Contoh:
 - **Integer x ;**
 - **int ii ;**
 - **Ingat:** mendeklarasikan sebuah *reference variable* belum membuat object!



Object

- Menyimpan alamat dari object agar dapat digunakan lagi: gunakan operator `=`
 - Contoh:
`Integer x = new Integer (20) ;`
 - **Ingat:** operator `=` tidak meng-**copy** (*clone*) object
- Mengakses field dan memanggil method: gunakan operator `.` (*dot*)
 - Contoh:

```
Dimension size;  
size = getSize();  
System.out.println(size.width);  
System.out.println(size.getWidth());
```



Field

- Menyimpan status dari object atau class
- Terdapat dua jenis *field*:
 - ***Instance variable***
 - Tiap object punya nilai variable masing-masing $\Rightarrow$ **object** variable
 - ***Static variable***
 - Satu *class* hanya punya satu nilai variable, walaupun *class* tersebut memiliki beberapa *object* $\Rightarrow$ **class** variable



Method

- Definisi *method* terdiri dari:
 - Header
 - Access specifier (mis: `public`)
 - Return type (mis : `double`, `String`)
 - Nama method (mis: `deposit`, `openFile`)
 - Kumpulan parameter
 - Body (implementasi dari *method*)
- Method yang tidak memiliki *body* (implementasi) disebut ***abstract method***.
- Mengapa ada method yang tidak memiliki body (implementasi)?



Method

- Sebagaimana *field*, ada dua jenis *method*:
 - ***Instance method*** (non-static method)
 - method dapat dijalankan dari sebuah *object* ⇒ **object method**
 - ada parameter implisit **this** ⇒ method dapat mengakses dirinya sendiri.
 - ***Static method***
 - method dapat dijalankan tanpa object ⇒ **class method**
 - tidak ada parameter **this** ⇒ ***static method*** tidak dapat mengakses *instance variable*
- Parameter Passing bersifat ***call by value***: pemanggil memberikan ***copy*** nilai kepada method yang dipanggil.



Reference Type

- Sebuah *reference variable* dalam Java menyimpan lokasi (alamat) dari objects dalam memory komputer.
- Sebuah *primitive variable* menyimpan nilai sebenarnya dari sebuah *primitive data type*.
- Dengan kata lain, *reference variable tidak menyimpan object secara konkret (sebenarnya) tapi primitive variable menyimpan 'object'*.



Reference Type (2)

■ Operator `==`

- Untuk *primitive data type*, operator `==` digunakan untuk menguji apakah kedua variable memiliki **nilai yang sama**.
- Untuk *reference data type*, operator `==` digunakan untuk menguji apakah dua reference variables menunjuk pada object yang sama, **TIDAK** menunjukkan apakah dua objects berbeda memiliki nilai yang sama.

■ Method `equals`

- Metode `equals` dapat digunakan untuk menguji apakah kedua object berbeda memiliki nilai yang sama.
- Metode `equals` pada beberapa object seperti `Button`, akan mengembalikan `true` bila keduanya menunjukkan pada object yang sama.



Array

- Deklarasi Array (array adalah object juga)

```
int [] array1;
```

- ingat: mendeklarasikan sebuah reference variable tidak berarti mengalokasikan memory dan membuat object.

- Membuat array

```
array1 = new int[100];
```

```
int[] array2 = { 2, 3, 7, 5 };
```

- Untuk mengakses element dalam array, gunakan index

```
array1[0] = 20;
```

- Karena array adalah object juga, operator = tidak mengcopy(clone) array.

```
int[] lhs = new int[100];
```

```
int[] rhs = new int[100];
```

```
...
```

```
lhs = rhs;
```



Dynamic Array Expansion

- Misalkan, kita hendak membaca sederetan bilangan bulat dari sebuah file dan menyimpannya dalam array untuk kemudian diproses.
- Untuk membuat array, kita harus menyatakan ukuran dari array.
- Jika kita tidak tahu berapa banyak bilangan bulat dalam file, maka kita tidak dapat menentukan ukuran dari array.



Dynamic Array Expansion (2)

■ contoh:

```
int[] a = new int[10];
```

- Jika array tidak cukup, kita dapat melakukan pengembangan array dengan contoh berikut:

```
int[] original = a;  
a = new int[original.length * 2];  
for (int ii = 0; ii < original.length; ii++) {  
    a[ii] = original[ii];  
}
```

Apakah hal ini efisien?



Multidimensional Array

- Array dapat diakses dengan menggunakan lebih dari satu index.

```
int[][] x = new int[2][5];
```

- latihan:

- Buat prosedur pembuatan matriks (diberikan ukurannya) yang akan mengembalikan *upper triangle matrix* (2 dimensi). Tipe data dari tiap elemen adalah **double**.
- Contoh: dengan parameter **size** = 5, maka method tersebut mengembalikan output berikut (dalam format desimal):

```
1/1  1/2  1/3  1/4  1/5  ...
2/1  2/2  2/3  2/4
3/1  3/2  3/3
4/1  4/2
5/1
```



Contoh solusi: Cantor

```
public static double[][] cantor (int size)
{
    double[][] result = new double[size][];
    for (int ii = 0; ii < size; ii++) {
        result[ii] = new double[size - ii];
        for (int jj = 0; jj < result[ii].length; jj++) {
            result[ii][jj] = (ii + 1.0) / (jj + 1.0);
            // not the type of the operands
            // (ii + 1) / (jj + 1) will produce an integer
        }
    }

    return result;
}
```



Exceptions

- *Exception* menyatakan bahwa sebuah “kejadian aneh” telah terjadi.
- Sebuah object **Exception** menyimpan informasi tentang keanehan tersebut.
 - **throw** $\Rightarrow$ “menandakan” keanehan
 - **catch** $\Rightarrow$ tindakan “mengatasi” keanehan
- Error vs. Exception
 - Error $\Rightarrow$ kesalahan fatal, mis: out of memory
 - Exception $\Rightarrow$ masih bisa “dibetulkan”, mis: **EOFException**, **ArrayIndexOutOfBoundsException**.



Exception throwing (*≈ tidak bertanggungjawab!*)

- Klausula **throws** menyatakan bahwa sebuah method tidak menangani *checked exception*.
- Exception yang tidak ditangani akan *di-propagate* ke *caller* method.
- Exception yang tidak ditangani sampai di method **main(String[] args)** menyebabkan JVM mencetak error message, stack trace, lalu berhenti.



Exception handling (*≈bertanggungjawab!*)

- Gunakan statement **try/catch/finally**
 - **try** menandakan kumpulan statement (**try** block) yang ***mungkin*** menyebabkan exception.
 - **try** block diikuti 0 atau lebih **catch** block yang menangani sebuah tipe exception.
 - Sesudah **catch** block, boleh ada **finally** block yang biasanya berisi statement “beres-beres”. Statement ini dieksekusi apapun yang terjadi di try block.
 - Jika flow control keluar dari **try** block karena **break**, **continue** atau **return**, **finally** block tetap dijamin akan dijalankan.



Contoh: exception handling

```
try {  
    // Statement yang biasanya berjalan tanpa masalah  
    // tapi mungkin menyebabkan exception  
}  
catch (SebuahException e1) {  
    // Menangani exception dengan tipe SebuahException  
    // atau subclass-nya. Info tentang exception bisa  
    // ditemukan dalam object e1  
}  
catch (SatuLagiException e2) {  
    // Menangani exception dengan tipe SatuLagiException  
    // atau subclass-nya. Info tentang exception bisa  
    // ditemukan dalam object e2  
}  
finally {  
    // Statement yang pasti dijalankan, bagaimanapun  
    // flow control berakhir dalam try block.  
}
```



Prinsip-prinsip OOP

- Prinsip utama OOP:
 - *Abstraction*: membuang detail yang “**tidak penting**”
 - *Encapsulation*: menggabungkan **data** dan **prosedur** di dalam sebuah object
 - *Inheritance*: tambahkan fungsionalitas dengan membuat **subclass** baru
 - *Polymorphism*: “banyak bentuk”. Dalam Java, ada 2 jenis:
 - **Overloading**: polymorphism secara trivial
 - **Overriding**: polymorphism yang sebenarnya



Overloading

- Memakai nama sama untuk method berbeda
- Method–method ini dibedakan berdasarkan **signature**–nya:
 - Jumlah parameter
 - Tipe parameter
 - Urutan parameter
- Perhatikan bahwa signature tidak tergantung
 - Nama parameter
 - Return type
- Overloading ditangani pada saat kompilasi (**static binding**)



Overriding

- Sebuah method yang memiliki nama **dan signature** yang sama dengan method dari superclass.
- Method dari subclass dikatakan meng-**override** method dari superclass.
- Ditangani pada saat *runtime* (**dynamic binding**): *instance type* menentukan method mana yang digunakan.



Polymorphism

- Dalam mendefinisikan method untuk sebuah subclass, ada tiga kemungkinan:
 - **override a method from the superclass** (Nama dan signature sama dengan method milik superclass).
 - **inherit method from the superclass**. Methods yang tidak ditulis ulang otomatis akan diturunkan kepada seluruh subclass.
 - **create a new method** (Nama dan signature berbeda dengan method milik superclass)



Polymorphism

- Ketika sebuah method dari sebuah object dipanggil (called), class/type dari object tersebut menentukan implementasi method yang mana yang akan dijalankan.
- Sebuah method/field dapat atau tidaknya dipanggil/diakses ditentukan oleh type of the reference variable.
- Kata kunci (keyword) “**super**” berfungsi sebagai reference terhadap object tapi diperlakukan sebagai instance dari superclass.
- Polymorfism dapat diartikan bahwa sebuah object dapat memiliki banyak ‘bentuk’, yaitu sebagai object dari class-nya sendiri atau sebagai object dari superclass-nya.



Abstract Class

- Abstract class sangat bermanfaat untuk menyimpan aspect generic dari subclass-nya.
- Sebuah method dapat didefinisikan tanpa implementasi dengan menyatakan method tersebut sebagai *abstract*.
- Abstract method tidak memiliki *body*, *body*-nya 'digantikan' dengan (;).
- Sebuah Class yang memiliki abstract method harus dinyatakan sebagai *abstract class*.
- Abstract class tidak dapat dibuatkan langsung instance-nya. (objectnya)
- Subclass dari abstract class dapat di buat instance-nya jika subclass *override dan implement seluruh abstract methods* dari superclass-nya.



Abstract Class

- Jika sebuah subclass dari sebuah abstract class **tidak meng-implement seluruh abstract methods dari superclass**, maka subclass tersebut adalah abstract class juga.



```
public abstract class Shape{
    public abstract double area ();
    public abstract double circumference ();
}
public class Circle extends Shape{
    protected double radius;
    public Circle (double r){
        radius = r;
    }
    public Circle (){
        this (1.0);
    }
    public double area (){
        return Math.PI * radius * radius;
    }
    public double circumference (){
        return 2 * Math.PI * radius;
    }
    public double getRadius (){
        return radius;
    }
}
```



```
public class Rectangle extends Shape{
    protected double length, width;
    public Rectangle (double length, double width)
    {
        this.length = length;
        this.width = width;
    }
    public Rectangle ()
    {
        this (1.0, 1.0);
    }
    public double area ()
    {
        return length * width;
    }
    public double circumference ()
    {
        return 2 * (length + width);
    }
    public double getLength ()
    {
        return length;
    }
    public double getWidth ()
    {
        return width;
    }
}
```



```
public class Square extends Rectangle
{
    public BujurSangkar (double edge)
    {
        super (edge, edge);
    }
    public BujurSangkar ()
    {
        this (1.0);
    }
    public double getEdge ()
    {
        return length;
    }
}
```



```

public class TryShape {
    public static void main (String args[]){
        Shape[] shape2 = new Shape[3];
        shape2[0] = new Circle (3.0);
        shape2[1] = new Rectangle (1.0, 2.0);
        shape2[2] = new Square (4.0);

        double totalArea = 0.0;
        double totalCircumference = 0.0;

        for (int ii = 0; ii < Shape2.length; ii++) {
            totalArea += shape2[ii].luas();
            totalCircumference += shape2[ii].keliling();
        }

        System.out.println ("Total Area = " + totalArea);
        System.out.println ("Total Circumference = "
            + totalCircumference);
    }
}

```



Interface

- Sebuah class hanya boleh meng-**extend satu superclass**, tapi boleh meng-**implement banyak interface**.
- Ketika meng-extend superclass, sebuah class mewarisi interface (definisi method) dan juga implementasinya.
- Ketika meng-implement interface, sebuah class hanya mewarisi interface (definisi method).
- Contoh: sifat generik “*comparable*” didefinisikan di dalam sebuah interface:

```
public interface Comparable {  
    public int compareTo (Object ob);  
}
```

- **compareTo** mengembalikan:
 - **<0**: object **this** “lebih kecil” dari parameter ‘ob’
 - **0**: object **this** sama nilainya dengan parameter ‘ob’
 - **>0**: object **this** “lebih besar” dari parameter ‘ob’



Contoh: interface

```
public class CircleComparable extends Circle
    implements Comparable
{
    public CircleComparable (double r)
    {super (r);}
    public int compareTo (Object other)
    {
        CircleComparable otherCircle =
            (CircleComparable) other;
        if (radius < otherCircle.getRadius ())
        {return -1;}
        else if (radius > otherCircle.getRadius ())
        {return 1;}
        else
        {return 0;}
    }
}
```



Interface

- Seluruh methods dalam sebuah interface adalah abstract method (keyword **abstract** dapat dinyatakan tapi optional)
- Seluruh variables dalam sebuah interface harus dinyatakan *static* dan *final* (*constant*)
- Class yang mengimplementasikan sebuah interface, harus memberikan implementation (menuliskan method body) untuk seluruh method yang diturunkan dari interface.
- Sebagai mana class, *interface* adalah tipe data (data type).
- Interface dapat juga digunakan untuk menyatakan *constant values*.



Interface

- Sebuah interface dapat meng- **extends** satu atau lebih interfaces.
- Sebuah sub-interface menurunkan seluruh abstract method dan constants dari super-interfaces.
- Sub-interface dapat menambahkan new abstract methods dan juga new constants.
- Contoh:

```
public interface DrawingObject
    extends Drawable, Transformable
{ ... }

public class Shape
    implements DrawingObject { ... }
```



Ringkasan

- Tipe data dalam Java: *primitive* dan *reference*
- Dalam meng-evaluasi sebuah *expression*, perhatikan *precedence* dan *associativity* dari operator, dan tipe operand
- Flow Control: branch, loop
- Class & Object
 - Object adalah ***instance*** dari class
 - **Behaviour** sebuah object dinyatakan dalam method
 - **State** sebuah object is dinyatakan dalam field



Ringkasan

- Memperbesar ukuran array adalah tidak efficient.
- Abstract Class: menyimpan aspect generic dari subclass-nya.
- Interface: Sekelompok method tanpa implementasi untuk menyamakan standarisasi penggunaan dengan implementasi yang bisa berbeda-beda.
- Exception digunakan untuk mengindikasikan kondisi special yang dapat terjadi.

