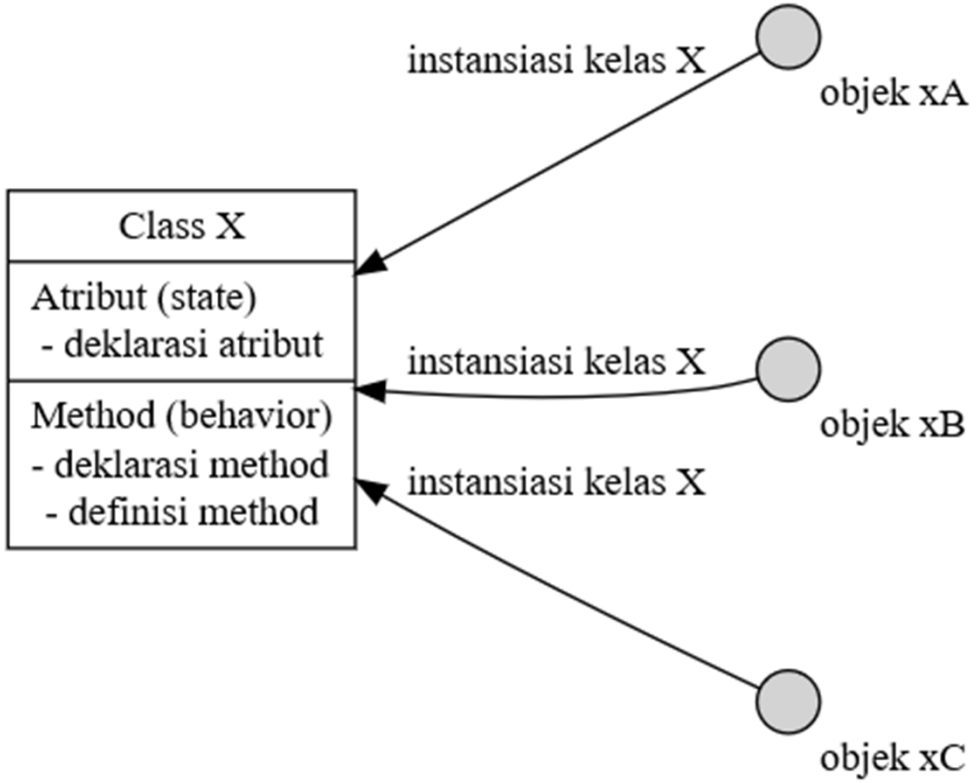
OOP di Java

IF2210 – Semester II 2020/2021

by: Yohanes Nugroho; rev: AI, SA, YW, SAR

Mengingat kembali OOP

- Objek: satuan unit, memiliki *state* dan *behavior*.
- Kelas: definisi statik dari objek, menyatakan tipe objek.
- Objek adalah *instance* dari suatu kelas.



Kelas di Java

- Kelas dituliskan dengan keyword `class`, dengan isi kelas menyatu dengan deklarasinya.
  - Di C++ deklarasi dan definisi boleh dipisah maupun disatukan.
- *Access modifier* harus ditulis untuk setiap member (baik atribut maupun *method*).

Contoh kelas sederhana

```java
// file Mahasiswa.java
class Mahasiswa {

  private String nama;
  private String nim;

  public Mahasiswa(String nim, String nama) {
    this.nim = nim;
    this.nama = nama;
  }

  public String getNama() { return nama; }
  public void setNama(String n) { nama = n; }
  public String getNIM() { return nim; }
}

// file MahasiswaTest.java
public class MahasiswaTest {
  public static void main(String args[]) {
    Mahasiswa mhs = new Mahasiswa("13577012", "Clark Kent");
    System.out.println("Nama: " + mhs.getNama());
  }
}
```

- class: definisi tipe baru.
  - variabel (*instance*) untuk tipe ini disebut objek.
- konstruktor: method khusus untuk penciptaan objek.
  - nama konstruktor sama dengan nama kelas.
- method: operasi yang disediakan suatu kelas.
- Contoh pemanggilan konstruktor dan method

Konvensi penamaan di Java

- Nama kelas diawali huruf kapital "`Mahasiswa`"
- Nama variabel, baik sebagai atribut kelas maupun variabel lokal dalam sebuah method (termasuk argumen method), diawali huruf kecil "`name`"
- Nama method diawali huruf kecil "`getName()`"
- Atribut (private) kelas yang disertai getter dan setter dengan nama yang sesuai (`name`, `getName()`, `setName()`) disebut sebagai property
  - Property ini yang dapat secara otomatis "dibuatkan editornya" di IDE, seperti saat mendesain GUI secara WYSIWYG
  - *Don't frantically generate getters and setters for all your attributes just because the IDE has that feature* – ingat kembali bahwa OOP "menyembunyikan" representasi internal

Konstruktor

- Di Java hanya ada konstruktor.
  - Tidak ada *copy constructor*.
  - Tidak ada destruktor.
    - Ada *garbage collection* (pemberesan memori secara otomatis).
    - Ada *finalizer* `finalize()`.
- Tidak ada operator =, tapi ada method `clone()`
  - Baca dokumentasi `clone` di API Java.

Deklarasi Konstruktor

- Nama konstruktor sama dengan nama kelas dan tidak memiliki nilai kembalian (sama seperti C++).
- Boleh ada banyak konstruktor (sama seperti C++).
- Contoh konstruktor:

```
Point(int x, int y) {
  this.x = x;
  this.y = y;
}
```

Mekanisme enkapsulasi

- Java menyediakan mekanisme pendefinisian *scope* anggota sebuah kelas dengan *access modifier*:
  - `private`: hanya dapat diakses objek itu sendiri.
  - `protected`: hanya dapat diakses objek dari kelas turunannya.
  - `public`: dapat diakses objek lain.
  - `default`: hanya dapat diakses oleh objek dari kelas turunannya yang berada di *package* yang sama.
- *Package*: ekuivalen dengan *namespace* di C++.
- Bacaan: https://medium.com/@iqbal_farabi/kembali-ke-dasar-bagian-pertama-enkapsulasi-cf2408c2528e

Penempatan *access modifier*

- *Access modifier* dilakukan dengan menempatkannya langsung di depan nama setiap atribut atau *method*.
  - Di C++ ada tanda ':' setelah *access modifier* dan berlaku untuk semua *method*/atribut sampai *access modifier* berikutnya.

```cpp
// C++
class Point {
  private:
    int x, y;
  public:
    int getX() { return x; }
    int getY() { return y; }
};
```

```java
// Java
class Point {
  private int x, y;
  public int getX() { return x; }
  public int getY() { return y; }
}
```

Contoh public

```
// file Point.java
class Point {
  public int x, y;
}
```

```
// file PointTest.java
public class PointTest {
  public static void main(String[] args) {
    Point p = new Point();
    System.out.println(p.x + " " + p.y);
  }
}
```

Contoh private

```
// file Point.java
class Point {
  private int x, y;
  public int getX() { return x; }
  public int getY() { return y; }
}

// file PointTest.java
public class PointTest {
  public static void main(String[] args) {
    Point p = new Point();
    System.out.println(p.x + " " + p.y); // error, karena private
    System.out.println(p.getX() + " " + p.getY());
  }
}
```

Reference this

- Pada Java, `this` adalah reference yang mengacu ke objek itu sendiri:

```
// file Point.java
class Point {
  private int x, y;
  public int getX() { return this.x; }
  public int getY() { return this.y; }
}
```

- sama dengan:

```
  // ...
  public int getX() { return x; }
  public int getY() { return y; }
  // ...
```

- Biasanya digunakan untuk mencegah ambiguitas (misal: ada parameter method bernama sama)

static dan alokasi memori

- Atribut & method hanya dapat diakses jika objek telah dibuat, contoh:

```
Mahasiswa mhs;
System.out.println(mhs.getNama()); // error
mhs = new Mahasiswa("Clark Kent");
System.out.println(mhs.getNama()); // ok
```

- Atribut & method `static` adalah milik kelas (diakses tanpa melalui objek).
  - Alokasi statis, hanya ada satu *instance* dalam seluruh program.
  - Diakses dengan menyebut nama kelas sebelum titik.


```java
// file Mahasiswa.java
class Mahasiswa {
  static int jumlah;
  String nama;
  String nim;
  public Mahasiswa(String n) {
    nim = n;
    jumlah = jumlah+1;
  }
  public static int getJumlah() {
    return jumlah;
  }
}

// file StaticTest.java
public class StaticTest {
  public static void main(String args[]) {
    Mahasiswa mhs1 = new Mahasiswa("13577012");
    System.out.println("Jumlah: " + Mahasiswa.getJumlah());
    Mahasiswa mhs2 = new Mahasiswa("13577013");
    System.out.println("Jumlah: " + Mahasiswa.getJumlah());
  }
}
```

Kelas Sederhana

- Kelas paling sederhana merepresentasi type (atribut) dan prototype (*method*).
- Paket ADT dalam bahasa C dapat ditranslasi menjadi sebuah kelas dengan konvensi sbb:
  - File `adt.h` tidak perlu ditulis.
  - File `adt.c` langsung menjadi sebuah kelas.
  - File `madt.c` menjadi sebuah kelas terpisah.
  - Bagian data dari ADT menjadi atribut kelas.
  - Fungsi/operasi terhadap ADT menjadi method kelas.
  - Setiap atribut dan method harus diberi *access modifier* secara eksplisit.

Contoh kelas ADT

- Lihat contoh program kecil:
  - Point
  - Stack

Method di Java

- Hanya ada sedikit perbedaan antara *method* di C++ dengan Java:
  - Java tidak memiliki parameter *default*.
  - *method* tidak bisa ditandai `const`.
- Tidak ada *pointer/reference* untuk tipe dasar.
  - Konsekuensi paling sederhana: tidak bisa membuat *method* untuk menukar (swap) dua tipe dasar.

Passing parameter

- *Passing* parameter selalu *by value*.
- Tipe primitif:
  - Dibuat salinannya.
  - Variabel parameter boleh diubah/dipakai, tapi tidak mengubah nilai/variabel pemanggil.
- Tipe *reference* (objek):
  - *Pass by reference*, *method* yang dipanggil mempengaruhi objek pada parameter.
  - Jika di-*assign* nilai baru, maka tidak akan terlihat efeknya oleh pemanggil.

Bandingkan kedua potongan kode

```
class A {
  int v = 0;
  void incA() { v++; }
  static void hello(A a, int z) {
    a.incA();
    z = 9;
  }
}


// ... somewhere else ...
A x = new A();
int m = 2;
A.hello(x, m);
//x.v = 1, m tetap 2
```

```
class A {
  int v = 0;
  void incA() { v++; }
  static void hello(A a, int z) {
    a = new A();
    a.incA();
    z = 9;
  }
}

// ... somewhere else ...
A x = new A();
int m = 2;
A.hello(x, m);
//x.v tetap 0, m tetap 2
```