

Class & Object in JAVA – part 2

Pertemuan 4



Topics

1. Relationship Between Class
2. Static Field & Method
3. Method Parameters
4. Object Construction
5. Packages



1. Relationship Between Class

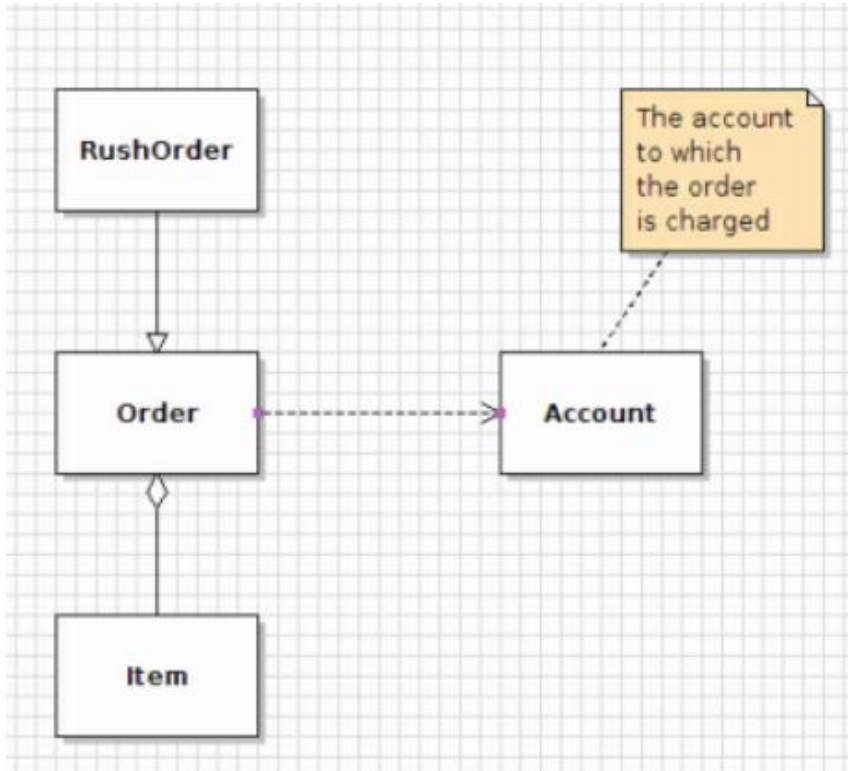


Relationship Between Class

- Dependence (“uses-a”)
- Aggregation (“has-a”)
- Inheritance (“is-a”)



Contoh “Class relationship”



Relationship	UML Connector
Inheritance	
Interface implementation	
Dependency	
Aggregation	
Association	
Directed association	

2. Static Fields & Methods



Static Fields

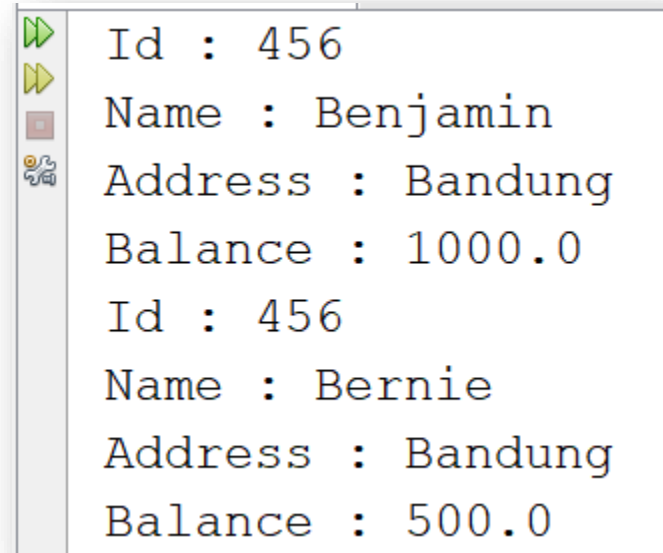
- Digunakan untuk mendefinisikan field yang bersifat static dan dapat diakses global (diambil dan diset nilainya) oleh semua objek yang merepresentasikan classnya.

```
public class Customer {  
    //encapsulation  
    public static String id = "123";  
    private String name;  
    private String address;  
    private double balance;
```

Desain Class

```
Customer customer = new Customer();  
customer.id = "456";  
customer.setName("Benjamin");  
customer.setAddress("Bandung");  
customer.setBalance(1000);
```

Instansiasi Objek



```
Id : 456  
Name : Benjamin  
Address : Bandung  
Balance : 1000.0  
Id : 456  
Name : Bernie  
Address : Bandung  
Balance : 500.0
```

Hasil Output

Static Constant

- Static constant pada implementasinya lebih sering digunakan dibandingkan dengan static fields.
- Static constant artinya variable bersifat tetap (tidak dapat diubah), namun dapat diakses (diambil nilainya) secara global oleh objek dari class tersebut.
- Contoh : dapat dipanggil nilainya melalui **Math.PI**

```
public class Math
{
    . . .
    public static final double PI = 3.14159265358979323846;
    . . .
}
```



Static Methods

- Static methods merupakan method yang tidak mengoperasikan objek, artinya tidak memiliki implicit parameter.
- Dapat digunakan pada 2 situasi berikut :
 - Ketika method tidak membutuhkan akses ke implicit parameter (field dari suatu objek), tetapi hanya disuplai dari explicit parameter.
 - Ketika method dalam memprosesnya hanya membutuhkan akses ke static fields saja



Factory Method

- Method yang digunakan untuk menginstansiasikan objek, namun bukan method constructor.
- Alasan kenapa perlu factory method :
 - Karena method constructor namanya tidak bisa diubah, sehingga agar lebih jelas memberikan nama method sendiri untuk mengkonstruksi objeknya.
 - Ketika anda menggunakan konstruktor, anda tidak dapat bervariasi tipe objek yang dibangun.

```
NumberFormat currencyFormatter = NumberFormat.getCurrencyInstance();  
NumberFormat percentFormatter = NumberFormat.getPercentInstance();  
double x = 0.1;  
System.out.println(currencyFormatter.format(x)); // prints $0.10  
System.out.println(percentFormatter.format(x)); // prints 10%
```



The main method

- Main method merupakan method static yang tidak memiliki implicit parameter. Namun secara umum main method digunakan untuk mengkonstruksi objek.

```
class Employee
{
    public Employee(String n, double s, int year, int month, int day)
    {
        name = n;
        salary = s;
        LocalDate hireDay = LocalDate.now(year, month, day);
    }
    . . .
    public static void main(String[] args) // unit test
    {
        Employee e = new Employee("Romeo", 50000, 2003, 3, 31);
        e.raiseSalary(10);
        System.out.println(e.getName() + " " + e.getSalary());
    }
    . . .
}
```

3. Method Parameters



Method parameters

- Tipe parameter :
 - By value
 - By reference



4. Object Construction



Object Construction

- Secara umum terdapat 2 cara untuk mengkonstruksi sebuah objek :
 - Dengan memanggil konstruktor dan menyisipkan nilainya melalui parameter
 - Dengan meng-assign value pada saat deklarasi objek
- Namun ada cara yang ketiga, yakni dengan initialization block.



Initialization Block

- Initialization block umumnya hanya men-set nilai awal (default) dari objek saat deklarasi, seperti 0, false atau null.

```
class Employee
{
    private static int nextId;

    private int id;
    private String name;
    private double salary;

    // object initialization block
    {
        id = nextId;
        nextId++;
    }

    public Employee(String n, double s)
    {
        name = n;
        salary = s;
    }

    public Employee()
    {
        name = "";
        salary = 0;
    }
}
```



Constructor

- Constructor adalah method yang secara otomatis dipanggil/dieksekusi saat sebuah class diinstansiasi.
- Nama constructor harus sama dengan nama class-nya.
- Sama halnya dengan method constructor juga dapat memiliki satu atau lebih dari satu parameter.



Contoh implementasi constructor

```
public class Mahasiswa
{
    private String nim, nama;
    public Mahasiswa()
    {
        this.nim = "";
        this.nama = "";
    }
}
```



Contd..

```
public class Mahasiswa
{
    private String nim, nama;
    public Mahasiswa()
    {
        this.nim="";
        this.nama = "";
    }
    public Mahasiswa(String nim, String nama)
    {
        this.nim = nim;
        this.nama = nama;
    }
}
```

Contoh class yang memiliki lebih dari satu constructor disebut **multiple constructor**



Function Overloading

- Function **Overloading** merupakan suatu kondisi dimana suatu class memiliki fungsi yang sama tetapi deklarasi dan parameternya berbeda.



Contoh implementasi :

```
public class Matematika
{
    ....
    public int Tambah (int a, int b)
    {
        return a + b;
    }
    public int Tambah (double a, double b)
    {
        return a + b;
    }
}
```



5. Package



Package

- Java memungkinkan sekumpulan class dikelompokkan kedalam satu packages.
- Packages dapat meningkatkan konsep modularity, sehingga dapat mengelola kode dengan lebih jelas (tanggungjawabnya).



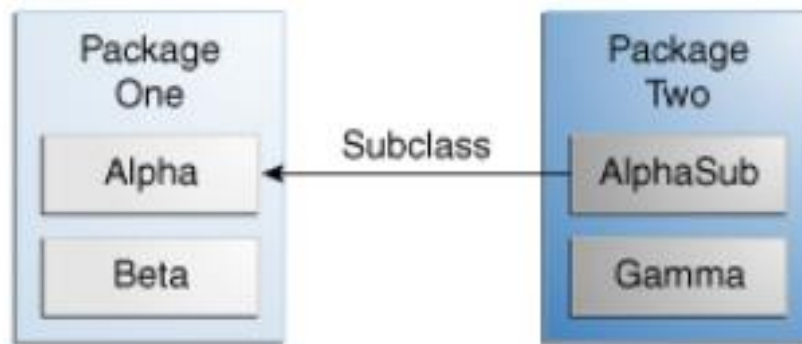
Class Importation

- Sebuah class dapat menggunakan semua kelas pada packagenya dan class public lain dari package yang berbeda.
- Untuk menggunakannya dapat melalui import package
- Kita juga dapat menggunakan notasi * untuk mengambil semua package pada prefix java (contoh : import java.*)

```
import java.util.ArrayList;  
import java.util.Date;
```



Package Scope



Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
<code>public</code>	Y	Y	Y	Y
<code>protected</code>	Y	Y	Y	N
<code>no modifier</code>	Y	Y	N	N
<code>private</code>	Y	N	N	N

Static Imports

- Disamping kelas, kita juga dapat mengimport method / fields yang sifatnya static.

For example, if you add the directive

```
import static java.lang.System.*;
```

to the top of your source file, then you can use the static methods and fields of the `System` class without the class name prefix:

```
out.println("Goodbye, World!"); // i.e., System.out  
exit(0); // i.e., System.exit
```

You can also import a specific method or field:

```
import static java.lang.System.out;
```



Menambahkan kelas pada packages

- Menuliskan package disertai nama pada awal deklarasi kelas

```
package com.horstmann.corejava;  
  
public class Employee  
{  
    . . .  
}
```



Demo Aplikasi

- Demo Aplikasi pada Netbeans



Kesimpulan :

1. Relationship Between Class
2. Static Field & Method
3. Method Parameters
4. Object Construction
5. Packages



Questions



Tugas

- Carilah sebuah contoh kode program Java yang memuat relasi antar kelas melalui :
 - Dependency
 - Aggregation
 - Inheritance
- Jelaskan masing-masing relasi tersebut pada kasus yang dibuat!
- Source code berikut penjelasannya dikumpulkan dalam dokumen PDF



References :

1. **Core Java Volume 1 (10th Edition), Chapter 4 Class & Object**

