4️⃣

Modul 4 - *Array of object* dan Asosiasi

Durasi Modul = 2-5 menit

Capaian pembelajaran:

- Memahami konsep dan implementasi *Array of Object* dalam pemrograman berorientasi objek
- Menjelaskan penggunaan *Array of Object* untuk menyimpan multiple objek dalam satu variabel
- Mengidentifikasi dan menerapkan hubungan asosiasi antar kelas dalam pemrograman berorientasi objek
- Membedakan asosiasi *unidirectional* dan *bidirectional* dalam hubungan antar kelas
- Menganalisis dan menggambarkan hubungan asosiasi dalam diagram kelas UML

Deskripsi Pembelajaran

Modul ini membahas dua konsep penting dalam pemrograman berorientasi objek: Array of Object dan Asosiasi. Pembelajaran akan mencakup:

- Pengenalan konsep *Array of Object* sebagai cara untuk menyimpan multiple objek dalam satu variabel array
- Implementasi dan penggunaan Array of Object dalam program
- Pemahaman tentang hubungan asosiasi antar kelas dalam pemrograman berorientasi objek
- Perbedaan antara asosiasi *unidirectional* dan *bidirectional*
- Representasi hubungan asosiasi dalam diagram kelas UML

Melalui penjelasan konsep, contoh kode, dan diagram, peserta akan memperoleh pemahaman komprehensif tentang kedua topik ini dan bagaimana menerapkannya dalam pengembangan perangkat lunak berorientasi objek.

4.1. *Array of Object*

4.1.1. Array

Sebelum membahas mengenai array of objek, ada baiknya kita mengulang kembali konsep array. Array adalah suatu variabel yang terdiri dari sekumpulan data dimana data-data tersebut mempunyai tipe data yang sama. Setiap data disimpan dalam alamat memori yang berbeda-beda dan disebut elemen array. Setiap elemen array diakses langsung melalui indeksnya.

Cara untuk mendeklarasikan array sebagai berikut:

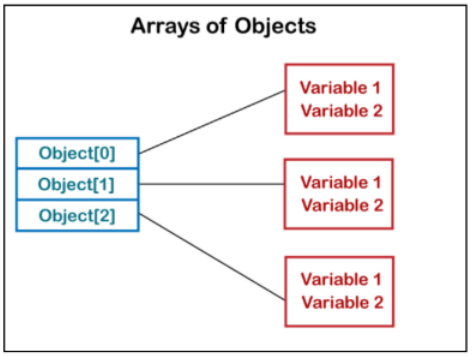
```
tipeData [] namaArray;

atau

tipeData namaArray[];
```

Contoh :

```
int [] dataJual;

atau

int dataJual[];
```

4.1.2. Array of Object

Jika **array** pada umumnya menyimpan banyak nilai dalam sebuah variable, maka Array of Object adalah sebuah cara untuk **menyimpan banyak object** dalam **sebuah variable array**. Jadi **elemen** dari *array of object* adalah ***object*** yang (nantinya juga) menyimpan nilai dari atribut. *Array of object* dapat digunakan jika kamu ingin membuat sebuah program dimana terdapat banyak object yang memiliki *behavior* atau aksi yang sama.



▼ Cara pembuatan array of object:

a. Deklarasi Array

```
ClassName obj[] = new ClassName [array_length] ;
```

Contoh:

```
Account obj[] = new Account[2] →
```

"Account obj []" adalah pendeklarasian sebuah array of object dari class " `Account"` yang akan menampung banyak object dengan nama "obj". "**New Account [2]"** menunjukkan bahwa **"obj"** adalah **object dari class "** `Account` **" yang akan menampung object sebanyak maksimal 2.**

b. Inisialisasi Objet of Array:

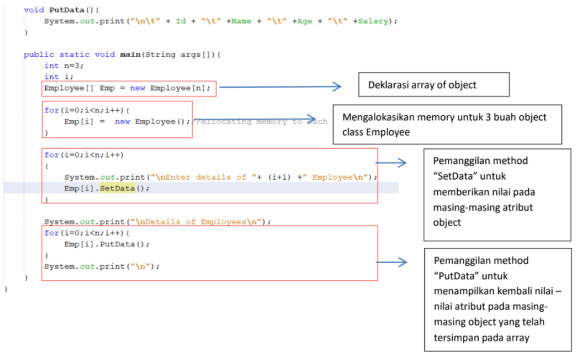
```
obj[0] = new Account();
obj[1] = new Account();
```

Alokasi memori untuk masing-masing object, jika kamu ingin menyimpan sebanyak 2 array, maka kamu harus mengalokasikan memory semua object dengan cara yang tersebut.

Contoh 1 :

Class di bawah hanyalah class biasa yang memiliki atribut dan sebuah method.

```
import java.util.Scanner;

class Employee{
    int Id;
    String Name;
    int Age;
    long Salary;

    void SetData(){
        Scanner sc = new Scanner(System.in);

        System.out.print("\n\tEnter Employee Id : ");
        Id = Integer.parseInt(sc.nextLine());

        System.out.print("\n\tEnter Employee Name : ");
        Name = sc.nextLine();

        System.out.print("\n\tEnter Employee Age : ");
        Age = Integer.parseInt(sc.nextLine());

        System.out.print("\n\tEnter Employee Salary : ");
        Salary = Integer.parseInt(sc.nextLine());
```

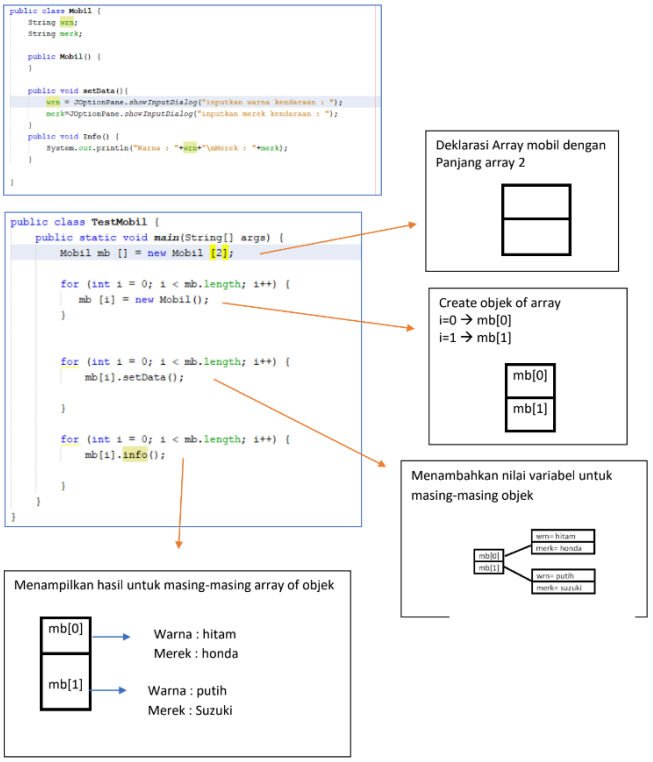


Pada potongan program ini terdapat pendeklarasian array of object dan cara pemanggilan
method dengan array of object.

Contoh 2:

Buatlah kelas

`mobil` yang memiliki warna dan merk, memiliki method setdata untuk input
data warna dan merek dan info untuk menampilkan informasinya dari mobil. Simpanlah data
dalam array of object dengan banyak data 2.



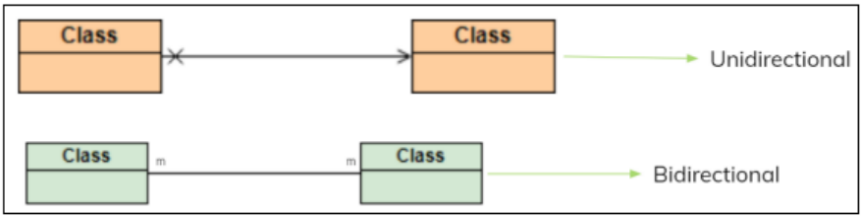
4.2. Asosiasi

1. **Asosiasi** adalah **hubungan antar kelas yang tidak saling memiliki**, **jika** sebuah **class**

   dihapus, **tidak akan mempengaruhi class lainnya.**

2. **Simbol** yang digunakan adalah **garis tegas dari suatu kelas ke kelas yang lain. Sering juga digunakan** satu garis tidak putus - putus dan tidak memiliki anak panah pada kedua ujungnya. Kemudian juga dibubuhkan kardinal pada garis tersebut.

Contoh Class Diagram:



Sebagai contoh, terdapat 2 buah class sebagai berikut:

```java
public class Bank {
    private String name;
    private int jmlPegawai=0;
    private String namaPegawai[]=new String[10];

    public Bank(String name){
        this.name=name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getJmlPegawai(){
        return this.jmlPegawai;
    }

    public String getNamaPegawai(int i) {
        return (namaPegawai[i]);
    }

    public void setNamaPegawai(String namaPegawai) {
        if (jmlPegawai<10){
            this.namaPegawai[jmlPegawai]=namaPegawai;
            jmlPegawai++;
        }
    }
}
```

```java
public class Pegawai {
    private String name;

    public Pegawai(String name){
        this.name=name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

> Dikatakan kedua class di atas memiliki hubungan asosiasi jika class `Bank` dan `Pegawai` tersebut saling **"memanfaatkan"** (*bidirectional*) atau salah satu class memanfaatkan **class lain** (*unidirectional*) melalui objectnya.

```java
public class AsosiasiBankPegawai {
    public static void main(String[] args) {
        Bank bank=new Bank("Mandiri");

        Pegawai pg1=new Pegawai("Budi");
        Pegawai pg2=new Pegawai("Wati");

        bank.setNamaPegawai(pg1.getName());
        bank.setNamaPegawai(pg2.getName());

        System.out.println("Jumlah pegawai: "+bank.getJmlPegawai());
        System.out.println("Nama-nama Pegawai: ");
        for (int i=0;i<bank.getJmlPegawai();i++){
            System.out.println((i+1)+". "+bank.getNamaPegawai(i));
        }
    }
}
```

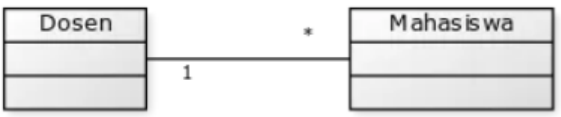
Asosiasi

Class Bank dan Class Pegawai berasosiasi melalui object-nya. Hubungan yang terbangun : One to many (1 Bank dapat memiliki banyak pegawai)

```
Jumlah pegawai: 2
Nama-nama Pegawai:
1. Budi
2. Wati
```

Pada contoh berikut, asosiasi yang terbangun adalah unidirectional, dimana hanya class bank saja yang menggunakan class pegawai melalui object class `pegawai`, sebagai berikut:

Contoh :



kelas diagram dengan hubungan asosiasi

📌 Dapat dilihat pada gambar kelas diagram di atas, hubungan **asosiasi** digambarkan dengan satu garis tidak putus - putus dan tidak memiliki anak panah pada kedua ujungnya. Kemudian juga dibubuhkan kardinal pada garis tersebut. Pada gambar di atas terdapat kelas `Dosen` dan kelas `Mahasiswa`, dilihat dari kardinalitasnya, satu `dosen` bisa memiliki banyak `mahasiswa`. Apabila `dosen` dihilangkan, `mahasiswa` masih bisa digunakan karena tidak saling memiliki.

Jenis multiplicity dapat dilihat pada gambar tabel dibawah ini.

Simbol	Arti
0	None/ nol
1	Satu
m	Nilai bertipe integer
0..1	Nol atau satu
m,n	M atau n
m..n	Minimal m tetapi tidak lebih dari n
*	Nol atau banyak
0..*	Nol atau banyak (dapat ditulis dengan *)
1..*	Satu atau banyak

📚REFERENSI

Aziefah, & Nurmasari, J. (2013). Pemrograman Berorientasi Objek. Pekanbaru: Politeknik Caltex Riau.

Barclay, K., & Savage, J. (2004). Object Oriented Design with UML and Java. Elsevier.

Deitel, P., & Deitel, H. (2012). Java: How to Program (9 ed.). US: Prentice Hall.