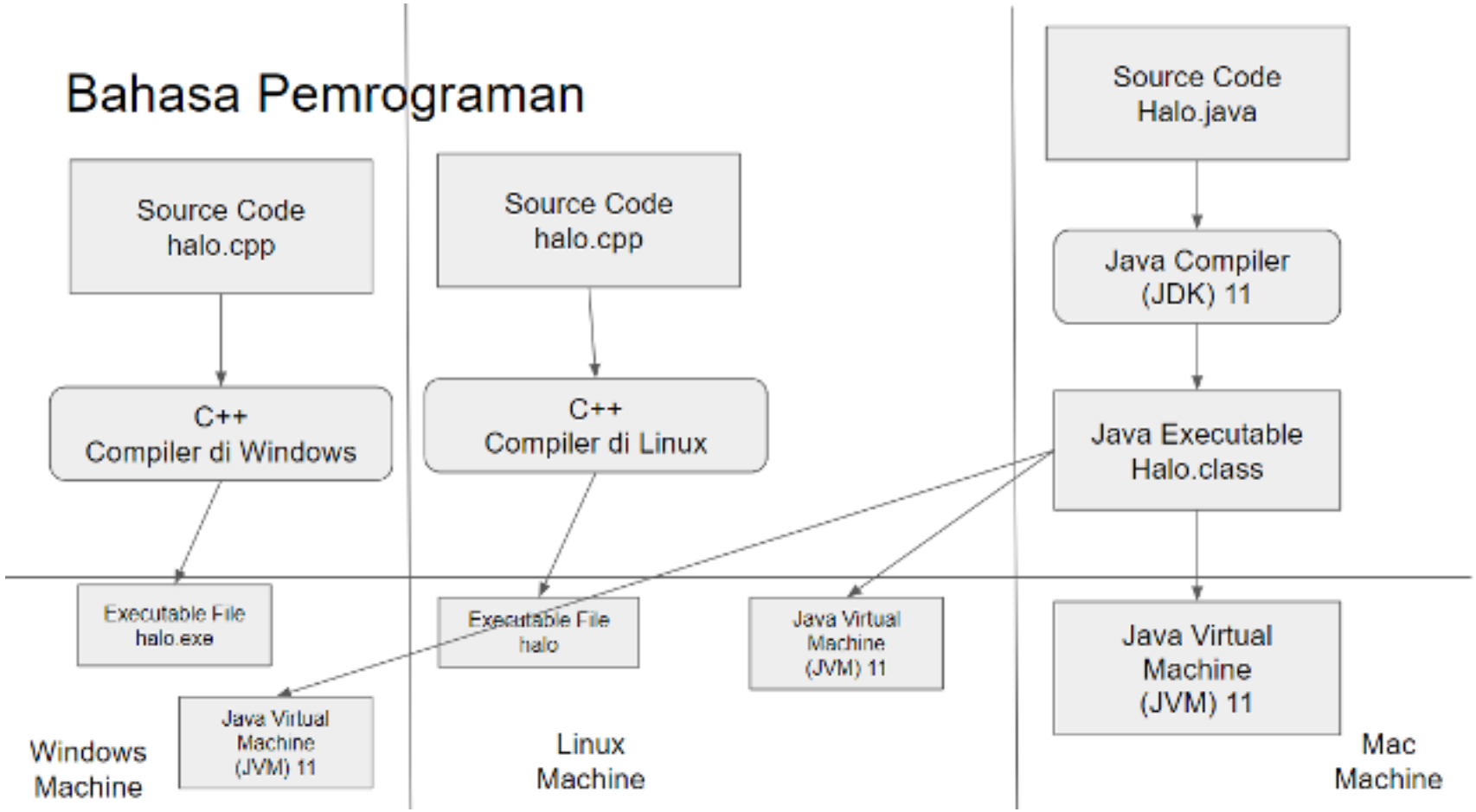
PBO P1: Object dan Class

Prepared by Adi Wahyu Pribadi

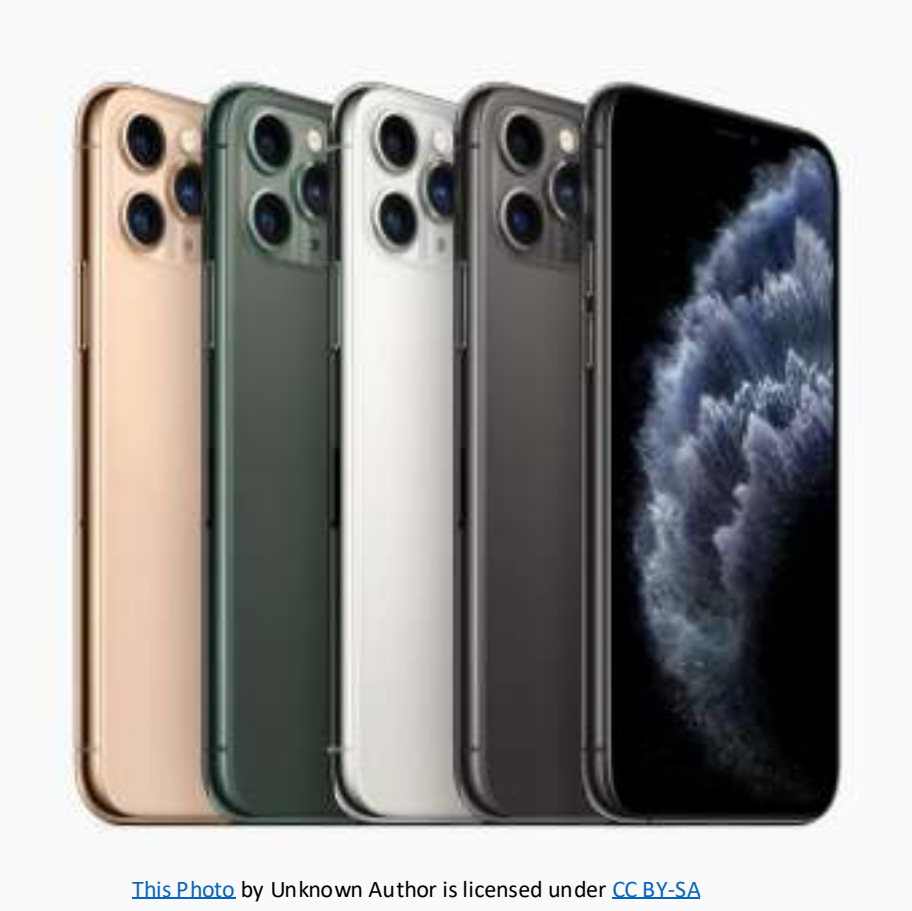
Pendahuluan

- Pemrograman Dasar => C++ Prosedural
  - Variable
  - Constant
  - Selection (if, if else, swith of)
  - Repetion/looping (for do, while do, do while)
  - Function/method
  - Recursive
  - Array
- Analisa Algoritma
  - Sorting
  - Searching
  - Tree
- Desain Web (HTML, CSS, JavaScript)
- Pemrograman Berbasis Object: Java
  - Pemrograman Berbasis Mobile (Flutter/Java/Kotlin)
  - Pemrograman Berbasis Web (PHP)

Java


Bahasa Pemrograman
Source Code
halo.cpp
C++
Compiler di Windows
Executable File
halo.exe
Java Virtual
Machine
(JVM) 11
Windows
Machine
Source Code
halo.cpp
C++
Compiler di Linux
Executable File
halo
Java Virtual
Machine
(JVM) 11
Linux
Machine
Source Code
Halo.java
Java Compiler
(JDK) 11
Java Executable
Halo.class
Java Virtual
Machine
(JVM) 11
Mac
Machine

Konsep Object & Class



Class iPhone:
Properties:
1. Color
2. Storage

Methods:
1. iPhone menyala
2. iPhone mati
3. iPhone berdering
4. iPhone video call

Object iPhone:
1. iPhone (gold color, 64GB)
   Nyala, mati, berdering, VC
2. iPhone (green color, 128GB)
   Nyala, mati, berdering, VC
3. iPhone (white color, 512GB)
   Nyala, mati, berdering, VC
4. iPhone (grey color, 64GB)
   Nyala, mati, berdering, VC
5. iPhone (dark grey color, 128GB)
   Nyala, mati, berdering, VC

Implementasi Object dan Class di Java

Class iPhone:
Properties:
1. Color
2. Storage
3. size

Methods:
1. iPhone menyala
2. iPhone mati
3. iPhone berdering
4. iPhone video call

```
class iPhone {

  String color;
  String storage;
  String size;

  nyala()
  mati()
  berdering()
  videocall()
}
```

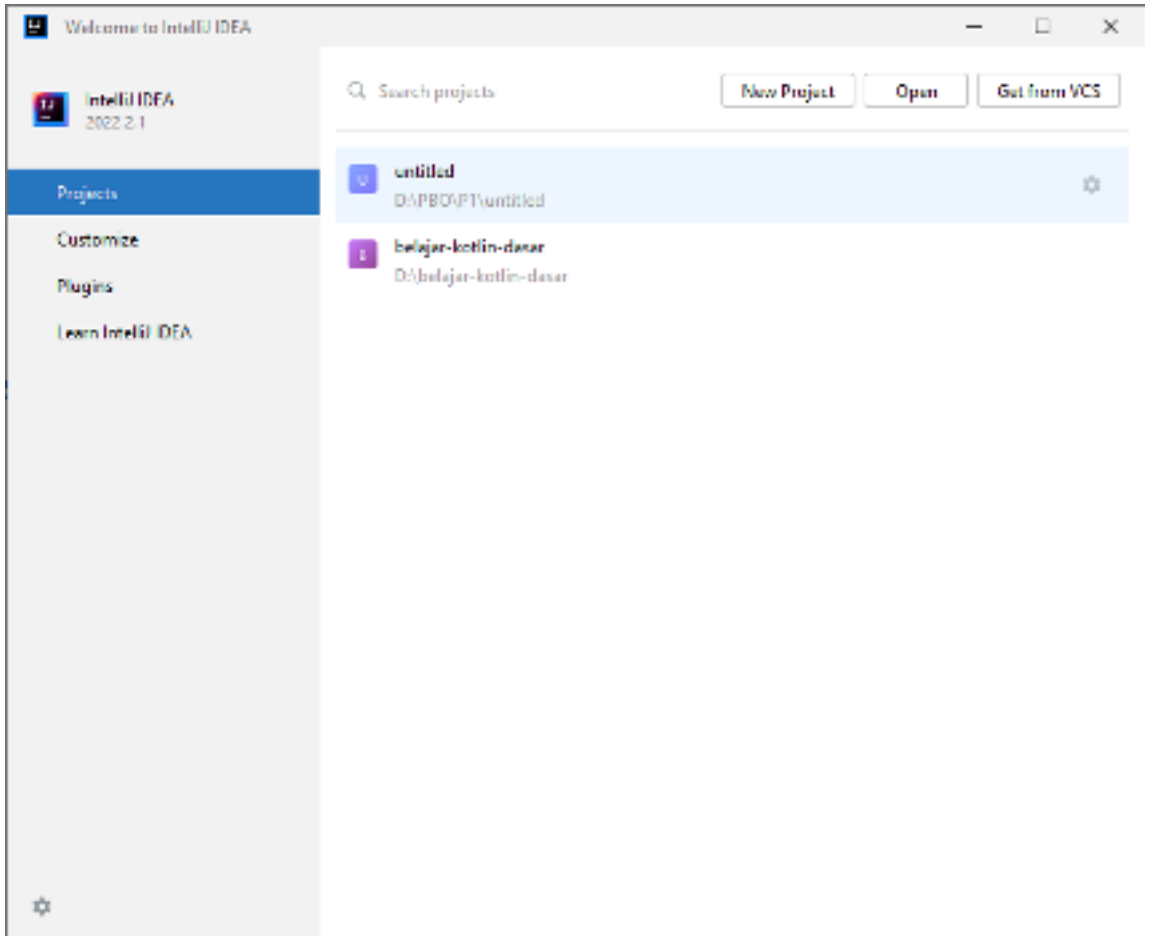
```
namaKelas namaObject = new
namaKelas();

iPhone iGold = new iPhone();
iPhone iGreen = new iPhone();
iPhone iGrey = new iPhone();
iPhone iDarkGrey = new iPhone();

iGold.color = “Gold”;
iGreen.color = “Green”;
iGrey.color = “Grey”
iDarkGrey.color = “Dark Grey”

iGold.storage = “64GB”’;
iGreen.size = “Max”;
```

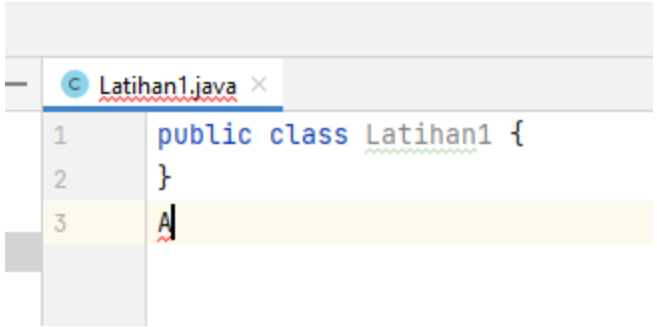
```
iGold.nyala();
iGold.videocall();

iGreen.nyala();
iGrey.nyala();
iDarkGrey.nyala();
```

Tools

- IntelliJ IDEA ➔ unduh di https://www.jetbrains.com/idea/download/#section=windows
- MS VS Code
  - Unduh Java di Oracle ➔ cari tutorial di YouTube

IntelliJ Idea





Nama file dan nama class harus sama

Nama file : Latihan1.java
Nama class : Latihan1

Huruf kecil dan besarnya harus sama persis

Apa itu Object Oriented Programming?

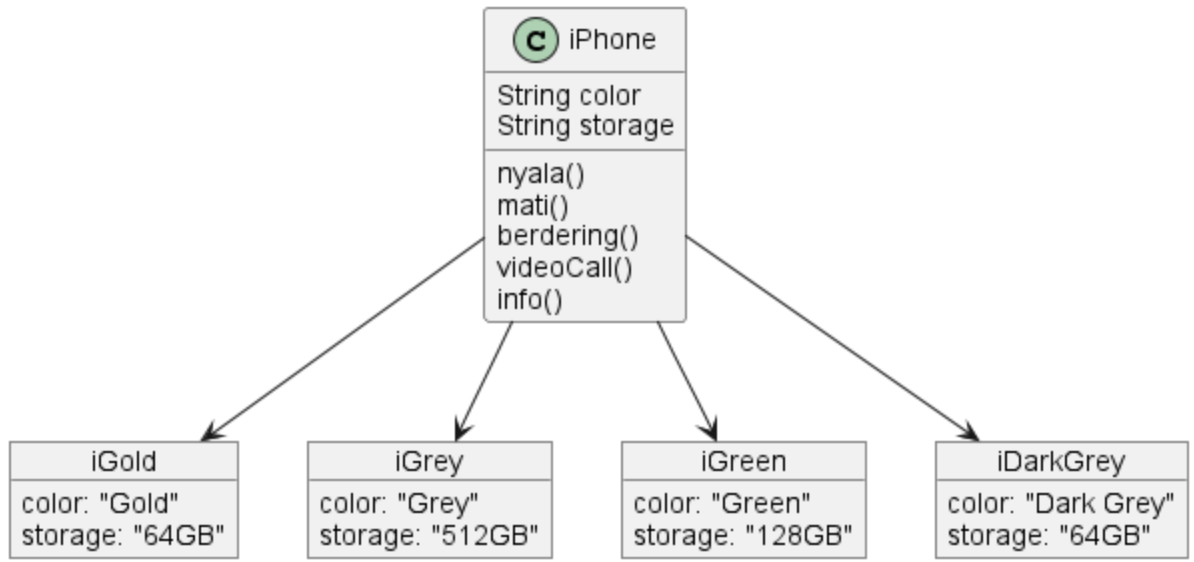
- Object Oriented Programming adalah sudut pandang bahasa pemrograman yang berkonsep “objek”
- Ada banyak sudut pandang bahasa pemrograman, namun OOP adalah yang sangat populer saat ini.
- Ada beberapa istilah yang perlu dimengerti dalam OOP, yaitu: Object dan Class

Apa itu Object?

- Object adalah data yang berisi field / properties / attributes dan method / function / behavior
- Semua data bukan primitif di Java adalah object, dari mulai Integer, Boolean, Character, String dan yang lainnya

Apa itu Class?

- Class adalah blueprint, prototype atau cetakan untuk membuat Object
- Class berisikan deklarasi semua properties dan functions yang dimiliki oleh Object
- Setiap Object selalu dibuat dari Class
- Dan sebuah Class bisa membuat Object tanpa batas


iPhone
String color
String storage
nyala()
mati()
berdering()
videoCall()
info()
iGold
color: "Gold"
storage: "64GB"
iGrey
color: "Grey"
storage: "512GB"
iGreen
color: "Green"
storage: "128GB"
iDarkGrey
color: "Dark Grey"
storage: "64GB"

Membuat Class

- Untuk membuat class, kita bisa menggunakan kata kunci class
- Penamaan class biasa menggunakan format CamelCase

```
2 usages
class iPhone {
    9 usages
    String color;
    5 usages
    String storage;

    4 usages
    public void nyala() { System.out.println("iPhone warna " + this.color + " menyala."); }
    4 usages
    public void mati() { System.out.println("iPhone warna " + this.color + " mati."); }
    2 usages
    public void bordering() { System.out.println("iPhone warna " + this.color + " bordering."); }
    2 usages
    public void videoCall() { System.out.println("iPhone warna " + this.color + " videoCall."); }
    4 usages
    public void info() {...}
}
```

Membuat Object

- Object adalah hasil instansiasi dari sebuah class
- Untuk membuat object kita bisa menggunakan kata kunci new, dan diikuti dengan nama Class dan kurung ()

```
// proses inisiasi objek dan pemberian properties kepada objek
iPhone iGold = new iPhone();
iPhone iGreen = new iPhone();
iPhone iGrey = new iPhone();
iPhone iDarkGrey = new iPhone();

iGold.color = "Gold";
iGreen.color = "Green";
iGrey.color = "Grey";
iDarkGrey.color = "Dark Grey";

iGold.storage = "64GB";
iGreen.storage = "128GB";
iGrey.storage = "512GB";
iDarkGrey.storage = "64GB";
```

Field

- Fields / Properties / Attributes adalah data yang bisa kita sisipkan di dalam Object
- Namun sebelum kita bisa memasukkan data di fields, kita harus mendeklarasikan data apa aja yang dimiliki object tersebut di dalam deklarasi class-nya
- Membuat field sama seperti membuat variable, namun ditempatkan di block class

Manipulasi Field

- Fields yang ada di object, bisa kita manipulasi. Tergantung final atau bukan.
- Jika final, berarti kita tidak bisa mengubah data field nya, namun jika tidak, kita bisa mengubah field nya
- Untuk memanipulasi data field, sama seperti cara pada variable
- Untuk mengakses field, kita butuh kata kunci . (titik) setelah nama object dan diikuti nama fields nya

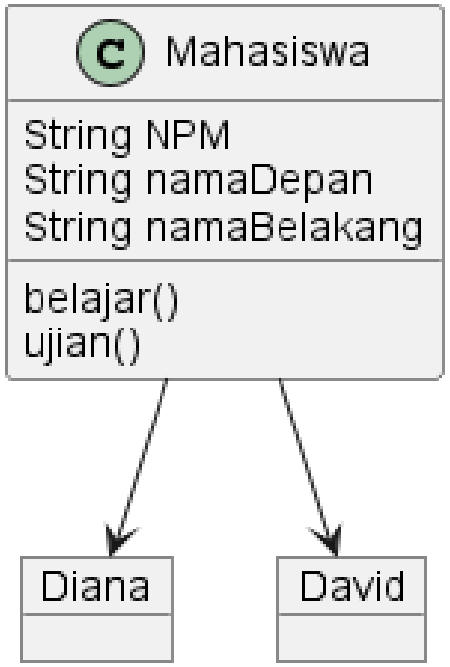
Method

- Selain menambahkan field, kita juga bisa menambahkan method ke object
- Cara dengan mendeklarasikan method tersebut di dalam block class
- Sama seperti method biasanya, kita juga bisa menambahkan return value, parameter dan method overloading di method yang ada di dalam block class
- Untuk mengakses method tersebut, kita bisa menggunakan tanda titik (.) dan diikuti dengan nama method nya. Sama seperti mengakses field

PBO P2: Constuctor, Visibility, This

Prepared by Adi Wahyu Pribadi

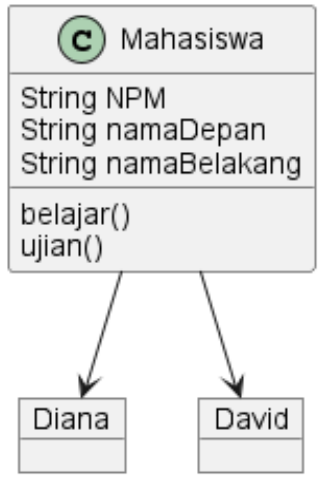
Konsep Class Lanjutan



- Konsep Class = membungkus data dan method / encapsulation

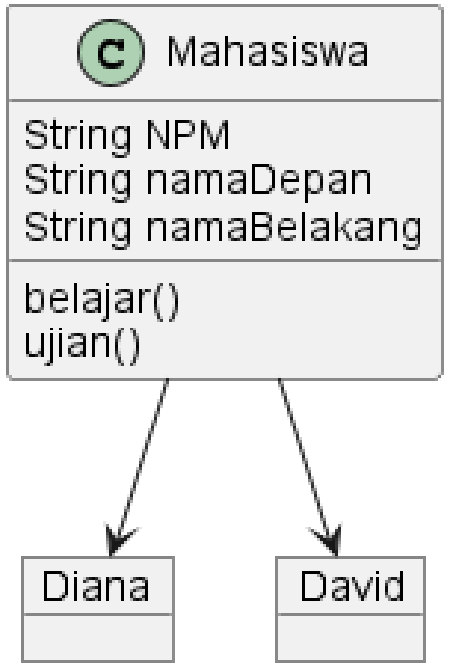
- Sebagai contoh ada di perkuliahan ada mahasiswa
- Kita memiliki Class Mahasiswa
- Dan terdapat Diana dan David.
- Kedua orang tersebut adalah objek dari Class Mahasiswa

Class Mahasiswa




```java
class Mahasiswa {
    String NPM;
    String namaDepan;
    String namaBelakang;

    void belajar() {
        System.out.println(this.NPM + " " + this.namaDepan + " " + this.namaBelakang + " " + " sedang belajar.");
    }
    void ujian() {
        System.out.println(this.NPM + " " + this.namaDepan + " " + this.namaBelakang + " " + " sedang ujian.");
    }
}
```

Membuat Object Mahasiswa dan Implementasi Aplikasi di main method




```java
public class Latihan2 {
    public static void main(String[] args) {
        Mahasiswa David = new Mahasiswa();
        Mahasiswa Diana = new Mahasiswa();

        David.NPM = "123";
        David.namaDepan = "David";
        David.namaBelakang = "Antonio";

        Diana.NPM = "124";
        Diana.namaDepan = "Diana";
        Diana.namaBelakang = "Nasution";

        David.belajar();
        Diana.belajar();

        David.ujian();
        Diana.ujian();
    }
}
```

Output

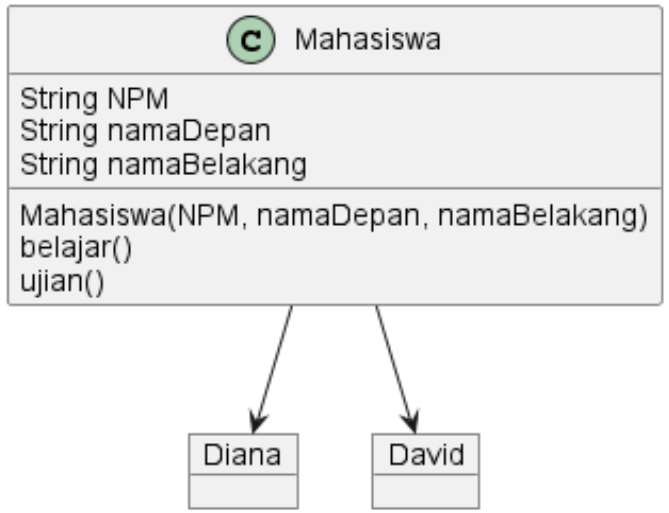
```
123 David Antonio  sedang belajar.
124 Diana Nasution  sedang belajar.
123 David Antonio  sedang ujian.
124 Diana Nasution  sedang ujian.

Process finished with exit code 0
```

Apa itu Constructor?

- method yang berada di dalam class fungsinya adalah memberikan nilai dari suatu properti/field Ketika sebuah objek pertama kali dibuat.
- Kalau di Java, deklarasi konstruktor adalah sama seperti nama class. Jika class nya Mahasiswa maka constructor dari class Mahasiswa adalah Mahasiswa() ==> namun kita juga bisa menambahkan variabel input.

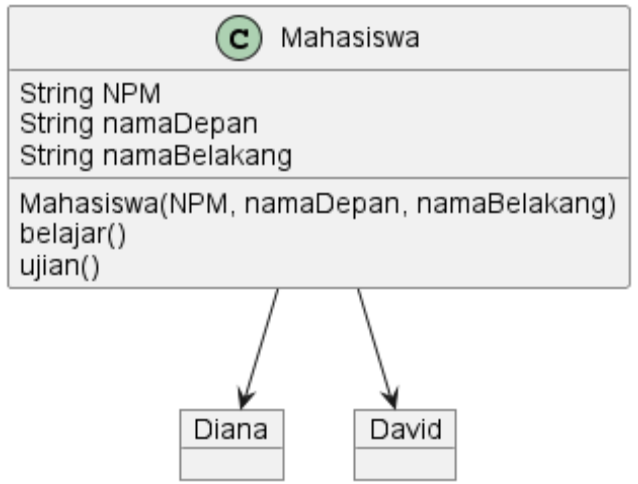
Class Mahasiswa dengan **Constructor**

```java
class Mahasiswa {
    String NPM;
    String namaDepan;
    String namaBelakang;

    Mahasiswa(String NPM, String namaDepan, String namaBelakang) {
        this.NPM = NPM;
        this.namaDepan = namaDepan;
        this.namaBelakang = namaBelakang;
    }
    void belajar() {
        System.out.println(this.NPM + " " + this.namaDepan + " " + this.namaBelakang + " " + " sedang belajar.");
    }
    void ujian() {
        System.out.println(this.NPM + " " + this.namaDepan + " " + this.namaBelakang + " " + " sedang ujian.");
    }
}
```



Membuat Object Mahasiswa dan Implementasi Aplikasi di main method **yang baru**

```
19  public class Latihan2 {
20      public static void main(String[] args) {
21          Mahasiswa David = new Mahasiswa( NPM: "123",  namaDepan: "David",  namaBelakang: "Antonio");
22          Mahasiswa Diana = new Mahasiswa( NPM: "124",  namaDepan: "Diana",  namaBelakang: "Nasution");
23
24          David.belajar();
25          Diana.belajar();
26
27          David.ujian();
28          Diana.ujian();
29      }
30  }
31
```



Output

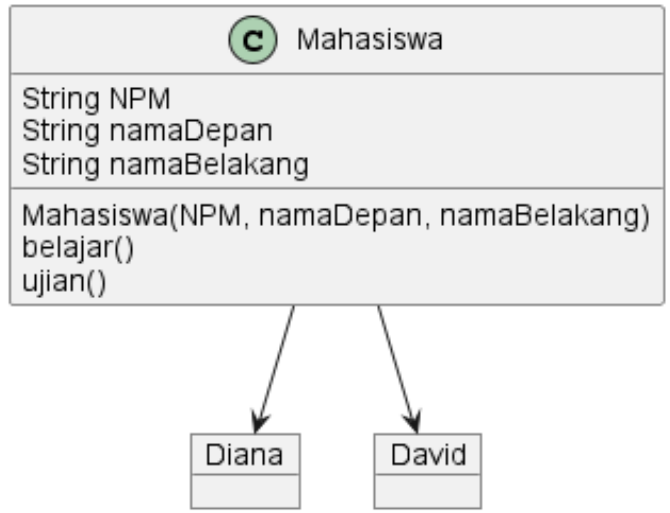
```
123 David Antonio  sedang belajar.
124 Diana Nasution  sedang belajar.
123 David Antonio  sedang ujian.
124 Diana Nasution  sedang ujian.

Process finished with exit code 0
```

Visibility pada Method dan Properti

- Property, Method dan Konstanta (khusus konstanta mulai PHP 7.1.0) dapat dikontrol aksesnya menggunakan visibility keyword
- Terdapat tiga keyword yaitu ***public***, ***protected*** dan ***private***
  - ***Public***: Artinya property, method atau Konstanta dapat diakses dari dalam maupun luar class.
  - ***Protected***: Artinya property, method atau konstanta hanya dapat diakses dari dalam class dan extended/inherited class (akan dijelaskan pada pelajaran tentang pewarisan).
  - ***Private***: Artinya property, method atau konstanta hanya dapat diakses dari dalam class itu sendiri.

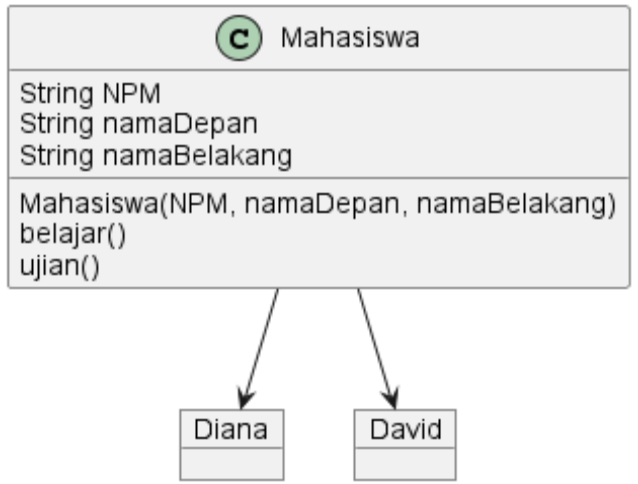
Class Mahasiswa dengan Visibility

```java
class Mahasiswa {
    private String NPM;
    private String namaDepan;
    private String namaBelakang;

    public Mahasiswa(String NPM, String namaDepan, String namaBelakang) {
        this.NPM = NPM;
        this.namaDepan = namaDepan;
        this.namaBelakang = namaBelakang;
    }
    public void belajar() {
        System.out.println(this.NPM + " " + this.namaDepan + " " + this.namaBelakang + " " + " sedang belajar.");
    }
    public void ujian() {
        System.out.println(this.NPM + " " + this.namaDepan + " " + this.namaBelakang + " " + " sedang ujian.");
    }
}
```



Membuat Object Mahasiswa dan Implementasi Aplikasi di main method **yang baru**

```
19  public class Latihan2 {
20      public static void main(String[] args) {
21          Mahasiswa David = new Mahasiswa( NPM: "123",  namaDepan: "David",  namaBelakang: "Antonio");
22          Mahasiswa Diana = new Mahasiswa( NPM: "124",  namaDepan: "Diana",  namaBelakang: "Nasution");
23
24          David.belajar();
25          Diana.belajar();
26
27          David.ujian();
28          Diana.ujian();
29      }
30  }
31
```



Output

```
123 David Antonio  sedang belajar.
124 Diana Nasution  sedang belajar.
123 David Antonio  sedang ujian.
124 Diana Nasution  sedang ujian.

Process finished with exit code 0
```

Definisi dan Penggunaan *this*

- Kata kunci this mengacu pada objek saat ini dalam metode atau konstruktor.
- Penggunaan paling umum dari kata kunci this adalah untuk menghilangkan kebingungan antara atribut kelas dan parameter dengan nama yang sama (karena atribut kelas dibayangi oleh metode atau parameter konstruktor).
- ***this*** juga dapat digunakan untuk:
  - Panggil konstruktor kelas saat ini
  - Panggil metode kelas saat ini
  - Kembalikan objek kelas saat ini
  - Berikan argumen dalam pemanggilan metode
  - Berikan argumen dalam panggilan konstruktor

Definisi dan Penggunaan *this*

- Silakan baca referensi tambahan mengenai this pada link berikut:
  - https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html
  - https://www.javatpoint.com/this-keyword
  - https://www.geeksforgeeks.org/this-reference-in-java/
  - https://www.guru99.com/java-this-keyword.html

Inheritance

Prepared by Adi Wahyu Pribadi

Review Pertemuan Minggu ke-4

- Kita sudah tahu apa itu
  - Class
  - Object
  - Attributes
  - Methods
  - Instantiate Objects
- Membuat program sederhana Java yang berorientasi object
- Tools yang digunakan VS Code dengan plugins Java

Sub CPMK ke-4 (Minggu ke-5)

- Mahasiswa mampu mengimplementasikan konsep inheritance pada Bahasa Java
    - Parent Class, Child Class, Contoh Karyawan Parent Class dan Dosen sebagai Child Class

Konsep Inheritance/Penurunan

Inheritance pada prinsipnya adalah menurunkan atribut/method suatu Super Class (Parent Class) ke Sub Class (Child Class).

Misalkan ada Karyawan di UP yang memiliki

Artribut: kode karyawan, nama
Method: absenDatang(), kerja(), absenPulang()

Lalu ada karyawan lain misalkan Dosen. Dosen juga merupakan karyawan. Jadi semua atribut dan method di Karyawan dapat digunakan di Dosen.

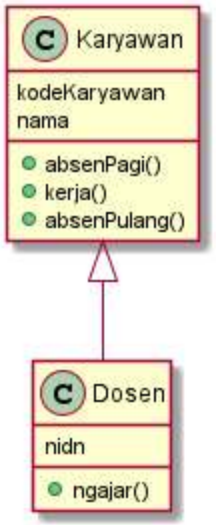
Namun begitu biasanya ada yang membedakan.

Dosen ada
Atribut: NIDN (Nomor Induk Dosen)
Method: mengajar().

Kita bisa bilang bahwa Dosen adalah Child/Sub Class dari Karyawan

Karyawan adalah Parent/Super Class dari Dosen

UML: Class Diagram



Misalkan kita membuat 1 objek Karyawan dan 1 objek Dosen

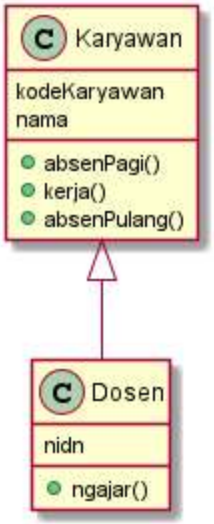
Dadang adalah Karyawan

Amir adalah Dosen

Atribut Dadang yang merupakan objek dari Karyawan dapat dipakai semua oleh Amir

Begitu juga method Dadang absenPagi(), kerja(), dan absenPulang() dapat dipakai Amir

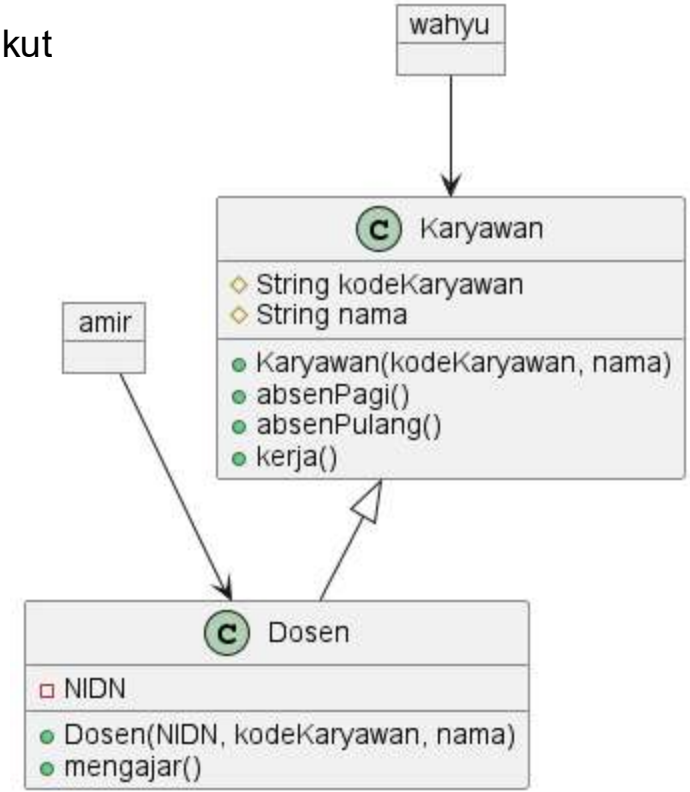
Implementasi Class Diagram Karyawan → Dosen




```
class Karyawan {
   String kodeKaryawan;
   String nama;
   absenPagi(){}
   kerja(){}
   absenPulan(){}
}

class Dosen extends Karyawan {
   String NIDN;
   ngajar(){}
}
```

```
Karyawan dadang = new Karyawan();
dadang.kodeKaryawan = “12345”;
Dadang.nama = “Dadang Suradang”;

Dosen amir = new Dosen();
amir.kodeKaryawan = “12346”;
amir.NIDN = “23467”;
Amir.nama = “Amir Murtako”;

dadang.absenPagi();
dadang.kerja();

amir.absenPagi();
amir.ngajar();

dadang.absenPulang();
amir.absenPulang();
```

Implementasi di Java 18 berikut
Class Diagram UML nya



Contoh Lain Inheritance

Sekarang contoh lain, misalkan bangun datar.

Yang termasuk bangun datar adalah:

- Segitiga
- PersegiPanjang
- Lingkaran

Ketiga bangun datar tersebut pasti bisa dihitung:

- Keliling, dan
- Luas

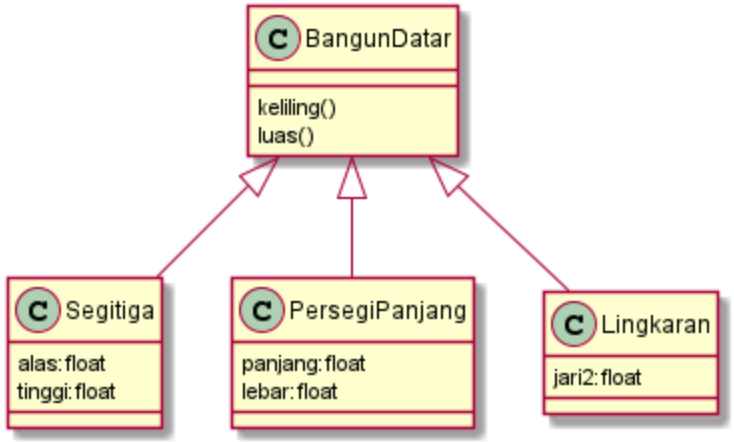
Yang membedakan ketiganya adalah atributnya

- Segitiga → alas dan tinggi
- PersegiPanjang → panjang dan lebar
- Lingkaran → diameter/jari-jari

Dari sini kita bisa membuat Super/Parent Class BangunDatar yang memiliki Sub/Child Class

- Segitiga
- PersegiPanjang
- Lingkaran

UML: Class Diagram



Dari gambar class Diagram di samping

Bisa kita simpulkan bahwa: Segitiga, PersegiPanjang, dan Lingkaran dapat menggunakan method keliling() dan luas()

Namun untuk menghitung keliling() dan luas() masing-masing BangunDatar menggunakan rumus yang berbeda karena parameter input untuk menghitungnya berbeda-beda

Untuk lebih memahami konsep BangunDatar sila kunjungi Website PetaniKode

https://www.petanikode.com/java-oop-inheritance/

Method Overriding

Method Overriding dilakukan saat kita ingin membuat ulang sebuah method pada sub-class atau class anak.

Method Overriding dapat dibuat dengan menambahkan anotasi @Override di atas nama method atau sebelum pembuatan method.

Seperti pada contoh BangunDatar, cara menghitung keliling() dan luas() dari turunan BangunDatar berbeda-beda. Maka kita gunakan @override pada method keliling() dan luas() dari setiap class turunan BangunDatar.

Lihat lagi di https://www.petanikode.com/java-oop-inheritance/

Contoh Method Overriding

Contoh lain method overriding

Pada class Karyawan ada method kerja() yang menampilkan kata “saya kerja di gedung fakultas teknik”, lalu ada child Dosen yang juga memiliki method kerja() namun di dalamnya menampilkan “saya rapat di ruangan prodi”

Kita bisa bilang bahwa method kerja() di Dosen menimpa/*overriding* method kerja() di Karyawan. Sedangkan method kerja() di Karyawan ditimpa/*overriden* oleh method kerja() Dosen.

Java Modifiers

Class dalam program Java dapat saling berhubungan dengan cara memberikan akses terhadap member mereka.

Semua yang ada di dalam class (atribut dan method) disebut member. Biasanya akan ada tingkatan akses yang disebut modifier.

Pada hubungan inheritance, semua member di dalam class induk akan bisa diakses oleh class anak (subclass), kecuali member tersebut diberikan modifier private.

Java Modifiers: public, private, dan protected.

Apabila kita tidak menggunakan tiga kata kunci tersebut, maka member atau class itu tidak menggunakan modifier (no modifier).

Masing-masing modifier akan menentukan di mana saja member bisa diakses.

Di diagram UML

Public tanda plus ‘+’
Private tanda min ‘-’
Protected tanda pagar ‘#’

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

- Y artinya bisa diakses;
- N artinya tidak bisa diakses;
- Subclass artinya class anak;
- World artinya seluruh package di aplikasi.

Java Modifiers

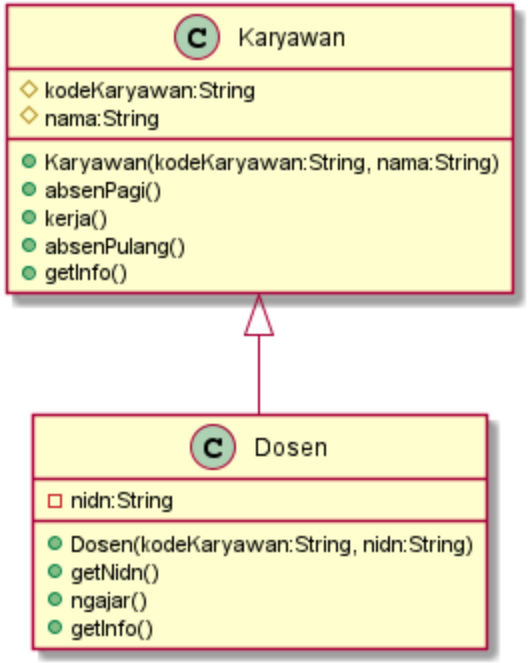
Public: modifier public akan membuat member dan class bisa diakses dari mana saja.

Private: Modifier private akan membuat member hanya bisa diakses oleh dari dalam class itu sendiri.

Protected: Modifier protected akan membuat member dan class hanya bisa diakses dari:

- Class itu sendiri;
- Sub class atau class anak;
- Package (class yang berada satu package dengannya).

UML: Class Diagrams



Karyawan.java

```java
package app;

class Karyawan {
    protected String kodeKaryawan;
    protected String nama;

    public Karyawan(String kodeKaryawan, String nama) {
        this.kodeKaryawan = kodeKaryawan;
        this.nama = nama;
    }

    public void absenPagi() { System.out.println(this.nama + ": absen pagi");
    }

    public void kerja() { System.out.println(this.nama + ": sedang bekerja");
    }

    public void absenPulang() { System.out.println(this.nama + ": absen pulang");
    }

    public void getInfo() {
        System.out.println("Kode Karyawan: " + this.kodeKaryawan);
        System.out.println("Nama: " + this.nama);
    }
}
```

Dosen.java

```java
package app;

class Dosen extends Karyawan {
    private String NIDN;

    public Dosen(String kodeKaryawan, String NIDN, String nama) {
        super(kodeKaryawan, nama);
        this.NIDN = NIDN;
    }

    public void setNIDN(String NIDN) { this.NIDN = NIDN;
    }

    public void getNIDN() { System.out.println("NIDN: " + this.NIDN);
    }

    public void ngajar() { System.out.println(this.nama + ": sedang mengajar");
    }

    @Override // method getInfo() akan menimpa getInfo() parent
    public void getInfo() {
        super.getInfo();
        System.out.println("NIDN: " + this.NIDN);
    }
}
```

App.java

```
package app;

public class App {
    public static void main(String[] args) throws
Exception {
        Karyawan Ridho = new Karyawan("123", "Ridho");

        Ridho.getInfo();
        Ridho.absenPagi();
        Ridho.kerja();
        Ridho.absenPulang();

        Dosen Amir = new Dosen("124", "32", "Amir");

        Amir.getInfo();
        Amir.absenPagi();
        Amir.kerja();
        Amir.ngajar();
        Amir.absenPulang();
    }
}
```

Output

```
Kode Karyawan: 123
Nama: Ridho
Ridho: absen pagi
Ridho: sedang bekerja
Ridho: absen pulang
Kode Karyawan: 124
Nama: Amir
NIDN: 32
Amir: absen pagi
Amir: sedang bekerja
Amir: sedang mengajar
Amir: absen pulang
```

Penutup

- Konsep Inheritance
- Karyawan dan Dosen
- Implementasi Class dan Object Karyawan dan Dosen di Java

- Next: Relasi antar object.
    - Supir relasi ke sebuah mobil
    - Mobil memiliki ban dan mesin
    - Fakultas memiliki beberapa Program Studi

PERTEMUAN KEEMPAT

Polymorphism, Class Association

Prepared by Adi Wahyu Pribadi

Polymorphism

Polimorfisme dalam OOP adalah sebuah prinsip di mana class dapat memiliki banyak bentuk method yang berbeda-beda meskipun namanya sama. “Bentuk” di sini dapat kita artikan: isinya berbeda, parameternya berbeda, dan tipe datanya berbeda.

Polimorfisme pada Java ada dua macam:

1. Static Polymorphism (Polimorfisme statis);
2. Dynamic Polymorphism (Polimorfisme dinamis).

Polimorfisme statis menggunakan **method overloading** sedangkan polimorfisme dinamis menggunakan **method overriding**.

Static Polymorphism/Method Overloading

Method overloading terjadi pada sebuah class yang memiliki nama method yang sama tapi memiliki parameter dan tipe data yang berbeda.

Sebagai contoh ada dua bangun datar segi empat yaitu segi empat yang memiliki panjang dan lebar berbeda, dan segi empat yang panjang dan lebarnya sama atau biasa disebut bujur sangkar. Menghitung luas dan keliling kedua bangun datar tersebut kita gunakan rumus berbeda. Maka method luas dan keliling memiliki paramater yang berbeda.

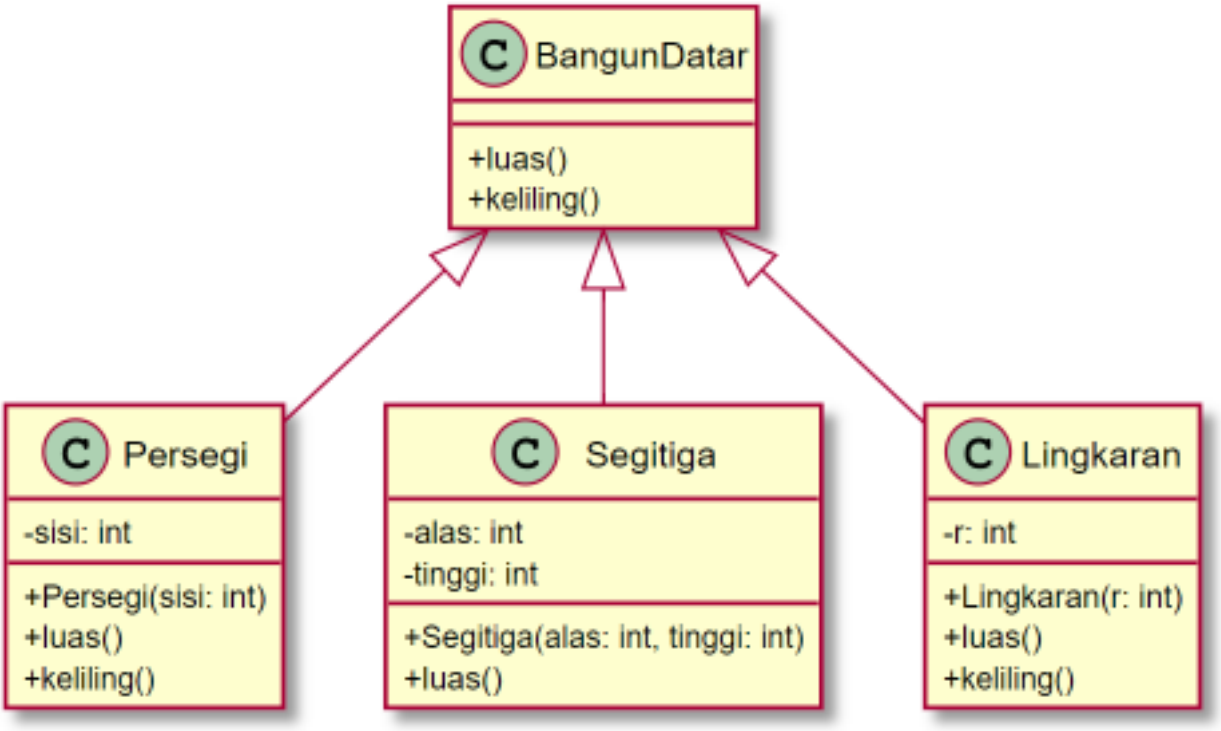
Parameter input luas dan keliling segi empat adalah panjang dan lebar, dan parameter input bujur sangkar adalah sisi. Sebagaimana terlihat pada berikut yaitu class diagram SegiEmpat yang memiliki 4 method. Ada dua method untuk menghitung luas karena luas **SEGI EMPAT** adalah ***panjang x lebar***, dan luas **BUJUR SANGKAR** adalah ***sisi x sisi***. Sedangkan method untuk menghitung keliling juga ada dua. Keliling **SEGI EMPAT** adalah ***(panjang + lebar) x 2*** dan keliling **BUJUR SANGKAR** adalah ***sisi x 4***.


```
class SegiEmpat {
    public int luas(int panjang, int lebar) {
        return panjang * lebar;
    }

    public int luas(int sisi) {
        return sisi * 4;
    }

    public int keliling(int panjang, int lebar) {
        return (panjang + lebar) * 2;
    }

    public int keliling(int sisi) {
        return sisi * 4;
    }
}


public class App {
    public static void main(String[] args) throws Exception {
        SegiEmpat s1 = new SegiEmpat();
        SegiEmpat bs = new SegiEmpat();

        System.out.println("Luas Segi Empat: " + s1.luas(10, 4));
        System.out.println("Luas Bujursangkar: " + bs.luas(8));
        System.out.println("Keliling Segi Empat: " + s1.keliling(10, 4));
        System.out.println("Keliling Bujur Sangkar: " + bs.keliling(8));
    }
}
```

Output:

```
Luas Segi Empat: 40
Luas Bujursangkar: 32
Keliling Segi Empat: 28
Keliling Bujur Sangkar: 32
```

Dynamic Polymorphism/Method Overriding



Pada diagram tersebut, terdapat class BangunDatar yang memiliki tiga subclass, yaitu: Persegi, Lingkaran, dan Segitiga.

Setiap class memiliki method yang sama yaitu luas() dan keliling(). Akan tetapi method-method ini memiliki isi rumus yang berbeda.

Berikut Source Code dari class diagram di atas yaitu BangunDatar.java, Persegi.java, Segitiga.java dan Lingkaran.java

BangunDatar.java

```
package app;
```

```java
public class BangunDatar {
    float luas(){
        System.out.println("Menghitung luas bangun datar");
        return 0;
    }

    float keliling(){
        System.out.println("Menghitung keliling bangun datar");
        return 0;
    }
}
```

Persegi.java

```java
package app;

class Persegi extends BangunDatar{
    private int sisi;

    public Persegi(int sisi) {
        this.sisi = sisi;
    }

    @Override
    public float luas() {
        return this.sisi * this.sisi;
    }

    @Override
    public float keliling(){
        return this.sisi * 4;
    }
}
```

Segitiga.java

```java
package app;

class Segitiga extends BangunDatar{
    private int alas;
    private int tinggi;

    public Segitiga(int alas, int tinggi) {
        this.alas = alas;
        this.tinggi = tinggi;
    }

    @Override
    public float luas(){
        return this.alas * this.tinggi;
    }
 }
```

Lingkaran.java

```java
package app;

class Lingkaran extends BangunDatar {
    private int r;

    public Lingkaran(int r) {
        this.r = r;
    }

    @Override
    public float luas(){
        return (float) (Math.PI * r * r);
    }

    @Override
    public float keliling(){
        return (float) (2 * Math.PI * r);
    }

}
```

App.java

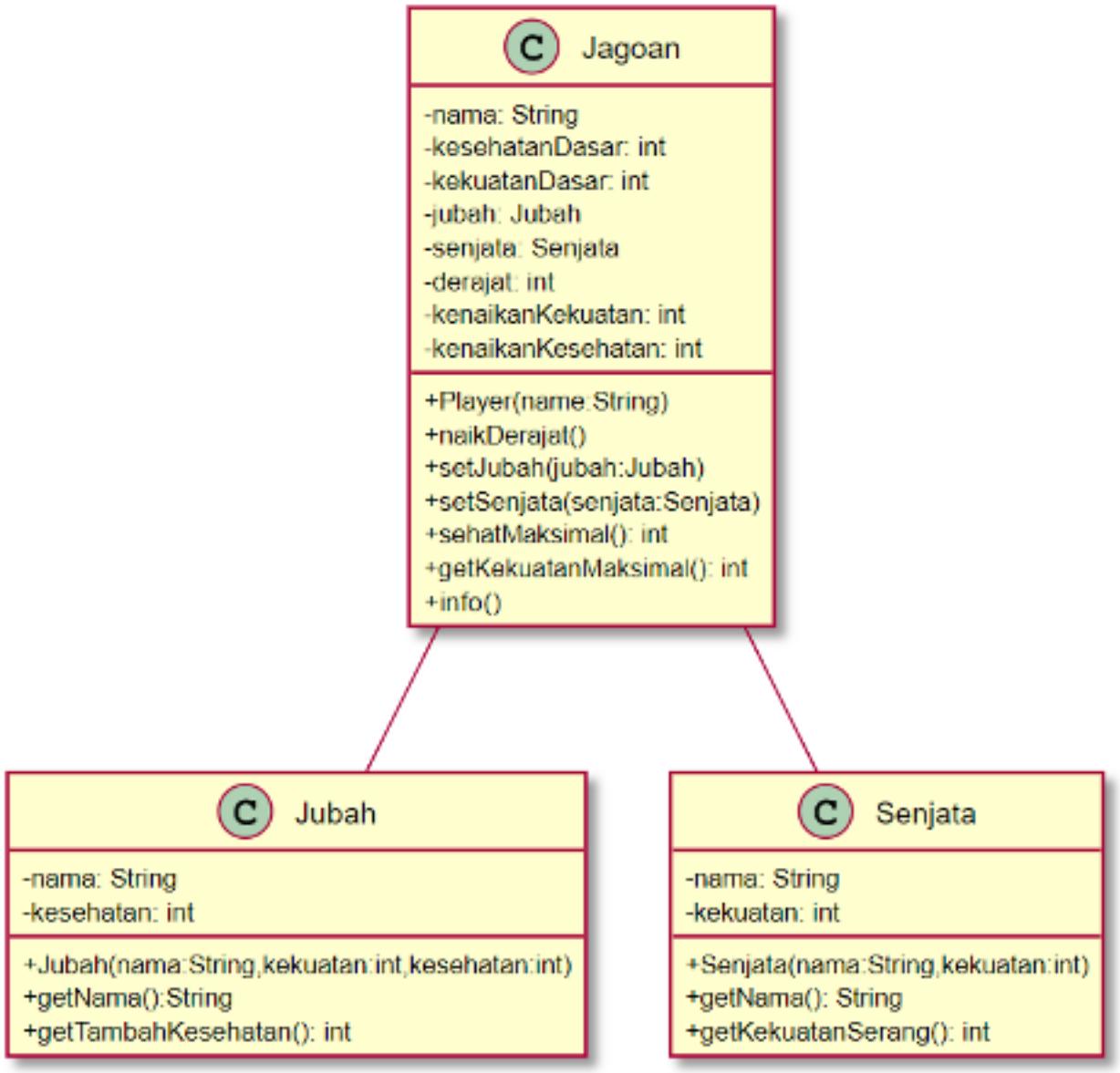
```
package app;

public class App {
    public static void main(String[] args) throws Exception {
        BangunDatar bangunDatar = new BangunDatar();
        Persegi persegi = new Persegi(4);
        Segitiga segitiga = new Segitiga(6, 3);
        Lingkaran lingkaran = new Lingkaran(50);

        // memanggil method luas dan keliling
        bangunDatar.luas();
        bangunDatar.keliling();
        System.out.print("\n");
        System.out.println("Luas persegi: " + persegi.luas());
        System.out.println("keliling persegi: " + persegi.keliling());
        System.out.print("\n");
        System.out.println("Luas segitiga: " + segitiga.luas());
        System.out.print("\n");
        System.out.println("Luas lingkaran: " + lingkaran.luas());
        System.out.println("keliling lingkaran: " + lingkaran.keliling());
    }
}
```

Output:

```
Menghitung luas bangun datar
Menghitung keliling bangun datar

Luas persegi: 16.0
keliling persegi: 16.0

Luas segitiga: 18.0

Luas lingkaran: 7853.9814
keliling lingkaran: 314.15927
```

Studi Kasus: Games Jagoan

Class Diagram



Pada Class Diagram di atas, terlihat class Jagoan, class Jubah, dan class Senjata. Biasa pada Games Pertempuran, seorang jagoan memiliki jubah dan senjata andalannya. Bisa ada banyak jagoan, banyak jubah, dan banyak senjata dalam suatu pertempuran.

Berikut adalah Source Code

Jagoan.java

```java
package app;

class Jagoan {
    private String nama;
    private int kesehatanDasar;
    private int kekuatanDasar;
    private Jubah jubah; // asosiasi
    private Senjata senjata; // asosiasi
    private int derajat;
    // private int kenaikanKekuatan;
    // private int kenaikanKesehatan;

    public Jagoan(String nama) {
        this.nama = nama;
        this.kesehatanDasar = 100;
        this.kekuatanDasar = 100;
        this.derajat = 1;
        // this.kenaikanKekuatan = 20;
        // this.kenaikanKesehatan = 20;
    }

    public void naikDerajat() {
        this.derajat++;
    }

    public void setJubah(Jubah jubah) {
        this.jubah = jubah;
    }

    public void setSenjata(Senjata senjata) {
        this.senjata = senjata;
    }

    public int sehatMaksimal() {
        return this.kesehatanDasar + this.jubah.getTambahKesehatan();
    }

    public int getKekuatanMaksimal() {
        return this.kekuatanDasar + this.senjata.getKekuatanSerang();
    }
```

```
    public void info() {
        System.out.println("Jagoan\t\t\t: " + this.nama);
        System.out.println("Derajat\t\t\t: " + this.derajat);
        System.out.println("Kesehatan Dasar\t\t: " + this.kesehatanDasar);
        System.out.println("Kekuatan Dasar\t\t: " + this.kekuatanDasar);
        System.out.println("Jubah\t\t\t: " + this.jubah.getNama());
        System.out.println("Senjata\t\t\t: " + this.senjata.getNama());
        System.out.println("Kesehatan Maksimal\t: " +
this.sehatMaksimal());
        System.out.println("Kekuatan Maksimal\t: " +
this.getKekuatanMaksimal() + "\n");
    }
}
```

Jubah.java

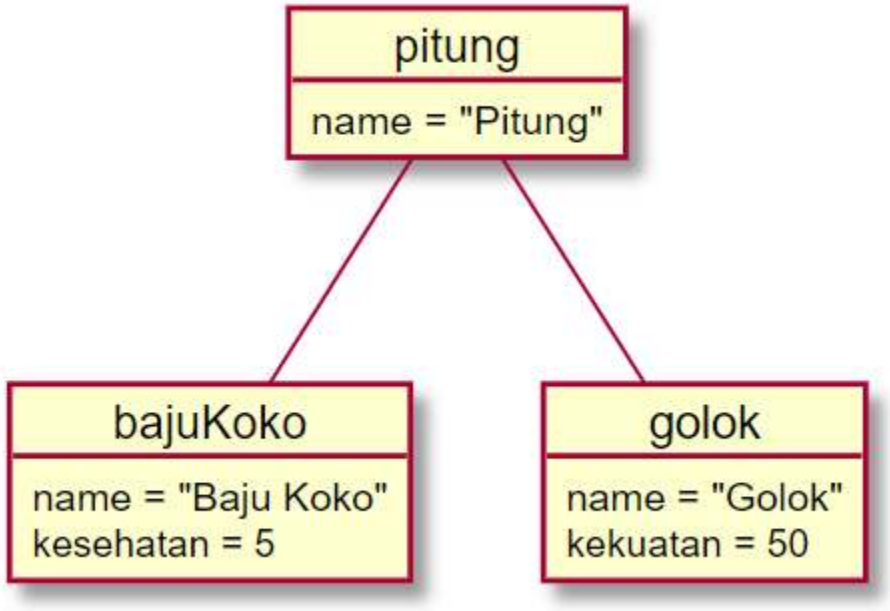
```
package app;

class Jubah {
    private String nama;
    private int kesehatan;

    public Jubah(String nama, int kesehatan) {
        this.nama = nama;
        this.kesehatan = kesehatan;
    }

    public String getNama() {
        return this.nama;
    }

    public int getTambahKesehatan() {
        return this.kesehatan * 10;
    }
}
```

Senjata.java

```java
package app;

class Senjata {
    private String nama;
    private int kekuatan;

    public Senjata(String nama, int kekuatan) {
        this.nama = nama;
        this.kekuatan = kekuatan;
    }

    public String getNama() {
        return this.nama;
    }

    public int getKekuatanSerang() {
        return this.kekuatan * 2;
    }
}
```

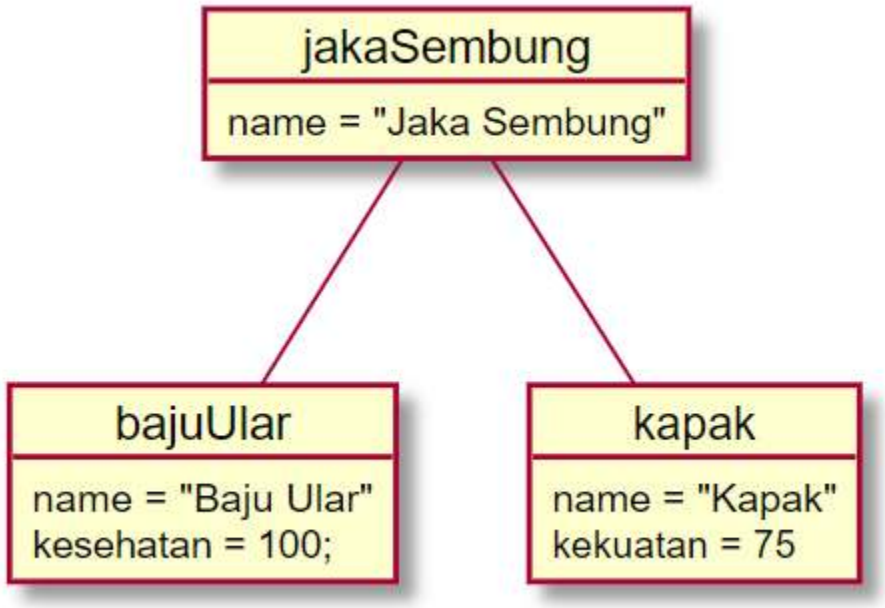
Object Diagram UML

Pitung



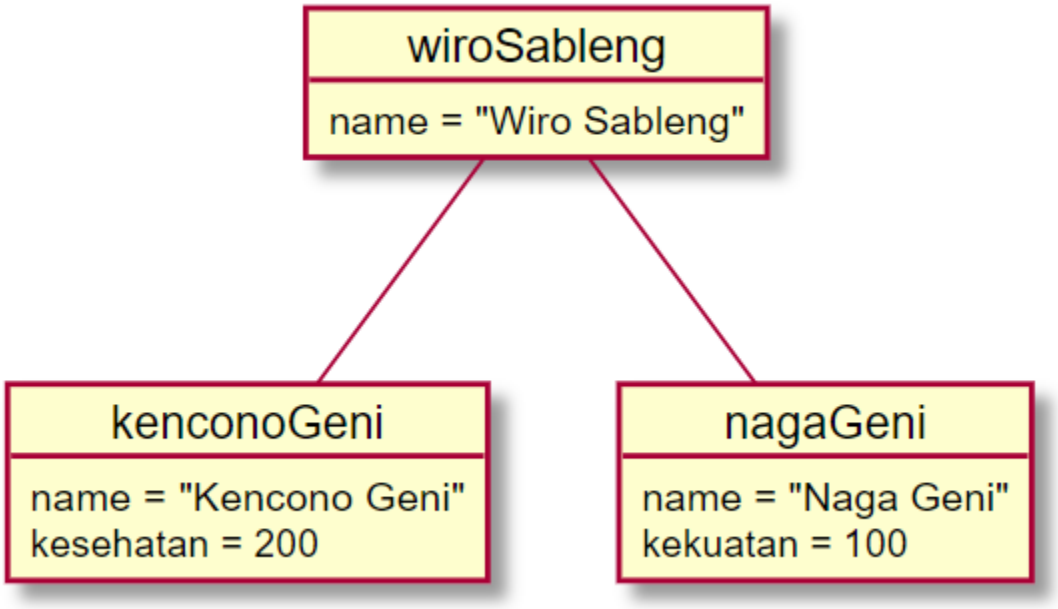

```
public class App {
    public static void main(String[] args) {
        Jagoan pitung = new Jagoan("Pitung");
        Jubah bajuKoko = new Jubah("Baju Koko", 5);
        Senjata golok = new Senjata("Golok", 50);

        pitung.setJubah(bajuKoko);
        pitung.setSenjata(golok);

        pitung.info();
    }
}
```

Jaka Sembung




```
public static void main(String[] args) {
        Jagoan jakaSembung = new Jagoan("Jaka Sembung");
        Jubah bajuUlar = new Jubah("Baju Ular", 100);
        Senjata kapak = new Senjata("Kapak", 75);

        jakaSembung.setJubah(bajuUlar);
        jakaSembung.setSenjata(kapak);

        jakaSembung.info();
}
```

Wiro Sableng




```
public static void main(String[] args) {
        Jagoan wiroSableng = new Jagoan("Wiro Sableng");
        Jubah kenconoGeni = new Jubah("Kencono Geni", 200);
        Senjata nagaGeni = new Senjata("Kapak Naga Geni", 100);

        wiroSableng.setJubah(kenconoGeni);
        wiroSableng.setSenjata(nagaGeni);

        wiroSableng.info();
}
```

PERTEMUAN KELIMA

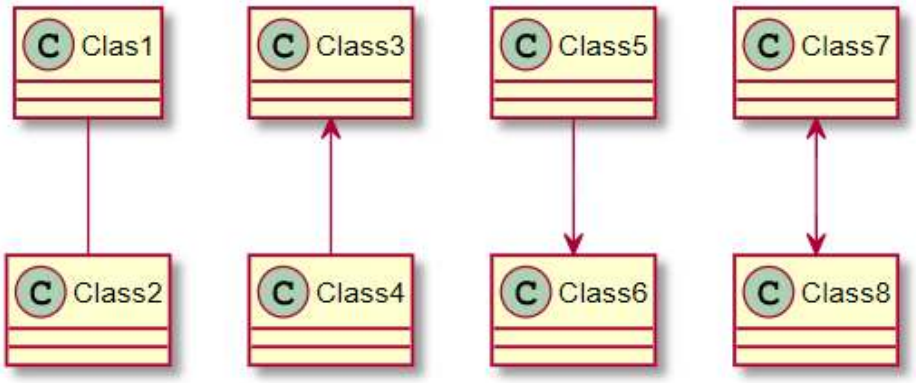
Association, Aggregation, and Composition

Prepared by Adi Wahyu Pribadi

Pengenalan

Pada konsep pemrograman berbasis objek, hubungan antar kelas terdapat empat yaitu pewarisan, asosiasi, agregasi, dan komposisi. Pewarisan telah dipelajari pada pertemuan sebelumnya. Materi kali ini akan fokus pada asosiasi, agregasi, dan komposisi.

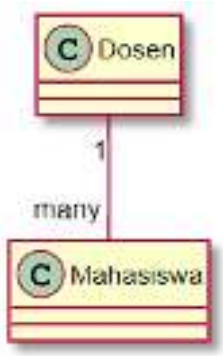
Association/Asosiasi



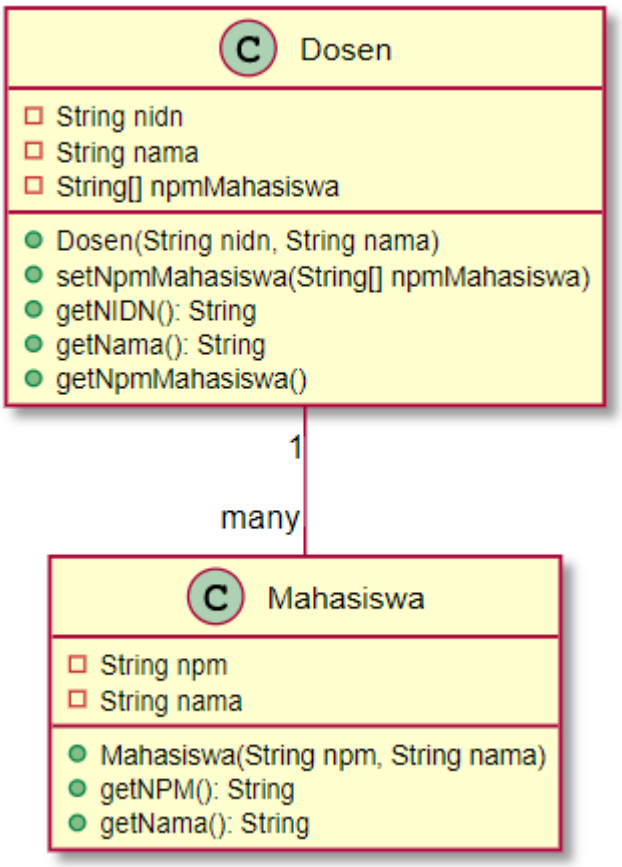
Pada gambar di atas, terlihat hubungan asosiasi antar Clas. Relasi assosiasi biasa disebut is “a” relationship. Assosiasi berarti bahwa sebuah object “menggunakan” object yang lain.

Assosiasi adalah sebuah relasi dimana semua object memiliki lifecycle nya sendiri dan tidak ada yang bertindak sebagai owner.

Pada gambar class diagram di samping, terdapat hubungan asosiasi antara Class Dosen dan Class Mahasiswa. Hubungan keduanya terlihat dari garis lurus tersambung antara Class Dosen dan Class Mahasiswa. Terdapat keterangan kardinalitas yang menyatakan bahwa Class Dosen dapat memiliki banyak Class Mahasiswa.



Penjelasan Class Diagram di samping terlihat pada Source Code berikut.



Mahasiswa.java

```
class Mahasiswa {
    private String npm;
```

```
    private String nama;

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

    public String getNPM() { return this.npm; }

    public String getNama() { return this.nama; }
}
```

Dosen.java

```
class Dosen {
    private String nidn;
    private String nama;
    private String[] npmMahasiswa;

    public Dosen(String nidn, String nama) {
        this.nidn = nidn;
        this.nama = nama;
    }

    public void setNpmMahasiswa (String[] npmMahasiswa) {
        this.npmMahasiswa = npmMahasiswa;
    }

    // menampilkan npm-npm mahasiswa-nya Dosen
    public void getNpmMahasiswa() {
        // mendapatkan panjang Array
        int n = this.npmMahasiswa.length;
        int i = 0;  // counter
        while (i < n) {
            System.out.println("Peserta ke-" + (i+1) +": " +
this.npmMahasiswa[i]);
            i++;
        }
    }

    public String getNama() {
```

```
        return this.nama;
    }

    public String getNIDN() {
        return this.nidn;
    }
}
```

Implementasi App.java

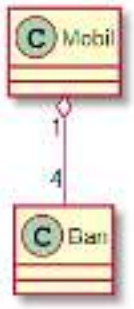
```
public class App {
    public static void main(String[] args) throws Exception {
        Dosen amir = new Dosen("12345", "Amir Murtako");
        Dosen bambang = new Dosen("12346", "Bambang Riono");

        Mahasiswa rina = new Mahasiswa("123", "Rina");
        Mahasiswa rano = new Mahasiswa("124", "Rano");
        Mahasiswa doel = new Mahasiswa("125", "Doel");
        Mahasiswa jajang = new Mahasiswa("126", "Jajang");

        String[] PAPakAmir = {doel.getNPM(), jajang.getNPM()};
        String[] PAMasBambang = {rina.getNPM(), rano.getNPM()};

        amir.setNpmMahasiswa(PAPakAmir);
        bambang.setNpmMahasiswa(PAMasBambang);

        System.out.println("Pak " + amir.getNama() + " dengan NIDN" +
amir.getNIDN());
        amir.getNpmMahasiswa();
        System.out.println("Mas " + bambang.getNama() + " dengan NIDN" +
bambang.getNIDN());
        bambang.getNpmMahasiswa();
    }
}
```

Output

```
Pak Amir Murtako dengan NIDN12345
Peserta ke-1: 125
Peserta ke-2: 126
Mas Bambang Riono dengan NIDN12346
Peserta ke-1: 123
Peserta ke-2: 124
```

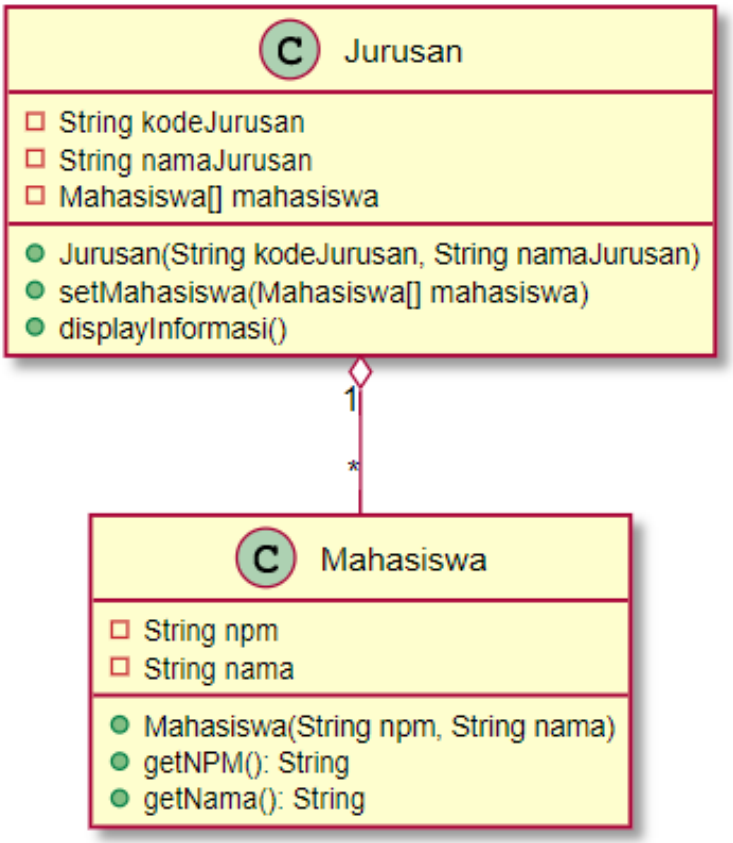
Aggregation/Agregasi

Agregasi adalah bentuk hubungan yang lebih khusus dari Assosiasi dimana sebuah object memiliki lifecycle nya sendiri tapi dengan kepemilikan. Relasinya biasa di sebut relasi “has-a”. Hubungan agregasi digambarkan dengan diamond putih, yang ditempelkan pada kelas yang memiliki, tidak dibubuhkan panah pada ujung yang tidak memiliki simbol diamond putih. Kemudian juga dibubuhkan kardinalitas seperti pada hubungan asosiasi.



Gambar Class Diagram di samping menunjukkan bahwa Class Mobil memiliki hubungan agregasi dengan Class Ban. Class Mobil memiliki 4 Class Ban. Ketika Class Mobil dihancurkan, Class Ban masih dapat berdiri sendiri.

Contoh lain adalah hubungan agregasi antara Class Jurusan dan Class Mahasiswa seperti pada gambar di bawah:



Jurusan.java

```
class Jurusan {
    private String kodeJurusan;
    private String namaJurusan;
    private Mahasiswa[] mahasiswa;

    public Jurusan(String kodeJurusan, String namaJurusan) {
        this.kodeJurusan = kodeJurusan;
        this.namaJurusan = namaJurusan;
    }

    public void setMahasiswa(Mahasiswa[] mahasiswas) {
        this.mahasiswa = mahasiswas;
    }

    public void displayInformasi() {
        System.out.println("Kode Jurusan" + this.kodeJurusan);
        System.out.println("JURUSAN: " + this.namaJurusan);
        System.out.println("Daftar Mahasiswa");
        // hitung jumlah mahasiswa
        int n = this.mahasiswa.length;
        int i = 0; // counter

        while (i < n) {
            System.out.println(mahasiswa[i].getNPM() + ": " +
mahasiswa[i].getNama());
            i++;
        }
    }
}
```

Implementasi App.java

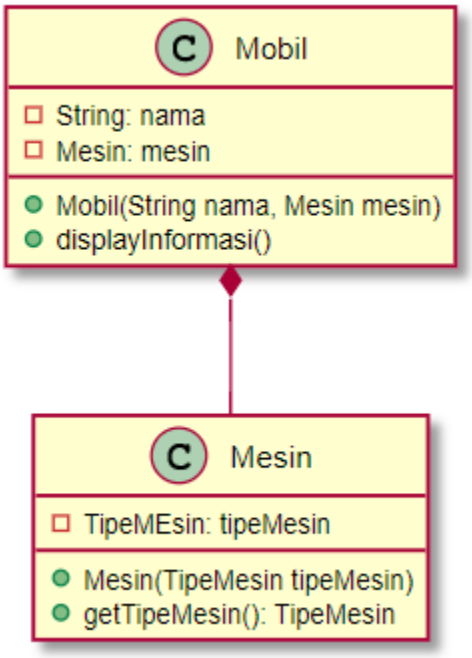
```
public class App {
    public static void main(String[] args) throws Exception {
        Mahasiswa rina = new Mahasiswa("123", "Rina");
        Mahasiswa rano = new Mahasiswa("124", "Rano");
        Mahasiswa doel = new Mahasiswa("125", "Doel");
        Mahasiswa jajang = new Mahasiswa("126", "Jajang");

        Jurusan informatika = new Jurusan("45", "Teknik Informatika");
        Jurusan elektro = new Jurusan("41", "Teknik Elektro");

        Mahasiswa[] mahasiswaInformatika = {doel, jajang};
        Mahasiswa[] mahasiswaElektro = {rina, rano};

  informatika.setMahasiswa(mahasiswaInformatika);
        elektro.setMahasiswa(mahasiswaElektro);

        informatika.displayInformasi();
        elektro.displayInformasi();
    }
}
```

Output

```
Kode Jurusan45
JURUSAN: Teknik Informatika
Daftar Mahasiswa
125: Doel
126: Jajang
Kode Jurusan41
JURUSAN: Teknik Elektro
Daftar Mahasiswa
123: Rina
124: Rano
```

Composition/Komposisi

Komposisi digambarkan menggunakan diamond penuh yang menyatakan memiliki bagian seperti pada agregasi. Pada komposisi lifecycle object bergantung pada object ownernya.



Pada gambar Class Diagram di atas, terlihat bahwa terdapat hubungan Komposisi antara Mobil dan Mesin. Ketika Mobil dihancurkan, Mesin juga ikut hancur.

TipeMesin.java

```
public enum TipeMesin {
    BENSIN,
    DIESEL
}
```

Mesin.java

```
class Mesin {
    private final TipeMesin tipeMesin;

    public Mesin(TipeMesin tipeMesin) {
        this.tipeMesin = tipeMesin;
    }

    public TipeMesin getTipeMesin() {
        return this.tipeMesin;
    }
}
```

Mobil.java

```
class Mobil {
    private String nama;
    private final Mesin mesin;

    public Mobil(String nama, Mesin mesin) {
        this.nama = nama;
        this.mesin = mesin;
    }

    public void displayInformasi() {
        System.out.println("Mobil " + this.nama);
        System.out.println("Mesinnya " +
this.mesin.getTipeMesin().toString());
    }
}
```

Implementasi App.java

```
public class App {
    public static void main(String[] args) throws Exception {
        Mobil avanza = new Mobil("Avanza", new Mesin(TipeMesin.BENSIN));
        Mobil pajero = new Mobil("Pajero", new Mesin(TipeMesin.DIESEL));

        avanza.displayInformasi();
        pajero.displayInformasi();
    }
}
```

PERTEMUAN KEENAM

Abstract dan Interface

Prepared by Adi Wahyu Pribadi

Abstract

Class abstract adalah class yang masih dalam bentuk abstract. Karena bentuknya masih abstract, dia tidak bisa dibuat langsung menjadi objek. Sebuah class agar dapat disebut class abstract setidaknya memiliki satu atau lebih method abstract. Objek hanya bisa dibuat dari non-abstract class maka suatu abstract class haruslah diturunkan dimana pada suatu subclass tersebut berisi implementasi dari abstract method yang ada pada Super/Parent Class-nya.

Method abstract adalah method yang tidak memiliki implementasi atau tidak ada bentuk konkritnya (hanya deklarasi method).

Mengapa Menggunakan Abstract?

Class abstrak belum bisa digunakan secara langsung. Karena itu, agar class abstrak dapat digunakan, maka ia harus dibuat bentuk konkritnya. Cara membuat class abstrak menjadi konkrit adalah dengan membuat implementasi dari method-method yang masih abstrak. Ini bisa kita lakukan dengan pewarisan (inheritance).

Class abstrak biasanya digunakan sebagai class induk dari class-class yang lain. Class anak akan membuat versi konkrit dari class abstrak.

Mengapa sih class harus dibuat menjadi abstrak? Pada suatu kondisi tertentu, class induk tidak ingin kita buat sebagai objek. Bisa jadi kode methodnya belum jelas mau bagaimana implementasinya.

Sebagai perumpamaan terdapat class Database yang digunakan untuk keperluan koneksi dan query database. Implementasi dari class Database tersebut adalah sesuai dari aplikasi database yang digunakan seperti MySQL, PostgreSQL, dan lain sebagainya.

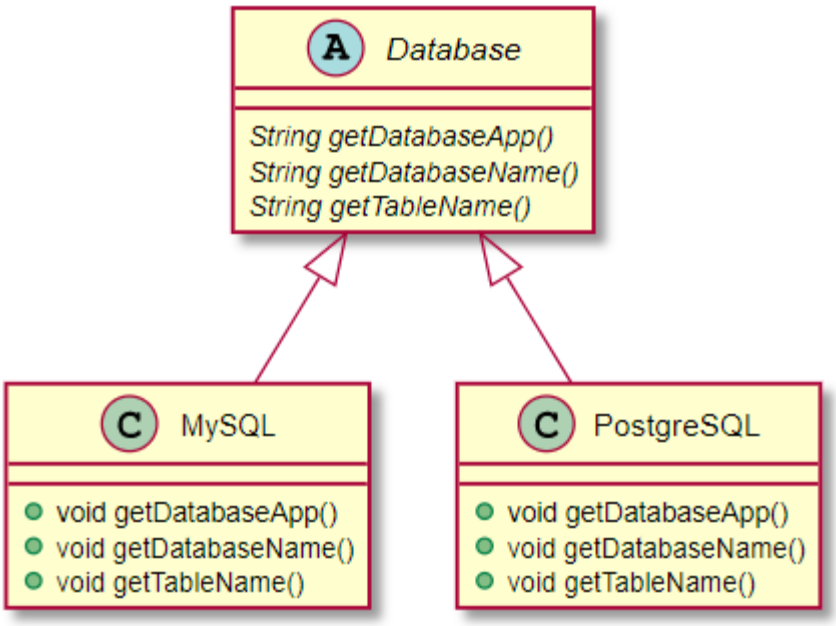
```
class Database {
    String getTableName() {
        return null;
    }
}

class MySQL extends Database {
    @Override
    String getTableName() {
        return "SELECT table_name FROM DATABASE";
    }
}

public class App {
    public static void main(String[] args) throws Exception {
        Database db = new Database();
        System.out.println(db.getTableName());
        MySQL mysql = new MySQL();
        System.out.println(mysql.getTableName());
    }
}
```

Output

```
null
SELECT table_name FROM DATABASE
```

Dari output terlihat bahwa, jika membuat objek dari class Database, belum ada implementasi bagaimana cara mendapatkan nama tabel. Namun pada class MySQL yang merupakan implementasi class Database, baru bisa mendapatkan nama tabel.

Solusinya adalah dengan membuat abstract class Database, sehingga ketika akan menggunakan database harus memilih implementasi dari aplikasi Databasenya seperti MySQL atau PostgreSQL. Lihat Class Diagram berikut. Abstract class terdapat huruf A besar dan nama abstract class ditulis miring, begitu juga dengan abstract method ditulis miring.



Implementasinya adalah:

Abstract Class Database

```
abstract class Database {
    abstract String getDatabaseApp();
    abstract String getDatabaseName();
    abstract String getTableName();
}
```

Class MySQL

```
class MySQL extends Database {
    String getDatabaseApp() {
        return "MySQL DATABASE Server";
    }
    String getDatabaseName() {
        return "SIAK";
    }
    String getTableName() {
        return "Table Mahasiswa";
    }
```

```
}
```

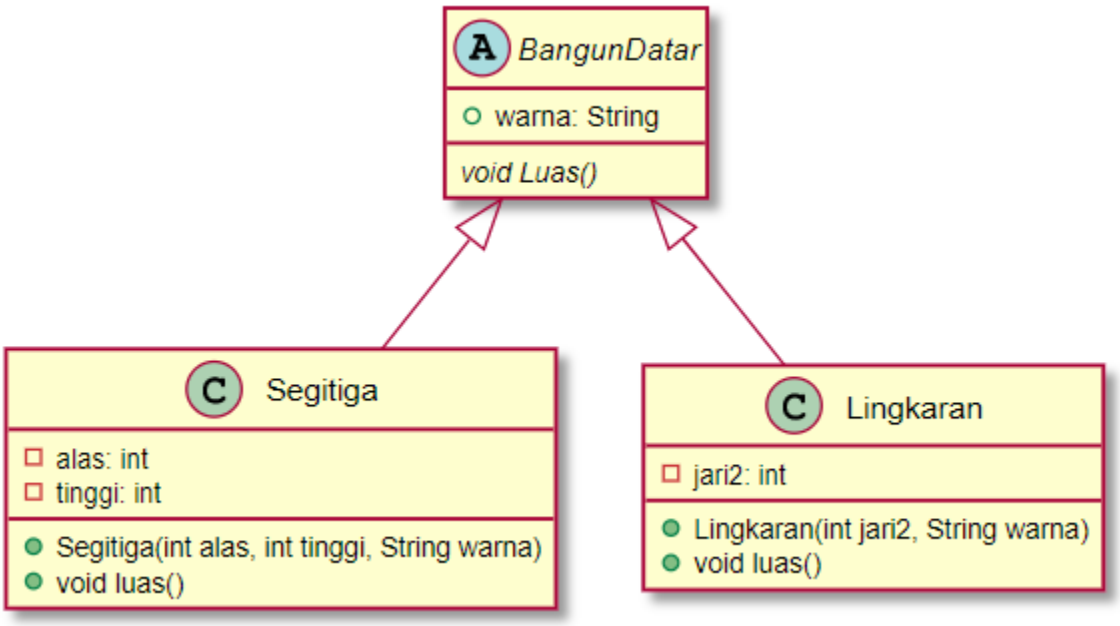
Class PostgreSQL

```
class PostgreSQL extends Database {
    String getDatabaseApp() {
        return "PostgreSQL DATABASE Server";
    }
    String getDatabaseName() {
        return "Rumah Sakit";
    }
    String getTableName() {
        return "Table Pasien";
    }
}
```

Implementasi App

```
public class App {
    public static void main(String[] args) throws Exception {
        MySQL mysql = new MySQL();
        System.out.println(mysql.getDatabaseApp());
        System.out.println(mysql.getDatabaseName());
        System.out.println(mysql.getTableName());

        PostgreSQL psql = new PostgreSQL();
        System.out.println(psql.getDatabaseApp());
        System.out.println(psql.getDatabaseName());
        System.out.println(psql.getTableName());
    }
}
```

Contoh lain: BangunDatar



Implementasinya adalah

Abstract Class BangunDatar

```
abstract class BangunDatar {
    public String warna;
    abstract void luas();
}
```

Class Segitiga

```
class Segitiga extends BangunDatar {
    private int alas;
    private int tinggi;

    public Segitiga(int alas, int tinggi, String warna) {
        this.alas = alas;
        this.tinggi = tinggi;
        this.warna = warna;
    }
```

```
    public void luas() {
        float l = (float) (this.alas * this.tinggi) / 2;
        System.out.println("Luas Segitiga: " + l);
    }
}
```

Class Lingkaran

```
class Lingkaran extends BangunDatar {
    private int jari2;

    public Lingkaran(int jari2, String warna) {
        this.jari2 = jari2;
        this.warna = warna;
    }

    public void luas() {
        float l = (float) (this.jari2 * this.jari2 * (22.0/7.0));
        System.out.println("Luas Lingkaran: " + l);
    }
}
```

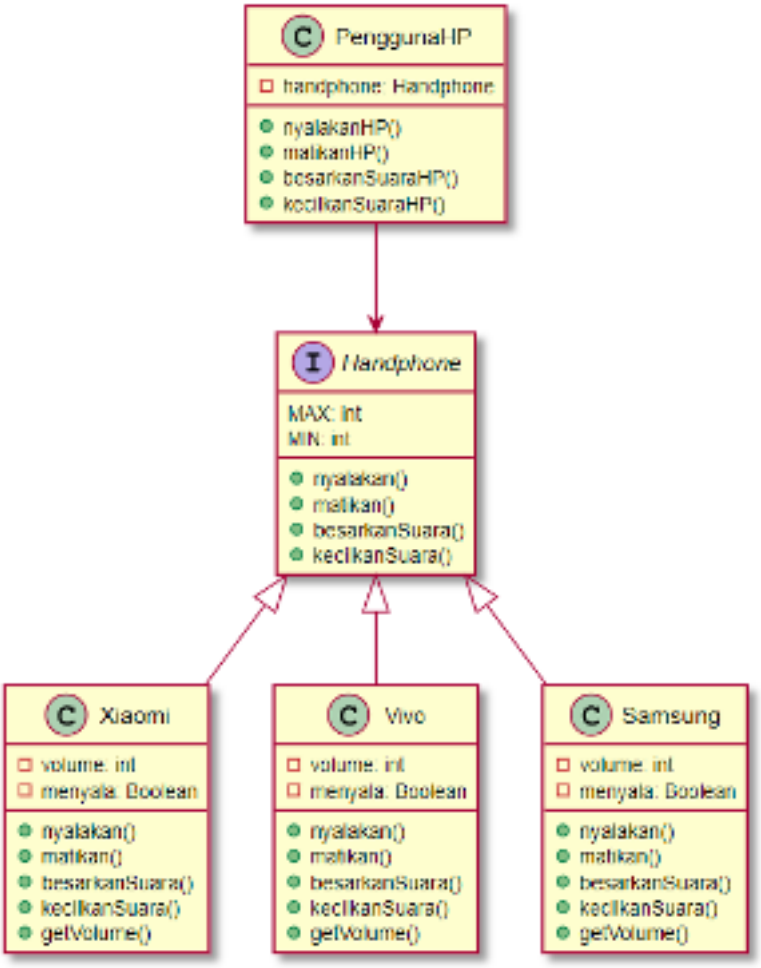
Implementasi App

```
public class App {
    public static void main(String[] args) throws Exception {
        Segitiga s1 = new Segitiga(10, 6, "Merah");
        Lingkaran l1 = new Lingkaran(10, "Biru");

        s1.luas();
        l1.luas();
    }
}
```

Interface

Interface secara harfiah adalah antar muka. Secara umum, interface berfungsi sebagai penghubung antara sesuatu yang ‘abstrak’ dengan sesuatu yang nyata. Seperti sebuah kelas, interface dapat memiliki metode dan variabel, tetapi method yang dideklarasikan pada interface hanya nama method tanpa body.

Sebagai contoh Interface adalah tombol on/off dan tombol up/down volume suara pada handphone. Cukup tekan tombol on/off untuk menyalakan atau mematikan handphone dan menekan tombol up untuk mengeraskan suara dan tombol down untuk mengecilkan suara. Sebagai pengguna, kita tidak mau tahu bagaimana caranya handphone bisa menyala, mati, volume membesar maupun mengecil.



Class Diagram adalah contoh bagaimana implementasi Interface Handphone. Pengguna HP pasti memiliki salah satu dari Xiaomi, Vivo, atau Samsung. Karena ketiga merek Handphone tersebut merupakan implementasi dari Handphone, maka ketiganya memiliki method yang sama namun bisa saja implementasi di dalam kodenya berbeda-beda.

Implementasinya adalah:

PenggunaHP

```
public class PenggunaHP {
    private Handphone phone;

    public PenggunaHP(Handphone phone) {
        this.phone = phone;
    }

    void nyalakanHP(){
        this.phone.nyalakan();
    }

    void matikanHP(){
        this.phone.matikan();
    }

    void besarkanSuaraHP(){
        this.phone.besarkanSuara();
    }

    void kecilkanSuaraHP(){
        this.phone.kecilkanSuara();
    }
}
```

Handphone

```
public interface Handphone {
    int MAX_VOLUME = 100;
    int MIN_VOLUME = 0;

    void nyalakan();
    void matikan();
    void besarkanSuara();
    void kecilkanSuara();
}
```

Xiaomi

```
public class Xiaomi implements Handphone {

    private int volume;
    private boolean menyala;

    public Xiaomi() {
        // set volume awal
        this.volume = 50;
    }

    @Override
    public void nyalakan() {
        menyala = true;
        System.out.println("Handphone menyala...");
        System.out.println("Selamat datang di XIAOMI");
        System.out.println("Android version 10");
    }

    @Override
    public void matikan() {
        menyala = false;
        System.out.println("Handphone dimatikan");
    }

    @Override
    public void besarkanSuara() {
```

```
        if (menyala) {
            if (this.volume == MAX_VOLUME) {
                System.out.println("Volume FULL!!");
                System.out.println("sudah " + this.getVolume() + "%");
            } else {
                this.volume += 10;
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }

    @Override
    public void kecilkanSuara() {
        if (menyala) {
            if (this.volume == MIN_VOLUME) {
                System.out.println("Volume = 0%");
            } else {
                this.volume -= 10;
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }

    public int getVolume() {
        return this.volume;
    }
}
```

Vivo

```
public class Vivo implements Handphone {
    private int volume;
    private boolean menyala;

    public Vivo() {
        // set volume awal
        this.volume = 50;
```

```
}

@Override
public void nyalakan() {
    menyala = true;
    System.out.println("Handphone menyala...");
    System.out.println("Selamat datang di Vivo");
    System.out.println("Android version 10");
}

@Override
public void matikan() {
    menyala = false;
    System.out.println("Handphone dimatikan");
}

@Override
public void besarkanSuara() {
    if (menyala) {
        if (this.volume == MAX_VOLUME) {
            System.out.println("Volume FULL!!");
            System.out.println("sudah " + this.getVolume() + "%");
        } else {
            this.volume += 10;
            System.out.println("Volume sekarang: " + this.getVolume());
        }
    } else {
        System.out.println("Nyalakan dulu donk HP-nya!!");
    }
}

@Override
public void kecilkanSuara() {
    if (menyala) {
        if (this.volume == MIN_VOLUME) {
            System.out.println("Volume = 0%");
        } else {
            this.volume -= 10;
            System.out.println("Volume sekarang: " + this.getVolume());
        }
    } else {
        System.out.println("Nyalakan dulu donk HP-nya!!");
```

```
        }
    }

    public int getVolume() {
        return this.volume;
    }
}
```

Samsung

```
public class Samsung implements Handphone {
    private int volume;
    private boolean menyala;

    public Samsung() {
        // set volume awal
        this.volume = 50;
    }

    @Override
    public void nyalakan() {
        menyala = true;
        System.out.println("Handphone menyala...");
        System.out.println("Selamat datang di SAMSUNG");
        System.out.println("Android version 11");
    }

    @Override
    public void matikan() {
        menyala = false;
        System.out.println("Handphone dimatikan");
    }

    @Override
    public void besarkanSuara() {
        if (menyala) {
            if (this.volume == MAX_VOLUME) {
                System.out.println("Volume FULL!!");
                System.out.println("sudah " + this.getVolume() + "%");
            } else {
                this.volume += 10;
```

```
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }


    @Override
    public void kecilkanSuara() {
        if (menyala) {
            if (this.volume == MIN_VOLUME) {
                System.out.println("Volume = 0%");
            } else {
                this.volume -= 10;
                System.out.println("Volume sekarang: " + this.getVolume());
            }
        } else {
            System.out.println("Nyalakan dulu donk HP-nya!!");
        }
    }

    public int getVolume() {
        return this.volume;
    }
}
```

Implementasi App.java

```
package app;
import java.util.Scanner; // library untuk input data, bisa dari Console

public class App {
    public static void main(String[] args) throws Exception {
        Handphone redmiNote8 = new Xiaomi();

        PenggunaHP dian = new PenggunaHP(redmiNote8);

        dian.nyalakanHP();

        Scanner input = new Scanner(System.in);
```

```
        String aksi;

        while (true) {
            System.out.println("=== APLIKASI INTERFACE ===");
            System.out.println("[1] Nyalakan HP");
            System.out.println("[2] Matikan HP");
            System.out.println("[3] Perbesar Volume");
            System.out.println("[4] Kecilkan Volume");
            System.out.println("[0] Keluar");
            System.out.println("--------------------------");
            System.out.print("Pilih aksi> ");
            aksi = input.nextLine();

            if(aksi.equalsIgnoreCase("1")){
                dian.nyalakanHP();
            } else if (aksi.equalsIgnoreCase("2")){
                dian.matikanHP();
            } else if (aksi.equalsIgnoreCase("3")){
                dian.besarkanSuaraHP();
            } else if (aksi.equalsIgnoreCase("4")){
                dian.kecilkanSuaraHP();
            } else if (aksi.equalsIgnoreCase("0")){
                input.close();
                System.exit(0);
            } else {
                System.out.println("Kamu memilih aksi yang salah!");
            }
        }
    }
}
```

Perbedaan Abstract dan Interface

- **Jenis method**: Interface hanya dapat memiliki metode abstrak. Abstract dapat memiliki metode abstract dan non-abstract. Sejak Java 8, ia dapat memiliki metode default dan statis juga.
- **Final Variables**: Variabel yang dideklarasikan dalam interface Java secara default adalah final. Class abstract dapat berisi variabel non-final.
- **Jenis variabel**: Class abstract dapat memiliki variabel final, non-final, statis, dan non-statis. Interface hanya memiliki variabel statis dan final.
- **Implementasi**: Abstract class dapat menyediakan implementasi interface. Interface tidak dapat memberikan implementasi abstract class.
- **Inheritance vs Abstraction**: Interface Java dapat diimplementasikan menggunakan kata kunci "implement" dan class abstract dapat diperpanjang menggunakan kata kunci "extends".
- **Multi Implementasi**: Intreface hanya dapat memperluas interface Java lainnya saja, class abstract dapat memperluas class Java lainnya dan mengimplementasikan beberapa antarmuka Java.
- **Aksesibilitas Data Members**: Members dari Java interface secara default ada public. Class abstract java bisa memiliki members dengan aksesibilitas seperti private, protected, etc.
- **Class abstract** kita bisa buat properti atau variabel sedangkan di **interface** kita cuma bisa buat konstanta saja.
- **Class abstract** kita bisa implementasikan kode method seperti class biasa, sedangkan di **interface** harus menggunakan default
- **Class abstract** dapat memiliki member private dan protected sedangkan **interface** harus publik semua
- **Class abstract** diimpelentasikan dengan pewarisan (extends) sedangkan **interaface** menggunakan implements

PERTEMUAN KESEMBILAN

Java Packages, API, Enum

Prepared by Adi Wahyu Pribadi

Pendahuluan

Package atau paket di Java digunakan untuk mengelompokkan class-class yang terkait. Package di Java terbagi menjadi dua macam:

- Package Built-in (Paket dari Java API)
- Package buatan sendiri

Built-in Package

Java API adalah library kumpulan Class yang sudah ditulis dan dapat langsung digunakan secara bebas. Java API terdapat dalam Java Development Environment. Library berisi komponen untuk mengelola input, pemrograman basis data, dan banyak lagi lainnya. Daftar lengkapnya dapat ditemukan di situs web Oracles:

https://docs.oracle.com/javase/8/docs/api/

https://docs.oracle.com/en/java/javase/11/docs/api/index.html

Library terbagi atas packages dan class. Berarti kita dapat mengimpor satu class (beserta metode dan atributnya), atau seluruh package yang berisi semua class yang ada di dalam package yang kita pilih.

Untuk menggunakan sebuah class atau package dari library, kita gunakan keyword **import**

Syntax

```
import package.name.Class;   // Import sebuah class
import package.name.*;       // Import seluruh package
```

Import sebuah class

```
import java.util.Scanner;
```

Pada baris di atas, nama package-nya adalah java.util dan nama class-nya adalah Scanner. Untuk menggunakan buat objek class dan gunakan salah satu metode yang tersedia yang ditemukan dalam dokumentasi class Scanner. Kita akan menggunakan method nextLine(), yang digunakan untuk membaca baris lengkap:

```
import java.util.Scanner;
class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Masukkan nama Anda");

    String userName = myObj.nextLine();
    System.out.println("Nama Anda adalah " + userName);
  }
}
```

Import Sebuah Package

Ada banyak paket untuk dipilih. Pada contoh sebelumnya, kita menggunakan class Scanner dari package java.util. Package ini juga berisi fungsi utilitas seperti tanggal dan waktu, pembuat nomor acak, dan class utilitas lainnya.

Untuk mengimpor seluruh package, akhiri kalimat dengan tanda bintang (*). Contoh berikut akan mengimpor semua class di package java.util:

```
import java.util.*;
```

Package Buatan Sendiri

Untuk membuat package sendiri, Java menggunakan sistem folder file untuk menyimpan, sama seperti folder di komputer. Misalkan:

```
|___ src
   |___ App
      |___ ClassSaya.java
```

Untuk menggunakan sebuah package gunakan keyword package;

```
package App;
class ClassSaya {
  public static void main(String[] args) {
    System.out.println("Ini sebuah package!");
  }
}
```

Simpan file di atas dengan nama ClassSaya.java di dalam folder src

Compile File ClassSaya.java dengan perintah

```
C:\Users\Adiwa\src> javac ClassSaya.java
```

Menjalankan file ClassSaya.class

```
C:\Users\Adiwa\src>java ClassSaya
Error: Could not find or load main class ClassSaya
Caused by: java.lang.NoClassDefFoundError: App/ClassSaya (wrong name:
ClassSaya)
```

Muncul Error dikarenakan ClassSaya terdapat package App

Maka gunakan perintah javac -d . ClassSaya.java

```
C:\Users\Adiwa\src>javac -d . ClassSaya.java
```

Dan untuk menjalankan ClassSaya adalah dengan perintah `java NamaPackage.NamaClass`

```
C:\Users\Adiwa\src>java App.ClassSaya
Ini sebuah package!
```

Java Enums

Enum adalah Class spesial yang merepresentasikan grup KONSTAN/CONSTANTS (variable yang tidak bisa diubah, seperti variabel final). Untuk membuath sebuah enum, gunakan keyword enum, dan pisahkan constant dengan tanda koma (,). Ingat, constant ditulis dengan HURUF BESAR.

```
enum Level {
   LOW, MEDIUM, HIGH
}
```

Untuk mengakses konstan di enum gunakan titik / dot syntax:

```
Level kemahiran = Level.MEDIUM;
Level rasaPedas = Level.MEDIUM;
Level rasaPedas = Level.HIGH;
```

Enum di dalam sebuah Class

```
Public class ClassKu {
   enum Level {
      LOW, MEDIUM, HIGH
   }
   Public static void main(String[] args) {
      Level kemahiran = Level.HIGH;
      System.out.println(kemahiran);
   }
}
```

Outputnya:

```
HIGH
```

Enum di dalam Statement Switch

```
enum Level {
  LOW, MEDIUM, HIGH
}

public class ClassKu {
  public static void main(String[] args) {
    Level kemahiran = Level.MEDIUM;

    switch(kemahiran) {
      case LOW:
        System.out.println("Anak bawang");
        break;
      case MEDIUM:
         System.out.println("Anggota");
        break;
      case HIGH:
        System.out.println("Suhu");
        break;
    }
  }
}
```

Outputnya:

```
Anggota
```

Looping di dalam Enum

Tipe enum memiliki sebuah method bernama values(), method tersebut akan mengembalikan array dari seluruh constant yang ada di dalam enum tersebut. Method ini bermanfaat ketika ingin menampilkan seluruh constant yang ada di enum.

```
for (Level kemahiran : Level.values()) {
  System.out.println(kemahiran);
}
```

Outputnya:

```
LOW
MEDIUM
HIGH
```

PERTEMUAN KEDELAPAN

Static Keyword, Static Method, Static Blocks

Prepared by Adi Wahyu Pribadi

Static Keyword

Sebagaimana yang telah diketahui bahwa untuk mengakses anggota Class, kita harus membuat instance/objek terlebih dahulu dari Class tersebut. Namun, terkadang terdapat situasi di mana, kita ingin mengakses anggota Class tanpa harus membuat objek terlebih dahulu.

Pada situasi tersebut, kita gunakan keyword static sebagai solusinya. Jadi kita harus mendeklarasikan anggota Class yang static (*declare the class members static*).

Sebagai contoh terdapat `Class Math` di Java yang hampis semua membernya static. Sehingga kita dapat mengaksesnya tapa harus membuat instances dari `Class Math`.

```
package app;

public class App {
    public static void main(String[] args) throws Exception {
        // accessing the methods of the Math class
        System.out.println("Absolute value of -12 =  " + Math.abs(-12));
        System.out.println("Value of PI = " + Math.PI);
        System.out.println("Value of E = " + Math.E);
        System.out.println("2^2 = " + Math.pow(2,2));
    }
}
```

Output:

```
Absolute value of -12 =  12
Value of PI = 3.141592653589793
Value of E = 2.718281828459045
2^2 = 4.0
```

Pada contoh di atas, kita tidak perlu membuat object apapun dari `Class Math`. Walau begitu kita mengakses method `abs()`, `pow()` dan variabel `PI` dan `E`.

Static Methods

Static Method adalah method yang dapat diakses langsung dari nama Classnya.

```
package app;
class StaticTest {

    // non-static method
    int multiply(int a, int b){
        return a * b;
    }

    // static method
    static int add(int a, int b){
        return a + b;
    }
}

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

        // create an instance of the StaticTest class
        StaticTest st = new StaticTest();

        // call the nonstatic method
        System.out.println(" 2 * 2 = " + st.multiply(2,2));

        // call the static method
        System.out.println(" 2 + 3 = " + StaticTest.add(2,3));
   }
}
```

Output:

```
 2 * 2 = 4
 2 + 3 = 5
```

Pada contoh di atas terdapat non-static method bernama `multiply()` dan static method `add` di dalam class StaticTest. Di class App, kita liat bahwa non-static method diakses melalu objek `st` menggunakan `st.multiply(2, 2)`. Sedangkan kita dapat langsung memanggil static method menggunakan `StaticTest.add(2, 3)` tanpa harus membuat objek.

Static Variabels

Static Varibles adalah variable yang dapat diakses langsung dari nama Classnya.

```
package app;
class Test {
   // static variable
   static int max = 10;
   // non-static variable
   int min = 5;
}

public class App {
   public static void main(String[] args) {
       Test obj = new Test();
       // access the non-static variable
       System.out.println("min + 1 = " + (obj.min + 1));
       // access the static variable
       System.out.println("max + 1 = " + (Test.max + 1));
   }
}
```

Output:

```
min + 1 = 6
max + 1 = 11
```

Pada contoh di atas, terdapat non-static variable `min` dan static variable `max` di dalam class `Test`. Di dalam App class, kita dapat memanggil non-static variable melalui objek `obj` dengan cara `obj.min + 1`. Sedangakn untuk memanggil static variable cukup gunakan nama class `Test.max + 1`.

Catatan: Static variables jarang digunakan di Java. Sebagai gantinya, digunakan static constants. Static constants didefinisikan dengan keyword static final dan ditulis dengan huruf besar. Inilah mengapa, static variables juga sering ditulis dalam huruf besar.

Mengakses Static Variables dan Methods di dalam Class

Pada contoh-contoh sebelumnya kita mengakses static variables dan static methods dari Class yang berbeda. Namun, mengakses static variables dan static method dari dalam Class juga bisa dilakukan secara langsung.

```
package app;
public class App {
   // static variable
   static int age;
   // static method
   static void display() {
       System.out.println("Static Method");
   }
   public static void main(String[] args) {

       // access the static variable
       age = 30;
       System.out.println("Age is " + age);

       // access the static method
       display();
   }
}
```

Output:

```
Age is 30
Static Method
```

Pada contoh di atas, kita dapat mengakses static variable dan static method langsung tanpa menggunakan nama Class. Static variable dan static method defaultnya adalah public. Karena kita mengakses dari Class yang sama, kita tidak perlu menyebutkan nama Class-nya.

Static Blocks

Static blocks digunakan untuk inisiasi static variable.

```
package app;
class App {
   // static variables
   static int a = 23;
   static int b;
   static int max;
   // static blocks
   static {
       System.out.println("First Static block.");
       b = a * 4;
   }
   static {
       System.out.println("Second Static block.");
       max = 30;
   }
   // static method
   static void displaystatic() {
       System.out.println("a = " + a);
       System.out.println("b = " + b);
       System.out.println("max = " + max);
   }

   public static void main(String args[]) {
       // calling the static method
       displaystatic();
   }
}
```

Output:

```
First Static block.
Second Static block.
a = 23
b = 92
max = 30
```

PERTEMUAN KESEMBILAN

Nested Class, Inner Class, Nested Anonymous Class

Prepared by Adi Wahyu Pribadi

Java Nested Class dan Inner Class

Di Java, kita dapat mendeklarasikan Class di dalam Class, yang biasa dinamakan Nested Class. Terdapat dua macam nested Class:

- Static Nested Class
- Non-static Nested Class

Non-Static Nested Class (Inner Class)

Non-static nested class adalah class di dalam class. Class tersebut memiliki akses ke anggota dari class luarnya. Non-static nested class biasanya disebut sebagai `inner Class`.

```
package app;

class CPU {
    double price;
    // nested class
    class Processor{
        // members of nested class
        double cores;
        String manufacturer;

        double getCache(){
            return 4.3;
        }
    }

    // nested protected class
    protected class RAM{
        // members of protected nested class
        double memory;
```

```
        String manufacturer;

        double getClockSpeed(){
            return 5.5;
        }
    }
}

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

        // create object of Outer class CPU
        CPU cpu = new CPU();

       // create an object of inner class Processor using outer class
        CPU.Processor processor = cpu.new Processor();

        // create an object of inner class RAM using outer class CPU
        CPU.RAM ram = cpu.new RAM();
        System.out.println("Processor Cache = " + processor.getCache());
        System.out.println("Ram Clock speed = " + ram.getClockSpeed());
    }
}
```

Output:

```
Processor Cache = 4.3
Ram Clock speed = 5.5
```

Pada program di atas, terdapat dua nested Class: Processor dan RAM di dalam outer Class: CPU. Kita dapat mendeklarasikan inner Class sebagai protected. Sehingga kita dapat mendeklarasikan class RAM sebagai protected.

Di dalam App class

- Kita membuat instance dari outer class CPU dinamakan cpu.
- Menggunakan instance dari outer class, kemudian kita membuat objek dari inner Class:

```
CPU.Processor processor = cpu.new Processor;
CPU.RAM ram = cpu.new RAM();
```

Catatan: Kita gunakan operator dot/titik (.) untuk membuat instance dari inner Class menggunakan outer class.

Mengakses Member Outer Class dari Inner Class

Kita dapat mengakses members dari outer Class menggunakan keyword `this`.

```
package app;
class Car {
    String carName;
    String carType;
    // assign values using constructor
    public Car(String name, String type) {
        this.carName = name;
        this.carType = type;
    }
    // private method
    private String getCarName() {
        return this.carName;
    }

    // inner class
    class Engine {
        String engineType;
        void setEngine() {

           // Accessing the carType property of Car
            if(Car.this.carType.equals("4WD")){

                // Invoking method getCarName() of Car
                if(Car.this.getCarName().equals("Crysler")) {
                    this.engineType = "Bigger";
                } else {
                    this.engineType = "Smaller";
                }

            }else{
                this.engineType = "Bigger";
```

```
            }
        }
        String getEngineType(){
            return this.engineType;
        }
    }
}
public class App {
    public static void main(String[] args) {

        // create an object of the outer class Car
        Car car1 = new Car("Mazda", "8WD");

        // create an object of inner class using the outer class
        Car.Engine engine = car1.new Engine();
        engine.setEngine();
        System.out.println("Engine Type for 8WD= " +
engine.getEngineType());

        Car car2 = new Car("Crysler", "4WD");
        Car.Engine c2engine = car2.new Engine();
        c2engine.setEngine();
        System.out.println("Engine Type for 4WD = " +
c2engine.getEngineType());
    }
}
```

Output:

```
Engine Type for 8WD= Bigger
Engine Type for 4WD = Smaller
```

Pada contoh di atas, kita memiliki Class `Engine` sebagai inner Class dari Outer Class `Car`. Sehingga pada baris berikut, digunakan keyword `this` untuk mengakses variable carType dari Outer Class.

```
if (Car.this.carType.equals("4WD") ) { ... }
```

Kita juga dapat mengakses method dari outer Class dari dalam inner Class

```
if (Car.this.getCarName().equals("Crysler") ) { .. }
```

Catatan

- Java memperlakukan inner class sama seperti anggota class biasa.
- Sejak inner class adalah member dari outer class, kita dapat mengakses modifiers seperti private, protected ke dalam inner class dimana tidak mungkin dikerjakan pada class normal.
- Sejak nested class anggota dari outer class, kita gunakan notasi titik/dot (.) untuk mengakses nested class dan anggotanya.
- Menggunakan nested class akan membuat kode lebih mudah dibaca dan lebih baik dalam pembungkusan (paradigma *encapsulation*).
- Non-static nested class (inner class) memiliki akses ke anggota dari outer class, walaupun mereka dideklarasikan sebagai private.

Nested Static Class

Di Java, kita juga dapat mendefinisikan Static class di dalam sebuah Class. Class tersebut dinamakan static nested class. Static nested classes TIDAK DINAMAKAN static inner classes.

Tidak seperti inner Class, static nested class tidak dapat mengakses variable dari outer Class. Karena static nested class tidak membutuhkan instance dari outer class

```
OuterClass.NestedClass obj = new OuterClass.NestedClass();
```

Di sini, kita membuat objek dari static nested class. Jadi outer class tidak dapat direferensikan menggunakan this.

```
package app;
class MotherBoard {
   // static nested class
   static class USB {
       int usb2 = 2;
       int usb3 = 1;
       int getTotalPorts() {
           return usb2 + usb3;
       }
   }
}

public class Main {
   public static void main(String[] args) {
       // create an object of the static nested class
       // using the name of the outer class
       MotherBoard.USB usb = new MotherBoard.USB();
       System.out.println("Total Ports = " + usb.getTotalPorts());
   }
}
```

Output:

```
Total Ports = 3
```

Contoh lain

```
class Animal {

   // inner class
   class Reptile {
      public void displayInfo() {
        System.out.println("I am a reptile.");
      }
   }

   // static class
   static class Mammal {
      public void displayInfo() {
        System.out.println("I am a mammal.");
      }
   }
}

class Main {
   public static void main(String[] args) {
      // object creation of the outer class
      Animal animal = new Animal();

      // object creation of the non-static class
      Animal.Reptile reptile = animal.new Reptile();
      reptile.displayInfo();

      // object creation of the static nested class
      Animal.Mammal mammal = new Animal.Mammal();
      mammal.displayInfo();

   }
}
```

Output:

```
I am a reptile.
I am a mammal.
```

Pada contoh di atas terdapat dua nested class yaitu `class Reptile` dan `static class Mammal`.

Untuk membuat objek dari non-static class Reptile digunakan

```
Animal.Reptile reptile = animal.new Reptile()
```

Untuk membuat objek dari static class Mammal digunakan

```
Animal.Mammal mammal = new Animal.Mammal()
```

Catatan:

Hanya nested class yang dapat dijadikan static. Outer class tidak dapat dijadikan static.

Nested Anonymous Class

Nested class yang tidak memiliki nama dinamakan anonymous class. Sebuah anonymous class harus didefinisikan di dalam sebuah class. Jadi bisa dikatakan sebagai anonymous inner class. Sintaksnya adalah sebagai berikut:

```
Class outerClass {
   // defining anonymous class
   Object1 = new Type(parameterList) {
      // body of the anonymous class
   };
}
```

Anonymous class biasanya extend subclass atau implement interface

Jadi, Type bisa berupa:

- Superclass yang di mana anonymous class extends dari Superclass tersebut
- Interface yang di mana anonymous class implements dari Interface tersebut

Contoh 1: Nested Anonymous Class Extending Class

```
package app;
class Polygon {
   public void display() {
      System.out.println("Inside the Polygon class");
   }
}
class AnonymousDemo {
   public void createClass() {
      // creation of anonymous class extending class Polygon
      Polygon p1 = new Polygon() {
         public void display() {
            System.out.println("Inside an anonymous class.");
         }
      };
      p1.display();
   }
}
```

```
class Main {
   public static void main(String[] args) {
       AnonymousDemo an = new AnonymousDemo();
       an.createClass();

       Polygon pol = new Polygon();
       pol.display();
   }
}
```

Output:

```
Inside an anonymous class.
Inside the Polygon class
```

Pada contoh di atas, kita membuat class `Polygon` yang memiliki method `display()`. Kita kemudian membuat anonymous class yang extends class `Polygon` dan override method `display()`.

Ketika program dijalankan, `p1` adalah objek dari anonymous class. Objek tersebut kemudian memanggil method `display()` dari anonymous class.

Contoh 2: Nested Anonymous Class Implementasi Interface

```
interface PolygonInt {
    public void display();
}

class AnonymousDemoInt {
    public void createClass() {
        // anonymous class implementing interface
        PolygonInt p1 = new PolygonInt() {
          public void display() {
             System.out.println("Inside an anonymous class.");
          }
        };
        p1.display();
    }
```

```
}
class App {
    public static void main(String[] args) {
       AnonymousDemoInt an = new AnonymousDemoInt();
       an.createClass();
    }
}
```

Output:

```
Inside an anonymous class.
```

Java Anonymous Class

Anonymous class adalah class yang tidak memiliki nama. Anonymous class tidak boleh memiliki konstruktor. Biasanya anonymous class dibuat sekali pakai dengan tujuan untuk mengimplementasikan interface dan class abstrak secara langsung.

OOP Design dengan UML

Studi Kasus ATM

Objectives

- Memahami kebutuhan perancangan dan mendokumentasikannya (*requirements documentation*)
- Mengidentifikasikan Class dan atributnya berdasarkan *requirements*.
- Mengidentifikasikan status, aktivifitas, dan operasi objek berdasarkan *requirements*.
- Menentukan kolaborasi antar objek dalam system
- Diagram UML: usecase, class, state, activity, communication, dan sequence.

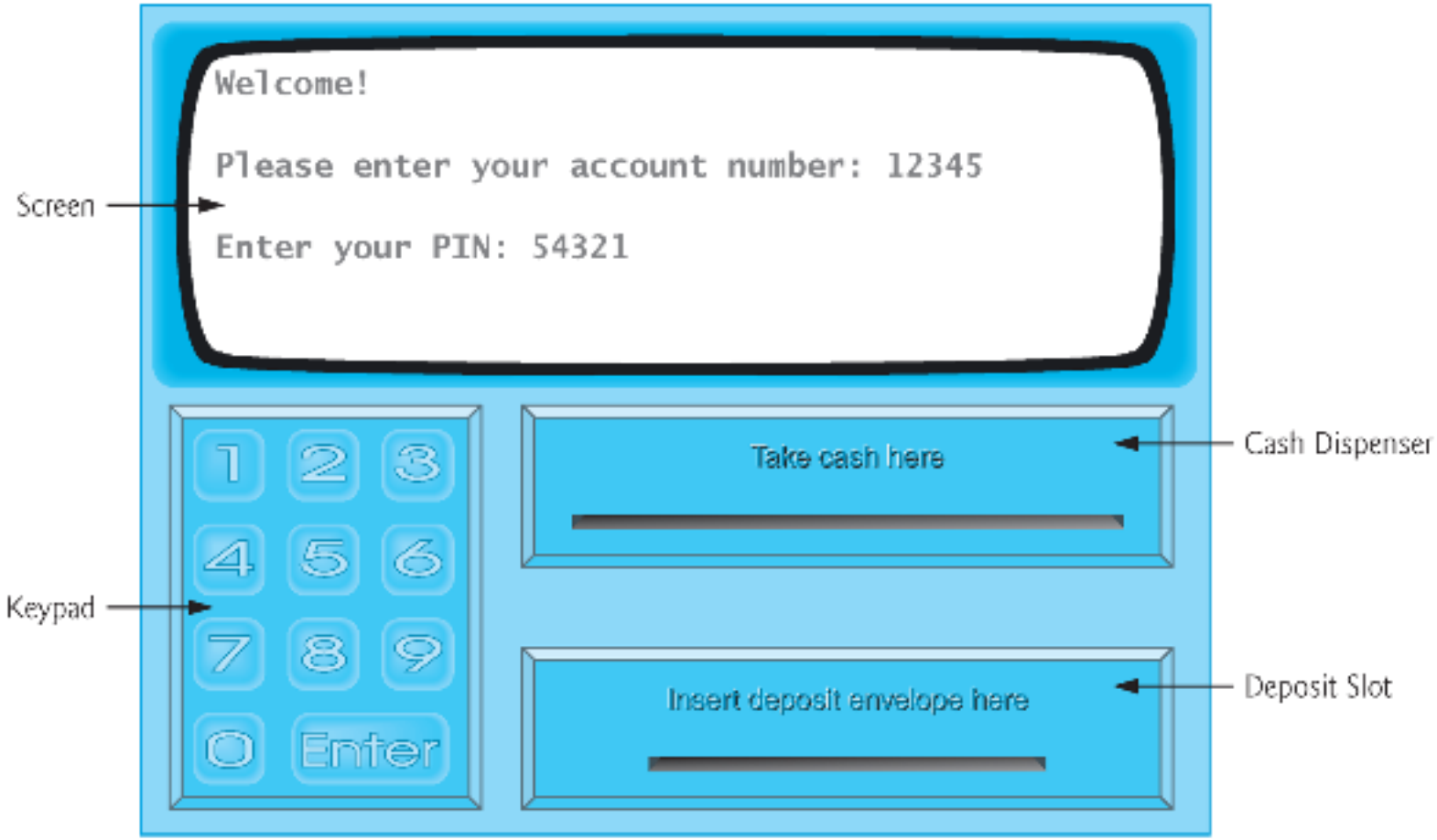


Fig. 25.1 | Automated teller machine user interface.

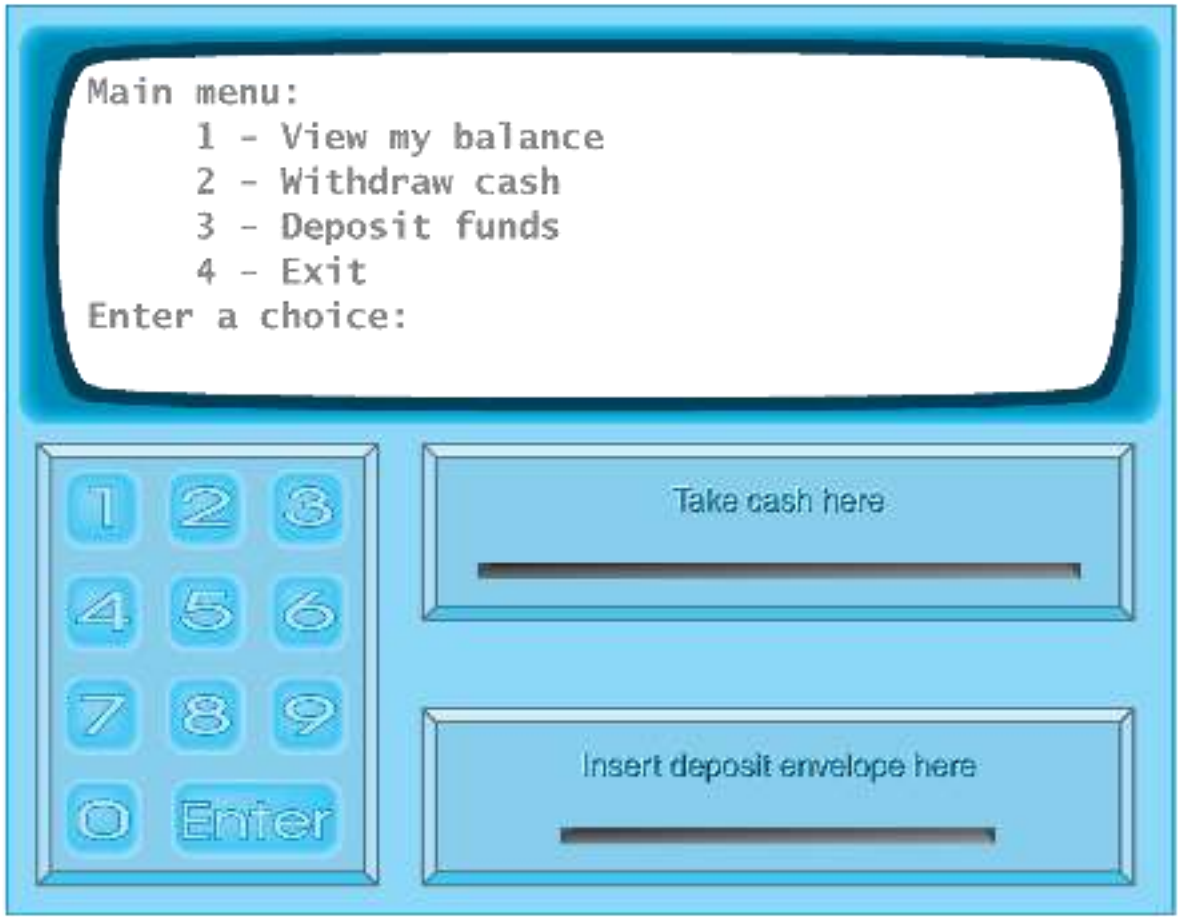


Fig. 25.2 | ATM main menu.

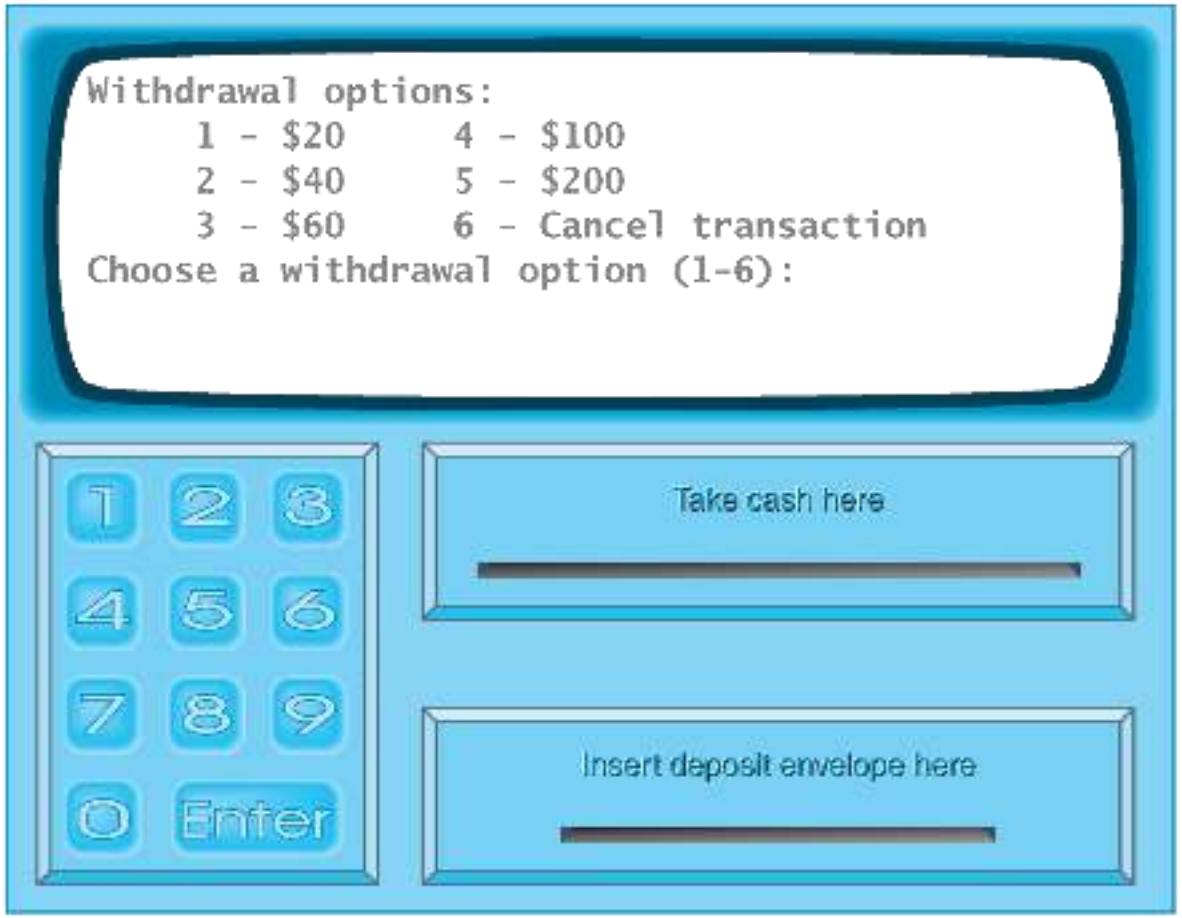


Fig. 25.3 | ATM withdrawal menu.

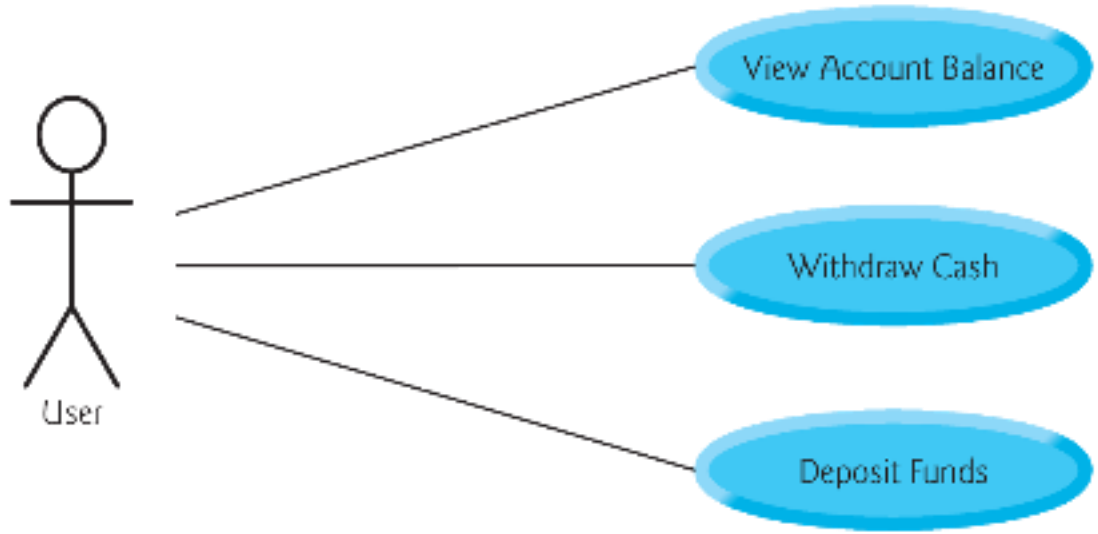


Fig. 25.4 | Use case diagram for the ATM system from the User's perspective.

Nouns and noun phrases in the requirements document			
bank	money / fund	account number	ATM
screen	PIN	user	keypad
bank database	customer	cash dispenser	balance inquiry
transaction	$20 bill / cash	withdrawal	account
deposit slot	deposit	balance	deposit envelope

Fig. 25.5 | Nouns and noun phrases in the requirements document.

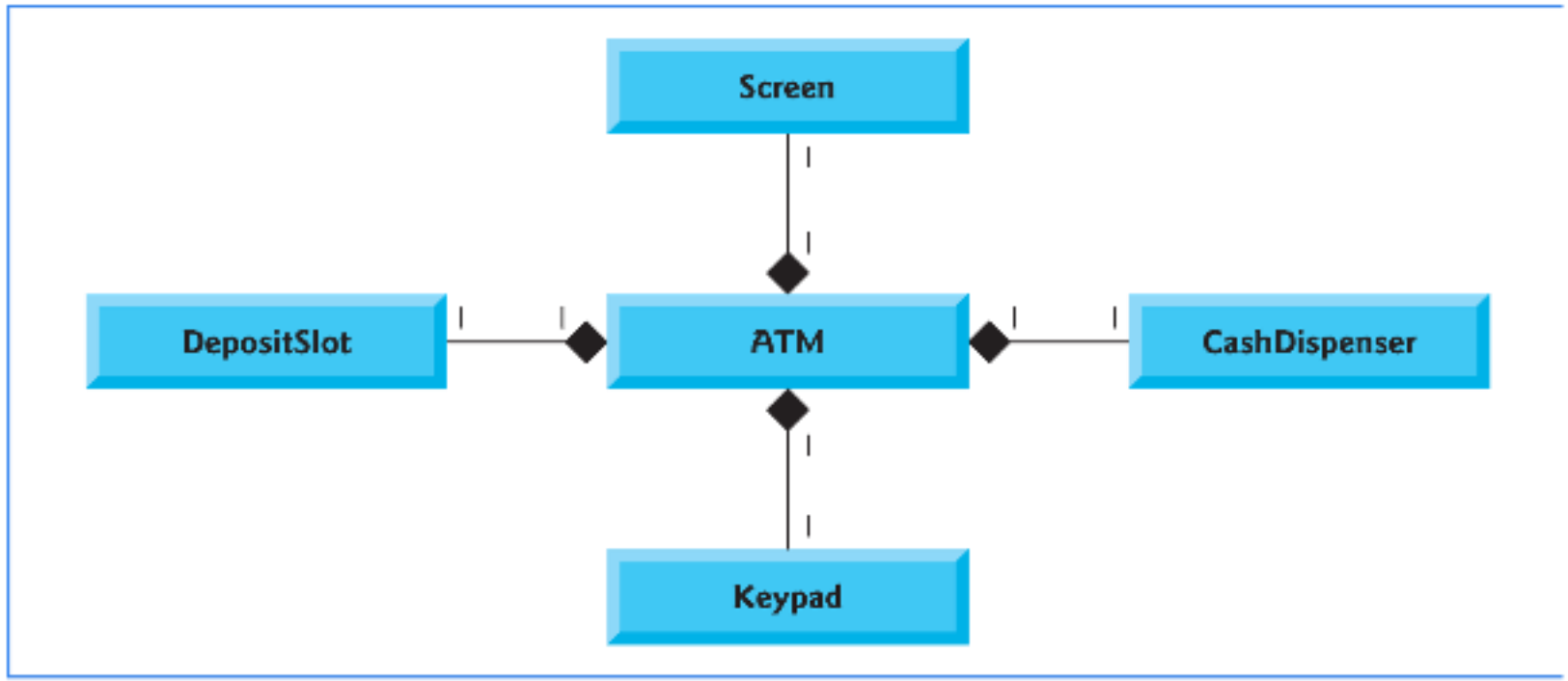
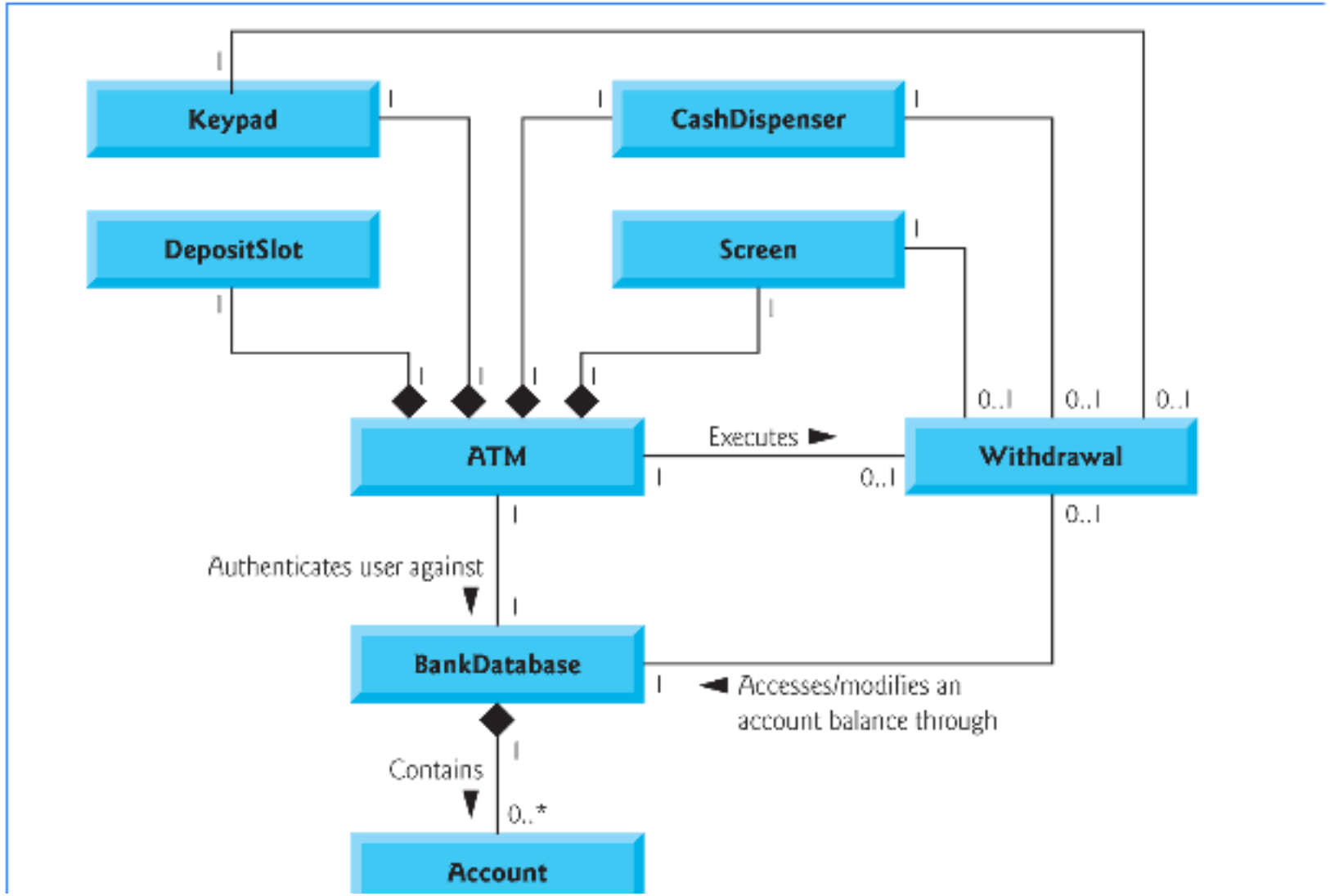


Fig. 25.9 | Class diagram showing composition relationships.


Keypad
CashDispenser
DepositSlot
Screen
ATM
Executes
Withdrawal
Authenticates user against
BankDatabase
Accesses/modifies an account balance through
Contains
Account
1
0..1
0..*

Class	Descriptive words and phrases
`ATM`	user is authenticated
`BalanceInquiry`	account number
`Withdrawal`	account number amount
`Deposit`	account number amount
`BankDatabase`	[no descriptive words or phrases]
`Account`	account number PIN balance
`Screen`	[no descriptive words or phrases]
`Keypad`	[no descriptive words or phrases]
`CashDispenser`	begins each day loaded with 500 $20 bills
`DepositSlot`	[no descriptive words or phrases]

Fig. 25.11 | Descriptive words and phrases from the ATM requirements.

Fig. 25.12 | Classes with attributes.

Fig. 25.13 | State diagram for the ATM object.

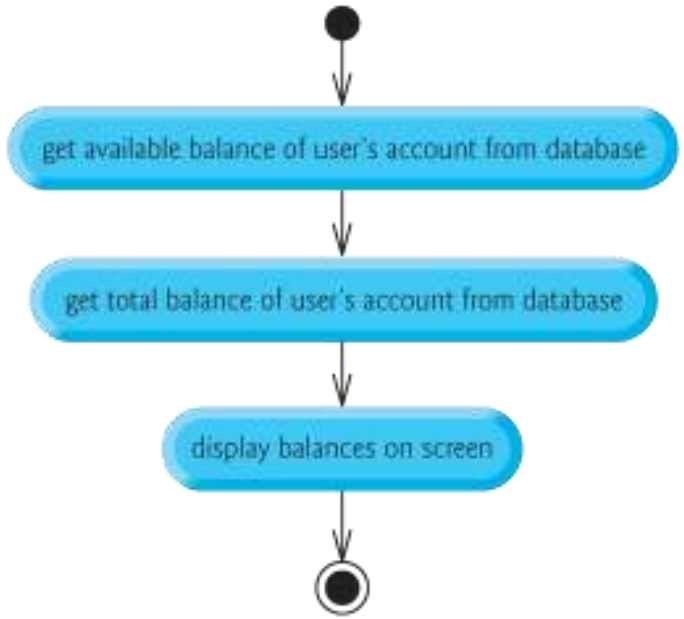


Fig. 25.14 | Activity diagram for a `BalanceInquiry` transaction.

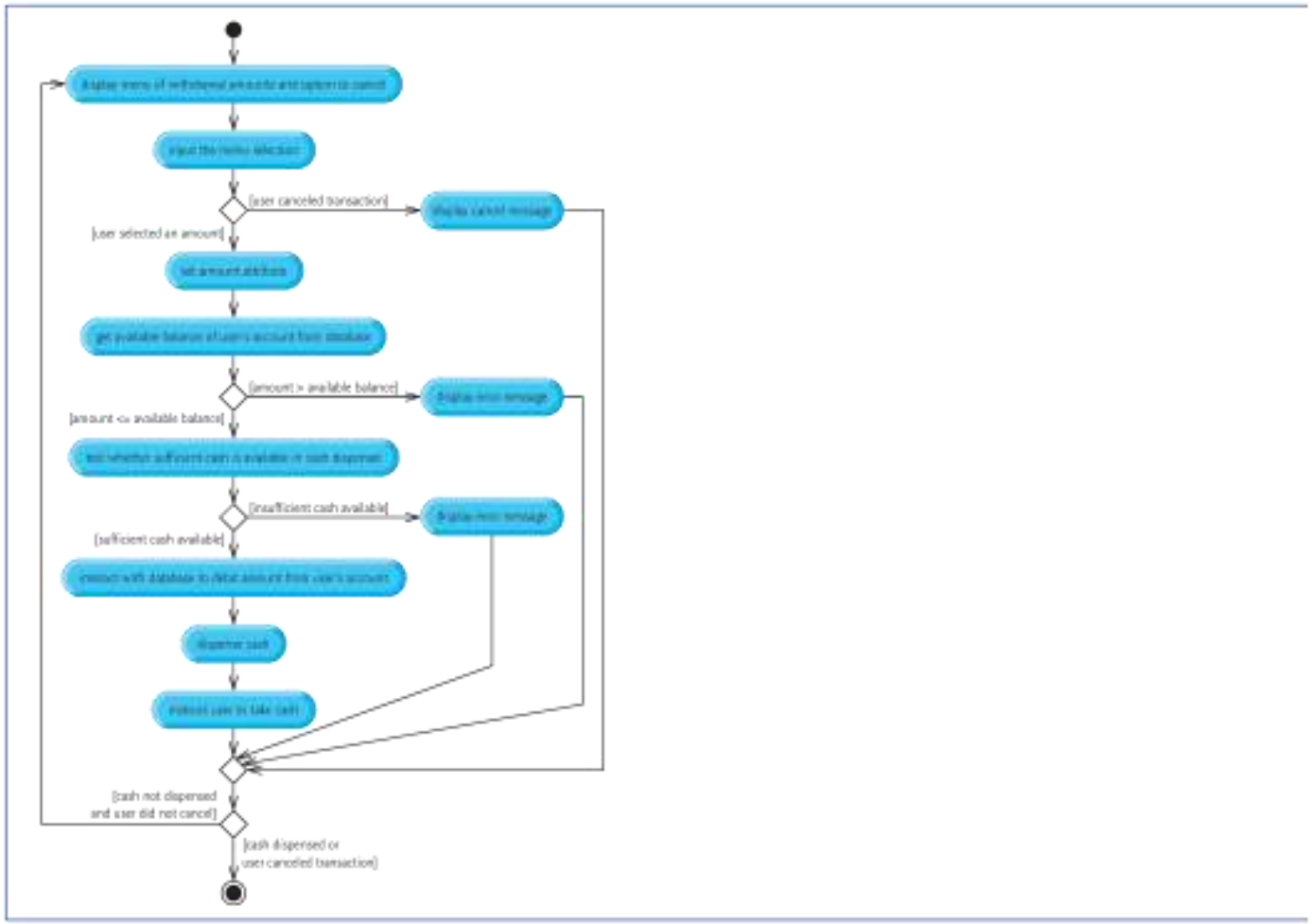


Fig. 25.15 | Activity diagram for a `Withdrawal` transaction.

Class	Verbs and verb phrases
`ATM`	executes financial transactions
`BalanceInquiry`	[none in the requirements document]
`Withdrawal`	[none in the requirements document]
`Deposit`	[none in the requirements document]
`BankDatabase`	authenticates a user, retrieves an account balance, credits a deposit amount to an account, debits a withdrawal amount from an account
`Account`	retrieves an account balance, credits a deposit amount to an account, debits a withdrawal amount from an account
`Screen`	displays a message to the user
`Keypad`	receives numeric input from the user
`CashDispenser`	dispenses cash, indicates whether it contains enough cash to satisfy a withdrawal request
`DepositSlot`	receives a deposit envelope

Fig. 25.16 | Verbs and verb phrases for each class in the ATM system.

Fig. 25.17 | Classes in the ATM system with attributes and

An object of class...	sends the message...	to an object of class...
ATM	displayMessage	Screen
	getInput	Keypad
	authenticateUser	BankDatabase
	execute	BalanceInquiry
	execute	Withdrawal
	execute	Deposit
BalanceInquiry	getAvailableBalance	BankDatabase
	getTotalBalance	BankDatabase
	displayMessage	Screen
Withdrawal	displayMessage	Screen
	getInput	Keypad
	getAvailableBalance	BankDatabase
	isSufficientCashAvailable	CashDispenser
	debit	BankDatabase
	dispenseCash	CashDispenser

Fig. 25.22 | Collaborations in the ATM system. (Part 1 of 2.)

An object of class...	sends the message...	to an object of class...
Deposit	displayMessage	Screen
	getInput	Keypad
	isEnvelopeReceived	DepositSlot
	credit	BankDatabase
BankDatabase	validatePIN	Account
	getAvailableBalance	Account
	getTotalBalance	Account
	debit	Account
	credit	Account

Fig. 25.22 | Collaborations in the ATM system. (Part 2 of 2.)

Fig. 25.23 | Communication diagram of the ATM executing a balance inquiry.

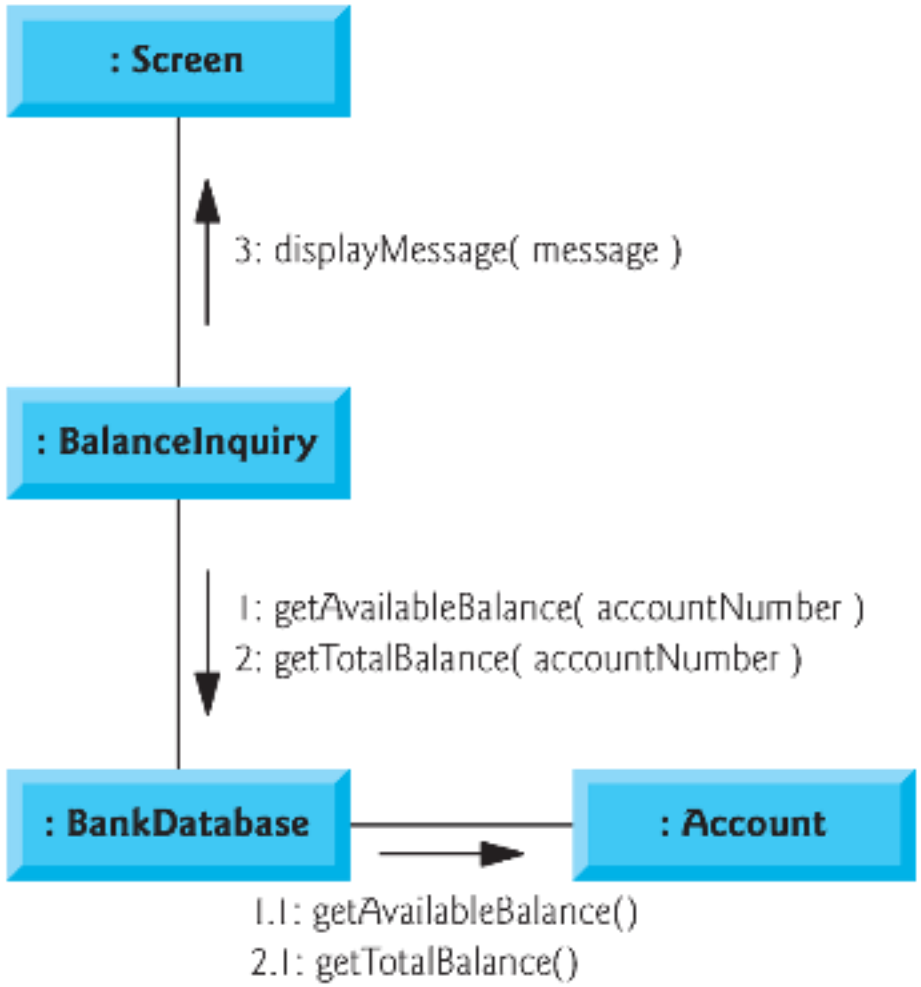


Fig. 25.24 | Communication diagram for executing a balance inquiry.

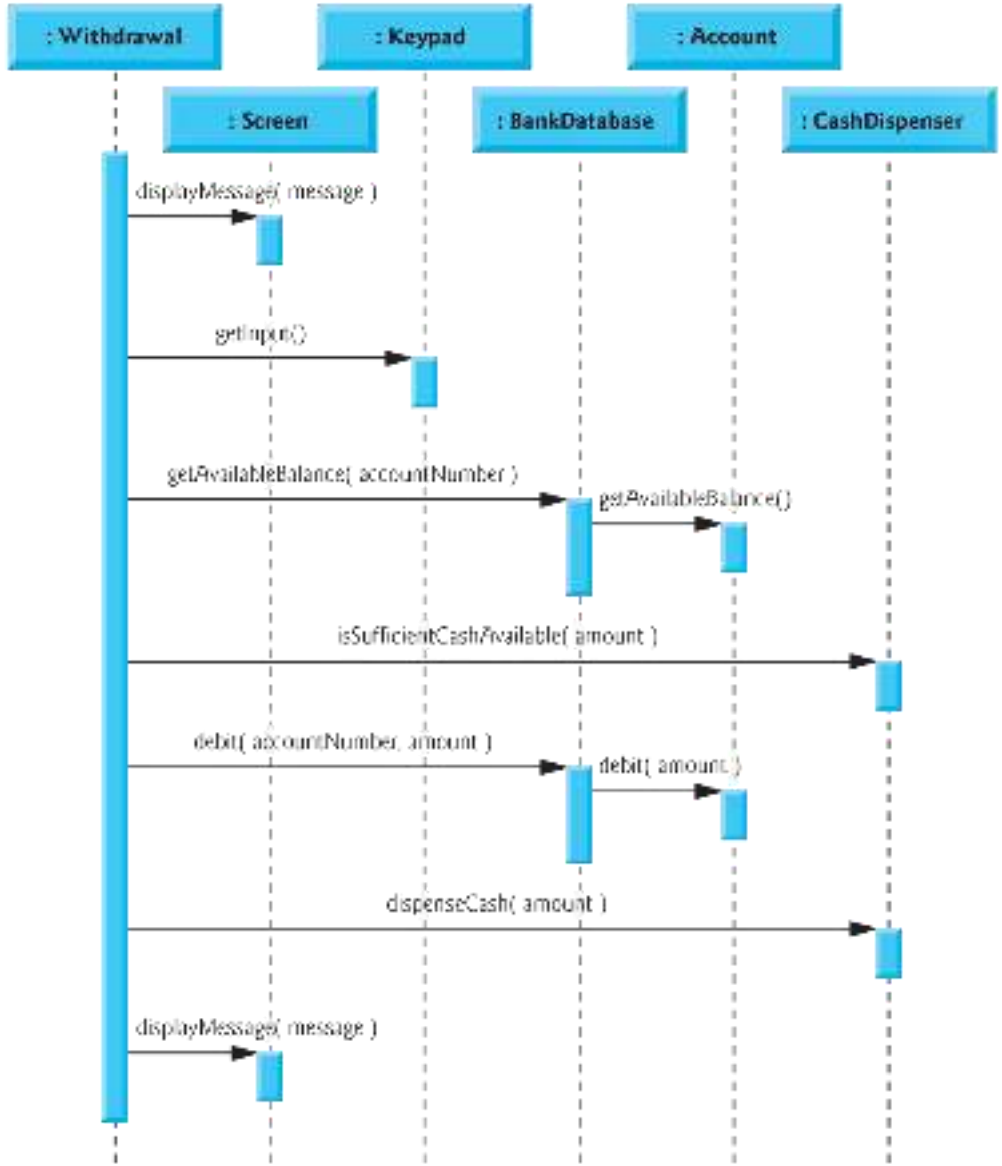


Fig. 25.25 | Sequence diagram that models a `Withdrawal` executing.

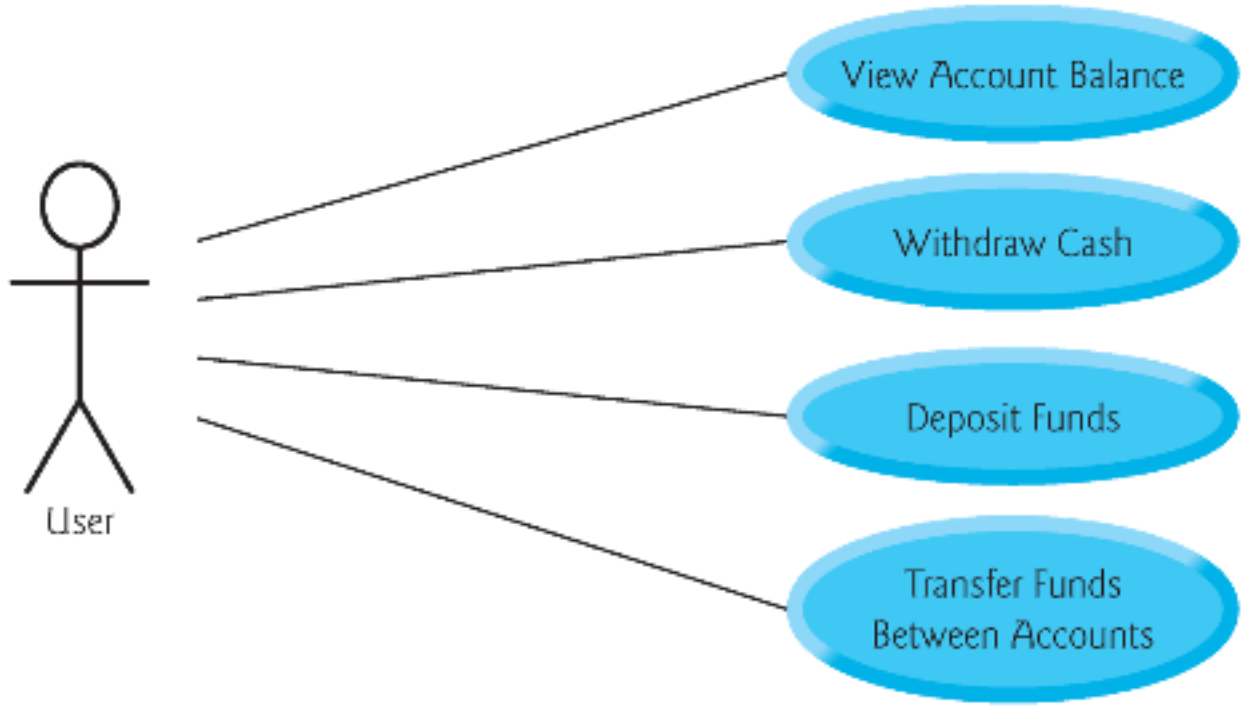


Fig. 25.26 | Use case diagram for a modified version of our ATM system that also allows users to transfer money between accounts.

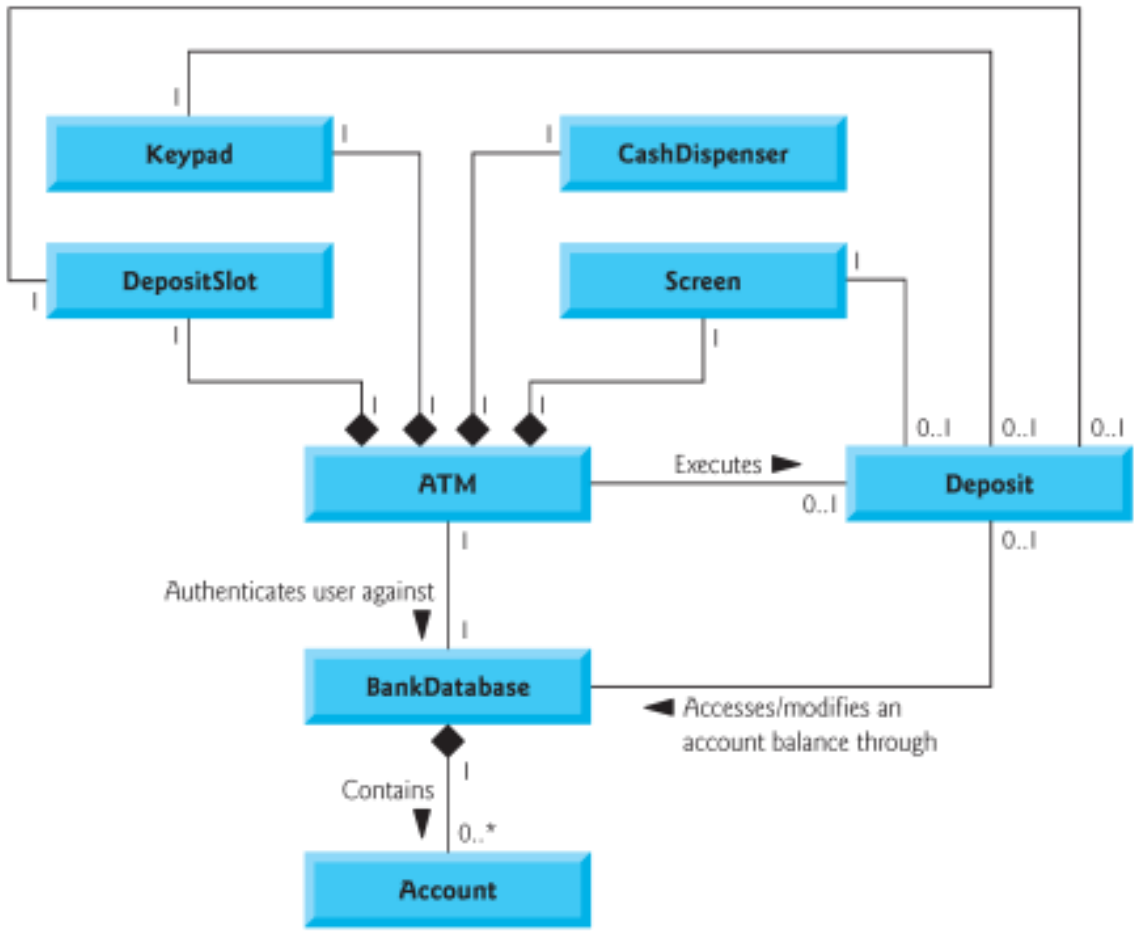


Fig. 25.28 | Class diagram for the ATM system model including

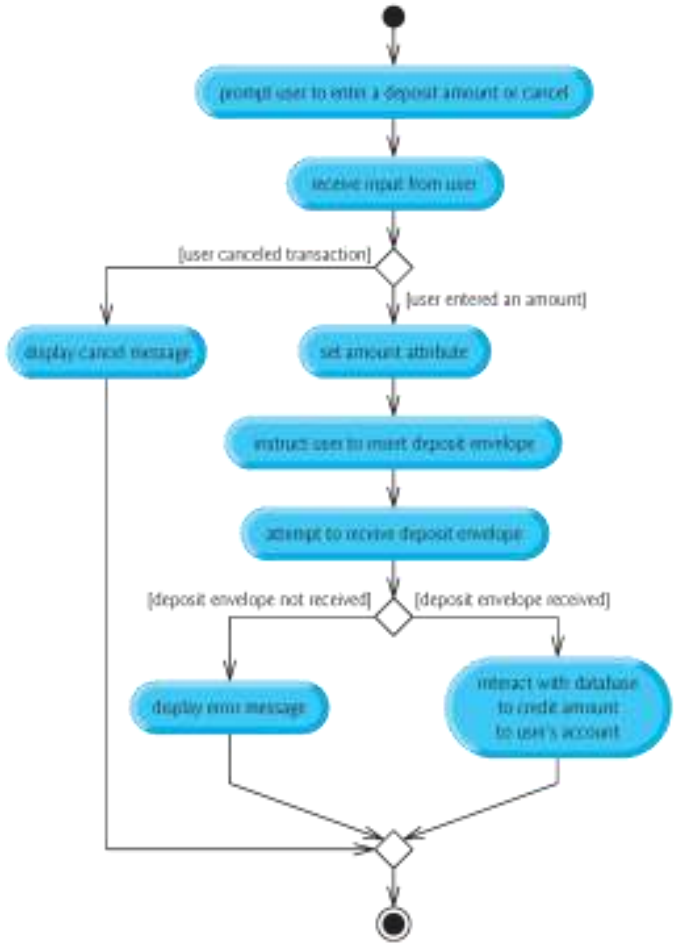


Fig. 25.29 | Activity diagram for a `Deposit` transaction.

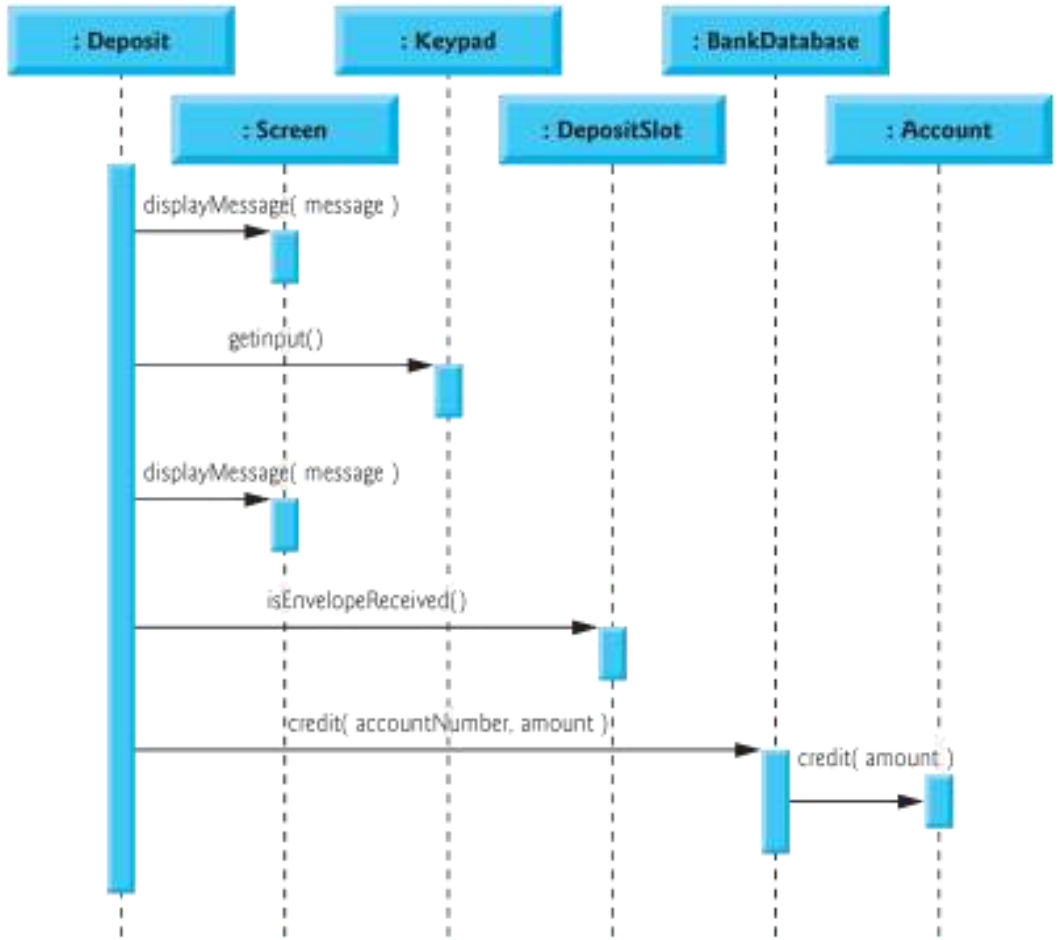


Fig. 25.30 | Sequence diagram that models a `Deposit` executing.

PERTEMUAN KEDUABELAS

Java Lambda

Fungsi Anonymous

Anonymous menggambarkan sesuatu yang tidak memiliki nama. Di Java terdapat fungsi *anonymous* dan *class anonymous* seperti yang telah dibahas pada materi sebelumnya.

Fungsi *anonymous* adalah fungsi yang tidak memiliki nama. Fungsi *anonymous* di Java dikenal dengan nama ***Lambda Expression***.

Fungsi anonymous biasanya dibuat hanya untuk sekali pakai.

Artinya, saat kita membuat fungsi anonymous, kita akan mengeksekusinya saat itu juga. Tidak bisa dipanggil lagi seperti fungsi biasa.

Fungsi ini mulai ditambahkan pada JDK 8.

Membuat Fungsi Anonymous di Java

Bentuk umum *lambda expression* adalah

```
(int param1, int param2 ) -> {
    // kode fungsi
}
```

Simbol yang perlu diingat adalah

```
() -> { }
```

Keterangan:

- () tempat menaruh paramater
- -> adalah operator lambda yang menandakan bahwa fungsi ini adalah anonymous
- {} body fungsinya

Contoh:

```
(int x, int y) -> { return x + y };
```

Fungsi anonymous dapat dibuat di berbagai tempat seperti:

- Deklarasi variable

```
int variabel = () -> { return 0 };
```

- Pengisian variable dan array

```
int variabel;
int arr;
variabel = () -> { return 0 };
arr = () -> { return {0,4,3,2,1} };
```

- Saat mengembalikan nilai dengan return

```
int methodName(){
    return () -> { return 0 };
}
```

- Body lambda

```
() -> {
    return () -> 5 + 2;
};
```

- Ekspresi kondisional

```
String jawab = (int x) -> { x < 10} ? () -> return "yes": () -> return
"no";
```

Kenapa Menggunakan Fungsi Anonymous?

Lambda expression atau fungsi anonymous sebenarnya hadir untuk menyempurnakan class anonymous. Class anonymous biasanya digunakan untuk mengimplementasikan interface dan class abstrak. Tapi kendalanya saat interface hanya memiliki satu method saja untuk diimplementasikan. Kita harus membuat class (anonymous) baru. Padahal hanya dibutuhkan method-nya saja. Ini lah saatnya untuk menggunakan fungsi anonymous atau lambda expression.

Contoh Program Fungsi Anonymous

Interface Clickable

```
public interface Clickable {
    void onClick();
}
```

Class Button

```
public class Button {
    private Clickable action;

    public void setClickAction(Clickable action){
        this.action = action;
    }

    public void doClick(){
        action.onClick();
    }
}
```

Main Class dengan Anonymous Class

```
public class Main {
    public static void main(String[] args) {
        Button btn = new Button();
        String name = "Jagoan Neon";

        // membuat class anonymous untuk implementasi interface
        btn.setClickAction(new Clickable() {
            @Override
            public void onClick() {
                System.out.println("Tombol sudah diklik!");
                System.out.println("Yay!");
                System.out.println("Hello, " + name + "!");
            }
        });

        // mencoba klik tombol
        btn.doClick();
    }
}
```

Output

```
Tombol sudah diklik!
Yay!
Hello, Jagoan Neon!
```

Main Class dengan Lambda Expression

```
public class Main {
    public static void main(String[] args) {
        Button btn = new Button();
        String name = "Jagoan Neon";

        // membuat lambda expression untuk implementasi interface
        btn.setClickAction(() -> {
            System.out.println("Tombol sudah diklik!");
            System.out.println("Yay!");
            System.out.println("Hello, " + name + "!");
        });

        // mencoba klik tombol
        btn.doClick();
    }
}
```

Output

```
Tombol sudah diklik!
Yay!
Hello, Jagoan Neon!
```

PERTEMUAN KESEMBILAN

Java Packages dan Enumerasi

Prepared by Adi Wahyu Pribadi

Pendahuluan

Java Package adalah mekanisme untuk mengelompokkan kelas (class), antarmuka (interface), enumerasi (enum), dan anotasi (annotation) yang terkait dalam sebuah unit namespace yang terstruktur.

Package di Java terbagi menjadi dua macam:

- *Built-in Package* (Paket dari Java API)
- *User-defined Package* atau buatan sendiri

Kegunaan Package

- Mengorganisir kode
  Memudahkan pengelolaan dan pencarian kelas yang terkait.
- Menghindari konflik dalam penamaan
  Kelas dengan nama yang sama dapat dibedakan berdasarkan package-nya.
- Kontrol akses
  Menentukan kelas atau anggota kelas mana yang dapat diakses dari package lain.

Built-in Package

Java API adalah library kumpulan Class yang sudah ditulis dan dapat langsung digunakan secara bebas. Library berisi komponen untuk mengelola input, pemrograman basis data, dan banyak lagi lainnya. Dengan built-in packages, pengembang dapat fokus pada logika bisnis aplikasi tanpa harus mengimplementasikan fungsi dasar dari awal.

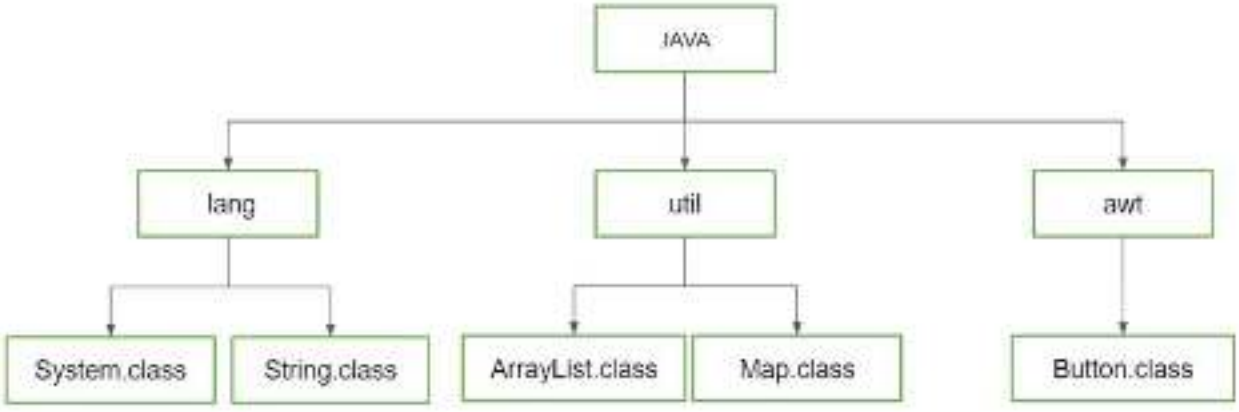
Daftar lengkapnya dapat ditemukan di situs web Oracles:

https://docs.oracle.com/javase/8/docs/api/

https://docs.oracle.com/en/java/javase/11/docs/api/index.html

Built-in Package Java 19 (Oktober 2023)

https://www.geeksforgeeks.org/built-in-packages-in-java/



Beberapa built-in package Oracle Java 19:

java.lang	java.math
java.util	java.io
java.net	java.sql
java.security	java.util.concurrent
java.util.stream	java.util.regex
java.util.zip	java.awt
java.swing	

Built-in package menyediakan fungsi dasar yang dibutuhkan sebagian besar aplikasi Java. Beberapa kegunaan built-in package adalah:

- memanipulasi string dan angka
- mengelola file dan jaringan
- mengakses database
- melakukan enkripsi dan dekripsi
- mengelola multithreading
- mengembangkan aplikasi GUI

Library terbagi atas packages dan class. Berarti kita dapat mengimpor satu class (beserta metode dan atributnya), atau seluruh package yang berisi semua class yang ada di dalam package yang kita pilih.

Untuk menggunakan sebuah class atau package dari library, kita gunakan keyword **import**

Syntax

```
import package.name.Class;   // Import sebuah class
import package.name.*;       // Import seluruh package
```

Import sebuah class

```
import java.util.Scanner;
```

Pada baris di atas, nama package-nya adalah java.util dan nama class-nya adalah Scanner. Untuk menggunakan buat objek class dan gunakan salah satu metode yang tersedia yang ditemukan dalam dokumentasi class Scanner. Kita akan menggunakan method nextLine(), yang digunakan untuk membaca baris lengkap:

```
import java.util.Scanner;
class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Masukkan nama Anda");

    String userName = myObj.nextLine();
    System.out.println("Nama Anda adalah " + userName);
  }
}
```

Import Sebuah Package

Ada banyak paket untuk dipilih. Pada contoh sebelumnya, kita menggunakan class Scanner dari package java.util. Package ini juga berisi fungsi utilitas seperti tanggal dan waktu, pembuat nomor acak, dan class utilitas lainnya.

Untuk mengimpor seluruh package, akhiri kalimat dengan tanda bintang (*). Contoh berikut akan mengimpor semua class di package java.util:

```
import java.util.*;
```

User-Defined Package atau Paket Buatan Sendiri

Untuk membuat package sendiri, Java menggunakan sistem folder file untuk menyimpan, sama seperti folder di komputer. Misalkan:

```
|___ src
   |___ App
      |___ ClassSaya.java
```

Untuk menggunakan sebuah package gunakan keyword package;

```
package App;
class ClassSaya {
  public static void main(String[] args) {
    System.out.println("Ini sebuah package!");
  }
}
```

Simpan file di atas dengan nama ClassSaya.java di dalam folder src

Compile File ClassSaya.java dengan perintah

```
C:\Users\Adiwa\src> javac ClassSaya.java
```

Menjalankan file ClassSaya.class

```
C:\Users\Adiwa\src>java ClassSaya
Error: Could not find or load main class ClassSaya
Caused by: java.lang.NoClassDefFoundError: App/ClassSaya (wrong name:
ClassSaya)
```

Muncul Error dikarenakan ClassSaya terdapat package App

Maka gunakan perintah javac -d . ClassSaya.java

```
C:\Users\Adiwa\src>javac -d . ClassSaya.java
```

Dan untuk menjalankan ClassSaya adalah dengan perintah `java NamaPackage.NamaClass`

```
C:\Users\Adiwa\src>java App.ClassSaya
Ini sebuah package!
```

Java Enums

Enum adalah Class spesial yang merepresentasikan grup KONSTAN/CONSTANTS (variable yang tidak bisa diubah, seperti variabel final). Untuk membuath sebuah enum, gunakan keyword enum, dan pisahkan constant dengan tanda koma (,). Ingat, constant ditulis dengan HURUF BESAR.

```
enum Level {
   LOW, MEDIUM, HIGH
}
```

Untuk mengakses konstan di enum gunakan titik / dot syntax:

```
Level kemahiran = Level.MEDIUM;
Level rasaPedas = Level.MEDIUM;
Level rasaPedas = Level.HIGH;
```

Enum di dalam sebuah Class

```
Public class ClassKu {
   enum Level {
      LOW, MEDIUM, HIGH
   }
   Public static void main(String[] args) {
      Level kemahiran = Level.HIGH;
      System.out.println(kemahiran);
   }
}
```

Outputnya:

```
HIGH
```

Enum di dalam Statement Switch

```
enum Level {
  LOW, MEDIUM, HIGH
}

public class ClassKu {
  public static void main(String[] args) {
    Level kemahiran = Level.MEDIUM;

    switch(kemahiran) {
      case LOW:
        System.out.println("Anak bawang");
        break;
      case MEDIUM:
         System.out.println("Anggota");
        break;
      case HIGH:
        System.out.println("Suhu");
        break;
    }
  }
}
```

Outputnya:

```
Anggota
```

Looping di dalam Enum

Tipe enum memiliki sebuah method bernama values(), method tersebut akan mengembalikan array dari seluruh constant yang ada di dalam enum tersebut. Method ini bermanfaat ketika ingin menampilkan seluruh constant yang ada di enum.

```
for (Level kemahiran : Level.values()) {
  System.out.println(kemahiran);
}
```

Outputnya:

```
LOW
MEDIUM
HIGH
```

Java Enum juga dapat digunakan untuk membuat method dan properti. Berikut contoh penggunaannya:

```
enum Warna {
  MERAH(255, 0, 0),
  HIJAU(0, 255, 0),
  BIRU(0, 0, 255);

  private int r, g, b;

  Warna(int r, int g, int b) {
    this.r = r;
    this.g = g;
    this.b = b;
  }

  public int getRed() {
    return r;
  }
  public int getGreen() {
    return g;
  }
  public int getBlue() {
    return b;
  }
}
```

```
public class DemoWarna {
    public static void main(String[] args) {
        // Menampilkan semua warna dan komponen RGB mereka
        for (Warna warna : Warna.values()) {
            System.out.println("Warna: " + warna.name());
            System.out.println("  Merah  : " + warna.getRed());
            System.out.println("  Hijau  : " + warna.getGreen());
            System.out.println("  Biru   : " + warna.getBlue());
            System.out.println();
        }

        // Demonstrasi penggunaan enum dalam kondisi
        Warna baju = Warna.HIJAU;

        if (baju == Warna.HIJAU) {
            System.out.println("Baju berwarna hijau cocok untuk pergi ke
alam!");
        } else if (baju == Warna.MERAH) {
            System.out.println("Baju berwarna merah cocok untuk pesta!");
        } else if (baju == Warna.BIRU) {
            System.out.println("Baju berwarna biru cocok untuk ke kantor!");
        }
    }
}
```

Output:

```
Warna: MERAH
  Merah  : 255
  Hijau  : 0
  Biru   : 0

Warna: HIJAU
  Merah  : 0
  Hijau  : 255
  Biru   : 0

Warna: BIRU
  Merah  : 0
  Hijau  : 0
  Biru   : 255

Baju berwarna hijau cocok untuk pergi ke alam!
```

Fitur	Enum	Class
Nilai	Terbatas	Tidak terbatas
Metode	Tidak ada	Ada
Properti	Tidak ada	Ada
Kegunaan	Representasi nilai-nilai yang terbatas	Representasi kumpulan data dan perilaku