Mata Kuliah	:	Pemrograman Berorientasi Objek
Tanggal	:	17 September 2024
Semester	:	3 (Gasal)
Topik	:	Java OOP
Minggu/Pertemuan/Sesi	:	03/02/01-02
Aktivitas	:	Praktikum
Durasi	:	170 menit
Setoran	:	Dokumen Laporan PDF
Batas Penyerahan	:	-
Tempat Penyerahan	:	https://ecourse.del.ac.id/

Daftar Isi

A. Tujuan & Penilaian

1. Tujuan

- ✓ Mahasiswa mampu memahami dan menggunakan kode program Java OOP.
- ✓ Mahasiswa mampu mengimplementasikan Class Diagram menggunakan kode program Java OOP.

2. Penilaian

No	Kriteria	Bobot Penilaian (%)
1	Latihan Java OOP	20
2	Mengimplementasikan Class Diagram	20
3	Studi Kasus 1: Student Information	60

3. Dokumen Laporan

Buat laporan praktikum menggunakan [Template Laporan Praktikum] [Contoh Laporan Praktikum]. Sebelum mengirimkan dokumen laporan pada E-Course silahkan ubah penamaan file dengan format:

{NIM}-pbo-laporan-praktikum-4.pdf

Sebagai contoh:

11S18005-pbo-laporan-praktikum-4.pdf

4. Membuat Workspace

Buat *workspace* baru pada VSCode dengan format penamaan "**{username}-pbo-praktikum-4**".

5. Prettier

Prettier adalah sebuah alat (tool) yang digunakan untuk memformat kode secara otomatis dan konsisten dalam berbagai bahasa pemrograman. Tujuannya adalah untuk menghilangkan perbedaan gaya penulisan yang umumnya terjadi antara pengembang yang berbeda, sehingga membuat kode lebih mudah dibaca dan dipelihara.

a) Buat file baru dengan nama "**.prettierrc**" pada workspace kamu.

b) Modifikasi isi file "**.prettierrc**", seperti berikut:

.prettierrc

```
{
  "semi": true,
  "singleQuote": true,
  "tabWidth": 2,
  "overrides": [
    {
      "files": "*.java",
      "options": {
        "parser": "java-parser",
        "java": true,
        "bracketSpacing": false,
        "importOrder": ["java", "javax", "org", "com"]
      }
    }
  ]
}
```

B. Aktivitas Praktikum

1. Latihan Java OOP

a) Class

Untuk membuat class, kita bisa menggunakan kata kunci class. Penamaan class biasa menggunakan format CamelCase. Nama class di Java harus sama dengan nama filenya.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2
3  }
```

b) Object

Object adalah hasil dari instansisasi dari sebuah class. Untuk membuat object kita bisa menggunakan kata kunci new, dan diikuti dengan nama class dan kurung ().
Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      var person1 = new Person();
5      Person person2 = new Person();
6      Person person3;
7      person3 = new Person();
8
9      System.out.println(person1);
10     System.out.println(person2);
11     System.out.println(person3);
12
13   }
14 }
```

Analisis kegunaan dari kode program pada line 4, 5, 6, dan 7. Amati hasilnya pada terminal.

c) Field

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

| Manipulasi Field

Fields yang ada di object, bis akita 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

variabel. Untuk mengakses field, kita butuh simbol titik (.) setelah nama object dan diikuti nama field-nya.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2    String name;
3    String address;
4    final String country = "Indonesia";
5  }
```

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      var person1 = new Person();
5      person1.name = "Abdullah";
6      person1.address = "Sitoluama";
7      // person1.country = "Singapura";
8
9      System.out.println(person1.name);
10     System.out.println(person1.address);
11     System.out.println(person1.country);
12
13   }
14 }
```

Analisis kegunaan kode program pada line 5, 6, 9, 10, dan 11. Mengapa kode program pada line 7, jika dihilangkan komen-nya akan mengalami error. Amati hasilnya pada terminal.

d) Method

Selain menambahkan field, kita juga bisa menambahkan method ke object. Caranya 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 simbol titik (.) dan diikuti dengan nama method-nya. Sama seperti mengakses field.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2    String name;
3    String address;
4    final String country = "Indonesia";
5
6    void sayHello(String paramName) {
7      System.out.println("Hello " + paramName + ", My name is " + name);
8    }
9  }
```

Analisis kegunaan kode program pada line 6 – 8.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      var person1 = new Person();
5      person1.name = "Abdullah";
6      person1.address = "Sitoluama";
7      // person1.countrt = "Singapura";
8
9      System.out.println(person1.name);
10     System.out.println(person1.address);
11     System.out.println(person1.country);
12
13     person1.sayHello("Kevin");
14
15     Person person2;
16     person2 = new Person();
17     person2.name = "Kevin";
18
19     person2.sayHello("Abdullah");
20
21   }
22 }
```

Analisisi kegunaan kode program pada line 13 dan 19. Amati hasilnya pada terminal.

e) Constructor

Saat kita membuat object, maka kita seperti memanggil sebuah method, karena kita menggunakan simbol kurung (). Di dalam class Java, kita bisa membuat constructor, constructor adalah method yang akan dipanggil saat pertama kali object dibuat. Mirip seperi di method, kitab isa memberi parameter pada constructor. Nama constructor harus sama dengan nama class, dan tidak membutuhkan kata kunci void atau return value.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2    String name;
3    String address;
4    final String country = "Indonesia";
5
6    Person(String paramName, String paramAddress) {
7      name = paramName;
8      address = paramAddress;
9    }
10
11   void sayHello(String paramName) {
12     System.out.println("Hello " + paramName + ", My name is " + name);
13   }
14 }
```

Analisis kegunaan kode program pada line 6 – 9.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
```

App.java

```
2     public static void main(String[] args) {
3
4       var person1 = new Person("Abdullah", "Sitoluama");
5
6       System.out.println(person1.name);
7       System.out.println(person1.address);
8       System.out.println(person1.country);
9
10      person1.sayHello("Kevin");
11
12      Person person2;
13      // person2 = new Person("Kevin");
14      person2 = new Person("Kevin", "Sitoluama");
15
16      person2.sayHello("Abdullah");
17
18    }
19  }
```

Analisis kegunaan kode program pada line 4 dan 14. Apa yang terjadi jika komentar pada line 13 dihilangkan dan mengapa hasilnya bisa seperti itu. Amati hasilnya pada terminal.

f) Constructor Overloading

Sama seperti di method, di constructor kita bisa melakukan overloading. Kita bisa membuat constructor lebih dari satu, dengan syarat tipe data parameter harus berbeda, atau jumlah parameter harus berbeda.

| Memanggil Constructor Lain

Constructor bisa memanggil constructor lain. Hal ini memudahkan saat kita butuh menginisialisasi data dengan berbagai kemungkinan. Cara untuk memanggil constructor lain, dapat dilakukan seperti memanggil method, namun dengan kata kunci this.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1   class Person {
2     String name;
3     String address;
4     final String country = "Indonesia";
5
6     Person() {
7       this(null);
8     }
9
10    Person(String paramName) {
11      this(paramName, null);
12    }
13
14    Person(String paramName, String paramAddress) {
15      name = paramName;
16      address = paramAddress;
17    }
18
19    void sayHello(String paramName) {
20      System.out.println("Hello " + paramName + ", My name is " + name);
```

Person.java

```
21    }
22  }
```

Analisis kegunaan kode program pada line 6-8, 10-12, dan 14-17.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      var person1 = new Person("Abdullah", "Sitoluama");
5
6      System.out.println(person1.name);
7      System.out.println(person1.address);
8      System.out.println(person1.country);
9
10     person1.sayHello("Kevin");
11
12     Person person2 = new Person("Kevin");
13
14     person2.sayHello("Abdullah");
15
16     Person person3 = new Person();
17
18   }
19 }
```

Analisis kegunaan kode program pada line 4, 10, dan 16. Amati hasilnya pada terminal.

g) Variable Shadowing

Variabel shadowing adalah kejadian Ketika kita membuat nama variabel dengan nama yang sama di scope yang menutupi variabel dengan nama yang sama di scope atasnya. Ini biasa terjadi seperti kita membuat nama parameter di method sama dengan nama field di class. Saat terjadi variable shadowing, maka secara otomatis variabel di scope atasnya tidak bisa diakses.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2    String name;
3    String address;
4    final String country = "Indonesia";
5
6    Person() {
7      this(null);
8    }
9
10   Person(String paramName) {
11     this(paramName, null);
12   }
13
14   Person(String name, String address) {
15     name = name;
16     address = address;
17   }
```

Person.java

```
18
19    void sayHello(String name) {
20      System.out.println("Hello " + name + ", My name is " + name);
21    }
22  }
```

Analisis kegunaan kode program pada line 14-17 dan 19-21. Amati hasilnya pada terminal.

h) this Keyword

Saat kita membuat kode di dalam block constructor atau method di dalam class, kita bisa menggunakan kata kunci this untuk mengakses object saat ini. Misal kadang kita butuh mengakses sebuah field yang namanya sama dengan dengan parameter method, hal ini tidak bisa dilakukan jika langsung menyebut nama field, kita bisa mengakses nama field tersebut dengan kata kunci this. This juga tidak hanya digunakan untuk mengakses field milik object saat ini, namun juga bisa digunakan untuk mengakses method. This bisa digunakan untuk mengatasi masalah variable shadowing.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1   class Person {
2     String name;
3     String address;
4     final String country = "Indonesia";
5
6     Person() {
7       this(null);
8     }
9
10    Person(String paramName) {
11      this(paramName, null);
12    }
13
14    Person(String name, String address) {
15      this.name = name;
16      this.address = address;
17    }
18
19    void sayHello(String name) {
20      System.out.println("Hello " + name + ", My name is " + this.name);
21    }
22  }
```

Analisis kegunaan kode program pada line 14-17 dan 19-21. Amati hasilnya pada terminal.

i) Inheritance

Inheritance atau pewarisan adalah kemampuan untuk menurunkan sebuah class ke class lain. Dalam artian, kita bisa membuat class Parent dan class Child. Class Child, hanya bisa punya satu class Parent, namun satu class Parent bisa punya banyak class Child. Saat sebuah class diturunkan, maka semua field

dan method yang ada di class Parent, secara otomatis akan dimiliki oleh class Child. Untuk melakukan pewarisan, di class child, kita harus menggunakan kata kunci extends lalu diikuti dengan nama class parent-nya.

Buat dan modifikasi isi file "**Manager.java**", seperti berikut:

Manager.java

```
1  public class Manager {
2    String name;
3
4    void sayHello(String name) {
5      System.out.println("Hi " + name + ", My Name is " + this.name);
6    }
7  }
```

Buat dan modifikasi isi file "**VicePresident.java**", seperti berikut:

VicePresident.java

```
1  public class VicePresident extends Manager {
2  }
```

Analisis kegunaan kode program pada line 1.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      var manager = new Manager();
5      manager.name = "Abdullah";
6      manager.sayHello("Kevin");
7
8      var vp = new VicePresident();
9      vp.name = "Kevin";
10     vp.sayHello("Abdullah");
11
12   }
13 }
```

Analisis kegunaan kode program pada line 4 dan 8. Amati hasilnya pada terminal.

j) Method Overriding

Method overriding adalah kemampuan mendeklarasikan ulang method yang terdapat pada parent class di child class. Saat kita melakukan proses overriding tersebut, secara otomatis ketika kita membuat object dari class child, maka method yang berada di class parent tidak bisa diakses lagi.

Modifikasi isi file "**Manager.java**", seperti berikut:

Manager.java

```
1  public class Manager {
2    String name;
```

Manager.java

```
3
4     void sayHello(String name) {
5       System.out.println("Hi " + name + ", My Name is " + this.name + "
      (Manager)");
6     }
7   }
```

Modifikasi isi file "**VicePresident.java**", seperti berikut:

VicePresident.java

```
1   public class VicePresident extends Manager {
2     void sayHello(String name) {
3       System.out.println("Hi " + name + ", My Name is " + this.name + "
      (Vice President)");
4     }
5   }
```

Analisis pada class apa terjadi overriding dan pada line berapa terjadinya overriding. Amati hasilnya pada terminal.

k) super Keyword

Kadang kita ingin mengakses method yang terdapat di class parent yang sudah terlanjur kita override di class child. Untuk mengakses method milik class parent, kita bisa menggunakan kata kunci super. Sederhananya, keyword super digunakan untuk mengakses class parent. Tidak hanya method, field milik parent class pun bisa kita akses menggunakan kata kunci super.

Buat dan modifikasi isi file "**Shape.java**", seperti berikut:

Shape.java

```
1   public class Shape {
2
3     int getCorner() {
4       return 0;
5     }
6
7   }
8
9   class Rectangle extends Shape {
10
11    int getCorner() {
12      return 4;
13    }
14
15    int getParentCornet() {
16      return super.getCorner();
17    }
18
19  }
```

Analisis kegunaan kode program pada line 16.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

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

App.java

```
3
4       var shape = new Shape();
5       System.out.println(shape.getCorner());
6
7       var rectangle = new Rectangle();
8       System.out.println(rectangle.getCorner());
9       System.out.println(rectangle.getParentCornet());
10
11    }
12  }
```

Analisis kegunaan kode program pada line 8 dan 9. Amati hasilnya pada terminal.

1) Super Constructor

Tidak hanya untuk mengakses method atau field yang ada di parent class, kata kunci super juga bisa digunakan untuk mengakses constructor. Namun syaratnya untuk mengakses parent class constructor, kita harus mengaksesnya di dalam class child constructor. Jika sebuah class parent memiliki constructor yang ada parameter-nya (tidak memiliki default constructor), maka class child wajib mengakses constructor class parent tersebut.

Modifikasi isi file "**Manager.java**", seperti berikut:

Manager.java

```
1   public class Manager {
2     String name;
3     String company;
4
5     Manager(String name) {
6       this(name, null);
7     }
8
9     Manager(String name, String company) {
10      this.name = name;
11      this.company = company;
12    }
13
14    void sayHello(String name) {
15      System.out.println("Hi " + name + ", My Name is " + this.name + "
    (Manager)");
16    }
17  }
```

Modifikasi isi file "**VicePresident.java**", seperti berikut:

VicePresident.java

```
1   public class VicePresident extends Manager {
2
3     VicePresident(String name) {
4       super(name);
5     }
6
7     void sayHello(String name) {
8       System.out.println("Hi " + name + ", My Name is " + this.name + "
    (Vice President)");
9     }
```

VicePresident.java

```
10  }
```

Analisis kegunaan kode program pada line 4.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1   public class App {
2     public static void main(String[] args) {
3
4       var manager = new Manager("Abdullah", "ITDel");
5       manager.sayHello("Kevin");
6
7       var vp = new VicePresident("Kevin");
8       vp.sayHello("Abdullah");
9
10    }
11  }
```

Analisis kegunaan kode program pada line 4 dan 7. Amati hasilnya pada terminal.

m) Object Class

Di java, setiap class yang kita buat secara otomatis adalah turunan dari class Object. Walaupun tidak secara langsung kita eksplisit menyebutkan extends Object, tetapi secara otomatis Java akan membuat class kita extends Object. Bisa dikatakan class Object adalah superclass untuk semua class yang ada di Java.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1   public class App {
2     public static void main(String[] args) {
3
4       var manager = new Manager("Abdullah", "ITDel");
5       manager.sayHello("Kevin");
6
7       var vp = new VicePresident("Kevin");
8       vp.sayHello("Abdullah");
9
10      System.out.println(manager);
11      System.out.println(manager.toString());
12      System.out.println(vp);
13      System.out.println(vp.toString());
14
15    }
16  }
```

Analisis apa tujuan dari kode program pada line 10, 11, 12, dan 13. Amati hasilnya pada terminal.

n) Polymorphism

Polymorphism berasal dari bahasa Yunani yang berarti banyak bentuk. Dalam OOP, Polymorphism adalah kemampuan sebuah object berubah bentuk menjadi bentuk lain. Polymorphism erat hubungannya dengan Inheritance.

Buat dan modifikasi isi file "**Employee.java**", seperti berikut:

Employee.java

```
1  public class Employee {
2
3    String name;
4
5    Employee(String name) {
6      this.name = name;
7    }
8
9    void sayHello(String name) {
10     System.out.println("Hi " + name + ", My Name is " + this.name + "
   (Employee)");
11   }
12
13 }
```

Modifikasi isi file "**Manager.java**", seperti berikut:

Manager.java

```
1  public class Manager extends Employee {
2    String name;
3    String company;
4
5    Manager(String name) {
6      this(name, null);
7    }
8
9    Manager(String name, String company) {
10     super(name);
11
12     this.name = name;
13     this.company = company;
14   }
15
16   void sayHello(String name) {
17     System.out.println("Hi " + name + ", My Name is " + this.name + "
   (Manager)");
18   }
19 }
```

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      Employee employee = new Employee("Abdullah");
5      employee.sayHello("Kevin");
6
7      employee = new Manager("Abdullah");
8      employee.sayHello("Kevin");
9
10     employee = new VicePresident("Abdullah");
11     employee.sayHello("Kevin");
12
13     sayHello(new Employee("Abdullah"));
14     sayHello(new Manager("Kevin"));
15     sayHello(new VicePresident("Aprialdy"));
16   }
17
18   static void sayHello(Employee employee) {
19     System.out.println("Hello " + employee.name);
```

App.java

```
20    }
21  }
```

Analisis tujuan dari kode program pada line 4, 7, 10, 13, 14, 15, dan 18 – 20. Amati hasilnya pada terminal.

o) Type Check dan Casts

Sebelumnya kita sudah tahu cara melakukan konversi tipe data (casts) di tipe data primitif. Casts juga bisa digunakan untuk tipe data bukan primitif. Namun agar aman, sebelum melakukan casts, pastikan kita melakukan type check (pengecekan tipe data), dengan menggunakan kata kunci instanceof. Hasil operator instanceof adalah boolean, true jika tipe data sesuai, false jika tidak sesuai.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1   public class App {
2     public static void main(String[] args) {
3
4       Employee employee = new Employee("Abdullah");
5       employee.sayHello("Kevin");
6
7       employee = new Manager("Abdullah");
8       employee.sayHello("Kevin");
9
10      employee = new VicePresident("Abdullah");
11      employee.sayHello("Kevin");
12
13      sayHello(new Employee("Abdullah"));
14      sayHello(new Manager("Kevin"));
15      sayHello(new VicePresident("Aprialdy"));
16    }
17
18    static void sayHello(Employee employee) {
19      if (employee instanceof VicePresident) {
20        VicePresident vicePresident = (VicePresident) employee;
21        System.out.println("Hello VP " + vicePresident.name);
22      } else if (employee instanceof Manager) {
23        Manager manager = (Manager) employee;
24        System.out.println("Hello Manager " + manager.name);
25      } else {
26        System.out.println("Hello " + employee.name);
27      }
28    }
29  }
```

Analisis tujuan dari kode program pada line 19, 20, 22 dan 23. Amati hasilnya pada terminal.

p) Variable Hiding

Variabel hiding merupakan masalah yang terjadi ketika kita membuat nama field yang sama di class child dengan nama field di class parent. Tidak ada yang namanya field overriding, ketika kita membuat ulang nama field di class child, itu disebut sebagai variable hiding. Untuk mengatasi variable

hiding, kita bisa menggunakan keyword super. Yang membedakan variabel hiding dan method overriding adalah ketika sebuah object di casts. Saat object di casts, method akan tetap mengakses method overriding, namun variable akan mengakses variable yang ada di class saat ini.

Buat dan modifikasi isi file "**Parent.java**", seperti berikut:

Parent.java

```
1  public class Parent {
2    String name;
3
4    void info() {
5      System.out.println("Ini adalah class Parent");
6    }
7  }
8
9  class Child extends Parent {
10   String name;
11
12   void info() {
13     System.out.println("Ini adalah class Child");
14     System.out.println("Nama parent class adalah " + super.name);
15   }
16 }
```

Analisis apakah field pada line 2 dan 10 sama. Apa tujuan kode program pada line 14.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      Child child = new Child();
5      child.name = "Abdullah";
6      child.info();
7      System.out.println(child.name);
8
9      Parent parent = child; // Boleh langsung tanpa casts, karena class
   child anak dari class parent
10     parent.info();
11     System.out.println(parent.name);
12   }
13
14 }
```

Analisis hasil kode program pada line 7 dan 11 dan mengapa hasilnya bisa seperti itu. Amati hasilnya pada terminal.

(!) Dikarenakan class Child dibuat di dalam file class Parent maka cara compilenya seperti berikut:

```
pbo-praktikum3-11s18005> javac App.java Parent.java
```

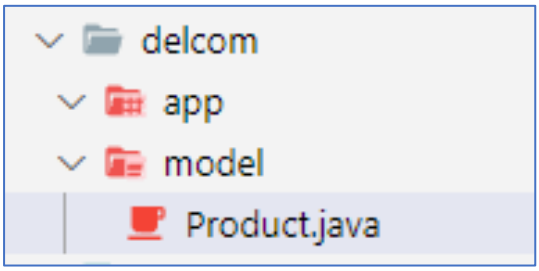
Selanjutnya, setelah melakukan compile jalankan program java seperti biasa.

```
pbo-praktikum3-11s18005> java App
```

q) Package

Saat kita membuat aplikasi, bisa dipastikan kita akan banyak sekali membuat class. Jika class terlalu banyak, kadang akan menyulitkan kita untuk mencari atau mengklasifikasikan jenis-jenis class. Java memiliki fitur package, yaitu fitur mirip folder / direktori, dimana kita bisa menyimpan class-class kita di dalam package. Sama seperti folder / direktori, package juga bisa nested, kita bisa menggunakan tanda titik (.) untuk membuat nested package. Ketika kita menyimpan class di dalam package, maka di atas file Java-nya, wajib untuk kita menyebutkan nama package-nya.

Buat struktur folder dan file berikut di VSCode kamu:



Modifikasi isi file **"delcom/model/Product.java"**, seperti berikut:

Product.java

```
1  package delcom.model;
2
3  public class Product {
4    String name;
5    int price;
6
7    public Product(String name, int price) {
8      this.name = name;
9      this.price = price;
10   }
11 }
```

Analisis tujuan kode program pada line 1.
(!) Tidak perlu dijalankan di terminal untuk saat ini.

r) Access Modifier

Access Modifier adalah kemampuan membuat class, field, method dan constructor dapat diakses dari mana saja. Sebelumnya kita telah menggunakan 2 access modifier, yaitu public dan default (no-modifier).

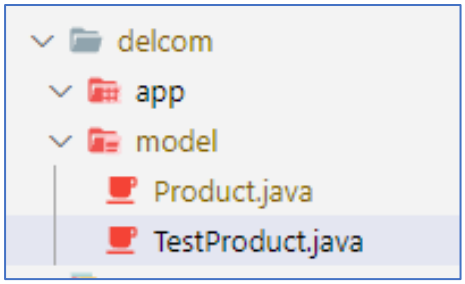
| Access Level

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

| Public Class

Saat kita membuat public class, kita hanya bisa membuat 1 public class di 1 file java. Selain itu, nama public class harus sama dengan nama file.

Buat file baru dengan nama "delcom/model/**TestProduct.java**", seperti berikut:



Modifikasi isi file "**delcom/model/Product.java**", seperti berikut:

Product.java

```
1  package delcom.model;
2
3  public class Product {
4    private String name;
5    protected int price;
6
7    public Product(String name, int price) {
8      this.name = name;
9      this.price = price;
10   }
11 }
```

Analisis tujuan kode program pada line 4 dan 5. Apakah field pada line 4 dapat diakses di class TestProduct.java

Modifikasi isi file "**delcom/model/TestProduct.java**", seperti berikut:

TestProduct.java

```
1  package delcom.model;
2
3  public class TestProduct {
4    public static void main(String[] args) {
5      Product product = new Product("Sari Gandum", 10_000);
6      product.price = 12_000;
7      System.out.println(product.price);
8    }
9  }
```

Analisis kenapa field price dapat diakses pada line 6. Amati hasilnya pada terminal.

- Compile

```
> javac ./delcom/model/TestProduct.java ./delcom/model/Product.java
```

- Run

```
> java delcom.model.TestProduct
```

s) Import

Import adalah kemampuan untuk menggunakan class yang berbeda di package yang berbeda. Syarat class yang bisa digunakan jika package-nya berbeda adalah class-nya harus public.

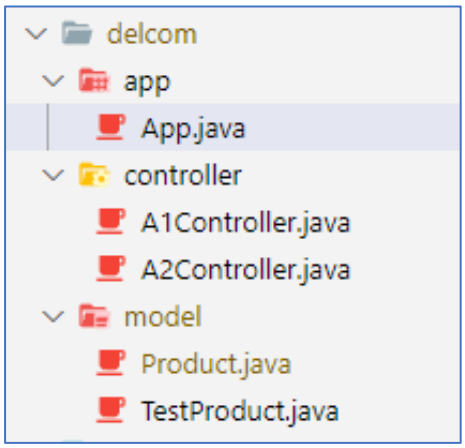
| Import Semua Package

Jika kita ingin mengimport semua class di dalam sebuah package, kita bisa menggunakan simbol bintang (*). Contoh: **import java.util.*;**

| Default Import

Secara default, semua class yang ada di package java.lang sudah auto import, jadi kita tidak perlu melakukan import secara manual. Contoh class String, Integer, Long, Boolean, dan lain-lain terdapat di package java.lang. Oleh karena itu, kita tidak perlu melakukan import secara manual.

Buat stuktur folder dan file seperti berikut pada VSCode:



Modifikasi isi file "**delcom/controller/A1Controller.java**", seperti berikut:

A1Controller.java

```
1  package delcom.controller;
2
3  public class A1Controller {
4
5  }
```

Modifikasi isi file "**delcom/controller/A2Controller.java**", seperti berikut:

A2Controller.java

```
1  package delcom.controller;
2
3  public class A2Controller {
4
5  }
```

Modifikasi isi file "**delcom/app/App.java**", seperti berikut:

App.java

```
1  package delcom.app;
2
3  import delcom.controller.*;
4  import delcom.model.Product;
5
6  public class App {
7
8    A1Controller a1 = new A1Controller();
9    A2Controller a2 = new A2Controller();
10
11   Product product = new Product("Indomie", 3_000);
12   // product.price = 5_000;
13
14 }
```

Analisis tujuan kode program pada line 3 dan 4. Apa yang terjadi jika komentar pada line 12 di hilangkan dan mengapa hal tersebut dapat terjadi.

t) Abstract Class

Saat kita membuat class, kita bisa menjadikan sebuah class sebagai abstract class. Abstract class artinya, class tersebut tidak bisa dibuat sebagai object secara langsung, hanya bisa diturunkan. Untuk membuat sebuah class menjadi abstract, kita bisa menggunakan kata kunci abstract sebelum kata kunci class. Dengan demikian abstract class bisa kita gunakan sebagai kontrak untuk class child.

Buat dan modifikasi isi file "**Location.java**", seperti berikut:

Location.java

```
1  public abstract class Location {
2    public String name;
3  }
4
5  class City extends Location {
6
7  }
```

Analisis tujuan kode program pada line 1.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      // var location = new Location();
5      var city = new City();
6      city.name = "Balige";
7      System.out.println(city.name);
8
9    }
10
11 }
```

Analisis apa yang terjadi jika kode program pada line 4 dihilangkan komentarnya. Amati hasilnya pada terminal.

- Compile

```
> javac App.java Location.java
```

- Run

```
> java App
```

u) Abstract Method

Saat kita membuat class yang abstract, kita bisa membuat abstract method juga di dalam class abstract tersebut. Saat kita membuat sebuah abstract method, kita tidak boleh membuat block method untuk method tersebut. Artinya, abstract method wajib di override di class child. Abstract method tidak boleh memiliki access modifier private.

Buat dan modifikasi isi file "**Animal.java**", seperti berikut:

Animal.java

```
1  public abstract class Animal {
2    public String name;
3
4    public abstract void run();
5  }
```

Analisis kegunaan kode program pada line 1 dan 4.

Buat dan modifikasi isi file "**Cat.java**", seperti berikut:

Cat.java

```
1  public class Cat extends Animal {
2    public void run() {
3      System.out.println("Cat " + name + " is run");
4    }
5  }
```

Analisis apa yang terjadi apabila kode program pada line 2 sampai 4 dikomentari.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      Animal animal = new Cat();
5      animal.name = "Anggora";
6      animal.run();
7
8    }
9
10 }
```

Analisis kegunaan kode program pada line 4, 5, dan 6. Amati hasilnya pada terminal.

v) Getter dan Setter

Encapsulation artinya memastikan data sensitive sebuah object tersembunyi dari akses luar. Hal ini bertujuan agar kita bisa menjaga agar data sebuah object tetap baik dan valid. Untuk mencapai ini, biasanya kita akan membuat semua field menggunakan access modifier private, sehingga tidak bisa diakses atau diubah dari luar. Agar bisa diubah, kita akan menyediakan method untuk mengubah dan mendapatkan field tersebut.

| Getter dan Setter

Di java, proses encapsulation sudah dibuat standarisasinya, dimana kita bisa menggunakan Getter dan Setter method. Getter adalah function yang dibuat untuk mengambil data field. Setter adalah function untuk mengubah data field.

| Getter dan Setter Method

Tipe Data	Getter Method	Setter Method
boolean	isName()	setName(boolean value)
primitif	getName()	setName(primitif value)
Object	getName()	setName(Object value)

Buat dan modifikasi isi file "**Category.java**", seperti berikut:

Category.java

```
1  public class Category {
2    private String id;
3    private boolean expensive;
4
5    public String getId() {
6      return id;
7    }
8
9    public void setId(String id) {
10     if (id != null) {
11       this.id = id;
12     }
13   }
14
15   public boolean isExpensive() {
16     return expensive;
17   }
18
19   public void setExpensive(boolean expensive) {
20     this.expensive = expensive;
21   }
22
23 }
```

Analisis kegunaan kode program pada line 5-7, 9-13, 15-17, dan 19-21.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      var category = new Category();
5      category.setId("123456");
6      category.setId(null);
7
8      System.out.println(category.getId());
9
10   }
11
12 }
```

Analisis kegunaan kode program pada line 5, 6, dan 8. Apakah output dari kode program pada line 8. Amati hasilnya pada terminal.

w) Interface

Sebelumnya kita sudah tahu bahwa abstract class bisa digunakan sebagai kontrak untuk class child-nya. Namun sebenarnya yang lebih tepat untuk kontrak adalah interface. Jangan salah sangka bahwa Interface disini bukanlah User Interface. Interface mirip seperti abstract class, yang membedakan adalah di interface, semua method otomatis abstract (tidak memiliki block / body). Di interface kita tidak boleh memiliki field, kita hanya boleh memiliki constant (field yang tidak bisa diubah). Untuk mewariskan interface, kita menggunakan keyword implements.

Buat dan modifikasi isi file "**Car.java**", seperti berikut:

Car.java

```
1  public interface Car {
2    void drive();
3
4    int getTier();
5  }
```

Analisi kegunaan kode program pada line 1.

Buat dan modifikasi isi file "**Avanza.java**", seperti berikut:

Avanza.java

```
1  public class Avanza implements Car {
2
3    public void drive() {
4      System.out.println("Avanza Drive");
5    }
6
7    public int getTier() {
8      return 4;
9    }
10
11 }
```

Analisis kegunaan kode program pada line 3-5 dan 7-9.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      Car car = new Avanza();
5      System.out.println(car.getTier());
6      car.drive();
7
8    }
9  }
```

Amati hasilnya pada terminal.

x) Interface Inheritance

Sebelumnya kita tahu bahwa child class hanya bisa punya 1 class parent. Namun berbeda dengan interface, sebuah child class bisa melakukan implement lebih dari 1 interface. Bahkan interface pun bisa implement interface lain dan bisa lebih dari 1. Namun jika interface ingin mewarisi interface lain, kita menggunakan kata kunci extends, bukan implements.

Buat dan modifikasi isi file "**HasBrand.java**", seperti berikut:

HasBrand.java

```
1  public interface HasBrand {
2    String getBrand();
3  }
```

Buat dan modifikasi isi file "**IsMaintenance.java**", seperti berikut:

IsMaintenance.java

```
1  public interface IsMaintenance {
2    boolean isMaintenance();
3  }
```

Modifikasi isi file "**Car.java**", seperti berikut:

Car.java

```
1  public interface Car extends HasBrand, IsMaintenance {
2    void drive();
3
4    int getTier();
5  }
```

Analisi kegunaan kode program pada line 1.

Modifikasi isi file "**Avanza.java**", seperti berikut:

Avanza.java

```
1  public class Avanza implements Car, Catalog {
2
3    public void drive() {
4      System.out.println("Avanza Drive");
5    }
6
```

Avanza.java

```
7     public int getTier() {
8       return 4;
9     }
10
11    public String getBrand() {
12      return "Toyota";
13    }
14
15    public boolean isMaintenance() {
16      return false;
17    }
18
19    public int getPrice() {
20      return 100_000_000;
21    }
22
23  }
```

Analisis kegunaan kode program pada line 3-5 dan 7-9.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1   public class App {
2     public static void main(String[] args) {
3
4       Car car = new Avanza();
5       System.out.println(car.getTier());
6       System.out.println(car.getBrand());
7       // System.out.println(car.getPrice());
8       System.out.println(((Avanza) car).getPrice());
9       car.drive();
10
11    }
12  }
```

Analisis tujuan kode program pada line 8. Analisis apa yang terjadi jika kode program pada line 7 dihilangkan komentarnya. Amati hasilnya pada terminal.

y) Default Method

Sebelumnya kita tahu bahwa di interface, kita tidak bisa membuat method konkrit yang memiliki block method. Namun sejak versi Java 8, ada fitur default method di interface. Fitur ini terjadi karena sulit untuk melakukan maintenance kontrak interface jika sudah terlalu banyak class yang implement interface tersebut. Ketika kita menambahkan satu method di interface, secara otomatis semua class yang implement akan rusak karena harus meng-override method tersebut. Dengan menggunakan default method, kita bisa menambahkan konkrit method di interface.

Modifikasi isi file "**Car.java**", seperti berikut:

Car.java

```
1   public interface Car extends HasBrand, IsMaintenance {
2     void drive();
3
4     int getTier();
```

Car.java

```
5
6     default boolean isBig() {
7       return false;
8     }
9
10    default boolean isMaintenance() {
11      return false;
12    }
13  }
```

Analisis kegunaan kode program pada line 6-8 dan 10-12.

Buat dan modifikasi isi file "**Bus.java**", seperti berikut:

Bus.java

```
1   public class Bus implements Car {
2
3     public void drive() {
4       System.out.println("Bus Drive");
5     }
6
7     public int getTier() {
8       return 8;
9     }
10
11    public String getBrand() {
12      return "Hino";
13    }
14
15    public boolean isMaintenance() {
16      return false;
17    }
18
19    public boolean isBig() {
20      return true;
21    }
22
23  }
```

Analisis kegunaan kode program pada line 19-21.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1   public class App {
2     public static void main(String[] args) {
3
4       Car car = new Avanza();
5       System.out.println(car.isBig());
6       car.drive();
7
8       car = new Bus();
9       System.out.println(car.isBig());
10      car.drive();
11
12    }
13  }
```

Analisis hasil kode program pada line 5 dan 9. Amati hasilnya pada terminal.

z) toString() Method

toString() adalah method yang terdapat di class Object. Method ini biasanya digunakan untuk merepresentasikan object dalam bentuk String. Secara default, method toString() akan mengembalikan nilai dari "**namaclass + @ + hashcode**". Namun kita bisa mengubahnya jika kita mau, agar object yang kita buat lebih mudah dibaca.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2    String name;
3    String address;
4    final String country = "Indonesia";
5
6    Person() {
7      this(null);
8    }
9
10   Person(String paramName) {
11     this(paramName, null);
12   }
13
14   Person(String name, String address) {
15     this.name = name;
16     this.address = address;
17   }
18
19   public String toString() {
20     return "Person name: " + name + ", address: " + address + ", country:
   " + country;
21   }
22 }
```

Analisis kegunaan kode program pada line 19-21.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      Person person = new Person("Abdullah", "Sitoluama");
5      System.out.println(person);
6
7    }
8  }
```

Analisis hasil dari kode program pada line 5. Amati hasilnya pada terminal.

aa) equals() Method

Hal yang agak membingunkan di Java adalah, cara membandingkan object. Di bahasa pemrograman lain, untuk mengecek apakah sebuah objek sama, biasanya menggunakan operator "==", di Java, operator "==" hanya untuk mengecek data primitf. Untuk non primitive pengecekannya menggunakan method equals. Dana secara default, method akan membandingkan dua object secara

kesamaan posisi object di memory, artinya jika kita membuat 2 object yang isi field-nya sama, tetap dianggap berbeda oleh method equals. Oleh karena itu, ada baiknya kita meng-override method equals milik class Object tersebut.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2    String name;
3    int age;
4    final String country = "Indonesia";
5
6    Person(String name, int age) {
7      this.name = name;
8      this.age = age;
9    }
10
11   public String toString() {
12     return "Person name: " + name + ", age: " + age + ", country: " +
   country;
13   }
14
15   public boolean equals(Object o) {
16     if (o == this) {
17       return true;
18     }
19
20     if (!(o instanceof Person)) {
21       return false;
22     }
23
24     Person person = (Person) o;
25
26     if (this.age != person.age) {
27       return false;
28     }
29
30     if (!this.name.equals(person.name)) {
31       return false;
32     }
33
34     return true;
35   }
36 }
```

Analisis kegunaan kode program pada line 15 – 36.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      Person p1 = new Person("Abdullah", 23);
5      Person p2 = new Person("Abdullah", 23);
6      Person p3 = new Person("Abdullah", 20);
7      System.out.println(p1.equals(p2));
8      System.out.println(p1.equals(p3));
9
10   }
11 }
```

Analisis hasil kode program pada line 7 dan 8. Amati hasilnya pada terminal.

bb) hashCode() Method

Method hasCode adalah method representasi integer dari object, mirip seperti toString yang merupakan representasi String, hashCode adalah representasi integer. HashCode sangat bermanfaat untuk membuat struktur data unique seperti HashMap, Set dan lain-lain, karena cukup menggunakan hashCode method untuk mendapatkan identitas unique object kita. Secara default hashCode akan mengembalikan nilai integer sesuai data di memory, namun kita bisa melakukan override jika kita ingin.

Tidak mudah melakukan override method hashCode, karena ada kontraknya. Sebanyak apapun hashCode dipanggil, untuk object yang sama, harus mengembalikan data integer yang sama. Jika ada 2 object yang sama jika dibandingkan menggunakan method equals, maka nilai hashCode juga harus sama. Tidak wajib hashCode berbeda jika method equals menghasilkan false, karena memang keterbatasan jumlah integer sekitar 2 milyar.

Modifikasi isi file "**Person.java**", seperti berikut:

Person.java

```
1  class Person {
2    String name;
3    int age;
4    final String country = "Indonesia";
5
6    Person(String name, int age) {
7      this.name = name;
8      this.age = age;
9    }
10
11   public String toString() {
12     return "Person name: " + name + ", age: " + age + ", country: " +
   country;
13   }
14
15   public int hashCode() {
16     int result = name != null ? name.hashCode() : 0;
17     result = 31 * result + age;
18     return result;
19   }
20
21   public boolean equals(Object o) {
22     if (this == o)
23       return true;
24
25     if (o == null || getClass() != o.getClass())
26       return false;
27
28     Person person = (Person) o;
29
30     if (age != person.age)
31       return false;
32
33     return name != null ? name.equals(person.name) : person.name == null;
34   }
35 }
```

Analisis kegunaan kode program pada line 15 – 19.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      Person p1 = new Person("Abdullah", 23);
5      Person p2 = new Person("Abdullah", 23);
6      Person p3 = new Person("Abdullah", 20);
7
8      System.out.println(p1.equals(p2));
9      System.out.println(p1.hashCode() == p2.hashCode());
10
11   }
12 }
```

Analisis hasil dari kode program pada line 8 dan 9. Amati hasilnya pada terminal.

cc) Final Class

Sebelumnya kita pernah menggunakan kata kunci final di Java. Jika digunakan di variabel, maka variabel tersebut tidak bisa berubah lagi datanya. Final pun bisa digunakan di class, dimana jika kita membuat kata kunci final sebelum class, maka kita menandakan bahawa class tersebut tidak bisa diwariskan lagi. Secara otomatis semua class child-nya akan error.

Buat dan modifikasi isi file "**SocialMedia.java**", seperti berikut:

SocialMedia.java

```
1  public class SocialMedia {
2    String name;
3  }
4
5  final class Facebook extends SocialMedia {
6  }
7
8  // class FakeFacebook extends Facebok{}
```

Analisis kegunaan kode program pada line 5. Analisis mengapa kode program pada line 8 mengalami error.
(!) Tidak perlu menjalankan pada terminal.

dd) Final Method

Kata kunci final juga bisa digunakan di Method. Jika sebuah method di tambahkan kata kunci final, maka artinya method tersebut tidak bisa di override lagi di class child-nya. Hal ini cocok jika ingin mengimplementasikan sebuah method agar tidak dapat diubah lagi oleh class child-nya.

Modifikasi isi file "**SocialMedia.java**", seperti berikut:

SocialMedia.java

```
1  public class SocialMedia {
2    String name;
```

SocialMedia.java

```
3   }
4
5   class Facebook extends SocialMedia {
6     final void login(String username, String password) {
7       // isi method
8     }
9   }
10
11  class FakeFacebook extends Facebook {
12    // void login(String username, String password) {
13    // // isi method
14    // }
15  }
```

Analisis kegunaan kode program pada line 6. Analisis mengapa kode program pada line 12 - 14 mengalami error.
(!) Tidak perlu menjalankan pada terminal.

ee) Inner Class

Di Java, kita bisa membuat class di dalama class, atau disebut dengan Inner Class. Salah satu kasus kita membuat inner class biasanya Ketika kita butuh membuat beberapa class yang saling berhubungan, dimana sebuah class tidak bisa dibuat tanpa class lain. Misal kita perlu membuat class Employee, dimana membutuhkan class Company, maka kita bisa membuat class Employee sebagai inner class Company. Cara membuat inner class, cukup membuatnya di dalam block class outer class-nya.

| Mengakses Outer Class

Keuntungan saat kita membuat inner class adalah, kemampuan untuk mengakses outer class-nya. Inner class bisa membaca semua private member yang ada di outer class-nya. Untuk mengakses object outer class-nya, kita bisa menggunakan nama class outer-nya diikuti dengan kata kunci this, misal Company.this dan untuk mengakses super class outer class-nya, kita bisa menggunakan nama class outer-nya diikuti dengan kata kunci super, misal Company.super

Buat dan modifikasi isi file "**Company.java**", seperti berikut:

Company.java

```
1   public class Company {
2     private String name;
3
4     public String getName() {
5       return name;
6     }
7
8     public void setName(String name) {
9       this.name = name;
10    }
11
12    public class Employee {
13      private String name;
14
15      public String getName() {
```

Company.java

```
16        return name;
17      }
18
19      public void setName(String name) {
20        this.name = name;
21      }
22
23      public String getCompany() {
24        return Company.this.name;
25      }
26    }
27  }
```

Analisis kegunaan kode program pada line 12 dan 24.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1   public class App {
2     public static void main(String[] args) {
3
4       Company company = new Company();
5       company.setName("IT DEL");
6
7       Company.Employee employee = company.new Employee();
8       employee.setName("Abdullah");
9
10      System.out.println(employee.getName());
11      System.out.println(employee.getCompany());
12
13      Company company2 = new Company();
14      company2.setName("BCA");
15
16      Company.Employee employee2 = company2.new Employee();
17      employee2.setName("Kevin");
18
19      System.out.println(employee2.getName());
20      System.out.println(employee2.getCompany());
21
22    }
23  }
```

Analisis kegunaan kode program pada line 7 dan 16. Analisis hasil kode program pada line 11 dan 20. Amati hasilnya pada terminal.

ff) Anonymous Class

Anonymous class atau class tanpa nama. Adalah kemampuan mendeklarasikan class, sekaligus meng-instansiasi object-nya secara langsung. Anonymous class sebenarnya termasuk inner class, dimana outer class-nya adalah tempat dimana kita membuat anonymous class tersebut. Anonymous class sangat cocok ketika kita berhadapan dengan kasus membuat implementasi interface atau abstract class sederhana, tanpa harus membuat implementasi class-nya.

Buat dan modifikasi isi file "**Hello.java**", seperti berikut:

Hello.java

```
1  public interface Hello {
2    void sayHello();
3
4    void sayHello(String name);
5  }
```

Modifikasi isi file "**App.java**", seperti berikut:

App.java

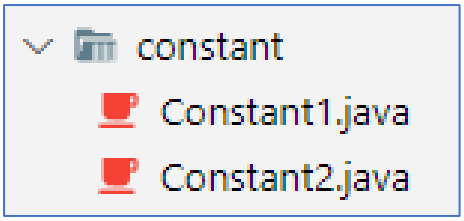
```
1  public class App {
2    public static void main(String[] args) {
3
4      Hello hello = new Hello() {
5
6        public void sayHello() {
7          System.out.println("Hello");
8        }
9
10       public void sayHello(String name) {
11         System.out.println("Hello " + name);
12       }
13
14     };
15
16     hello.sayHello();
17     hello.sayHello("Abdullah");
18
19   }
20 }
```

Analisis kegunaan kode program pada line 4 – 14. Analisis hasil dari kode program pada line 16 dan 17. Amati hasilnya pada terminal.

gg) static Keyword

Static field, atau disebut class variable, artinya field tersebut bisa diakses langsung tanpa membuat object terlebih dahulu. Static method, atau disebut class method, artinya method tersebut bisa diakses langsung tanpa membuat object terlebih dahulu. Static block akan otomatis dieksekusi ketika sebuah class di load. Static Inner Class, artinya inner class tersebut bisa diakses secara langsung tanpa harus membuat object outer class terlebih dahulu. Static pada inner class menyebabkan kita tidak bisa mengakses lagi object outer class-nya.

Buat package baru dengan nama "**constant**", selanjutnya tambahkan 2 file baru di dalamnya dengan nama "**Constant1.java**" dan "**Constant2.java**", seperti berikut:



Modifikasi isi file "**constant/Constant1.java**", seperti berikut:

Constant1.java

```
1  package constant;
2
3  public class Constant1 {
4    public static final String APP = "Bahasa Pemrograman Berorientasi
   Objek";
5    public static final Integer VERSION = 1;
6  }
```

Analisis kegunaan kode program pada line 4 dan 5.

Modifikasi isi file "**constant/Constant2.java**", seperti berikut:

Constant2.java

```
1  package constant;
2
3  public class Constant2 {
4    public static final int PROCESSOR;
5
6    static {
7      PROCESSOR = Runtime.getRuntime().availableProcessors();
8    }
9  }
```

Analisis kegunaan kode program pada line 4 dan 6 – 8.

Modifikasi isi file "**Constant.java**", seperti berikut:

Constant.java

```
1  public class Constant {
2    public static final double phi = (22 / 7.0);
3  }
```

Modifikasi isi file "**MathUtil.java**", seperti berikut:

MathUtil.java

```
1  public class MathUtil {
2
3    public static int sum(int... values) {
4      int total = 0;
5      for (var value : values) {
6        total += value;
7      }
8      return total;
9    }
10
11 }
```

Analisis kegunaan kode program pada line 3 – 9.

Modifikasi isi file "**Country.java**", seperti berikut:

Country.java

```
1  public class Country {
2    private String name;
3
4    public String getName() {
5      return name;
6    }
7
8    public void setName(String name) {
9      this.name = name;
10   }
11
12   public static class City {
13     private String name;
14
15     public String getName() {
16       return name;
17     }
18
19     public void setName(String name) {
20       this.name = name;
21     }
22   }
23
24 }
```

Analisis kegunaan kode program pada line 12.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  import static constant.Constant1.*;
2  import static constant.Constant2.PROCESSOR;
3
4  public class App {
5    public static void main(String[] args) {
6
7      System.out.println(APP);
8      System.out.println(VERSION);
9      System.out.println(PROCESSOR);
10     System.out.println(Constant.phi);
11
12     int sum = MathUtil.sum(1, 2, 3, 4, 5);
13     System.out.println(sum);
14
15     Country.City city = new Country.City();
16     city.setName("Sitoluama");
17     System.out.println(city.getName());
18
19   }
20 }
```

Analisis tujuan kode program pada line 1, 2, 12 dan 15. Analisis hasil dari kode program pada line 7, 8, 9, 10, 13 dan 17. Amati hasilnya pada terminal.

hh) Record Class

Fitur ini tersedia untuk versi Java 14 ke atas. Kadang kita sering membuat class, hanya untuk class yang berisikan data. Hanya berisi getter, equals, hashCode, dan toString method. Record class digunakan untuk mempermudah pembuatan jenis

class tersebut. Saat kita membuat record class, secara otomatis Java akan membuatkan getter, equals, hashCode dan toString method secara otomatis, dan constructor secara otomatis. Saat membuat record class, secara otomatis kita akan meng-extends class java.lang.Record yang artinya kita tidak bisa extends class lain. Namun kita tetap bisa meng-implement interface.

| Record Class Constructor

Secara default, constructor di record cass akan dibuat secara otomatis, sesuai dengan definisi record class parameter. Namun jika kita ingin melakukan sesuatu di constructor tersebut, kita bisa membuat compact constructor, yaitu constructor tanpa tanda (). Selain itu, kita juga bisa melakukan constructor overloading, namun ada syaratnya, yaitu constructor overloading-nya harus tetap memanggil constructor utama yang secara otomatis dibuat di record class.

Modifikasi isi file "**LoginRequest.java**", seperti berikut:

LoginRequest.java

```
1  public record LoginRequest(String username, String password) {
2
3    public LoginRequest {
4      System.out.println("Membuat object LoginRequest");
5    }
6
7    public LoginRequest(String username) {
8      this(username, "");
9    }
10
11   public LoginRequest() {
12     this("", "");
13   }
14
15 }
```

Analisis kegunaan kode program pada line 1, 3-5, 8, dan 12.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1  public class App {
2    public static void main(String[] args) {
3
4      LoginRequest loginRequest = new LoginRequest("abdullah", "***");
5
6      System.out.println(loginRequest.username());
7      System.out.println(loginRequest.password());
8      System.out.println(loginRequest);
9
10     System.out.println(new LoginRequest());
11     System.out.println(new LoginRequest("abdullah"));
12     System.out.println(new LoginRequest("abdullah", "***"));
13
14   }
15 }
```

Analisis hasil dari kode program pada line 10, 11, dan 12. Amati hasilnya pada terminal.

ii) Enum Class

Saat kita membuat aplikasi, kadang kita akan bertemu dengan jenis-jenis data yang nilainya terbatas. Misal, gender, ada male dan female, atau tipe customer, ada standard, premium atau vip, dan lain-lain. Dalam kasus seperti ini, kita bisa menggunakan enum class, yaitu class yang berisikan nilai terbatas yang sudah ditentukan. Saat membuat enum class, secara otomatis dia akan meng-extends calss java.lang.Enum, oleh karena itu class enum tidak bisa extends class lain, namun masih tetap bisa implements interface.

| Enum Members

Sama seperti class biasanya, di class enum pun kita bisa menambahkan members (field, method dan constructor). Khusus constructor, kita tidak bisa membuat public constructor, karena memang tujuan enum bukan untuk di instansiasi secara bebas.

Buat dan modifikasi isi file "**Level.java**", seperti berikut:

Level.java

```
1  public enum Level {
2    STANDARD("Standard Level"),
3    PREMIUM("Premium Level"),
4    VIP("VIP Level");
5
6    private String description;
7
8    Level(String description) {
9      this.description = description;
10   }
11
12   public String getDescription() {
13     return description;
14   }
15 }
```

Analisis kode program pada line 1-15.

Buat dan modifikasi isi file "**Customer.java**", seperti berikut:

Customer.java

```
1  public class Customer {
2    private String name;
3    private Level level;
4
5    public String getName() {
6      return name;
7    }
8
9    public void setName(String name) {
10     this.name = name;
11   }
```

Customer.java

```
12
13    public Level getLevel() {
14      return level;
15    }
16
17    public void setLevel(Level level) {
18      this.level = level;
19    }
20
21  }
```

Analisis kode program pada line 3 dan 17.

Modifikasi isi file "**App.java**", seperti berikut:

App.java

```
1   public class App {
2     public static void main(String[] args) {
3
4       Customer customer = new Customer();
5       customer.setName("Abdullah");
6       customer.setLevel(Level.VIP);
7
8       System.out.println(customer.getName());
9       System.out.println(customer.getLevel());
10      System.out.println(customer.getLevel().getDescription());
11
12      // Konversi Enum ke String
13      Level level = Level.valueOf("VIP");
14      System.out.println(level);
15      System.out.println(Level.STANDARD.name());
16
17      System.out.println("Print Level:");
18      for (var value : Level.values()) {
19        System.out.println("- " + value);
20      }
21
22    }
23  }
```

Analisis kode program pada line 6, 9, 10, 13, 15, 18 - 20. Amati hasilnya pada terminal.

2. Mengimplementasikan Class Diagram

Class diagram adalah diagram yang digunakan untuk menggambarkan struktur kelas-kelas dalam suatu sistem berbasis objek. Class diagram membantu dalam merancang dan memahami bagaimana kelas-kelas saling terhubung dalam sistem dan membantu pengembang dalam memvisualisasikan desain perangkat lunak dengan lebih jelas.

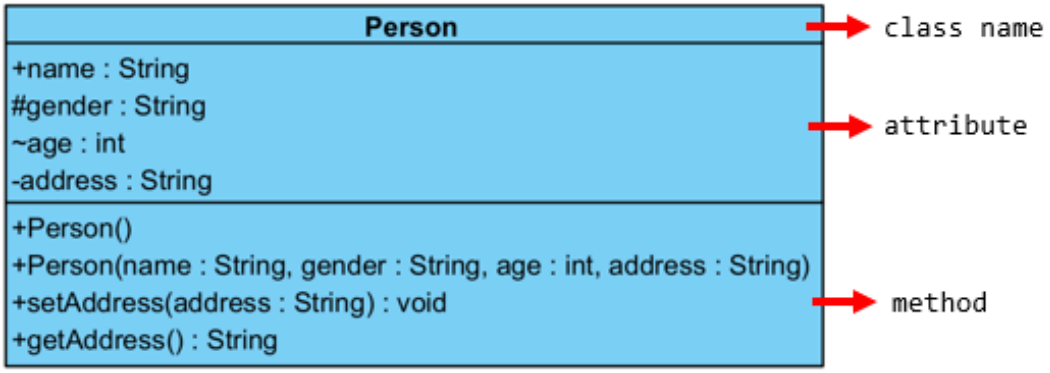
Visual Paradigm merupakan Tools yang dapat digunakan untuk membuat Class Diagram. Silahkan download tools tersebut pada link berikut:
https://www.visual-paradigm.com/download/community.jsp

a) Class Name, Attribute dan Method

Dalam class diagram, setiap kelas direpresentasikan dengan kotak yang berisi nama kelas di atasnya, atribut di tengah, dan metode di bawahnya. Saat menuliskan attribute dan method, terdapat simbol access visibility yang perlu diperhatikan:

Hak Akses	Public (+)	Package (~)	Protected (#)	Private (-)
Anggota dari kelas yang sama	Y	Y	Y	Y
Anggota kelas turunan	Y	Y	Y	N
Anggota dari kelas lain	Y	dalam paket yang sama	N	N

- Class diagram



- Kode program

Person.java

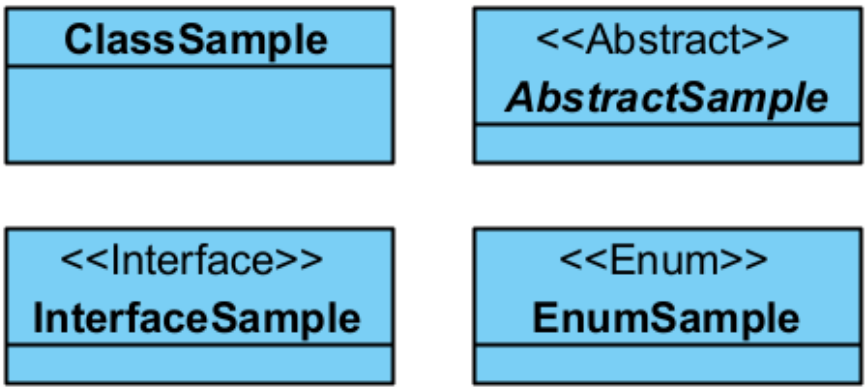
```
1  public class Person {
2    public String name;
3    protected String gender;
4    int age; // access modifier: default atau package
```

Person.java

```
5     private String address;
6
7     public Person() {
8     }
9
10    public Person(String name, String gender, int age, String address) {
11      this.name = name;
12      this.gender = gender;
13      this.age = age;
14      this.address = address;
15    }
16
17    public void setAddress(String address) {
18      this.address = address;
19    }
20
21    public String getAddress() {
22      return this.address;
23    }
24  }
```

b) Class, Class Abstract, Interface dan Enum

- Class diagram



- Kode program

ClassSample.java

```
1   public class ClassSample {
2
3   }
```

AbstractSample.java

```
1   public abstract class AbstractSample {
2
3   }
```

InterfaceSample.java

```
1   public interface InterfaceSample {
2
3   }
```

EnumSample.java

```
1   public enum EnumSample {
```

EnumSample.java

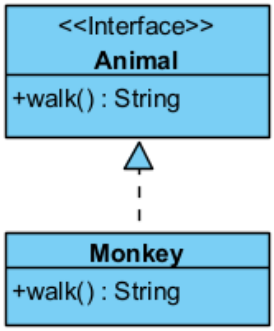
```
2
3   }
```

Class Relationships

Sebuah kelas mungkin terlibat dalam satu atau lebih hubungan dengan kelas lain. Suatu relasi dapat berupa salah satu dari jenis berikut:

c) Realisasi

Realisasi merupakan penerapan konkret dari suatu abstraksi seperti interface atau kelas abstrak oleh kelas yang mewujudkannya.



- Kode program

Animal.java

```
1   public interface Animal {
2     public String walk();
3   }
```

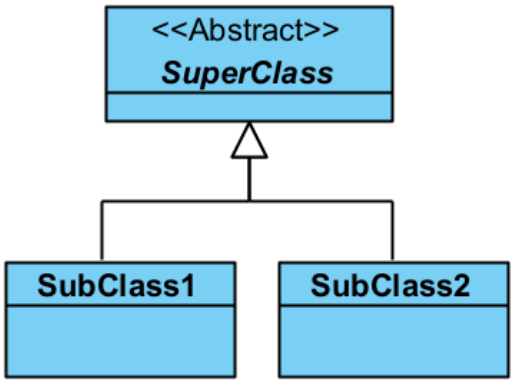
Monkey.java

```
1   public class Monkey implements Animal {
2
3     @Override
4     public String walk() {
5       return "Menggunakan Kaki";
6     }
7
8   }
```

d) Inheritance (Generalisasi)

Inheritance (Generalisasi) merupakan hubungan **"is-a"**, di mana kelas anak (SubClass) mewarisi attribute dan method dari kelas induknya (SuperClass). Nama kelas abstrak dibuat dalam huruf miring.

- Class diagram



- Kode program

SuperClass.java

```
1  public abstract class SuperClass {
2
3  }
```

SubClass1.java

```
1  public class SubClass1 extends SuperClass {
2
3  }
```

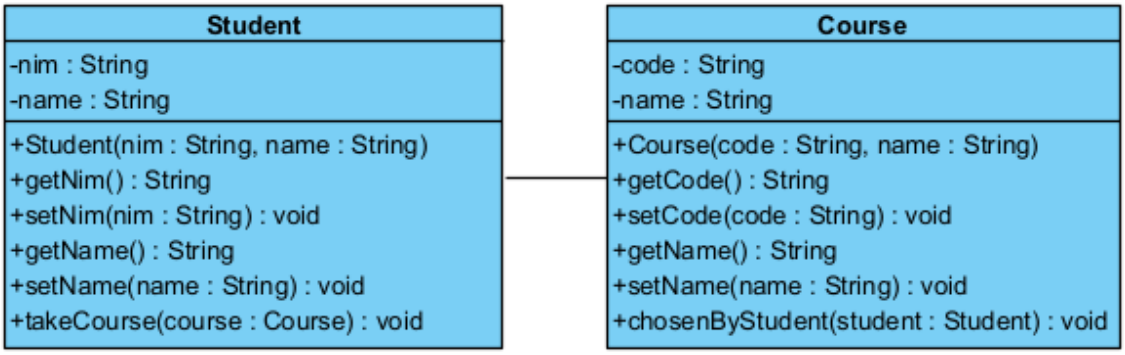
SubClass2.java

```
1  public class SubClass2 extends SuperClass {
2
3  }
```

e) Association

Association adalah hubungan antara dua atau lebih kelas yang menunjukkan bahwa kelas-kelas tersebut memiliki kaitan atau berinteraksi satu sama lain.

- Class diagram



- Kode Program

Student.java

```
1  public class Student {
2
3    private String nim;
4    private String name;
5
6    public Student(String nim, String name) {
```

Student.java

```
7       this.nim = nim;
8       this.name = name;
9     }
10
11    public String getNim() {
12      return this.nim;
13    }
14
15    public void setNim(String nim) {
16      this.nim = nim;
17    }
18
19    public String getName() {
20      return this.name;
21    }
22
23    public void setName(String name) {
24      this.name = name;
25    }
26
27    // Method asosiasi untuk mengaitkan Course dengan kelas Student
28    public void takeCourse(Course course) {
29      System.out.println("Mata kuliah " + course.getName() + " dipilih oleh "
    + name);
30    }
31  }
```

Course.java

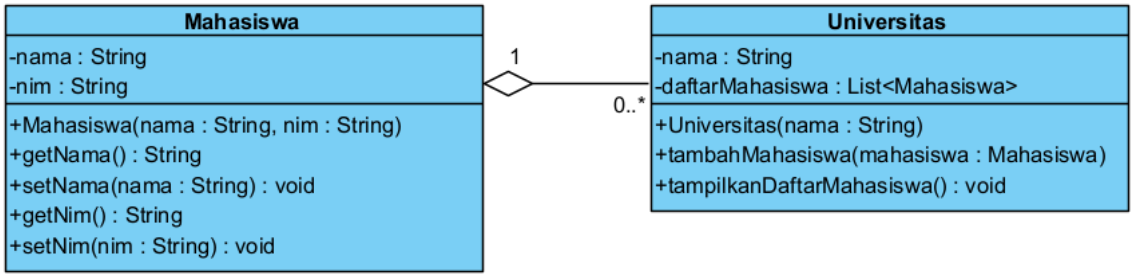
```
1   public class Course {
2
3     private String code;
4     private String name;
5
6     public Course(String code, String name) {
7       this.code = code;
8       this.name = name;
9     }
10
11    public String getCode() {
12      return this.code;
13    }
14
15    public void setCode(String code) {
16      this.code = code;
17    }
18
19    public String getName() {
20      return this.name;
21    }
22
23    public void setName(String name) {
24      this.name = name;
25    }
26
27    // Method asosiasi untuk mengaitkan Student dengan kelas Course
28    public void chosenByStudent(Student student) {
29      System.out.println(name + " mengambil mata kuliah " +
    student.getName());
30    }
31  }
```

Kelas Student mempunyai method takeCourse() dan kelas Course mempunyai method chosenByStudent() untuk mengkaitkan antara kelas Student dengan kelas Course.

f) Aggregation

Aggregation adalah jenis association khusus yang menunjukkan **"part-of"**. Ini merupakan hubungan antara dua kelas di mana satu kelas (kelas keseluruhan atau container) berisi objek dari kelas lain (kelas bagian). Dengan kata lain, kelas bagian merupakan bagian dari kelas keseluruhan dan dapat berada di banyak kelas keseluruhan lainnya.

- Class diagram



- Kode Program

Mahasiswa.java

```
1   public class Mahasiswa {
2
3     private String nama;
4     private String nim;
5
6     public Mahasiswa(String nama, String nim) {
7       this.nama = nama;
8       this.nim = nim;
9     }
10
11    public String getNama() {
12      return this.nama;
13    }
14
15    public void setNama(String nama) {
16      this.nama = nama;
17    }
18
19    public String getNim() {
20      return this.nim;
21    }
22
23    public void setNim(String nim) {
24      this.nim = nim;
25    }
26
27  }
```

Universitas.java

```
1   import java.util.List;
2   import java.util.ArrayList;
3
4   public class Universitas {
5
6     private String nama;
7     private List<Mahasiswa> daftarMahasiswa;
```

Universitas.java

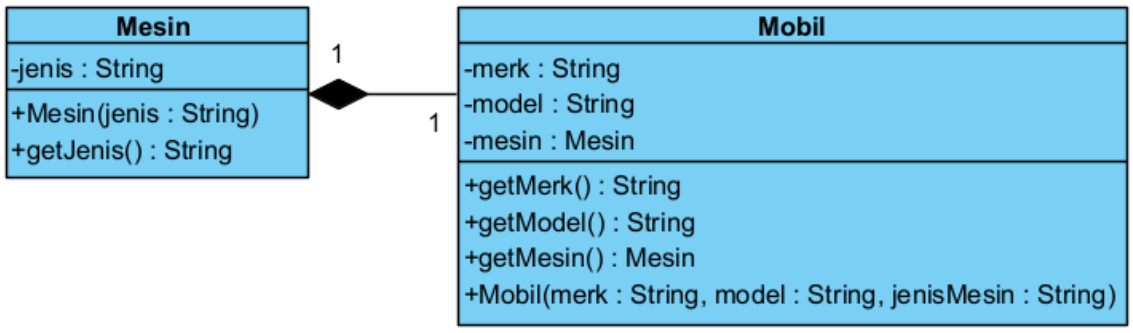
```
8
9     public Universitas(String nama) {
10      this.nama = nama;
11      this.daftarMahasiswa = new ArrayList<>();
12    }
13
14    public void tambahMahasiswa(Mahasiswa mahasiswa) {
15      daftarMahasiswa.add(mahasiswa);
16    }
17
18    public void tampilkanDaftarMahasiswa() {
19      System.out.println("Daftar Mahasiswa di Universitas " + nama +
":");
20      for (Mahasiswa mahasiswa : daftarMahasiswa) {
21        System.out.println("Nama: " + mahasiswa.getNama() + ", NIM: " +
mahasiswa.getNim());
22      }
23    }
24
25  }
```

Kelas "Universitas" menggunakan agregasi untuk mengandung objek-objek dari kelas "Mahasiswa" sebagai bagian dari universitas.

g) Composition

Composition adalah konsep di mana sebuah objek besar terdiri dari satu atau lebih objek lain yang merupakan bagian dari objek besar tersebut. Jika objek besar dihapus, maka objek-objek bagian juga akan dihapus.

- Class diagram



- Kode Program

Mesin.java

```
1   public class Mesin {
2     private String jenis;
3
4     public Mesin(String jenis) {
5       this.jenis = jenis;
6     }
7
8     public String getJenis() {
9       return jenis;
10    }
11  }
```

Mobil.java

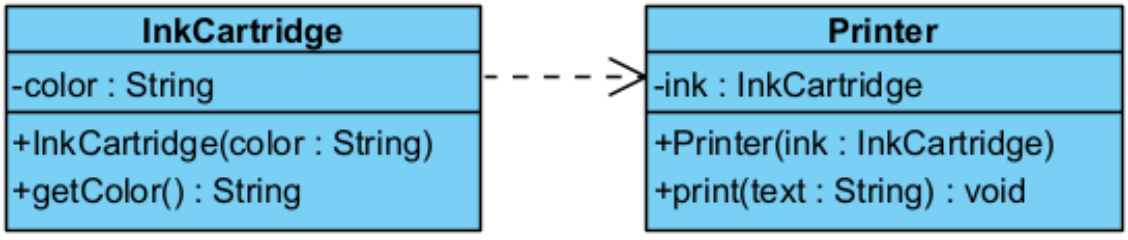
```
1  public class Mobil {
2    private String merk;
3    private String model;
4    private Mesin mesin;
5
6    public Mobil(String merk, String model, String jenisMesin) {
7      this.merk = merk;
8      this.model = model;
9      this.mesin = new Mesin(jenisMesin);
10   }
11
12   public String getMerk() {
13     return merk;
14   }
15
16   public String getModel() {
17     return model;
18   }
19
20   public Mesin getMesin() {
21     return mesin;
22   }
23 }
```

Kelas "Mobil" menggunakan composition untuk mengandung objek dari kelas "Mesin" sebagai bagian dari mobil. Ketika sebuah objek "Mobil" dibuat, objek "Mesin" juga akan dibuat sebagai bagian dari objek "Mobil". Jika objek "Mobil" dihapus, objek "Mesin" juga akan dihapus bersama dengan mobil tersebut.

h) Dependency

Dependency adalah hubungan antara dua kelas di mana satu kelas membutuhkan kelas lain untuk bekerja dengan benar. Seperti saat Anda membutuhkan pena untuk menulis surat. Jika pena rusak, Anda tidak dapat menulis surat dengan baik. Dalam pemrograman, kelas yang membutuhkan kelas lain disebut "dependency" pada kelas tersebut. Jika kelas yang menjadi ketergantungan berubah atau dihapus, kelas yang bergantung mungkin tidak berfungsi dengan baik.

- Class diagram



- Kode Program

InkCartridge.java

```
1  public class InkCartridge {
2    private String color;
3
4    public InkCartridge(String color) {
```

InkCartridge.java

```
5         this.color = color;
6     }
7
8     public String getColor() {
9         return color;
10    }
11 }
```

Printer.java

```
1  public class Printer {
2    private InkCartridge ink;
3
4    public Printer(InkCartridge ink) {
5      this.ink = ink;
6    }
7
8    public void print(String text) {
9      System.out.println("Printing: " + text);
10     System.out.println("Using ink color: " + ink.getColor());
11   }
12 }
```

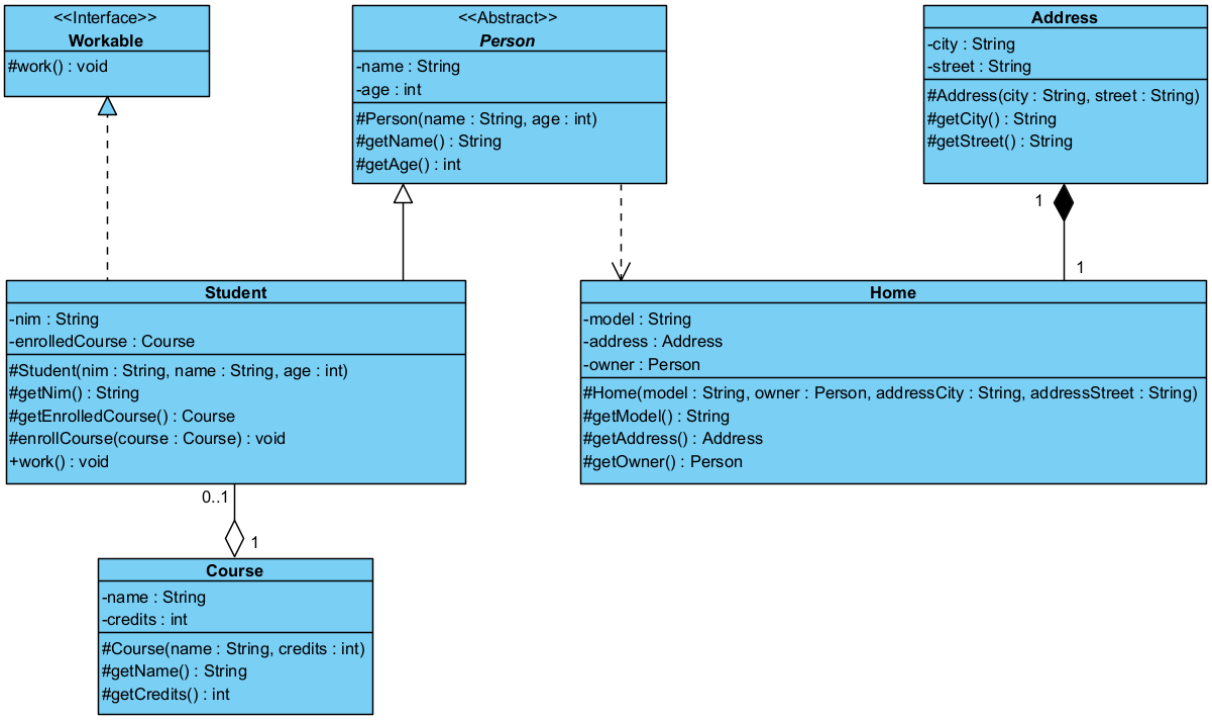
Kelas "Printer" memiliki ketergantungan (dependency) pada kelas "InkCartridge". Dependency terjadi ketika kelas "Printer" menerima objek "InkCartridge" melalui konstruktor, sehingga "Printer" bergantung pada "InkCartridge" untuk melaksanakan fungsionalitasnya. Jika kita mengubah atau menghapus kelas "InkCartridge", maka kelas "Printer" tidak akan bisa berfungsi dengan baik atau bahkan tidak bisa berjalan sama sekali karena kehilangan objek yang diperlukan untuk melaksanakan method-nya.

Multiplicity adalah cara untuk menunjukkan seberapa banyak objek dari satu kelas yang dapat terhubung atau berhubungan dengan objek dari kelas lain dalam suatu hubungan. Multiplicity digunakan untuk menggambarkan batasan atau aturan berapa banyak objek yang terlibat dalam hubungan tertentu. Dapat dinyatakan sebagai:

- Exactly one – 1
- Zero or one – 0..1
- Many – 0..* or *
- One or more – 1..*
- Exact number – e.g. 3..4 or 6
- Complex relationship – e.g. 0..1, 3..4, 6.* yang berarti sejumlah objek selain 2 atau 5.

3. Studi Kasus 1: Student Information

Silahkan implementasikan class diagram berikut ke dalam kode program Java.



Silahkan menggunakan class **App.java** berikut untuk dapat menjalankan program.

App.java

```
1  import java.util.Scanner;
2
3  public class App {
4    public static void main(String[] args) {
5
6      Scanner sc = new Scanner(System.in);
7
8      String studentNIM = sc.nextLine();
9      String studentName = sc.nextLine();
10     int studentAge = Integer.parseInt(sc.nextLine());
11
12     String homeModel = sc.nextLine();
13     String addressCity = sc.nextLine();
14     String addressStreet = sc.nextLine();
15
16     String courseName = sc.nextLine();
17     int courseCredits = Integer.parseInt(sc.nextLine());
18
19     Person person = new Student(studentNIM, studentName, studentAge);
20     Home home = new Home(homeModel, person, addressCity, addressStreet);
21     Course course = new Course(courseName, courseCredits);
22
23     if (person instanceof Student) {
24       Student student = (Student) person;
25       student.enrollCourse(course);
26     }
27
28     System.out.println("Person Name: " + person.getName());
29     System.out.println("Person Age: " + person.getAge());
30
31     if (person instanceof Student) {
```

App.java

```
32          Student student = (Student) person;
33          System.out.println("Student ID: " + student.getNim());
34          System.out.println("Student Home Model: " + home.getModel());
35          System.out.println("Student Address: " +
    home.getAddress().getStreet() + ", " + home.getAddress().getCity());
36          System.out.println("Enrolled Course: " +
    student.getEnrolledCourse().getName() + " ["
37              + student.getEnrolledCourse().getCredits() + "sks]");
38          student.work();
39      }
40
41      sc.close();
42    }
43  }
```

Contoh program saat dijalankan:

Input	Output
11S18005 Abdullah Ubaid 23 AAA Sitoluama Jl. PI Del PBO 4	Person Name: Abdullah Ubaid Person Age: 23 Student ID: 11S18005 Student Home Model: AAA Student Address: Jl. PI Del, Sitoluama Enrolled Course: PBO [4sks] Student is studying