MODUL PRAKTIKUM

TI03209 – PEMROGRAMAN BERORIENTASI OBJEK

Disusun oleh

Eko Mailansa, S.Kom., M.T.

D3 – TEKNIK INFORMATIKA

POLITEKNIK HASNUR

BARTIO KUALA

TAHUN 2023

MODUL VI

Abstraction And Interface Class

VI.1. Tujuan Praktikum

Praktikum yang dilakukan pada modul 6 bertujuan untuk:

1. Mengerti dan memahami konsep dari ***abstract class*** dalam pemrograman Java
2. Mengerti dan memahami konsep dari ***interface class*** dalam pemrograman Java
3. Mengerti definisi ***abstract and interface class*** serta penerapannya dalam pemrograman Java
4. Mengerti dan memahami perbedaan antara ***abstract and interface class*** dalam pemrograman Java

VI.2. Indikator Pencapaian

Setelah melakukan praktikum maka, di harapkan praktikan mampu:

1. Membuat program sederhana yang mengimplementasikan ***abstract class***.
2. Membuat program sederhana yang mengimplementasikan ***interface class***.
3. Membuat program sederhana yang mengimplementasikan ***abstract class*** dan ***interface class*** dalam pemrograman Java.

VI.3. Materi

VI.3.1. Abstraction

Abstraksi Data di java adalah properti yang hanya menampilkan detail penting kepada pengguna. Unit yang sepele atau tidak penting tidak ditampilkan kepada pengguna.

Contoh: Mobil dipandang sebagai mobil, bukan komponen individualnya.

Algoritma untuk mengimplementasikan abstraksi di Java

1. Tentukan kelas atau *interface* yang akan menjadi bagian dari abstraksi.
2. Buat kelas abstrak atau *interface* yang mendefinisikan perilaku dan properti umum pada kelas-kelas yang di tulis.

3. Tentukan metode abstrak dalam kelas abstrak atau interface yang tidak memiliki detail implementasi apa pun.
4. Menerapkan kelas konkrit yang ***extends*** kelas abstrak atau mengimplementasikan *interface*.
5. Ganti metode abstrak di kelas konkret untuk menyediakan implementasi spesifiknya.
6. Gunakan kelas konkrit untuk mengimplementasikan logika program.

Kapan menggunakan kelas abstrak dan metode abstrak

Ada situasi di mana kita ingin mendefinisikan superclass yang mendeklarasikan struktur abstraksi tertentu tanpa menyediakan implementasi lengkap dari setiap metode. Terkadang kita ingin membuat superclass yang hanya mendefinisikan bentuk generalisasi yang akan digunakan bersama oleh semua subclassnya, menyerahkan kepada masing-masing subclass untuk mengisi detailnya.

Berikut beberapa alasan mengapa menggunakan abstrak di Java:

1. ***Abstraksi***

   Kelas abstrak digunakan untuk menentukan templat umum untuk diikuti kelas lain. Mereka mendefinisikan seperangkat aturan dan pedoman yang harus diikuti oleh subkelas mereka. Dengan menyediakan kelas abstrak, kita dapat memastikan bahwa kelas yang memperluasnya (*extends*) memiliki struktur dan perilaku yang konsisten. Hal ini membuat kode lebih terorganisir dan lebih mudah dipelihara.
2. ***Polimorfisme***

   Kelas dan metode abstrak mengaktifkan polimorfisme di Java. Polimorfisme adalah kemampuan suatu objek untuk mengambil berbagai bentuk. Ini berarti bahwa variabel bertipe abstrak dapat menampung objek dari subkelas konkret mana pun dari kelas abstrak tersebut. Hal ini membuat kode lebih fleksibel dan mudah beradaptasi dengan berbagai situasi.
3. ***Framework dan API:***

   Java memiliki banyak *framework* dan API yang menggunakan kelas abstrak. Dengan menggunakan kelas abstrak, pengembang dapat menghemat waktu dan

tenaga dalam mengembangkan kode yang sudah ada dan berfokus pada aspek spesifik aplikasi mereka.

Singkatnya, kata kunci abstract digunakan untuk menyediakan template secara umum untuk diikuti oleh kelas lain. Ini digunakan untuk menegakkan konsistensi, pewarisan, polimorfisme, dan enkapsulasi di Java.

Enkapsulasi vs Abstraksi Data

1. Enkapsulasi adalah penyembunyian data (penyembunyian informasi) sedangkan Abstraksi adalah penyembunyian detail (penyembunyian implementasi).
2. Enkapsulasi mengelompokkan data dan metode yang bertindak atas data tersebut, abstraksi data berkaitan dengan mengekspos *interface* kepada pengguna dan menyembunyikan detail implementasi.
3. Kelas yang dienkapsulasi adalah kelas Java yang mengikuti penyembunyian dan abstraksi data. Kita dapat mengimplementasikan abstraksi dengan menggunakan kelas abstrak dan kelas *interface*.
4. Enkapsulasi adalah prosedur yang terjadi pada tingkat implementasi sedangkan abstraksi adalah proses tingkat desain.

Keuntungan Abstraksi

1. Mengurangi kerumitan dalam melihat sesuatu.
2. Menghindari duplikasi kode dan meningkatkan penggunaan kembali (*reusebility*).
3. Membantu meningkatkan keamanan suatu aplikasi atau program karena hanya rincian penting yang diberikan kepada pengguna.
4. Meningkatkan pemeliharaan aplikasi.
5. Meningkatkan modularitas aplikasi.
6. Peningkatannya akan menjadi sangat mudah karena tanpa mempengaruhi pengguna akhir dapat melakukan segala jenis perubahan pada sistem internal kami.
7. Meningkatkan penggunaan kembali (*reusebility*) dan pemeliharaan kode.
8. Menyembunyikan detail implementasi dan hanya menampilkan informasi yang relevan.
9. Memberikan *interface* yang jelas dan sederhana kepada pengguna.
10. Meningkatkan keamanan dengan mencegah akses ke detail kelas internal.

11. Mendukung modularitas, karena sistem yang kompleks dapat dibagi menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola.
12. Abstraksi menyediakan cara untuk menyembunyikan kompleksitas detail implementasi dari pengguna sehingga lebih mudah untuk dipahami dan digunakan.
13. Abstraksi memungkinkan adanya fleksibilitas dalam implementasi suatu program, karena perubahan pada detail implementasi yang mendasarinya dapat dilakukan tanpa mempengaruhi *interface* yang dilihat pengguna.
14. Abstraksi memungkinkan modularitas dan pemisahan perhatian yang membuat kode lebih mudah dipelihara dan lebih mudah untuk di-debug.

Kekurangan Abstraksi di Java:

1. Abstraksi dapat mempersulit pemahaman cara kerja sistem.
2. Dapat menyebabkan peningkatan kompleksitas, terutama jika tidak digunakan dengan benar.
3. Hal ini mungkin membatasi fleksibilitas implementasi.
4. Abstraksi dapat menambah kompleksitas yang tidak perlu pada kode jika tidak digunakan dengan tepat sehingga, menyebabkan peningkatan waktu dan upaya pengembangan.
5. Abstraksi dapat mempersulit proses debug dan memahami kode, terutama bagi mereka yang tidak terbiasa dengan lapisan abstraksi dan detail implementasi.
6. Penggunaan abstraksi yang berlebihan dapat mengakibatkan penurunan kinerja karena adanya tambahan lapisan kode.

VI.3.2. Interface

Interface adalah tipe referensi di Java dan mirip dengan kelas. *Interface* adalah kumpulan metode abstrak. Sebuah kelas mengimplementasikan *interface*, sehingga mewarisi metode abstrak *interface*.

Selain metode abstrak, *interface* juga dapat berisi konstanta, metode default, metode statis, dan tipe bersarang. Badan metode hanya ada untuk metode default dan metode statis. Menulis *interface* mirip dengan menulis kelas. Tapi kelas menggambarkan atribut dan perilaku suatu objek. Sedangkan sebuah *interface* berisi perilaku yang diimplementasikan

oleh suatu kelas. Kecuali kelas yang mengimplementasikan *interface* bersifat abstrak, semua metode *interface* perlu didefinisikan di kelas.

Untuk mengakses metode *interface*, *interface* harus di "**implements**" (seperti diwarisi) oleh kelas lain dengan kata kunci **implements** (bukan **extends**). Isi metode *interface* disediakan oleh kelas "**implement**":

***Interface* mirip dengan kelas dalam hal berikut:**

- Sebuah *interface* dapat berisi sejumlah metode.
- *Interface* ditulis dalam file dengan ekstensi **.java** dengan nama *interface* sesuai dengan nama file.
- Kode byte *interface* muncul dalam file **.class** .
- *Interface* muncul dalam paket, dan file bytecode yang sesuai harus berada dalam struktur direktori yang cocok dengan nama paket.
- Namun, *interface* berbeda dari kelas dalam beberapa hal, termasuk -
- Anda tidak dapat membuat instance *interface* .
- *Interface* tidak mengandung konstruktor apa pun.
- Semua metode dalam *interface* bersifat abstrak.
- *Interface* tidak boleh berisi bidang instan. Satu-satunya bidang yang dapat muncul di *interface* harus dinyatakan statis dan final.
- Sebuah *interface* tidak diperluas oleh suatu kelas; itu diimplementasikan oleh kelas.
- Sebuah *interface* dapat memperluas banyak *interface* .

Mengapa Dan Kapan Menggunakan *Interface*?

1) Untuk mencapai keamanan - sembunyikan detail tertentu dan hanya tampilkan detail penting dari suatu objek (*interface*).
2) Java tidak mendukung "pewarisan berganda" (sebuah kelas hanya dapat mewarisi dari satu superkelas). Namun, hal ini dapat dicapai dengan *interface*, karena kelas dapat mengimplementasikan banyak *interface*.
   Catatan: Untuk mengimplementasikan beberapa *interface*, harus di pisahkan dengan tanda baca koma.

VI.4. Alat dan Bahan

Peralatan dan bahan-bahan yang digunakan antara lain:

- ✓ Satu set Komputer
- ✓ Aplikasi IDE Netbeans 8.2
- ✓ Modul 6 – Mata kuliah Pemrograman Berorientasi Objek

VI.5. Praktikum

1. *Class abstract*

a. Class Ayam

```
// Abstract class
abstract class Binatang {
    // Abstract method (does not have a body)

    public abstract void SuaraBinatang();
    // Regular method

    public void tidur() {
        System.out.println("Zzz");
    }
}

// Subclass (inherit from Binatang)
class Ayam extends Binatang {

    @Override
    public void SuaraBinatang() {
        // The body of SuaraBinatang() is provided here
        System.out.println("Kukuruyuuuuukk!!!");
    }
}
```

b. Class Main

```
public class Main {

    public static void main(String[] args) {
        Ayam myAyam = new Ayam();  // Create a Ayam object
        myAyam.SuaraBinatang();
        myAyam.tidur();
    }
}
```

2. *Class interface (Single)*

a. Class Ayam

```
// Interface
interface Binatang {

    public void SuaraBinatang(); // interface method (does not have a body)

    public void tidur(); // interface method (does not have a body)
}

// Ayam "implements" interface Binatang
public class Ayam implements Binatang {

    @Override
    public void SuaraBinatang() {
        // The body of SuaraBinatang() is provided here
        System.out.println("Kukuruyuuuuukk!!!");
    }

    @Override
    public void tidur() {
        // The body of tidur() is provided here
        System.out.println("Zzz");
    }
}
```

b. Class Main

```
public class Main {

    public static void main(String[] args) {
        Ayam myAyam = new Ayam();  // Create a Ayam object
        myAyam.SuaraBinatang();
        myAyam.tidur();
    }
}
```

3. *Class interface (Multiple)*

a. Class DemoClass

```
interface FirstInterface {

    public void myMethod(); // interface method
}

interface SecondInterface {

    public void myOtherMethod(); // interface method
}

interface ThirtInterface {

    public void myOtherMethod1(); // interface1 method
}

public class DemoClass implements FirstInterface, SecondInterface, ThirtInterface {

    @Override
    public void myMethod() {
        System.out.println("Teknik Informatika");
    }

    @Override
    public void myOtherMethod() {
        System.out.println("Teknik Otomotif");
    }

    @Override
    public void myOtherMethod1() {
        System.out.println("Budidaya Tanaman Perkebunan");
    }
}
```

b. Class MainMultiple

```
public class MainMultiple {

    public static void main(String[] args) {
        System.out.println(":: Program Studi ::");
        DemoClass myObj = new DemoClass();
        myObj.myMethod();
        myObj.myOtherMethod();
        myObj.myOtherMethod1();
    }
}
```

VI.6. Referensi

V.6.1. Utama

- ✓ Pecinovský, Rudolf, CSc. OOP – Learn Object Oriented Thinking and Programming. ISBN 978-80-904661-8-0 (paperback) & ISBN 978-80-904661-9-7 (PDF). Published in the Czech Republic by Tomáš Bruckner, Řepín – Živonín, Academic Series. 2013.
- ✓ Wiener, Richard (University of Colorado, Colorado Springs) & J. Pinson, Lewis (University of Colorado, Colorado Springs). Fundamentals of OOP and Data Structures in Java. ISBN 0 521 66220 6 hardback and eISBN 0-511-00168-1 virtual (netLibrary Edition). PUBLISHED BY CAMBRIDGE UNIVERSITY PRESS. 2000.

V.6.2. Pendukung

- ✓ Object Oriented Programming in JAVA. Di akses pada 16 Agustus 2023. https:// www. tutorialspoint.com/object-oriented-programming-in-java/index.asp
- ✓ Object Oriented Programming (OOPs) Concept In Java - GreekforGreeks. Di akses pada 16 Agustus 2023. https:// www.google.com/amp/s/www.greekforgreeks.com/ object-oriented-programming-oops-concept-in-java/
- ✓ Object Oriented Thinking and Programming. Di akses pada 16 Agustus 2023. https://www.w3schools.in/java/java_oop.asp

VI.7. Tugas Tambahan

Buatlah 1 *interface method* bernana ***QuartInterface*** dengan parameter String x di dalam ***class DemoClass*** (praktikum nomor 3). Tulislah perintah di dalam ***class MainMultiple1*** agar nilai dari variable x di inputkan oleh user seperti tampilan di bawah ini:

```
:: Program Studi ::
Teknik Informatika
Teknik Otomotif
Budidaya Tanaman Perkebunan

Tambah Program Studi: Bisnis Digital

:: Program Studi (Setelah Di tambahkan) ::
Teknik Informatika
Teknik Otomotif
Budidaya Tanaman Perkebunan
Bisnis Digital
```

⇨ Input dari user

⇨ Hasil Akhir di panggil kembali

VI.8. Laporan

- ✓ Laporan "Praktikum" di tulis sesuai dengan contoh format yang telah di berikan.
- ✓ Laporan "Tugas Tambahan" di tulis sesuai dengan contoh format yang telah di berikan dan di lampirkan di setiap laporan "Praktikum" sesuai dengan Modul Praktikum.
- ✓ Isi laporan praktikum ➔ Praktikum dan Tugas Tambahan.
- ✓ *Rename* file laporan dengan format nama yang benar (nama_modul + nim + kelas) dan berikan ekstensi atau tipe file .pdf.
  (contoh: **Modul1_20202020_3B.pdf**)