UNIVERSITAS TEKNOLOGI SUMBAWA

Pemrograman Berorientasi Objek

Prodi Informatika – Fakultas Rekayasa Sistem

Semester Ganjil, 2024/2025

Oleh : I Made Widiarta

Agenda

- Polymorphism
- Constructor

UNIVERSITAS TEKNOLOGI SUMBAWA

Polymorphism

Polymorphism

UNIVERSITAS TEKNOLOGI SUMBAWA

- 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

Contoh Penggunaan Polymorphism

UNIVERSITAS TEKNOLOGI SUMBAWA

```
class Polygon {

  // method to render a shape
  public void render() {
    System.out.println("Rendering Polygon...");
  }
}

class Square extends Polygon {

  // renders Square
  public void render() {
    System.out.println("Rendering Square...");
  }
}

class Circle extends Polygon {

  // renders circle
  public void render() {
    System.out.println("Rendering Circle...");
  }
}
```

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

    // create an object of Square
    Square s1 = new Square();
    s1.render();

    // create an object of Circle
    Circle c1 = new Circle();
    c1.render();
  }
}
```

Output

```
Rendering Square...
Rendering Circle...
```

Contoh Penggunaan Polymorphism

UNIVERSITAS TEKNOLOGI SUMBAWA

- Pada contoh di atas, kita telah membuat sebuah superclass Polygon dan dua subclass Square dan Circle. Perhatikan penggunaan methods render()
- Fungsi utama dari methods render() adalah untuk merender garis. Namun, output dari methods render() yang ada pada class Square akan berbeda jika dibandingkan pada class Circle
- Maka, methods render() memiliki karaketeristik yang berbeda di setiap class atau dapat kita katakana bahwa method render() adalah polymorpic

Why Polymorphism?

UNIVERSITAS TEKNOLOGI SUMBAWA

- Polymorphism membantu kita membuat kode program yang konsisten.
- Pada contoh sebelumnya, kita bisa saja membuat methode rendeSquare() dan renderCircle() untuk methods pada class Square dan Circle. Cara seperti ini juga dapat bekerja, namun untuk setiap subclass selanjutnya yang kita buat, maka kita juga harus membuat methods baru. Ini yang membuat kode program kita tidak konsisten
- Untuk menyelesaikan masalah ini, Java mengijinkan kita untuk membuat sebuah methods yang memiliki karakteristik yang berbeda untuk setiap sub classnya.

Penerapan Polymorphism

Kita bisa menerapkan polymorphism dengan beberapa cara, diantaranya:

1. Method Overriding
2. Method Overloading
3. Operator Overloading

1. Java Method Overriding

UNIVERSITAS TEKNOLOGI SUMBAWA

- During inheritance in Java, if the same method is present in both the superclass and the subclass. Then, the method in the subclass overrides the same method in the superclass. This is called method overriding.

- In this case, the same method will perform one operation in the superclass and another operation in the subclass. For example,

Example 1: Polymorphism using method overriding

UNIVERSITAS TEKNOLOGI SUMBAWA

```java
class Language {
  public void displayInfo() {
    System.out.println("Common English Language");
  }
}

class Java extends Language {
  @Override
  public void displayInfo() {
    System.out.println("Java Programming Language");
  }
}

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

    // create an object of Java class
    Java j1 = new Java();
    j1.displayInfo();

    // create an object of Language class
    Language l1 = new Language();
    l1.displayInfo();
  }
}
```

Output:

```
Java Programming Language
Common English Language
```

1. Java Method Overriding (Lanjutan ...)

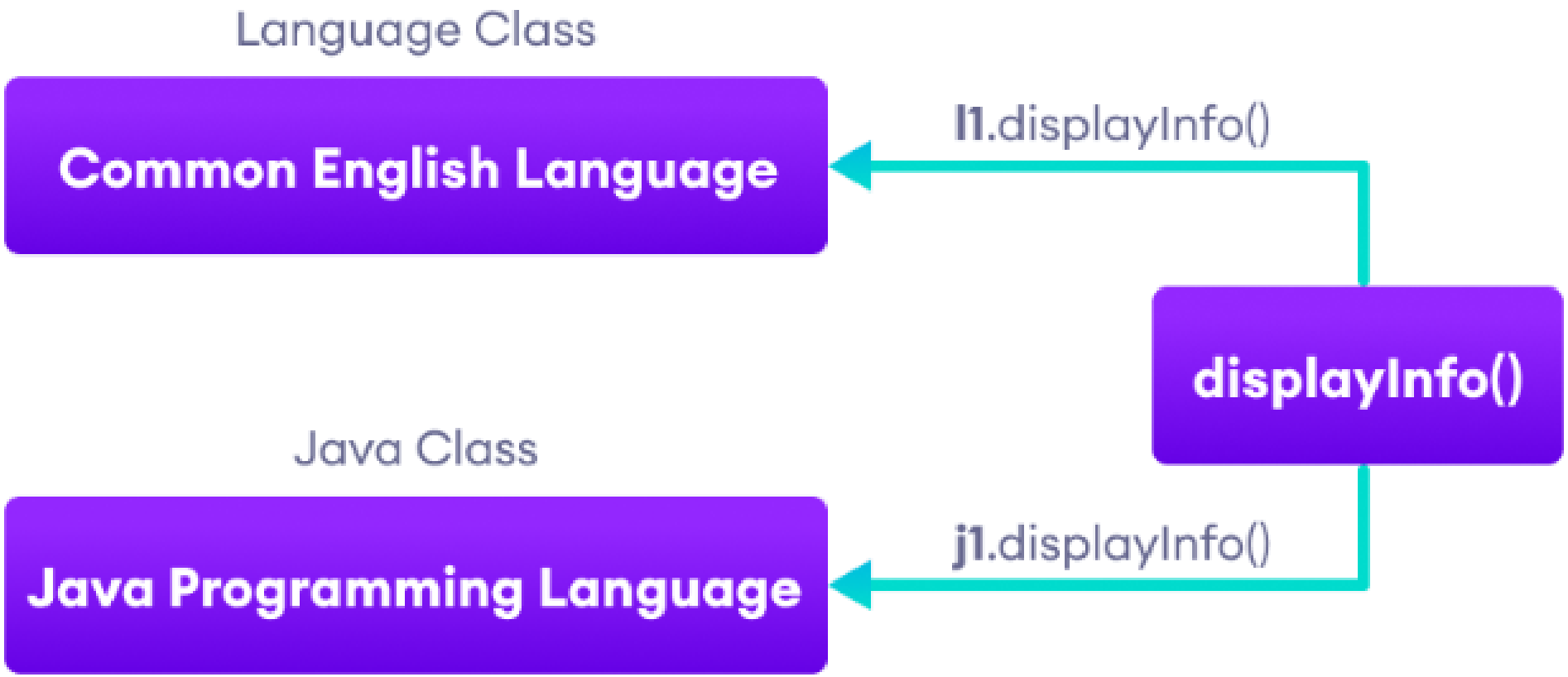
UNIVERSITAS TEKNOLOGI SUMBAWA

- In the above example, we have created a superclass named Language and a subclass named Java. Here, the method displayInfo() is present in both Language and Java.

- The use of displayInfo() is to print the information. However, it is printing different information in Language and Java.

- Based on the object used to call the method, the corresponding information is printed.

1. Java Method Overriding (Lanjutan ...)



2. Java Method Overloading

UNIVERSITAS TEKNOLOGI SUMBAWA

- In a Java class, we can create methods with the same name if they differ in parameters. For example,

```
void func() { ... }
void func(int a) { ... }
float func(double a) { ... }
float func(int a, float b) { ... }
```

- This is known as method overloading in Java. Here, the same method will perform different operations based on the parameter.

Example 3: Polymorphism using method overloading

UNIVERSITAS TEKNOLOGI SUMBAWA

```
class Pattern {

  // method without parameter
  public void display() {
    for (int i = 0; i < 10; i++) {
      System.out.print("*");
    }
  }

  // method with single parameter
  public void display(char symbol) {
    for (int i = 0; i < 10; i++) {
      System.out.print(symbol);
    }
  }
}

class Main {
  public static void main(String[] args) {
    Pattern d1 = new Pattern();

    // call method without any argument
    d1.display();
    System.out.println("\n");

    // call method with a single argument
    d1.display('#');
  }
}
```

Output:

```
**********

##########
```

Example 3: Polymorphism using method overloading

UNIVERSITAS TEKNOLOGI SUMBAWA

- In the above example, we have created a class named Pattern. The class contains a method named display() that is overloaded.
- Here, the main function of display() is to print the pattern. However, based on the arguments passed, the method is performing different operations:
  - prints a pattern of *, if no argument is passed or
  - prints pattern of the parameter, if a single char type argument is passed.

```
// method with no arguments
display() {...}

// method with a single char type argument
display(char symbol) {...}
```

3. Java Operator Overloading

UNIVERSITAS TEKNOLOGI SUMBAWA

- Some operators in Java behave differently with different operands. For example,

- + operator is overloaded to perform numeric addition as well as string concatenation, and
- operators like &, |, and ! are overloaded for logical and bitwise operations.
- Let's see how we can achieve polymorphism using operator overloading.

- The + operator is used to add two entities. However, in Java, the + operator performs two operations.

3. Java Operator Overloading

UNIVERSITAS TEKNOLOGI SUMBAWA

- When + is used with numbers (integers and floating-point numbers), it performs mathematical addition. For example,

```
int a = 5;
int b = 6;

// + with numbers
int sum = a + b;   // Output = 11
```

3. Java Operator Overloading

UNIVERSITAS TEKNOLOGI SUMBAWA

- When we use the + operator with strings, it will perform string concatenation (join two strings). For example,

```
String first = "Java ";
String second = "Programming";

// + with strings
name = first + second;   // Output = Java Programming
```

- Here, we can see that the + operator is overloaded in Java to perform two operations: addition and concatenation.

Polymorphic Variables

UNIVERSITAS TEKNOLOGI SUMBAWA

- A variable is called polymorphic if it refers to different values under different conditions.
- Object variables (instance variables) represent the behavior of polymorphic variables in Java. It is because object variables of a class can refer to objects of its class as well as objects of its subclasses.

Polymorphic Variables

UNIVERSITAS TEKNOLOGI SUMBAWA

```
class ProgrammingLanguage {
  public void display() {
    System.out.println("I am Programming Language.");
  }
}

class Java extends ProgrammingLanguage {
  @Override
  public void display() {
    System.out.println("I am Object-Oriented Programming Language.");
  }
}
```

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

    // declare an object variable
    ProgrammingLanguage pl;

    // create object of ProgrammingLanguage
    pl = new ProgrammingLanguage();
    pl.display();

    // create object of Java class
    pl = new Java();
    pl.display();
  }
}
```

Output:

```
I am Programming Language.
I am Object-Oriented Programming Language.
```

Polymorphic Variables

UNIVERSITAS TEKNOLOGI SUMBAWA

- In the above example, we have created an object variable pl of the ProgrammingLanguage class. Here, pl is a polymorphic variable. This is because,
- In statement pl = new ProgrammingLanguage(), pl refer to the object of the ProgrammingLanguage class.
- And, in statement pl = new Java(), pl refer to the object of the Java class.

UNIVERSITAS TEKNOLOGI SUMBAWA

Java Constructors

Java Constructors

UNIVERSITAS TEKNOLOGI SUMBAWA

- A constructor in Java is similar to a method that is invoked when an object of the class is created.
- Unlike Java methods, a constructor has the same name as that of the class and does not have any return type. For example,

```
class Test {
  Test() {
    // constructor body
  }
}
```

- Here, Test() is a constructor. It has the same name as that of the class and doesn't have a return type.

Example: Java Constructor

```java
class Main {
  private String name;

  // constructor
  Main() {
    System.out.println("Constructor Called:");
    name = "Programiz";
  }

  public static void main(String[] args) {

    // constructor is invoked while
    // creating an object of the Main class
    Main obj = new Main();
    System.out.println("The name is " + obj.name);
  }
}
```

Output:

```
Constructor Called:
The name is Programiz
```

- In the above example, we have created a constructor named Main().
- Inside the constructor, we are initializing the value of the name variable.
- Notice the statement creating an object of the Main class.

  Main obj = new Main();

- Here, when the object is created, the Main() constructor is called. And the value of the name variable is initialized.
- Hence, the program prints the value of the name variables as Programiz.

Types of Constructor

- In Java, constructors can be divided into three types:
  1. No-Arg Constructor
  2. Parameterized Constructor
  3. Default Constructor

1. Java No-Arg Constructors

UNIVERSITAS TEKNOLOGI SUMBAWA

- Similar to methods, a Java constructor may or may not have any parameters (arguments).
- If a constructor does not accept any parameters, it is known as a no-argument constructor. For example,

```
private Constructor() {
    // body of the constructor
}
```

1. Java No-Arg Constructors

UNIVERSITAS TEKNOLOGI SUMBAWA

```
class Main {

  int i;

  // constructor with no parameter
  private Main() {
    i = 5;
    System.out.println("Constructor is called");
  }

  public static void main(String[] args) {

    // calling the constructor without any parameter
    Main obj = new Main();
    System.out.println("Value of i: " + obj.i);
  }
}
```

Output:

```
Constructor is called
Value of i: 5
```

- In the above example, we have created a constructor Main().
- Here, the constructor does not accept any parameters. Hence, it is known as a no-arg constructor.
- Notice that we have declared the constructor as private.
- Once a constructor is declared private, it cannot be accessed from outside the class.
- So, creating objects from outside the class is prohibited using the private constructor.
- Here, we are creating the object inside the same class.
- Hence, the program is able to access the constructor. To learn more, visit Java Implement Private Constructor.
- However, if we want to create objects outside the class, then we need to declare the constructor as public.

2. Java Parameterized Constructor

UNIVERSITAS TEKNOLOGI SUMBAWA

- A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructors with parameters).

Example: Parameterized Constructor

UNIVERSITAS TEKNOLOGI SUMBAWA

```
class Main {

  String languages;

  // constructor accepting single value
  Main(String lang) {
    languages = lang;
    System.out.println(languages + " Programming Language");
  }

  public static void main(String[] args) {

    // call constructor by passing a single value
    Main obj1 = new Main("Java");
    Main obj2 = new Main("Python");
    Main obj3 = new Main("C");
  }
}
```

Output :

```
Java Programming Language
Python Programming Language
C Programming Language
```

- In the above example, we have created a constructor named Main().
- Here, the constructor takes a single parameter. Notice the expression:

Main obj1 = new Main("Java");

- Here, we are passing the single value to the constructor.
- Based on the argument passed, the language variable is initialized inside the constructor.

3. Java Default Constructor

UNIVERSITAS TEKNOLOGI SUMBAWA

- If we do not create any constructor, the Java compiler automatically creates a no-arg constructor during the execution of the program.
- This constructor is called the default constructor.

Example: Default Constructor

UNIVERSITAS TEKNOLOGI SUMBAWA

```
class Main {

  int a;
  boolean b;

  public static void main(String[] args) {

    // calls default constructor
    Main obj = new Main();

    System.out.println("Default Value:");
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
  }
}
```

Output :

```
Default Value:
a = 0
b = false
```

- Here, we haven't created any constructors.
- Hence, the Java compiler automatically creates the default constructor.
- The default constructor initializes any uninitialized instance variables with default values.

Example: Default Constructor

Type	Default Value
`boolean`	`false`
`byte`	**0**
`short`	**0**
`int`	**0**
`long`	**0L**
`char`	`\u0000`
`float`	**0.0f**
`double`	**0.0d**
`object`	`Reference null`

Important Notes on Java Constructors

UNIVERSITAS TEKNOLOGI SUMBAWA

- Constructors are invoked implicitly when you instantiate objects.
- The two rules for creating a constructor are:
  1. The name of the constructor should be the same as the class.
  2. A Java constructor must not have a return type.
- If a class doesn't have a constructor, the Java compiler automatically creates a default constructor during run-time. The default constructor initializes instance variables with default values. For example, the int variable will be initialized to 0
- Constructor types:
- No-Arg Constructor - a constructor that does not accept any arguments
- Parameterized constructor - a constructor that accepts arguments
- Default Constructor - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
- A constructor cannot be abstract or static or final.
- A constructor can be overloaded but can not be overridden.

Constructors Overloading in Java

UNIVERSITAS TEKNOLOGI SUMBAWA

- Similar to Java method overloading, we can also create two or more constructors with different parameters. This is called constructor overloading.

Constructors Overloading in Java

UNIVERSITAS TEKNOLOGI SUMBAWA

```java
class Main {

  String language;

  // constructor with no parameter
  Main() {
    this.language = "Java";
  }

  // constructor with a single parameter
  Main(String language) {
    this.language = language;
  }

  public void getName() {
    System.out.println("Programming Language: " + this.language);
  }

  public static void main(String[] args) {

    // call constructor with no parameter
    Main obj1 = new Main();

    // call constructor with a single parameter
    Main obj2 = new Main("Python");

    obj1.getName();
    obj2.getName();
  }
}
```

Output:

```
Programming Language: Java
Programming Language: Python
```

- In the above example, we have two constructors: Main() and Main(String language).
- Here, both the constructors initialize the value of the variable language with different values.
- Based on the parameter passed during object creation, different constructors are called, and different values are assigned.
- It is also possible to call one constructor from another constructor. To learn more, visit Java Call One Constructor from Another.

UNIVERSITAS TEKNOLOGI SUMBAWA

Terima kasih