OBJECT ORIENTED PROGRAMMING

Polymorphism

by
Dr. Nor Saradatul Akmar Zulkifli
Faculty of Computer Systems & Software Engineering
saradatulakmar@ump.edu.my

OER Object Oriented Programming by Dr. Nor Saradatul Akmar Binti Zulkifli work is under licensed Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

Content Overview

- ✓ What is Polymorphism in programming?
- ✓ Why we use Polymorphism?
- ✓ Types of Polymorphism in Java
  - ➢ Overloading
  - ➢ Overriding
- ✓ When we use Polymorphism?

Learning Objective

- To understand
- To differentiate between member method and constructor method overloading
- To write method overloading

WHAT IS POLYMORPHISM?

<u>POLY</u> MORPHISM ?

Many

Form

- Has the ability to appear in many shapes.
- In OO
  - The ability of objects of different types to respond to functions of the same name
  - The user does not have to know the exact type of the object in advance

WHAT IS THE SAME CONCEPT THAT A AND B SHARE?

On/Off Switch

BUT

Internally, how the on/off switch is totally different

POLYMORPHISM

When 2 same-named item (on/off switches) performing the same task (turning something on or off), despite being very different internally

WHY USE POLYMORPHISM?

1. Help programmers to write code that is easy to modify and extend

2. Allows a basic class variable to refer to objects from different subclasses in the same inheritance hierarchy

WHY USE POLYMORPHISM?

3. Allows programmers to send the same message (or call the same function) for object from different classes.

HOW?

OVERRIDING

- When a method that has already been defined in a parent class is redefined using the exact same signature.
- Classes that methods appear in MUST be in parent/child relationship
- Signatures MUST match
- Methods SOMETIMES COMBINED

OVERLOADING

- When two methods were defined with the same name, in the same class , distinguished by their signature.
- Classes that methods appear in do NOT have to be in parent/child relationship
- Signatures MUST NOT match
- Methods are SEPARATE

EXAMPLE OF POLYMORPHISM

```
class Student {
      public void Write (int ID, int Grad,
String Fname, String Lname) {
            m_ID = ID;
            m_Graduation = Grad;
            m_First = Fname;
            m_Last = Lname;
            }
     public void Display ( ) {
            System. Out.println ("Student: " +
m_ID + " " + m_First + " " + m_Last +
"Graduated : " + m_Graduation);
       }
   private int m_ID, m_Graduation;
   private String m_First;
   private String m_Last;
}
```

```
Class GradStudent extends Student {
      public void Write (int ID, int Grad,
String Fname, String Lname, int
yrGrad, String unSch, String major) {
      super.Write (ID, Fname, Lname, Grad);
      m_UndergradSchool = unSch;
      m_Major = major;
      YearGraduated = yrGrad;
      }
      public void Display ( ) {
      super.Display ( );
        System.out.println ("Graduated: " +
m_Graduation + " " +
m_UndergradSchool + " " + m_Major + "
" + YearGraduated);
      }
      private YearGraduated;
      private String_m_UndergradSchool,
      m_Major;
```

OVERLOADING METHODS

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

1. When several methods with the same name exist within a class

2. But MUST have different formal parameter list (method signature)

3. Both **member** and **constructor methods** can be overloaded

4. Two methods are said to have different formal parameter lists:
   a. if both methods have different number of formal parameters.

   b. If the number of formal parameters is the same in both methods, the data type of the formal parameters in the order we list must differ in at least one position.

Instance of the `Student` class

```
public static void Display ( Student s) {
        s.Display ( );
}

public static void Display (GradStudent g) {
        g.Display ( );
}
```

Instance of the `GradStudent` class

POLYMORPHISM: BENEFITS

1. It gives programmers the ability to develop **interfaces** for complex application

WHAT IS INTERFACE?

This topic will be discuss further in the Next Chapter….

Communitising Technology

EXAMPLE :

```
public int determineResult ( )

public int determineResult (int studentMark)

public int determineResult (int idMatric, int studentMark)

public int determineResult (String idMatric, int studentMark)
```

EXAMPLE :

Student Class

Overloaded Member Methods

```
public String determineResult (double mark)
{
     if (mark > 39)
         result = "PASS";
     else
         result = "FAIL";
     return result;
}

public String determineResult (double mark, double passingMark)
{
     if (mark > passingMark)
         result = "PASS";
     else
         result = "FAIL";
     return result;
}
}
```

EXAMPLE : Client Class

```
package student;
Import java.util.*;
public class membOver {
    public static void main (String [ ] args)
    {
        double studMark, passMark;
        Scanner read = new Scanner (System.in);
        Student UnderGradStud=new Student("Aminah", 01189);
        System.out.println ("Name: " +UnderGradStud.name);
        System.out.println ("Matric No: "+UnderGradStud.idMatric);
        System.out.print ("Enter Student's Mark: ");
        studMark = read.nextDouble ();
        System.out.println ("Result: "+UnderGradStud.determineResult(studMark));
        System.out.println();

        Student PostGradStud=new Student ("Ammar", 01982);
        System.out.println ("2nd Constructor:");
        System.out.println ("Name: " +PostGradStud.name);
        System.out.println ("Matric No: "+PostGradStud.idMatric);
        System.out.print ("Enter Student's Mark: ");
        studMark = read.nextDouble ();
        System.out.print ("Enter student's passing mark: ");
        passMark = read.nextDouble ();
        System.out.println ("Result: "
            +PostGradStud.determineResult(studMark,passMark));
```

Invoke different member methods in Student class

EXAMPLE :

Output:

```
run:
Name: Aminah
Matric No: 01189
Enter student's mark: 40
Result: PASS

Name: Ammar
Matric No: 01982
Enter student's mark: 40
Enter student's passing mark: 50
Result: FAIL
BUILD SUCCESSFUL (total time: 12 seconds)
```

CONSTRUCTOR METHOD OVERLOADING

1. Constructor Overloading :- When a class have more than one constructors

2. Properties of Constructor Method:
   a. Same name with a class name
   b. A constructors are automatically executed when a class object is instantiated
   c. The different constructor is executed based on the type of value passed to the constructor during object instantiation.

EXAMPLE :

```
package student;
public class Student
{
    public String name;
    public int idMatric;
    public String result;

public Student (String studentName, int matricNo)
{
    name = studentName;
    idMatric = matricNo;
    result = "unknown";
}
public Student (String studentName, int matricNo, double mark)
{
    name = studentName;
    idMatric = matricNo;
    result = determineGrade (mark);
}
```

Overloaded Constructor Methods

EXAMPLE :

Client Class

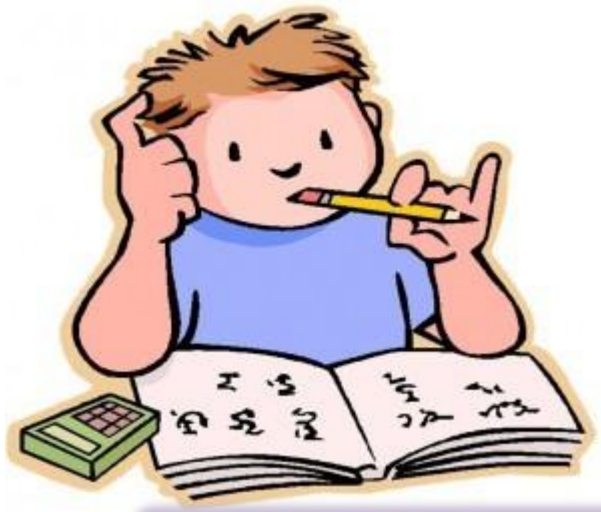
```
package student;
public class constMethod
{
    public static void main (String [ ] args)
    {
        Student UnderGradStud=new Student("Aminah", 01189);
        System.out.println ("1st Constructor:");
        System.out.println ("Name: " +UnderGradStud.name);
        System.out.println ("Matric No: "+UnderGradStud.idMatric);
        System.out.println ("Result: "+UnderGradStud.result);

        Student PostGradStud=new Student ("Ammar", 01982);
        System.out.println ("2nd Constructor:");
        System.out.println ("Name: " +PostGradStud.name);
        System.out.println ("Matric No: "+PostGradStud.idMatric);
        System.out.println ("Result: "+PostGradStud.result);
```

Invoke different constructor methods in Student class

EXAMPLE :

Output:

```
run:
1st Constructor:
Name: Aminah
Matric No: 01189
Result: unknown
2nd Constructor:
Name: Ammar
Matric No: 01982
Result: PASS
BUILD SUCCESSFUL (total time: 0 seconds)
```



OVERRIDING METHODS

Revise Lecture note on Week 6 : Inheritance

INHERITANCE : Constructor

- ✓ Used `super` keyword to refer to the parent class and often used to invoke the parent's constructor
- ✓ A child's constructor is responsible for calling the parent's constructor
- ✓ To call the parent's constructor – the first line of child's constructor can be the `super` keyword.
- ✓ The `super` keyword can also be used to reference other variables and method defined in the parent's class

```
super ( );
```

Used to call the constructor from superclass (parents) with appropriate arguments

INHERITANCE : Overriding Methods

- ✓ A child class can override the definition of an inherited method in favor of its own.
- ✓ The new method must have the SAME SIGNATURE (name and parameters) as the parent's method BUT can have a DIFFERENT BODY (implementation
- ✓ The type of the object executing the method determines which version of the method is invoked

```
super ( );
```

- ✓ Invoked explicitly the parents method using `super` reference.
- ✓ Method with `final` modifier, cannot be overridden.
- ✓ *Shadowing variables* is when an overriding concept applied to data and should be avoided – cause unnecessarily confusing code.

INHERITANCE : Overriding Methods

Superclass (Parent)

```
class student
{
int power;
public void setPower (int Power)
…..
}
```

Sub-class (Child)

```
class GradStud extends Student
{
int power; //shadowing variable power
//override the method setPower
Public void setPower (int Power);
int matricNo;     //Student class members
}
```

Author Information

Dr. Nor Saradatul Akmar Binti Zulkifli

Senior Lecturer
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang