WEEK 9

User-Defined Class (Method Overloading)

Sharmila Mat Yusof

Outline

- Constructor method overloading
  - Example of constructor method overloading
- Member method overloading
  - Example of member method overloading

Learning Objectives

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

Method Overloading

- In Java, within a class, several methods can have the same name. This is known as method overloading.
- However, overloaded method must have different formal parameter list (method signature).
- Both member and constructor methods can be overloaded.
- Two methods are said to have different formal parameter lists:
  - If both methods have a different number of formal parameters.
  - 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.

Constructor Method Overloading

- A class can have more than one constructors which is known as Constructor Overloading.
- Properties of constructor method:
  - Same name with a class name.
  - A constructor method has no return type.
  - Constructors are automatically executed when a class object is instantiated.
  - The different constructor is executed based on the type of value passed to the constructor during object instantiation.

Constructor Method Overloading: Example

Student class

```
1   package student;
2   public class Student
3   {
4   public String name;
5   public  int matricNo;
6   public String grade;
7
8
9   public Student(String studName,int matricNum)
10  {
11      name = studName;
12      matricNo = matricNum;
13      grade = "unknown";
14  }
15  public Student(String studName,int matricNum,double mark)
16  {
17      name = studName;
18      matricNo = matricNum;
19      grade = determineGrade(mark);
20  }
```

Overloaded constructor methods

Constructor Method Overloading: Example

Client class

```
1   package student;
2   public class constructorMethod
3   {
4       public static void main(String[] args)
5       {
6           Student UGStudent = new Student("Ali",12345);
7           System.out.println("First Constructor:" );
8           System.out.println("Name: "+UGStudent.name);
9           System.out.println("Matric number: "+UGStudent.matricNo);
10          System.out.println("Grade: "+UGStudent.grade);
11          Student PGStudent = new Student("Muhammad",88888,95);
12          System.out.println("Second Constructor:" );
13          System.out.println("Name: "+PGStudent.name);
14          System.out.println("Matric number: "+PGStudent.matricNo);
15          System.out.println("Grade: "+PGStudent.grade);
16
17      }
18
19  }
```

Invoke different constructor methods in **Student** class

Constructor Method Overloading: Example

Output:

```
run:
First Constructor:
Name: Ali
Matric number: 12345
Grade: unknown
Second Constructor:
Name: Muhammad
Matric number: 88888
Grade: PASS
BUILD SUCCESSFUL (total time: 0 seconds)
```

Member Method Overloading: Example

Example:

```
public int determineGrade( )
public int determineGrade(int studMark)
public int determineGrade(int matricNo, int studMark)
public int determineGrade(String matricNo, int studMark)
```

Member Method Overloading: Example

Student class

```
36      public String determineGrade(double mark)
37      {
38          if (mark > 39)
39              grade = "PASS";
40          else
41              grade = "FAIL";
42          return grade;
43      }
44
45      public String determineGrade(double mark, double passingMark)
46      {
47          if (mark > passingMark)
48              grade = "PASS";
49          else
50              grade = "FAIL";
51          return grade;
52      }
53
54      }
```

Overloaded member methods

Member Method Overloading: Example

```
1   package student;
2   import java.util.*;
3   public class memberOverload {
4       public static void main(String[] args)
5       {
6        double studMark,passMark;
7        Scanner read = new Scanner(System.in);
8
9           Student UGStudent = new Student("Ali",55555);  //Undergraduate student
10          System.out.println("Undergraduate Student");
11          System.out.println("Name: "+UGStudent.name);
12          System.out.println("Matric number: "+UGStudent.matricNo);
13          System.out.print("Enter student's mark: ");
14          studMark = read.nextDouble();
15          System.out.println("Student's grade: "+UGStudent.determineGrade(studMark));
16          System.out.println();
17          Student PGStudent = new Student("Muhamad",88888);  //Postgraduate student
18          System.out.println("Postgraduate Student");
19          System.out.println("Name: "+PGStudent.name);
20          System.out.println("Matric number: "+PGStudent.matricNo);
21          System.out.print("Enter student's mark: ");
22          studMark = read.nextDouble();
23          System.out.print("Enter student's passing mark: ");
24          passMark = read.nextDouble();
25          System.out.println("Student's grade: "+PGStudent.determineGrade(studMark,passMark));
26
27      }
28
29  }
```

Client class

Invoke different member methods in **Student** class

Member Method Overloading: Example

Output:

```
run:
Undergraduate Student
Name: Ali
Matric number: 55555
Enter student's mark: 40
Student's grade: PASS

Postgraduate Student
Name: Muhamad
Matric number: 88888
Enter student's mark: 40
Enter student's passing mark: 50
Student's grade: FAIL
BUILD SUCCESSFUL (total time: 12 seconds)
```

Summary

- Both member and constructor methods can be overloaded.
- Overloaded method must have the same name but different formal parameter list (method signature).
- Two methods are said to have different formal parameter lists:
  - If both methods have a different number of formal parameters.
  - 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.
- A constructor method has no return type.
- The different constructor is executed based on the type of value passed to the constructor during object instantiation.