Tutorial No. 3

Title: Program based on overloading.

Batch: SY-IT (B3) **Roll No.: 16010423076** **Tutorial No.: 3**

Aim: Program based on overloading

Resources needed: JDK 8 or later, Text Editor/IDE

Theory:

Method Overloading

Method overloading in Java allows a class to have multiple methods with the same name but different parameter lists, which is a helpful feature. This function enables different types or amounts of inputs to achieve similar tasks, enhancing the code's flexibility and readability. Unlike traditional approaches, overloaded methods are distinguished by their parameter lists rather than their names, allowing for the utilization of a single method name for multiple purposes. Method overloading simplifies code management by grouping related functions under one method name, improving clarity and ease of maintenance. This feature is crucial in the development of adaptable and user-friendly APIs, as it streamlines method invocation and adapts to different situations, ultimately improving the clarity and efficiency of Java programs.

Properties / Features of Method Overloading:

1. **Same Name:** Overloaded methods utilize the same name to group similar operations together.
2. **Various Parameter Lists:** To have overloaded methods, there must be variations in the parameter lists such as type, number, or both, permitting multiple methods to exist under the same name.
3. **Return Type Is Not Taken into Account:** Overloaded methods cannot be distinguished based solely on their return type; differences in the method's parameter list are required for overloading to occur.
4. **Compile-Time Polymorphism:** Method overloading is resolved during compilation, enabling the compiler to select the suitable method depending on the provided arguments.
5. **Improved flexibility:** Method overloading increases code flexibility by allowing a single method name to manage different types and quantities of inputs, making method calls easier.
6. **Enhanced Code Readability:** The use of method overloading allows for the grouping of related methods, which enhances code readability and maintainability.

Different Ways to Overload a Method

In Java, method overloading can be accomplished in several ways by varying the parameter lists. Overloading allows methods with the same name to coexist, provided their parameter lists differ in one or more aspects. Here are the primary ways to achieve method overloading:

1. Number of Parameters: Methods can be overloaded by changing the number of parameters. For example:

```
void add(int a, int b) {
    // Method for adding two integers
}
```

```
void add(int a, int b, int c) {
    // Method for adding three integers
}
```

Calling `add(20, 30)` invokes the method with two parameters, while `add(10, 20, 30)` calls the method with three parameters.

2. Data Type of Parameters: Overloading can also be achieved by varying the data types of parameters:

```
void print(int a) {
    // Method for printing an integer
}

void print(double a) {
    // Method for printing a double
}
```

Here, `print(10)` matches the method for integers, while `print(10.5)` matches the method for doubles.

3. Sequence of Parameter Types: Changing the order of parameter types can also be used to overload methods:

```
void display(int a, double b) {
    // Method with an int followed by a double
}

void display(double a, int b) {
    // Method with a double followed by an int
}
```

The method call `display(10, 5.5)` matches the first method, whereas `display(5.5, 10)` matches the second.

By applying these variations in the number, type, or sequence of parameters, methods with the same name can be effectively overloaded, providing greater flexibility and functionality within a class.

Task: Prepare a document containing the answers of the following question

1. Write a Java program to model a simple **Contact Management System** using a class named **Contact**. The **Contact** class should include overloaded methods to handle contact information in various ways. Implement the **addContact** method in three forms: **addContact(String name, String phoneNumber)** for basic contact details, **addContact(String name, String phoneNumber, String email)** to include an email address and **addContact(String name, String phoneNumber, String email, String address)** to also add a physical address. Additionally, provide overloaded **displayContact** methods: **displayContact(String name)** to show basic contact details and **displayContact(String name, boolean showAddress)** to conditionally include the address. The program should demonstrate adding and displaying contacts

using these methods to showcase the flexibility of method overloading in managing and presenting contact information.

Code :

```
class Contact {
String contactName;
String contactPhoneNumber;
String contactEmail;
String contactAddress;

void addContact(String name, String phoneNumber) {
contactName = name;
contactPhoneNumber = phoneNumber;
}

void addContact(String name, String phoneNumber, String email) {
contactName = name;
contactPhoneNumber = phoneNumber;
contactEmail = email;
}

void addContact(String name, String phoneNumber, String email, String address) {
contactName = name;
contactPhoneNumber = phoneNumber;
contactEmail = email;
contactAddress = address;
}
```

```
void display(String name) {
if (name.equals(contactName)) {
System.out.println("Name : " + contactName);
System.out.println("Phone Number : " + contactPhoneNumber);
}
else {
System.out.println("Contact not found.");
}
}

void display(String name, boolean showAddress) {
if (name.equals(contactName)) {
System.out.println("Name : " + contactName);
System.out.println("Phone Number : " + contactPhoneNumber);
if (contactEmail != null) {
System.out.println("Email : " + contactEmail);
}
if (showAddress && contactAddress != null) {
System.out.println("Address : " + contactAddress);
}
} else {
System.out.println("Contact not found.");
}
}
}
```

```
public class ContactManagementSystem {

public static void main(String[] args) {

Contact contact = new Contact();


contact.addContact("Ritesh Jha", "+91-7777777777");

contact.display("Ritesh Jha");

System.out.println();

contact.addContact("Aditya Singh", "+91-8888888888", "aditya@outlook.com");

contact.display("Aditya Singh", false);

System.out.println();

contact.addContact("Akshay Dixit", "+91-9999999999", "akshay@yahoo.com", "Ujjain, Madhyapradesh");

contact.display("Akshay Dixit", true);

}

}
```

Output :

```
Output
java -cp /tmp/6BYbRokJe0/ContactManagementSystem
Name : Ritesh Jha
Phone Number : +91-7777777777

Name : Aditya Singh
Phone Number : +91-8888888888
Email : aditya@outlook.com

Name : Akshay Dixit
Phone Number : +91-9999999999
Email : akshay@yahoo.com
Address : Ujjain, Madhyapradesh

=== Code Execution Successful ===
```

Outcomes:

CO1. Apply fundamental Object Oriented Methodology concepts using Java programming

---

Conclusion: (Conclusion to be based on the outcomes achieved)

I implemented basic concepts of Object-Oriented, including class creation, method overloading and encapsulation. During the development process, I learned how to structure and manipulate data. This practical exercise increased my understanding of the key principles of OOP for the development applications in Java.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References Books

1. Herbert Schildt; JAVA The Complete Reference; Seventh Edition, Tata McGraw-Hill Publishing Company Limited 2007.
2. Java 7 Programming - Black Book : Kogent Learning Solutions Inc.
3. Sachin Malhotra, Saurabh Chaudhary "Programming in Java", Oxford University Press, 2010
4. Jaime Nino, Frederick A. Hosch, 'An introduction to Programming and Object Oriented Design using Java', Wiley Student Edition.