Tutorial No. 4

Title: Program based on overriding.

Batch: SY-IT(B3) **Roll No.:** 16010423076 **Tutorial No.: 4**

Aim: Program based on overriding

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

Theory:

Method Overriding

Method overriding in Java is a basic concept in object-oriented programming that enables a subclass to give a particular implementation of a method that already exists in its superclass. This functionality allows for runtime polymorphism, allowing a subclass to change or add to the functionality of an inherited method to better suit its requirements. In contrast to method overloading, where methods have the same name but different parameters, method overriding involves the subclass method having the same name, return type, and parameters as the superclass method. This guarantees that the subclass can modify how the inherited method works without changing the method signature. Method overriding is crucial for enabling dynamic method dispatch, which in turn results in code that is more flexible and easier to maintain. It enhances the extensibility and adaptability of Java programs by allowing the suitable method to be called based on the runtime type of the object, supporting polymorphic behaviour.

Properties / Features of Method Overloading:

1. The overridden methods in the subclass need to have identical names, return types, and parameter lists as the method in the superclass. This guarantees that the subclass method perfectly matches the superclass method signature.
2. By using method overriding, a subclass can implement a unique version of a method inherited from its superclass, allowing the subclass to modify or enhance the functionality of the method.
3. Method overriding allows for runtime polymorphism by determining which method implementation to execute based on the actual class of the object at runtime, rather than the reference type.
4. The subclass's overriding method can have an equal or wider access modifier compared to the superclass's method, but not a narrower one.
5. It is not possible to override static methods since they cannot be overridden. When a subclass's static method shares the same name as one in its super class, it masks the superclass method instead of replacing it.
6. Consistent method signature is required for overriding methods to maintain behavior and support dynamic method dispatch in the superclass.

Different Ways of Method Overriding

1. **Simple Overriding:** Subclass provides a specific implementation of a method defined in the superclass with the same method signature.

2. **Method Overriding with Different Access Modifiers:** Subclass method has the same name and signature as the superclass method but can use a more permissive access modifier.
3. **Overriding Abstract Methods:** Subclass implements abstract methods from an abstract class or interface, providing concrete implementations.
4. **Overriding Methods in Interfaces:** Implementing classes provide concrete implementations for methods declared in interfaces.
5. **Overriding Methods with Covariant Return Types:** Subclass method overrides a superclass method and returns a type that is a subclass of the original return type.

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

Design a simple Vehicle Management System using Java to demonstrate method overriding and class inheritance.

Algorithm:

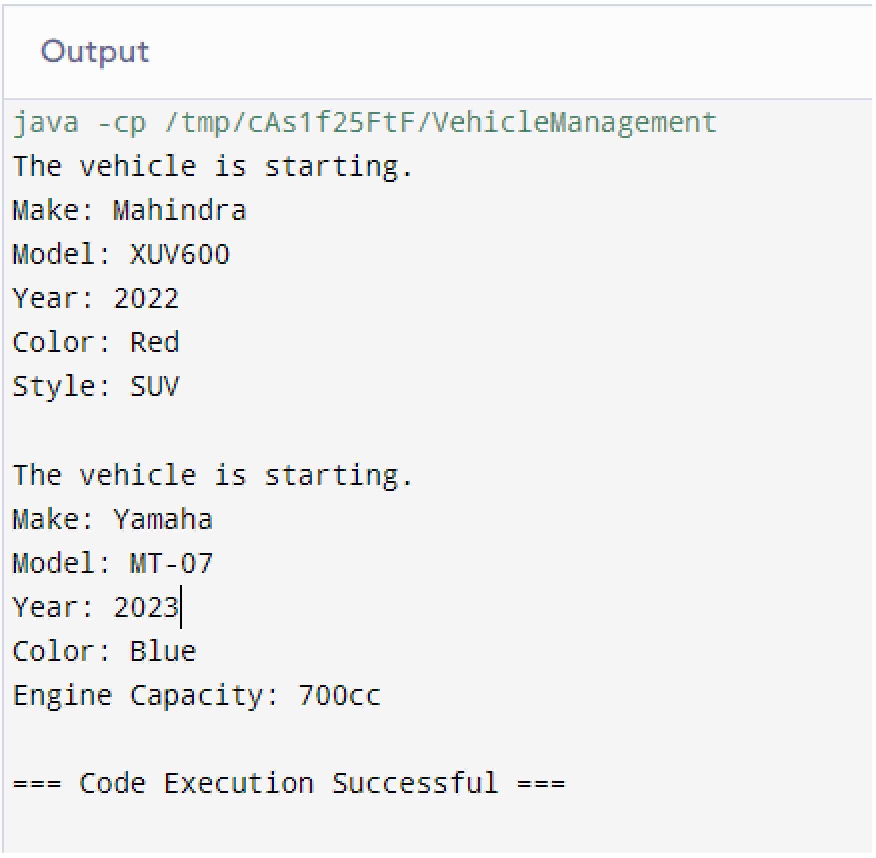
1. Create Vehicle and Car and Motorcycle class.
2. Class Vehicles have 2 methods namely start and display_info. Start indicates the vehicle is working. display_info gives information about basic vehicle attributes like: make: The manufacturer or brand of the vehicle.
   model: The specific model of the vehicle.
   year: The year the vehicle was manufactured.
   color: The color of the vehicle
3. Class Motorcycle and Car , inherit the property of Vehicle class viz. all attributes and methods such as start and display_info.
4. Additionally subclass such as Car has features like Style of Car is override in it viz. Sedan,hatchback,SUV etc. while motorcycle has engine capacity as example 400 CC or 650 CC etc.

Expected Outcome: The program should demonstrate how method overriding can be used to customize behavior and output for different types of vehicles..

Code :
```
class Vehicle {
   String make;
   String model;
   int year;
   String color;

   public Vehicle(String make, String model, int year, String color) {
      this.make = make;
      this.model = model;
```

```
        this.year = year;
        this.color = color;
    }

    public void start() {
        System.out.println("The vehicle is starting.");
    }

    public void display_info() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
        System.out.println("Color: " + color);
    }
}

class Car extends Vehicle {
    String style;

    public Car(String make, String model, int year, String color, String style) {
        super(make, model, year, color);
        this.style = style;
    }

    @Override
    public void display_info() {
        super.display_info();
        System.out.println("Style: " + style);
    }
}

class Motorcycle extends Vehicle {
    String engineCapacity;

    public Motorcycle(String make, String model, int year, String color, String
engineCapacity) {
    super(make, model, year, color);
    this.engineCapacity = engineCapacity;
    }

    @Override
    public void display_info() {
    super.display_info();
    System.out.println("Engine Capacity: " + engineCapacity);
    }
}

public class VehicleManagement {
    public static void main(String[] args) {
        Car car = new Car("Mahindra", "XUV600", 2022, "Red", "SUV");
        Motorcycle bike = new Motorcycle("Yamaha", "MT-07", 2023, "Blue", "700cc");

        car.start();
        car.display_info();
        System.out.println();

        bike.start();
        bike.display_info();
    }
}
```

Output :



Output

```
java -cp /tmp/cAs1f25FtF/VehicleManagement
The vehicle is starting.
Make: Mahindra
Model: XUV600
Year: 2022
Color: Red
Style: SUV

The vehicle is starting.
Make: Yamaha
Model: MT-07
Year: 2023
Color: Blue
Engine Capacity: 700cc

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

Post lab Questions:

1. What is the role of super in method overriding?

When you override a method in a subclass, the `super` keyword is used to call the method from the parent class. This is useful if you want to extend or modify the behavior of the parent method rather than completely replacing it. For example, if the parent class has a method `display()` and you override it in the child class, using `super.display()` inside the child's `display()` method will first execute the parent's version before any additional code.

2. What are the differences between method overriding and method overloading?

   Method Overriding : This happens when a subclass provides a specific implementation of a method that is already defined in its parent class. The method in the subclass must have the same name, return type, and parameters as in the parent class. It allows a subclass to define behavior specific to its type.
   Method Overloading : This is when you create multiple methods with the same name but with different parameters (different type, number, or both) in the same class. It allows methods to handle different data types or a different number of inputs.

3. What is the importance of static in java. Explain its usage for declaring variables, methods and class with suitable examples.

   The `static` keyword in Java means that the particular member (variable, method, or class) belongs to the class itself rather than instances of the class. Here's how it works:
   Static Variable: A static variable is shared among all instances of a class. For example, if you have a `static int count;` in a class, every instance of that class will share the same `count` variable. This is useful when you want to keep track of something at the class level.
   Static Methods: A static method belongs to the class and can be called without creating an instance of the class. For example, `Math.max()` is a static method, meaning you don't need to create a `Math` object to use it.
   Static Class: In Java, you can declare a class as static, but only nested classes can be static. A static nested class cannot access non-static members of the outer class directly. It's used when you want to logically group a class but not tie it to an instance of the outer class.

4. What is the final keyword? Where can we use final keywords ? Explain in short.

   The `final` keyword in Java is used to declare constants, prevent method overriding, or inheritance. Here's where you can use it:
   Final Variables: When a variable is declared as `final`, it can't be modified once it's initialized. It becomes a constant.
   Final Methods: If you declare a method as `final`, it cannot be overridden by subclasses. This is useful when you want to prevent any changes to a method's behavior in inherited classes.
   Final Class: A class declared as `final` cannot be extended. This is used to prevent inheritance, ensuring that the class's behavior remains unchanged. For example, the `String` class is `final` to prevent others from altering its functionality.

Outcomes:

CO1: Apply fundamental Object Oriented Methodology concepts using java programming

CO2: Apply String manipulation functions , inheritance and polymorphism using Java

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

I learned how to implement basic object-oriented principles in Java, specifically working with classes and inheritance. I gained an understanding of how to create a superclass and extend its functionality with subclasses to represent different types of vehicles.

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.