WEEK 11

Inheritance: Method Overriding

Nurnasran Puteh

Outline

- Child class revisited
- Method overriding concept
- Calling overridden parent method
- Preventing Method Overriding
- Overriding vs Overloading

Learning Objectives

- To explain about method overriding
- To explain how to override method in Java
- To show how to call parent method that has been overridden in the child using super keyword
- To differentiate between overriding and overloading

Child class revisited

- A child class inherits data and methods from the parent class (except the constructors)

- In addition, the child class can also:
  - Add new data members
  - Add new methods

  -If a child overloads its parent method, it is adding a new method (that has similar name but different parameter signatures)

Child class revisited

What if a child wants to use a method (inherited from its parent) but the method is not suitable to the child in terms of its implementation?

Note: The child can't create a new method because if it does, it has to change the signature of the method.

Example:

Suppose we add a method in the parent named **displayInfo()** that can display all the data (name and age) of a person object

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void displayInfo() {
        System.out.println("Name = " +getName());
        System.out.println("Age = " +getAge());
    }
    public void setName(String name) {
```

Test Program (with parent object) :

```
public class TestProgram {

    public static void main(String[] args) {
        Person p1 = new Person("Ali Ahmad",30);
        System.out.println("Person Info:");
        p1.displayInfo();
    }
}
```

Output

```
run:
Person Info:
Name = Ali Ahmad
Age = 30
BUILD SUCCESSFUL (total time: 2 seconds)
```

Inherited method not fully suits the child' s need

- Being the child class, Student, will also inherit the Person displayInfo() method

- If the Student object call the displayInfo() method, it gets the behaviour expected from the method in Person object, which displays the Student's name & age only

- Other data of Student object, i.e. studID & mark, will not be displayed

Test Program (with parent & child object) :

```
public class TestProgram {
    public static void main(String[] args) {
        Person p1 = new Person("Ali Ahmad",30);
        System.out.println("Person Info:");
        p1.displayInfo();
        Student s1 = new Student("Siti Hajar", 22, 10001, 80.0);
        System.out.println("Student Info:");
        s1.displayInfo();
    }
}
```

Output

```
run:
Person Info:
Name = Ali Ahmad
Age = 30
Student Info:
Name = Siti Hajar
Age = 22
BUILD SUCCESSFUL (total time: 1 second)
```

displayInfo() method can only display the name and age of a Student object

Method Overriding

In this case, a child class needs to modify the methods that it inherits from the parent class

- This is known as method overriding
- When a child overrides its parent method, it actually changes the implementation of the method (only change the body part) but not the signature of the method
- Thus, signature of the parent method is similar with the signature of the child method
- Note: the child only modifies the inherited method, not creating a new one.

Example:

The Student class need to override the displayInfo() method of the Person class so that all data about Student object can be displayed:

```
public void displayInfo() {
    System.out.println("Name = " + getName());
    System.out.println("Age = " + getAge());
    System.out.println("Stud ID = " + getID());
    System.out.println("Mark = " + getMark());
}
```

Example:

The overridding displayInfo() method is added in Student class:

```
public class Student extends Person {

    private int studID;
    private double mark;

    public Student(String name, int age, int studID, double mark) {
        super(name, age);
        this.studID = studID;
        this.mark = mark;
    }
    public void displayInfo() {
        System.out.println("Name = " + getName());
        System.out.println("Age = " + getAge());
        System.out.println("Stud ID = " + getID());
        System.out.println("Mark = " + getMark());
    }
    public void setID(int matric) {
```

Run Test Program (with parent & child object) again:

```
public class TestProgram {
    public static void main(String[] args) {
        Person p1 = new Person("Ali Ahmad",30);
        System.out.println("Person Info:");
        p1.displayInfo();
        Student s1 = new Student("Siti Hajar", 22, 10001, 80.0);
        System.out.println("Student Info:");
        s1.displayInfo();
    }
}
```

Calling parent version of displayInfo() method

Calling child version of displayInfo() method

Run Test Program (with parent & child object) again:

Output

```
run:
Person Info:
Name = Ali Ahmad
Age = 30
Student Info:
Name = Siti Hajar
Age = 22
Stud ID = 10001
Mark = 80.0
BUILD SUCCESSFUL (total time: 2 seconds)
```

Overridding displayInfo() method can now display all data of Student object

Calling Parent's method that has been overridden

- A child class can call the parent's method that it has overridden by using the super keyword
- Syntax:

```
super.methodName();
```

- Unlike the case when super() is used in a constructor, in this case, super does not have to be the first statement.

Compare the two methods:

The Person class displayInfo() method:

```
public void displayInfo() {
    System.out.println("Name = " +getName());
    System.out.println("Age = " +getAge());
}
```

The Student class displayInfo() method:

```
public void displayInfo() {
    System.out.println("Name = " + getName());
    System.out.println("Age = " + getAge());
    System.out.println("Stud ID = " + getID());
    System.out.println("Mark = " + getMark());
}
```

Compare the two methods:

The Person class displayInfo() method:

```
public void displayInfo() {
    System.out.println("Name = " +getName());
    System.out.println("Age = " +getAge());
}
```

The Student class displayInfo() method:

```
public void displayInfo() {
    System.out.println("Name = " + getName());
    System.out.println("Age = " + getAge());
    System.out.println("Stud ID = " + getID());
    System.out.println("Mark = " + getMark());
}
```

These tasks are performed in the parent method but are repeated in the child

We can replace these by calling the parent's **displayInfo()** method

Using the super keyword

The Person class displayInfo() method:

```
public void displayInfo() {
    System.out.println("Name = " +getName());
    System.out.println("Age = " +getAge());
}
```

The Student class displayInfo() method:

```
public void displayInfo() {
    super.displayInfo();
    System.out.println("Stud ID = " + getID());
    System.out.println("Mark = " + getMark());
}
```

Calling the parent displayInfo() method

Preventing Method Overriding

Methods stated as final cannot be overridden

A child class cannot override a method that is declared as **final** in the parent class.

Example:

The Person class displayInfo() method:

```
public final void displayInfo() {
    System.out.println("Name = " +getName());
    System.out.println("Age = " +getAge());
}
```

Error because parent's **displayInfo()** has been declared as final .

The Student class displayInfo() method:

```
public void displayInfo() {
    super.displayInfo();
    System.out.println("Stud ID = " + getID());
    System.out.println("Mark = " + getMark());
}
```

Overriding vs Overloading

Overriding	Overloading
Methods are in different classes related by inheritance	Methods can be either in the same class or different classes related by inheritance
Requires signatures to be identical (name and return type)	Requires signatures to be different (same name but a different parameter list)
Provides programmer with the ability to redefine the behavior of a particular method in a subclass	Allows programmers to use the same name for two distinct methods

Summary

- A child can modifies or overrides an inherited methods if its implementation is not suitable to the child.
- An overridden method has similar signature with the parent method but has different implementation
- Use the ‘super’ keyword in the child class to call the parent method that has been overridden in the child class.
- ‘final’ methods cannot be overridden
- Unlike overloading, overriding must be related to inheritance.