Lab Manual

Programming with Python and Java (CS 29008)

School of Electronics Engineering, KIIT DU

Chapter 5

Week 3: Inheritance in Java

Inheritance in Java is a mechanism by which an object acquires all the properties and behaviors of a parent object. It is an important feature of the object-oriented programming language.

The idea behind inheritance in Java is that we can create new classes that are built upon existing classes. When we inherit from an existing class, we can reuse the members of the existing class into the present class. The existing class is called the **parent class**, while the class that inherits from the parent class is known as the **child class**. Furthermore, we can include new methods and variables in the child class to define its distinct properties and characteristics.

Inheritance facilitates method overriding[1] and promotes the reuse of code.

5.1 Important terminology

Here are some key terms commonly used to describe the relationships among different classes.

Class: A class is a collection of objects that share common properties. A class acts as a template from which objects are instantiated.

Parent class: A parent class is the class from which a subclass inherits the features. It is also called a **base class** or a **superclass**.

Child class: A child class is the class which inherits the features from other classes. It is also called a **derived class** or a **subclass** or an **extended class**.

[1]Method overriding will be discussed in the upcoming labs.

5.2 Syntax of inheritance

The syntax of inheritance in Java is

```
class ChildClassName extends ParentClassName{
   // child class body
}
```

The **extends** keyword indicates that we are making a new class that derives from an existing class. The meaning of **extends** is to increase functionality. In other words, by utilizing the **extends** keyword, the child class acquires the properties and behaviors (including methods and variables) of the parent class. This enables the reuse of code and the establishment of a hierarchical structure.

5.3 Types of inheritance in Java

There are three types of inheritance in java:

1. single inheritance
2. multi-level inheritance
3. hierarchical inheritance

Note that in addition to the three aforementioned types of inheritance, there are two additional types- multiple and hybrid inheritances, which are exclusively supported through interfaces. Detailed discussions on interfaces will be covered in upcoming labs.

5.3.1 Single inheritance

When a class inherits another class, it is known as a **single inheritance**.

In the example given below, **Student** class inherits the **Person** class which is an example of the single inheritance.

Listing 5.1: An example of single inheritance

```
class Person {
void name(){
System.out.println("Name is...");
}  }
class Student extends Person {
void roll(){
```

```
System.out.println("Roll is...");
} }
class Main {
public static void main(String args[]) {
Student stud = new Student();
stud.roll();
stud.name();
}}
```

After executing the above program, we get the following output:

Roll is...
Name is...

5.3.2 Multi-level inheritance

When there is a chain of inheritance, it is known as **multi-level inheritance**. As we can see in the example given below, **FirstYear** class inherits the **Student** class which again inherits the **Person** class, so this is a multi-level inheritance.

Listing 5.2: An example of multi-level inheritance

```
class Person {
void name(){
System.out.println("Name is...");
} }
class Student extends Person {
void roll(){
System.out.println("Roll is...");
} }
class FirstYear extends Student {
void year(){
System.out.println("1st year...");
} }
class Main {
public static void main(String args[]) {
FirstYear stud = new FirstYear();
stud.year();
stud.roll();
stud.name();
}}
```

After executing the above program, we get the following output:

1st year...
Roll is...
Name is...

5.3.3 Hierarchical inheritance

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, **Student** and **Teacher** classes inherit the **Person** class.

Listing 5.3: An example of hierarchical inheritance

```
class Person {
void name(){
System.out.println("Name is...");
} }
class Student extends Person {
void roll(){
System.out.println("Roll is...");
} }
class Teacher extends Person {
void empid(){
System.out.println("ID is...");
} }
class Main {
public static void main(String args[]) {
Teacher t = new Teacher();
t.empid();
t.name();
}}
```

After executing the above program, we get the following output:
ID is...
Name is...

5.4 Week 3 Exercises

1. Modify the Java program shown in Listing 5.1 to do the following:

    1. Add a function **void gender()** to class **Person** that prints the statement "Gender is..."
    2. Add a function **void branch()** to class **Student** that prints the statement "Branch is ECSC".

2. Modify the Java program shown in Listing 5.2 to do the following:

    1. Add a class **FirstSem** that is an extension of class **FirstYear**. Add a function **void subjects()** to the class **FirstSem** that displays "6 theory courses and 2 lab courses". Compile and run the program. Apply the concept of multi-level inheritance.
    2. Add three more classes, **SecondYear**, **ThirdYear**, **FourthYear** which are extensions of class **Student**. Add three functions, **void year2()**, **void year3()**, **void year4()** to three new classes **SecondYear**, **ThirdYear**, **FourthYear** respectively. The function **void year2()** displays "2nd year...", function **void year3()** prints "3rd year...", and function **void year4()** shows "4th year..." Apply the concept of hierarchical inheritance.

3. Modify the program written for Problem 2 to do the following:

    1. Add variables to each of the class. Add a no-argument constructor in each of the classes to initialize the variables to default values (0 for integers and NULL/EMPTY for strings).
    2. Add parameterized constructors in each class to assign user inputs to the member variables.

4. Using a multi-level inheritance, write a Java program to implement the relationship shown in Figure 5.1. Also, include constructors in every class to initialize the member variables.

5. Using a hierarchical inheritance, write a Java program to implement the relationship shown in Figure 5.2. Also, include constructors in every class to initialize the member variables.

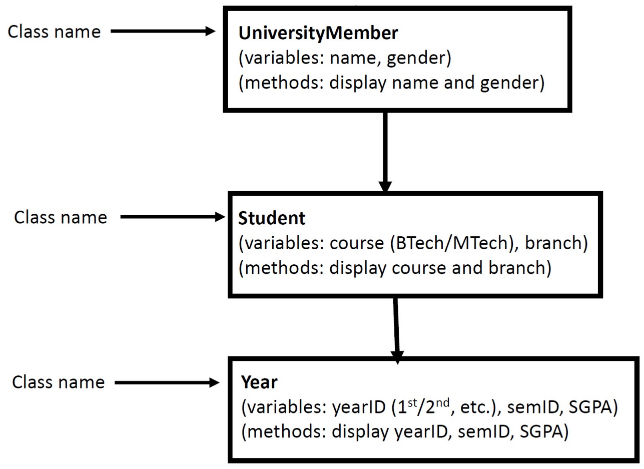
Class name
UniversityMember
(variables: name, gender)
(methods: display name and gender)
Class name
Student
(variables: course (BTech/MTech), branch)
(methods: display course and branch)
Class name
Year
(variables: yearID (1st/2nd, etc.), semID, SGPA)
(methods: display yearID, semID, SGPA)

Figure 5.1: Exercise for multi-level inheritance

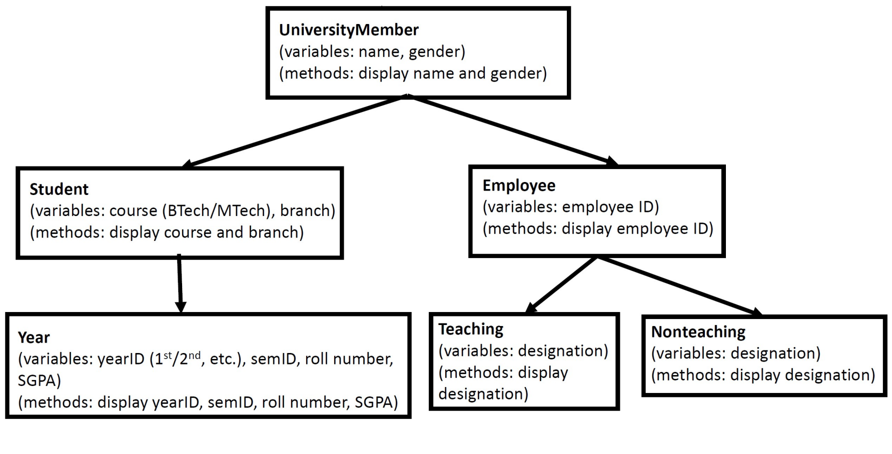
UniversityMember
(variables: name, gender)
(methods: display name and gender)
Student
(variables: course (BTech/MTech), branch)
(methods: display course and branch)
Employee
(variables: employee ID)
(methods: display employee ID)
Year
(variables: yearID (1st/2nd, etc.), semID, roll number, SGPA)
(methods: display yearID, semID, roll number, SGPA)
Teaching
(variables: designation)
(methods: display designation)
Nonteaching
(variables: designation)
(methods: display designation)

Figure 5.2: Exercise for hierarchical inheritance