1DV532 – Starting Out with Java

Lesson 10

Inheritance

Dr. Nadeem Abbas

nadeem.abbas@lnu.se

Linnæus University

Code Reuse

- Suppose we have a class X, which is well tested and works fine.
  - Now, we need a class that provides the same services (behavior) as the class X, and few other services in addition.
- How may we reuse existing code, in this case services of the class X?

Code Reuse and Maintenance

- One way to reuse code from an existing class X is to make a copy of the class X, add missing members to the new copy or make changes to the existing member methods and data members so that the new class provides required services.

- The above approach sounds reasonable, however, may lead to maintenance problems!

Maintenance Problem

- Suppose, we’ve found an error in the original class X and want to fix it.
  - We go into the source code (.java file) for the class X, locate and correct the error there.
  - However, a problem with this approach is that we have to go into all classes that have copied the source code from the class X, because the error is likely to be there as well.
- How to find all classes that have copied the source code from the class X?
- Will the correction work there too?
  - Remember, we've changed the code!

Maintenance Problem

- Suppose we’ve found an error in the class Y which has been copied from the class X, and we want to correct it.
- Should we then go to the class X and fix something there too, or is the problem caused by the changes made in the new code in class Y?
- It's also easy to forget that you have copied a class while fixing your program.

Why Inheritance?

- Copying code from one class to another may lead to large maintenance problems

- We want to reuse services from already defined classes, without being required to copy the source code and make changes in it.
  - We can achieve this using Inheritance!

Inheritance - Introduction

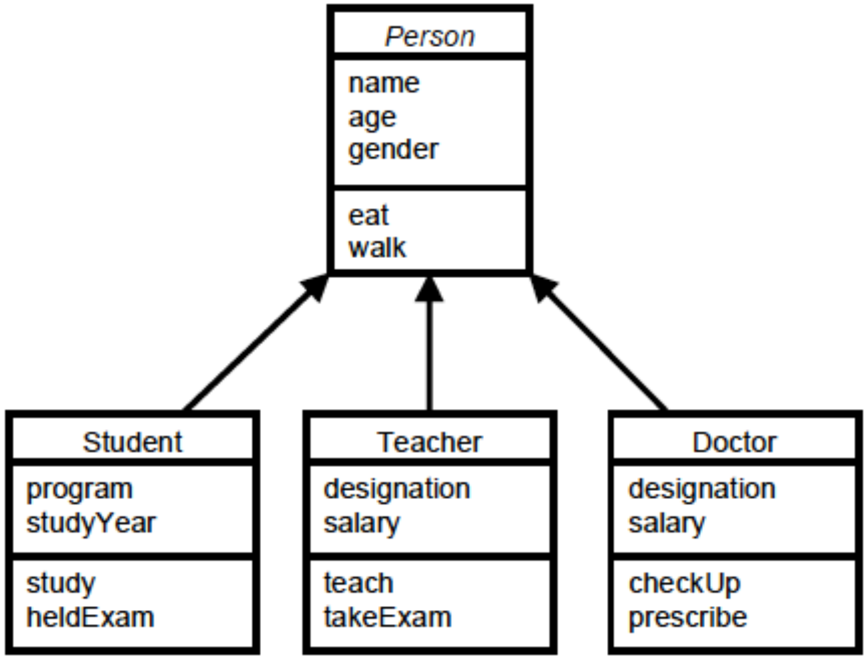
- **Inheritance** is one of the cornerstones of OOP that allows a (sub)class to acquire (inherit) properties (both member methods and data), in addition to the subclass's own specifically declared properties.
  - **Basic Idea:** *Enable new objects to take on and extend properties and behavior of existing objects*

- **Inheritance** is a process by which a new class—known as a ***derived, child, specialized or subclass*** —is created from another class known as ***a base, parent, general or superclass***.

- **Inheritance** supports reusability
  - reuse of data members and member methods from existing classes without having to (re)write (and debug!) them.

Generalization – Specialization Hierarchy

- Inheritance implies a **generalization - specialization** hierarchy
  - First, we define a General Class, for example, Person class, as shown in the figure below.
  - Then, we extend the General Class, Person for example, by adding new members to get a few or more Specialized classes such as Student, Teacher, Employee.



Inheritance and Hierarchies

- When designing certain classes, there is often a natural hierarchy for grouping them
  - For example,
    - **Teachers**, Students, Doctors, **Employees**, etc. all are humans and are often grouped under a general class **Person**.
    - Pre-School Teacher, Primary School Teacher, High School Teacher, etc. all are Teachers and are often grouped under a general Class **Teacher**
    - Regular Employee, Hourly Employee, Contract Employee, etc, all are Employees and are often grouped under a general class **Employee**
    - Circle, Square, Triangle, etc., all are shapes and can be grouped under a general class Shape.

- Inheritance takes advantage of such natural hierarchies by creating a top level Class in a hierarchy and extending the top-level class to get all sub-classes in the hierarchy

Inheritance in Java

- To practice inheritance in Java, **First step is to define a General Class** with all general level attributes (data members) and methods

- Once a general class has been defined, it can be extended to as many number of classes as required.

- How can we extend a (base) class to another (sub) class?
  - See the next slide, for an answer.

Inheritance in Java++

General Syntax:

```
class SubClassName extends BaseClassName{
    //Class Memebers
 }
```

Example:

```
public class Student extends Person {
    private String program, school; // student specific fields
    private ArrayList courses;      // student specific fields

    //constructors
    public Student(String n, String p, int h, int w) {
            super(n, h, w);
            program=p;
    }
    public Student() {        }        // an empty constructor

    // setter/getter methods
    public String getSchool() {return school; }
    public void setSchool(String school) { this.school = school;}

    // … more Student specific methods including getters and setters
}
```

Example Program – PersonDemo.java

```
public class PersonDemo {
  public static void main(String[] args) {
    Student robert= new Student();
    robert.setName("Robert");
    robert.setHeight(170);
    robert.setWeight(78);

    ArrayList <String> robertsCourses=new
    ArrayList<>();
    robertsCourses.add("Foundations of
    Software Technology");
    robertsCourses.add("Software Design");
    robert.setCourses(robertsCourses);
    robert.setProgram("Network Security");

    Teacher anna = new Teacher("Anna",
    "Professor","Mathematics");
    anna.setHeight(180);
    anna.setWeight(75);
    //…
    System.out.println("Total number of Persons initiated: ” +
    Person.getPersonCounter());
    }
}
```

- Example Program, *PersonDemo.java*, demonstrates use of the Student and Teacher classes which are derived (inherited) from Person class.
- Now, if we look at the code lines with bold text, we can see that although we have not defined the data members `name, height weight` and associated member methods `setName, setHeight`, and `setWeight`, in the Student class object, yet the Student type object `robert`, has access to these member methods.
  - *Student* Class inherits these member methods from its parent class that is *Person*.

Inheritance – What We Can Do in a Subclass?

- A subclass inherits all of he *public* and *protected* members of its parent, no matter what package the subclass is in.
- If the subclass is in the same package as its parent, it also inherits the *package-private* members of the parent.
  - *package-private* members are those declared without any access modifier.
- We can use the inherited members as is, replace them, hide them, or supplement them with new members:

Inheritance – What We Can Do in a Subclass?

Inside a Subclass:

- We can use inherited fields directly, just like any other fields.
- We can declare a field in the subclass with the same name as the one in the superclass.
  - This *hides* the field defined in the superclass and is not a recommended practice to do.
- We can declare new fields in the subclass that are not in the superclass.
- The inherited methods can be used directly as they are.
- We can write a new *instance* method in the subclass that has the same signature as the one in the superclass,
  - This is called **method overriding**.
- We can write a new *static* method in the subclass that has the same signature as the one in the superclass, thus *hiding* it.
- We can declare new methods in the subclass that are not in the superclass.
- You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword **super**.

Inheritance and Private Members

- A subclass does not inherit the *private* members of its parent class.
- However, if the superclass has *public* or *protected* methods for accessing its *private* fields, these can also be used by the subclass to access *private* fields of the superclass.
  - Our example subclass *Student* uses public getter and setter methods of its superclass *Person* to access private data member of the Person class.

Overriding a Method Definition

- Although a derived class inherits methods from the base class, it can change or *override* an inherited method if necessary
  - In order to override a method definition, a new definition of the method is simply placed in the class definition, just like any other method that is added to the derived class

Changing the Access Permission of an Overridden Method

- The access permission of an overridden method can be changed from private in the base class to public (or some other more permissive access) in the derived class
- However, the access permission of an overridden method can not be changed from public in the base class to a more restricted access permission in the derived class

Changing the Access Permission of an Overridden Method

- Given the following method header in a base case:

  ```
  private void doSomething()
  ```

- The following method header is valid in a derived class:

  ```
  public void doSomething()
  ```

- However, the opposite is not valid
- Given the following method header in a base case:

  ```
  public void doSomething()
  ```

- The following method header is <u>not</u> valid in a derived class:

  ```
  private void doSomething()
  ```

Pitfall: Overriding Versus Overloading

- Do not confuse ***overriding*** a method in a derived class with ***overloading*** a method
  - When a method is overridden, the new method definition given in the derived class has the exact same number and types of parameters as in the base class
  - When a method in a base or derived class has a different signature from the method in the base class, that is overloading
  - Note that when a derived class overloads the original method, it still inherits the original method from the base class as well.

The `final` Modifier

- If the modifier **`final`** is placed before the definition of a *method*, then that method may not be redefined in a derived class
- It the modifier **`final`** is placed before the definition of a *class*, then that class may not be used as a base class to derive other classes

Inheritance – Multiplicity

- There are languages such as C++ that allow a class to inherit from more than one classes
  - Such inheritance is called **Multiple Inheritance**
- Java does not support multiple inheritance
- As we seen in the example programs, a Java Class can inherit only from one base or superclass
- Multiple inheritance is prohibited in Java because it may lead to the issue of *multiple inheritance of state and behavior*, which is ability to inherit fields and methods from multiple classes.
  - For example, if we derived a class from two or more superclasses where each superclass has a method with same signature. On calling the method with a subclass, the compiler will not be able to determine which class method to be called.

Suggested Readings

- Absolute Java, Global Edition, 6/E by Walter J. Savitch, Chap 7 Inheritance
- Introduction to Java Programming, Brief Version, Global Edition, 11/E Liang, Chap 11 Inheritance and Polymorphism

- Java Tutorials
  - https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Lnu.se