CS1421: Object Oriented Programming Lab

LAB # 06

Instructors
Safiyah Batool
Samia Bashir

Lab 6
Inheritance

Objective

After completing this lab, the students should be able to

- Understand the usage and implementation of inheritance
- Understand usage and implementation of multi-level inheritance
- Understand super keyword

Why use inheritance?

· For Method Overriding (so runtime polymorphism can be achieved).
· For Code Reusability

Inheritance in Java

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features (fields and methods) of another class.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class. Moreover, you can add new methods and fields in your current class also.
Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

§ **Super Class:** The class whose features are inherited is known as super class (or a base class or a parent class).
§ **Sub Class:** The class that inherits the other class is known as sub-class (or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
§ **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

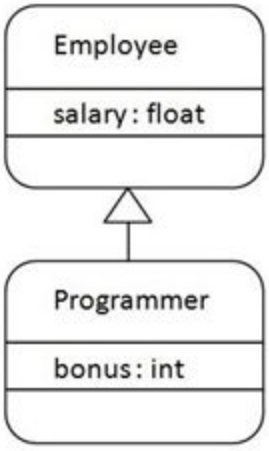
How to use inheritance in Java?

The keyword used for inheritance is **extends.**

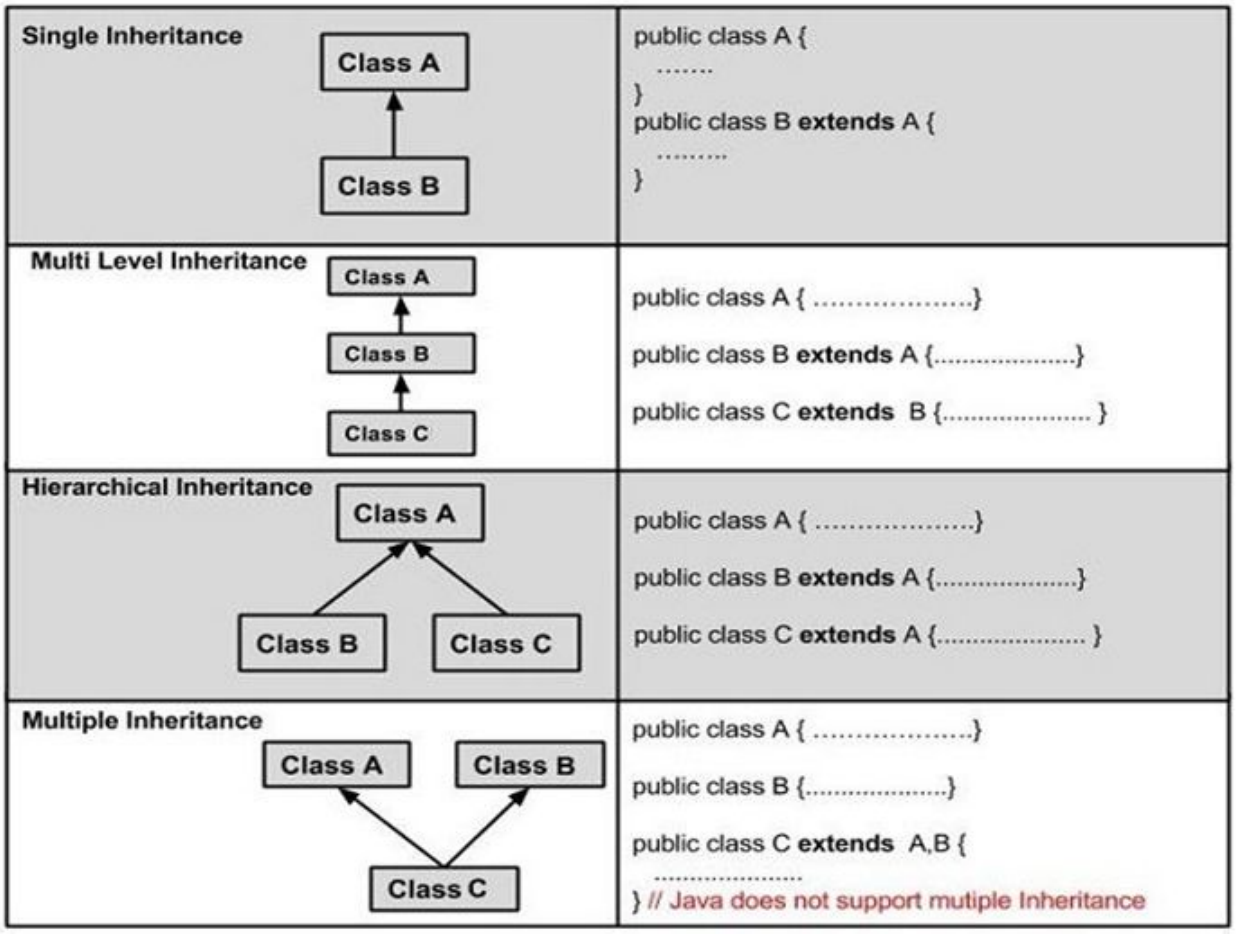
Syntax :

```
class derived-class extends base-class
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.



Types of Inheritance



Single Level Inheritance: Example

```
class Employee{
 float salary=40000;
}

class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Output:

```
Programmer salary is:40000.0
Bonus of Programmer is:10000
```

Multilevel Inheritance : Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
```

```
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

Example:

```
public class Vehicle {
protected int NoOfTyres;
protected String Fuel;
protected boolean engine;
protected int Seats;

public  void start(){
    if(engine){
        System.out.println("Vehicle is
starting....");
    }
    else
        System.out.println("Turn On the Engine
First!");
}

    public boolean isEngine() {
    return engine;
    }
    public void startEngine(){
    engine=true;
    }
    public void turnOffEngine(){
    engine=false;
    }
}
```

Car Is-A type of Vehicle so, it extends Vehicle class.

```
public class Car extends Vehicle {
    Car(){
        NoOfTyres=4;
        Seats=5;
    }
    Car(String TypeOfFuel){
        this.Fuel=TypeOfFuel;
        NoOfTyres=4;
        Seats=5;
    }

}
```

Autonomous Car is a type of Vehicle and Car so it entends Car.

```
public class AutonomousCars extends Car {
    int LevelOfAutomation; //0 to 5
    AutonomousCars(String TypeofFuel){
        Fuel=TypeofFuel;
    }
}
```

Test Class

```
public class TestVehicle {
    public static void main(String[] args) {
        Car c= new Car("Petrol");
        c.start();
        c.startEngine();
        c.start();
        c.turnOffEngine();
        c.start();
    }
}
```

Hierarchical Inheritance : Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Relation between classes in Inheritance??

IS-A Relationship

IS-A is a way of saying: This object is a type of that object. Let us see how the **extends** keyword is used to achieve inheritance.

```
public class Animal {

}

public class Mammal extends Animal {

}

public class Reptile extends Animal {

}

public class Dog extends Mammal {

}
```

in Object-Oriented terms, the following are true −

- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say −

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence: Dog IS-A Animal as well

Example:

we will consider a simple hierarchy of classes. Most likely, you have taken computer graded quizzes. A quiz consists of questions, and there are different kinds of questions:

• Fill-in-the-blank

• Choice (single or multiple)

• Numeric (where an approximate answer is ok; e.g., 1.33 when the actual answer is 4/3)

• Free response

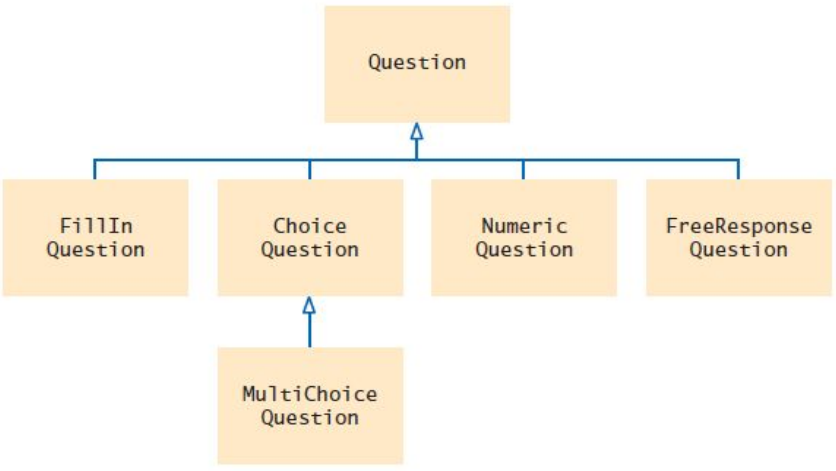


Figure 3
Inheritance Hierarchy of Question Types

At the root of this hierarchy is the Question type. A question can display its text, and it can check whether a given response is a correct answer.

Question.java

```
public class Question {
/**
A question with a text and an answer.
*/
        private String text;
        private String answer;
       /**
         Constructs a question with empty question and answer.
         */
        public Question()
       {
            text = "";
            answer = "";
            }

        /**
         Sets the question text.
         @param questionText the text of this question
         */
        public void setText(String questionText)
       {
            text = questionText;
            }

        /**
         Sets the answer for this question.
         @param correctResponse the answer
        */
        public void setAnswer(String correctResponse)
       {
            answer = correctResponse;
            }

   /**
         Checks a given response for correctness.
         @param response the response to check
         @return true if the response was correct, false
otherwise
         */
        public boolean checkAnswer(String response)
        {
           return response.equals(answer);
          }

        /**
         Displays this question.
```

```
         */
        public void display()
       {
            System.out.println(text);
            }
}
```

This question class is very basic. It does not handle multiple-choice questions, numeric questions, and so on. In the following sections, you will see how to form subclasses of the Question class.
Here is a simple test program for the Question class:

```
import java.util.ArrayList;
import java.util.Scanner;

public class TestQuestion {

         /**
5 This program shows a simple quiz with one
question.
6 */

        public static void main(String[] args)
       {
            Scanner in = new Scanner(System.in);

            Question q = new Question();
            q.setText("Who was the inventor of
Java?");
            q.setAnswer("James Gosling");

            q.display();
            System.out.print("Your answer: ");
            String response = in.nextLine();

System.out.println(q.checkAnswer(response));
       }

}
```

```
Who was the inventor of Java?
Your answer: James Gosling
true
```

Multiple Inheritance:

```
public class fish extends Animal, Mammal{}
```

Example above is illegal in Java as it does not support multiple inheritance

The super keyword

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
- It is used to **invoke the superclass** constructor from subclass.

Differentiating the Members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

the private instance variables of the superclass are inaccessible. Because these variables are private data of the superclass, only the superclass has access to them. The subclass has no more access rights than any other class.

Syntax:

```
super.variable
super.method();
```

Example:

```
class Super_class {
  int num = 20;
```

```
  // display method of superclass
  public void display() {
        System.out.println("This is the display method of superclass");
  }
}

public class Sub_class extends Super_class {
  int num = 10;

  // display method of sub class
  public void display() {
        System.out.println("This is the display method of subclass");
  }

  public void my_method() {
        // Instantiating subclass
        Sub_class sub = new Sub_class();

        // Invoking the display() method of sub class
        sub.display();

        // Invoking the display() method of superclass
        super.display();

        // printing the value of variable num of subclass
        System.out.println("value of the variable named num in sub class:"+
sub.num);

        // printing the value of variable num of superclass
        System.out.println("value of the variable named num in super class:"+
super.num);
  }

  public static void main(String args[]) {
        Sub_class obj = new Sub_class();
```

```
        obj.my_method();
    }
}
```

Output:

```
This is the display method of subclass
This is the display method of superclass
value of the variable named num in sub class:10
value of the variable named num in super class:20
```

Invoking Superclass Constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the superclass. But if you want to call a parameterized constructor of the superclass, you need to use the super keyword as shown below.

Syntax:

```
super(values);
```

Example:

```
class Superclass {

    int age;

    Superclass(int age) {

        this.age = age;

    }

    public void getAge() {

        System.out.println("The value of the variable named age in super class is: "
+age);

    }

}
```

```
public class Subclass extends Superclass {

   Subclass(int age) {

       super(age);

   }


   public static void main(String argd[]) {

       Subclass s = new Subclass(24);

       s.getAge();

   }

}
```

Output

```
The value of the variable named age in super class is: 24
```

Practice Tasks

Note: Draw UML class hierarchy diagram for all tasks.

Walkthrough Tasks:
Task 1:
Implement a superclass Person. Make two classes, Student and Instructor, that inherit from Person. A person has a name and a year of birth. A student has a major, and an instructor has a salary. Write the class declarations, the constructors, and the methods toString for all classes. Supply a test program that tests these classes and Methods.

Task 2:
a) Add a Multiple Choice Question to the question hierarchy of Question Class. If the response and the expected option matches, then accept the response as correct.

b) Add a class NumericQuestion to the question hierarchy of Question Class. If the response and the expected answer differ by no more than 0.01, then accept the response as Correct.
c) Modify the checkAnswer method of the Question class so that it does not take into account different spaces or upper/lowercase characters. For example, the response "JAMES gosling" should match an answer of "James Gosling".

Practice Tasks:

Task 1

IAL Saatchi & Saatchi (Pakistani large advertising firm) have approached the students of MAJU (especially the OOP students) to design and create the future episodes of Commander Safeguard animated series. They have given you general information about some characters. Now you have to understand the characters properly and design a system which clearly represents the nature of many characters.

The information from IAL Saatchi & Saatchi is as below:

We have many characters in our series.
Each character has a name.
Every character can move and can speak.

Germs are one kind of character in our series.
The difference is that every germ has a disease.

The germ can fire a disease and/or can spread the disease.

There are some special germs that can fly.

The characters other than Germs are human.
Every human can wash hands.

There are basically two types of Humans; Protectors
Protector includes Commander Safeguard along with his team that may contain some special kids and doctors.

Using information given above design some classes using the concept of inheritance to get the contract of being the animator of world known animated series.

Task 2:

Implement a superclass Appointment and subclasses Onetime, Daily, and Monthly. An appointment has a description (for example, "see the dentist") and a date. Write a method occursOn(int year, int month, int day) that checks whether the appointment occurs on that date. For example, for a monthly appointment, you must check whether the day of the month matches. Then fill an array of Appointment objects with a mixture of appointments. Have the user enter a date and print out all appointments that occur on that date.

Task:

Improve the appointment book program. Give the user the option to add new appointments. The user must specify the type of the appointment, the description, and the date.

Task :

Improve the appointment book program by letting the user save the appointment data to a file and reload the data from a file. The savingpart is straightforward: Make a method save. Save the type, description, and date to a file. The loading part is not so easy. First determine the type of the appointment to be loaded, create an object of that type, and then call a load method to load the data.

Evaluation Task

Task 1

Hyperstar is Pakistan's most dynamic, fast-moving and exciting hypermarket chain. The company has global expertise which helps them offer the shoppers here in the Pakistan the same quality, variety and value-for-money that is provided all over the world. Hyperstar Pakistan has hired you to represent following products available in their store through classes. The classification required is as follows:

1. All the products have a productId and a name.

2. Utility products have name, productId and discount. Also calculate the discount for each product.
3. Food products having attributes name, productId and dateOfExpiry.

Also define a method to show data in each class. Your code should avoid redundancy and be reusable wherever possible.