Advance Programming with Java

Prativa Nyaupane

Recap

- Access Modifiers
- Exception Handling
- Java Collections
- Review of object oriented principles

Today's Objectives

- Abstraction
- Super class, sub class, inheritance and member access
- Extends and super keyword
- Types of inheritance
- Overriding/Overloading
- Final classes and methods

Abstraction

- Abstraction is also an object oriented approach where we hide the implementation details of a code and expose only the necessary information to user.
- Java Abstract class is used to provide common method implementation to all the subclasses or to provide default implementation.(NW Call, DB Connection)
- Abstraction can be achieved with either abstract classes or interfaces.
- The abstract keyword is a non-access modifier, used for classes and methods.
- Abstract class : It is a restricted class that cannot be used to create objects. To access it, it must be inherited from another class.
- Abstract method: It can only be used in abstract class, and it does not have a body. The body is provided by the subclass(inherited from).

```java
abstract class Student {
    private String name;

    public Student(String name){
        this.name = name;
    }

    //abstract method for attending class
    public abstract void attendClass();

    //getter for name
    public String getName(){
        return this.name;
    }

    //setter for name
    public void setName(String name){
        this.name = name;
    }

}
```

```java
class UndergraduateStudent extends Student{ no usag

}
```


```java
class UndergraduateStudent extends Student{
    public UndergraduateStudent(String name){
        super(name);
    }

    @Override
    public void attendClass(){
        System.out.println(getName() + "is attending
class");
    }
}
```

```
public class Main {
   public static void main(String[] args) {
       UndergraduateStudent undergraduateStudent = new
UndergraduateStudent("Ram");
       undergraduateStudent.attendClass();

   }
}
```

- In this example, Student class is declared as abstract with common attribute “name”. It also has an abstract method “attendClass()” which is implemented by subclass Undergraduate.
- **Why abstract class?**
  - Simplify complex system by focusing only on the essential aspects.
  - Allows encapsulation of implementation details.
  - Enhances dependability by reducing the dependency between different parts of the system.

```
public abstract MyAbstractProcess {
        public void stepBefore() {
         // Implementation directly in abstract superclass
        }
        public abstract void action(); // Implemented by subclasses
        public

        void stepAfter() {
         // Implementation directly in abstract superclass
        }
}
```

Inheritance

- A new class objects can be created conveniently by inheritance - the new class(called the subclass) starts with the characteristics of an existing class(called the superclass) by using extend keyword.
- The subclass can also be customized, so that we can add unique characteristics of its own.
- This promotes code reuse.
- Inheritance enables polymorphism behavior through method overriding.
- Inheritance allows abstraction. We can create abstract classes that define the common attributes and behaviors. These can be shared by multiple subclases.
- Provides hierarchical structure, where subclasses are organized under the superclass.

super keyword

- super keyword is used to refer to the immediate parent class of a subclass.
- super keyword is a reference to the superclass in Java.
- It provides mechanisms for accessing superclass members, constructors and overriden methods within a subclass.
- Accessing superclass member:

extends keyword

- In Java, extends keyword is used to create a subclass that inherits attributes and methods from another class, known as superclass.

```
class SuperClass {
    // some fields and methods
     void superClassMethod(){
               System.out. println(“Super class method called.”)
     }
}
class SubClass extends SuperClass {
    // SubClass inherits all accessible fields and methods from SuperClass
     Void subClassMethod(){
               System.out. println(“Sub class method called”)
     }
}
public class Main(){
     public static void main(String argos[]){
               SubClass subClassObject = new SubClass();
               subClassObject.subClassMethod();
               subClassObject.superClassMethod();
}
}
```

Super Class

- The class from which the sub class is derived is called a superclass(also called base class or parent class).
- Every class can have one and only one direct superclass.

```
public class Animal {

    String sound;

    Animal(String sound) {
        this.sound = sound;
    }

    public static void main(String[] args){
        System.out.println("Inheritance
example");
    }
}
```

Sub Class

- A class that is derived from another class is called a sub class(also called a derived class, extended class, or child class)

```
public class Dog extends Animal{

    // super() method can act like the
parent constructor inside the child class
constructor.
    Dog(){
        super("woof");
    }

}
```

```
class Superclass {
    int number = 10;
    void display() {
        System.out.println("Superclass method");
    }
}

class Subclass extends Superclass {
    int number = 20;
    void display() {
        System.out.println("Subclass method");
        System.out.println("Value of number in Superclass: " + super.number); // Accessing superclass
field
        super.display(); // Calling superclass method
    }
}
```

- Accessing superclass constructor

```
class Superclass {
    Superclass() {
        System.out.println("Superclass constructor");
    }
}

class Subclass extends Superclass {
    Subclass() {
        super(); // Calling superclass constructor
        System.out.println("Subclass constructor");
    }
}
```

Types of Inheritance

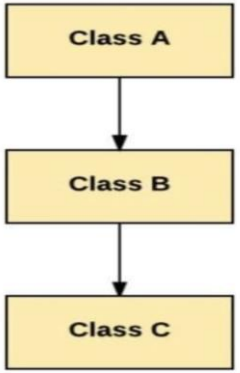
Single Inheritance: When there is only one derived class.

```
class A {

}
class B extends A {

}
```

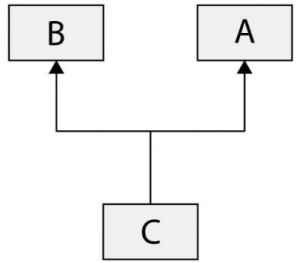
Multilevel Inheritance: There is a level of inheritance. Class B is derived from Class A and Class C is derived from Class B.



Types of Inheritance Contd...

Multiple Inheritance: When a derived class is inheriting from more than one parent class.

```
class A {
}
class B {
}
class C extends A, B {
}
```



Java does not support multiple inheritance through Classes primarily to avoid the complexities associated with it, such as the diamond problem.

Diamond problem creates ambiguity during method calls. In above example, when class C inherits from class B and A, the compiler does not know whether to inherit from B or A.

Multiple Inheritance Contd..

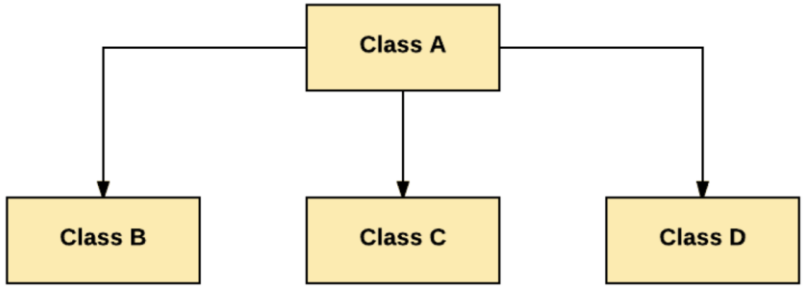
In this diagram, classes B and C both inherit from class A, and class D inherits from both B and C.

The problem occurs when class D inherits a method or attribute from class A, but classes B and C also override that method or attribute differently. This creates ambiguity for the compiler, which cannot determine which implementation to use.

Programming languages like Java avoid this problem by allowing single inheritance for classes but permitting multiple inheritance through interfaces. Interfaces in Java only declare method signatures, not implementations, so there's no risk of conflicting behavior.

Hierarchical Inheritance:

One class is inherited by multiple subclasses.



In the above example, we can see that the three classes Class B, Class C, and Class D are inherited from the single Class A. All the child classes have the same parent class in hierarchical inheritance.

Overloading

a. If a class has multiple methods having same name but different parameters, it is method overloading.
b. Their parameters can differ in number, type or order.
c. Java determines which overloaded method to use based on number and type of parameters passed during the method invocation.
d. Is resolved at compile time.
e. It is a form of static polymorphism.

Overriding

a. If a subclass has same method as parent, it is called method overriding.
b. The method signature of the subclass must match the method signature of the super class.
c. Uses @override keyword.
d. It is a form of runtime polymorphism.
e. Which method is called is determined dynamically based on the object's type.

Method Overloading

```
public class Student {
   private String name;
   private int rollNumber;
   public Student() {
   }
   public void setStudentAtrributes(String name) {
       this.name = name;
   }

   public void setStudentAttributes(String name,
int rollNumber){
       this.name = name;
       this.rollNumber = rollNumber;
   }

}
public class Main {
   public static void main(String[] args) {

       //Student student = new Student();
       Student student = new Student();
       student.setStudentAtrributes("Ram Poudel");
       student.setStudentAttributes("Ram
Paudel",1);


   }
}
```

Method Overriding

```
public class Animal {

   public void greeting() {
       System.out.println("The animal greets you.");
   }


}
public class Dog extends Animal {

   public void greeting() {
       System.out.println("The dog barks.");
   }

}
public class Example {

   public static void main(String[] args) {

       Animal animal = new Animal();  // Animal object
       animal.greeting(); // prints "The animal greets
       Animal dog = new Dog();  // Dog object
       dog.greeting(); // prints "The dog barks

   }
}
```