Lesson:

Inheritance

PW SKILLS

List of Concepts Involved:

- Inheritance introduction
- extends keyword
- Types of Inheritance
- Important key points (5 key points)
- Types of methods Inherited, overridden, specialised
- Rules to override method
- Constructor execution in case of Inheritance

Inheritance Introduction

- It is one of the pillars of Object Orientation.
- It always speaks about reusability.
- Using inheritance productivity of the code can be improved.
- If we use inheritance, lines of code can be reduced in the application.
- In java inheritance is achieved through the "extends" keyword.

Extends keyword

If we use extends keyword, then we can take the properties and behaviours from parent class to child class.

Example

```
class Parent{
    public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
    public void methodTwo(){System.out.println("methodTwo from child");}
 }
 public class TestApp{
        public static void main(String... args){
               Parent p=new Parent();
                p.methodOne();

               Child c=new Child();
                c.methodOne();
                  c.methodTwo();

        }
 }
```

Output

methodOne from parent
methodOne from parent
methodOne from child

Note:

- Whatever is there in the Parent class by default would be available to the Child class.
- Whatever is there in the Child class by default won't be available to the Parent class.
- On Child reference we can call both Parent class and Child class methods whereas using Parent reference we can call only Parent class methods.

Loan activities

The common methods which are required for HousingLoan,EducationLoan,VehicleLoan are defined in separate class Loan class and it can be reused so that production cost can be increased effectively and time consumption for building projects can be reduced.

Example

```
class Loan{
     //common methods for every type of  loan is written here
}
class VehicleLoan extends Loan{
    //specific methods for VehicleLoan is written here
}
class EducationLoan extends Loan{
   //specific methods for EducationLoan is written here
}
class HomeLoan extends Loan{
   //specific methods for EducationLoan is written here
}
```

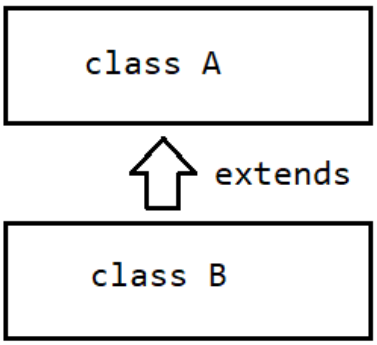
Types of inheritance in java

- Single-level inheritance
- Multi-level Inheritance
- Hierarchical Inheritance
- Multiple Inheritance
- Hybrid Inheritance

Single-Level Inheritance

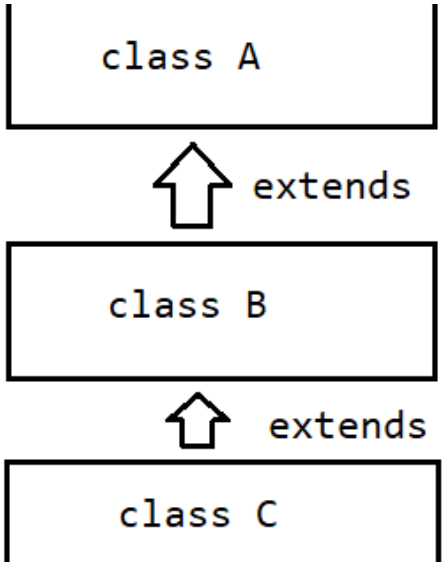
As the name suggests, this type of inheritance occurs for only a single class. Only one class is derived from the parent class. In this type of inheritance, the properties are derived from a single parent class and not more than that. As the properties are derived from only a single base class the reusability of a code is facilitated along with the addition of new features.

The flow diagram of a single inheritance is shown below:



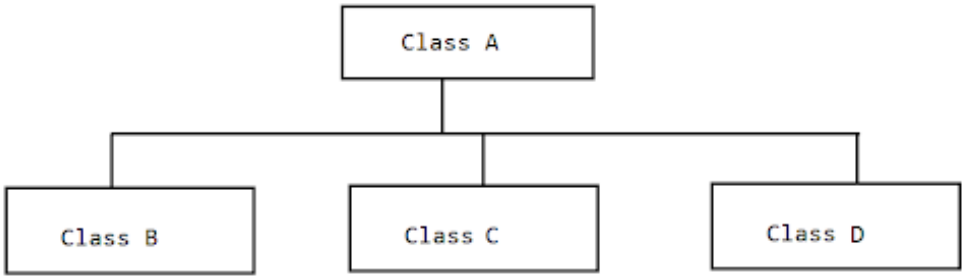
MultiLevel Inheritance

- The multi-level inheritance includes the involvement of at least two or more than two classes.
- One class inherits the features from a parent class and the newly created sub-class becomes the base class for another new class.
- As the name suggests, in the multi-level inheritance the involvement of multiple base classes is there.
- In the multilevel inheritance in java, the inherited features are also from the multiple base classes as the newly derived class from the parent class becomes the base class for another newly derived class.



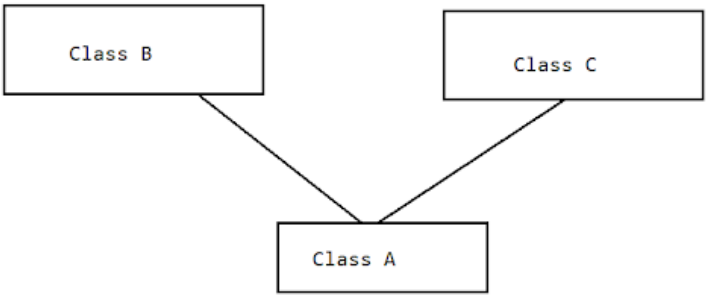
Hierarchical Inheritance

- The type of inheritance where many subclasses inherit from one single class is known as Hierarchical Inheritance.
- Hierarchical Inheritance a combination of more than one type of inheritance.
- It is different from the multilevel inheritance, as the multiple classes are being derived from one superclass.
- These newly derived classes inherit the features, methods, etc, from this one superclass.
- This process facilitates the reusability of a code and dynamic polymorphism (method overriding).



Multiple inheritance

- Multiple inheritance is a type of inheritance where a subclass can inherit features from more than one parent class.
- Multiple inheritances should not be confused with multi-level inheritance, in multiple inheritances the newly derived class can have more than one superclass.
- And this newly derived class can inherit the features from these superclasses it has inherited from, so there are no restrictions.
- In java, multiple inheritance can be achieved through interfaces.



Hybrid Inheritance

- Hybrid inheritance is a combination of more than two types of inheritances, single and multiple.
- It can be achieved through interfaces only as multiple inheritance is not supported by Java.
- It is basically the combination of simple, multiple, hierarchical inheritances.

Note

- All the common methods which every java class needs is a part of the Object class, because the Object class is the parent class for all the inbuilt classes in java.
- For all Exception and Error commonly required methods is a parent of "Throwable",hence Throwable is parent class for all "Exception and Error".

Types of Methods in Inheritance

1. Inherited
2. Overridden
3. Specialized

Inherited method

- The method which would come from parent to child due to inheritance is called inherited method.

Example

```
class Parent{
        public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
          public void methodTwo(){System.out.println("methodTwo from child");}
}

public class TestApp{
       public static void main(String... args){
                Parent p=new Parent();
               p.methodOne();

                Child c=new Child();
               c.methodOne();//inherited method
                c.methodTwo();//Specialized method

                Parent p1=new Child();
                p1.methodOne();
                p1.methodTwo();//CE: can't find the symbol methodTwo in Parent
       }
    }
```

Overridden Method

The method which is taken from Parent and changes the implementation as per the needs of the requirement in the class is called the “overridden method”.

Example

```
class Parent{
    public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
        @Override
        public void methodOne(){System.out.println("methodOne from child");}
 }
 public class TestApp{
        public static void main(String... args){
              Parent p=new Parent();
               p.methodOne();//methodOne from parent

             Child c=new Child();
              c.methodOne();//methodOne from child
        }
   }
```

Inherited method

- The method which would come from parent to child due to inheritance is called inherited method.

Example

```
class Parent{
        public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
          public void methodTwo(){System.out.println("methodTwo from child");}
}

public class TestApp{
       public static void main(String... args){
                Parent p=new Parent();
               p.methodOne();

                Child c=new Child();
               c.methodOne();//inherited method
                c.methodTwo();//Specialized method

                Parent p1=new Child();
                p1.methodOne();
                p1.methodTwo();//CE: can't find the symbol methodTwo in Parent
       }
    }
```

Overridden Method

The method which is taken from Parent and changes the implementation as per the needs of the requirement in the class is called the “overridden method”.

Example

```
class Parent{
    public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
        @Override
        public void methodOne(){System.out.println("methodOne from child");}
 }
 public class TestApp{
        public static void main(String... args){
              Parent p=new Parent();
               p.methodOne();//methodOne from parent

             Child c=new Child();
              c.methodOne();//methodOne from child
        }
   }
```

Specialized Method

The methods which are specific to the particular class are called “Specialised method”.

Example

```
class Parent{
        public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
       public void methodOne(){System.out.println("methodOne from child");}
       public void methodTwo(){System.out.println("methodTwo from child");}
}
public class TestApp{
       public static void main(String... args){
                Parent p=new Parent();
                 p.methodOne();//methodOne from parent

                 Child c=new Child();
                   c.methodOne();//methodOne from child
                   c.methodTwo();//methodOne from child

                 Parent p1=new Child();
                  p1.methodOne();
                  p1.methodTwo();//CE: can't find the symbol methodTwo in Parent
       }
}
```

Rules to Override a method

- Whatever the Parent has by default available to the Child through inheritance, if the Child is not satisfied with Parent class method implementation then Child is allowed to redefine that Parent class method in Child class in its own way this process is called overriding.
- The Parent class method which is overridden is called the overridden method.
- The Child class method which is overriding is called the overriding method.
- In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism or dynamic polymorphism or late binding.
- The process of overriding method resolution is also known as dynamic method dispatch.
- In overriding method names and arguments must be the same. That is, method signatures must be the same.
- Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.
- According to this Child class method return type need not be same as Parent class method return type and Child types are also allowed.

Constructor Execution in case of inheritance

In case of Constructor the Parent class constructor would be executed followed by Child class constructor with the help of “super()”.
It is basically used to make a call to the parent class constructor.
Internally jvm uses super() to promote constructor chaining at inheritance level.

Example

```
class Parent{
             int x=10;
            {
                   methodOne();
                   System.out.println("Parent first instance block");
            }
            Parent(){
                    System.out.println("Parent class constructor");
            }

            public static void main(String... args){
                   Parent p=new Parent();
                   System.out.println("Parent class main()");
            }
            public void methodOne(){
                   System.out.println(y);
            }
            int y=20;
}
class Child extends Parent{
      int i=100;
      {
              methodTwo();
              System.out.println("Child first instance block");
      }
      Child(){
             System.out.println("Child class constructor");
      }

      public static void main(String... args){
             Child c=new Child();
             System.out.println("Child class main()");
      }
      public void methodTwo(){
              System.out.println(j);
      }
      int j=200;
}
public class Test {
       public static void main(String[] args) {

       }
}
```

Rule

Whenever we create child class Object,the following things take place

a. Identification of instance variables,instance blocks from parent to child.
b. Execution of instance variables assignment,instance block only of parent class.
c. Execution of parent class constructor.
d. Execution of instance variables assignment,instance block only of child class.
e. Execution of child class constructor.

Output

java Parent
0
Parent first instance block
Parent class constructor
Parent class main()

Output

java Child
0
Parent first instance block
Parent class constructor
0
Child first instance block
Child class constructor
Child class main()