1 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.
Important terminology:

- **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.

The keyword used for inheritance is extends.

```
1  class A
2  //parent class OR base class OR super class
3  {
4      m1(){    }
5      m2(){    }
6  }
7  class B extends A
8  //child class OR derived class OR sub class
9  {
10     m3(){   }
11     m4(){   }
12 }
13 class C extends A,B //not allowed
14 {
15     m5(){   }
16     m6(){   }
17 }
```

Inheritance.txt

```
//code Reusability

class A contains m1 m2 methods

class B contains m1,m2,m3,m4 methods

java not supports multiple inheritance
 so class C is wrong
```

root class of all java classes is object class Object class is present in java.lang package. Every class in Java is directly or indirectly derived from the Object class. If a Class does not extend any other class then it is direct child class of Object and if extends other class then it is an indirectly derived. Therefore the Object class methods are available to all Java classes. Hence Object class acts as a root of inheritance hierarchy in any Java Program.

```java
class Parent
//parent or –base or – super
{
    public void m1()
    {
        System.out.println("parent class m1 method");
    }
}
class Child extends Parent
{
    //sub or – child or – derived
    public void m2()
    {
        System.out.println("child class m2 method");
    }
}
class Inhtest1
{
    public static void main(String[] args)
    {
        //Assigning parent object to parent reference
        Parent p1=new Parent();
        p1.m1();
        //p1.m2();
        //Assigning child object to child reference
        Child c1=new Child();
        c1.m1();
        c1.m2();
        //Assigning child object to parent  reference
        Parent p2=new Child();
        p2.m1();
        //p2.m2();
        //Child c2=new Parent();
        //Parent cannot be convert to Child
        //c2.m1();
        //c2.m2();

    }
}
```

Inhtest1.java

//inheriting child class methods

D:\phani>javac Inhtest1.java

D:\phani>java Inhtest1

by using parent class reference we can call parent method

for parent class reference parent object

parent class m1 method

for child class reference child object

parent class m1 method

child class m2 method

for parent reference to child object

parent class m1 method

2 Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.
In java programming, multiple and hybrid inheritance is supported through interface only.

3 Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly"means many and "morphs"means forms. So polymorphism means many forms.
There are two types of polymorphism in Java:
1) **compile-time polymorphism or static binding or early binding** We can perform polymorphism in java by **method overloading**
2)**Runtime polymorphism or dynamic binding or late binding.** and method overriding We can perform polymorphing by using **method overriding**.
If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

3.1 static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

1. method overloading
2. constructor overloading
3. operator overloading

Method Overloading

Method overloading means providing two separate methods in a class with the same name but different arguments, while the method return type may or may not be different, which allows us to reuse the same method name.
And this becomes very handy for the consumer of our class. They can pass different types of parameters to the same method (in their eyes, but they are actually different) and get the response according to the input. For example, we might have a System.out.println() method that accepts all primitive object types and prints them, but in reality, there several PrintStream classes

```java
class Polytest1
{   //method overloading
    void m1(int a)
    {
        System.out.println("int m1 method");
    }
    void m1(int a,int b)
    {
        System.out.println("int int m1 method");
    }
    void m1(char ch)
    {
        System.out.println("char m1 method");
    }
    public static void main(String[] args)
    {
        Polytest1 pt1=new Polytest1();
        pt1.m1(10);
        pt1.m1(11,22);
        pt1.m1('a');
    }
}
```

Polytest1.java

```
//method overloading
D:\phani>javac Polytest1.java

D:\phani>java Polytest1
int m1 method
int int m1 method
char m1 method

This THREE methos are called overloaded
```

Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

```
1  class Polytest2
2  {   //constructor overloading
3      Polytest2(int a)
4      {
5          System.out.println("int arg constructor");
6      }
7      Polytest2(int a,int b)
8      {
9          System.out.println("int int arg constructor
               ");
10     }
11     Polytest2(char ch)
12     {
13         System.out.println("char arg constructor");
14     }
15     public static void main(String[] args)
16     {
17         new Polytest2(11);//named approach
18         new Polytest2(888,999); //nameless approach
19         new Polytest2('b');
20         System.out.println(10+20+"bvrit"+"nsp"
               +10+20);
21     }
22 }
```

Polytest2.java

```
//constructor overloading
D:\phani>javac Polytest2.java

D:\phani>java Polytest2
int arg constructor
int int arg constructor
char arg constructor
```

Operator Overloading

Unlike C++, Java doesn't allow user defined overloaded operators. Internally Java overloads operators, for example + is overloaded for concatenation.

```
class Test
{
  public static void main(String[] args)
 {
  System.out.println(10+20+"bvrit"+"nsp"+10+20);
 }
}
output:-
30bvritnsp1020
```

3.2 Dynamic binding

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
In this process, an overridden method is called through the reference variable of a superclass. The

determination of the method to be called is based on the object being referred to by the reference variable.

Overriding in Java

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

Method overriding is one of the way by which java achieve Run Time Polymorphism.The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

```
1  class Parent
2  {
3      void m1()//overridden method
4      {
5          System.out.println("parent class m1 method");
6      }
7  }
8  class Child extends Parent
9  {
10     void m1()//overriding method
11     {
12         System.out.println("child class m1 method");
13     }
14 }
15 //Method Overriding
16 class Polytest3
17 {
18     public static void main(String[] args)
19     {
20         Child c1=new Child();
21         c1.m1();
22     }
23 }
```

Polytest3.java

```
//Method Overriding
D:\phani>javac Polytest3.java

D:\phani>java Polytest3
child class m1 method
```

Advantage of method overriding

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent

3.3 Rules for method overriding

1. **method signatures must be same**

```
class Parent
{
    void m1(int a)//overridden method
    {
        System.out.println("parent class m1 method");
    }
}
class Child extends Parent
{
    void m1(int x)//overriding method
    {
        System.out.println("child class m1 method");
    }
}
//Method Overriding
class Polytest4
{
    public static void main(String[] args)
    {
        Child c1=new Child();
        c1.m1(10);
    }
}
```

Polytest4.java

```
//method overriding
D:\phani>javac Polytest4.java

D:\phani>java Polytest4
child class m1 method

parent class
overridden methos signature m1(int
child class
overriding method signature m1(int
both must be same
```

2. **return types must be same at primitive level**

```
class Parent
{
    int m1()//overridden method
    {
        System.out.println("parent class m1 method");
        return 10;
    }
}
class Child extends Parent
{
    int m1()//overriding method
    {
        System.out.println("child class m1 method");
        return 10;
    }
}
class Polytest5
{
    public static void main(String[] args)
    {
        Child c1=new Child();
        c1.m1();
    }
```

```
//check return types
D:\phani>javac Polytest5.java

D:\phani>java Polytest5
child class m1 method.

parent class
overridden methos signature m1()
 retun type int
child class
overriding method signature m1()
 retun type int
both must be  same retun type

otherwise it  is an error
```

```
24 }
```

Polytest5.java

3. **retun type changing is also possible** (covariant retun type intorduced in 1.5 version)

```
1  class Animal
2  {    void eat()
3       {System.out.println("Animal class eat method");}
4  }
5  class Dog extends Animal
6  {    void eat()
7       {System.out.println("Dog class eat method");}
8  }
9  class Parent
10 {
11      Animal m1()//overridden method
12      {
13          System.out.println("parent class m1 method");
14          return new Animal();
15      }
16 }
17 class Child extends Parent
18 {
19      Dog m1()//overriding method
20      {
21          System.out.println("child class m1 method");
22          return new Dog();
23      }
24 }
25 class Polytest6
26 {
27      public static void main(String[] args)
28      {
29          Child c1=new Child();
30          c1.m1();
31      }
32 }
```

Polytest6.java

```
//check return types
(covariant return type)
D:\phani>javac Polytest6.java

D:\phani>java Polytest6
child class m1 method

parent class  retun type
must be parent type

child class retun type
 must be sub type
```

4. **Final methods can not be overridden** : If we don't want a method to be overridden, we declare it as final. Please see Using final with Inheritance

```
1  class Parent
2  {
3      // Can't be overridden
4      final void show() {  }
5  }
6  class Child extends Parent
7  {
8      // This would produce error
9      void show() {  }
10 }
```

```
//overriding fianl method
D:\phani>javac Polytest7.java
Polytest7.java:9: error: show()
in Child cannot override show() in Parent
    void show() {  }
         ^
  overridden method is final
Polytest7.java:16:
error: cannot find symbol
                c1.m1();
                  ^
```

```
class Polytest7
{
    public static void main(String[] args)
    {
        Child c1=new Child();
        c1.m1();
    }
}
```

Polytest7.java

```
  symbol:   method m1()
  location: variable c1 of type Child
2 errors
```

final class variable are not final
final class methods are final
if a class final we cannot extend that class

5. **Static methods can not be overridden(Method Overriding vs Method Hiding)** : When you defines a static method with same signature as a static method in base class, it is known as method hiding.
The following table summarizes what happens when you define a method with the same signature as a method in a super-class.

```
class Parent
{
    // Static method in base class which will be hidden in
        subclass
    static void m1()
    {
        System.out.println("From parent static m1()");
    }

    // Non-static method which will be overridden in derived
        class
    void m2()
    {
        System.out.println("From parent non-static(instance) m2
            ()");
    }
}

class Child extends Parent
{
    // This method hides m1() in Parent
    static void m1()
    {
        System.out.println("From child static m1()");
    }

    // This method overrides m2() in Parent
    @Override
    public void m2()
    {
        System.out.println("From child non-static(instance) m2()
            ");
    }

}
```

```
//static methods

D:\phani>javac Polytest8.j

D:\phani>java Polytest8
From parent static m1()
From child non-static(inst
```

```
// Driver class
class Polytest8
{
    public static void main(String[] args)
    {
        Parent obj1 = new Child();

        // As per overriding rules this should call to class
            Child static
        // overridden method. Since static method can not be
            overridden, it
        // calls Parent's m1()
        obj1.m1();

        // Here overriding works and Child's m2() is called
        obj1.m2();
    }
}
```

Polytest8.java

	SUPERCLASS INSTANCE METHOD	SUPERCLASS STATIC METHOD
SUBCLASS INSTANCE METHOD	Overrides	Generates a compile-time error
SUBCLASS STATIC METHOD	Generates a compile-time error	Hides

6. **Private methods can not be overridden** : Private methods cannot be overridden as they are bonded during compile time. Therefore we can't even override private methods in a subclass

```
class Parent
{
    private void m1()
    {
        System.out.println("parent class m1()");
    }
}

class Child extends Parent
{
     void m1()
    {
        System.out.println("child class m1()");
    }
}
// Driver class
class Polytest8
{
    public static void main(String[] args)
    {
        Parent p1 = new Parent();
        p1.m1();
    }
}
```

Polytest9.java

```
//private methods
D:\phani>javac Polytest9.java
Polytest9.java:22: error:
m1() has private access in Parent
                p1.m1();
                  ^
1 error
```

7. **Invoking overridden method from sub-class** : We can call parent class method in overriding method using super keyword.

```java
class Parent
{
    void show()
    {
        System.out.println("Parent's show()");
    }
}

// Inherited class
class Child extends Parent
{
    // This method overrides show() of Parent
    @Override
    void show()
    {
        super.show();
        System.out.println("Child's show()");
    }
}
// Driver class
class Polytest10
{
    public static void main(String[] args)
    {
        Parent obj = new Child();
        obj.show();
    }
}
```

```
//method can be called from sub-class
D:\phani>javac Polytest10.java

D:\phani>java Polytest10
Parent's show()
Child's show()
```

Polytest10.java

8. **Overriding and constructor** : We can not override constructor as parent and child class can never have constructor with same name(Constructor name must always be same as Class name).

9. **Overriding and synchronized/stricfp method** : The presence of synchronized/stricfp modifier with method have no effect on the rules of overriding, i.e. it's possible that a synchronized/stricfp method can override a non synchronized/stricfp one and vice-versa.

10. **Overriding and Exception-Handling** : Below are two rules to note when overriding methods related to exception-handling.

    (a) If the super-class overridden method does not throws an exception, subclass overriding method can only throws the unchecked exception, throwing checked exception will lead to compile-time error.

    (b) If the super-class overridden method does throws an exception, subclass overriding method can only throw same, subclass exception. Throwing parent exception in Exception hierarchy will lead to compile time error.Also there is no issue if subclass overridden method is not throwing any exception.