**Difference between Inheritance and aggregation**

Aggregation in Java is a relationship between two classes that is best described as a "has-a" and "whole/part" relationship. The aggregate class contains a reference to another class and is said to have ownership of that class. Each class referenced is considered to be part-of the aggregate class.

In this example, we have created the reference of Operation class in the Circle class.

class Operation{

int square(int n){

return n\*n;

}

}

class Circle{

Operation op;//aggregation

double pi=3.14;

double area(int radius){

op=new Operation();

int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).

return pi\*rsquare;

}

public static void main(String args[]){

Circle c=new Circle();

double result=c.area(5);

System.out.println(result);

}

}

Output:78.5

OR,



Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Inheritance is nothing but is-a relationship.

**extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax:

class Super {

.....

.....

}

class Sub extends Super {

.....

.....

}

Example

class A

{

void msg1()

{

System.out.println(“method of super class”);

}

}

class B extends A

{

void msg1()

{

System.out.println(“ overriding method of super class”);

}

void msg2()

{

System.out.println(“Another method of sub class”);

}

}

class Demo

{

Public static void main(String args[])

{

A a=new A();

B b=new B();

a.msg1();

b.msg1();

b.msg2();

}

}

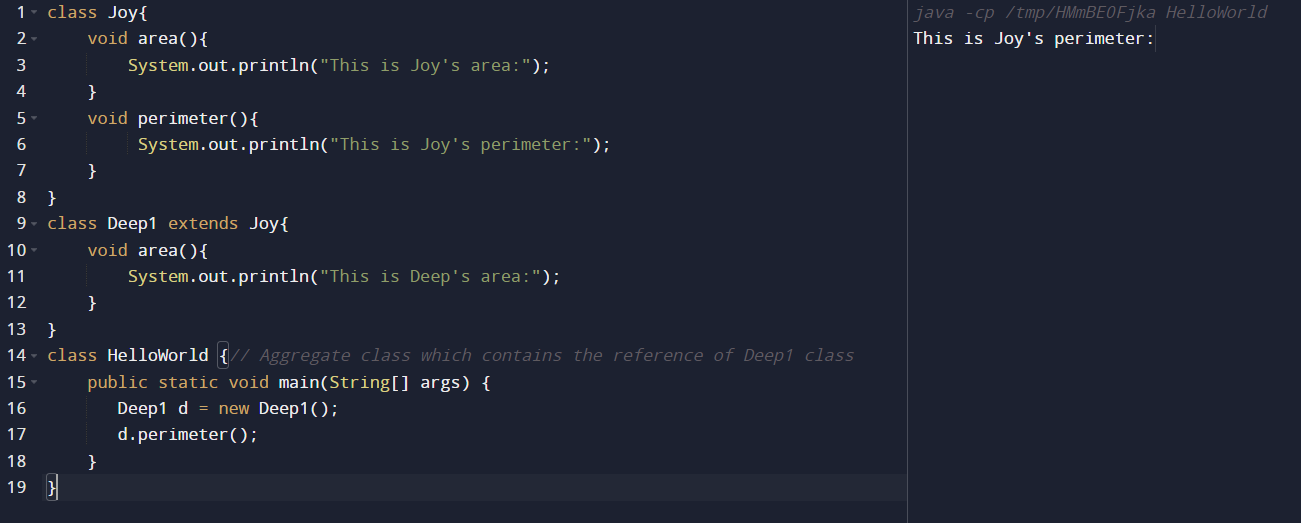
Output:

method of super class

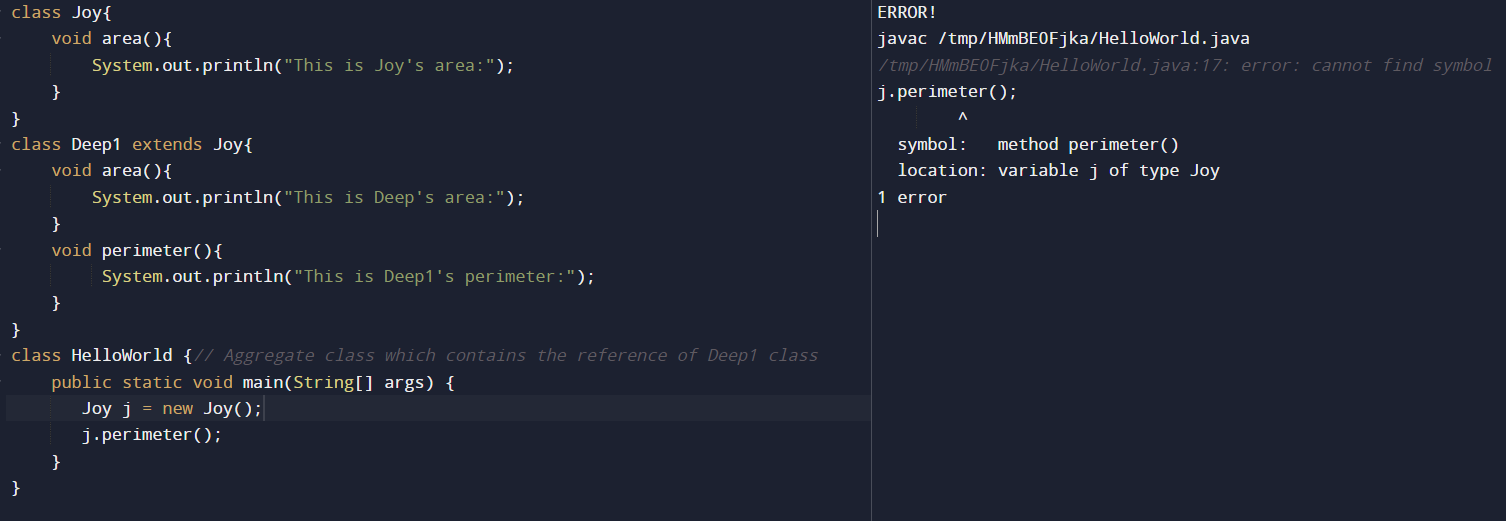
overriding method of super class

Another method of sub class

**\*\*\*\*\*If we invoke any method which is not present in sub but present in super class, we can able to invoke that one by using the reference variable of sub .**

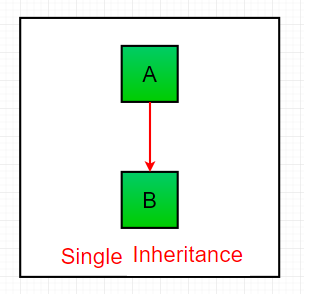
****

**\*\*\*\*\*If we invoke any method which is not present in super class but present in sub class, we can't able to invoke that one by using the reference variable of super class.**

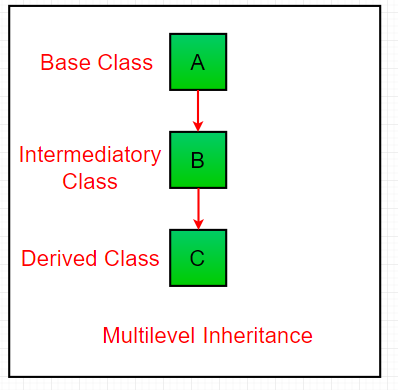
****

**Types of inheritance**

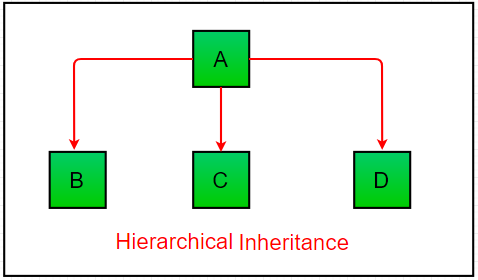
**Single Inheritance :**In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance1.png)

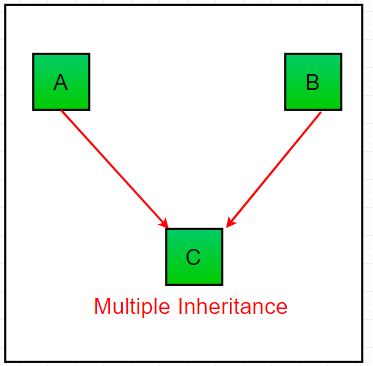
**Multilevel Inheritance :**In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the[grandparent’s members](https://www.geeksforgeeks.org/g-fact-91/).

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance3.png)

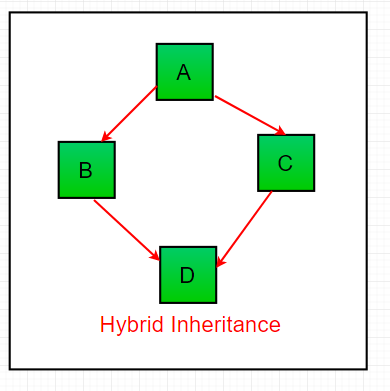
**Hierarchical Inheritance :**In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance4.png)

[**Multiple Inheritance**](https://www.geeksforgeeks.org/java-and-multiple-inheritance/)**(Through Interfaces) :**In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. In java, we can achieve multiple inheritance only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/). In image below, Class C is derived from interface A and B.

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance2-1.png)

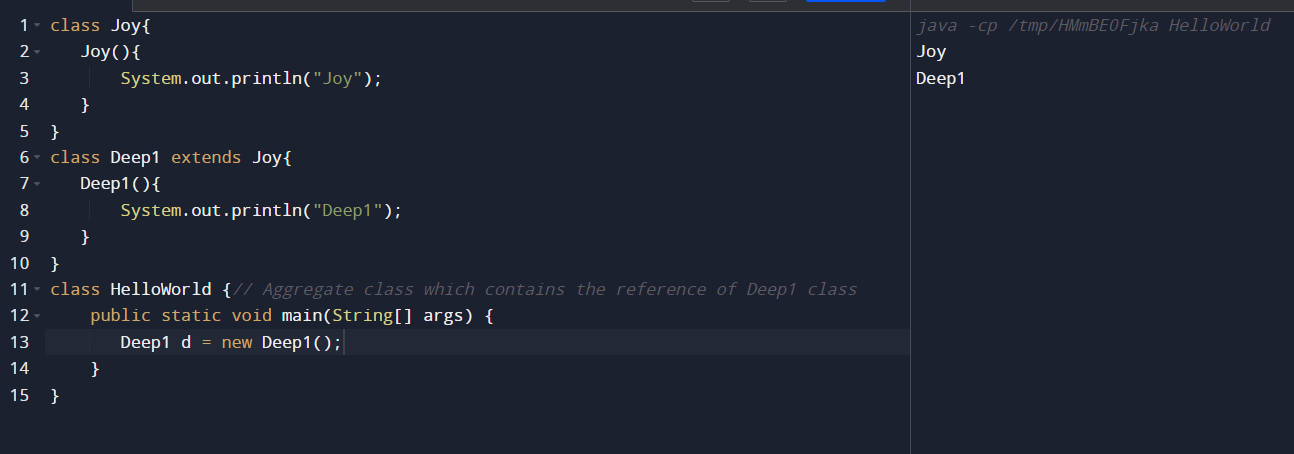
**Hybrid Inheritance(Through Interfaces or Class) :**It is a mix of two or more of the above types of inheritance. Since java doesn’t support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/).

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance-1.png)

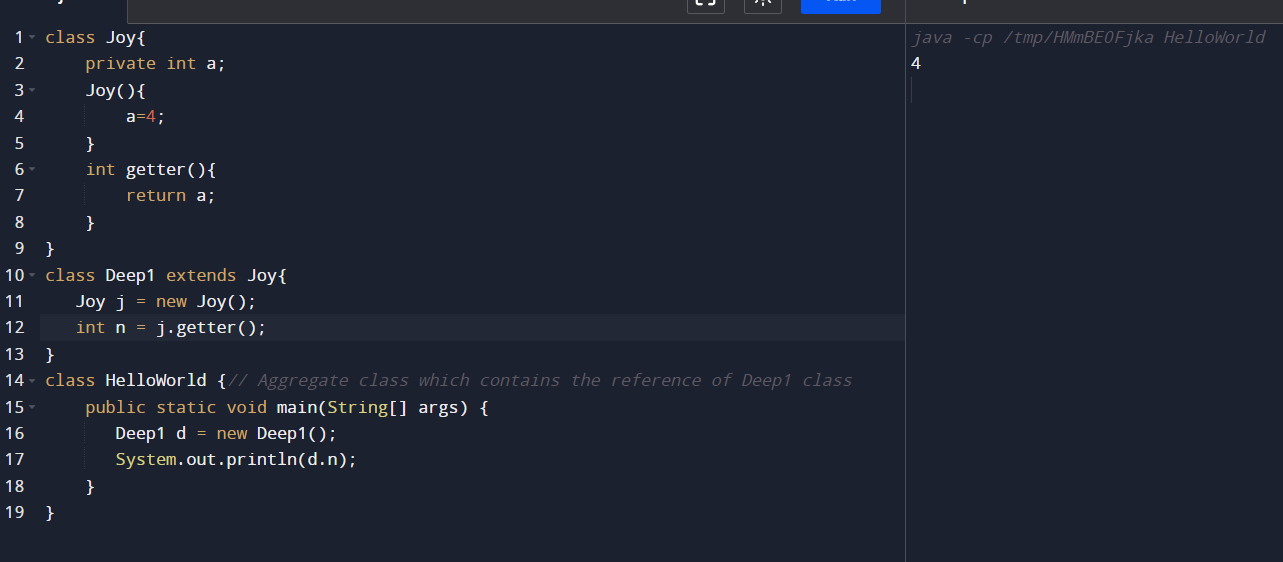
**Important facts about inheritance in Java**

* **Default superclass**: Except [Object](https://www.geeksforgeeks.org/object-class-in-java/) class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of [Object](https://www.geeksforgeeks.org/object-class-in-java/) class.
* **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only **one** superclass. This is because Java does not support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. Although with interfaces, multiple inheritance is supported by java.
* **Inheriting Constructors:**A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

Suppose the sub and super both classes have constructor and using the reference of sub class if we create object of sub class then the constructor of super class will be invoked first then the sub class.



* **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods(like getters and setters) for accessing its private fields, these can also be used by the subclass.

****

**Method Overriding in Java**

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

1. //Java Program to illustrate the use of Java Method Overriding
2. //Creating a parent class.
3. **class** Vehicle{
4. //defining a method
5. **void** run1(){System.out.println("Vehicle is running");}
6. **void** run2(){System.out.println(“running");}
7. }
8. //Creating a child class
9. **class** Bike2 **extends** Vehicle{
10. //defining the same method as in the parent class
11. **void** run1(){System.out.println("Bike is running safely");}
13. **public** **static** **void** main(String args[]){
14. Bike2 obj = **new** Bike2();//creating object
15. obj.run1();
16. obj.run2();//calling method
17. }
18. }

Output:

Bike is running safely

running

**“Super can refer to sub but vice versa is not possible”-** by using reference variable of super class we can able to create object of sub class but by using reference of sub class we can’t able to create object of super. After that by using that reference if we invoke any method which is present in both then version of sub will invoke but in case of variable version of super will invoke.

class A

{

int a=5;

void msg()

{

System.ou.println(“method of super class”);

}

}

class B extends A

{

int a=10;

void msg()

{

System.ou.println(“method of sub class”);

}

}

class Demo

{

public static void main(String args[])

{

A x=new B();

x.msg();

System.out.println(x.a);

}

}

Output:

method of sub class

5

**Super Keyword in Java**

The**super** keyword in java is a reference variable that is used to refer parent class objects.  The keyword “super” came into the picture with the concept of Inheritance. It is majorly used in the following contexts:

**1. Use of super with variables:**This scenario occurs when a derived class and base class has same data members. In that case there is a possibility of ambiguity for the JVM. We can understand it more clearly using this code

|  |
| --- |
| /\* Base class vehicle \*/  class Vehicle  {      int maxSpeed = 120;  }    /\* sub class Car extending vehicle \*/  class Car extends Vehicle  {      int maxSpeed = 180;        void display()      {          /\* print maxSpeed of base class (vehicle) \*/          System.out.println("Maximum Speed: " + super.maxSpeed);      }  }    /\* Driver program to test \*/  class Test  {      public static void main(String[] args)      {          Car small = new Car();          small.display();      }  } |

Output:

Maximum Speed: 120

In the above example, both base class and subclass have a member maxSpeed. We could access maxSpeed of base class in subclass using super keyword.

**2. Use of super with methods:**This is used when we want to call parent class method. So whenever a parent and child class have same named methods then to resolve ambiguity we use super keyword. This code snippet helps to understand the said usage of super keyword.

|  |
| --- |
| /\* Base class Person \*/  class Person  {      void message()      {          System.out.println("This is person class");      }  }    /\* Subclass Student \*/  class Student extends Person  {      void message()      {          System.out.println("This is student class");      }        // Note that display() is only in Student class      void display()      {          // will invoke or call current class message() method          message();            // will invoke or call parent class message() method          super.message();      }  }    /\* Driver program to test \*/  class Test  {      public static void main(String args[])      {          Student s = new Student();            // calling display() of Student          s.display();      }  } |

Output:

This is student class

This is person class

In the above example, we have seen that if we only call method message() then, the current class message() is invoked but with the use of super keyword, message() of superclass could also be invoked.

**3**. **Use of super with constructors:**super keyword can also be used to access the parent class constructor. One more important thing is that, ‘’super’ can call both parametric as well as non parametric constructors depending upon the situation. Following is the code snippet to explain the above concept:

|  |
| --- |
| /\* superclass Person \*/  class Person  {      Person()      {          System.out.println("Person class Constructor");      }  }    /\* subclass Student extending the Person class \*/  class Student extends Person  {      Student()      {          // invoke or call parent class constructor          super();            System.out.println("Student class Constructor");      }  }    /\* Driver program to test\*/  class Test  {      public static void main(String[] args)      {          Student s = new Student();      }  } |

Output:

Person class Constructor

Student class Constructor

In the above example we have called the superclass constructor using keyword ‘super’ via subclass constructor.

# final keyword in java

final keyword is used in different contexts. First of all, final is a [non-access modifier](https://www.geeksforgeeks.org/access-and-non-access-modifiers-in-java/) applicable **only to a variable, a method or a class**.Following are different contexts where final is used.

final variable::::::::::::::::::: to create constant variable

final method::::::::::::::::::: to prevent method overriding

final class:::::::::::::::::::::::: to prevent inheritance