can a Construtor is Inherited ?

Answer: No, a Constructor cant be inherited.

Whenver a child class extends parent class, the sub class inherits all its all the members (fields, methods, and nested classes) from its superclass but it does not inherit constructor of super class because of following reasons:

1)Constructor should have same name as class name in java. So if constructors were inherited in child class it should have same name which is defined in Parent class which is against the rule of constructor that constructor and class name should be same.

Class Parent {

    public Parent()

    {

    }

    public void print()

    {

    }

}

public class Child extends Parent {

    public Parent()

    {

    }

    public void print()

    {

    }

    public static void main(String[] args)

    {

        Child c1 = new Child(); // allowed

        Child c2 = new Parent(); // not allowed

    }

}

If we define Parent class constructor inside Child class it will give compile time error for return type and consider it a method. But for print method it does not give any compile time error and consider it a overriding method.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[**Why constructors cannot be final**](https://stackoverflow.com/questions/9477476/why-constructors-cannot-be-final)**, static and abstract?**

**When you set a method as final it means**: "I don't want any class override it." But according to the Java Language Specification:

[JLS 8.8](https://docs.oracle.com/javase/specs/jls/se11/html/jls-8.html#jls-8.8) - "Constructor declarations are not members. They are never inherited and therefore are not subject to hiding or overriding."

**When you set a method as abstract it means**: "This method doesn't have a body and it should be implemented in a child class." But the constructor is called implicitly when the new keyword is used so it can't lack a body.

**When you set a method as static it means**: Everything that is marked static belongs to the class only, for example **static method cannot be inherited in the sub class because they belong to the class in which they have been declared**.

Important Points on constructor:

**1)We can overload constructors but we can’t override constructors**

**2) We can declare constructor with these modifiers : public private, protected and default**

**3)Every class in java including abstract class can contain Constructor but interface cannot contain constructor.**

**4)** **When we create the object of sub class, the new keyword invokes the**[**constructor**](https://beginnersbook.com/2013/03/constructors-in-java/)**of child class, which implicitly invokes the constructor of parent class. So the order to execution when we create the object of child class is: parent class constructor is executed first and then the child class constructor is executed. It happens because compiler itself adds super()(this invokes the no-arg constructor of parent class) as the first statement in the constructor of child class.**

**See for more ino wit example:** <https://beginnersbook.com/2014/07/super-keyword-in-java-with-example/>

**5)**

**Use of Private constructor in Java:**

The **use of private constructor** is to serve singleton classes. A singleton class is one which limits the number of objects creation to one. Using private constructor we can ensure that no more than one object can be created at a time. By providing a private constructor you prevent class instances from being created in any place other than this very class. We will see in the below example how to use private constructor for limiting the number of objects for a singleton class.

1) Can you create an object without using new operator in Java?

Yes, We can create an object without using new operator. There are some other ways to create objects other than using new operator. But, 95% of object creation in java is done through new operator only.

**a) Using newInstance() Method**

|  |  |
| --- | --- |
| 1  2  3 | Class c = Class.forName("packageName.MyClass");    MyClass object = (MyClass) c.newInstance(); |

**b) Using clone() method.**

|  |  |
| --- | --- |
| 1  2  3 | MyClass object1 = new MyClass();    MyClass object2 = object1.clone(); |

**c) Using object deserialization**

|  |  |
| --- | --- |
| 1  2  3 | ObjectInputStream inStream = new ObjectInputStream(anInputStream );    MyClass object = (MyClass) inStream.readObject(); |

**d) Creating string and array objects :**

|  |  |
| --- | --- |
| 1  2  3 | String s = "string object";    int[] a = {1, 2, 3, 4}; |

What is Constructor Chaining?

Constructor Chaining is a technique of calling another constructor from one constructor. **this()** is used to call same class constructor where as **super()** is used to call super class constructor.

**package** constructors;

**public** **class** SuperClassConstructorChaning {

**public** SuperClassConstructorChaning(**int** i)

{

System.***out***.println("Super Class Constructor");

}

}

**package** constructors;

**public** **class** SubClassConstructorChaning **extends** SuperClassConstructorChaning{

**public** SubClassConstructorChaning()

{

**this**(10);//Calling same class constructor

System.***out***.println("Sub Class Constructor");

}

**public** SubClassConstructorChaning(**int** i)

{

**super**(i); //Calling super class constructor

}

}

**package** constructors;

**public** **class** MainClassConstructorChaining {

**public** **static** **void** main(String[] args) {

SubClassConstructorChaning sc=**new** SubClassConstructorChaning();

}

}

3) Can we call sub class constructor from super class constructor?

No. There is no way in java to call sub class constructor from a super class constructor.

4) What happens if you keep return type for a constructor?

It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class MyClass  {      int MyClass()      {          return 0;    //No Compile time error but just a warning      }  } |

5) What is No-arg constructor?

Constructor without arguments is called no-arg constructor. Default constructor in java is always a no-arg constructor.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class MyClass  {      public MyClass()      {          //No-arg constructor      } |

6) What is the use of private constructor?

Private constructors are used to restrict the instantiation of a class. When a class needs to prevent other classes from creating it’s objects then private constructors are suitable for that. Objects to the class which has only private constructors can be created within the class. A very good use of private constructor is in singleton pattern. This ensures only one instance of a class exist at any point of time. Here is an example of singleton pattern using private constructor.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | class MyClass  {      private static MyClass object = null;        private MyClass()      {          //private constructor      }        public MyClass getObject()      {          if(object == null)          {              object = new MyClass();   //Creating object using private constructor          }            return object;      }  } |

7) Can we use this() and super() in a method?

No, We can’t use this() and super() in a method.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | class SuperClass  {      public SuperClass()      {          System.out.println("Super Class Constructor");      }  }    class SubClass extends SuperClass  {      public SubClass()      {          System.out.println("Sub Class Constructor");      }        void method()      {          this();     //Compile time error            super();    //Compile time error      }  } |