|  |
| --- |
| **2) Interpreter:** Read bytecode stream then execute the instructions. |
| **3) Just-In-Time(JIT) compiler:** It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU. Overloadding And Overrrding  |  | | --- | | **Overloading:** There are two ways to overload the method in java |  1. By changing number of arguments 2. By changing the data type 3. Method Overloading is not possible by changing the return type of the method.   Q1. Why Method Overloaing is not possible by changing the return type of method?  **Due to ambiguity of the method.**  Q2. Can we overload main() method?  **Yes we can overload main method.** Constructor in Java : There are two types of constructor  1.Default Constructor  2.Parametrized constructor  **Q) What is the purpose of default constructor?**  Default constructor provides the default values to the object like 0, null etc. depending on the type.  **Constructor Overloading in Java**   |  | | --- | | Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. |  Difference between constructor and method in java There are many differences between constructors and methods. They are given below.   |  |  | | --- | --- | | **Java Constructor** | **Java Method** | | Constructor is used to initialize the state of an object. | Method is used to expose behaviour of an object. | | Constructor must not have return type. | Method must have return type. | | Constructor is invoked implicitly. | Method is invoked explicitly. | | The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. | | Constructor name must be same as the class name. | Method name may or may not be same as class name. |  Static class,Method And Variable in Java: **1.Can a class be static in Java ?**  The answer is YES, we can have static class in java. In java, we have [static instance variables](http://www.geeksforgeeks.org/static-keyword-in-java/) as well as [static methods](http://www.geeksforgeeks.org/static-keyword-in-java/) and also [static block](http://www.geeksforgeeks.org/g-fact-79/). Classes can also be made static in Java. Java allows us to define a class within another class. Such a class is called a nested class. The class which enclosed nested class is known as Outer class. In java, we can’t make Top level class static. **Only nested classes can be static**.  **2. What are the differences between static and non-static nested classes?**  Following are major differences between static nested class and non-static nested class. Non-static nested class is also called Inner Class.  **1)** Nested static class doesn’t need reference of Outer class, but Non-static nested class or Inner class requires Outer class reference.  **2)** Inner class(or non-static nested class) can access both static and non-static members of Outer class. A static class cannot access non-static members of the Outer class. It can access only static members of Outer class.  **3)**An instance of Inner class cannot be created without an instance of outer class and an Inner class can reference data and methods defined in Outer class in which it nests, so we don’t need to pass reference of an object to the constructor of the Inner class. For this reason Inner classes can make program simple and concise.   |  | | --- | | /\* Java program to demonstrate how to implement static and non-stati classes in a java program. \*/  class OuterClass {     private static String msg = "GeeksForGeeks";     // Static nested class     public static class NestedStaticClass{       // Only static members of Outer class is directly accessible in nested static class       public void printMessage() {            // Try making 'message' a non-static variable, there will be compiler error           System.out.println("Message from nested static class: " + msg);         }      }       // non-static nested class - also called Inner class      public class InnerClass{          //Both static and non-static members of Outer class are accessible in this Inner class         public void display(){            System.out.println("Message from non-static nested class: "+ msg);         }      }}  class Main  {      // How to create instance of static and non static nested class?      public static void main(String args[]){        // create instance of nested Static class         OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();        // call non static method of nested static class         printer.printMessage();         // In order to create instance of Inner class we need an Outer class instance. Let us create Outer class instance for creating non-static nested class.         OuterClass outer = new OuterClass();         OuterClass.InnerClass inner  = outer.new InnerClass();          // calling non-static method of Inner class         inner.display();          // we can also combine above steps in one step to create instance of Inner class         OuterClass.InnerClass innerObject = new OuterClass().new InnerClass();          // similarly we can now call Inner class method         innerObject.display();      }  } |   Output:  Message from nested static class: GeeksForGeeks  Message from non-static nested class: GeeksForGeeks  Message from non-static nested class: GeeksForGeeks  **3.Can we overload static methods?**  The answer is ‘Yes’. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program.   |  | | --- | | // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public static void foo(int a) {          System.out.println("Test.foo(int) called ");      }      public static void main(String args[])      {          Test.foo();          Test.foo(10);      }  } |   **4.Can we overload methods that differ only by static keyword?**  We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same). See following Java program for example. This behaviour is same in C++ (See point 2 of [this](http://www.geeksforgeeks.org/function-overloading-in-c/)).   |  | | --- | | // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public void foo() { // Compiler Error: cannot redefine foo()          System.out.println("Test.foo(int) called ");      }  public static void main(String args[]) {          Test.foo();  } } |   **5.Can we Override static methods in java?**  We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is **‘No’.** If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.   |  | | --- | | /\* Java program to show that if static method is redefined by     a derived class, then it is not overriding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // This method hides display() in Base      public static void display() {           System.out.println("Static or class method from Derived");      }        // This method overrides print() in Base      public void print() {           System.out.println("Non-static or Instance method from Derived");     }  }    // Driver class  public class Test {      public static void main(String args[ ])  {         Base obj1 = new Derived();           // As per overriding rules this should call to class Derive's static         // overridden method. Since static method can not be overridden, it         // calls Base's display()         obj1.display();           // Here overriding works and Derive's print() is called         obj1.print();      }  } |   Run on IDE  Output:  Static or class method from Base  Non-static or Instance method from Derived  Following are some important points for method overriding and static methods in Java. **a)** For class (or static) methods, the method according to the type of reference is called, not according to the abject being referred, which means method call is decided at compile time.  **b)** For instance (or non-static) methods, the method is called according to the type of object being referred, not according to the type of reference, which means method calls is decided at run time.  **c)** An instance method cannot override a static method, and a static method cannot hide an instance method. For example, the following program has two compiler errors.   |  | | --- | | /\* Java program to show that if static methods are redefined by     a derived class, then it is not overriding but hidding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // Static is removed here (Causes Compiler Error)      public void display() {          System.out.println("Non-static method from Derived");      }        // Static is added here (Causes Compiler Error)      public static void print() {          System.out.println("Static method from Derived");     }  } |   **d)** In a subclass (or Derived Class), we can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods — they are new methods, unique to the subclass. |
| In this tutorial we will discuss the use of **static keyword in Java**. It can be used along with Class name, Variables, Methods and block.  1. static class 2. static block 3. [**static methods**](http://beginnersbook.com/2013/05/static-vs-non-static-methods/) 4. [**static variables**](http://beginnersbook.com/2013/05/static-variable/) Static Class A Class can be made **static** only if it is a nested Class. The nested static class can be accessed without having an object of outer class.  **Example 1:**  class Example1{  //Static class  static class X{  static String str="Inside Class X";  }  public static void main(String args[])  {  X.str="Inside Class Example1";  System.out.println("String stored in str is- "+ X.str);  }  }  Output:  String stored in str is- Inside Class Example1  **Example 2: Compile time Error!!**  class Example2{  int num;  //Static class  static class X{  static String str="Inside Class X";  num=99;  }  public static void main(String args[])  {  Example2.X obj = new Example2.X();  System.out.println("Value of num="+obj.str);  }  }  Output: Compile time error. [**Static inner class**](http://beginnersbook.com/2013/05/inner-class/) cannot access instance data of outer class. Static Block Static block is mostly used for changing the default values of static variables.This block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.  **Example 1: Single static block**  class Example3{  static int num;  static String mystr;  static{  num = 97;  mystr = "Static keyword in Java";  }  public static void main(String args[])  {  System.out.println("Value of num="+num);  System.out.println("Value of mystr="+mystr);  }  }  Output:  Value of num=97  Value of mystr=Static Keyword in Java  **Example 2: Multiple Static blocks**  class Example4{  static int num;  static String mystr;  //First Static block  static{  System.out.println("Static Block 1");  num = 68;  mystr = "Block1";  }  //Second static block  static{  System.out.println("Static Block 2");  num = 98;  mystr = "Block2";  }  public static void main(String args[])  {  System.out.println("Value of num="+num);  System.out.println("Value of mystr="+mystr);  }  }  **Output:**  Static Block 1  Static Block 2  Value of num=98  Value of mystr=Block2 Static Methods Static Methods can access class variables without using object of the class. It can access non-static methods and non-static variables by using objects. Static methods can be accessed directly in static and non-static methods.  **Example 1: public static void main itself is a static method**  class Example5{  static int i;  static String s;  public static void main(String args[]) //Its a Static Method  {  Example5 obj=new Example5();  //Non Static variables accessed using object obj  System.out.println("i:"+obj.i);  System.out.println("s:"+obj.s);  }  }  **Output:**  i:0  s:null  **Example 2: Static method display()**  class Example6{  static int i;  static String s;  //Static method  static void display()  {  //Its a Static method  Example6 obj1=new Example6();  System.out.println("i:"+obj1.i);  System.out.println("i:"+obj1.i);  }  void funcn()  {  //Static method called in non-static method  display();  }  public static void main(String args[]) //Its a Static Method  {  //Static method called in another static method  display();  }  }  Output:  i:0  i:0 4. Static Variables  * Static variables are also known as Class Variables. * Such variables get default values based on the data type. * Data stored in static variables is common for all the objects( or instances ) of that Class. * Memory allocation for such variables only happens once when the class is loaded in the memory. * These variables can be accessed in any other class using class name. * Unlike **non-static variables**, such variables can be accessed directly in static and non-static methods.   **Example 1: Static variables can be accessed without reference in Static method**  class Example7{  static int var1;  static String var2;  //Its a Static Method  public static void main(String args[])  {  System.out.println("Var1 is:"+Var1);  System.out.println("Var2 is:"+Var2);  }  }  **Output:**  Var1 is:0  Var2 is:null  As you can see in the above example that both the variables are accessed in void main method without any object(reference).  **Example 2: Static variables are common for all instances**  package beginnersbook.com;  class Example8{  static int Var1=77; //Static integer variable  String Var2;//non-static string variable  public static void main(String args[])  {  Example8 ob1 = new Example8();  Example8 ob2 = new Example8();  ob1.Var1=88;  ob1.Var2="I'm Object1";  ob2.Var2="I'm Object2";  System.out.println("ob1 integer:"+ob1.Var1);  System.out.println("ob1 String:"+ob1.Var2);  System.out.println("ob2 integer:"+ob2.Var1);  System.out.println("ob2 STring:"+ob2.Var2);  }  }  Output:  ob1 integer:88  ob1 String:I'm Object1  ob2 integer:88  ob2 String:I'm Object2  In above example String variable is non-static and integer variable is Static. So you can see that String variable value is different for both objects but integer variable value is common for both the instances as all the objects share the same copy of a static variable. |

**Java static constructor – Is it really Possible to have them in Java?**

Have you heard of **static constructor in Java**? I guess yes but the fact is that they are not allowed in Java. A constructor can not be static in Java.  Before I explain the reason let’s have a look at the below piece of code:

public class StaticTest

{

/\* See below - I have marked the constructor as static \*/

public static StaticTest()

{

System.out.println("Static Constructor of the class");

}

public static void main(String args[])

{

/\*Below: I'm trying to create an object of the class

which would intern call the constructor\*/

StaticTest obj = new StaticTest();

}

}

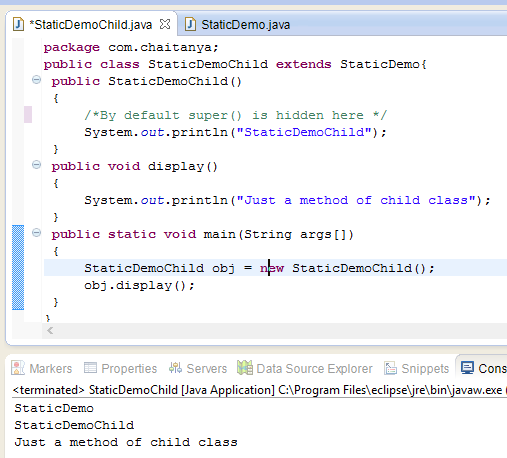
**Output:** You would get the below error message when you try to run the above java code.

**“modifier static not allowed here”**

**Why java doesn’t support static constructor?**

It’s actually pretty simple to understand – 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. Refer**[**static keyword**](http://beginnersbook.com/2013/04/java-static-class-block-methods-variables/)**.**  
Lets back to constructors, Since each constructor is being called by its subclass during creation of the object of its subclass, so if you mark constructor as static the subclass will not be able to access the constructor of its parent class because it is marked static and thus belong to the class only. This will violate the whole purpose of inheritance concept and that is reason why a constructor cannot be static.

Let’s understand this with the help of an example –



public class StaticDemo

{

public StaticDemo()

{

/\*Constructor of this class\*/

System.out.println("StaticDemo");

}

}

public class StaticDemoChild extends StaticDemo

{

public StaticDemoChild()

{

/\*By default this() is hidden here \*/

System.out.println("StaticDemoChild");

}

public void display()

{

System.out.println("Just a method of child class");

}

public static void main(String args[])

{

StaticDemoChild obj = new StaticDemoChild();

obj.display();

}

}

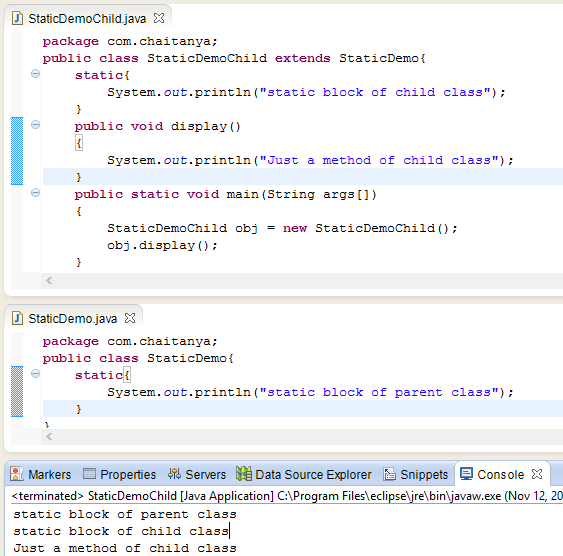
**Output:**  
StaticDemo  
StaticDemoChild  
Just a method of child class

**Did you notice?**We just created the object of child class and as a result it first called the constructor of parent class and then the constructor of it’s own class. It happened because the object creation calls constructor implicitly and every **child class constructor by default has super()** as first statement which calls it’s parent class’s constructor. The statement super() is used to call the parent class(base class) constructor.

Above explanation is the reason why constructor cannot be static – Because if we make them static they cannot be called from child class thus object of child class couldn’t be created.

**Static Constructor Alternative – Static Blocks**

Java has static blocks which can be treated as static constructor. Let’s consider the below program –



public class StaticDemo{

static{

System.out.println("static block of parent class");

}

}

public class StaticDemoChild extends StaticDemo{

static{

System.out.println("static block of child class");

}

public void display()

{

System.out.println("Just a method of child class");

}

public static void main(String args[])

{

StaticDemoChild obj = new StaticDemoChild();

obj.display();

}

}

**Output:**  
static block of parent class  
static block of child class  
Just a method of child class

In the above example we have used static blocks in both the classes which worked perfectly. We cannot use static constructor so it’s a good alternative if we want to perform a static task during object creation.