How many ways we can load the .class file into JVM memory ?
----

* There are so many ways to load the .class file into JVM memory. The following are the common examples

1) By using java command

```
Example :
public class Test
{
}

java Test [We are making a request to load Test.class file into JVM memory]
```

2) By using Constructor that means by creating the Object

```
Example :
public class Demo
{
}

new Demo();  [First Demo class will be loaded and then Object will be created]
```

3) By accessing the static data member of the class.

```
Example
public class Alpha
{
   public static void m1()
   {
   }
}

Alpha.m1(); //Alpha.class will be loaded
```

4) By using Inheritance : **[Child cannot exist without Parent]**

```
class A
{
}
class B extends A
{
}

new B();   //First of A.class file will be loaded then B.class file will be loaded
```

5) By using Reflection API

There is a predefined class called **Class** available in java.lang package, It contains a static factory method called forName(String className) through which we can load the .class file into JVM memory.

```
public static java.lang.Class forName(String className) throws ClassNotFoundException
```

```
//Program :
package com.ravi.class_loading;

class Test
{
    static
    {
        System.out.println("Static Block of Test class");
    }
}
public class DynamicClassLoading
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("com.ravi.class_loading.Test");  //FQN is required
    }
}
```

NOte : FQN (Fully Qualified Name) [Package Name + class name]

*** A static method does not act on instance variable directly ?

Common Topic Discussion :

Multiple Inheritance topic :

```
class A                     class B        It should pass all the Test
{                           {              cases.
   A(int x)                    B()
   {                           {
   }                           }
}                           }
      class C extends A,B
      {
          C()
          {
            super();
          }
      }
```

All the static members (static variable, static block, static method, static nested inner class) are loaded/executed at the time of loading the .class file into JVM Memory.

At class loading phase object is not created because object is created in the 2nd phase i.e Runtime data area so at the TIME OF EXECUTION OF STATIC METHOD AT CLASS LOADING PAHSE, NON STATIC VARIABLE WILL NOT BE AVAILABLE BY DEFAULT henec we can't access non static variable from static context[static block, static method and static nested inner class] without creating the object.

```
public class StaticDemo1
{
    int x = 100;

    public static void main(String[] args)
    {
        System.out.println("x value is :"+x); //error
    }
}
```

```
class Test
{
    private int x;

    public Test(int x)
    {
        this.x = x;
    }

    public static void access()
    {
        System.out.println("x value is :"+x);  //error
    }
}

public class StaticDemo2
{
    public static void main(String[] args)
    {
        Test t1 = new Test(10);
        Test.access();
    }
}
```

Note : From any static context [static block, static methods, static nested inner class] if we want to access non static variable, Object is required.

```
package com.ravi.class_loading;

public class StaticDemo3
{
     int x;

    static
    {
        StaticDemo3 s = new StaticDemo3();
        s.x = 900;
        System.out.println(s.x);
    }

    public static void main(String[] args)
    {

    }
}
```

IQ :

```
package com.ravi.class_loading;

class Alpha
{
    protected int x = 100;
}
class Beta extends Alpha
{
    protected int x = 200;  //Variable Hiding

    public static void print()
    {
        Beta beta = new Beta();
        System.out.println("Beta class x :"+beta.x);

        Alpha alpha = beta;
        System.out.println("Alpha class x :"+alpha.x);

    }
}
public class VariableAccess
{
    public static void main(String[] args)
    {
        Beta.print();

    }

}
```

Runtime Data Areas :
----

* Once a class is loaded successfully then based on the content type, It will be divided into different Memory section, Which are as follows :

  1) Method Area
  2) HEAP Area
  3) Stack Area
  4) PC(Program Counter) Register
  5) Native Method Stack

Method Area :
----

* A .class file is loaded inside Method area and returns java.lang.Class class object.

* From this method area we can get complete information of a class like class information, variable information, method information and so on.

* We have only one method Area per JVM.

* All the static data members are stored in this Method area only and this area is sharable by all the Objects.

Method Area OR Class Area