WAP to describe Application class loader is reponsible to load user-defined .class file :

* As we know, **.class file will be loaded in the Method Area and returns Class class object.**
Example :

public **ClassLoader** getClassLoader();

```
package com.ravi.class_loading;

class Test
{
}

public class ApplicationClassLoaderDemo
{
    public static void main(String[] args)
    {
        IO.println("Test.class file is loaded by :");
        IO.println(Test.class.getClassLoader());
    }

}
```

WAP to show Platform class loader is the super class for Application class loader :

* ClassLoader class has provided a predefined non static method called getParent() through which we will get the Parent class name.

public ClassLoader getParent()

```
package com.ravi.class_loading;

class Sample
{
}

public class PlatformClassLoader {

    public static void main(String[] args)
    {
        IO.println("Super class of Application class loader is :");
        IO.println(Sample.class.getClassLoader().getParent());
    }

}
```

Bootstrap class loader :

Bootstrap class loader implementation is not provided by java because It is the internal (built-in) class loader of JVM so it provide null

```
package com.ravi.class_loading;

class Demo{}

public class BootstrapClassLoader {

    public static void main(String[] args)
    {
        IO.println(String.class.getClassLoader());
        IO.println(Demo.class.getClassLoader().getParent().getParent());
    }

}
```

Linking :

Verify :-

It ensures the correctness of the .class files, If any suspicious activity is there in the .class file then It will stop the execution immediately by throwing a runtime error i.e java.lang.VerifyError.

There is something called ByteCodeVerifier(Component of JVM), responsible to verify the loaded .class file i.e byte code. Due to this verify module JAVA is highly secure language.

java.lang.VerifyError is the sub class of java.lang.InkageError.

Prepare

[Static Field memory allocation + Static Field initialization with default value even the field is final]

It will allocate the memory for all the Static Fields, here all the Static Fields will get the default values so if we have static int x = 100; then for field x memory will be allocated (4 bytes) and now it will initialize with default value i.e 0, even the field is final.

static Test t = new Test();

Here, t is a static reference field so for t field (reference variable) memory will be allocated as per JVM implementation i.e for 32 bit JVM (4 bytes of Memory) and for 64 bit (8 bytes of memory) and initialized with null.

Resolve :

All the symbolic references (#7) will be converted into direct references OR actual reference.

Javap -verbose FileName.class

Note :- By using above command we can read the internal details of .class file.

Initialization :

Here class initialization will takes place. All the static field member will get their actual/original value and we can also use static block for static field member initialization.

Here, In this class initialization phase static field and static block is having same priority so it will executed according to the order.(Top to bottom)

**Can we write a Java Program without main method ?

```
class WithoutMain
{
    static
    {
        System.out.println("Hello User!!");
        System.exit(0);
    }
}
```

It was possible to write a java program without main method till JDK 1.6V.
From JDK 1.7v onwards, at the time of loading the .class file JVM will verify the presence of main method in the .class file. If main method is not available then it will generate a runtime error that "main method not found in so so class".

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

* The following are the ways to load the .class file into JVM memory :

1) By using java command :
Example :
java Test -> At the time of execution, We are making a request to class loader subsystem to load this Test.class file

2) By using Constructor [At the time of creating the object]
Example :

```
public class Demo
{
}

new Demo();  //Making a request to load Demo.class file
```

3) By accessing static variable OR static method
Whenver we are accessing static variable OR static method then automatically class will be loaded.

```
class Sample
{
    static int x = 100;
}

Sample.x;  -> Sample class will be loaded
```

4) By using Inheritance
Whenver we try to load sub class then first of all super class will be loaded then sub class will be loaded.

```
class Alpha
{
    static
    {
        IO.println("Alpha class SB");
    }
}
class Beta extends Alpha
{
    static
    {
        IO.println("Beta class SB");
    }
}

public class Test
{
    public static void main(String [] args)
    {
        new Beta();
    }

}
```

5) By using reflection API