Bootstrap/Primordial class Loader :-

It is responsible for loading all the predefined .class files that means all API(Application Programming Interface) level predefined classes are loaded by Bootstrap class loader.

It has the highest priority becuase Bootstrap class loader is the super class for Platform class loader.

It loads the classes from the following path
C -> Program files -> Java -> JDK -> lib -> jrt-fs.jar

Platform/Extension class loader :

Before java 9V, This class loader was known as Extension class loader because It was loading the .class file from the following path.

C -> Program files -> Java -> JDK -> lib -> ext (extension) -> 3rd Party jar file. [ojdbc14.jar]

It has highest priority than Application class loader.

From Java 9V onwards we have a new concept called JPMS (Java Platform Module System) so, instead of using jar file we can directly use module.
Example : java.base, java.compiler and so on.

Command to create the jar file :
jar cf FileName.jar FileName.class [*.class]

[If we want to compile more than one java source file at a time then the command is : javac *.java]

Application OR System class loader :

It is responsible to load all userdefined .class file into JVM memory.

It has the lowest priority because it is the sub class Platform class loader.

It loads the .class file from class path level or environment variable.

Note :-

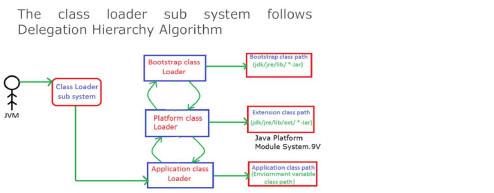
If all the class loaders are failed to load the .class file into JVM memory then we will get a Runtime exception i.e **java.lang.ClassNotFoundException.**

How Delegation Hierarchy algorithm works :-

Whenever JVM makes a request to class loader sub system to load the required .class file into JVM memory, first of all, class loader sub system makes a request to Application class loader, Application class loader will delegate(by pass) the request to the Platform class loader, Platform class loader will also delegate the request to Bootstrap class loader.

Bootstrap class loader will load the .class file from lib folder(jrt-fs.jar) and then by pass the request back to Platform class loader, Platform class loader will load the .class file from ext folder(*.jar [3rd party jar file]) OR JPMS (Java Platform Module System) and by pass the request back to Application class loader, It will load the .class file from environment variable into JVM memory.

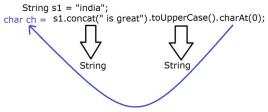
Note : java.lang.Object is the first class to be loaded into JVM Memory.



What is Method Chaning in Java ?

* It is a technique through which we can **execute multiple methods** in a single statement.

Example :



* In this method chaining, always for calling next method we depend upon current method return type.

* The final return type of the method depends upon last method call as shown in the program.

MethodChaning.java

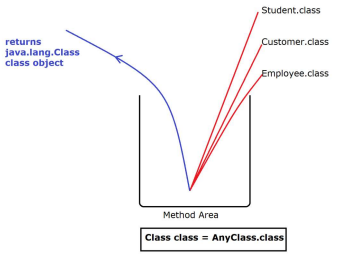
```
void main()
{
	String s1 = "india";
	char ch = s1.concat(" is great").toUpperCase().charAt(0);
	IO.println(ch);

	String s2 = "Java";
	int length = s2.concat(" technology").toUpperCase().length();
	IO.println(length);

}
```

Role of java.lang.Class class object in class loading :

* There is a predefined class called **Class** available in java.lang package.

* Whenever we load a .class file into **JVM memory** then actually the **.class file will be loaded in a very** special memory area called **Method Area and returns java.lang.Class class object.**

Example :



* java.lang.Class class has provided a predefined non static method called **getName()** through which we will get the Fully Qualified Name (Package Name + class name).

public String getName();

```
package com.ravi.class_loading;

class Employee{}
class Customer{}
class Student{}

public class ClassLoadingDescription {

    public static void main(String[] args)
    {
        Class cls = Employee.class;
        IO.println(cls.getName());

        cls = Customer.class;
        IO.println(cls.getName());

        cls = Student.class;
        IO.println(cls.getName());
    }

}
```

```
Employee.class.getClassLoader()
        ⇩
  java.lang.Class
```