\*\*\*\*\* Document about Java Class Loading \*\*\*\*\*

In this document you will learn the following things:

What is Class Loading?

Class Loader and its types.

How class loaders work?

When classes are loaded?

**What is Class Loading?**

In Java, every programmer will work on a class. We will compile and run a java class with the help of its className and execute it.

Java compilation: javac className.java

Java execution: java className

But what happens internally when we try to run a java program is the corresponding class is loaded into the JVM and all the classes that are imported in java program will be loaded into JVM.

At some point of time, While loading the class, you have encountered ClassNotFoundException, ClassCastException.

To know about class loading in depth, go through the following topics:

**Class Loader and its types:**

**Class Loader:**

In a JVM, each and every class is loaded by some instance of a java.lang.ClassLoader. The ClassLoader class is located in the java.lang package and developers are free to subclass it to add their own functionality to class loading.

**Types of ClassLoaders:**

1. Java Bootstrap Class Loader
2. Java Extension Class Loader
3. Java Application Class Loader

**Java Bootstrap Class Loader:**

Whenever a new JVM is started by typing java className, the "bootstrap class loader" is responsible for loading key Java classes like java.lang.Object and other runtime code into memory first.

**Java Extension Class Loader:**

We can store extension libraries, those that provide features that go beyond the core java runtime code.

ExtClassLoader is responsible for loading all .jar files kept in the given java.ext.dirs path.

**Java Application Class Loader:**

The application class loader is responsible for loading all of the classes kept in the path corresponding to the java.class.path system property.

Listed below are a few other class Loaders in the JDK.

java.net.URLClassLoader

java.security.SecureClassLoader

java.rmi.server.RMIClassLoader

sun.applet.AppletClassLoader

**How class loaders work?**

All class loaders except the bootstrap class loader have a parent class loader. All class loaders are of the type java.lang.ClassLoader. The parent class loader for any class loader is the class loader instance that loaded that class loader.

A class is requested out of a class loader using the loadClass() method. The internal working of this method can be seen from the source code for java.lang.ClassLoader, given below:

protected synchronized Class<?> loadClass(String className, Boolean resolve) throws ClassNotFoundException {

Class classObject=findLoadedClass(className);

if(classObject==null) {

try {

if(parent!=null) {

classObject =parent.loadClass(className,false);

}

else{

classObject=findBootstrapClass(className);

}

}

catch(ClassNotFoundException e) {

classObject=findClass(className);

}

}

if(resolve) {

resolveClass(classObject);

}

return classObject;

}

The default implementation of loadClass() method searches for classes in the following order:

1. Invoke findLoadedClass(String) to check if the class has already been loaded.
2. Invoke the loadClass method on the parent class loader. If the parent is null the class loader

built-in to the virtual machine is used,instead.

1. Invoke the findClass(String) method to find the class.

If the class was found using the above steps, the resolve flag is set true, this method will then invoke the resolveClass(Class) method on the resulting class object.

To set the parent class loader:

public class MyClassLoader extends ClassLoader {

public MyClassLoader() {

super(MyClassLoader.class.getClassLoader());

} }

The default implementation of findClass() is shown below:

protected Class<?> findClass(String className) throws ClassNotFoundException{

throw new ClassNotFoundException(className);

}

Inside of the findClass() method, the class loader needs to fetch the byte codes from some arbitrary source which can be the file system, a network URL or a database.

**When classes are loaded?**

Classes are loaded in two cases:

case 1: When the new bytecode is executed

Example: Class className=new Class();

case 2: When the bytecode make a static reference to a class

Example: System.out

**References:**

[**http://www.onjava.com/pub/a/onjava/2005/01/26/classloading.html**](http://www.onjava.com/pub/a/onjava/2005/01/26/classloading.html)

[**http://www.onjava.com/pub/a/onjava/2004/06/30/classloader2.html**](http://www.onjava.com/pub/a/onjava/2004/06/30/classloader2.html)

[**http://www.theserverside.com/news/1364680/Understanding-J2EE-Application-Server-ClassLoading-Architectures**](http://www.theserverside.com/news/1364680/Understanding-J2EE-Application-Server-ClassLoading-Architectures)

[**https://docs.oracle.com/javase/tutorial/ext/basics/load.html**](https://docs.oracle.com/javase/tutorial/ext/basics/load.html)

[**http://download.oracle.com/javase/tutorial/ext/basics/load.html**](http://download.oracle.com/javase/tutorial/ext/basics/load.html)