**[How ClassLoader Works in Java](http://javarevisited.blogspot.in/2012/12/how-classloader-works-in-java.html" \o "How ClassLoader Works in Java)**

<http://javarevisited.blogspot.in/2012/12/how-classloader-works-in-java.html#axzz51KuGV79E>

Java class loaders are used to load classes at runtime. ClassLoader in Java works on three principle: delegation, visibility and uniqueness.

Delegation principle forward request of class loading to parent class loader and only loads the class, if parent is not able to find or load class. Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader can not see classes loaded by child. Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent. Correct understanding of class loader is must to resolve issues like [NoClassDefFoundError in Java](http://javarevisited.blogspot.sg/2011/06/noclassdeffounderror-exception-in.html) and [java.lang.ClassNotFoundException](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html), which are related to class loading. ClassLoader is also an important topic in advanced Java Interviews, where good knowledge of working of Java ClassLoader and [How classpath works in Java](http://javarevisited.blogspot.ca/2011/01/how-classpath-work-in-java.html) is expected from Java programmer. I have always seen questions like, **Can one class be loaded by two different ClassLoader in Java** on various [Java Interviews](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html).  In this Java programming tutorial, we will learn what is ClassLoader in Java, How ClassLoader works in Java and some specifics about Java ClassLoader.

Default class loader: java.lang.Classloader

**What is ClassLoader in Java**

ClassLoader in Java is a class which is used to load [class files in Java](http://javarevisited.blogspot.ca/2012/05/10-points-about-class-file-in-java.html). Java code is compiled into class file by [javac](http://javarevisited.blogspot.sg/2012/12/javac-is-not-recognized-as-internal-or-external-command.html)compiler and [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html)executes Java program, by executing byte codes written in class file. ClassLoader is responsible for loading class files from file system, network or any other source. There are three default class loader used in Java, **Bootstrap** , **Extension** and **System or Application class loader**. 

Every class loader has a predefined location, from where they loads class files. Bootstrap ClassLoader is responsible for loading standard JDK class files from rt.jar and it is parent of all class loaders in Java.

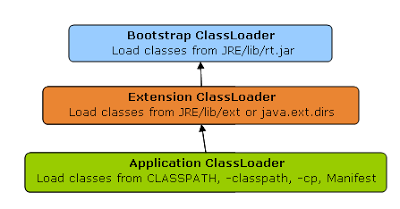
Bootstrap class loader don't have any parents, if you call String.class.getClassLoader() it will return null and any code based on that may throw [NullPointerException in Java](http://javarevisited.blogspot.com/2012/06/common-cause-of-javalangnullpointerexce.html). Bootstrap class loader is also known as**Primordial ClassLoader** in Java.    
  
  
Extension ClassLoader delegates class loading request to its parent, Bootstrap and if unsuccessful, loads class form jre/lib/ext directory or any other directory pointed by java.ext.dirs system property. Extension ClassLoader in JVM is implemented by  sun.misc.Launcher$ExtClassLoader.   
  
Third default class loader used by JVM to load Java classes is called System or Application class loader and it is responsible for loading application specific classes from [CLASSPATH](http://javarevisited.blogspot.sg/2011/01/how-classpath-work-in-java.html) environment variable, -classpath or -cp command line option, Class-Path attribute of Manifest file inside JAR. Application class loader is a child of Extension ClassLoader and its implemented by sun.misc.Launcher$AppClassLoader class. Also, except Bootstrap class loader, which is implemented in native language mostly in C,  all  Java class loaders are implemented using java.lang.ClassLoader.

In short here is the location from which Bootstrap, Extension and Application ClassLoader load Class files.

1) Bootstrap ClassLoader - JRE/lib/rt.jar

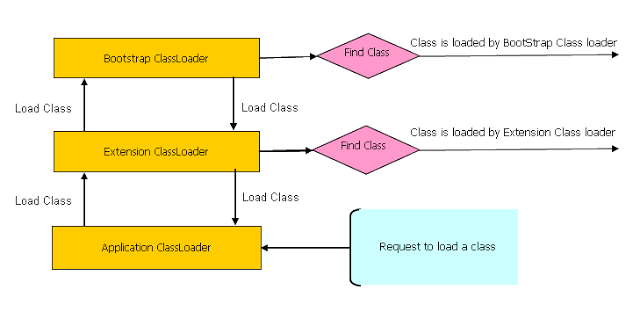
2) Extension ClassLoader - JRE/lib/ext or any directory denoted by java.ext.dirs

3) Application ClassLoader - CLASSPATH environment variable, -classpath or -cp option, Class-Path attribute of Manifest inside [JAR file](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html).

[](http://2.bp.blogspot.com/-HCTsr-j_ojw/USTOh1f8JwI/AAAAAAAAAjg/YegPspR5K48/s1600/java_classloader_hierarchy.PNG)

## How ClassLoader works in Java

[What is ClassLoader in Java, How classloader works in Java](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)As I explained earlier Java ClassLoader works in three principles : delegation, visibility and uniqueness. In this section we will see those rules in detail and understand working of Java ClassLoader with example. By the way here is a diagram which explains How ClassLoader load class in Java using delegation.

[](http://1.bp.blogspot.com/-0gOWex7Pb2E/USTOh2K7zpI/AAAAAAAAAjc/_viQADzxrsk/s1600/Java+classloader+working.PNG)

**Delegation principles**

As discussed on [when a class is loaded and initialized in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), a class is loaded in Java, when its needed. Suppose you have an application specific class called Abc.class, first request of loading this class will come to Application ClassLoader which will delegate to its parent Extension ClassLoader which further delegates to Primordial or Bootstrap class loader. Primordial will look for that class in rt.jar and since that class is not there, request comes to Extension class loader which looks on jre/lib/ext directory and tries to locate this class there, if class is found there than Extension class loader will load that class and Application class loader will never load that class but if its not loaded by extension class-loader than Application class loader loads it from [Classpath in Java](http://java67.blogspot.sg/2012/08/what-is-path-and-classpath-in-java-difference.html). Remember Classpath is used to load class files while [PATH](http://javarevisited.blogspot.ca/2011/10/how-to-set-path-for-java-unix-linux-and.html) is used to locate executable like javac or java command.

**Visibility Principle**

According to visibility principle, Child ClassLoader can see class loaded by Parent ClassLoader but vice-versa is not true. Which mean if class Abc is loaded by Application class loader than trying to load class ABC explicitly using extension ClassLoader will throw either [java.lang.ClassNotFoundException](http://javarevisited.blogspot.ca/2011/08/classnotfoundexception-in-java-example.html). as shown in below Example

**package** test;  
  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
/\*\*  
 \* Java program to demonstrate How ClassLoader works in Java,

 \* in particular about visibility principle of ClassLoader.

 \*  
 \* @author Javin Paul  
 \*/  
  
**public** **class** ClassLoaderTest {  
    
    **public** **static** **void** main(**String** args[]) {  
        **try** {            
            *//printing ClassLoader of this class*  
            **System**.out.println("ClassLoaderTest.getClass().getClassLoader() : "  
                                 + ClassLoaderTest.**class**.getClassLoader());  
  
            
            *//trying to explicitly load this class again using Extension class loader*  
            **Class**.forName("test.ClassLoaderTest", **true**   
                            ,  ClassLoaderTest.**class**.getClassLoader().getParent());  
        } **catch** (**ClassNotFoundException** ex) {  
            **Logger**.getLogger(ClassLoaderTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        }  
    }  
  
}  
  
**Output:**  
ClassLoaderTest.getClass().getClassLoader() : sun.misc.Launcher$AppClassLoader@601bb1  
16/08/2012 2:43:48 AM test.ClassLoaderTest main  
SEVERE: **null**  
java.lang.**ClassNotFoundException**: test.ClassLoaderTest  
        at java.net.**URLClassLoader**$1.run(**URLClassLoader**.java:202)  
        at java.security.**AccessController**.doPrivileged(Native **Method**)  
        at java.net.**URLClassLoader**.findClass(**URLClassLoader**.java:190)  
        at sun.misc.Launcher$ExtClassLoader.findClass(Launcher.java:229)  
        at java.lang.**ClassLoader**.loadClass(**ClassLoader**.java:306)  
        at java.lang.**ClassLoader**.loadClass(**ClassLoader**.java:247)  
        at java.lang.**Class**.forName0(Native **Method**)  
        at java.lang.**Class**.forName(**Class**.java:247)  
        at test.ClassLoaderTest.main(ClassLoaderTest.java:29)

**Uniqueness Principle**

According to this principle a class loaded by Parent should not be loaded by Child ClassLoader again. Though its completely possible to write class loader which violates Delegation and Uniqueness principles and loads class by itself, its not something which is beneficial. You should follow all  class loader principle while writing your own ClassLoader.

## How to load class explicitly in Java

Java provides API to explicitly load a class by Class.forName(classname) and Class.forName(classname, initialized, classloader), remember JDBC code which is used to load JDBC drives we have seen in [Java program to Connect Oracle database](http://javarevisited.blogspot.ca/2012/04/java-program-to-connect-oracle-database.html). As shown in above example you can pass name of ClassLoader which should be used to load that particular class along with binary name of class. Class is loaded by calling loadClass() method of java.lang.ClassLoader class which calls findClass() method to locate bytecodes for corresponding class. In this example Extension ClassLoader uses java.net.URLClassLoader which search for class files and resources in [JAR](http://javarevisited.blogspot.ca/2012/10/5-ways-to-add-multiple-jar-to-classpath-java.html) and directories. any search path which is ended using "/" is considered directory. If findClass() does not found the class than it throws[java.lang.ClassNotFoundException](http://javarevisited.blogspot.de/2012/03/jdbc-javalangclassnotfoundexception.html) and if it finds it calls defineClass() to convert bytecodes into a .class instance which is returned to the caller.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | [**next →**](https://www.javatpoint.com/connectivity-with-access-without-dsn)[**← prev**](https://www.javatpoint.com/example-to-connect-to-the-oracle-database) Example to connect to the mysql database in java For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity.  In this example we are using MySql as the database. So we need to know following informations for the mysql database:   1. **Driver class:**The driver class for the mysql database is **com.mysql.jdbc.Driver**. 2. **Connection URL:**The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name. 3. **Username:**The default username for the mysql database is **root**. 4. **Password:**Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.   Let's first create a table in the mysql database, but before creating table, we need to create database first.   1. create database sonoo; 2. use sonoo; 3. create table emp(id **int**(10),name varchar(40),age **int**(3));  Example to Connect Java Application with mysql database In this example, sonoo is the database name, root is the username and password.   1. **import** java.sql.\*; 2. **class** MysqlCon{ 3. **public** **static** **void** main(String args[]){ 4. **try**{ 5. Class.forName("com.mysql.jdbc.Driver"); 6. Connection con=DriverManager.getConnection( 7. "jdbc:mysql://localhost:3306/sonoo","root","root"); 8. //here sonoo is database name, root is username and password 9. Statement stmt=con.createStatement(); 10. ResultSet rs=stmt.executeQuery("select \* from emp"); 11. **while**(rs.next()) 12. System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3)); 13. con.close(); 14. }**catch**(Exception e){ System.out.println(e);} 15. } 16. }   [download this example](https://www.javatpoint.com/src/jdbc/MysqlCon.zip)  The above example will fetch all the records of emp table.  To connect java application with the mysql database mysqlconnector.jar file is required to be loaded.  [download the jar file mysql-connector.jar](https://www.javatpoint.com/src/jdbc/mysql-connector.jar) Two ways to load the jar file:  1. paste the mysqlconnector.jar file in jre/lib/ext folder 2. set classpath  1) paste the mysqlconnector.jar file in JRE/lib/ext folder:  |  | | --- | | Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here. |  2) set classpath:  |  | | --- | | There are two ways to set the classpath:   * temporary * permanent |  How to set the temporary classpath  |  | | --- | | open command prompt and write: |  1. C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.; | |

**Where to use ClassLoader in Java**

ClassLoader in Java is a powerful concept and used at many places. One of the *popular example of ClassLoader* is AppletClassLoader which is used to load class by Applet, since Applets are mostly loaded from internet rather than local file system, By using separate ClassLoader you can also loads same class from multiple sources and they will be treated as different class in [JVM](http://javarevisited.blogspot.ca/2011/12/jre-jvm-jdk-jit-in-java-programming.html). J2EE uses multiple class loaders to load class from different location like classes from WAR file will be loaded by Web-app ClassLoader while classes bundled in EJB-JAR is loaded by another class loader. Some web server also supports hot deploy functionality which is implemented using ClassLoader. You can also use ClassLoader to load classes from database or any other persistent store.

That's all about **What is ClassLoader in Java** and **How ClassLoader works in Java**. We have seen delegation, visibility and uniqueness principles which is quite important to debug or troubleshoot any ClassLoader related issues in Java. In summary knowledge of How ClassLoader works in Java is must for any Java developer or architect to design Java application and packaging.

Read more: [http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html#ixzz51Kvz6vGM](http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html" \l "ixzz51Kvz6vGM)