**2 Solution of java.lang.OutOfMemoryError in Java**

Everyone in java development face **java.lang.OutOfMemoryError** now and then, OutOfMemoryError in Java is one problem which is more due to system's limitation (memory) rather than due to programming mistakes in most cases though in certain cases you could have **memory leak** which causing **OutOfMemoryError**. I have found that even though java.lang.OutOfMemoryError is quite common basic knowledge of its cause and solution is largely unknown among junior developers. In this article we will explore *what is java.lang.OutOfMemoryError*; Why OutOfMemoryError comes in Java application, different type of OutOfMemoryError and *How to fix OutOfMemoryError in Java*. This article is purely meant to provide basic knowledge of java.lang.OutMemoryError and won't discuss profiling in detail.

**What is java.lang.OutOfMemoryError in Java**

OutOfMemoryError in Java is a subclass **of java.lang.VirtualMachineError** and JVM throws java.lang.OutOfMemoryError when it ran *out of memory in heap*. OutOfMemoryError in Java can come any time in heap mostly while you try to create an object and there is not enough space in heap to allocate that object. [javavdoc of OutOfMemoryError](http://download.oracle.com/javase/6/docs/api/) is not very informative about this though.

Types of OutOfMemoryError in Java

I have seen mainly two types of OutOfMemoryError in Java:

1) **Java.lang.OutOfMemoryError: Java heap space**  
2) **Java.lang.OutOfMemoryError: PermGen space**  
  
though both of them occur because JVM ran out of memory they are quite different to each other and there solutions are independent to each other.

**Difference between "java.lang.OutOfMemoryError: Java heap space" and "java.lang.OutOfMemoryError: PermGen space"**

If you are familiar with different generations on heap and [How garbage collection works in java](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html) and aware of new, old and permanent generation of heap space then you would have easily figured out this OutOfMemoryError in Java. Permanent generation of heap is used to store String pool and various Meta data required by JVM related to Class, method and other java primitives. Since **in most of JVM default size of Perm Space is around "64MB"** you can easily ran out of memory if you have too many classes or huge number of Strings in your project. Important point to remember is that it doesn't depends on **–Xmx** value so no matter how big your total heap size you can ran OutOfMemory in perm space. Good think is you can specify **size of permanent generation** using JVM options **"-XX:PermSize"** and  **"-XX:MaxPermSize"** based on your project need.  
  
One small thing to remember is that "=" is used to separate parameter and value while specifying **size of perm space in heap** while "=" is not required while [**setting maximum heap size in java**](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html), as shown in below example.  
  
**export JVM\_ARGS="-Xmx1024m -XX:MaxPermSize=256m"**

Another reason of "**java.lang.OutOfMemoryError: PermGen**" is memory leak through Classloaders and it’s very often surfaced in WebServer and application server like tomcat, webshere, glassfish or weblogic. In Application server different classloaders are used to load different web application so that you can deploy and undeploy one application without affecting other application on same server, but while undeploying if container some how keeps reference of any class loaded by application class loader than that class and all other related class will not be garbage collected and can quickly fill the PermGen space if you deploy and undeploy your application many times. "*java.lang.OutOfMemoryError: PermGen*” has been observed many times in tomcat in our last project but solution of this problem are really tricky because first you need to know which class is causing memory leak and then you need to fix that. Another reason of OutOfMemoryError in PermGen space is if any thread started by application doesn't exit when you undeploy your application.

These are just some example of infamous classloader leaks, anybody who is writing code for loading and unloading classes have to be very careful to avoid this. You can also use **visualgc** for monitoring PermGen space, this tool will show graph of PermGen space and you can see how and when Permanent space getting increased. I suggest using this tool before reaching to any conclusion.

Another rather unknown but interesting cause of "java.lang.OutOfMemoryError: PermGen" we found is introduction of JVM options **"-Xnoclassgc**". This option sometime used to avoid loading and unloading of classes when there is no further live references of it just to avoid performance hit due to frequent loading and unloading, but using this option is J2EE environment can be very dangerous because many framework e.g. Struts, spring etc uses reflection to create classes and with frequent deployment and undeployment you can easily ran out of space in **PermGen** if earlier references was not cleaned up. This instance also points out that some time bad JVM arguments or configuration can cause OutOfMemoryError in Java.

So conclusion is avoid using ***"-Xnoclassgc*"** in J2EE environment especially with AppServer.

**Tomcat to Solve OutOfMemoryError in PermGen Space**

From tomcat > 6.0 onward tomcat provides memory leak detection feature which can detect many common memory leaks on web-app perspective e.g ThreadLocal memory leaks, JDBC driver registration, RMI targes, LogFactory and Thread spawned by web-apps. You can check complete details on htp://wiki.apache.org/tomcat/MemoryLeakProtection you can also detect memory leak by accessing manager application which comes with tomcat, in case you are experiencing memory leak on any java web-app its good idea to run it on tomcat.

**How to solve java.lang.OutOfMemoryError: Java heap space**

1. Easy way to solve OutOfMemoryError in java is to [*increase the maximum heap size*](http://javarevisited.blogspot.com/2011/08/increase-heap-size-maven-ant.html) by using JVM options "-Xmx512M", this will immediately solve your OutOfMemoryError. This is my preferred solution when I get OutOfMemoryError in Eclipse, Maven or ANT while building project because based upon size of project you can easily ran out of Memory.here is **an example of increasing maximum heap size of JVM**, Also its better to keep **-Xmx to -Xms** ration either 1:1 or 1:1.5 if you are setting heap size in your java application  
     
   **export JVM\_ARGS="-Xms1024m -Xmx1024m"**  
     
   **2**) Second way to resolve OutOfMemoryError in Java is rather hard and  comes when you don't have much memory and even after increase maximum heap size you are still getting java.lang.OutOfMemoryError, in this case you probably want to profile your application and look for any memory leak. You can use [**Eclipse Memory Analyzer**](http://www.eclipse.org/mat/) to examine your heap dump or you can use any profiler like Netbeans or JProbe. This is tough solution and requires some time to analyze and **find memory leaks**.

**How to solve java.lang.OutOfMemoryError: PermGen space**

As explained in above paragraph this OutOfMemory error in java comes when Permanent generation of heap filled up. To fix this OutOfMemoryError in Java you need to *increase heap size of Perm space* by using JVM option   **"-XX:MaxPermSize".** You can also specify initial size of Perm space by using    **"-XX:PermSize"** and keeping both initial **and maximum Perm Space** you can prevent some full garbage collection which may occur when Perm Space gets re-sized. Here is **how you can specify initial and maximu Perm size in Java**:  
  
**export JVM\_ARGS="-XX:PermSize=64M -XX:MaxPermSize=256m"**  
  
 Some time java.lang.OutOfMemoryError  in Java gets tricky and on those cases profiling remain ultimate solution.Though you have freedom to increase heap size in java, it’s recommended that to follow memory management practices while coding and setting null to any unused references.  
That’s all from me on **OutOfMemoryError in Java** I will try to write more about finding memory leak in java and using profiler in some other post. Please share what is your approach to solve *java.lang.OutOfMemoryError in Java*.  
**Important Note:** From Tomcat > 6.0 onward tomcat provides memory leak detection feature which can detect many common memory leaks on Java application e.g ThreadLocal memory leaks, JDBC driver registration, RMI targes, LogFactory and Thread spawned by webapps. You can check complete details on htp://wiki.apache.org/tomcat/MemoryLeakProtection. You can also detect memoy leak by accessing manager application which comes with tomcat, in case you are experiencing memory leak on any java webapp its good idea to run it on tomcat to find out reason of OutOfMemoryError in PermGen space.

**Tools to investigate and fix OutOfMemoryError in Java**

Java.lang.OutOfMemoryError is a kind of error which needs lot of investigation to find out root cause of problem, which object is taking memory, how much memory it is taking or finding dreaded memory leak and you can't do this without having knowledge of available tools in java space. Here I am listing out some free tools which can be used to analyze heap and will help you to find culprit of OutOfMemoryError

1) **Visualgc**: Visualgc stands for Visual Garbage Collection Monitoring Tool and you can attach it to your instrumented hostspot JVM. Main strength of visualgc is that it displays all key data graphically including class loader, garbage collection and JVM compiler performance data.

The target JVM is identified by its virtual machine identifier also called as vmid. You can read more about visualgc and vmid options here.

**2) Jmap**: Jmap is a command line utility comes with JDK6 and allows you to take a memory dump of heap in a file. It’s easy to use as shwon below:

**jmap -dump:format=b,file=heapdump 6054**

Here file specifies name of memory dump file which is "heapdump" and 6054 is PID of your Java progress. You can find the PDI by using "ps -ef” or windows task manager or by using tool called "jps"(Java Virtual Machine Process Status Tool).

**3) Jhat**: Jhat was earlier known as hat (heap analyzer tool) but it is now part of JDK6. You can use jhat to analyze heap dump file created by using "**jmap**". Jhat is also a command line utility and you can rum it from cmd window as shown below:

jhat -J-Xmx256m heapdump

Here it will analyze memory-dump contained in file "heapdump". When you start **jhat** it will read this heap dump file and then start listening on http port, just point your browser into port where jhat is listening by default 7000 and then you can start analyzing objects present in heap dump.

**4) Eclipse memory analyzer:**Eclipse memory analyzer (MAT) is a tool from eclipse foundation to analyze java heap dump. It helps to find classloader leaks and memory leaks and helps to minimize memory consumption.you can use MAT to analyze heap dump carrying millions of object and it also helps you to extract suspect of memory leak. See here for more information.

**Difference between ClassNotFoundException vs NoClassDefFoundError in Java**

**What is ClassNotFoundException and NoClassDefFoundError**

From last few weeks I have been facing a cluster of **ClassNotFoundException and NoClassDefFoundError** while setting up a new project in Java. This new java project has lots of dependency on various jars and some of the jar even contains the same name of file which makes my problem even more problematic. While working with NoClassDefFoundError and ClassNotFoundException I thought to document my experience and I have already shared some on [3 ways to resolve NoClassDefFoundError in java](http://javarevisited.blogspot.com/2011/07/classnotfoundexception-vs.html) and [how to resolve ClassNotFoundException in java](http://javarevisited.blogspot.com/2011/08/classnotfoundexception-in-java-example.html). in this article though focus will be on similarity and  *differences between java.lang.ClassNotFoundException and java.lang.NoClassDefFoundError in Java.*

**NoClassDefFoundError vs ClassNotFoundException**

Before seeing the differences between ClassNotFoundException and NoClassDefFoundError let's see some similarities which are main reason of confusion between these two errors:  
  
1) Both NoClassDefFoundError and ClassNotFoundException are related to unavailability of a class at run-time.  
2) Both ClassNotFoundException and NoClassDefFoundError are related to java classpath.  
  
Now let's see the **difference between NoClassDefFoundError and ClassNotFoundException**:  
  
1) ClassNotFoundException comes in java if we try to load a class at run-time using with **Class.forName()** or **ClassLoader.loadClass()** or **ClassLoader.findSystemClass()** method and requested class is not available in Java. the most of the time it looks like that we have the class in classpath but eventually it turns out to be issue related to classpath and application may not be using classpath what we think it was using e.g. classpath defined in jar's manifest file will take precedence over CLASSPATH or -cp option, for more details see [How classpath works in java](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html). On the other hand NoClassDefFoundError is little different than ClassNotFoundException, in this case culprit class was present during compile time and let's application to compile successfully and linked successfully but not available during run-time due to various reason.  
  
2) ***ClassNotFoundException*** is a checked Exception derived directly from java.lang.Exception class and you need to provide explicit handling for it while *NoClassDefFoundError* is an Error derived from LinkageError.  
  
3) If you are using [ClassLoader in Java](http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html) and have two classloaders then if a ClassLoader tries to access a class which is loaded by another classloader will result in **ClassNoFoundException**.  
  
4) **ClassNotFoundException** comes up when there is an explicit loading of class is involved by providing name of class at runtime using **ClassLoader.loadClass**, **Class.forName** while **NoClassDefFoundError** is a result of implicit loading of class because of a method call from that class or any variable access.  
  
Please let us know if you are aware of any other difference between NoClassDefFoundError and ClassNotFoundException in Java , I would be happy to incorporate those.   
  
 **3 ways to solve java.lang.NoClassDefFoundError in Java J2EE**

**What is Exception in thread "main" java.lang.NoClassDefFoundError?**

I know how frustrating is to see Exception in thread "main" java.lang.NoClassDefFoundError Which is a manifestation of NoClassDefFoundError in Java , I have seen it couple of times and spent quite a lot time initially to figure out what is wrong , which class is missing etc. First mistake I did was [mingling java.lang.ClassNotfoundException and NoClassDefFoundError](http://javarevisited.blogspot.com/2011/07/classnotfoundexception-vs.html), in reality they are totally different and second mistake  was using trial and error method to solve this **java.lang.NoClassDefFoundError** instead of understanding why NoClassDefFoundError is coming, what is real reason behind NoClassDefFoundError and how to resolve this. In this Java tutorial I have tried to rectify that mistakes and uncover some secrets of NoClassDefFoundError in Java and will share my experience around it. **NoClassDefFoundError** is not something which cannot be resolved or hard to resolve it’s just its manifestation which puzzles most of Java developer. This is the most common error in Java development along with [java.lang.OutOfMemoroyError: Java heap space](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html) and [java.lang.OutOfMemoryError: PermGen space](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html)Anyway let’s see Why NoClassDefFoundError comes in Java and what to do to resolve NoClassDefFoundError in Java.

**What is reason of NoClassDefFoundError in Java?**

NoClassDefFoundError in Java comes when [Java Virtual Machine](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) is not able to find a particular class at runtime which was available during compile time. For example if we have a method call from a class or accessing any static member of a Class and that class is not available during run-time then JVM will throw NoClassDefFoundError. It’s important to understand that this is different than ClassNotFoundException which comes while trying to load a class at [run-time](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) only and name was provided during runtime not on [compile time](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html). Many Java developer mingle this two Error and gets confused.

In short **NoClassDefFoundError** will come if a class was present during compile time but not available in java classpath during runtime. Normally you will see below line in log when you get NoClassDefFoundError:

**Exception in thread "main" java.lang.NoClassDefFoundError**

Exception in thread “main” simply indicate that its [“main” thread](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html) which is not able to find a particular class it could be any thread so just don’t worry . Difference between this error coming in main thread and other thread is , when Exception in thread “main” comes program crashes or shut it self down as opposed to other [thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) in which case your program will continue to run.,

**Difference between java.lang.NoClassDefFoundError and ClassNotFoundException in Java**

Many a times we confused ourselves with java.lang.ClassNotFoundException and java.lang.NoClassDefFoundError, though both of them related to [Java Classpath](http://javarevisited.blogspot.sg/2011/01/how-classpath-work-in-java.html) they are completely different to each other. ClassNotFoundException comes when JVM tries to load a class at runtime dynamically means you give the name of class at runtime and then JVM tries to load it and if that class is not found in classpath it throws [java.lang.ClassNotFoundException](http://javarevisited.blogspot.com/2011/08/classnotfoundexception-in-java-example.html). While in case of NoClassDefFoundError the problematic class was present during Compile time and that's why program was successfully compile but not available during runtime by any reason. NoClassDefFoundError is easier to solve than ClassNotFoundException in my opinion because here we know that Class was present during build time but it totally depends upon environment, if you are working in J2EE environment than you can get NoClassDefFoundError even if class is present because it may not be visible to corresponding ClassLoader. See my post [NoClassDefFoundError vs ClassNotFoundException](http://javarevisited.blogspot.com/2011/07/classnotfoundexception-vs.html) in Java for more details.

**How to resolve java.lang.NoClassDefFoundError:**

[java.lang.NoClassDefFoundError in Java solution ](http://javarevisited.blogspot.sg/2010/10/difference-between-hashmap-and.html) Obvious reason of NoClassDefFoundError is that a particular class is not available in Classpath, so we need to add that into Classpath or we need to check why it’s not available in Classpath if we are expecting it to be. There could be multiple reasons like:

1) Class is not available in [Java Classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html).

2) You might be running your program using [jar command](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html) and class was not defined in manifest file's [ClassPath attribute](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html).

3) Any start-up script is overriding Classpath environment variable.

4) Because NoClassDefFoundError is a sub class of java.lang.LinkageError it can also come if one of it dependency like native library may not available.

4) Check for java.lang.ExceptionInInitializerError in your log file. NoClassDefFoundError due to failure of [static initialization](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) is quite common.

5) If you are working in J2EE environment than visibility of Class among multiple Classloaders can also cause java.lang.NoClassDefFoundError, see examples and scenario section for detailed discussion.

We will now see couple of example and scenarios when ***java.lang.NoClassDefFoundError*** has came before and how its been resolved. This can help you to troubleshoot root cause of NoClassDefFoundError in Java application.

**NoClassDefFoundError in Java - Example and Scenarios**

1. Simple example of NoClassDefFoundError is class belongs to a missing JAR file or JAR was not added into classpath or sometime jar's name has been changed by someone like in my case one of my colleague has changed **tibco.jar** into **tibco\_v3.jar** and by program is failing with java.lang.NoClassDefFoundError and I was wondering what's wrong.

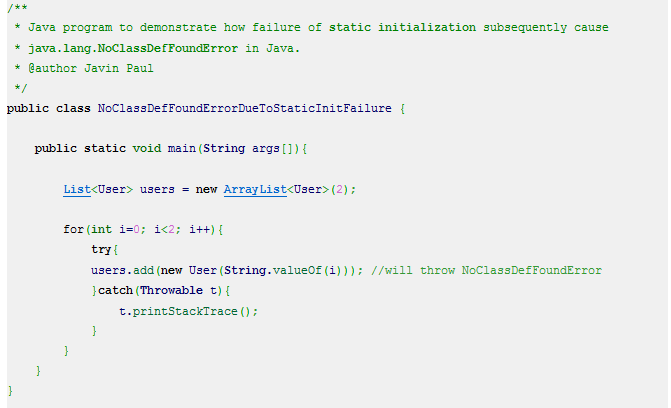
2. Class is not in Classpath, there is no sure shot way of knowing it but many a times you can just have a look to print System.getproperty("java.classpath")and it will print the classpath from there you can at least get an idea of your actual runtime classpath.

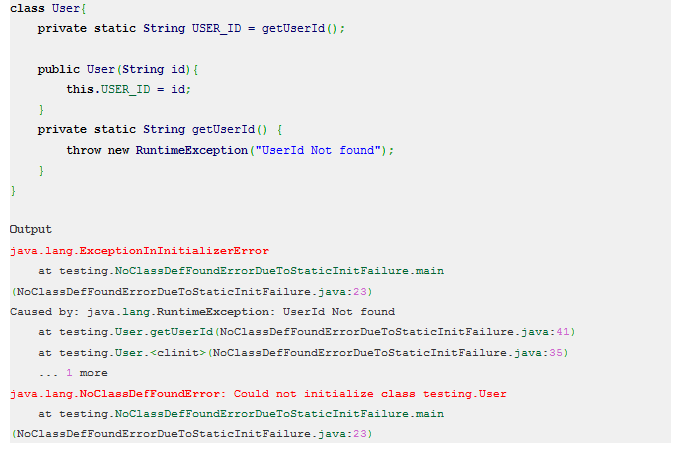
3. Just try to run with explicitly -classpath option with the classpath you think will work and if its working then it's sure short sign that some one is [overriding java classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html).

**NoClassDefFoundError in Java due to Exception in Static Initializer block**

4) This is another common reason of java.lang.NoClassDefFoundError, when your class perform some static initialization in static block like many [Singleton classes](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html) initialized itself on static block  to take advantage of [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) provided by JVM during class initialization process, and if static block throw an Exception, the class which is referring to this class will get NoclassDefFoundError in Java. If you look at your log file you should watch for any java.lang.ExceptionInInitializerError because that could trigger java.lang.NoClassDefFoundError: Could not initialize class on other places. Like in below code example, During [class loading and initialization](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html) User class is [throwing Exception](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) from static initializer block, which trigger ExceptionInInitializerError during first time loading of User class in response to new User() call. Later rest of new User() are failing as java.lang.NoClassDefFoundError. situation gets worst if original ExceptionInInitializerError, which is root cause here is silently eaten by any code.

Code Example of NoClassDefFoundError due to Static block Exception:





5) Since NoClassDefFoundError is a  also a LinkageError which arises due to dependency on some other class , you can also get java.lang.NoClassDefFoundError if your program is dependent on native library and corresponding dll is not there. Remember this can also trigger [**java.lang.UnsatisfiedLinkError: no dll in java.library.path Exception Java**](http://javarevisited.blogspot.com/2012/03/javalangunsatisfiedlinkerror-no-dll-in.html). In order to solve this keep your dll along with JAR.

6) If you are using ANT build file [create JAR and manifest file](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html) than its worth noting to debug till that level to ensure that [ANT build](http://javarevisited.blogspot.sg/2010/10/ant-basics.html) script is getting correct value of classpath and appending it to manifest.mf file.

7) Permission issue on JAR file can also cause NoClassDefFoundError in Java. If you are running your Java program in multi-user operating system like Linux than you should be using  application user id for all your application resources like JAR files, libraries and configuration. If you are using shared library which is shared among multiple application which runs under different users  then you may run with permission issue , like JAR file is owned by some other user and not accessible to your application. One of our reader “it’s me said” faced java.lang.NoClassDefFoundError due to this reason. See his comment also.

8) Typo on XML Configuration can also cause NoClassDefFoundError in Java. As most of Java frameworks like [Spring](http://javarevisited.blogspot.sg/2011/09/spring-interview-questions-answers-j2ee.html), [Struts](http://javarevisited.blogspot.sg/2011/11/struts-interview-questions-answer-j2ee.html) they all use xml configuration for specifying beans. By any chance if you put the bean name wrong, it may surface as java.lang.NoClassDefFoundError while loading other class which has dependency on wrongly named bean. This is quite common on Spring MVC framework and Apache Struts where you get tons of **Exception in thread "main" java.lang.NoClassDefFoundError ,** while deploying your WAR or EAR file.

9) Another example of java.lang.NoClassDefFoundError as mentioned by our reader Nick is that when your compiled class which is defined in a package, doesn’t present in same package while loading like in case of JApplet it will throw NoClassDefFoundError in Java. Also see Nick’s comment on this error.

10) java.lang.NoClassDefFoundError can be caused due to multiple classloaders in J2EE environments. Since J2EE doesn’t mention standard class-loader structure and it depends upon different vendors like Tomcat, WebLogic, WebSphere on how they load different components of J2EE like WAR file or EJB-JAR file. In order to troubleshoot **NoClassDefFoundError in J2EE application** knowledge of [How ClassLoader works in Java](http://javarevisited.blogspot.sg/2011/01/how-classpath-work-in-java.html) is mandatory. Just to recap ClasLoader works on three principle delegation, visibility and uniqueness. Delegation means every request to load a class is delegated to parent classloader, visibility means ability to found classes loaded by classloader, all child classloader can see classes loaded by parent classloader but parent classloader can not see the class loaded by child classloaders. Uniqueness enforce that class loaded by parent will never be reloaded by child clasloaders. Now suppose if a class say User is present in both WAR file and EJB-JAR file and loaded by WAR classloader which is child classloader which loads class from EJB-JAR. When a code in EJB-JAR refer to this User class, Classloader which loaded all EJB class doesn’t found that because it was loaded by WAR classloader which is child of it. This will result in java.lang.NoClassDefFoundError for User class. Also If class is present in both JAR file and you will call [equals method](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) to compare those two object, it will result in ClassCastException as object loaded by two different classloader can not be equal.

11) Some of reader of this blog also suggested that they get Exception in thread "main" java.lang.NoClassDefFoundError: com/sun/tools/javac/Main , this error means either your [Classpath](http://javarevisited.blogspot.sg/2011/01/how-classpath-work-in-java.html), [PATH](http://javarevisited.blogspot.sg/2011/10/how-to-set-path-for-java-unix-linux-and.html)  or [JAVA\_HOME](http://javarevisited.blogspot.sg/2012/02/how-to-set-javahome-environment-in.html) is not setup properly or JDK installation is not correct. which can be resolved by re-installing JDK. IF you are getting this error try to reinstall JDK . One of our reader got this issue after installing jdk1.6.0\_33 and then reinstalling JDK1.6.0\_25, he also has his JRE and JDK on different folder. See his comment also by searching JDK1.6.0\_33 .

12) Java program can also throw java.lang.NoClassDefFoundError during linking which occurs [during class loading in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html). One of the example of this scenario is just delete the User class from our static initializer failure example after compilation and they try to run the program. This time you will get java.lang.NoClassDefFoundError directly without  java.lang.**ExceptionInInitializerError** and message for NoClassDefFoundError is also just printing name of class as testing/User i.e. User class from testing package. Watch out for this carefully as here root cause is absent of User.class file. 

Let me know how exactly you are facing **NoClassDefFoundError in Java**  and I will guide you how to troubleshoot it, if you are facing with something new way than I listed above we will probably document if for benefit of others and again don’t afraid with Exception in thread "main" java.lang.NoClassDefFoundError

**How to resolve java.lang.ClassNotFoundException in Java**

**What is ClassNotFoundException in Java**

**ClassNotFoundException** is one of Java nightmare every Java developer face in there day to day life. [java.lang.NoClassDefFoundError](http://javarevisited.blogspot.sg/2011/06/noclassdeffounderror-exception-in.html) and java.lang.ClassNotFoundException are two errors  which occurs by and now and chew up of your precious time while finding and fixing root cause. From the name java.lang.ClassNotFoundException looks quite simple but underlying cause of it is always different and which classifies it as an environmental issue. In this java tutorial we will see *what is ClassNotFoundException in java*, what is real cause of it and how to fix it along with some more frequent and infamous examples of java.lang.ClassNotFoundException in Java or J2EE, Don’t mistake this exception with [**NoClassDefFoundError in Java**](http://javarevisited.blogspot.com/2011/06/noclassdeffounderror-exception-in.html) which is also due to incorrect [classpath in Java](http://java67.blogspot.sg/2012/08/what-is-path-and-classpath-in-java-difference.html).  Though both of them are related to missing class file when Java tries to load class in Java they are completely different to each other.  Correct understanding of  [When class is loaded in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html) and [How Classpath works](http://javarevisited.blogspot.sg/2011/01/how-classpath-work-in-java.html)  is must to troubleshoot and fix this error quickly.

**What is java.lang.classNotFoundException in Java**

As the name suggests classNotFoundException in Java is a subclass of java.lang.Exception and Comes when [Java Virtual Machine](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) tries to load a particular class and doesn't found the requested class in classpath. Another important point about this Exception is that, It is a [checked Exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html) and you need to provide explicitly Exception handling while using methods which can possibly throw classnotfoundexception in java either by using try-catch block or by using [throws clause](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html). Though underlying concept of this exception is simple but it always manifest itself in such format that you need to spend some time to figure out what exactly wrong with your classpath. If you want to know nasty [secrets of java classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html)  which can cause issue see the link.

**When ClassNotFoundException occurs in Java:**  
  
As per [java doc java.lang.classNotFoundException](http://download.oracle.com/javase/1.4.2/docs/api/java/lang/ClassNotFoundException.html) comes in following cases:

1) When we try to [load a class](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html) by using Class.forName() method and [.class file](http://javarevisited.blogspot.sg/2012/05/10-points-about-class-file-in-java.html) or binary of class is not available in classpath.  
2) When Classloader try to load a class by using findSystemClass () method.  
3) While using loadClass() method of class ClassLoader in Java.

These statements are completely true in terms of theory of ClassNotFoundExcepiton in Java but as per my experience the concept is "ClassNotFoundException will come only when JVM tries to load a class at run-time, nothing related to [compile time](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) unlike NoClassDefFoundError". Also since till run time JVM doesn't know about this Class it can only be done by above specified method or by employing Reflection to read the name of class from some configuration file like in case of struts its struts-config.xml file and then load the class specified on those configuration file. [Reflection](http://javarevisited.blogspot.sg/2012/04/how-to-invoke-method-by-name-in-java.html) is great power of Java but you need to be aware of java.lang.classNotFoundException while using it or [loading class in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html).

**Examples of classnotfoundexception in java**  
Though java.lang.classNotFoundException is very common and it can come for any classes, I usually see it while doing JDBC connectivity like when I was writing [Java program to connect Oracle database](http://javarevisited.blogspot.sg/2012/04/java-program-to-connect-oracle-database.html). I am going to list some of the most common scenario where you will get classnotfoundexception in java.

**java.lang.classnotfoundexception com.mysql.jdbc.driver**

This is classical and most infamous example of  and also my first encounter with java.lang.ClassNotFoundException and comes when you are writing JDBC connectivity code and trying to load the [JDBC driver](http://javarevisited.blogspot.sg/2012/05/different-types-of-jdbc-drivers-in-java.html). In this particular case of ClassNotFoundException looks like mysql driver jar file is missing from Classpath. If you pay attention you will find that we use method Class.forName (“driver”) to load the driver class which resides in a particular jar in case of this its mysql-connector.jar and if that jar is not in classpath or not accessible to JVM it will throw  [java.lang.ClassNotFoundException: com.mysql.jdbc.Driver](http://javarevisited.blogspot.sg/2012/03/jdbc-javalangclassnotfoundexception.html)

Here are few more infamous examples of java.lang.ClassnotFoundException which comes here and there while doing any Java J2EE project.

java.lang.classnotfoundexception org.hibernate.hql.ast.hqltoken

java.lang.classnotfoundexception org.springframework.web.context.contextloaderlistener

java.lang.classnotfoundexception org.eclipse.core.runtime.adaptor.eclipsestarter

java.lang.classnotfoundexception org.apache.catalina.startup.catalina

java.lang.classnotfoundexception javax.mail.messagingexception

java.lang.classnotfoundexception oracle.jdbc.driver.oracledriver  
  
This ClassNotfoundException comes when you are trying to [connect Oracle database from Java program using JDBC](http://javarevisited.blogspot.sg/2012/04/java-program-to-connect-oracle-database.html) but you don't have corresponding Oracle driver e.g.ojdbc6.jar is not in classpath of your Java program

**More Complicated ClassNotFoundException**

With the advent of dynamic library e.g. OSGi and ClassLoader in Java, this exception can be more tricky and hard to find. Thanks to Mr. Anonymous who has summarized this beautifully, here it is what he says

“It can become a bit more complicated than that. In truth a class does not have to be just visible by the [JVM](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html) through its classpath, but be visible by the Classloader being used. When you are in a multi-classloader environment (In a EE environment, for example, but not limited to), each classloader may have its own rules to search for the classes, and this behavior might depend on the dynamic hierarchy of the Classloaders.

For example, in a project that uses an EAR packaging with WARs inside it, libraries in the lib folder of the EAR are visible to classes inside a WAR, but any classes packaged in a jar put in the WEB-INF/lib on the WAR cannot be seen by classes in different modules (other WARs, EJB-JARS, etc).

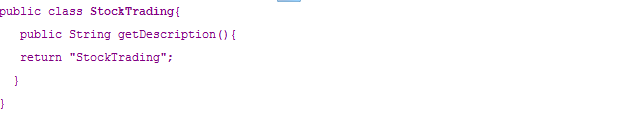
It can get really complicated as its common for different modules depending on different versions of the same libraries as different modules depend on each other. It can be a challenge to manage this. Sometimes the classloader can see multiple versions of the same class; sometimes they can see no version at all. Sometimes different dependency paths end in different versions of the same class. And many of this cases end in a ClassNotFoundException.

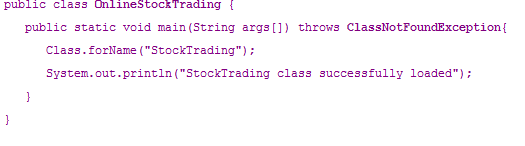
And then we have OSGi... “. If a class is not visible to ClassLoader than it can also throw NoClassDefFoundError in Java as explained in  [3 ways to resolve NoClassDefFoundError in Java](http://javarevisited.blogspot.sg/2011/06/noclassdeffounderror-exception-in.html).

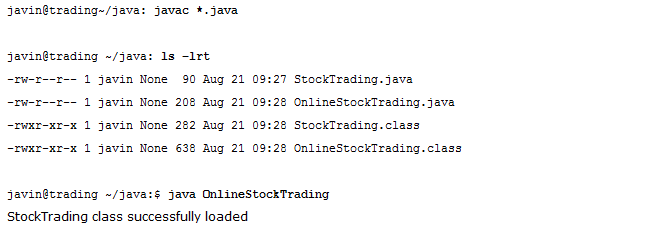
**How to fix java.lang.ClassNotFoundException in Java**  
As you have seen from above examples its clear problem of [classpath](http://javarevisited.blogspot.sg/2011/01/how-classpath-work-in-java.html), so here is **my approach to fix or resolve java.lang.ClassNotFoundException**:  
  
1) First find out the jar file on which problematic class file is present for example in case of "**com.mysql.jdbc.driver**" its mysql-connector-java.jar. If you don't know how to find which [jar file](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html) a particular class you can see [eclipse shortcuts](http://javarevisited.blogspot.com/2010/10/eclipse-tutorial-most-useful-eclipse.html) to do that or you can simply do "Ctrl+T" in Eclipse and type the name of class, It will list all the jar in the order they appear in eclipse classpath.

2) Check whether your classpath contains that jar, if your classpath doesn't contain the jar then just add that class in your classpath.

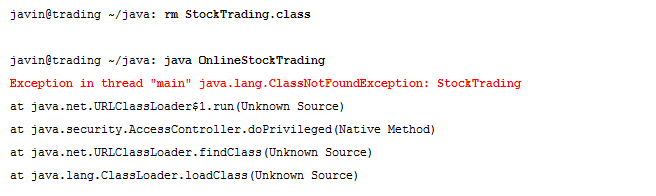
3) If it’s present in your classpath then there is high chance that your classpath is getting [overridden](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) or application is using classpath specified in jar file or start-up script and to fix that you need to find the exact classpath used by your application. **Live example of reproducing and Fixing ClassNotFoundException in java**  
I think if we are able to reproduce and solve certain problem we become more comfortable dealing with that, that’s why here we will reproduce java.lang.ClassNotFoundException and solve it by following the concept we have discussed so far.   
  
1) Create a Class called **StockTrading.java**



2) create a Class called OnlineStockTranding.java and load the class StockTrading.java as Class.forName ("stocktrading");  


3) Compile both Java source file which will create two [class files](http://java67.blogspot.sg/2012/08/what-is-class-file-in-java-how-to-create-class.html) and run the program should run fine.  


4) Now just remove the .class file for stocktrading.java and [run the Java program](http://javarevisited.blogspot.sg/2011/11/run-java-program-from-command-prompt.html) and it will [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) java.lang.ClassNotFoundException in java.



**ClassFoundException vs NoClassDefFoundError vs UnSupportedClassVersionError**There are lots of exceptions in java but these three are the one who most haunted the java developer most mainly because these three are mostly related to environment issues and they all depends upon JVM and Classpath behaviour. Though they look similar there is slight [difference between ClassFoundException and NoClassDefFoundError](http://javarevisited.blogspot.com/2011/07/classnotfoundexception-vs.html) and UnSupportedClassVersionError and we will highlight those differences here for easy understanding and differentiating these three:  
  
1) ClassNotFoundException comes on Runtime when requested class is not available in classpath and mainly due to call to Class.forName () or Classloader.loadClass () or ClassLoader.findSystemClass ().  
  
2) NoClassDefFoundError comes when problematic class was present when your compiled your application but they are not in classpath while you running your program.  
  
3) [UnSupportedClassVersionError](http://javarevisited.blogspot.com/2011/07/javalangunsupportedclassversionerror.html) is easy to differentiate because it’s related to version of classpath and usually comes when you compile your code in higher Java version and try to run on lower java version. Can be resolved simply by using one java version for [compiling and running your application](http://javarevisited.blogspot.sg/2011/11/run-java-program-from-command-prompt.html).  
  
So that's all on ClassNotFoundException in java for now , please let me know if you have any tip or  any personal experience on solving java.lang.ClassNotFoundException in Java which you would like to share.  
  
**How to resolve java.lang.UnsupportedClassVersionError with example**

**java.lang.UnsupportedClassVersionError**  is a quite common error after [NoClassDefFoundError](http://javarevisited.blogspot.com/2011/06/noclassdeffounderror-exception-in.html) or ClassNotFoundException they all seems to related to class files but they all are different and there cause and resolution are different. In this java tutorial we will see what is **UnsupportedClassVersionError** in Java? Why UnsupportedClassVersionErrorcomes in Java? What is class file format and version numbers associated with it and finally how to resolve **UnsupportedClassVersionError** in Java.

This article is in continuation of debugging tutorials like [How to remote debug Java program in Eclipse](http://javarevisited.blogspot.com/2011/02/how-to-setup-remote-debugging-in.html) and [10 Java debugging tips in Eclipse](http://javarevisited.blogspot.com/2011/07/java-debugging-tutorial-example-tips.html). If you have not read those article you may find them useful.

**How to resolve UnsupportedClassVersionError in Java**

**What is UnSupportedClassVersionError in Java?**

Java.lang.UnsupportedClassVersionError is a subclass of java.lang.ClassFormatError. This is a kind of linking error which occurs during linking phase accordingly java.lang.ClassFormatError has also derived from java.lang.LinkageError. As the name suggests "UnSupportedClassVersionError" so it’s related to unsupported class version, now questions comes what is class version in Java? Well every source file is compiled into class file and each class file has two versions associated with it, major version and minor version. **Version of class file is represented as major\_version.minor\_version**. This version is used to determine *format of class file in Java*.

According to Java Virtual Machine specification, “A JVM implementation can support a class file format of version v if and only if v lies in some contiguous range Mi.0 v Mj.m. Only Sun can specify what range of versions a JVM implementation conforming to a certain release level of the Java platform may support.” For example: JDK 1.2 supports class file formats from version 45.0 to version 46.0 inclusive. So if a class file has version 48.0 it means that major version of class file is "48" and minor version is "0", which tells us that JDK 1.4 has been used to compile and generate that class file.  
  
**When UnSupportedClassVersionError in Java comes:**

So now we got the theory behind class file format and major and minor version of class file in Java. Now a million dollar question is when UnSupportedClassVersionError in Java does occur?  precise answer of this is "When JVM tries to load a class and found that class file version is not supported it throws *UnSupportedClassVersionError* and it generally occurs if a **higher JDK version  is used to compile the source file and  a lower JDK version is used to run the program**. for example if you compile your java source file in JDK 1.5 and you will try to run it on JDK 1.4 you will get error **"java.lang.UnsupportedClassVersionError: Bad version number in .class file** [at java.lang.ClassLoader.defineClass1(Native Method)]".  
  
 But its important to note is that vice-versa is not true "you can compile your program in J2SE 1.4 and run on J2SE 1.5 and you will not get any UnSupportedClassVersionError". When a higher JDK is used for compilation it creates class file with higher version and when a lower JDK is used to run the program it found that higher version of class file not supported at JVM level and results in java.lang.UnsupportedClassVersionError.

**How to fix UnSupportedClassVersionError**

Now we know the *root cause of UnSupportedClassVersionError* that we are using a lower JVM for running the program. But major problem is that stack trace of UnSupportedClassVersionError will not tell you for which class it’s coming. So if you are using multiple third party jars in your application you find that it comes at a particular part when JVM tries to load a class from a particular jar. anyway we all know that latest version of JDK is 1.6 so maximum version of class file could be generated by JDK 6, so by using JDK 6 we can solve UnSupportedClassVersionError, but many times its not easy to just move to higher JDK version. So I would suggest:

1) Find out due to which jar or class file this UnSupportedClassVersionError is coming?

2) Try to compile source code of that jar with the JDK version you are using to run your program, if source is available.

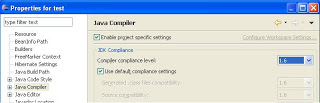
3) If you don't have source try to find the compatible version of that library.

4) Increase the JRE version you are using to run your program.

You can go by any approach to resolve UnSupportedClassVersionError based upon your need. Generally a higher JVM version is ok and does not cause any problem unless the class file format is quite old and no more supported by Sun in higher JVMs. The best way to deal with UnSupportedClassVersionError in Java is to use same version or JDK and JRE for compiling and running your program.

**Example of UnSupportedClassVersionError in Java**

You can easily reproduce UnSupportedClassVersionError by using javac of higher JDK and "java" from lower Java version. Let’s see some of examples of UnSupportedClassVersionError in Java:  
  
1) **java.lang.UnsupportedClassVersionError: EquityTradingManager (Unsupported major.minor version 49.0)**  
      at java.lang.ClassLoader.defineClass0(Native Method)  
      at java.lang.ClassLoader.defineClass(ClassLoader.java:539)  
==>Since we know that major version 49 is supported by JDK 1.5, so these will "java.lang.UnsupportedClassVersionError” will come if JVM used to run this program is lower than Java 1.5.  
  
2**) Java.lang.UnsupportedClassVersionError: Bad version number in .class file**  
  
3) **java.lang.unsupportedclassversionerror unsupported classversion 50.0**  
==> Compile in JDK 1.6 and running on lower version than Java 6.  
  
4) **java.lang.unsupportedclassversionerror unsupported classversion 49.0**  
==> compiled in Java 5 and running on lower JVM than JDK 5.  
  
5) **java.lang.unsupportedclassversionerror bad version number in eclipse.**

[](http://1.bp.blogspot.com/-w6p4e0W4E-Q/Tg6ZWAtmUZI/AAAAAAAAAMU/1LZYAVb3Yvg/s1600/unsupportedclassversionerror_in_java.GIF)

==> Most of us use eclipse for building and running project some of us also use ant for building project. In eclipse there is some setting related to java source version which if you got incorrect can result in "java.lang.unsupportedclassversionerror bad version number". so make sure you have correct configuration. For example if you compile with source compatible 1.6 you need JRE 6 to execute the program. To check the compiler setting in eclipse go to project  ==>Properties==>Java Compiler as shown in image

**Important point about UnSupportedClassVersionError in Java:**

1) If you encounter UnSupportedClassVersionError, check the JRE version you are using to run program and switch to higher version for quick solution.  
2) java.lang.UnsupportedClassVersionError is derived from java.lang.LinkageError, so it will not be detected in compile time and it will only come on runtime, precisely when JVM tries to load a class.  
3) Class file format which is identified using major version and minor version. Class file format is assigned when you compile source file and its depends on JDK version used to compile.  
4) Its always best practice to use same version of java for compilation and execution to avoid any chance of UnSupportedClassVersionError.  
5) *UnSupportedClassVersionError* is not related to [java classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html) , so don't confuse this with NoClassDefFoundError or ClassNotFoundException.

**Major Class Versions of Various JDK**

Following are the major version of *class file format* in standard JDK environment.  
JDK 1.1 = 45  
JDK 1.2 = 46  
JDK 1.3 = 47  
JDK 1.4 = 48  
JDK 1.5 = 49  
JDK 1.6 = 50  
You can also get version of "javac" (used for compilation) and version of "java" (used for execution) as below  
**C:\equity trading\stocks>javac -version**  
javac 1.6.0-beta2  
  
**C:\equity trading\stocks>java -version**  
java version "1.6.0-beta2"  
Java(TM) SE Runtime Environment (build 1.6.0-beta2-b86)  
Java HotSpot(TM) Client VM (build 1.6.0-beta2-b86, mixed mode, sharing)  
  
Now you can identify your JDK version based on class file format version whenever you see java.lang.UnsupportedClassVersionError :)  
  
So next time when you see **UnsupportedClassVersionError** don't be afraid and follow the best approach based upon your need.