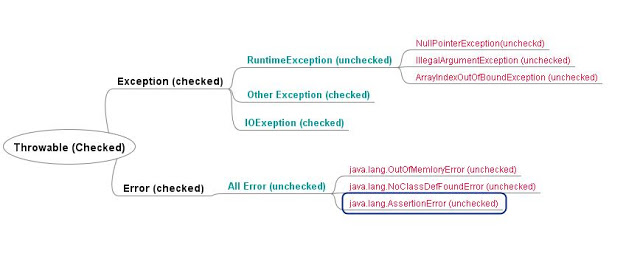
Difference between Checked vs Unchecked Exception in Java

What is Checked Exception in Java?

**Checked Exception in Java** is all those Exception which requires being catches and handled during [compile time](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html). If Compiler doesn’t see try or catch block handling a Checked Exception, it throws Compilation error. Now Which Exception is checked Exception and Why Checked Exception are introduced in first place? All the Exception which are direct sub Class of Exception but not inherit RuntimeException are Checked Exception.

Since a picture is worth 1000 words I have put together Exception hierarchy in mind map which clearly says which Exceptions are checked and which Exceptions are unchecked.

[](http://4.bp.blogspot.com/-ocSXMYac3Eo/T9IGCdsbHQI/AAAAAAAAAYM/EjDyZa9ykik/s1600/checked+exception+vs+unchecked+exception+in+java.JPG)

When to use Checked Exception in Java

Knowing **Checked Exception** is not that useful until you know how to use Checked Exception in Java. Java has often been criticized for its Checked Exception strategy, arguments given are that checked Exception adds lot of boiler plate code and makes whole class or function unreadable. Somewhat I agree with this and java also recognize this by introducing [improved Exception handling mechanism in Java7](http://javarevisited.blogspot.com/2011/07/jdk7-multi-cache-block-example-tutorial.html) but Checked Exception does have its real purpose. Following are some scenarios where I would prefer to use Checked Exception to ensure that [Code is Robust and stable](http://javarevisited.blogspot.com/2011/09/how-to-write-production-quality-code.html):

1) All Operation where chances of failure is more e.g. IO Operation, Database Access or Networking operation can be handled with Checked Exception.

2) When you know what to do (i.e. you have alternative) when an Exception occurs, may be as part of Business Process.

3) **Checked Exception** is a reminder by compiler to programmer to handle failure scenario.

Example of checked Exception in Java APIFollowing are some Examples of Checked Exception in Java library:

IOException

[SQLException](http://javarevisited.blogspot.com/2012/01/javasqlsqlexception-invalid-column.html)

DataAccessException

[ClassNotFoundException](http://javarevisited.blogspot.com/2011/08/classnotfoundexception-in-java-example.html)

InvocationTargetException

What is Unchecked Exception in Java?

**Unchecked Exception in Java** is those Exceptions whose handling is not verified during Compile time. Unchecked Exceptions mostly arise due to programming errors like accessing method of a null object, accessing element outside an array bonding or invoking method with illegal arguments. In Java, Unchecked Exception is direct sub Class of RuntimeException. *What is major benefit of Unchecked Exception* is that it doesn't reduce code readability and keeps the client code clean.

When to use UnCheckedException in Java

A good strategy of **Exception handling in Java** is wrapping a checked Exception into **UnCheckedException**. Since most of [Database operation](http://javarevisited.blogspot.com/2011/11/database-transaction-tutorial-example.html) throws SQLException but it’s not good to let SQLException propagate from your DAO layer to up higher on business layer and client code provide exception handling you can handle SQLException in DAO layer and you can wrap the cause in a RuntimeException to propagate through client code. Also as I said earlier unchecked exceptions are mostly programming errors and to catch them is real hard until you do a load test with all possible input and scenario. See [Core Java for Impatient](http://aax-us-east.amazon-adsystem.com/x/c/QgwvghZl6iak2gVlWCUc5KAAAAFdHfafBQEAAAFKAf-luA8/https:/assoc-redirect.amazon.com/g/r/http:/www.amazon.com/Core-Java-Impatient-Cay-Horstmann/dp/0321996321/ref=as_at?creativeASIN=0321996321&linkCode=w61&imprToken=zhT3AAPWIuF9K1frkdkLpg&slotNum=0&tag=javamysqlanta-20) for more details.

**Difference between Checked and Unchecked Exception in Java**

Now we have enough information to differentiate Checked Exception with Unchecked Exception:

1) Checked Exception is required to be handled by compile time while Unchecked Exception doesn't.

2) Checked Exception is direct sub-Class of Exception while Unchecked Exception are of RuntimeException.

3) CheckedException represent scenario with higher failure rate while UnCheckedException are mostly programming mistakes.

Example of unchecked Exception in Java API

Here are few **examples of Unchecked Exception in Java** library:

NullPointerException

ArrayIndexOutOfBound

IllegalArgumentException

IllegalStateException

**Summary:**

1. Both Checked and Unchecked Exception are handled using keyword try, catch and finally.

2. In terms of Functionality Checked and Unchecked Exception are same.

3. Checked Exception handling verified during compile time.

4. Unchecked Exception are mostly programming errors

5. JDK7 provides improved Exception handling code with c[atching multiple Exception in one catch block](http://javarevisited.blogspot.sg/2011/07/jdk7-multi-cache-block-example-tutorial.html) and reduce amount of boiler plate code required for exception handling in Java.

Java Exception and Error Interview Questions

Here is my list of frequently asked questions from Java Error and Exception topics in various programming interviews to Java and J2EE developers.  I have also shared my answers for these questions for quick revision, and provided source for more in depth understanding.  
  
I have tried to include questions of various difficulty level, including simplest of simple for freshers and some tricky questions for senior Java developers.  
  
If you think, there is a good question, which is not included in this list, please feel free to share it via comment. You can also share error handling questions asked to you on interviews or any question, for which you don’t know the answer.

**1) What is Exception in Java?**

This is always been first interview question on Exception and mostly asked on fresher level interviews. I haven't seen anybody asking about what is Exception in senior and experienced level interviews, but this is quite popular at entry level. In simple word Exception is Java’s way to convey both system and programming errors.  
  
In Java Exception feature is implemented by using class like Throwable, Exception, RuntimeException and keywords like throw, throws, try, catch and finally. All Exception are derived form Throwable class.  
  
Throwable further divides errors in too category one is java.lang.Exception and other is java.lang.Error.  java.lang.Error deals with system errors like java.lang.StackOverFlowError or [Java.lang.OutOfMemoryError](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html) while Exception is mostly used to deal with programming mistakes, non availability of requested resource etc.

**2) What is difference between Checked and Unchecked Exception in Java ?**

This is another popular Java Exception interview question appears in almost all level of Java interviews. Main difference between Checked and Unchecked Exception lies in there handling. Checked Exception requires to be handled at compile time using try, catch and finally keywords or else compiler will flag error. This is not a requirement for Unchecked Exceptions. Also all exceptions derived from java.lang.Exception classes are checked exception, exception those which extends RuntimeException, these are known as unchecked exception in Java. You can also check next article for [more differences between Checked and Unchecked Exception](http://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html).

**3) What is similarity between NullPointerException and ArrayIndexOutOfBoundException in Java?**

This is Java Exception interview question was not very popular, but appears in various fresher level interviews, to see whether candidate is familiar with concept of checked and unchecked exception or not. By the way answer of this interview question is both of them are example of unchecked exception and derived form RuntimeException. This question also opens door for difference of array in Java and C programming language, as arrays in C are unbounded and never throw ArrayIndexOutOfBoundException.

**4) What best practices you follow while doing Exception handling in Java ?**

This Exception interview question in Java is very popular while hiring senior java developer of Technical Lead. Since exception handling is crucial part of project design and good knowledge of this is desirable. There are lot of best practices, which can help to make your code robust and flexible at same time, here are few of them:

1) Returning boolean instead of returning null to avoid NullPointerException at callers end. Since NPE is most infamous of all Java exceptions, there are lot of techniques and [coding best practices to minimize NullPointerException](http://javarevisited.blogspot.com/2013/05/ava-tips-and-best-practices-to-avoid-nullpointerexception-program-application.html). You can check that link for some specific examples.

2) Non empty catch blocks. Empty catch blocks  are considered as one of the bad practices in Exception handling because they just ate Exception without any clue, at bare minimum print stack trace but you should do alternative operation which make sense or defined by requirements.

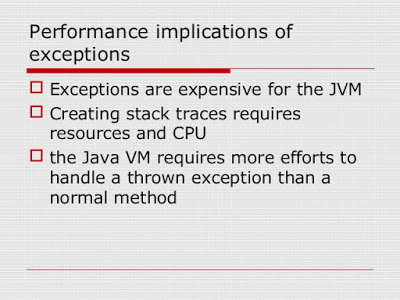
3) Prefer Unchecked exception over checked until you have a very good reason of not to do so. it improves readability of

code by removing boiler plate exception handling code

.

4) Never let your database Exception flowing till client error. since most of application deal with database and [SQLException](http://javarevisited.blogspot.com/2012/01/javasqlsqlexception-invalid-column.html) is a checked Exception in Java you should consider handling any database related errors in DAO layer of your application and only returning alternative value or something meaningful RuntimeException which client can understand and take action.

5) calling close() methods for connections, statements, and streams on finally block in Java.

[](http://2.bp.blogspot.com/-Xya2SK0Iva0/VXhOeWE7svI/AAAAAAAAC-8/FzhFhLejgQY/s1600/Java+Exception+Performance+impact.jpg)

I have already shared lot of these in my post [Top 10 Java exception handling best practices](http://javarevisited.blogspot.com/2013/03/0-exception-handling-best-practices-in-Java-Programming.html), you can also refer that for more knowledge on this topic.

**5) Why do you think Checked Exception exists in Java, since we can also convey error using RuntimeException ?**

This is a controversial question and you need to be careful while answering this interview question. Though they will definitely like to hear your opinion, what they are mostly interested in convincing reason. One of the reason I see is that its a design decision, which is influenced by experience in programming language prior to Java e.g. C++. Most of checked exceptions are in java.io package, which make sense because if you request any system resource and its not available, than a robust program must be able to handle that situation gracefully. By declaring IOException as checked Exception, Java ensures that your provide that gracefully exception handling. Another possible reason could be to ensuring that system resources like file descriptors, which are limited in numbers, should be released as soon as you are done with that using catch or finally block. Effective Java book from Joshua Bloch has couple of items in this topic, which is again worth reading.

**6) What is difference between throw and throws keyword in Java?**

One more Java Exception interview questions from beginners kitty. throw and throws keyword may look quite similar, especially if you are new to Java programming and haven't seen much of it. Though they are similar in terms that both are used in Exception handling, they are different on how and where they are used in code. throws keyword is used in method signature to declare which checked exception method can throw, you can also declare unchecked exception, but that is not mandatory by compiler. This signifies lot of things like method is not going to handle Exception instead its throwing it, if method throws checked Exception then caller should provide compile time exception handling etc. On the other hand throw keyword is actually used to throw any Exception. Syntactically you can throw any Throwable (i.e. Throwable or any class derived from Throwable) , throw  keyword transfers control of execution to caller so it can be used in place of return keyword. Most common example of using throw in place of return is throwing UnSupportedOperationException from an empty method as shown below :

private static void show() {

throw new UnsupportedOperationException("Not yet implemented");

}

See [this article](http://javarevisited.blogspot.com/2012/02/difference-between-throw-and-throws-in.html) for more differences between these two keywords in Java.

**7) What is Exception chaining in Java?**

Exception chaining is a popular exception handling concept in Java, where another exception is thrown in response of an exception and creating a chain of Exceptions. This technique mostly used to wrap a checked exception into an unchecked or RuntimeException. By the way if you are throwing new exception due to another exception then always include original exception so that handler code can access root cause by using methods like getCause() and initCause().

**8) Have you written your own custom Exception in Java? How do you do that?**

Ofcourse most of us has written custom or business Exceptions like AccountNotFoundExcepiton. Main purpose of asking this Java Exception interview question is to find out how you use this feature. This can be used for sophisticated and precise exception handling with tweak involved in whether you would choose a checked or unchecked exception. By creating a specific exception for specific case, you also gives lot of options to caller to deal with them elegantly. I always prefer to have a precise exception than a general exception. Though creating lots of specific exceptions quickly increase number of classes in your project, maintaining a practical balance between specific and general exceptions are key to success.

**9) What changes has been introduced in JDK7 related to Exception handling in Java ?**

A relatively new and recent Exception interview question in Java. JDK7 has introduced two major feature which is related to Error and Exception handling,  one is ability to handle [multiple exception in one catch block](http://javarevisited.blogspot.com/2011/07/jdk7-multi-cache-block-example-tutorial.html), popularly known as multi cache block and other is [ARM blocks in Java 7](http://javarevisited.blogspot.sg/2011/09/arm-automatic-resource-management-in.html) for automatic resource management, also known as try with resource. Both of these feature can certainly help to reduce boiler plate code required for handling checked exceptions in Java and significantly improves readability of code. Knowledge of this feature, not only helps to write better error and exception code in Java, but also helps to do well during interviews. I also recommend reading Java 7 Recipes book to get more insight on useful features introduced in Java 7, including these two.

**10) Have you faced OutOfMemoryError in Java? How did you solved that?**

This Java Error interview questions is mostly asked on senior level Java interviews and here interviewer is interested on your approach to tackle dangerous OutOfMemoryError. Admit it we always face this error no matter which kind of project you are working so if you say no it doesn't go very well with interviewer. I suggest even if you are not familiar or not faced it in reality but have 3 to 4 years of experience in Java, be prepare for it. At the same time, this is also a chance to impress interviewer by showing your advanced technical knowledge related to finding memory leaks, profiling and debugging. I have noticed that these skills almost always creates a positive impression. You can also see my post on [how to fix java.lang.OutOfMemoryError](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html) for more detail on this topic.

**11) Does code form finally executes if method returns before finally block or JVM exits ?**

This Java exception interview question can also be asked in code format, where given a code with System.exit() in try block and something in finally block. It’s worth knowing that, finally block in Java executes even when return keyword is used in try block. Only time they don’t execute is when you call JVM to exit by executing System.exit(0)from try block in Java.

**12) What is difference in final, finalize and finally keyword in Java?**

Another classic interview question in core Java, this was asked to one of my friend on his telephonic interview for core Java developer with Morgan Stanley. final and finally are keyword, while finalize is method. final keyword is very useful for creating ad [Immutable class in Java](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html) By making a class final, we prevent it from being extended, similarly by making a method final, we prevent it from being overridden,. On the other hand, finalize() method is called  by garbage collector, before that object is collected, but this is not guaranteed by Java specification. finally keyword is the only one which is related to error and exception handling and you should always have finally block in production code for closing connection and resources. See [here](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html) for more detailed answer of this question.

**13) What is wrong with following code :**

 public static void start() throws IOException, RuntimeException{

**throw** **new** RuntimeException("Not able to Start");

 }

 public static void main(String args[]) {

**try** {

          start();

    } **catch** (Exception ex) {

            ex**.**printStackTrace();

    } **catch** (RuntimeException re) {

            re**.**printStackTrace();

    }

 }

This code will throw compiler error on line where RuntimeException  variable “re” is written on catch block. since Exception is super class of RuntimeException, all RuntimeException thrown by start() method will be captured by first catch block and code will never reach second catch block and that's the reason compiler will flag error as  *“exception java.lang.RuntimeException has already been caught"*.

**14) What is wrong with following code in Java:**

public class *SuperClass* {

    public void **start**() throws IOException{

**throw** **new** IOException("Not able to open file");

    }

}

public class *SubClass* extends *SuperClass*{

    public void **start**() throws Exception{

**throw** **new** Exception("Not able to start");

    }

}

In this code compiler will complain on sub class where start() method gets overridden. As per [rules of method overriding in Java](http://java67.blogspot.com/2012/08/what-is-method-overriding-in-java-example-tutorial.html), an overridden method can not throw Checked Exception which is higher in hierarchy than original method. Since here start() is throwing IOException in super class, start() in sub class can only throw either IOException or any sub class of IOException but not super class of IOException e.g. Exception.

**15) What is wrong with following Java Exception code:**

public static void **start**(){

   System.**out**.println("Java Exception interivew question Answers for Programmers");

}

public static void **main**(String args[]) {

**try**{

      start();

   }**catch**(**IOException** ioe){

      ioe.printStackTrace();

   }

}

In this Java Exception example code, compiler will complain on line where we are handling IOException, since IOException is a checked Exception and start() method doesn't throw IOException, so compiler will flag error as "exception java.io.IOException is never thrown in body of corresponding try statement", but if you change IOException to Exception compiler error will disappear because Exception can be used to catch all RuntimeException which doesn't require declaration in throws clause. I like this little tricky Java Exception interview question because its not easy to figure out result by chaining IOException to Exception. You can also check Java Puzzlers by Joshua Bloch and Neil Gafter for some tricky questions based on Java Errors and Exceptions.

These are some of Java Error and Exception interview questions, I have mostly seen in both fresher and experienced level of Java interviews. There are a lot more questions on Exception which I haven't included and if you think you have a good question missed out than let me know and I will make effort to include it on this list of java exceptions question and answers. One last question of Java Exception I am leaving for you guys is "Why Java Exception considered to be better alternative of returning error codes" , let me know what is your thought on this list of Java Exception interview questions and answers.

**Recommended Books in this article**

* [Effective JavaBy Joshua Bloch](http://aax-us-east.amazon-adsystem.com/x/c/QrxrT3IGqQMu2Hjwwgvnx9QAAAFdHfQGjwEAAAFKASIaXxM/https:/assoc-redirect.amazon.com/g/r/http:/www.amazon.com/dp/0321356683/ref=as_at?creativeASIN=0321356683&linkCode=w61&imprToken=rza1bSQUTPSacsqigofoYg&slotNum=0&tag=javamysqlanta-20)
* [Java 7 Recipes: A Problem-Solution Approach](http://aax-us-east.amazon-adsystem.com/x/c/QrxrT3IGqQMu2Hjwwgvnx9QAAAFdHfQGjwEAAAFKASIaXxM/https:/assoc-redirect.amazon.com/g/r/http:/www.amazon.com/dp/1430240563/ref=as_at?creativeASIN=1430240563&linkCode=w61&imprToken=rza1bSQUTPSacsqigofoYg&slotNum=1&tag=javamysqlanta-20) By Josh Juneau, Carl Dea, Freddy Guime and John O'Conner
* [Java Puzzlersby Joshua a Bloch and Neil Gafter](http://aax-us-east.amazon-adsystem.com/x/c/QrxrT3IGqQMu2Hjwwgvnx9QAAAFdHfQGjwEAAAFKASIaXxM/https:/assoc-redirect.amazon.com/g/r/http:/www.amazon.com/dp/032133678X/ref=as_at?creativeASIN=032133678X&linkCode=w61&imprToken=rza1bSQUTPSacsqigofoYg&slotNum=2&tag=javamysqlanta-20)

### What are the rules we need to follow when overriding a method that throws an exception?

Several rules dictate how exceptions must be declared in the context of inheritance.

When the parent class method doesn’t throw any exceptions, the child class method can’t throw any checked exception, but it may throw any unchecked.

Here’s an example code to demonstrate this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Parent {      void doSomething() {          // ...      }  }    class Child extends Parent {      void doSomething() throws IllegalArgumentException {          // ...      }  } |

The next example will fail to compile since the overriding method throws a checked exception not declared in the overridden method:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Parent {      void doSomething() {          // ...      }  }    class Child extends Parent {      void doSomething() throws IOException {          // Compilation error      }  } |

When the parent class method throws one or more checked exceptions, the child class method can throw any unchecked exception; all, none or a subset of the declared checked exceptions, and even a greater number of these as long as they have the same scope or narrower.

Here’s an example code that successfully follows the previous rule:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | class Parent {      void doSomething() throws IOException, ParseException {          // ...      }        void doSomethingElse() throws IOException {          // ...      }  }    class Child extends Parent {      void doSomething() throws IOException {          // ...      }        void doSomethingElse() throws FileNotFoundException, EOFException {          // ...      }  } |

Note that both methods respect the rule. The first throws fewer exceptions than the overridden method, and the second, even though it throws more; they’re narrower in scope.

However, if we try to throw a checked exception that the parent class method doesn’t declare or we throw one with a broader scope; we’ll get a compilation error:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Parent {      void doSomething() throws FileNotFoundException {          // ...      }  }    class Child extends Parent {      void doSomething() throws IOException {          // Compilation error      }  } |

When the parent class method has a throws clause with an unchecked exception, the child class method can throw none or any number of unchecked exceptions, even though they are not related.

Here’s an example that honors the rule:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class Parent {      void doSomething() throws IllegalArgumentException {          // ...      }  }    class Child extends Parent {      void doSomething()        throws ArithmeticException, BufferOverflowException {          // ...      }  } |

**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 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. 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 comes in following cases:

1) When we try to load a class 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.

[What is ClassNotFoundException in Java - Fix Solution](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)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 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.

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 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 aClassNotFoundException.

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, 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 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**  
  
public class **StockTrading**{  
   public String getDescription(){  
   return "StockTrading";  
  }  
}   
  
2) **create a Class called OnlineStockTranding.java** and load the class StockTrading.java as Class.forName ("stocktrading");  
  
public class **OnlineStockTrading**{  
   public static void main(String args[]) throws **ClassNotFoundException**{  
      Class.forName("StockTrading");  
      System.out.println("StockTrading class successfully loaded");  
   }  
}  
  
3) Compile both Java source file which will create two class files and run the program should run fine.  
  
javin@trading~/java: **javac \*.java**  
  
javin@trading ~/java: **ls –lrt**-rw-r--r-- 1 javin None  90 Aug 21 09:27 StockTrading.java  
-rw-r--r-- 1 javin None 208 Aug 21 09:28 OnlineStockTrading.java  
-rwxr-xr-x 1 javin None 282 Aug 21 09:28 StockTrading.class  
-rwxr-xr-x 1 javin None 638 Aug 21 09:28 OnlineStockTrading.class  
  
javin@trading ~/java:$ **java OnlineStockTrading**  
StockTrading class successfully loaded  
  
  
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 java.lang.ClassNotFoundException in java.  
  
javin@trading ~/java: **rm StockTrading.class**  
  
javin@trading ~/java: **java OnlineStockTrading  
Exception in thread "main" java.lang.ClassNotFoundException: StockTrading**  
at java.net.URLClassLoader$1.run(Unknown Source)  
at java.security.AccessController.doPrivileged(Native Method)  
at java.net.URLClassLoader.findClass(Unknown Source)  
at java.lang.ClassLoader.loadClass(Unknown Source) **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.  
  
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.

**Some more Interesting tutorials:**

[How to use Comparator and Comparable in Java? With example](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html)  
[How Classpath work in Java? How to set classpath in Unix Linux](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html)  
[How to override equals method in Java](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html)  
[How to implement Thread in Java ?Example of Runnable interface](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html)  
[Difference between ConcurrentHashMap and Collections.synchronizedMap and Hashtable in Java](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html)  
[How to create update or remove symbolic or soft link in Unix](http://javarevisited.blogspot.com/2011/04/symbolic-link-or-symlink-in-unix-linux.html)  
[What is Abstraction in java](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html)?  
[What is the difference between Enumeration and Iterator?](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html)