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

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 two 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 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.

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. 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 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.

I have already shared lot of these in my post Top 10 Java exception handling best practices, 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 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, popularly known as multi cache block and other is ARM blocks in Java 7 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 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 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 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 “exceptionjava.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, 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.