|  |
| --- |
| **Exceptions Interview Questions**  **Q0) Difference between Errors & Exceptions?**  **Ans:**  **Errors :>**   1. Errors are thrown by java runtime system and indicate some irrecoverable conditions that occurred during program execution. 2. Once thrown by the system, application cannot recover from it and will come to halt. 3. Errors need not to be handled (Cached).   **e.g.:** java.lang.OutOfMemoryError, java.lang.StackOverflowError, java.lang.NoClassDefFoundError etc.  **Exceptions :>**   1. Exceptions can be handled to prevent the programs to be terminated automatically. 2. Exceptions can be handled by using try, catch blocks.   **Class hierarchy of Exceptions:**  http://www.javamex.com/tutorials/exceptions/ExceptionHierarchy.png Figure 1: The base of the Java exception hierarchy. Types in red, and their subclasses, are **unchecked**.  **Checked vs. unchecked exceptions**  We mentioned above that certain exceptions are generally indicative of a **programming error** rather than errors that we'd expect to occur in the normal course of events. (Whereas, it is reasonable for the programmer to expect that a networking or file system error will occur from time to time.) We also mentioned that subclasses of **Error** are not necessarily programming errors, but still errors that we wouldn't really expect to occur in the normal course of events.  A feature built into the Java language is that **Error**s and **RuntimeException**s (and their subclasses– marked in red in Figure 1) are what are called **unchecked exceptions**:   * unchecked exceptions can be thrown "at any time"; * **methods don't explicitly have to declare** that they can throw an unchecked exception; * Callers **don't have to handle them** explicitly. |
| **Q1) What is an Exception?**  Ans) The exception is said to be thrown whenever an exceptional event occurs in java which signals that something is not correct with the code written and may give unexpected result. An exceptional event is a occurrence of condition which alters the normal program flow. Exceptional handler is the code that does something about the exception. |
| **Q2) Exceptions are defined in which java package?**  Ans)All the exceptions are subclasses of java.lang.Exception |
| **Q3) How are the exceptions handled in java?**  Ans) When an exception occurs the execution of the program is transferred to an appropriate exception handler. The try-catch-finally block is used to handle the exception. The code in which the exception may occur is enclosed in a try block, also called as a guarded region. The catch clause matches a specific exception to a block of code which handles that exception. And the clean up code which needs to be executed no matter the exception occurs or not is put inside the finally block |
| **Q4) Explain the exception hierarchy in java.**  Ans) The hierarchy is as follows:  Throwable is a parent class off all Exception classes. They are two types of Exceptions: Checked exceptions and UncheckedExceptions. Both type of exceptions extends Exception class. |
| **Q5) What is Runtime Exception or unchecked exception?**  Ans) Runtime exceptions represent problems that are the result of a programming problem. Such problems include arithmetic exceptions, such as dividing by zero; pointer exceptions, such as trying to access an object through a null reference; and indexing exceptions, such as attempting to access an array element through an index that is too large or too small. Runtime exceptions need not be explicitly caught in try catch block as it can occur anywhere in a program, and in a typical one they can be very numerous. Having to add runtime exceptions in every method declaration would reduce a program's clarity. Thus, the compiler does not require that you catch or specify runtime exceptions (although you can). The solution to rectify is to correct the programming logic where the exception has occurred or provide a check. |
| **Q6) What is checked exception?**  Ans) Checked exception are the exceptions which forces the programmer to catch them explicitly in try-catch block. It is a subClass of Exception. Example: IOException. |
| **Q7) What is difference between Error and Exception?**  Ans) An error is an irrecoverable condition occurring at runtime. Such as OutOfMemoryError. These JVM errors and you cannot repair them at runtime. Though error can be caught in catch block but the execution of application will come to a halt and is not recoverable.  While exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.) |
| **Q8) What is difference between ClassNotFoundException and NoClassDefFoundError?**  Ans) A ClassNotFoundException is thrown when the reported class is not found by the ClassLoader in the CLASSPATH. It could also mean that the class in question is trying to be loaded from another class which was loaded in a parent classloader and hence the class from the child classloader is not visible.  Consider if NoClassDefFoundError occurs which is something like  java.lang.NoClassDefFoundError  src/com/TestClass  does not mean that the TestClass class is not in the CLASSPATH. It means that the class TestClass was found by the ClassLoader however when trying to load the class, it ran into an error reading the class definition. This typically happens when the class in question has static blocks or members which use a Class that's not found by the ClassLoader. So to find the culprit, view the source of the class in question (TestClass in this case) and look for code using static blocks or static members. |
| **Q9) What is throw keyword?**  Ans) Throw keyword is used to throw the exception manually. It is mainly used when the program fails to satisfy the given condition and it wants to warn the application.The exception thrown should be subclass of Throwable.    public void parent(){   try{    child();   }catch(MyCustomException e){ }   }     public void child{   String iAmMandatory=null;    if(iAmMandatory == null){     throw (new MyCustomException("Throwing exception using throw keyword");    }   } |
| **Q10) What is use of throws keyword?**  Ans) If the function is not capable of handling the exception then it can ask the calling method to handle it by simply putting the **throws clause**at the function declaration.    public void parent(){   try{    child();   }catch(MyCustomException e){ }   }     public void child throws MyCustomException{    //put some logic so that the exception occurs.   } |
| **Q11) What are the possible combination to write try, catch finally block?**  Ans)  1) try{  //lines of code that may throw an exception  }catch(Exception e){  //lines of code to handle the exception thrown in try block  }finally{  //the clean code which is executed always no matter the exception occurs or not.  }  2 try{  }finally{}  3 try{  }catch(Exception e){  //lines of code to handle the exception thrown in try block  }  The catch blocks must always follow the try block. If there are more than one catch blocks they all must follow each other without any block in between. The finally block must follow the catch block if one is present or if the catch block is absent the finally block must follow the try block. |
| **Q12) How to create custom Exception?**  Ans) To create your own exception extend the Exception class or any of its subclasses.  e.g.  1 class New1Exception extends Exception { } // this will create Checked Exception  2 class NewException extends IOExcpetion { } // this will create Checked exception  3 class NewException extends NullPonterExcpetion { } // this will create UnCheckedException |
| **Q13) When to make a custom checked Exception or custom unchecked Exception?**  Ans) If an application can reasonably be expected to recover from an exception, make it a checked exception. If an application cannot do anything to recover from the exception, make it an unchecked exception. |
| **Q14)What is StackOverflowError?**  Ans) The StackOverFlowError is an Error Object thorwn by the Runtime System when it Encounters that your application/code has ran out of the memory. It may occur in case of recursive methods or a large amount of data is fetched from the server and stored in some object. This error is generated by JVM.  e.g. void swap(){  swap();  } |
| **Q15) Why did the designers decide to force a method to specify all uncaught checked exceptions that can be thrown within its scope?**  Ans) Any Exception that can be thrown by a method is part of the method's public programming interface. Those who call a method must know about the exceptions that a method can throw so that they can decide what to do about them. These exceptions are as much a part of that method's programming interface as its parameters and return value. |
| **16) Once the control switches to the catch block does it return back to the try block to execute the balance code?**  Ans) No. Once the control jumps to the catch block it never returns to the try block but it goes to finally block(if present). |
| **Q17) Where is the clean up code like release of resources is put in try-catch-finally block and why?**  Ans) The code is put in a finally block because irrespective of try or catch block execution the control will flow to finally block. Typically finally block contains release of connections, closing of result set etc. |
| **Q18) Is it valid to have a try block without catch or finally?**  Ans) NO. This will result in a compilation error. The try block must be followed by a catch or a finally block. It is legal to omit the either catch or the finally block but not both.  e.g. The following code is illegal.  try{ int i =0;  } int a = 2; System.out.println(“a = “+a); |
| **Q19) Is it valid to place some code in between try the catch/finally block that follows it?**  Ans) No. There should not be any line of code present between the try and the catch/finally block. e.g. The following code is wrong.  try{} String str = “ABC”; System.out.println(“str = “+str); catch(Exception e){} |
| **Q20) What happens if the exception is never caught and throws down the method stack?**  Ans) If the exception is not caught by any of the method in the method’s stack till you get to the main() method, the main method throws that exception and the JVM halts its execution. |
| **Q21) How do you get the descriptive information about the Exception occurred during the program execution?**  Ans) All the exceptions inherit a method printStackTrace() from the Throwable class. This method prints the stack trace from where the exception occurred. It prints the most recently entered method first and continues down, printing the name of each method as it works its way down the call stack from the top. |
| **Q22) Can you catch more than one exceptions in a single catch block?**  Ans)Yes. If the exception class specified in the catch clause has subclasses, any exception object that is a subclass of the specified Exception class will be caught by that single catch block.  E.g..  try {  // Some code here that can throw an IOException  }  catch (IOException e) {  e.printStackTrace();  }  The catch block above will catch IOException and all its subclasses e.g. FileNotFoundException etc. |
| **Q23)Why is not considered as a good practice to write a single catchall handler to catch all the exceptions?**  Ans) You can write a single catch block to handle all the exceptions thrown during the program execution as follows :  try {  // code that can throw exception of any possible type  }catch (Exception e) {  e.printStackTrace();  }  If you use the Superclass Exception in the catch block then you will not get the valuable information about each of the exception thrown during the execution, though you can find out the class of the exception occurred. Also it will reduce the readability of the code as the programmer will not understand what is the exact reason for putting the try-catch block. |
| **Q24) What is exception matching?**  Ans) Exception matching is the process by which the the jvm finds out the matching catch block for the exception thrown from the list of catch blocks. When an exception is thrown, Java will try to find by looking at the available catch clauses in the top down manner. If it doesn't find one, it will search for a handler for a supertype of the exception. If it does not find a catch clause that matches a supertype for the exception, then the exception is propagated down the call stack. This process is called exception matching. |
| **Q25) What happens if the handlers for the most specific exceptions is placed above the more general exceptions handler?**  Ans) Compilation fails. The catch block for handling the most specific exceptions must always be placed above the catch block written to handle the more general exceptions.  e.g. The code below will not compile.  1 try {  // code that can throw IOException or its subtypes  } catch (IOException e) {  // handles IOExceptions and its subtypes  } catch (FileNotFoundException ex) {  // handle FileNotFoundException only  }  The code below will compile successfully :-  try {  // code that can throw IOException or its subtypes  } catch (FileNotFoundException ex) {  // handles IOExceptions and its subtypes  } catch (IOException e){  // handle FileNotFoundException only  } |
| **Q26) Does the order of the catch blocks matter if the Exceptions caught by them are not subtype or supertype of each other?**  Ans) No. If the exceptions are siblings in the Exception class’s hierarchy i.e. If one Exception class is not a subtype or supertype of the other, then the order in which their handlers(catch clauses) are placed does not matter. |
| **Q27) What happens if a method does not throw an checked Exception directly but calls a method that does? What does 'Ducking' the exception mean?**  Ans) If a method does not throw an checked Exception directly but calls a method that throws an exception then the calling method must handle the throw exception or declare the exception in its throws clause. If the calling method does not handle and declares the exception, the exceptions is passed to the next method in the method stack. This is called as ducking the exception down the method stack.  e.g. The code below will not compile as the getCar() method has not declared the CarNotFoundException which is thrown by the getColor () method.  void getCar() {  getColor();  }  void getColor () {  throw new CarNotFoundException();  }  Fix for the above code is  void getCar() throws CarNotFoundException {  getColor();  }  void getColor () {  throw new CarNotFoundException();  } |
| **Q28) Is an empty catch block legal?**  Ans) Yes you can leave the catch block without writing any actual code to handle the exception caught.  e.g. The code below is legal but not appropriate, as in this case you will nt get any information about the exception thrown.  try{  //code that may throw the FileNotFoundException  }catch(FileNotFound eFnf){  //no code to handle the FileNotFound exception  } |
| **Q29)Can a catch block throw the exception caught by itself?**  Ans) Yes. This is called rethrowing of the exception by catch block.  e.g. the catch block below catches the FileNotFound exception and rethrows it again.  void checkEx() throws FileNotFoundException {  try{  //code that may throw the FileNotFoundException  }catch(FileNotFound eFnf){  throw FileNotFound();  }  } |

**Q30) What will happen if a return from try, catch and finally block is done? And what will happen if finally block will also throw exception?**

**package** exceptionHandling;

**public** **class** ExceptionTestWithTryCatchFinallyAndReturn {

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

System.*out*.println("Returns: "+*returnNumber*());

*throwInFinally*();

}

**public** **static** **int** returnNumber() {

**try** {

**return** 1;

} **catch** (Exception e) {

System.*out*.println("Exception thrown: " + e);

**return** 2;

} **finally** {//Finally will override the return value of try.Because after return finaly will always be executed before JVM Stops the app.

//Warning: finally block does not complete normally.

**return** 3;

}

}

**public** **static** **void** throwInFinally() {

**try** {

System.*out*.println(1/0);

} **catch** (Exception e) {

System.*out*.println("Exception thrown: " + e);

} **finally** {

**throw** **new** MyOwnException("Exception occoured due to invalid inputs!");

/\*

\* Dispatch an uncaught exception to the handler. This method is intended to be called only by the JVM. when exception is not catched or rethrown from finally,

private void dispatchUncaughtException(Throwable e) {

getUncaughtExceptionHandler().uncaughtException(this, e);

}

\*/

}

}

}

**Output>>**

**Returns: 3**

**Exception thrown:** java.lang.ArithmeticException: / by zero

Exception in thread "main" exceptionHandling.MyOwnException: Exception occoured due to invalid inputs!

at exceptionHandling.ExceptionTestWithTryCatchFinallyAndReturn.throwInFinally(ExceptionTestWithTryCatchFinallyAndReturn.java:29)

at exceptionHandling.ExceptionTestWithTryCatchFinallyAndReturn.main(ExceptionTestWithTryCatchFinallyAndReturn.java:8)

**Q31) what is the bottom up & top down hierarchy flow when exception is cached in cached block during propagation the exception?**

**E.g. :**

**public** **class** ExceptionTestWithTryCatchFinallyAndReturn {

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

exceptionPropagation();

}

**public** **static** **void** exceptionPropagation () {

**try** {

System.*out*.println(1/0);

} **catch** (Exception e) {

System.*out*.println("Exception thrown: " + e);

}

}

}

}

**Output>>**

**Exception thrown:** java.lang.ArithmeticException: / by zero

**Bottom up hierarchy when exception is cached:**

ArithematicException(message)

RunTimeException(message)

Exception(message)

Throwable(message)

**new** **Throwable** object is constructed and registered with Object class.

Throwable cause is set as java.lang.ArithmeticException.

Stack trace is filled and **detailMessage** is populated.

**Top down hierarchy when exception is propagated:**

Throwable

Exception

RunTimeException

ArithematicException

Stack trace is printed.

**Q32) Difference between OutOfMemoryError & StackOverflowError?**

**Ans:**

**OutOfMemoryError>** This error occurs when there is no extra space available with JVM to create objects, RAM is occupied by the existing objects and there is no extra space on RAM.

**StackOverflowError >** It occurs when you invoke a method in cyclic manner,in this case stack memory gets overflow and hence this error occurs and program stops abruptly.

E.g.:

**package** exceptionHandling;

**public** **class** Stackoverflowerror {

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

*callme*();

}

**public** **static** **void** callme(){

*callme*();

}

}

**Output>>**

Exception in thread "main" java.lang.StackOverflowError

at exceptionHandling.Stackoverflowerror.callme(Stackoverflowerror.java:10)

at exceptionHandling.Stackoverflowerror.callme(Stackoverflowerror.java:10)

at exceptionHandling.Stackoverflowerror.callme(Stackoverflowerror.java:10)

at exceptionHandling.Stackoverflowerror.callme(Stackoverflowerror.java:10)