Java Exception Interview Questions and Answers

Java provides a robust and object-oriented approach to handle exception scenarios known as **Java Exception Handling**.

Sometime back I wrote a long post on Java Exception Handling and today I am listing some important **Java Exceptions Questions with Answers** to help you in interviews.

1. What is Exception in Java?

Exception is an error event that can happen during the execution of a program and disrupts it's normal flow. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure etc.

Whenever any error occurs while executing a java statement, an exception object is created and then **JRE** tries to find exception handler to handle the exception. If suitable exception handler is found then the exception object is passed to the handler code to process the exception, known as **catching the exception**. If no handler is found then application throws the exception to runtime environment and JRE terminates the program.

Java Exception handling framework is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

2. What are the Exception Handling Keywords in Java?

There are four keywords used in java exception handling.

A. **throw**: Sometimes we explicitly want to create exception object and then throw it to halt the normal processing of the program. **throw** keyword is used to throw exception to the runtime to handle it.
B. **throws**: When we are throwing any checked exception in a method and not handling it, then we need to use throws keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it's caller method using `throws` keyword. We can provide multiple exceptions in the throws clause and it can be used with **main()** method also.
C. **try-catch**: We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
D. **finally**: finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurrs or not.

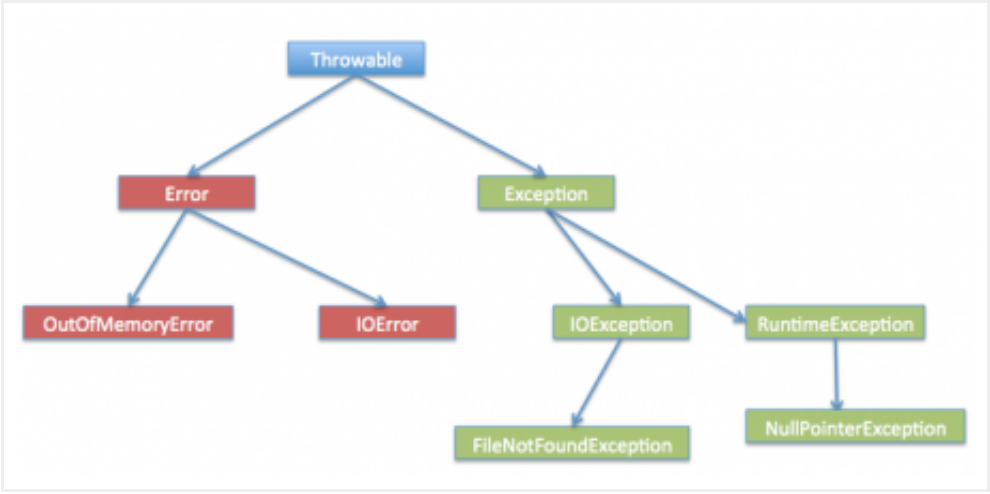
3. Explain Java Exception Hierarchy?

Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. `Throwable` is the parent class of Java Exceptions Hierarchy and it has two child objects – `Error` and `Exception`. Exceptions are further divided into checked exceptions and runtime exception.

Errors are exceptional scenarios that are out of scope of application and it's not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error.

Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. `Exception` is the parent class of all Checked Exceptions.

Runtime Exceptions are caused by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw `ArrayIndexOutOfBoundException` at runtime. `RuntimeException` is the parent class of all runtime exceptions.



4. What are important methods of Java Exception Class?

Exception and all of it's subclasses doesn't provide any specific methods and all of the methods are defined in the base class Throwable.

A. **String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it's constructor.
B. **String getLocalizedMessage()** – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use `getMessage()` method to return the exception message.
C. **synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
D. **String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
E. **void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

5. Explain Java 7 ARM Feature and multi-catch block?

If you are catching a lot of exceptions in a single try block, you will notice that catch block code looks very ugly and mostly consists of redundant code to log the error, keeping this in mind Java 7 one of the feature was multi-catch block where we can catch multiple exceptions in a single catch block. The catch block with this feature looks like below:

```
catch(IOException | SQLException | Exception ex){
     logger.error(ex);
     throw new MyException(ex.getMessage());
}
```

Most of the time, we use finally block just to close the resources and sometimes we forget to close them and get runtime exceptions when the resources are exhausted. These exceptions are hard to debug and we might need to look into each place where we are using that type of resource to make sure we are closing it. So java 7 one of the improvement was **try-with-**

resources where we can create a resource in the try statement itself and use it inside the try-catch block. When the execution comes out of try-catch block, runtime environment automatically close these resources. Sample of try-catch block with this improvement is:

```
try (MyResource mr = new MyResource()) {
            System.out.println("MyResource created in try-with-resources");
        } catch (Exception e) {
            e.printStackTrace();
        }
```

Read more about this at Java 7 ARM.

6. What is difference between Checked and Unchecked Exception in Java?

A. Checked Exceptions should be handled in the code using try-catch block or else main() method should use throws keyword to let JRE know about these exception that might be thrown from the program. Unchecked Exceptions are not required to be handled in the program or to mention them in throws clause.
B. `Exception` is the super class of all checked exceptions whereas `RuntimeException` is the super class of all unchecked exceptions.
C. Checked exceptions are error scenarios that are not caused by program, for example FileNotFoundException in reading a file that is not present, whereas Unchecked exceptions are mostly caused by poor programming, for example NullPointerException when invoking a method on an object reference without making sure that it's not null.

7. What is difference between throw and throws keyword in Java?

throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of program and handing over the exception object to runtime to handle it.

8. How to write custom exception in Java?

We can extend `Exception` class or any of it's subclasses to create our custom exception class. The custom exception class can have it's own variables and methods that we can use to pass error codes or other exception related information to the exception handler.

A simple example of custom exception is shown below.

MyException.java

```
package com.journaldev.exceptions;

import java.io.IOException;

public class MyException extends IOException {
```

```
6
7       private static final long serialVersionUID = 4664456874499611218L;
8
9       private String errorCode="Unknown_Exception";
10
11      public MyException(String message, String errorCode){
12          super(message);
13          this.errorCode=errorCode;
14      }
15
16      public String getErrorCode(){
17          return this.errorCode;
18      }
19
20
21  }
```

9. What is OutOfMemoryError in Java?

OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError and it's thrown by JVM when it ran out of heap memory. We can fix this error by providing more memory to run the java application through java options.

```
$>java MyProgram -Xms1024m -Xmx1024m -XX:PermSize=64M -XX:MaxPermSize=256m
```

10. What are different scenarios causing "Exception in thread main"?

Some of the common main thread exception scenarios are:

- **Exception in thread main java.lang.UnsupportedClassVersionError**: This exception comes when your java class is compiled from another JDK version and you are trying to run it from another java version.
- **Exception in thread main java.lang.NoClassDefFoundError**: There are two variants of this exception. The first one is where you provide the class full name with .class extension. The second scenario is when Class is not found.
- **Exception in thread main java.lang.NoSuchMethodError: main**: This exception comes when you are trying to run a class that doesn't have main method.
- **Exception in thread "main" java.lang.ArithmeticException**: Whenever any exception is thrown from main method, it prints the exception is console. The first part explains that exception is thrown from main method, second part prints the exception class name and then after a colon, it prints the exception message.

Read more about these at Java Exception in Thread main.

11. What is difference between final, finally and finalize in Java?

final and finally are keywords in java whereas finalize is a method.

final keyword can be used with class variables so that they can't be reassigned, with class to avoid extending by classes and with methods to avoid overriding by subclasses, finally keyword is used with try-catch block to provide statements that will always gets executed even if some exception arises, usually finally is used to close resources. finalize() method is executed by Garbage Collector before the object is destroyed, it's great way to make sure all the global resources are closed.

Out of the three, only finally is related to java exception handling.

12. What happens when exception is thrown by main method?

When exception is thrown by main() method, Java Runtime terminates the program and print the exception message and stack trace in system console.

13. Can we have an empty catch block?

We can have an empty catch block but it's the example of worst programming. We should never have empty catch block because if the exception is caught by that block, we will have no information about the exception and it wil be a nightmare to debug it. There should be at least a logging statement to log the exception details in console or log files.

14. Provide some Java Exception Handling Best Practices?

Some of the best practices related to Java Exception Handling are:

- Use Specific Exceptions for ease of debugging.
- Throw Exceptions Early (Fail-Fast) in the program.
- Catch Exceptions late in the program, let the caller handle the exception.
- Use Java 7 ARM feature to make sure resources are closed or use finally block to close them properly.
- Always log exception messages for debugging purposes.
- Use multi-catch block for cleaner close.
- Use custom exceptions to throw single type of exception from your application API.
- Follow naming convention, always end with Exception.
- Document the Exceptions Thrown by a method using @throws in javadoc.
- Exceptions are costly, so throw it only when it makes sense. Else you can catch them and provide null or empty response.

Read more about them in detail at Java Exception Handling Best Practices.

15. What is the problem with below programs and how do we fix it?

In this section, we will look into some programming questions related to java exceptions.

A. **What is the problem with below program?**

```
1  package com.journaldev.exceptions;
2
3  import java.io.FileNotFoundException;
4  import java.io.IOException;
5
6  public class TestException {
7
8      public static void main(String[] args) {
9          try {
10             testExceptions();
11         } catch (FileNotFoundException | IOException e) {
12             e.printStackTrace();
13         }
14     }
15
16
17
18     public static void testExceptions() throws IOException, FileNotFou
19
20     }
21 }
```

Above program won't compile and you will get error message as "The exception FileNotFoundException is already caught by the alternative IOException". This is because FileNotFoundException is subclass of IOException, there are two ways to solve this problem.

First way is to use single catch block for both the exceptions.

```
1  try {
2      testExceptions();
3  }catch(FileNotFoundException e){
4      e.printStackTrace();
5  }catch (IOException  e) {
6      e.printStackTrace();
7  }
```

Another way is to remove the FileNotFoundException from multi-catch block.

```
1  try {
2      testExceptions();
3  }catch (IOException  e) {
4      e.printStackTrace();
5  }
```

You can chose any of these approach based on your catch block code.

B. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.FileNotFoundException;
import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException1 {

    public static void main(String[] args) {
            try {
                go();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (JAXBException e) {
                e.printStackTrace();
            }
    }

    public static void go() throws IOException, JAXBException, FileNot

    }
}
```

The program won't compile because FileNotFoundException is subclass of IOException, so the catch block of FileNotFoundException is unreachable and you will get error message as "Unreachable catch block for FileNotFoundException. It is already handled by the catch block for IOException".

You need to fix the catch block order to solve this issue.

```
try {
    go();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (JAXBException e) {
    e.printStackTrace();
}
```

Notice that JAXBException is not related to IOException or FileNotFoundException and can be put anywhere in above catch block hierarchy.

C. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException2 {

    public static void main(String[] args) {
        try {
            foo();
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }catch(JAXBException e){
                e.printStackTrace();
            }catch(NullPointerException e){
                e.printStackTrace();
            }catch(Exception e){
                e.printStackTrace();
            }
        }

        public static void foo() throws IOException{

        }
    }
```

The program won't compile because JAXBException is a checked exception and foo() method should throw this exception to catch in the calling method. You will get error message as "Unreachable catch block for JAXBException. This exception is never thrown from the try statement body".

To solve this issue, you will have to remove the catch block of JAXBException.

Notice that catching NullPointerException is valid because it's an unchecked exception.

D. **What is the problem with below program?**

```
package com.journaldev.exceptions;

public class TestException3 {

    public static void main(String[] args) {
        try{
        bar();
        }catch(NullPointerException e){
            e.printStackTrace();
        }catch(Exception e){
            e.printStackTrace();
        }

        foo();
    }

    public static void bar(){

    }

    public static void foo() throws NullPointerException{

    }
}
```

This is a trick question, there is no problem with the code and it will compile successfully. We can always catch Exception or any unchecked exception even if it's not in the throws clause of the method.

Similarly if a method (foo) declares unchecked exception in throws clause, it is not

mandatory to handle that in the program.

E. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.IOException;

public class TestException4 {

    public void start() throws IOException{
    }

    public void foo() throws NullPointerException{

    }
}

class TestException5 extends TestException4{

    public void start() throws Exception{
    }

    public void foo() throws RuntimeException{

    }
}
```

The above program won't compile because start() method signature is not same in subclass. To fix this issue, we can either change the method singnature in subclass to be exact same as superclass or we can remove throws clause from subclass method as shown below.

```
@Override
    public void start(){
    }
```

F. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException6 {

    public static void main(String[] args) {
        try {
            foo();
        } catch (IOException | JAXBException e) {
            e = new Exception("");
            e.printStackTrace();
        }catch(Exception e){
            e = new Exception("");
            e.printStackTrace();
        }
    }

    public static void foo() throws IOException, JAXBException{

```

```
23          }
24      }
```

The above program won't compile because exception object in multi-catch block is final and we can't change it's value. You will get compile time error as "The parameter e of a multi-catch block cannot be assigned".

We have to remove the assignment of "e" to new exception object to solve this error.

Read more at Java 7 multi-catch block.

Thats all for java exception interview questions, I hope you will like it. I will be adding more to the list in future, make sure you bookmark it for future use.