# **Exception Handling**

**What is an exception in java?**

An exception is a run time error. In java exception is an object. Exceptions are created when an abnormal situations are arised in our program. Exceptions can be created by JVM or by our application code. All Exception classes are defined in java.lang.

**State some situations where exceptions may arise in java?**

1) Accessing an element that does not exist in array.

2) Dividing an integer by zero (Arithmeticexception)

2) Invalid conversion of number to string and string to number. (NumberFormatException)

3) Invalid casting of class (Class cast Exception)

4) Trying to create object for interface or abstract class (Instantiation Exception)

**What is Exception handling in java?**

Exception handling is a mechanism what to do when some abnormal situation arises in program. When an exception is raised in program it leads to termination of program when it is not handled properly. The significance of exception handling comes here in order not to terminate a program abruptly and to continue with the rest of program normally. This can be done with help of Exception handling.

**In how many ways we can do exception handling in java?**

We can handle exceptions in either of the two ways:

1) By specifying try catch block where we can catch the exception.

2) Declaring a method with throws clause.

**List out five keywords related to Exception handling?**

1) Try

2) Catch

3) throw

4) throws

5) finally

**Explain try and catch keywords in java?**

In try block we define all exception causing code. In java try and catch forms a unit. A catch block catches the exception thrown by preceding try block. Catch block cannot catch an exception thrown by another try block. If there is no exception causing code in our program or exception is not raised in our code jvm ignores the try catch block.

Syntax:

try {

}

Catch(Exception e) {

}

**Can we have try block without catch block?**

Each try block requires at least one catch block or finally block. A try block without catch or finally will result in compiler error. We can skip either of catch or finally block but not both.

**Can we have multiple catch block for a try block?**

In some cases our code may throw more than one exception. In such case we can specify two or more catch clauses, each catch handling different type of exception. When an exception is thrown jvm checks each catch statement in order and the first one which matches the type of exception is execution and remaining catch blocks are skipped.

**Explain importance of finally block in java?**

Finally, block is used for cleaning up of resources such as closing connections, sockets etc. if try block executes with no exceptions then finally is called after try block without executing catch block. If there is exception thrown in try block finally block executes immediately after catch block. If an exception is thrown, finally block will be executed even if the no catch block handles the exception.

**Can we have any code between try and catch blocks?**

We shouldn’t declare any code between try and catch block. Catch block should immediately start after try block.

**Can we catch more than one exception in single catch block?**

From Java 7, we can catch more than one exception with single catch block. This type of handling reduces the code duplication.

Note: When we catch more than one exception in single catch block , catch parameter is implicity final. We cannot assign any value to catch parameter.

Ex: catch(ArrayIndexOutOfBoundsException | ArithmeticException e) {

}

In the above example e is final we cannot assign any value or modify e in catch statement.

Example:

**package** package1;

**public** **class** ExceptionExample {

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

**int** arr1[] = {10,20,30};

**int** arr2[] = {2,0,10};

**for**(**int** i=0; i<3; i++) {

**for**(**int** j=0; j<4; j++) {

**try** {

System.***out***.println(arr1[i]/arr2[j]);

}

**catch**(ArrayIndexOutOfBoundsException | ArithmeticException e) {

System.***out***.println(e);

//}

//catch(ArithmeticException e) {

// System.out.println(e);

}

}

}

}

}

**What are checked Exceptions?**

1) All the subclasses of Throwable class except error, Runtime Exception and its subclasses are checked exceptions.

2) Checked exception should be thrown with keyword throws or should be provided try catch block, else the program would not compile.

We do get compilation error.

Examples:

1) IOException,

2) SQlException,

3) FileNotFoundException,

4) InvocationTargetException,

5) CloneNotSupportedException

6) ClassNotFoundException

7) InstantiationException

**What are unchecked exceptions in java?**

All subclasses of RuntimeException are called unchecked exceptions. These are unchecked exceptions because compiler does not checks if a method handles or throws exceptions. Program compiles even if we do not catch the exception or throws the exception. If an exception occurs in the program, program terminates. It is difficult to handle these exceptions because there may be many places causing exceptions.

Example :

1) Arithmetic Exception

3) ArrayIndexOutOfBoundsException

4) ClassCastException

5) IndexOutOfBoundException

6) NullPointerException

7) NumberFormatException

8) StringIndexOutOfBounds

9) UnsupportedOperationException

**Explain differences between checked and Unchecked exceptions in java?**

|  |  |
| --- | --- |
| **Unchecked Exceptions** | **Checked Exceptions** |
| All the subclasses of RuntimeException are called unchecked exception. | All subclasses of Throwable class except RuntimeException are called as checked exceptions |
| Unchecked exceptions need not be handled at compile time | Checked Exceptions need to be handled at compile time. |
| These exceptions arise mostly due to coding mistakes in our program. |  |
| ArrayIndexOutOfBoundsException, ClassCastException, IndexOutOfBoundException | SqlException, FileNotFoundException,ClassNotFoundException |

**What is default exception handling in Java?**

When JVM detects exception causing code, it constructs a new exception handling object by including the following information.

1) Name of Exception

2) Description about the Exception

3) Location of Exception.

After creation of object by JVM it checks whether there is exception handling code or not. If there is exception handling code then exception handles and continues the program. If there is no exception handling code JVM give the responsibility of exception handling to default handler and terminates abruptly. Default Exception handler displays description of exception, prints the stacktrace and location of exception and terminates the program.

Note: The main disadvantage of this default exception handling is program terminates abruptly.

Example:

**package** package1;

**public** **class** ExceptionExample {

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

**int** arr1[] = {10,20,30};

**int** arr2[] = {2,0,10};

**for**(**int** i=0; i<3; i++) {

**for**(**int** j=0; j<3; j++) {

System.***out***.println(arr1[i]/arr2[j]);

}

}

}

}

**Result:**

Exception in thread "main"

java.lang.ArithmeticException: / by zero

at package1.ExceptionExample.main(ExceptionExample.java:11)

**Explain throw keyword in java?**

Generally, JVM throws the exception and we handle the exceptions by using try catch block.

But there are situations where we have to throw user defined exceptions or runtime exceptions. In such case we use throw keyword to throw exception explicitly.

Syntax : throw throwableInstance;

Throwable instance must be of type throwable or any of its subclasses. After the throw statement execution stops and subsequent statements are not executed. Once exception object is thrown JVM checks is there any catch block to handle the exception. If not then the next catch statement till it finds the appropriate handler. If appropriate handler is not found, then default exception handler halts the program and prints the description and location of exception. In general, we use throw keyword for throwing user defined or customized exception.

**Can we write any code after throw statement?**

After throw statement jvm stop execution and subsequent statements are not executed. If we try to write any statement after throw we do get compile time error saying unreachable code

**Explain importance of throws keyword in java?**

Throws statement is used at the end of method signature to indicate that an exception of a given type may be thrown from the method.

The main purpose of throws keyword is to delegate responsibility of exception handling to the caller methods, in the case of checked exception.

In the case of unchecked exceptions, it is not required to use throws keyword. We can use throws keyword only for throwable types otherwise compile time error saying incompatible types.

An error is unchecked, it is not required to handle by try catch or by throws.

Syntax :

Class Test{

Public static void main(String args[]) throws IE { }

}

Note: The method should throw only checked exceptions and subclasses of checked exceptions. It is not recommended to specify exception super classes in the throws class when the actual exceptions thrown in the method are instances of their subclass.

**Explain the importance of finally over return statement?**

finally block is more important than return statement when both are present in a program. For example if there is any return statement present inside try or catch block , and finally block is also present first finally statement will be executed and then return statement will be considered.

**Explain a situation where finally block will not be executed?**

Finally block will not be executed whenever jvm shutdowns. If we use system.exit(0) in try statement finally block if present will not be executed.

**Can we use catch statement for checked exceptions?**

If there is no chance of raising an exception in our code then we can’t declare catch block for handling checked exceptions. This raises compile time error if we try to handle checked exceptions when there is no possibility of causing exception.

**What are user defined exceptions?**

To create customized error messages we use userdefined exceptions. We can create user defined exceptions as checked or unchecked exceptions. We can create user defined exceptions that extend Exception class or subclasses of checked exceptions so that userdefined exception becomes checked. Userdefined exceptions can extend RuntimeException to create userdefined unchecked exceptions.

Note: It is recommended to keep our customized exception class as unchecked,i.e we need to extend Runtime Exception class but not Excpetion class.

**Can we rethrow the same exception from catch handler?**

Yes we can rethrow the same exception from our catch handler. If we want to rethrow checked exception from a catch block we need to declare that exception.

**Can we nested try statements in java?**

Yes try statements can be nested. We can declare try statements inside the block of another try statement.

**Explain the importance of throwable class and its methods?**

Throwable class is the root class for Exceptions. All exceptions are derived from this throwable class. The two main subclasses of Throwable are Exception and Error.

The three methods defined in throwable class are :

1. void printStackTrace() :

This prints the exception information in the following format : Name of the exception, description followed by stack trace.

1. getMessage()

This method prints only the description of Exception.

1. toString():

It prints the name and description of Exception.

**Explain when ClassNotFoundException will be raised?**

When JVM tries to load a class by its string name, and couldn’t able to find the class classNotFoundException will be thrown. An example for this exception is when class name is misspelled and when we try to load the class by string name hence class cannot be found which raises ClassNotFoundException.

Example:

**package** intquestions;

**public** **class** testclass

{

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

**try** {

Class.*forName*("com.journaldev.MyInvisibleClass");

ClassLoader.*getSystemClassLoader*().loadClass("com.journaldev.MyInvisibleClass");

}

**catch**(Exception e) {

System.***out***.println(e);

}

}

}

**Explain when NoClassDefFoundError will be raised?**

This error is thrown when JVM tries to load the class but no definition for that class is found NoClassDefFoundError will occur. The class may exist at compile time but unable to find at runtime. This might be due to misspelled class name at command line, or classpath is not specified properly, or the class file with byte code is no longer available.