<https://howtodoinjava.com/best-practices/java-exception-handling-best-practices/>

<https://javaconceptoftheday.com/java-exception-handling-interview-questions-and-answers/>

<https://javahungry.blogspot.com/2019/11/exception-handling-interview-questions-and-answers.html>

<https://www.journaldev.com/1696/exception-handling-in-java>

<https://www.baeldung.com/java-exceptions-interview-questions>

<https://www.java67.com/2019/06/top-25-java-exception-interview-questions-answers.html>

<https://www.examveda.com/java-program/practice-mcq-question-on-exceptions/?page=3>

What is Exception in Java

The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

**Dictionary Meaning:** Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

## What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException etc.

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling.

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

|  |  |
| --- | --- |
| **Exception Handling in Java**  1. [Exception Handling](https://www.javatpoint.com/exception-handling-in-java) 2. [Advantage of Exception Handling](https://www.javatpoint.com/exception-handling-in-java#exceptionad) 3. [Hierarchy of Exception classes](https://www.javatpoint.com/exception-handling-in-java#exceptionhierarchy) 4. [Types of Exception](https://www.javatpoint.com/exception-handling-in-java#exceptiontypes) 5. [Exception Example](https://www.javatpoint.com/exception-handling-in-java#exceptionexample) 6. [Scenarios where an exception may occur](https://www.javatpoint.com/exception-handling-in-java#exceptionscenarios)   The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors   so that normal flow of the application can be maintained. What is Exception in Java **Dictionary Meaning:** Exception is an abnormal condition.  In Java, an exception is an event that disrupts the normal flow of the program.  It is an object which is thrown at runtime. What is Exception Handling Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException,  IOException, SQLException, RemoteException, etc. **Advantage of Exception Handling** The core advantage of exception handling is **to maintain the normal flow of the application**.  An exception normally disrupts the normal flow of the application that is why we use exception  handling. Let's take a scenario:   1. statement 1; 2. statement 2; 3. statement 3; 4. statement 4; 5. statement 5;//exception occurs 6. statement 6; 7. statement 7; 8. statement 8; 9. statement 9; 10. statement 10;   Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.  Do You Know?   |  | | --- | | * What is the difference between checked and unchecked exceptions? * What happens behind the code int data=50/0;? * Why use multiple catch block? * Is there any possibility when finally block is not executed? * What is exception propagation? * What is the difference between throw and throws keyword? * What are the 4 rules for using exception handling with method overriding? |  Hierarchy of Java Exception classes The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited  by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:  hierarchy of exception handling |

### **Types of Java Exceptions**

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

## Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place exception code.  The try block must be followed by either catch or finally.  It means, we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try  block which means we can't use catch block alone. It can be followed by finally  block later. |
| finally | The "finally" block is used to execute the important code of the program.  It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception.  It specifies that there may occur an exception in the method. It is always used with  method signature. |

## Difference between Checked and Unchecked Exceptions

### **1) Checked Exception**

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### **2) Unchecked Exception**

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### **3) Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

## Java try block

Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement of try block, the rest of the block code will not execute. So, it is recommended not to keeping the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

### **Syntax of Java try-catch**

1. **try**{
2. //code that may throw an exception
3. }**catch**(Exception\_class\_Name ref){}

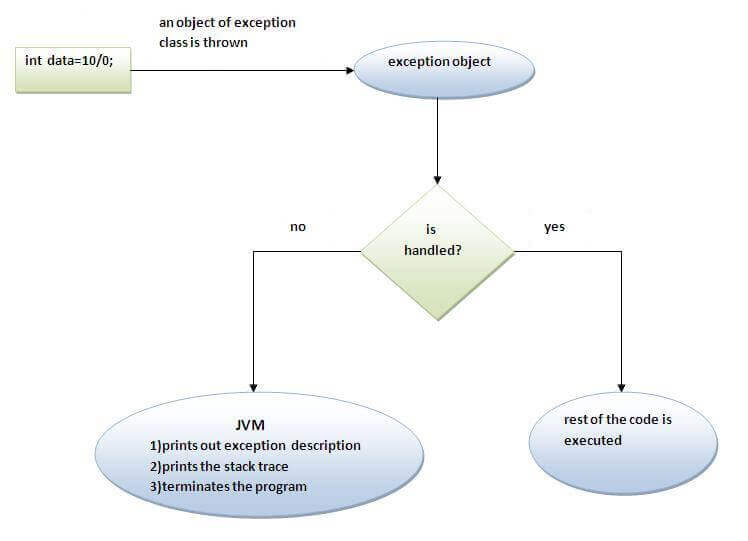
### **Syntax of try-finally block**

1. **try**{
2. //code that may throw an exception
3. }**finally**{}

## Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception ( i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

The catch block must be used after the try block only. You can use multiple catch block with a single try block.



Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Points to remember

* At a time only one exception occurs and at a time only one catch block is executed.
* All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

.Finally Block



# **Java finally block**

**Java finally block** is a block that is used to execute important code such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block

* Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

#### **Rule: For each try block there can be zero or more catch blocks, but only one finally block.**

#### **Note: The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).**

## Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

**throw** exception;

**throw** **new** IOException("sorry device error);

# **Java throws keyword**

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

### **Syntax of java throws**

1. return\_type method\_name() **throws** exception\_class\_name{
2. //method code
3. }

### **Advantage of Java throws keyword**

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

### Rule: If you are calling a method that declares an exception, you must either caught or declare the exception.

|  |
| --- |
| There are two cases:   1. **Case1:**You caught the exception i.e. handle the exception using try/catch. 2. **Case2:**You declare the exception i.e. specifying throws with the method. |

1. **import** java.io.IOException;
2. **class** Testthrows1{
3. **void** m()**throws** IOException{
4. **throw** **new** IOException("device error");//checked exception
5. }
6. **void** n()**throws** IOException{
7. m();
8. }
9. **void** p(){
10. **try**{
11. n();
12. }**catch**(Exception e){System.out.println("exception handled");}
13. }
14. **public** **static** **void** main(String args[]){
15. Testthrows1 obj=**new** Testthrows1();
16. obj.p();
17. System.out.println("normal flow...");
18. }
19. }

**ExceptionHandling with MethodOverriding in Java.**

* **If the superclass method does not declare an exception**
  + If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
* **If the superclass method declares an exception**
  + If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

#### **1) Rule: If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.**

# **throw and throws in Java**

**throw**

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either [checked or unchecked exception](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/). The throw keyword is mainly used to throw custom exceptions.

**throw *Instance***

Example:

**throw new ArithmeticException("/ by zero");**

But this exception i.e, Instance must be of type **Throwable** or a subclass of **Throwable**. For example Exception is a sub-class of Throwable and [user defined exceptions typically extend Exception class](https://www.geeksforgeeks.org/g-fact-32-user-defined-custom-exception-in-java/).

**throws**

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

In a program, if there is a chance of rising an exception then compiler always warn us about it and compulsorily we should handle that checked exception, Otherwise we will get compile time error saying **unreported exception XXX must be caught or declared to be thrown**. To prevent this compile time error we can handle the exception in two ways:

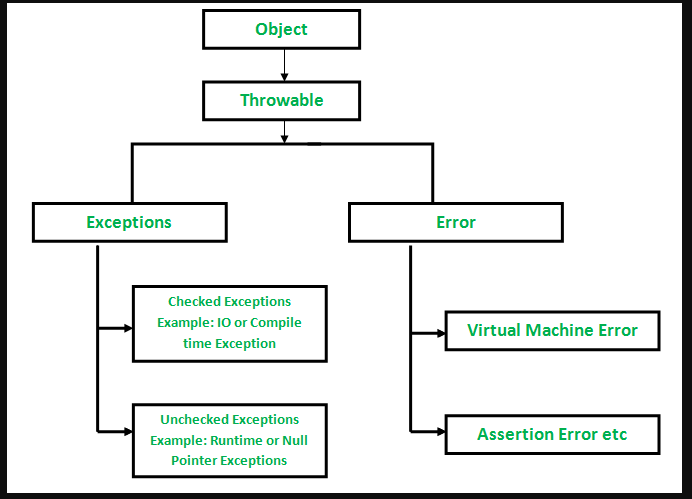
1. By using [try catch](https://www.geeksforgeeks.org/flow-control-in-try-catch-finally-in-java/)
2. By using **throws** keyword

We can use throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

**Important points to remember about throws keyword:**

* throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
* throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.
* By the help of throws keyword we can provide information to the caller of the method about the exception.

**Q6. Are we allowed to use only try block without a catch and finally blocks?**  
  
***Prior to Java 7:***  
No, it is not allowed. If used it shows compilation error. The try block must be followed by a catch block or finally block.  
  
***After Java 7 (Correct Answer):***  
Yes, it is possible to have a try block without a catch and finally blocks. The introduction of try-with-resources concept makes it possible.  
The only constraint is resources which we are passing as a parameter in try block must implement [AutoCloseable](https://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html" \t "_blank) interface.



### **Q12. Can You Throw Any Exception Inside a Lambda Expression's Body?**

When using a standard functional interface already provided by Java, you can only throw unchecked exceptions because standard functional interfaces do not have a “throws” clause in method signatures:

|  |  |
| --- | --- |
|  | List<Integer> integers = Arrays.asList(3, 9, 7, 0, 10, 20);  integers.forEach(i -> {      if (i == 0) {          throw new IllegalArgumentException("Zero not allowed");      }      System.out.println(Math.PI / i);  }); |

However, if you are using a custom functional interface, throwing checked exceptions is possible:

|  |  |
| --- | --- |
|  | @FunctionalInterface  public static interface CheckedFunction<T> {      void apply(T t) throws Exception;  } |
|  | public void processTasks(    List<Task> taks, CheckedFunction<Task> checkedFunction) {      for (Task task : taks) {          try {              checkedFunction.apply(task);          } catch (Exception e) {              // ...          }      }  }    processTasks(taskList, t -> {      // ...      throw new Exception("Something happened");  }); |

# **The try-with-resources statement**

In Java, the try-with-resources statement is a try statement that declares one or more resources. The resource is as an object that must be closed after finishing the program. The try-with-resources statement ensures that each resource is closed at the end of the statement execution.

You can pass any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable.

The following example writes a string into a file. It uses an instance of FileOutputStream to write data into the file. FileOutputStream is a resource that must be closed after the program is finished with it. So, in this example, closing of resource is done by itself try.