EXCEPTION HANDLING

An exception is an event which disrupts the normal flow of execution.

When an exception occurs, and if you don’t handle it, the program will terminate abruptly (the piece of code after the line causing the exception will not get executed).

An exception can be handled either by programmer or by JVM.JVM handles it by using default exception handler and programmer can handle by using various blocks

REASONS FOR EXCEPTION

A user has entered an invalid data

File not found

A network connection is lost in middle of communication

The JVM has run out of memory

**CheckedException**  
It is an exception that occurs at compile time.

If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

**UncheckedException**  
It is an exception that occurs at the time of execution.

It is up to the programmers to specify or catch the exceptions.

**User-Defined Exceptions**

Users can also create their own exceptions called User-Defined Exceptions if built in-exceptions are not describing a certain situation.

Key points to note:

A user-defined exception must extend Exception class.

The exception is thrown using throw keyword.

**Exception Handling Methods**

**try**

**catch**

**finally**

**throw**

**throws**

**try block**

The try block contains a set of statements where an exception can occur.

It is always followed by a catch block, which handles the exception that occurs in the associated try block.

A try block must be followed by catch blocks or finally block or both.

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

**catch block**

A catch block is where you handle the exceptions.

This block must follow the try block and a single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks.

When an exception occurs in a try block, the catch block catches the exception and statements inside the catch block is executed.

If none of the statements in the try block generates an exception, the catch block is skipped.

class Main {

public static void main(String[] args) {

try {

// code that generate exception

int divideByZero = 5 / 0;

System.out.println("Rest of code in try block");

}

catch (ArithmeticException e) {

System.out.println("ArithmeticException => " + e.getMessage());

}

**finally block**

Finally block contains the clean up code.Clean up code is a type of code associated with closing the system resources which are opened inside the try block

The statements present in this block is always executed, no matter an exception occurs in the try block or not,such as closing a connection, stream etc.

The finally block is optional. And, for each try block, there can be only one finally block.

class Main {

public static void main(String[] args) {

try {

// code that generates exception

int divideByZero = 5 / 0;

}

catch (ArithmeticException e) {

System.out.println("ArithmeticException => " + e.getMessage());

}

finally {

System.out.println("This is the finally block");

}

}

}

Output: ArithmeticException => / by zero

This is the finally block

## **final vs finally vs finalize**

|  |  |  |
| --- | --- | --- |
| **final** | **finally** | **finalize** |
| It is a keyword. | It is a block. | It is a method. |
| Used to apply restrictions on class, methods & variables. | Used to place an important code. | Used to perform clean-up processing just before the object is garbage collected. |
| final class can’t be inherited, method can’t be overridden & the variable value can’t be changed. | It will be executed whether the exception is handled or not. | – |

## **throw vs throws**

|  |  |
| --- | --- |
| **throw** | **throws** |
| 1. Used to explicitly throw an exception | 1. Used to declare an exception |
| 2. Checked exceptions cannot be propagated using throw only | 2. Checked exceptions can be propagated |
| 3. Followed by an instance | 3. Followed by a class |
| 4. Used within a method | 4. Used with a method signature |
| 5. Cannot throw multiple exceptions | 5. Can declare multiple exceptions |

|  |  |
| --- | --- |
| 1  3  4  5  6  7  8  9  10  11  12  13  14  15 | //Java throw example  **void** a()  {  **throw** **new** ArithmeticException("Incorrect");  }  //Java throws example  **void** a()**throws** ArithmeticException  {    //method code  }  //Java throw and throws example  **void** a()**throws** ArithmeticException  {  **throw** **new** ArithmeticException("Incorrect");  } |

ClassNotFound Exception: During Class loading,if JVM is unable to find the .class file by using the class loader program inside the Hard Disk,then it results in ClassNotFound Exception

IO Exception: It is thrown when an input-output operation is failed or interrupted. It is generated when programmer performs file reading or file writing operation.For file reading,the programmer uses FileInput Stream and for FileWriting programmer uses FileOutput Stream.If streams are not created by Operating system,then jvm generates an Exception called as IO Exception

FileNotFound Exception: During file reading,the OS properly creates the input stream.But the file is not existing inside the hard-disk,then it generates FileNotFound Exception.

ArrayIndexOutOfBound Exception: It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

NullPointer Exception: occurs when a variable is accessed which is not pointing to any object and refers to nothing or null.

Arithmetic Exception:It is thrown when an exceptional condition has occurred in an arithmetic operation.

Interrupted Exception:It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

//Note: Runtime exception,doesn’t need to be caught and handled explicitly in application code.

EXCEPTION HIERARCHY

OBJECT CLASS

Arithmetic Exception

NullPointer Exception

IndexOutOfBoundException(AIOBE,SIOBE)

NegativeArraySize Exception

ClassNotFound Exception

IO Exception

FileNotFound Exception

Interrupted Exception

CHECKED EXCEPTION(CompileTime Exception)

UNCHECKED EXCEPTION(RunTime Exception

EXCEPTION CLASS

THROWABLE CLASS(Super class for exception class)