**UNIT-4**

**Exception:** An exception is an unexpected event that occurred during the execution of a program, and disrupts the normal flow of instructions.

**Exception Handling**

**E**xception handling is a technique to convert system generated errors in to user friendly messages.

There are two types of exceptions in java

1.**checked exceptions:**

These exceptions are checked at compile time,

Compile time errors occurs when the java programmer not followed syntaxes

Examples: IOException, SQLException

2. **unchecked exceptions:**

These exceptions checked at run time

Run time error occurs when user enters invalid input

Run time errors are called exceptions

Examples:

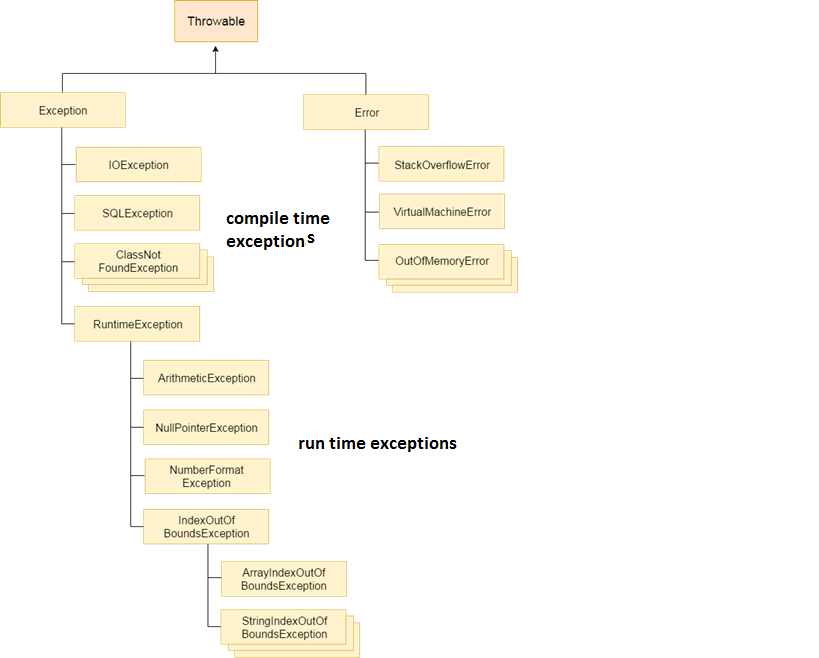
* ArithmeticException
* ArrayIndexOutOfBoundsException
* NullPointerException
* NumberFormatException
* StringIndexOutOfBoundsException
* FileNotFoundException

|  |  |
| --- | --- |
| **Checked Exception** | **Unchecked Exception** |
| Checked exceptions occur at compile time. | Unchecked exceptions occur at runtime. |
| The compiler checks a checked exception. | The compiler does not check these types of exceptions. |
| These types of exceptions can be handled at the time of compilation. | These types of exceptions cannot be a catch or handle at the time of compilation these are run time exceptions |
| They are the sub-class of the exception class. | These are the subclasses of RuntimeException class |
|  |  |
| Examples of Checked exceptions:   * File Not Found Exception * No Such Field Exception * Interrupted Exception * No Such Method Exception * Class Not Found Exception | Examples of Unchecked Exceptions:   * No Such Element Exception * Undeclared Throwable Exception * Empty Stack Exception * Arithmetic Exception * Null Pointer Exception * Array Index Out of Bounds Exception * Security Exception |

ExceptionHandling Heirarchy Chart

**Throwable class**

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:



(B)Contrast Error and Exception.

**Difference between Error and Exception**

| **Sr. No.** | **Error** | **Exception** |
| --- | --- | --- |
| 1 | Error is Classified as an unchecked type | Exception is Classified as checked and unchecked |
| 2 | It belongs to java.lang.Error | It belongs to java.lang.Exception |
| 3 | Error cannot be handled It is irrecoverable | Exception can be handled gy using exception handling concepts.It is recoverable |
| 4 | It occurs at runtime | It can occur at run time and compile time |
| 5 | OutOfMemoryError ,IOError | NullPointerException , SqlException |

There are two types of exceptions in java related to libraries

1.built in exception/pre-defined exception

2. user defined exception/custom defined exception

Built-in exceptions are pre-defined exceptions which are available in Java libraries

Examples

* ArithmeticException
* ArrayIndexOutOfBoundsException
* NullPointerException
* NumberFormatException
* StringIndexOutOfBoundsException

User defined(Custom exception)

exception are these are defined by the programmer

Example

Invalid salary of employee

Invalid age of human beign

To display user friendly messages instead of system generated messages

**Exception Handlers**

We need the following five keywords

1.try

2.catch

3.finally

4.throw

5.throws

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 close the files etc |
| Throw | The "throw" keyword is used to throw an custom defined 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. |

Built in exceptions examples

**Arithmetic exception :**It is thrown when an exceptional condition has occurred in an arithmetic operation

// Java program to demonstrate

// ArithmeticException

class ArithmeticException\_Demo {

public static void main(String args[])

{

try {

int a = 30, b = 0;

int c = a / b; // cannot divide by zero

System.out.println("Result = " + c);

}

catch (ArithmeticException e) {

System.out.println("Can't divide a number by 0");

}

}

}

Output

Can't divide a number by 0

ArrayIndexOutOfBoundsException :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.

// Java program to demonstrate

// ArrayIndexOutOfBoundException

class ArrayIndexOutOfBound\_Demo {

public static void main(String args[])

{

try {

int a[] = new int[5];

a[6] = 9; // accessing 7th element in an array of

// size 5

}

catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array Index is Out Of Bounds");

}

}

}

Output

Array Index is Out Of Bounds

**FileNotFoundException :** This Exception is raised when a file is not accessible or does not open.

// Java program to demonstrate

// FileNotFoundException

import java.io.\*;

class File\_notFound\_Demo {

public static void main(String args[])

{

try {

FileReader fr = new FileReader(“E:\\cse.txt”);

}

catch (FileNotFoundException e) {

System.out.println("File does not exist");

}

}

}

Output

File does not exist

**NullPointerException :**This exception is raised when referring to the members of a null object. Null represents nothing

// Java program to demonstrate NullPointerException

class NullPointer\_Demo {

public static void main(String args[])

{

try {

String a = null; // null value

System.out.println(a.charAt(0));

}

catch (NullPointerException e) {

System.out.println("NullPointerException..");

}

}

}

NullPointerException..

**NumberFormatException :**This exception is raised when a method could not convert a string into a numeric format.

// Java program to demonstrate

// NumberFormatException

class NumberFormat\_Demo {

public static void main(String args[])

{

try {

// "akki" is not a number

int num = Integer.parseInt("akki");

System.out.println(num);

}

catch (NumberFormatException e) {

System.out.println("Number format exception");

}

}

}

**StringIndexOutOfBoundsException :** It is thrown by String class methods to indicate that an index is less than the size of the string.

class StringIndexOutOfBound\_Demo {

public static void main(String args[])

    {

        try{

            String a = "This is like chipping "; // length is 22

            char c = a.charAt(24); // accessing 25th element

            System.out.println(c);

        }

        catch(StringIndexOutOfBoundsException e) {

            System.out.println("StringIndexOutOfBoundsException");

        }

    }

}

Output

StringIndexOutOfBoundsException

## 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.

**MultipleCatchBlock1.java**

public class MultipleCatchBlock1 {

public static void main(String[] args) {

try{

int a[]=new int[5];

a[5]=30/0;

}

catch(ArithmeticException e)

{

System.out.println("Arithmetic Exception occurs");

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("ArrayIndexOutOfBounds Exception occurs");

}

catch(Exception e)

{

System.out.println("Parent Exception occurs");

}

finally{

System.out.println("rest of the code");

}

}

}

Explain about finally exception handler with java code.

**Finally block**

Th finally block in java is used to put important codes such as clean up code e.g. closing the file or closing the connection. The finally block executes whether exception rise or not and whether exception handled or not. A finally contains all the crucial statements regardless of the exception occurs or not.

// Java program to demonstrate

// finally block in java When

// exception does not rise

import java.io.\*;

class GFG {

    public static void main(String[] args)

    {

        try {

            System.out.println("inside try block");

            // Not throw any exception

            System.out.println(34 / 2);

        }

        // Not execute in this case

        catch (ArithmeticException e) {

            System.out.println("Arithmetic Exception");

        }

        // Always execute

        finally {

            System.out.println(

                "finally : i execute always.");

        }

    }

}

**Throw Keyword**

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

We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

**Syntax:**

throw objectName;

or

throw new ClassName()

**ValidAge.java**

class ValidAge

{

public static void main(String args[])

{

int voterAge=17;

if(voterAge<18)

throw new ArithmeticException("Not Valid Age");

else

System.out.println("valid voter age");

}

}

javac ValidAge.java

java ValidAge

Exception in thread "main" java.lang.ArithmeticException: Not Valid Age

at ValidAge.main(ValidAge.java:11)

**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

Throws keyword is used along with the method signature

Syntax:

Returntype methodname(arg1,arg2,arg3,arg4…..) throws Exception1,Exception2,,,,,,,,,,

{

}

Example

void fileRead()throws FileNotFoundException,IOException

{

}

**ThrowsExample.java**

import java.io.\*;

class ThrowsExample

{

public static void main(String args[])throws FileNotFoundException,IOException

{

FileReader fr=new FileReader("E:\\cse.txt");

FileWriter fw=new FileWriter("E:\\ece.txt");

int ch;

while((ch=fr.read())!=-1)

{

fw.write((char)ch);

}

fw.close();

fr.close();

System.out.println("FileReading and Writing Successfull");

}

}

**Output**

javac ThrowsExample.java

java ThrowsExample

FileReading and Writing Successfull

Difference between throw and throws in Java

|  |  |
| --- | --- |
|  | |
|  | Throw | Throws |
| 1 | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
|  |  |  |
| 2 | Throw is followed by an instance. | Throws is followed by class. |
| 3 | Throw is used within the method. | Throws is used with the method signature. |
| 4 | You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |
| 5 | At a time we can throw one object at a time | At a time we can declare any no of exceptions using  Throws keyword |
| 6 | throw objectname;  (or)  throw new Classname() | 1. return\_type method\_name() **throws** exception\_class\_name{ 2. //method code 3. } |

**User defined exceptions**

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, user can also create exceptions which are called ‘user-defined Exceptions’.

How to create user defined exception:

* Step 1:The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:

classMyException extends Exception

* Step 2:We can write a default constructor in his own exception class.

MyException(){}

We can also create a parameterized constructor with a string as a parameter.  
We can use this to store exception details.

Step 3:We can call super class(Exception) constructor from this and send the string there.

MyException(String str)

{

super(str);

}

Step 4:To raise exception of user-defined type, we need to create an object to this exception class and throw it using throw clause, as:

MyException me = new MyException(“Exception details”);

throw me;

**User defined exception example**

VoterAgeLessException.java

import java.util.Scanner;

class VoterAgeLessException extends Exception

{

VoterAgeLessException(String str)

{

super(str);

}

public static void main(String args[])throws VoterAgeLessException

{

Scanner sc=new Scanner(System.in);

System.out.println("enter voter age");

try{

int age=sc.nextInt();

if(age<18)

{

throw new VoterAgeLessException("voter age less");

}

else

System.out.println("valid age");

}

catch(ArithmeticException a)

{

System.out.println(a);

}

finally

{

}

}

}

Output-1

javac VoterAgeLessException.java

java VoterAgeLessException

enter voter age

12

Exception in thread "main" VoterAgeLessException: voter age less

at VoterAgeLessException.main(VoterAgeLessException.java:17) Output-2

Output2

javac VoterAgeLessException.java

java VoterAgeLessException

enter voter age

18

valid age

**Chained Exception**

In Java, **chained exceptions** allow **you to associate one exception with another**, providing a more comprehensive context for debugging by showing the sequence of exceptions that led to the problem. It helps **trace the root cause of an exception**, which is **often useful when dealing with complex applications** where multiple exceptions might be involved.

**How to Use Chained Exceptions in Java**

You can create a chained exception using:

1. The constructor of an exception that takes a cause as an argument.
2. The initCause() method of the Throwable class.

**ChainedExceptionExample.java**

class ChainedExceptionExample {

public static void main(String[] args) {

try {

// Simulating a method call that causes an exception

method1();

} catch (Exception e) {

// Catching the outer exception

System.out.println("Caught exception: " + e);

System.out.println("Original cause: " + e.getCause());

}

}

static void method1() {

try {

// Code that throws an exception

int result = 10 / 0;

} catch (ArithmeticException e) {

// Chaining the original exception

throw new RuntimeException("Runtime exception occurred", e);

}

}

}

Output

Caught exception: java.lang.RuntimeException: Runtime exception occurred

Original cause: java.lang.ArithmeticException: / by zero