# UNIT 8: EXCEPTION HANDLING

**UNIT STRUCTURE**


8.1 Learning Objectives

8.2 Introduction

8.3 Exception Handling Fundamentals

8.4 Exception Types

8.5 Try and Catch

8.6 Throw, Throws and Finally

8.7 Java's Build-in Exceptions

8.8 User Defined Exception Subclasses

8.9 Let Us Sum Up

8.10 Further Readings

8.11 Answers to Check Your Progress

8.12 Model Questions


## 8.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- handle exceptions in Java programming language
- describe different types of exception and exception hierarchy
- use try, catch, throw, throws and finally keywords in programming.
- use user defined exception subclasses

## 8.2 INTRODUCTION

In the previous unit, we have learnt about packages and interfaces. In this unit, we will explore the different concepts related to exception handling. Exception handling are related to errors. There are two types of errors in a computer program i.e., system error and logic errors. Syntax error causes due to typing mistake of the programmer. On the other hand, logic errors occur due to poor understanding of the problem in the language. Some logic errors can be controlled during the execution of a program.

Exception is a problem arises during execution of a Java program. The normal flow of the program is disrupted and the program is terminated. This is not recommended procedure in a programme or application. That is why exception need to be handled. There is a list of built in exception classes which is under Java's standard package java.lang. Java also support user defined exception handling classes which is discussed later in this unit. In the next unit, we will explore the different issues related to I/O in Java programming.

## 8.3 EXCEPTION HANDLING FUNDAMENTALS

During execution of a program an exception (an event) arises. When an exception occurs the normal flow of the program is disrupted and the program or the application terminates abnormally, which is not recommended, therefore, these exceptions need to be handled.

An exception can occur for many different reasons. Some of the probable scenarios for an exception occurs is given bellow

- Divide by zero, accessing arrays out of its bounds, running out of memory, overflow and underflow.
- When a user entered an invalid data.
- A file that needs to be opened but cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, some by programmer error, and others by physical resources that have failed in some manner.

## 8.4 EXCEPTION TYPES

There are three categories of exceptions. They are *errors*, *compile time exceptions* and *runtime exceptions.*

- **Errors**

Some error are beyond the control of the programmer. Errors are typically ignored in the coding phase because the programmer cannot do anything doing coding. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation. Some of the common errors are **OutOfMemoryError** and **StackOverflowError.**

- **Compile time exceptions**

This type of exceptions occurs during the compile time. These exceptions need to be handled. For example, in a program to read data from a file, if the specified file does not exist, then a FileNotFoundException occurs, and the compiler prompts the programmer to handle the exception. The following program segment has shown compile time exceptions.

```
import  java.io.File;
import  java.io.FileReader;
public  class  FilenotFound_Example
 {
        public  static  void  main(String  args[])
         {
                File  file  =  new  File("C://file.txt");
                FileReader  fr  =  new  FileReader(file);
        }
}
```

If the file "file.txt" does not exist, once compile the above program, you will get the following exceptions.

FilenotFound_Example.java:8: error: unreported exception FileNotFoundException; must be caught or declared to be thrown

```
    FileReader fr = new FileReader(file);
               ^
```

1 error

- **Runtime exceptions**

This type of exception occurs at the time of program execution. This includes programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation. For example:

```
public class Kkhsou_Demo
 {          public  static  void  main(String  args[])
      {
              int num[] = {0, 1, 2, 3, 4};
             System.out.println(num[5]);
      }
}
```

In the above program declared an array of size 5 and in the next line trying to access the 6th element of the array, then an exception occurs.

Output: Exception in thread “main”

java.lang.ArrayIndexOutOfBoundsException: 5

at Kkhsou_Demo.main(Kkhsou_Demo.java:5)

- **Exception Hierarchy**

All exception classes are subclasses of the built-in class **Throwable**. Other than the exception class there is another subclass called **Error** which is derived from the Throwable class. As discussed above, errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Some of the common Errors are **OutOfMemoryError** and **StackOverflowError.**

This exceptions class is used for exceptional conditions that user programs should catch. There is another subclass of **Exception**, called **RuntimeException**. Exceptions of this type are automatically defined for the programs that you write and include things such as division by zero and invalid array indexing.

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 and if we are throwing a checked exception, we must catch it in the same method or we have to propagate it to the caller using throws keyword. Figure 8.1 shows the hierarchy of exception.

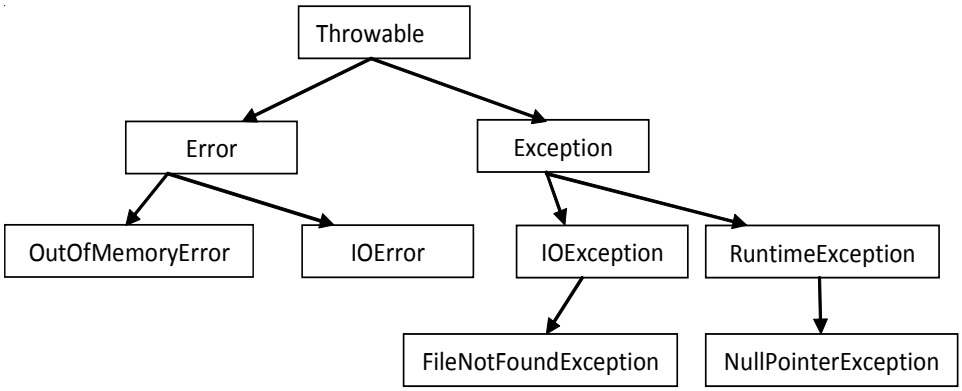



**Figure 8.1: Exception Hierarchy**



**CHECK YOUR PROGRESS**

Q1. What are the different type of exceptions?

Q2. ———— exception occurs at the time of program execution.

Q3. All exception classes are subclasses of the built-in class ———— ————————.

## 8.5 TRY AND CATCH

***try*** and ***catch*** are the keywords used for exception handling.

**try:** If you want to monitor a part of your code of a program for exceptions, that code must be contained within a try block. If an exception occurs within the try block, it is thrown.

**catch:** catch block can catch an exception thrown from the try block and handle it in some logical manner.

Using try and catch keywords, method can detect and throw an exception an exception. A try - catch block is placed around the code that might generate an exception. Code within a try - catch block is referred to as protected code, and the syntax for using try - catch looks like the following

**Syntax :**

```
try {
   // Protected code
}
catch(ExceptionName e1) {
   // Catch block
}
```

The code which may occur exceptions is placed under try block. When an exceptions occurs then it is handled by the catch block associated with it. Every try block should be immediately followed either by a catch block or finally block. There may be multiple catch block for a particular try block.

**Program 8.1 : Program showing try catch block.**

```
import java.io.*;
public class ExcepTest {
   public static void main(String args[]) {
```

```
      try {
     int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
       }catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown  :" + e);
       }
       System.out.println("Out of the block");
   }
}
```

**Output:**

Exception thrown: java.lang.ArrayIndexOutOfBoundsException: 3

Out of the block

In the above example, an array declared with 2 elements. Then the code tries to access the 4th element of the array which throws an exception.

## 8.6 THROW, THROWS AND FINALLY

Like **try** and **catch, throw, throws** and **finally** are the keywords.

**throw:** System-generated exceptions are automatically thrown by the Java run-time system. To throw an exception manually, the throw keyword is used. Only object of Throwable class or its sub classes can be thrown. On encountering **throw** statement program execution stops, and the closest **try** block is inspected to see if it has a **catch** statement that matches the type of exception.

Syntax: **throw** *ThrowableInstance*

**Program 8.2 : Program using throw keyword**

```
class  Test{
        static  void  avg()  {
                try   {
                        throw new ArithmeticException("demo");
                }
                catch(ArithmeticException  e){
                       System.out.println("Exception  caught");
```

```
                }
        }
         public static void main(String args[]) {
                avg();
        }
}
```

In the above example the avg() method throw an instance of ArithmeticException, which is successfully handled using the catch statement.

**throws:** Any method capable of causing an exceptions that it does not handle, must list all the exceptions possible during its execution, so that anyone calling that method gets a prior knowledge about which exceptions to handle. A method can do so by using the throws keyword.

**Syntax:**

***type method_name(parameter_list)*** **throws** ***exception_list***

**{**

**//definition of method**

**}**

**Program 8.3 : Program using throws keyword**

```
class Test{
        static void check() throws ArithmeticException {
                System.out.println("Inside check function");
                throw new ArithmeticException("demo");
        }
         public static void main(String args[]) {
                try {
                        check();
                }
                catch(ArithmeticException e){
                     System.out.println("caught" + e);
                }
        }
}
```

**finally:** Any code that absolutely must be executed before a method returns, is put in a finally block. A finally keyword is used to create a block of code that follows a try block. A finally block of code always executes whether or not exception has occurred. Using a finally block, lets you run any cleanup type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of catch block.

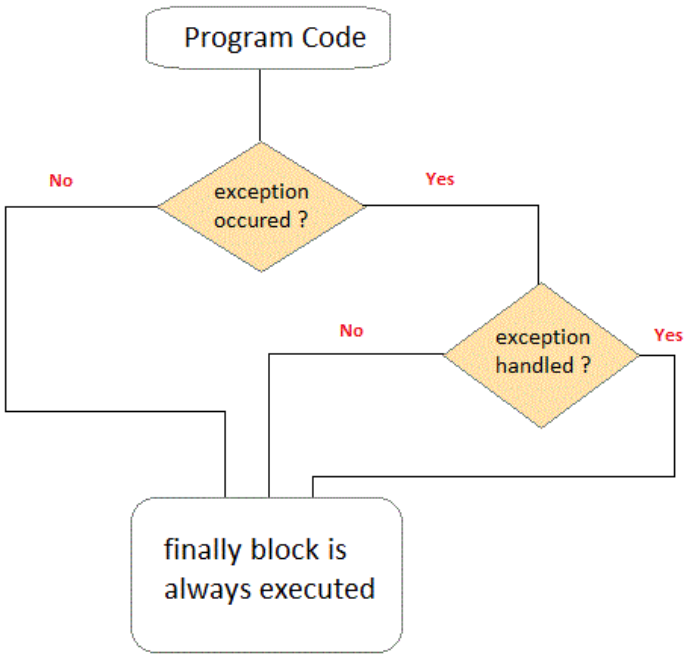



**Figure 8.2 : Finally block**

**Program 8.4 : Program using finally keyword**

```
Class  ExceptionTest
{
        public  static  void  main(String[  ]  args)
            {
            int  a[  ]= new  int[2];
                System.out.println("out  of  try");
                try
                        {
                  System.out.println("Access invalid element"+ a[3]);
   /* the above statement will throw ArrayIndexOutOfBoundException*/
                }
```

```
                finally
                        {
                    System.out.println("finally is always executed.");
                        }
            }
}
```

**Output:**

Out of try

finally is always executed.

Exception in thread main java. Lang. exception array Index out of bound exception.

## 8.7 JAVA'S BUILT- IN EXCEPTIONS

Java's standard package java.lang defines several exception classes. The most general of these exceptions are subclasses of the RuntimeException. Since java.lang is completely imported into all Java programs, most exceptions derived from RuntimeException are automatically available. These exceptions are called *unchecked exceptions* because the compiler does not check to see if a method handles or throws these exceptions as these exceptions need not be included in any method's **throws** list. The list of Java Unchecked RuntimeException is shown in table 8.1.

**Table 8.1: Unchecked RuntimeException Defined in java.lang**

| Sl No | Exceptions | Descriptions |
|---|---|---|
| 1. | ArithmeticException | Error such as divide-by-zero |
| 2. | ArrayIndexOutOfBounds Exception | Array index is out-of-bounds. |
| 3. | ArrayStoreException ment of an incompatible type. | Assignment to an array ele |
| 4. | ClassCastException | Invalid cast |
| 5. | EnumConstantNotPresent Exception | An attempt is made to use an undefined enumeration value |
| 6. | IllegalArgumentException | Illegal argument used to invoke a method. |
| 7. | IllegalMonitorStateException | Illegal monitor operation, such as waiting on an unlocked thread. |
| 8. | IllegalStateException | Environment or application is in incorrect state. |
| 9. | IllegalThreadStateException | Requested operation not compat ible with the current thread state. |
| 10. | IndexOutOfBoundsException | Some type of index is out-of-bounds. |
| 11. | NegativeArraySizeException | Array created with a negative size. |
| 12. | NullPointerException | Invalid use of a null reference. |
| 13. | NumberFormatException | Invalid conversion of a string to a numeric format. |
| 14. | SecurityException | Attempt to violate security. |
| 15. | StringIndexOutOfBounds | Attempt to index outside the bounds of a string. |
| 16. | TypeNotPresentException | Type not found |
| 17. | UnsupportedOperationException | An unsupported operation was encountered. |

There is another type of exceptions called *checked exceptions.* These exceptions are also defined by java.lang but that must be included in a method's **throws** list if that method can generate one of these exceptions and does not handle it itself.

**Table 8.2: Checked Exception Defined in java.lang**

| Sl No | Exceptions | Descriptions |
|---|---|---|
| 1. | ClassNotFoundException | Class not found |
| 2. | CloneNotSupportedException | Attempt to clone an object that does not implement the Cloneableinterface |
| 3. | IllegalAccessException | Access to a class is denied |
| 4. | InstantiationException | Attempt to create an object of an abstract class or inter face |
| 5. | InterruptedException | One thread has been inter rupted by another thread. |
| 6. | NoSuchFieldException | A requested field does not exist |
| 7. | NoSuchMethodException | A requested method does not exist |

## 8.8 USER DEFINED EXCEPTION SUBCLASSES

Most common errors are handled by Java's built-in exceptions but sometimes when we are developing an application in java, we need to create and throw our own exceptions to handle specific situations to an application. These exceptions are known as **user defined or custom exceptions.** To do this we need to define a subclass of Exception which is a subclass of Throwable**.** An example for user defined exception has been given below:

**Program 8.5 : Program showing use of Exception sub-classes**

```
class  JavaException
{
        public  static  void  main(String  args[  ])
                {
                try{
                          throw  new  MyException(2);
           // throw is used to create a new exception and throw it.
        }
        catch(MyException  e)
                {
                System.out.println(e)  ;
        }
         }
}
class  MyException  extends  Exception
{
        int  a;
        MyException(int  b)
                {
                a=b;
        }
        public  String  toString()
                {
                return  ("Exception  Number  =   "+a)  ;
        }
}
```

**Output: Exception Number = 2**

The keyword **"throw"** is used to create a new Exception and throw it to the catch block.



### CHECK YOUR PROGRESS

Q4. Choose the appropriate option:

i. Which of these keywords is used to manually throw an exception?

a) try b)finally c) Throw d) catch

ii. Which of these keywords must be used to monitor for exceptions?

a) try b)finally c)Throw d)catch

iii. Which of these keywords is not a part of exception handling?

a) try b) finally c) thrown d) catch

iv. Which of these class is related to all the exceptions that can be caught by using catch?

a) Error b) Exception

c) RuntimeExecption d) All of the above

v. Which of these handles the exception when no catch is used?

a) Default handler b)finally

c) throw handler d)Java run time system

vi. Which of these keywords is used to manually throw an exception?

a) try b)finally c) throw d)catch



## 8.9 LET US SUM UP

- Exception is an event that arises during execution of a program.
- There are three types of exceptions viz. compile time exceptions, run time exceptions and error.
- Java's standard package java.lang defines exception classes.
- All exception classes are subclasses of the built-in class Throwable.
- try block is used to check the exceptions in a part of code of a program.

- Catch block is used to catch exception and handle it
- Code within a try - catch block is protected code
- throw keyword is used to manually throw an exception.
- Code under finally block must be executed before a method return.



## 8.10 FURTHER READINGS

1) The Complete Reference, Java 2, Tata McGraw-Hill Edition
2) www.java.sun.com



## 8.11 ANSWERS TO CHECK YOUR PROGRESS

**Ans. to Q. No. 1 :**Compile time exceptions, Runtime exceptions, Error

**Ans. to Q. No. 2 :**Runtime exceptions

**Ans. to Q. No. 3 :**Throwable

**Ans. to Q. No. 4 :**

i. c) throw
ii. a) try
iii. c) thrown
iv. b) exception
v. a) default handler
vi. c) throw



## 8.12 MODEL QUESTIONS

**Q1.** What are the five keyword used to manage exception handling?

**Q2.** List the classes that is related to exceptions that are explicitly thrown?

**Q3.** How are the exceptions handled in java?

**Q4.** Explain the exception hierarchy in java.

**Q5.** What is the use of throws keyword?

**Q6.** Write a program in Java that shows exception handling.

**********