UNIT 10: EXCEPTION HANDLING

UNIT STRUCTURE

10.1 Learning Objectives

10.2 Introduction

10.3 Exception Handling Fundamentals

10.3.1 Exception

10.3.2 Exception Types

10.4 Exception Handling Mechanism

10.5 Built-in and User-Defined Exception

10.6 Let Us Sum Up

10.7 Further Reading

10.8 Answer to Check Your Progress

10.9 Model Questions

10.1 LEARNING OBJECTIVES

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

- describe the concept of exception and its fundamentals
- describe the process of handling exceptions in the Java language
- use built-in and user-defined exceptions.

10.2 INTRODUCTION

Exception handling in Java is one of the powerful mechanisms to handle runtime errors so that the normal flow of the application can be maintained. In the last unit, we discussed java.lang.Math class and the various methods under this class. In this unit, you will learn about java exception, their various types, and their differences. Further, you will get the knowledge of how to handle the exception. In the next unit, we will explore topics related to file handling.

10.3 EXCEPTION HANDLING FUNDAMENTALS

10.3.1 Exception

According to Oxford Dictionary, exception means an abnormal condition. In Java, exception refers to an unexpected event that disrupts the normal flow of the program. An exception may occur during the execution of a program(at run time), to disrupt the normal flow of the program's instructions. It leads to the abnormal termination of the program, which is not always recommended. In Object Oriented languages like Java, exceptions are known as objects. They are used to pass information about unusual events to and from the different parts of the program.

An exception may occur due to the following reasons.

- Invalid user data input.
- Network connection may be disturbed in the middle of communications
- JVM may run out of memory.
- File cannot be found/opened.

Exception Hierarchy:

The java.lang.The exception class is the base class for all exception classes. Figure 1 presents the exception hierarchy. All exception and error types are subclasses of the class Throwable, which is the base class of the exception hierarchy. One branch is headed by Exception. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. In another branch, Errors are used by the Java virtual machine (JVM) to indicate errors having to do with the runtime environment itself(JRE). StackOverflowError is an example of such an error. Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated by the runtime environment.JVM out of memory is an example of an Error. Normally, programs cannot recover from errors.

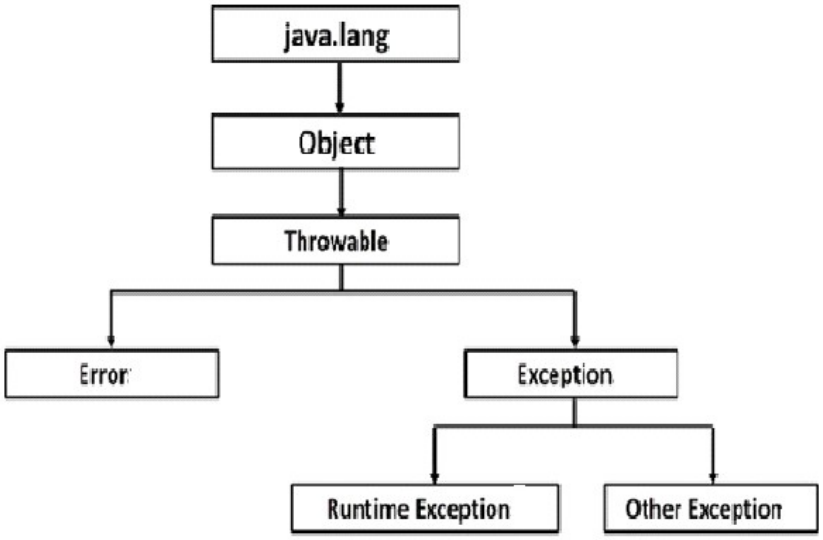


Figure10. 1: Exception Hierarchy

10.3.2 Exception Types

There are mainly three types of exceptions:

- Checked Exception
- Unchecked Exception
- Error
- **Checked Exception:**

These types of exceptions are checked by the compiler during the compilation to confirm whether the exception is handled by the programmer or not. The classes that extend the Throwable class except RuntimeException and Error are known as checked exceptions. Some checked exceptions include IOException, SQLException, etc. If some code within a method throws a checked exception, then the method must either handle the exception by using the throws keyword.

- **Unchecked Exception:**

Unchecked exceptions are not checked at compile-time rather they are checked at runtime. The classes that extend RuntimeException are known as unchecked exceptions such as ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.

- **Error:**

Errors are not exceptions, but these arise when the problem solution goes beyond the control of the user or the programmer. Fundamentally, an Error is used by the Java run-time system (JVM) to indicate errors that are associated with the run-time environment (JRE). Error is irretrievable. Some Errors include OutOfMemoryError, VirtualMachineError, AssertionError, etc.

10.4 EXCEPTION HANDLING MECHANISM

Exception Handling is a mechanism to deal with runtime errors such as ClassNotFound, ArrayIndexOutOfBound, etc. An exception is thrown when an error situation occurs during program execution. It is propagated by the JVM and caught by an exception handler that takes appropriate action to handle the situation. The core advantage of exception handling is to maintain the normal flow of the execution of an application. Exception normally disrupts the normal flow of the application due to which we use an exception handling mechanism. The exception handling mechanism works in a flow of execution which is depicted in figure2. If an exception is not handled at the proper time, it may lead to a system failure. That is why handling an exception is very considered important.



Figure 10.2. Workflow of Exception Handling

In Java, the following keywords are used for exception handling.

- try
- catch
- finally
- throw

- throws

Throwing and catching exceptions:

Catching Exceptions

A method catches an exception using a combination of the try and catch keywords. The program code that may generate an exception should be placed inside the try-catch block.

The try-catch block is used as follows:

```
try {
// block of code to monitor for errors
}
catch(ExceptionType1exOb)
{
// exception handler for ExceptionType1
}
catch(ExceptionType2exOb)
{
// exception handler for ExceptionType2
}
//optional
finally {
// block of code to be executed after try block ends
}
```

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception is handled by the catch block associated with it. Every try block should be immediately followed either by a catch block or a final block.

A catch statement involves declaring the type of exception which is tried to catch. If an exception occurs, then the catch block (or blocks) which follow the try block is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block similar to an argument that is passed into a method parameter.

```
classException_example{
publicstaticvoidmain(Stringargs[])
{
inta,b;
try { // monitor a block of code.
a = 0;
b = 10 / a; //raises the arithmetic exception
System.out.println("Tryblock.");
}
catch (ArithmeticExceptione)
{    //    catch    divide-by-zero    error
System.out.println("Exception found:Divisionbyzero.");
}
System.out.println("Aftertry/catchblock.");
}
}
```

Output:

Exception found: Division by zero.

Aftertry/catch block.

Multiple catch clauses:

In some cases, more than one exception could be raised by a single piece of code. To handle this multiple exception, two or more catch clauses can be specified. Here, each catch block catches different types of exceptions. When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed. After one catch statement executes, the others are by passed, and execution continues after the try/catch block. The following example traps two different exception types:

```
classMultiCatch{
publicstaticvoidmain(Stringargs[]){
try{
intx, y;
```

```
x = args.length; System.out.println("x="+x);
y=10/x;//maycause division-by-zeroerror
intarr[] = { 10,20 };
c[5] =100;
}
catch(ArithmeticExceptione)
{
System.out.println("Divideby0:"+ e);
}
catch(ArrayIndexOutOfBoundsExceptione)
{
System.out.println("Arrayindexoob:"+e);
}
System.out.println("Aftertry/catchblocks.");
}
}
```

Outputs generated by theprogram when executed in both ways:

C:\>javaMultiCatch

x = 0

Divideby0:java.lang.ArithmeticException:/byzero

After try/catch blocks.

C:\>javaMultiCatcharg1

x = 1

Array index oob:java.lang.ArrayIndexOutOfBoundsException:5

After try/catch blocks.

While multiple catch statements are used, it is important to remember that exception subclasses must come before their superclasses. A catch statement that uses a superclass will catch exceptions of that type plus any of its subclasses. Thus, a subclass would never be reached fit came after its superclass. Moreover, unreachable code in Java encounters an error. For example, consider the following program:

```
classMultiCatch{
publicstaticvoidmain(Stringargs[]){
try{
inta,b;
a = args.length; System.out.println("a="+a);
b=10/a;//maycausedivision-by-zeroerror intarr[] =
{ 10,20 };
c[5] =100;
}

catch(Exception e) {
System.out.println("GenericExceptioncatch.");
}
catch(ArithmeticExceptione)
{
System.out.println("Divideby0:"+ e);
}
catch(ArrayIndexOutOfBoundsExceptione)
{
System.out.println("Arrayindexoob:"+e);
}
      System.out.println("Aftertry/catchblocks.");
}
}
```

Nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested in the following way.

```
try
{
statement1;
```

```
statement2;
try
{
statement1;
statement2;
}
catch(Exceptione)
{
}
}
catch(Exceptione){}....
```

Throw keyword

The throw keyword is used to explicitly throw an exception. The general form of throw is:

```
throwThrowableInstance;
```

Here, Throwable Instance must be an object of type Throwable class or a subclass of Throwable. Primitive types,such as int or char,as well as nonThrow able classes, such as String and Object, cannot be used as exceptions.

There are two ways to obtain a Throwable object:

- using a parameter in a catch clause.
- creating one with the new operator.

The following program explains the use of throw keyword.

```
public class TestThrow1
{ staticvoidvalidate(intage){
try{
if(age<18)
thrownewArithmeticException("notvalid");
else
System.out.println("Eligibletovote");
}
catch(ArithmeticExceptione)
```

```
{
System.out.println("CaughtinsideArithmeticExceptions.");
throw e; // rethrow the exception
}
}
publicstaticvoidmain(Stringargs[]){
try{
validate(13);
}
catch(ArithmeticExceptione)
{
System.out.println("ReCaughtArithmeticExceptions.");
}
}
}
```

The flow of execution stops immediately after the throw statement and any subsequent statements that are not executed.The nearest try block is employed to see if it has a catch statement that matches the type of exception. If it finds a match, control is trans ferred to that statement. If not, then the next enclosing try statement is inspected, and so on. If no matching catch is found, then the default exception handle rhalts the programand prints the stack trace.

Throws keyword

The throws keyword is used to specify that a method throws an exception but does not handle it, it leaves it up to the caller of the method to handle the exception. In such case, the throws keyword can be specified in the method declaration along with the name of the exception that the method throws. If a method throws a checked exception, then it must either be caught and handled by the method or declared in the throws clause.Otherwise, a compilation error occurs. Unchecked exceptions need not be specified in the throws clause.

For example:

```
public class LoadClass {
public void load Class (String class Name) throws
Class Not Found Exception
{
  Class.forName(className);
 }
}

public class Main {
 public static void main(String[] args) {
  LoadClasslc = new LoadClass();
  try {
   lc.loadClass("com.MissingClass");
  }
catch (ClassNotFoundException e)
{
      System.out.println("This class is
notpresent:"+e.getMessage());
  }
}
}
```

Output:

This class is not present:com.MissingClass

The Finally Block

The final block follows a try block or a catch block. A final block of code always executes, irrespective of the occurrence of an Exception. A final block appears at the end of the catch blocks that follow the syntax as given below.

Syntax:

```
try {
//Protected code
```

```
}catch(ExceptionType1e1){// Catch block}
catch(ExceptionType2e2){// Catch block}
finally{
//Thefinallyblockalwaysexecutes.
}
```

Example:

```
publicclassFinally_Example{
publicstaticvoidmain(Stringargs[]){
try {
inta,b;
a=0;
b=10/a;
}
catch (ArithmeticException e) {
System.out.println("Exceptionthrown:"+e);
}
finally{
System.out.println("Thefinallyblockisexecuted");
}
}
}
```

Points to remember:

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.·

  The try block cannot be present withoute it her catch clause or finally clause.

  Any code cannot be present in between the try, catch, finally blocks.

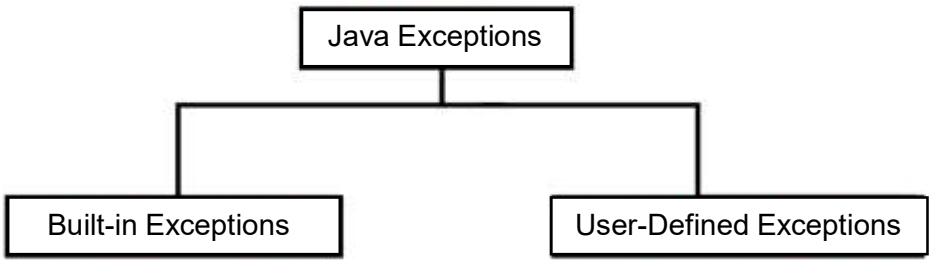
  Keywords used for exception handling-
- try block contains program statements that are to be to monitored for exceptions. If an exception occurs within the try block, it is thrown.
- catch block contain statements that catch this exception and handle it in some rational manner – System-generated exceptions are

automatically thrown by the Java runtime system.

- throw keyword is used to manually throw an exception.
- throws keyword is used to specify any exception that is thrown out of a method.
- finally block contains code that must be executed after a try block completes.

10.5 BUILT-IN AND USER DEFINED EXCEPTIONS

Java defines several types of built-in exceptions that relate to its various class libraries. Also, Java allows users to define their own exceptions.



Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations. Some important built-in exceptions in Java are stated in the following. catch block contain statements that catch this exception and handle it.

- **ArithmeticException:** It is thrown when an exceptional condition has occurred in an arithmetic operation.
- **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.
- **ClassNotFoundException:** This Exception is raised when we try to access a class whose definition is not found.
- **FileNotFoundException:** This Exception is raised when a file is not accessible or does not open.
- **IOException:** It is thrown when an input-output operation failed or interrupted.
- **InterruptedException:** It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

- **NoSuchFieldException:** It is thrown when a class does not contain the field (or variable) specified.
- **NoSuchMethodException:** It is thrown when accessing a method that is not found.
- **NullPointerException:** This exception is raised when referring to the members of a null object. Null represents nothing.
- **NumberFormatException:** This exception is raised when a method could not convert a string into a numeric format.
- **RuntimeException:** This represents an exception that occurs during runtime.
- **StringIndexOutOfBoundsException:** It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string.
- **IllegalArgumentException:** This exception will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.
- **IllegalStateException:** This exception will throw an error or error message when the method is not accessed for the particular operation in the application. It belongs to the unchecked exception.

Examples of using some Built-in Exceptions:

Example 10.1: Java program to demonstrate Aritmetic Exception

```
// 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

Example 10.2: Java program to demonstrate **StringIndexOutOfBoundException**

```
// Java program to demonstrate
//StringIndexOutOfBoundsException
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

Example 10.3 Java program to demonstrate **ArrayIndexOutOfBoundsException**

```
// 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

User-Defined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, the user can create exceptions which are called 'user-defined Exceptions'. Java allows the user to create their exception class which is derived from the built-in class Exception. The user-defined exception can be thrown using the throw keyword.

Syntax to define a user-defined exception:

```
class User_defined_name extends Exception
  {
            ……..
}
```

- We can write a default constructor in its 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. We can call the superclass(Exception) constructor from this and send a string.

```
MyException(String str)
{  super(str);}
```

- To raise an exception of a user-defined type, we need to create an object to his exception class and throw it using the throw clauseas given here:

```
MyException me = new MyException("Exception
details");
            throw me;
```

The following program explains how to create your own exception class MyException.

Example 10.4: Java program to demonstrate a user-defined exception. This program throws an exception whenever the balance amount is below Rs 1000

```
class MyException extends Exception
{
    //store account information
    private static int accno[] = {1001, 1002, 1003,
1004};
    private static String name[] =
            {"Nish", "Shubh", "Sush", "Abhi",
"Akash"};
    private static double bal[] =
         {10000.00, 12000.00, 5600.0, 999.00,
1100.55};
    // default constructor
    MyException() {    }
     // parameterized constructor
    MyException(String str) { super(str); }
```

```
      // write main()
    public static void main(String[] args)
    {
        try  {
            // display the heading for the table
            System.out.println("ACCNO"+ "\t"+
"CUSTOMER"+
         "\t"+ "BALANCE");

            // display the actual account information
            for(int i = 0; i<5;i++)
            {
                System.out.println(accno[i] + "\t"+
name[i] +
          "\t"+ bal[i]);

          // display own exception if balance < 1000
                if(bal[i] < 1000)
                {
                MyException me =
           new MyException("Balance is less than
1000");
                    throw me;
                }
            }
        } //end of try

        catch(MyException e) {
            e.printStackTrace();
        }
    }
}
```

Output: Runtime Error

MyException: Balance is less than 1000

at MyException.main(fileProperty.java:36)



CHECK YOUR PROGRESS

Q 1. When does Exceptions in Java arises in code sequence?

a) Run Time b) Compilation Time c) Can Occur Any Time

d) None of the mentioned

Q 2. Which of these keywords is not a part of exception handling?

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

Q 3. Which of these keywords must be used to handle the exception thrown by try block in some rational manner?

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

Q 4. The class at the top of exception class hierarchy is _____

a) ArithmeticException b) Throwable c) Object

d) Exception

Q 5. Which keyword is used to explicitly throw an exception?

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

Q 6. Which exception is thrown when divide by zero executes?

a) NumberFormatException b) ArithmeticException

c) NullPointerException d) None of these

Q 7. Consider the following program.

```
public class Test {
public static void main (String args[])
{  try  { int a=Integer.parseInt("One");  } }
}
```

Which exception to be handled by catch block in the above code?

a) IllegalStateException b) NumberFormatException

c) ArrayIndexOutOfBoundsException d) None of these

Q 8. What is the output of the following code with exception?

public class exception_test

```
{
public static void main(String args[]) throws
ArithmeticException
{
System.out.println ("We are the learners");
throw new  ArithmeticException();
}
}
```

a) We are the learners b) No output c) Compile time error
d) None of these

Q 9. Exception class is found in which package in java?

a) java.util b) java.io c) java.lang d) java.net

Q 10. Which block is always executed independently of an exception being raised?

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



10.6 LET US SUM UP

- An exception refers to an unexpected event that disrupts the normal flow of the program.
- An exception may occur during the execution of a program (at run time), to disrupt the normal flow of the program's instructions. It leads to the abnormal termination of the program, which is not always recommended. In Object Oriented languages like Java, exceptions are known as objects.
- An exception may occur due to the following reasons.
  - Invalid user data input.
  - Network connection may be disturbed in the middle of communications.
  - JVM may run out of memory.
  - File cannot be found/opened.
- The java.lang.Exception class is the base class for all exception classes. All exception and errors types are sub classes of class Throwable, which

is the base class of exception hierarchy.

- There are mainly three types of exceptions: Checked Exception, Unchecked Exception and Error.
- Checked Exception are checked by the compiler during the compilation to confirm whether the exception is handled by the programmer or not.
- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.
- Unchecked Exception are not checked at compile-time rather they are checked at runtime. The classes that extend RuntimeException are known as unchecked exceptions such as ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- Errors are not actually exceptions, but these arise when problem solution goes beyond the control of the user or the programmer. Fundamentally, an Error is used by the Java run-time system (JVM) to indicate errors that are associated with the run-time environment (JRE). Error is irretrievable. Some Errors include Out Of Memory Error, Virtual Machine Error, Assertion Error, etc.
- Exception Handling is a mechanism to deal with runtime errors such as Class Not Found, Array Index Out Of Bound etc.
- An exception is thrown when an error situation occurs during program execution. It is propagated by the JVM and caught by an exception handler that takes an appropriate action to handle the situation.
- Java provides five keywords to handle the exception. Those are try, catch, throw, throws and finally.
- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause. Any code can not be presentin between the try, catch, finally blocks.
- Java defines several types of built-in exceptions that relate to its various class libraries. Further, Java also allows users to define their own exceptions.



10.7 FURTHER READING

1) Schildt, H. (2014). Java: The Complete Reference, Oracle Press.
2) Rao, R. N. (2006). Core and Advanced Java Black Book, Dreamtech Press



10.8 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. a

Ans to Q No 2. c

Ans to Q No 3. d

Ans to Q No 4. b

Ans to Q No 5. d

Ans to Q No 6. b

Ans to Q No 7. b

Ans to Q No 8. a

Ans to Q No 9. c

Ans to Q No 10. b



10.9 MODEL QUESTIONS

Q 1. What do you mean by exception? Discuss various types of exceptions.

Q 2. How can we handle exception? Name various keywords used in Java to handle the exception.

Q 3. What is the difference between exception and error?

Q 4. Describe the use of the throw keyword with an example.

Q 5. Is there any difference between throw and throws in exception handling in Java? If yes, discuss it with suitable example.

Q 6. Define user-defined or custom exceptions in Java with proper example.

*** ***** ***