Unit 3: Introduction of Exception 3

Unit Structure

3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Study various types of errors while programming.
- Understand need of Error handling in java.
- Study various mechanisms to handle error and exceptions.
- Understand the types of exception
- Use exception handling mechanism using try ... catch, try with multiple catch and nested catch.

3.2 INTRODUCTION

An exception is an unwanted or unexpected event occurs during the execution of the program. Exception occurs at run time which disturbs the flow of execution program instructions. The java program terminates abnormally due to exception. It is not recommended therefore these exceptions are to be handled in our program. These exceptions are caused by error in data input, by programmer error, and by physical resources that have failed during execution of program.

3.3 ERROR

Error is unexpected event occur which stops program from compiling or executing. The programmer should know that there are very less chances that a program will run perfectly in first attempt. Though programmer has did nice designing and proper care has been taken while coding we can never predict the execution of program error free. The programmer must perform systematic effort to detect and rectify the errors present in the program. For this effort all programmer should know what types of error may present in the program.

3.3.1 TYPES OF ERRORS IN PROGRAMMING

The error can be classified into four categories as listed below.

1. syntax errors
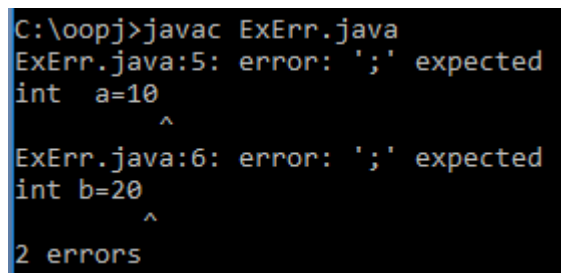2. logical errors
3. run-time errors

4. latent errors

➤   Syntax Errors

Each programming language has a rules to write a program. The violation of these rules and poor understanding of the programming language results in syntax errors. The syntax errors are detected by the compiler. If program has any syntax error compilation of program fails and it lists the syntax error with line number where syntax error is occurred.

For example,

```
public class ExErr
{
        public static void main ( String args[] )
        {
        int  a=10
        int b=20
        System. out. println  ( a + b );
        }
}
```

In above program line 5 and 6 of the program doesn't have semicolon at the end hence the compiler will show us errors.

```
C:\oopj>javac ExErr.java
ExErr.java:5: error: ';' expected
int   a=10
          ^
ExErr.java:6: error: ';' expected
int b=20
        ^
2 errors
```
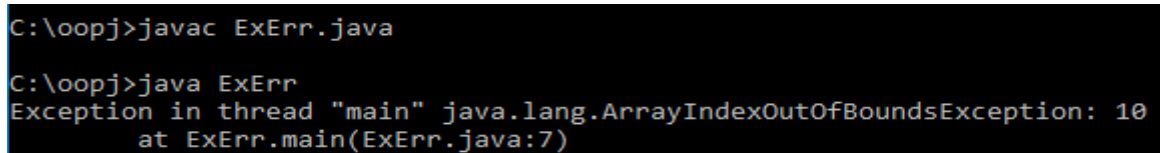
Figure-67 Output of program

➤   Run-time Errors

These error are not detected by compiler. They are the errors that occur during the execution of the program. For example, dividing by zero error, insufficient memory for dynamic memory allocation, referencing an out-of-range array element etc. A program with these kinds of errors will run but produce erroneous results or

may cause abnormal termination of program. Detection and removal of a run-time error is a very difficult task.

```
public class ExErr
{
       public static void main ( String args[] ) throws Exception
       {
       int  a[] = { 10, 23, 85, 52 };
       System. out. println  ( a[10] );
       }
}
```

```
C:\oopj>javac ExErr.java

C:\oopj>java ExErr
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at ExErr.main(ExErr.java:7)
```

Figure-68 Output of program

➢   Logical Errors

These errors are related to the logic of the program. Logical errors are also not detected by compiler and cause incorrect results. These errors occur due to incorrect translation of algorithm into the program, poor understanding of the problem and a lack of clarity of hierarchy of operators. Logic errors occur when there is a design flaw in your program. Common examples are:

- Multiplying when you should be dividing
- Adding when you should be subtracting
- Opening and using data from the wrong file
- Displaying the wrong message
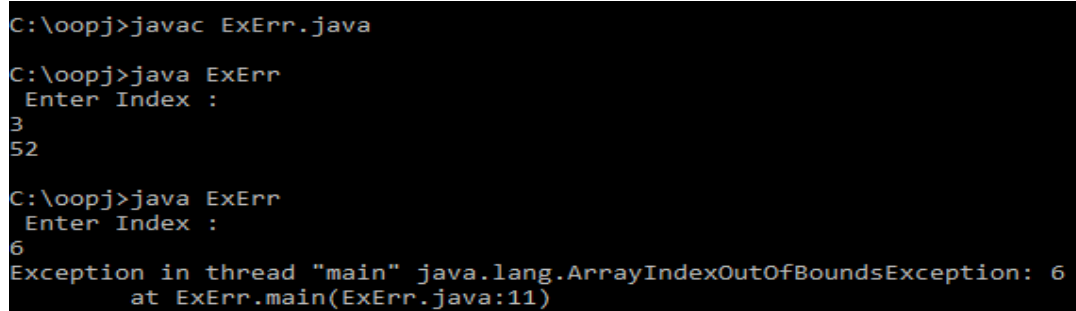
➢   Latent Errors

Latent Errors are the 'hidden' errors that occur only when a particular set of data is used. Such errors can be detected only by using all possible combinations of data.

For example,

```
import java.util.Scanner;
public class ExErr
{
        public static void main ( String args[] )
        {
        int  a[] = { 10, 23, 85, 52 };
        System. out. println  (" Enter Index : ");
        Scanner sc = new Scanner ( System.in );
        int i = sc.nextInt();
        System. out. println  ( a[i] );
        }
}
```


Figure-69 Output of program

An error occurs only when we input value of i more than 4.

3.4 HIERARCHY OF EXCEPTION AND ERROR CLASS

All exception and errors types are sub classes of class Throwable, which is base class of the hierarchy. The class Exception is a subclass of Throwable class. This class is used for exceptional conditions that user programs should catch. Mainly they are used to handle runtime error, logical errors and latent errors. The NullPointerException is an example of such an exception which is a subclass of Exception class. The class Error is also derived from Throwable class which is used by the Java virtual machine (JVM) to indicate errors. StackOverflowError is an example of such an error class which is derived from the Error class.
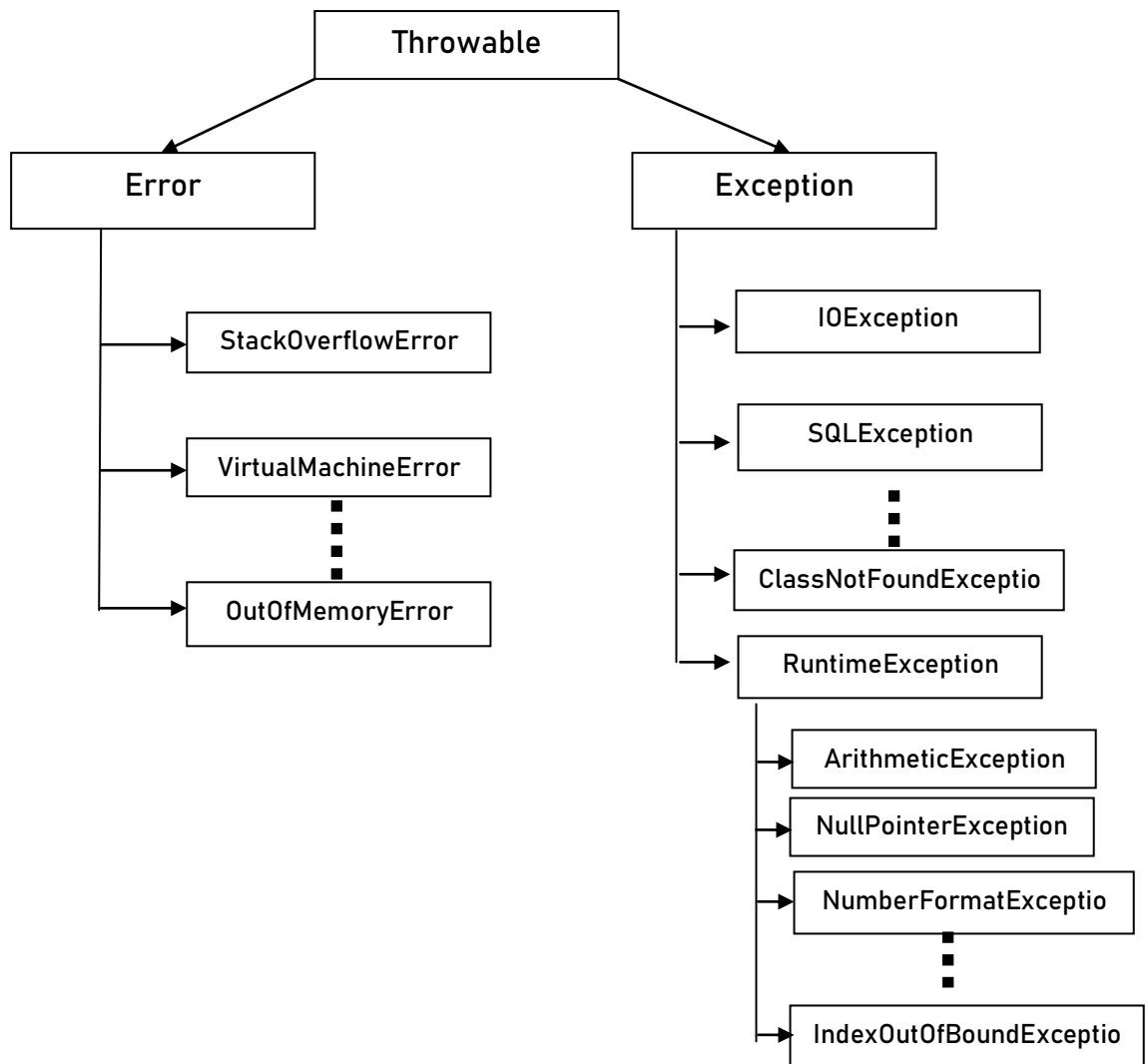
Figure-70 Hierarchy of exception class and error class

3.5 TYPES OF EXCEPTIONS

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception.

1) Checked Exception

All the classes which extend the Throwable class except RuntimeException and Error are known as checked exceptions for example, IOException, SQLException etc. Checked exceptions are checked and raised at compile-time. The check exceptions are forced to be checked and handled using try…catch block or declare it in function header using throws keyword in java program.

```
import java.io.File;

import java.io.FileReader;

public class ExErr {

   public static void main(String args[]) {

      File file = new File ( "E://file.txt" );

      FileReader fr = new FileReader ( file );

   }

}
```

```
C:\oopj>javac ExErr.java
ExErr.java:8: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
    FileReader fr = new FileReader(file);
                    ^
1 error
```

Figure-71 Output of program

The Compile time error as we have not handled check exception FileNotFoundException in our program. The error can be solved using following code,

```
import java.io.File;

import java.io.FileReader;

public class ExErr {

   public static void main(String args[]) throws Exception {

      File file = new File("E://file.txt");

      FileReader fr = new FileReader(file);

   }

}
```

```
C:\oopj>javac ExErr.java

C:\oopj>java ExErr
 file is found
```
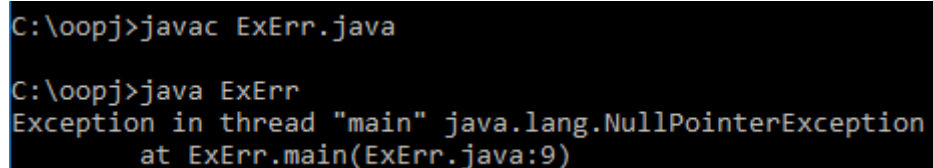
Figure-72 Output of program

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions. For example ArithmeticException, NumberFormatException, NullPointerException, ArrayIndexOutOfBoundsException etc. The unchecked exceptions are not checked at compile-time, but they are checked at runtime. These exceptions handle the unrecoverable programming errors.

For example,

```
import java.util.Scanner;
public class ExErr
{
public static void main ( String args[] )
{
String x = "abc ";
String s= null;
String c = x + s.length();
System. out. println  ( c );
}
}
```


Figure-73 Output of program

3.6 UNCAUGHT EXCEPTION

Java provides a strong built in exception handling mechanism. It has a list of exception classes derived from Exception class. The exception is raised when any error occurred which further throws errors in form of appropriate Exception class object. The main issue of this mechanism is that it terminates the program execution from the line where error found. The program code after that error will not be executed. For example, in following code the program will compiled successfully. When we execute the program it will ask for value of a and b. Here a and b should
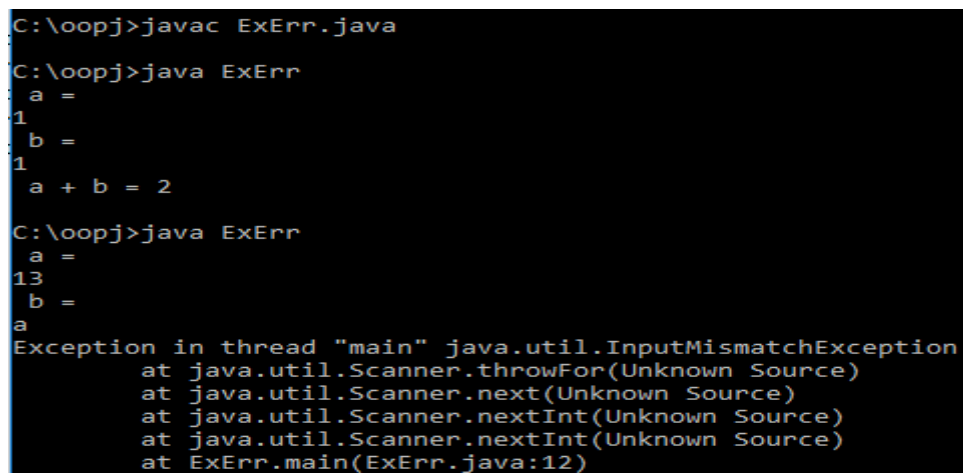
208

be integer value only. If we enter integer value for a and b, the program executes successfully and output will print summation of them. However when we enter character value for either a or b, at that point of time the run time error is raised and the object of InputMismatchExcpetion is thrown which prints an error message and terminate program execution.

```java
import java.util.Scanner;

public class ExErr {

  public static void main(String args[]) throws Exception {
    int a;
    int b;
    Scanner sc = new Scanner ( System.in );
    System. out. println  ( " a = ");
    a = sc.nextInt();
    System. out. println  ( " b = ");
    b = sc.nextInt();
    System. out. println  ( " a + b = " + ( a + b ));
  }
}
```

```
C:\oopj>javac ExErr.java

C:\oopj>java ExErr
 a =
1
 b =
1
 a + b = 2

C:\oopj>java ExErr
 a =
13
 b =
a
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at ExErr.main(ExErr.java:12)
```

Figure-74 Output of program

3.7 HANDLING EXCEPTION

The JVM automatically handle the unchecked Exception if we have not handled the in java program. This will print a system generated error message along with Exception class name. If we want to handle exception in our own way by printing our own error message, we can do that by using exception handling mechanism in java program.

The keywords try, catch and finally are used to handle exception in any java program. The try block can be used with either catch block or finally block. The syntax of using them in program is given below,

try {

// program logic

}

catch (Exception_Class obj) {

 // custom error message

// this block executes only when error occurred in program logic of the try block

}

finally {

// the code in finally will always be executed

}

We can not use try block without either catch or finally. It will give compilation error if we use try block only.

Example,

```
import java.util.Scanner;
public class ExErr1 {

    public static void main(String args[]) throws Exception {
    int a = 0;
    int b = 0;
    try {
    Scanner sc = new Scanner ( System.in );
    System. out. println  ( " a = ");
    a = sc.nextInt();
```

```
        System. out. println  ( " b = ");

      b = sc.nextInt();

      }

   }

}
```

```
C:\oopj>javac ExErr1.java
ExErr1.java:10: error: 'try' without 'catch', 'finally' or resource declarations
     try {
     ^
1 error
```

Figure-75 Output of program

The following example shows use of try block with catch block, finally block and with both catch and finally block.

Example 1 (try block with catch block)

```
import java.util.Scanner;

import java.util.InputMismatchException;

public class ExErr1 {

    public static void main(String args[]) throws Exception {

      int a = 0;

      int b = 0;

      try {

      Scanner sc = new Scanner ( System.in );

        System. out. println  ( " a = ");

      a = sc.nextInt();

        System. out. println  ( " b = ");

      b = sc.nextInt();

    System. out. println  ( " a + b = " + ( a + b ));

      } catch (InputMismatchException e ) {


          System. out. println  ( "Error occured as the value entered is a character ");

        }

    }

}
```

Figure-76 Output of program

The code which may raise an exception must be put in try block. When an error occurred during execution of program the try block throws an exception which will be caught in catch block. In catch block we can write a code to handle exception. In above example, we have printed a user message when exception is raised.

Example 2 (try block with finally block)

```java
import java.util.Scanner;
import java.util.InputMismatchException;

public class ExErr1 {

    public static void main(String args[]) throws Exception {
    int a = 0;
    int b = 0;
    try {
    Scanner sc = new Scanner ( System.in );
    System. out. println  ( " a = ");
    a = sc.nextInt();
    System. out. println  ( " b = ");
    b = sc.nextInt();
    } finally {
    System. out. println  ( " a + b = " + ( a + b ));
    }
  }
}
```

Figure-77 Output of program

In above example we have put code which may raise exception in try block. During program execution as we entered character for integer input an exception is raised at that statement of program. As we have not written catch block the exception will be handle by JVM, which prints an error message. After printing error message program will not be terminated. It executes the finally block which prints sum of a and b.

Example 3 (try block with both catch and finally block)

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class ExErr {

  public static void main(String args[]) throws Exception {
    int a = 0;
    int b = 0;
    try {
    Scanner sc = new Scanner ( System.in );
    System. out. println  ( " a = ");
    a = sc.nextInt();
    System. out. println  ( " b = ");
    b = sc.nextInt();
    }
catch (InputMismatchException e ) {
```

```
   System. out. println  ( "Error occured as the value entered is a character ");
  } finally {


  System. out. println  ( " a + b = " + ( a + b ));
  }
 }
}
```


Figure-78 Output of program

The code which may have possibility of error can be put in try block and when error occurred in try block the appropriate exception object will be thrown. This thrown object will be catch in catch block of the program (Here in above example as e object). We can handle exception in catch block by writing our own code segment. Here in above example we have printed our own error message. The finally block contains the code which must be executed in any way. If exception raised due to error, the program control will move to catch and then finally block. If exception is not raised, after executing try block program control moves to the finally block.

3.8 TRY WITH MULTIPLE CATCH

When in a try block there is only one possible error, we may handle that error by writing catch block with appropriate exception. However it may possible that our

try block may raise more than one exception during execution of different code statement. To handle such situation java allow us to write multiple catch block for single try block. Each catch block will handle the appropriate exception.

For example,

```java
import java.util.Scanner;
import java.util.InputMismatchException;

public class ExErr {

  public static void main(String args[]) throws Exception {
    int a[] = { 3, 4, 5, 6, 7, 8};
    int b = 0, i=0;

    try {
    Scanner sc = new Scanner ( System.in );
    System. out. println  ( " index = ");
    i = sc.nextInt();
    System. out. println  ( " a[i] = " +  a[i]);

    System. out. println  ( " b = ");
    b = sc.nextInt();
    System. out. println  ( " b = " +  b);
    }
        catch (InputMismatchException e ) {

        System. out. println  ( "Error occured as the value entered is a character ");
      }
        catch (ArrayIndexOutOfBoundsException e ) {

        System. out. println  ( "Error occured as the value of i is >=6 ");
    }
  }
}
```
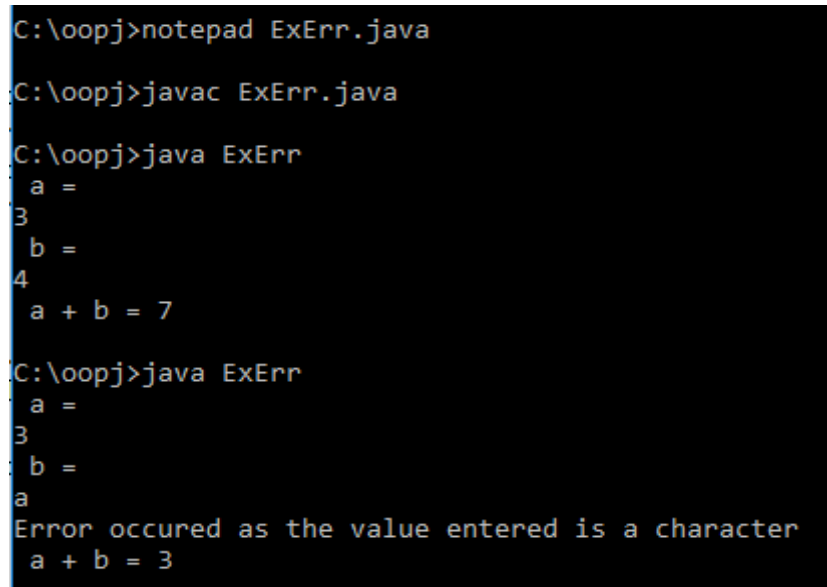
Figure-79 Output of program

In above example, the line 8 in main method may raise ArrayIndexOutOfBoundsException if the value of entered i is > = 6. The line 11 of main method will raise InputMismatchException if the entered value for b is non integer. To handle these two exceptions of try block we have written two catch blocks, one for handling each exception.

3.9 NESTED TRY...CATCH...FINALLY BLOCK

Like loops and if...else statement, try ... catch...finally block can also be nested. We can write a try...catch...finally block inside the try...catch... finally block. We can use this concept in our program when within try block we may have some program statements which causes an error and the other program statements cause the other error and we want to handle both errors independently. When we use nested try...catch block the inner block will be executed first.

For example,

```
import java.util.Scanner;
import java.util.InputMismatchException;
```

```java
public class ExErr {


  public static void main(String args[]) throws Exception {
    int a[] = { 3, 4, 5, 6, 7, 8};
    int b = 0, i=0;


    try {
    Scanner sc = new Scanner ( System.in );
    System. out. println  ( " index = ");
    i = sc.nextInt();
    System. out. println  ( " a[i] = " +  a[i]);
        try{
        System. out. println  ( " b = ");
        b = sc.nextInt();
        System. out. println  ( " b = " +  b);
        }
        catch (InputMismatchException e ) {


         System. out. println  ( "Error occured as the value entered is a character ");
      }
    }
        catch (ArrayIndexOutOfBoundsException e ) {


        System. out. println  ( "Error occured as the value of i is >=6 ");
    }
  }
}
```

Figure-80 Output of program

3.10 LET US SUM UP

Error : Error is something unexpected in your program which stop execution of the program.

Syntax Error : They are the design time error which is due to mistake done by programmer. They are detected at compile time.

Logical Error : these errors occur due to mistake in program logic. These errors occur when the output of the program is not as per the programmer expectation.

Runtime Error : They are not detected by compiler. They occur at runtime due to unexpected input or failure. Java handles run time error using exception.

Exception : Java handles the run time error using exception handling mechanism.

Checked Exception : The checked exception are the exception classes which are derived from Exception class. Checked exception is raised at compile time. These exceptions must be handled in program.

Unchecked Exception : These exceptions are raised at run time. All the exception classes derived from RuntimeException class are of unchecked type.

Uncaught Exception : The JVM will automatically handle the exception raised at run time (unchecked exception) . For handling checked exception programmer has to use try ... catch ... finally blocks or throws keyword.

Exception handling using try ... catch ... finally: The code which may have possibility of error can be put in try block and when error occurred in try block the appropriate exception object will be thrown. This thrown object will be catch in catch block of the program. The finally block always runs by the program. We can not use try block without catch or finally block.

Try with multiple catch : With a single try block we can write multiple catch block in our java program. One catch block for each Exception we want to handle.

Nested try ... catch ... finally : It is also possible to write try ... catch ... finally block within a try ... catch ... finally. It is called nested try ... catch ... finally.

3.11 CHECK YOUR PROGRESS

➢ True-False with reason

1. We can not handle errors in java program.
2. Syntax error will be caught at run time.
3. Runtime error will be caught by compiler.
4. Compiler can detect syntax error only.
5. Try can be used either with catch or finally block.
6. We can not write more than one catch block with try block.
7. Finally block will be run even though the exception is raised.
8. Nested try … catch is not supported in java.
9. Checked exception must be handled by programmer in java program.
10. Uncheck exception must be handled by programmer in java program.

➢ Match **A** and **B**.

A	B
1)Exception	a)unexpected event
2)Error	b) try … catch within try … catch
3)Checked Exception	c)must be handled by programmer

4)Unchecked Exception   d)handled by JVM

5)Nested try … catch   e)error which can be handle at run time

➢ Compare the following:

1. Error and Exception
2. Checked Exception and Unchecked Exception
3. Catch block and finally block
4. Syntax error and runtime error

➢ MCQ

1. Exception is a class/interface/abstract class/other?

a. Class   c. Abstract class

b. Interface   d. Other

2. Exception and Error are direct subclasses of?

a. BaseException   c. Object

b. Throwable   d. RuntimeException

3. Which of these are java.lang.Error in exception handling in java

a. VirtualMachineError   d. ThreadDeath

b. IOError   e. All

c. AssertionError

4. What type of Exceptions can be ignored at compile time?

a. Runtime   c. Both

b. Checked   d. None

5. What will be output of following program –
    public class ExceptionTest {
            public static void main (String args[]) {
                    System. out. println (“ method return -> “ + m());
            }
            static String m( ) {
                    try {
                            int i = 10 / 0;

```
                    } catch ( ArithmeticException e )
                        {        return "catch"; }
                      finally
                        { return "finally"; }
                }
        }
```

a. runtime exception   c. method return -> catch

b. method return -> finally   d. compile timeError

6. Which of the following are the most common run-time errors in Java programming?

i) Missing semicolons

ii) Dividing an integer by zero

iii) Converting invalid string to number

iv) Bad reference of objects

a) i and ii only   c) iii and iv only

b) ii and iii only   d) i and iv only

7. Which of the following are the most common compile time errors in Java programming?

i) Missing semicolons

ii) Use of undeclared variables

iii) Attempting to use a negative size for an array

iv) Bad reference of objects

a) i, ii and iii only   c) i, ii and iv only

b) ii, iii and iv only   d) All i, ii, iii and iv

8. The unexpected situations that may occur during program execution are

i) Running out of memory

ii) Resource allocation errors

iii) Inability to find a file

iv) Problems in network

a) i, ii and iii only   c) i, ii and iv only

b) ii, iii and iv only   d) All i, ii, iii and iv

9. The class at the top of the exception classes hierarchy is called
……………………

a) throwable   c) hierarchical

b) catchable   d) ArrayIndexOutofBounds

10. ………………… exception is thrown when an exceptional arithmetic condition
has occurred.

a) Numerical   c) Mathematical

b) Arithmetic   d) All of the above

11 …………………….. exception is thrown when an attempt is made to access
an array element beyond the index of the array.

a) Throwable   c) Security

b) Restricted   d) ArrayIndexOutofBounds

12. You can implement exception-handling in your program by using which of the
following keywords.

i) Try   ii) NestTry   iii) Catch   iv) Finally

a) i, ii and iii only   c) i, iii and iv only

b) ii, iii and iv only   d) All i, ii, iii and iv

13. When a ……………………. block is defined, this is guaranteed to execute,
regardless of whether or not in exception is thrown.

a) throw   c) finally

b) catch   d) try

14. Every try statement should be followed by at least one catch statement;
otherwise …………………. will occur.

a) no execution   c) zero

b) null   d) compilation error

15. If an exception occurs within the …………………….. block, the appropriate exception-handler that is associated with the try block handles the exception.

a) throw

c) finally

b) catch

d) try

16 Exception classes are available in the ……………………package.

a) java.lang

c) java.io

b) java.awt

d) java.applet

17 Consider the following code snippet:

………………

………………

try {

int x = 0;

int y = 50 / x;

System. out. println ("Division by zero");

}

catch(ArithmeticException e) {

System. out. println ("catch block");

}

………………

………………

What will be the output?

a) Error.

c) Catch block

b) Division by zero

d) Division by zero Catch block

18 When an exception in a try block is generated, the Java treats the multiple ………………. statements like cases in switch statement.

a) throw

c) finally

b) catch

d) try

19. The ………………… statement can be used to handle an exception that is not caught by any of the previous catch statement.

a) throw

c) finally

b) catch

d) try

20. What will be the output of the program?

```
public class Foo
{
    public static void main(String[] args)
    {
        try
        {
            return;
        }
        finally
        {
            System. out. println ( "Finally" );
        }
    }
}
```

a. Finally

b. Compilation fails.

c. The code runs with no output.

d. An exception is thrown at
    runtime

21 What will be the output of the program?

```
try
{
    int x = 0;
    int y = 5 / x;
}
catch (Exception e)
{
    System. out. println ("Exception");
}
catch (ArithmeticException ae)
{
    System. out. println (" Arithmetic Exception");
}
System. out. println ("finished");
```

a) finished

b) Exception

c) Compilation fails.

d) Arithmetic Exception

22. What will be the output of the program?

```
public class X
{
    public static void main(String [] args)
    {
        try
```

```
        {
            badMethod();
            System.out.print("A");
        }
        catch (Exception ex)
        {
            System.out.print("B");
        }
        finally
        {
            System.out.print("C");
        }
        System.out.print("D");
    }
    public static void badMethod() {}
}
```

a) AC   c) ACD

b) BC   d) ABCD

23 What will be the output of the program?

```
    public class X
    {
        public static void main(String [] args)
        {
            try
            {
                badMethod(); /* Line 7 */
                System.out.print("A");
            }
            catch (Exception ex) /* Line 10 */
            {
                System.out.print("B"); /* Line 12 */
            }
            finally /* Line 14 */
            {
                System.out.print("C"); /* Line 16 */
            }
            System.out.print("D"); /* Line 18 */
```

```
        }
        public static void badMethod()
    {
        throw new RuntimeException();  }
    }
```

a) AB

c) ABC

b) BC

d) BCD

24 What will be the output of the program?

```java
public class MyProgram
{
    public static void main( String args[] )
    {
        try
        {
            System.out.print( "Hello world " );
        }
        finally
        {
            System. out. println ( "Finally executing " );
        }
    }
}
```

a) Nothing. The program will not compile because no exceptions are specified.

b) Nothing. The program will not compile because no catch clauses are specified.

c) Hello world.

d) Hello world Finally executing

25. Types of exceptions in Java programming are

a) Checked exception

c) Both A & B

b) unchecked exception

d) None

26. What is the output of the following program?

```java
    public class Test
    {
        private void m1()
        {
            m2();
            System.out.printf("1");
        }
```

```java
        private void m2()
        {
            m3();
            System.out.printf("2");
        }
        private void m3()
        {
            System.out.printf("3");
            try
            {
                int sum = 4/0;
                System.out.printf("4");
            }
            catch(ArithmeticException e)
            {
                System.out.printf("5");
            }
            System.out.printf("7");
        }
        public static void main(String[] args)
        {
            Test obj = new Test();
            obj.m1();
        }
    }
```

a) 35721   c) 3521

b) 354721   d) 35

27. What is the output of the following program?

```java
    public class Test
    {
        public static void main(String[] args)
        {
            try
            {
                System.out.printf("1");
                int data = 5 / 0;
            }
            catch(ArithmeticException e)
            {
                System.out.printf("2");
                System.exit(0);
            }
            finally
            {
                System.out.printf("3");
            }
            System.out.printf("4");
```

```
    }
  }
```

a) 12

c) 124

b) 1234

d) 123

3.12 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

➢ True-False with reason

    1. False. We can handle errors using try … catch block.

    2. False. At compile time.

    3. False. By interpreter

    4. True

    5. True.

    6. False. We can write on try with multiple catch blocks.

    7. True

    8. False. Nested try … catch can be used in java.

    9. True

    10. False. It will be handled at rum time by JVM

➢ Match **A** and **B**.

A	B
1)Exception	a)unexpected event
2)Error	b) try … catch within try … catch
3)Checked Exception	c)must be handled by programmer
4)Unchecked Exception	d)handled by JVM
5)Nested try … catch	e)error which can be handle at run time

Answer :

    1) – e,   2) – a,   3) – c,   4) – d,   5) – b

➢ Compare the following:

    1. Error v/s Exception

Error	Exception
Error is something unexpected in your	Exception is means using which Java

program which stop execution of the program.	handles the run time errors.
Error can be syntax error, logical errors, run-time errors or latent errors	Exception can Checked Exception or Unchecked Exception
Examples are StackOverflowError, VirtualMachineError, OutofMemoryError etc.	Examples, are IOException, ClassNotFoundException etc.

2. Checked Exception v/s Unchecked Exception

Checked Exception	Unchecked Exception
All the classes which extend the Throwable class except RuntimeException and Error are known as checked exceptions	The classes which inherit RuntimeException are known as unchecked exceptions.
The checked exceptions are checked at compile time.	The unchecked exceptions are checked at runtime.
They are not derived from RuntimeException class.	They are derived from RuntimeException class.
Examples are IOException, SQLException etc.OutofMemoryError etc.	Examples, are IOException, ClassNotFoundException etc.

3. Catch block v/s finally block

Catch block	Finally block
This block is compulsory to use with try block.	This block is optional.
We can write the code to handle the exception in catch block	We can write the code which we want to execute in any case; with or without error in this block
We can write multiple catch block with one try block	You can only have one finally block per try/catch block

4. Syntax error v/s runtime error

Syntax error	Runtime error
It is a grammatical error while writing program.	It is the error in the logic of program.
It is indented at compile time	It is identified at runtime
This error must be remove from the program to compile it.	Java handles this error using Exception.

➤ MCQ

1) a	10) b	19) c
2) b	11) d	20) c
3) e	12) a	21) b
4) c	13) c	22) c
5) c	14) d	23) b
6)b	15) d	24) d
7) c	16) a	25) d
8) d	17) c	26)d
9) a	18) b	27) a

3.13 FURTHER READING

1) "Java 2: The Complete Reference" by Herbert Schildt, McGraw Hill Publications.

2) "Effective Java" by Joshua Bloch, Pearson Education

3) Exception Handling in Core Java | Core Java Tutorial | Studytonight
    https://www.studytonight.com/java/exception-handling.php

4) Exception Handling in Java | Java Exceptions - javatpoint
    https://www.javatpoint.com/exception-handling-in-java

5) Exception handling in java with examples - BeginnersBook.com
    https://beginnersbook.com/2013/04/java-exception-handling/

3.14 ASSIGNMENTS

1) Write a java program to find solution of quadratic equation. Take care of divide by zero error and other arithmetic exceptions.

2) Write a program to get value of radius through keyboard and calculate area of circle. Take care of InputMismatchException.

3) Write a program to create an array of 10 integers. Get value of those 10 integers using console. Now ask for an index of array through keyboard then divide the array into two from that index. Take care of array index out of bound exception. Also handle InputMismatchException.