

Unit 4: Exception classes

4

Unit Structure

- 4.1 Learning Objectives
- 4.2 throw keyword
- 4.3 Built in Exception classes
- 4.4 Use defined Exception class
- 4.5 throws keyword
- 4.6 Throwable class
- 4.7 Chained Exception
- 4.8 Let us sum up
- 4.9 Check your Progress
- 4.10 Check your Progress: Possible Answers
- 4.11 Further Reading
- 4.12 Assignments

4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand the use of throw keyword in exception handling.
- Study and understand the various built-in exception classes and their usage in java program.
- Learn how to create user defined exception class and use it in java program.
- Understand the throws keyword and its use in program.
- Study the Throwable class.

4.2 THROW KEYWORD

In Java exception handling mechanism uses the throw keyword to explicitly raise an exception from a function or a block of code. It can also be used to raise user defined exception.

Syntax:

```
throw obj;
```

Here the object must be of Throwable type or subclass of Throwable. The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing try block is checked to see if it has a catch statement that matches the type of exception. If it finds a match, controlled is transferred to that statement otherwise next enclosing try block is checked and so on. If no matching catch is found then the default exception handler will halt the program.

For example,

Example1,

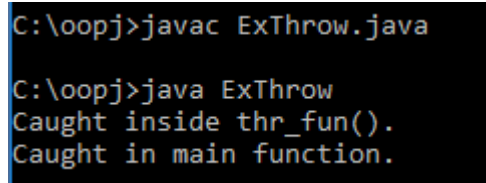
```
class ThrowExcep
{
    static void thr_fun()
    {
        try
        {
            throw new ArithmeticException ("demo");
        }
        catch( ArithmeticException e)
        {
```

```

        System. out. println ( "Caught inside thr_fun().");
        throw e;
    }
}

public static void main(String args[])
{
    try
    {
        thr_fun();
    }
    catch( ArithmeticException e)
    {
        System. out. println ("Caught in main function.");
    }
}
}

```



```

C:\oopj>javac ExThrow.java

C:\oopj>java ExThrow
Caught inside thr_fun().
Caught in main function.

```

Figure-81 Output of program

As you can see in above program the try block doesn't have any program logic which may raise an error/exception. We have used throw keyword which explicitly throws an object of ArithmeticException, which will be caught in catch block.

Example 2,

```

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 i;
        try {
            Scanner sc = new Scanner ( System.in );
            System. out. println ( " index = ");
            i = sc.nextInt();
            if ( i >= 6)
            {
                ArrayIndexOutOfBoundsException ex = new ArrayIndexOutOfBoundsException();
                Throw ex;
            }
        }
    }
}

```

```

else
    System.out.println ( " a[i] = " + a[i]);
} catch (ArrayIndexOutOfBoundsException e ) {
    System.out.println ( "Error occured as the value of i is >=6 ");
}
}
}
}

```

```

C:\ajava\oopj>java ExErr
index =
2
a[i] = 5

C:\ajava\oopj>java ExErr
index =
6
Error occured as the value of i is >=6

```

Figure-82 Output of program

4.3 BUILT IN EXCEPTION CLASSES

In java library (java.lang package), many built-in unchecked exception classes are available. Most common classes are the subclass of RuntimeException class. These classes are automatically handling the runtime errors while executing java program.

The following is the list of Java Unchecked Exception classes derived from RuntimeException.

- 1). **ArithmeticException**: Arithmetic error, such as divide-by-zero.
- 2). **ArrayIndexOutOfBoundsException** : Array index is out-of-bounds.
- 3). **ArrayStoreException** : Assignment to an array element of an incompatible type.
- 4). **ClassCastException** : Invalid cast.
- 5). **IllegalArgumentException** : Illegal argument used to invoke a method.
- 6). **IllegalMonitorStateException** :Illegal monitor operation, such as waiting on an unlocked thread.
- 7). **IllegalStateException** : Environment or application is in incorrect state.
- 8). **IllegalThreadStateException** : Requested operation not compatible with the current thread state.
- 9). **IndexOutOfBoundsException** : Some type of index is out-of-bounds.
- 10). **NegativeArraySizeException** : Array created with a negative size.

- 11). **NullPointerException** : Invalid use of a null reference.
- 12). **NumberFormatException** : Invalid conversion of a string to a numeric format.
- 13). **SecurityException** : Attempt to violate security.
- 14). **StringIndexOutOfBoundsException** : Attempt to index outside the bounds of a string.
- 15). **UnsupportedOperationException** : An unsupported operation was encountered.

There are also many built-in checked Exception classes readily available for handling various errors in various packages. The following is the list of such exception classes.

- 1). **IOException** : This class is available in java.io package. It handles the IO operation related runtime errors.
- 2). **FileNotFoundException** : This class is available in java.io package. It is used to handle the runtime error when we try to access a file which is not exists.
- 3). **ParseException** : This class is available in java.text package. For example, this exception raise when you are trying to parse a String to a Date Object and the string is not containing date format.
- 4). **ClassNotFoundException** : This class is available in java.lang package. It is a runtime exception that is thrown when an application tries to load a class at runtime using the Class.forName() or loadClass() or findSystemClass() methods ,and the class with specified name are not found in the classpath.
- 5). **CloneNotSupportedException** : This class is available in java.lang package. The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException. (refer last example of 3.12 block 1)
- 6). **InstantiationException** : This class is available in java.lang package. When we try to instantiate the abstract class or interface using the newInstance() method of Class class, then this exception will be thrown.
- 7). **InterruptedException** : This class is available in java.lang package. The InterruptedException is thrown when a thread is waiting or sleeping and another

thread interrupts it using the interrupt method in class Thread . (The thread will be discussed in detail in block 3).

- 8). **NoSuchMethodException** : This class is available in java.lang package. This exception occur when we are trying to run our java program which does not have main method in it.
- 9). **NoSuchFieldException** : This class is available in java.lang package. This exception is used to send a signals that the class doesn't have a field of a specified name.
- 10). **SQLException** : This class is available in java.sql package. This exception is raised when our program tries to interact with dbms software and due to some error not getting response.
- 11). **SocketException** : This class is available in java.net package. This exception occurs when our program is performing network programming using socket.
- 12). **RemoteException** : This class is available in java.rmi package. This exception occurs when we are calling remote method in our program and any error encounter.

4.3.1 EXAMPLES OF BUILT-IN EXCEPTION

➤ **ArithmeticException**

```
class ExException {
public static void main(String args[])
{
    try {
        int a = 30;
int b = 0;
        int c = a / b;
        System. out. println ( "Result = " + c);
    }
    catch (ArithmeticException e) {
        System. out. println ( "Divide by zero error" );
    }
}
}
```

```
C:\oopj>javac ExException.java
C:\oopj>java ExException
Divide by zero error
```

Figure-83 Output of program

➤ **ArrayIndexOutOfBoundsException**

```
class ExException {
public static void main(String args[])
{
    try {
        int a[ ] = { 1, 2, 3, 4, 5 };
        a[6] = 9;
    }
    catch (ArrayIndexOutOfBoundsException e) {
        System. out. println ("Index of array is more than 5");
    }
}
}
```

```
C:\oopj>javac ExException.java
C:\oopj>java ExException
Index of array is more than 5
```

Figure-84 Output of program

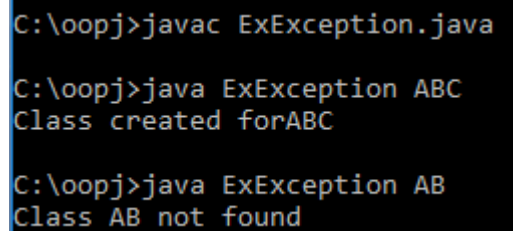
➤ **ClassNotFoundException**

```
class ABC {
}
class XYZ {
}
class ExException {
public static void main(String[] args)
{
    try{
        Object o = Class.forName(args[0]).newInstance();
        System. out. println ("Class created for" + o.getClass().getName());
    }
}
```

```

    }
    catch (Exception e)
    { System.out.println ( "Class " + args[0] + " not found "); }
}
}

```



```

C:\oopj>javac ExException.java

C:\oopj>java ExException ABC
Class created forABC

C:\oopj>java ExException AB
Class AB not found

```

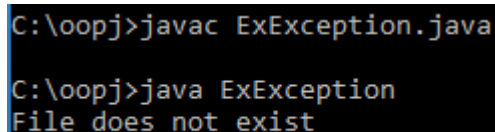
Figure-85 Output of program

➤ FileNotFoundException

```

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class File_notFound_Demo {
public static void main(String args[])
{
    try {
        File file = new File("E:// file.txt");
        FileReader fr = new FileReader(file);
    }
    catch (FileNotFoundException e) {
        System.out.println ("File does not exist");
    }
}
}

```



```

C:\oopj>javac ExException.java

C:\oopj>java ExException
File does not exist

```

Figure-86 Output of program

➤ IOException

```

import java.io.*;

```



```

class ExException {
public static void main(String args[])
    { FileInputStream f = null;
      f = new FileInputStream("abc.txt");
      int i;
      while ((i = f.read()) != -1) {
          System.out.print((char)i);
      }
      f.close();
    }
}

```

```

C:\oopj>javac ExException.java
ExException.java:7: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
    f = new FileInputStream("abc.txt");
        ^
ExException.java:9: error: unreported exception IOException; must be caught or declared to be thrown
    while ((i = f.read()) != -1) {
                ^
ExException.java:12: error: unreported exception IOException; must be caught or declared to be thrown
        f.close();
            ^
3 errors

```

Figure-87 Output of program

➤ InterruptedException

```

class ExException {
public static void main(String args[])
    {
      Thread t = new Thread();
      t.sleep(10000);
    }
}

```

```

C:\oopj>notepad ExException.java

C:\oopj>javac ExException.java
ExException.java:7: error: unreported exception InterruptedException; must be caught or declared to be thrown
    t.sleep(10000);
        ^
1 error

```

Figure-88 Output of program

➤ NullPointerException

```

class ExException {

```

```

public static void main(String args[])
{
    try {
        String a = null;
        System.out.println (a.charAt(0));
    }
    catch (NullPointerException e) {
        System.out.println ("NullPointerException..");
    }
}
}

```

```

C:\oopj>javac ExException.java
C:\oopj>java ExException
NullPointerException..

```

Figure-89 Output of program

➤ **NumberFormatException**

```

class ExException {
public static void main(String args[])
{
    try {
        int num = Integer.parseInt("hello");
        System.out.println (num);
    }
    catch (NumberFormatException e) {
        System.out.println ("Number format exception");
    }
}
}

```

```

C:\oopj>javac ExException.java
C:\oopj>java ExException
Number format exception

```

Figure-90 Output of program

➤ **StringIndexOutOfBoundsException**

```

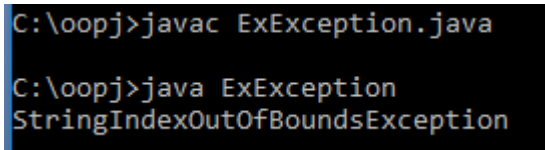
class ExException {

```

```

public static void main(String args[])
{
    try {
        String a = "Hello this is aryu";
        char c = a.charAt(24);
        System.out.println(c);
    }
    catch (StringIndexOutOfBoundsException e) {
        System.out.println ("StringIndexOutOfBoundsException");
    }
}
}

```



```

C:\oopj>javac ExException.java
C:\oopj>java ExException
StringIndexOutOfBoundsException

```

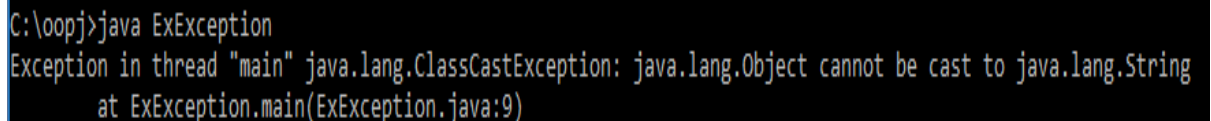
Figure-91 Output of program

➤ **ClassCastException**

```

class ExException {
public static void main(String[] args)
{
    String s = new String("Hello");
    Object obj = (Object) s;
    Object o1 = new Object();
    String s1 = (String) o1;
}
}

```



```

C:\oopj>java ExException
Exception in thread "main" java.lang.ClassCastException: java.lang.Object cannot be cast to java.lang.String
at ExException.main(ExException.java:9)

```

Figure-92 Output of program

4.4 USER DEFINED EXCEPTION CLASS

In java, we can create our own exception class by creating a class which extends Exception class.

The following example shows us the syntax of writing a custom exception class. In this example, the NegativeException class is created which extends an Exception class. We just have to write a constructor for initialization and toString function to print our own message. Here the negative exception is raised when entered value is negative. For this we have to check the entered value and throw an object of NegativeException if the value is negative. Similarly we can user defined exception class for checking our own condition for input.

```
import java.util.Scanner;

class NegativeException extends Exception
{
    private int x;
    NegativeException(int a)
    {
        x=a;
    }
    public String toString()
    {
        return "NegativeException[" + x +"] : value is less than zero";
    }
}

public class UDException
{
    public static void main ( String args[])
    {
        int a;
        Scanner sc = new Scanner ( System.in );
        try {
            System. out. println ( "Enter a: ");
            a = sc.nextInt();
            if( a < 0 )
                throw (new NegativeException(2) );
            else
                System. out. println ( " a = " + a );
        } catch (Exception e) { System. out. println (e); }
    }
}
```

```

C:\oopj>javac ExUDEXception.java

C:\oopj>java ExUDEXception
Enter a:
2
  a = 2

C:\oopj>java ExUDEXception
Enter a:
-9
NegativeException[2] : value is less than zero

```

Figure-93 Output of program

4.5 THROWS KEYWORD

If you do not handle the checked exception using a try catch block, compiler will give error message. Each and every program statement in java program is written in a method. Almost every method in the java library or even user defined may throw an exception or two. Handling all the exceptions using the try and catch block could be cumbersome and complex for coder.

Hence java provides an option, wherein whenever your code in the method definition may raise exception, you can declare that method with throws keyword followed by the exception or exceptions separated by comma. In this case we can omit writing code in try and catch block.

For example,

Example 1,

```

class ExException {
  public static void main(String[] args) throws ClassCasteException
  {
    String s = new String("Hello");
    Object obj = (Object) s;
    Object o1 = new Object();
    String s1 = (String) o1;
  }
}

```

Example 2,

```

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class File_notFound_Demo {

```

```

public static void main(String args[]) throws FileNotFoundException
{
    File file = new File("E:// file.txt");
    FileReader fr = new FileReader(file);
}
}

```

Example 3,

```

public class UDException
{
    public static void main ( String args[]) throws NegativeException,
    InputMismatchException

    {
        int a;
        Scanner sc = new Scanner ( System.in );
        System. out. println( "Enter a: ");
        a = sc.nextInt();
        if( a < 0 )
            throw (new NegativeException(2) );
        else
            System. out. println ( " a = " + a );
    }
}

```

4.6 THROWABLE CLASS

The `java.lang.Throwable` class is the super class of all errors and exceptions classes in the Java language. The objects which are the instances of this class are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. The `Throwable` class extends the `Object` class and implements `Serializable` interface.

The following are some of the methods of `Throwable` class.

- 1). **Throwable fillInStackTrace()** : This method fills in the execution stack trace.
- 2). **Throwable getCause()** : This method returns the cause of this throwable or null if the cause is nonexistent or unknown.
- 3). **String getLocalizedMessage()** : This method creates a localized description of this throwable.
- 4). **String getMessage()** : This method returns the detail message string of this throwable.
- 5). **StackTraceElement[] getStackTrace()** : This method provides programmatic access to the stack trace information printed by `printStackTrace()`.

- 6). **Throwable initCause(Throwable cause)** : This method initializes the cause of this throwable to the specified value.
- 7). **void printStackTrace()** : This method prints this throwable and its backtrace to the standard error stream.
- 8). **void printStackTrace(PrintStream s)** : This method prints this throwable and its backtrace to the specified print stream.
- 9). **void printStackTrace(PrintWriter s)** : This method prints this throwable and its backtrace to the specified print writer.
- 10). **void setStackTrace(StackTraceElement[] stackTrace)** : This method sets the stack trace elements that will be returned by `getStackTrace()` and printed by `printStackTrace()` and related methods.
- 11). **String toString()** : This method returns a short description of this Throwable.

4.7 CHAINED EXCEPTION

The chained exception allows you to link an exception with other exception. The former exception is the cause of later exception. For example, in a program we are getting numerator and denominator from a file. While reading a denominator number from a file, due to `IOException` if we get zero value, it we may get `ArithmeticException`. Thus the cause of `ArithmeticException` is the `IOException`. If we want to inform programmer about this, chain exception concept is used.

For example,

```
import java.io.*;
public class LinkedException
{
    static void raiseLinkedException() {
        ArithmeticException e = new ArithmeticException( " top most exception ");
        e.initCause( new IOException( " cause " ));
        throw e;
    }
    public static void main ( String args[] )
    {
        try {
            raiseLinkedException();
        } catch ( ArithmeticException ex) {
            System.out.println ( " caught : " + ex );
            System.out.println ( " cause : " + ex.getCause() );
        }
    }
}
```

}

```
C:\oopj>javac LinkedException.java  
  
C:\oopj>java LinkedException  
caught : java.lang.ArithmeticException: top most exception  
cause : java.io.IOException: cause
```

Figure-94 Output of program

4.8 LET US SUM UP

throw keyword : This keyword is used to throw an exception class object even if the error is not occurred.

Built in exception class : The java libraries have various readymade exception class available which can be used to handle different errors occurred during programming in java.

User defined exception class: If built in exception class can not be used for some programmer defined validation check user can create custom exception class.

throws keyword: They can be used in place of try and catch block. It can be used with method declaration along with exception name.

Throwable class : it is a parent class of all exception and error classes.

Chained Exception : It is a concept in java using which we can create a chain of exceptions. In this chain the upper exception is raised because of lower exception in the chain.

4.9 CHECK YOUR PROGRESS

- True-False with reason
 1. Throw and throws keyword can be used for the same purpose.
 2. Throwable is an interface.
 3. The exception must be handled using try ... catch block
 4. We can handle exception without using try and catch.
 5. Throw keyword explicitly raise an exception error.

- Match **A** and **B**.

A

- 1)Throw
- 2)Throws
- 3)Throwable
- 4)User define Exception
- 5)Built in Exception

B

- a)it is custom exception class
- b)this keyword is used to throw exception
- c)it is exception class available in java library
- d)it is an option of try and catch
- e)it is a parent of all exception class

➤ Answer the following:

1. Which keyword is used to raise an exception?
2. Compare throw and throws.
3. Compare built in exception and user define exception

➤ MCQ

1. Consider the following try..... catch block

```
class TryCatch
{
public static void main(String args[ ])
{
try
{
double x = 0.0;
throw ( new Exception("Thrown") );
return;
}
catch(Exception e)
{
System. out. println ("Exception caught");
return;
}
finally
{
System. out. println ("finally");
}
}
}
```

What will be the output.

- | | |
|-----------------------------|------------|
| a) Exception caught | c) finally |
| b) Exception caught finally | d) Thrown |

2. In below java program, which exception will occur?

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

```
FileReader file = new FileReader("test.txt");  
  
}
```

- a) NullPointerException at compile time
- b) NullPointerException at run time
- c) FileNotFoundException at compiler time
- d) FileNotFoundException at runtime

3. which answer most closely indicates the behavior of the program?

```
public class MyProgram  
{  
    public static void throwit()  
    {  
        throw new RuntimeException();  
    }  
    public static void main(String args[])  
    {  
        try  
        {  
            System.out.println ("Hello world ");  
            throwit();  
            System.out.println ("Done with try block ");  
        }  
        finally  
        {  
            System.out.println ("Finally executing ");  
        }  
    }  
}
```

- a) The program will not compile.
- b) The program will print Hello world, then will print that a RuntimeException has occurred, then will print Done with try block, and then will print Finally executing.
- c) The program will print Hello world, then will print that a RuntimeException has occurred, and then will print Finally executing.
- d) The program will print Hello world, then will print Finally executing, then will print that a RuntimeException has occurred.

4. What will be the output of the program?

```
public class RTExcept  
{
```

```

public static void throwit ()
{
    System.out.print("throwit ");
    throw new RuntimeException();
}
public static void main(String [] args)
{
    try
    {
        System.out.print("hello ");
        throwit();
    }
    catch (Exception re )
    {
        System.out.print("caught ");
    }
    finally
    {
        System.out.print("finally ");
    }
    System. out. println ("after ");
}
}

```

- a) hello throwit caught
- b) Compilation fails
- c) hello throwit RuntimeException caught after
- d) hello throwit caught finally after

5. What will be the output of the program?

```

class Exc0 extends Exception { }
class Exc1 extends Exc0 { }
public class Test
{
    public static void main(String args[])
    {
        try
        {
            throw new Exc1();
        }
        catch (Exc0 e0)
        {
            System. out. println ("Ex0 caught");
        }
        catch (Exception e)
        {
            System. out. println ("exception caught");
        }
    }
}

```

```
}  
}
```

- a) Ex0 caught
- b) exception caught
- c) Compilation fails because of an error at line 2.
- d) Compilation fails because of an error at line 9.

6. What is the output of following Java program

```
class Main {  
    public static void main(String args[]) {  
        try {  
            throw 10;  
        }  
        catch(int e) {  
            System.out.println ("Got the Exception " + e);  
        }  
    }  
}
```

- a) Got the Exception 10
- b) Got the Exception 0
- c) Compiler Error

7. What is the output of following Java program

```
class Test extends Exception { }  
  
class Main {  
    public static void main(String args[]) {  
        try {  
            throw new Test();  
        }  
        catch(Test t) {  
            System.out.println ("Got the Test Exception");  
        }  
        finally {  
            System.out.println ("Inside finally block ");  
        }  
    }  
}
```

- a) Got the Test Exception Inside finally block
- b) Got the Test Exception
- c) Inside finally block
- d) Compiler Error

8. What is the output of following Java program

```

class Base extends Exception {}
class Derived extends Base {}

public class Main {
    public static void main(String args[]) {
        // some other stuff
        try {
            // Some monitored code
            throw new Derived();
        }
        catch(Base b) {
            System.out.println ("Caught base class exception");
        }
        catch(Derived d) {
            System.out.println ("Caught derived class exception");
        }
    }
}

```

- a) Caught base class exception
- b) Caught derived class exception
- c) Compiler Error because derived is not throwable
- d) Compiler Error because base class exception is caught before derived class

9. What is the output of following Java program

```

class Test
{
    public static void main (String[] args)
    {
        try
        {
            int a = 0;
            System.out.println ("a = " + a);
            int b = 20 / a;
            System.out.println ("b = " + b);
        }

        catch(ArithmeticException e)
        {
            System.out.println ("Divide by zero error");
        }

        finally
        {
            System.out.println ("inside the finally block");
        }
    }
}

```

- a) Compile error
- b) Divide by zero error
- c) a = 0
Divide by zero error
inside the finally block
- d) a = 0
- e) inside the finally block

10. What is the output of following Java program

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            int a[]= {1, 2, 3, 4};
            for (int i = 1; i <= 4; i++)
            {
                System.out.println ("a[" + i + "]=" + a[i] + "\n");
            }
        }

        catch (Exception e)
        {
            System.out.println ("error = " + e);
        }

        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println ("ArrayIndexOutOfBoundsException");
        }
    }
}
```

- a) Compiler error
- b) Run time error
- c) ArrayIndexOutOfBoundsException
- d) Error Code is printed
- e) Array is printed

11. Given the following piece of code:

```
class SalaryCalculationException extends Exception{}
class Person{
    public void calculateSalary() throws SalaryCalculationException{
```

```

        //...
        throw new SalaryCalculationException();
        //...
    }
}
class Company{
    public void paySalaries(){
        new Person().calculateSalary();
    }
}

```

Which of the following statements is correct?

1. This code will compile without any problems.
2. This code will compile if in method paySalaries() we return a boolean in stead of void.
3. This code will compile if we add a try-catch block in paySalaries().
4. This code will compile if we add throws SalaryCalculationException in the signature of method paySalaries().

- | | |
|------------|------------|
| a) 1 and 4 | c) 2 and 4 |
| b) 2 and 3 | d) 3 and 4 |

12. What will be the output of the following piece of code:

```

class Person{
    public void talk() {}
}
public class Test{
    public static void main(String args[]){
        Person p = null;
        try{
            p.talk();
        }
        catch(NullPointerException e){
            System.out.print("There is a NullPointerException. ");
        }
        catch(Exception e){
            System.out.print("There is an Exception. ");
        }
        System.out.print("Everything went fine. ");
    }
}

```

- a) There is a NullPointerException. Everything went fine.
- b) There is a NullPointerException.
- c) There is a NullPointerException. There is an Exception.

- ```

 }

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

```
- a) `ArrayIndexOutOfBoundsException` is thrown
  - b) `ExceptionInInitializerError` is thrown
  - c) `IllegalStateException` is thrown
  - d) `StackOverflowException` is thrown

---

## 4.10 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

---

➤ True-False with reason

1. False. `throw` keyword is used to raise exception programmatically while `throws` used with method declaration declaring that this method might raised an exception.
2. False. `Throwable` is a class.
3. False. The checked exception must be handled using `try ... catch` block
4. True
5. True

➤ Match **A** and **B**.

- | <b>A</b>                  | <b>B</b>                                                      |
|---------------------------|---------------------------------------------------------------|
| 1) <code>Throw</code>     | a) it is custom exception class                               |
| 2) <code>Throws</code>    | b) this keyword is used to throw exception                    |
| 3) <code>Throwable</code> | c) it is exception class available in java library            |
| 4) User define Exception  | d) it is an option of <code>try</code> and <code>catch</code> |
| 5) Built in Exception     | e) it is a parent of all exception class                      |

**Answer:**

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

➤ Answer the following:

1. “`throw`” keyword is used to raise an exception.
2. `throw` v/s `throws`

| <b>throw</b>                                 | <b>throws</b>                                    |
|----------------------------------------------|--------------------------------------------------|
| It is used to raise exception explicitly     | It is used with method which may raise exception |
| It is used with user defined exception class | It is an option for try and catch                |

### 3. Built in exception v/sUser define exception

| <b>Built in exception</b>                                            | <b>User define exception</b>                                                                               |
|----------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| They are the readily available classes used to handle runtime errors | They are the class created by user which extends Exception class and raised by user on specific condition. |
| They are raised when runtime error.                                  | They must be raised by user using throw keyword                                                            |

#### ➤ MCQ

- |      |       |       |
|------|-------|-------|
| 1) b | 6) c  | 11) d |
| 2)c  | 7) a  | 12)a  |
| 3) d | 8) d  | 13)d  |
| 4) d | 9) c  | 14) d |
| 5) a | 10) a | 15) b |

---

## 4.11 FURTHER READING

---

- 1) Java - User Defined Exceptions | Learn JAVA Online | Fresh2Refresh ...  
<https://fresh2refresh.com> › Java Tutorial
- 2) User defined Exception subclass in Java Exception Handling | Core ...  
<https://www.studytonight.com/java/create-your-own-exception.php>
- 3) “Java 2: The Complete Reference” by Herbert Schildt, McGraw Hill Publications.
- 4) “Effective Java” by Joshua Bloch, Pearson Education

---

## 4.12 ASSIGNMENTS

---

- 1) Create a class name account with attributes like account number, name, type of account, balance etc. and methods like get account information, print account details, deposit and withdraw. Create an exception class which raised when account balance is below 2000 while withdrawal. Also raise exception when negative amount is sent to deposit function. Create a class with main method to demonstrate the function of account class and exception classes.
  
- 2) Create a class name student which stores information like roll number, name, phone number, address, course etc. Write a function which accepts an object of student to add a new student in existing list of student. While adding check for roll number. The roll number should be in 3 digit. Implement this check using user define exception class.