1 Exceptions in Java

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions.

2 Runtime stack mechanism in java

Runtime Stack Mechanism as its name indicates is a Stack that have been created by JVM for each Thread at the time of Thread creation, JVM store every methods calls performed by that Thread in the Stack.
Each entry or method call called activation record or "stack frame"
JVM removes each entry (method call record) after completing execution of it.
JVM destroys the empty stack after completing all methods call and terminates the program normally.

```java
class Exception1
{
    public static void main(String[] args)
    {
        callPrincipal();
    }
    public static void callPrincipal()
    {
        callHod();
    }
    public static void callHod()
    {
        callStudent();
    }
    public static void callStudent()
    {
        System.out.println("first student");
    }
}
```

Exception1.java

```
D:\phani>javac Exception1.java

D:\phani>java Exception1
first student
```

				call Student()					
			call Hod()	call Hod()	call Hod()				
		call Prinicila()	call Prinicila()	call Prinicila()	call Principal()	call Principal()			
	main()	main()	main()	main()	main()	main()	main()		

for every Thread JVM will create Runtime stack

each entry is called Active record or stack frame

this empty stack is destroyed by JVM

3 Default exception handler in java

If at all any exception are occur in the program execution, the JVM will identity what exact that exception is and will create the corresponding exception class object
It will display print stack trace i.e, it will display program name, class name and corresponding method where exact that exception is raised and terminating the program abnormally or unsuccessfully.

1. separation the error logic or exception logic from our regular business logic.
2. grouping and differentiating the exception types.
3. handling the exception and making the program to terminate normally or successfully.

```java
class Exception2
{
    public static void main(String[] args)
```

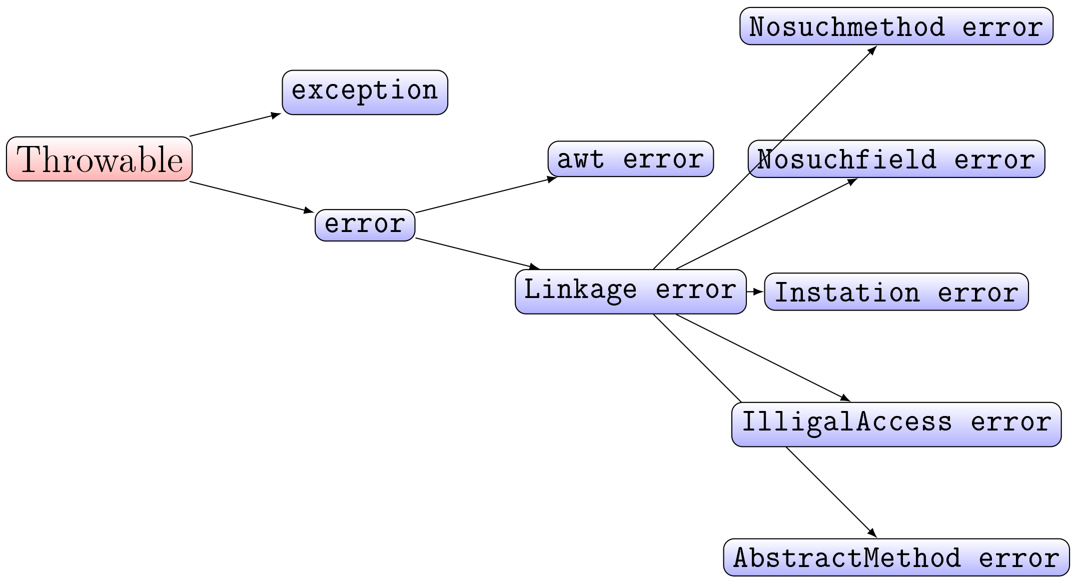
```
D:\phani>javac Exception2.java

D:\phani>java Exception2
Exception in thread "main"
```

```
    {
        callPrincipal();
    }
    public static void callPrincipal()
    {
        callHod();
    }
    public static void callHod()
    {
        callStudent();
    }
    public static void callStudent()
    {
        System.out.println(10/0);
    }
}
```

Exception2.java

```
 java.lang.ArithmeticException: / by zero
at Exception2.callStudent(Exception2.java:17)
at Exception2.callHod(Exception2.java:13)
at Exception2.callPrincipal(Exception2.java:9)
at Exception2.main(Exception2.java:5)
```

4 Exception hierarchy in java



Error: An Error indicates serious problem that a reasonable application should not try to catch.
Exception: Exception indicates conditions that a reasonable application might try to catch.

4.1 Checked Vs Unchecked

Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.
Unchecked are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

1. fully checked
2. partially checked

5 Customized Exception Handling by using try, catch

5.1 A try block followed by a catch block

The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.
If no exception occurs in try block then the catch blocks are completely ignored.

5.2 Catch block

A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes.

```
try
{
code that may throw exception / risk code
}
catch(name of the exception)
{
Hadling code
}
```

Types of Exception in Java with Examples

5.3 Arithmetic Exception

It is thrown when an exceptional condition has occurred in an arithmetic operation.

```
class Exception3 //Arithmetic Exception
{
    public static void main(String[] args)
    {
         try
         {
            int a = 20, 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");

        }
    }
}
```

Exception3.java

5.4 NullPointer Exception

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

```
class Exception4//NullPointer Exception
{
    public static void main(String[] args)
    {
        try
        {
            String a = null; //null value
            System.out.println(a.charAt(0));
        } catch(NullPointerException e)
        {
            System.out.println("
                NullPointerException..");
        }
    }
}
```

Exception4.java

5.5 StringIndexOutOfBound Exception

It is thrown by String class methods to indicate that an index is either negative than the size of the string

```
class Exception5//StringIndexOutOfBound
    Exception
{
    public static void main(String[] args)
    {
        try {
```

5.6 FileNotFound Exception

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

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class Exception6 //FileNotFound Exception
{
    public static void main(String[] args)
```

```
            String a = "B V. Raju Institute
                of Technology"; // length is
                 33
            char c = a.charAt(40);
            // accessing 40th element
            System.out.println(c);
        }
        catch(
            StringIndexOutOfBoundsException
            e) {
            System.out.println("
                StringIndexOutOfBoundsException
                ");
        }
    }

}
```

Exception5.java

```
    {
        try
        { // Following file does not exist
            File file = new File("E://file.txt")
                ;
            FileReader fr = new FileReader(file)
                ;
        }
        catch (FileNotFoundException e)
        {
           System.out.println("File does not
               exist");
        }
    }
}
```

Exception6.java

5.7 NumberFormat Exception

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

```
class Exception7 //NumberFormat Exception
{
    public static void  main(String[] args)
    {
            try
            {
                int num = Integer.parseInt ("
                    phani") ;
               System.out.println(num);
        } catch(NumberFormatException e)
        {
            System.out.println("Number format
                exception");
        }
    }
}
```

Exception7.java

5.8 ArrayIndexOutOfBounds Exception

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.

```
class Exception7 //ArrayIndexOutOfBounds
    Exception
{
    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");
        }
    }
}
```

Exception8.java

5.9 Other Exceptions

ClassNotFoundException This Exception is raised when we try to access a class whose definition is not found
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 which is not found.

5.10 A try block followed by one or more catch blocks

a single try block can have any number of catch blocks

```java
class Exception9
{
    public static void main(String[] args)
    {
            try
            {
                System.out.println(10/0);
             }
            catch(ArithmeticException e)
            {
                System.out.println(e);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
    }
}
```

Exception9.java

```
D:\phani>javac Exception9.java

D:\phani>java Exception9
java.lang.ArithmeticException: / by zero
```

```java
class Exception10
{
    public static void main(String[] args)
    {
            try
            {
                int[]a = new int[10];
                a[0]=20;
                System.out.println(a[20]/10);
             }
            catch(ArithmeticException e)
            {
                System.out.println(e);
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                System.out.println(e);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
    }
}
```

Exception10.java

```
D:\phani>javac Exception10.java

D:\phani>java Exception10
java.lang.ArrayIndexOutOfBoundsException: 20
```

What did we observe from the above two examples?

1. It is clear that when an exception occurs, the specific catch block (that declares that exception) executes. This is why in Exception9.java example first block executed.
2. Although I have not shown you above, but if an exception occurs in Exception10.java code which is not Arithmetic and ArrayIndexOutOfBounds then the last generic catch handler would execute.

5.11 print Exception Information

e.printStackTrace	Name of the Exception:Description stacktrace
toString()	Name of the Exception: Description
sop(e)	Name of the Exception: Description
e.getMessage()	Description

printStackTrace()

Prints all detailsName of the exception :description
and stack trace

```
class Exception12
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println(10/0);
        }
        catch(ArithmeticException e)
        {
            e.printStackTrace();
        }
    }
}
```

Exception12.java

```
D:\phani>javac Exception12.java

D:\phani>java Exception12
java.lang.ArithmeticException: / by zero
        at Exception12.main(Exception12.java:7)
```

toString()

Prints all details Name of the Exception : description

```
class Exception13
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println(10/0);
        }
        catch(ArithmeticException e)
        {
            System.out.println(e);
            System.out.println(e.toString());
        }
    }
}
```

Exception13.java

```
D:\phani>javac Exception13.java

D:\phani>java Exception13
java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException: / by zero
```

getMessage()

Prints all details description

```
class Exception14
{
    public static void main(String[] args)
    {
        try
        {
```

```
D:\phani>javac Exception14.java

D:\phani>java Exception14
/ by zero
```

```
                System.out.println(10/0);
            }
            catch(ArithmeticException e)
            {
                System.out.println(e.getMessage());
            }
    }
}
```

Exception14.java

internally Default exception handler will use printStackTrace method to print information to the console

6 finally keyword

Just as final is a reserved keyword, so in same way finally is also a reserved keyword in java i.e, we can't use it as an identifier. The finally keyword is used in association with a try/catch block and guarantees that a section of code will be executed, even if an exception is thrown. The finally block will be executed after the try and catch blocks, but before control transfers back to its origin.

1. **Exceptions do not occur in the program**

```
class Exception15
{
    public static void main(String[] args)
    {
            try
            {
                System.out.println("division
                    operation"+10/5);
            }
            catch(ArithmeticException e)
            {
                System.out.println(e.getMessage());
            }
            finally
            {
                System.out.println("Executes whether
                    exception occurs or not");
            }
    }
}
```

Exception15.java

```
D:\phani>javac Exception15.java

D:\phani>java Exception15
division operation2
Executes whether exception occurs or not
```

Here above exception not occurs but still finally block executes since finally is meant to execute whether exception occurs or not.
Flow of Above Program: First it starts from main method and then goes to try block and in try since no exception occurs so flow dosen't goes to catch block hence flow goes directly from try to finally block.

2. **Exception occurs and corresponding catch block matches**

```
class Exception16
{
    public static void main(String[] args)
    {
            try
            {
                System.out.println("division
                    operation"+10/0);
            }
            catch(ArithmeticException e)
            {
```

```
D:\phani>javac Exception16.java

D:\phani>java Exception16
java.lang.ArithmeticException: / by zero
        at Exception16.main(Exception16.ja
Executes whether exception occurs or not
```

```
                e.printStackTrace();
            }
            finally
            {
                System.out.println("Executes whether
                    exception occurs or not");
            }
    }
}
```

Exception16.java

Here, above exception occurs and corresponding catch block found but still finally block executes since finally is meant to execute whether exception occurs or not or whether corresponding catch block found or not.
Flow of Above Program: First, starts from main method and then goes to try block and in try an Arithmetic exception occurs and corresponding catch block is also available so flow goes to catch block. After that flow **don't** go to try block again since once an exception occurs in try block then flow dosen't come back again to try block. After that finally execute since finally is meant to execute whether exception occurs or not or whether corresponding catch block found or not.

3. **Exception occurs and corresponding catch block not found/match**

```
class Exception17
{
    public static void main(String[] args)
    {
            try
            {
                System.out.println(10/0);
            }
            catch(NullPointerException e)
            {
                e.printStackTrace();
            }
            finally
            {
                System.out.println("Executes whether
                    exception occurs or not");
            }
    }
}
```

Exception17.java

```
D:\phani>javac Exception17.java

D:\phani>java Exception17
Executes whether exception occurs or not
Exception in thread "main"
java.lang.ArithmeticException: / by zero
   at Exception17.main(Exception17.java:7)
```

Here above exception occurs and corresponding catch block not found/match but still finally block executes since finally is meant to execute whether exception occurs or not or whether corresponding catch block found/match or not.

4. **Application of finally block**: So basically the use of finally block is resource deallocation. Means all the resources such as Network Connections, Database Connections, which we opened in try block are needed to be closed, so that we won't lose our resources as opened. So those resources are needed to be closed in finally block.

7 throw and throws in Java

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

```
class Exception18
{
    static void pass(int marks)
            {
                if(marks<35)
```

```
D:\phani>javac Exception18.java

D:\phani>java Exception18
Exception in thread "main"
java.lang.ArithmeticException: not passed
        at Exception18.pass(Exception18.java:6)
```

```
                    throw new ArithmeticException ("
                        not passed");
                else
                    System.out.println("Passes");
            }
    public static void  main(String[] args)
    {
            pass(30);
            System.out.println("remaining code");
    }
}
```

Exception18.java

```
        at Exception18.main(Exception18.java:10)
```

In the above example, we have created the pass method that takes integer value as a parameter. If the marks are less than 35, we are throwing the ArithmeticException otherwise print a message Passed.

7.1 Throws

we can use throws keyword to deligate responisbility of ExceptionHandeling to the caller(it may be another method or JVM)then caller method is responsible to handel that exception

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

```
class Exception19
{
    public static void  main(String[] args) throws
        InterruptedException
    {
        Thread.sleep(1000);
    }
}
```

Exception19.java

```
D:\phani>javac Exception19.java

D:\phani>java Exception19
```