Unit 1.3
Exception handling in java

Introduction to Exception Handling

- An exception is an event, which occurs during the execution of a program that interrupts the normal flow of the program's instruction.
- Exception handling is the process of responding to exceptions when a computer program runs.
- An exception occurs when an unexpected event happens that requires special processing.
- When an Exception occurs the normal flow of the program is disrupted and the program terminates abnormally, which is not recommended, therefore, these exceptions **are to be handled using exception handling mechanism**.
- The main purpose of exception handling is to provide a means to detect and report an error to the user.
- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
  i. A user has entered an invalid data.
  ii. A file that needs to be opened cannot be found.
  iii. A network connection has been lost in the middle of communications
  iv. JVM has run out of memory.
  And many more….

Exception class in Java

- Some of the exception class derived from class **Exception class** defined in **java.lang package** for handling exception are given below
  1. **IOException** i.e. Thrown when some I/O exception of some sort occur during reading and writing files and directories.
  2. **FileNotFoundException** i.e. Thrown when trying to access a file that doesn't exist
  3. **IllegalArgumentException** i.e. Thrown when a method has been passed arguments which are not appropriate or illegal.
  4. **ClassNotFoundException** i.e. Thrown when an application tries to load in a class through its string name but no definition for the class with the specified name could be found.
  5. **ArithmeticException i.e.** Thrown when an exceptional arithmetic condition has occurred.
  6. **ArrayIndexOutOfBoundsException** i.e. thrown when 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.
  7. **InputMismatchException i.e.** Thrown by a Scanner to indicate that the token retrieved does not match the pattern for the expected type, or that the token is out of range for the expected type.

Exception handling mechanism in Java

- The steps in exception handling mechanism in java are
  1. Find the problem. (Hit the exception)
  2. Inform that an error has occurred. (Throw the exception)
  3. Receive the error information (Catch the exception)
  4. Take corrective actions. (Handel the exception)
- Exceptions in java are handled by using the keywords **try, catch, throw, throws and finally**.
- Try block consists of code that needs to be monitored for exceptions. Try block throw error if occur.
- We can catch that exception using catch keyword and handle it.
- **Throw keyword is used to manually** throw the exception however system generated exception are **automatically thrown by the JVM itself.**
- Throws clause is used to specify exception thrown by a method.
- Finally block consist of code that must be executed without concerning which either try or catch block was executed.
- The general form of an exception handling block is given below

```
try
{
        //section of code that needs to be monitored
}
catch(ExceptionType1 obj1)
{
        //Exception handler for ExceptionType1
}
catch(ExceptionType2 obj2)
{
        //Exception handler for ExceptionType2
}
finally
{
        //block of code to be executed whether or not an exception is thrown.
}
```

- **The above form is also called as exception handling with multiple catch statement or multilevel exception handling**.

Example

1. **WAP to demonstrate handling ArithmeticException.**

```
import java.util.Scanner;
public class ExceptionDemo
{
        public static void main(String [] args)
        {
                try
                {
                        Scanner sc=new Scanner(System.in);
                        System.out.println("Enter numerator");
                        int num=sc.nextInt();
                        System.out.println("Enter denominator");
                        int deno=sc.nextInt();
                        int c=num/deno;
                        System.out.println("Quotient="+c);
                }
                catch(ArithmeticException ae)
                {
                        System.out.println("Divide by zero exception occured");
                }
        }
}
```

Output:

```
Enter numerator
4
Enter denominator
2
Quotient=2
```

```
Enter numerator
4
Enter denominator
0
Divide by zero exception occured
```

2. **WAP to demonstrate multilevel exception handling.**

```
import java.util.*;
public class ExceptionHandling1
{
	public static void main(String [] args)
	{
		try
		{
			Scanner sc=new Scanner(System.in);
			System.out.println("Enter the numerator");
			int num=sc.nextInt();
			System.out.println("Enter the denominator");
			int deno=sc.nextInt();
			int res=num/deno;
			System.out.println("Enter the index to store the result");
			int ind=sc.nextInt();
			int [] b=new int[5];
			b[ind]=res;
			System.out.println("Successfully Stored");
		}
		catch(ArithmeticException ae)
		{
			System.out.println("Divide by zero error occured ");
		}
		catch(ArrayIndexOutOfBoundsException e)
		{
			System.out.println("Index out of bound exception");
		}
	}
}
```

Output:

```
Enter the numerator
4
Enter the denominator
2
Enter the index to store the result
1
Successfully Stored
```

```
Enter the numerator
4
Enter the denominator
2
Enter the index to store the result
8
Index out of bound exception
```

```
Enter the numerator
4
Enter the denominator
0
Divide by zero error occured
```

3. **WAP to demonstrate how a single exception handler can handle all the exception.**

Note: Since all the exception are derived from **Exception Class, so we use it to handle all the exception.**

```
import java.util.*;
public class ExceptionHandling
{

	public static void main(String [] args)
	{
		try
		{
			Scanner sc=new Scanner(System.in);
			System.out.println("Enter the numerator");
			int num=sc.nextInt();
			System.out.println("Enter the denominator");
			int deno=sc.nextInt();
			int res=num/deno;
			System.out.println("Enter the index to store the result");
			int ind=sc.nextInt();
			int [] b=new int[5];
			b[ind]=res;
			System.out.println("Successfully Stored");
		}
		catch(Exception ae)
		{
			System.out.println("Exception Occured "+ae.getMessage());
		}
	}
}
```

Output:

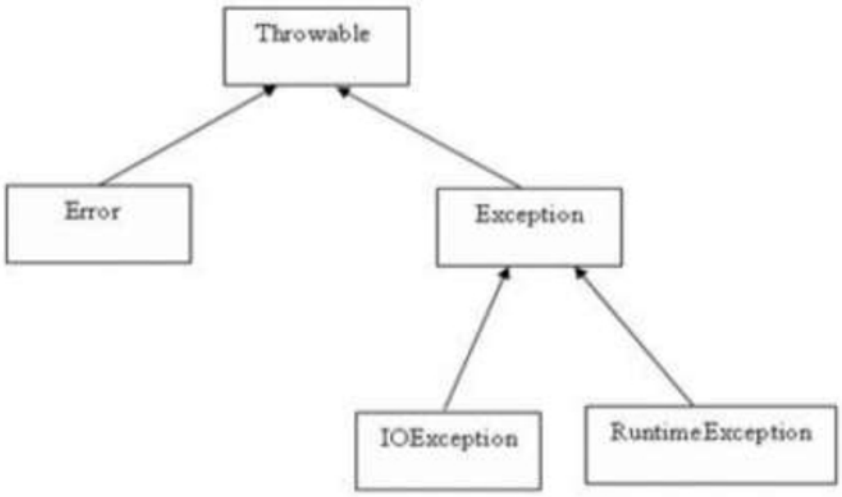
```
4
Enter the denominator
2
Enter the index to store the result
1
Successfully Stored
```

```
Enter the numerator
4
Enter the denominator
0
Exception Occured / by zero
```

```
Enter the numerator
4
Enter the denominator
2
Enter the index to store the result
9
Exception Occured 9
```

Types of Exception.

- All exception classes are subclass of the java.lang.Exception class.
- The exception class is a subclass of the Throwable class.
- The Exception class has two main subclasses: IOException i.e. checked exception class and RuntimeException i.e. unchecked exception Class.



There are two types of exception.

1. Checked Exception: I/O Exception
2. Unchecked Exception: Runtime Exception

1. **Checked Exception**
   - Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.
   - It is named as **checked exception** because these exceptions are **checked** at Compile time. They require a mandatory try catch finally code block to handle it.
   - Some of the checked exceptions are as follows
     i. ClassNotFoundException
     ii. FileNotFoundException
     iii. SQLException
     iv. IOException
     v. NoSuchFieldException
     vi. NoSuchMethodException

Example: let us consider to read the content of a file

```
import java.io.*;
public class Bh
{
        public static void main(String args [])
        {
                FileReader fr=new FileReader("abc.txt");
                BufferedReader br=new BufferedReader(fr);
                String s;
                while((s=br.readLine())!=null)
                {
                        System.out.println(s);
                }
        }
}
```

In the above program, if the FileReader class failed to read the file"abc.txt" then it results the compilation error as below.

```
Bh.java:6: error: unreported exception FileNotFoundException; must be caught or
declared to be thrown
```

Similarly the readLine() method of BufferedReader class throws the following type of compilation error .

```
Bh.java:9: error: unreported exception IOException; must be caught or declared t
o be thrown
while((s=br.readLine())!=null)
                      ^
```

So as we know that checked exceptions are checked during the compilation but we didn't handle it hence we get error.

Now we can use following two techniques to handle above checked exceptions.

a. Declare the exception using throws keyword.

```
import java.io.*;
public class Bh
{
        public static void main(String args [])  throws IOException // or throws Exception
        {
                FileReader fr=new FileReader("abc.txt");
                BufferedReader br=new BufferedReader(fr);
                String s;
                while((s=br.readLine())!=null)
                {
                        System.out.println(s);
                }
        }
}
```

b. Handling exception using try catch block

```
import java.io.*;
public class Bh
{
        public static void main(String args [])
        {
                FileReader fr=null;
                try
                {
                        fr=new FileReader("abc.txt");
                }
                catch(FileNotFoundException fnfe)
                {
                        System.out.println("The file you are trying to open doesn't exist"+
                fnfe.getMessage());
                }
                BufferedReader br=new BufferedReader(fr);
                String s;
                try
                {
                        while((s=br.readLine())!=null)
                        {
                                System.out.println(s);
                        }
                }
                catch(IOException ioe)
```

```
                {
                        System.out.println("Failed to read the file"+ioe.getMessage());
                }
        }
}
```

Or we can handle using single exception handler as below.

```
import java.io.*;
public class Bh
{
        public static void main(String args [])
        {
                try
                {
                        FileReader fr=new FileReader("abc.txt");
                        BufferedReader br=new BufferedReader(fr);
                        String s;
                        while((s=br.readLine())!=null)
                        {
                                System.out.println(s);
                        }
                }
                catch (IOException e) // or catch(Exception e)  // or catch(Throwable b)
                {
                        System.out.println("Error occured "+e.getMessage());
                }
        }
}
```

2. **Unchecked Exception**

- Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn't handle/declare that exception, the program won't give a compilation error.
- Most of the times these exception occurs due to the bad data provided by user during the user-program interaction.
- It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately.
- All Unchecked exceptions are direct sub classes of **RuntimeException** class.
- It **doesn't mean** that compiler is not checking these exceptions so we shouldn't handle them. In fact, we should handle them more carefully. Here are the few most frequently seen unchecked exceptions –

i. NullPointerException // (FileReader fr=null, and we try to access fr)
ii. ArrayIndexOutOfBoundsException
iii. ArithmeticException

Example: Let us consider the following program

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

```
                int a=10;
                int b=0;
                int c=a/b;
                System.out.println(c);
        }
}
```

If you compile this code, it would compile successfully however when you will run it, it would throw ArithmeticException. That clearly shows that unchecked exceptions are not checked at compile-time, they are being checked at runtime.

```
C:\Users\Hp-User\Desktop\Java programs\exception>java Bh
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Bh.main(Bh.java:7)
```

So we can handle the above unchecked exception as below

```
public class Test
{
        public static void main(String [] args)
        {
                try
                {
                        int a=10;
                        int b=0;
                        int c=a/b;
                        System.out.println(c);
                }
                catch(ArithmeticException e) // catch(Exception e)
                {
                        System.out.println("Attempte to diveide by zero is made "+e.getMessage());
                }
        }
}
```

After executing the program, the output in the console goes like this.

```
Attempt to divide by zero is made / by zero
```

The throws Statement

- The throws statement is used to throw an exception that is caused by a method.
- A throws statement can't throw unchecked exception i.e. RunttimeException and their subclasses.
- Also abstract method can't throw exception using throws.

The general form is

```
Type method _name (Parameter-list) throws Exception list
{
    // body of method
}
```

Example:

```
import java.io.*;
public class Bh
{
        public static void main(String args [])  throws IOException // or throws Exception
```

```
        {
            FileReader fr=new FileReader("abc.txt");
            BufferedReader br=new BufferedReader(fr);
            String s;
            while((s=br.readLine())!=null)
            {
                System.out.println(s);
            }
        }
    }
```

The throw Statement

- We can throw an exception, that you just caught, by using the **throw** keyword.
- Basically the Java run time system throws the exceptions and exception handler catch them but if we want to throw the exceptions explicitly then we have to use throw statement.
- The general form of throw is

  throw throwableInstance

  The throwableInstance must be an object of type Throwable class or subclass of Throwable class.

Example:

```
import java.util.Scanner;
public class ThrowableDemo
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the value of dividend");
        int a=sc.nextInt();
        System.out.println("Enter the value of divisor");
        int b=sc.nextInt();
        int res=0;
        try
        {
            if(b==0)
            {
                throw  new ArithmeticException("Divide By zero attempt is made");
            }
            res=a/b;
            System.out.println("The result is"+res);
        }
        catch(ArithmeticException e)
        {
            System.out.println(e.getMessage());
        }
    }

}
```

Output:

```
C:\Users\Hp-User\Desktop\Java programs\exception>java ThrowableDemo
Enter the value of dividend
8
Enter the value of divisor
2
The result is4

C:\Users\Hp-User\Desktop\Java programs\exception>java ThrowableDemo
Enter the value of dividend
8
Enter the value of divisor
0
Divide By zero attempt is made
```

Throwing your own exceptions / creating your own exception subclass

- We can create our own exception class by defining it as a subclass of Throwable classs or its subclasses such as Exception, ArithmeticException etc.
- Then we can use throw keyword to throw our own exception also.

Example:

JAVA program to demonstrate custom exception class

```
import java.util.Scanner;
public class ExceptionDemo
{
	public static void main(String [] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter dividend");
		int a=sc.nextInt();
		System.out.println("Enter divisor");
		int b=sc.nextInt();
		try
		{
			if(b==0)
			{
				throw new CustomException("Attempt to divide by zero");
			}
			int res=a/b;
			System.out.println("The result is"+res);
			}
		catch(CustomException e)
		{
			System.out.println(e.getMessage());
		}
	}
}

class CustomException extends Exception
{
	public CustomException(String message)
	{
		super(message);
	}
}
```

Output:

```
C:\Users\Hp-User\Desktop\Java programs\exception>java ExceptionDemo
Enter dividend
8
Enter divisor
2
The result is4

C:\Users\Hp-User\Desktop\Java programs\exception>java ExceptionDemo
Enter dividend
8
Enter divisor
0
Attempt to divide by zero
```