Assignment No. 2

Name : Ninad Rao Batch : C Roll No. 54

Exception Handling

When an Exception occurs the normal flow of the program is disrupted and the program / application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that normal flow of the application can be maintained. It is a mechanism to handle runtime errors such as :

1. ClassNotFoundException
2. IOException
3. SQLException
4. RemoteException etc.

There are three types of exceptions :

1. **Checked exception** : The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2. **Unchecked exception** : The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3. **Error** : Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

There are five keywords which are used in handling exceptions in Java :

1. **try** : Java try block is used to enclose the code that might throw an exception. It must be used within the method. If an exception occurs at the particular statement of try block, the rest of the block code will not execute.
2. **catch** : Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception or the generated exception type.
3. **finally** : Java finally block is a block that is used to execute important code such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not.
4. **throw** : It is used to throw an exception.

5. **throws** : It is used to declare exceptions. It specifies that there may occur an exception in the method. It is always used with method signatures.

Example with try-catch block :

```
public class Test_exception_handling
{
	public static void main(String args[])
	{
		try
		{
			int data=100/0;
		}
		catch(ArithmeticException e)
		{
			System.out.println(e);
		}
		System.out.println("rest of the code...");
	}
}
```

Output :

```
java.lang.ArithmeticException: / by zero
rest of the code...
```

Example with try-catch-finally block :

```
public class Try_catch_finally
{
	public static void main(String args[])
	{
		try
		{
			int data=100/0;
		}
		catch(ArithmeticException e)
		{
			System.out.println(e);
		}
		finally
		{
			System.out.println("finally block is executed");
		}
```

```
            System.out.println("rest of the code...");
        }
}
```

Output :

```
java.lang.ArithmeticException: / by zero
finally block is executed
rest of the code...
```

Common scenarios of Java Exceptions :

1. **ArithmeticException** : If we divide any number by zero, there occurs an ArithmeticException.

2. **NullPointerException** : If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

3. **NumberFormatException** : The wrong formatting of any value may occur NumberFormatException. Suppose, a string variable that has characters, converting this variable into digit will occur NumberFormatException.

4. **ArrayIndexOutOfBoundsException** : If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException.

5. **StringIndexOutOfBoundsException** : If you are inserting any value in the wrong index of the string, it would result in StringIndexOutOfBoundsException.

Multi-catch block : A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. At a time only one exception occurs and at a time only one catch block is executed.

Example :

```
public class MultipleCatchBlock
{
        public static void main(String[] args)
        {
                try
                {
                        int a[]=new int[5];
                        a[5]=30/0;
                }
                catch(ArithmeticException e)
```

```
            {
                System.out.println("Arithmetic Exception occurs");
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                System.out.println("ArrayIndexOutOfBounds Exception occurs");
            }
            catch(Exception e)
            {
                System.out.println("Parent Exception occurs");
            }
            System.out.println("rest of the code");
        }
}
```

Output :
Arithmetic Exception occurs
rest of the code

Nested try block : The try block within a try block is known as nested try block in java. Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Example :

```
public class NestedTry
{
    public static void main(String[] args)
    {
        int a[]=new int[5];
        try
        {
            try
            {
                a[5]=30/0;
            }
            catch(ArithmeticException e)
            {
                System.out.println("Arithmetic Exception occurs");
            }
            try
            {
```

```
                        System.out.println(a[10]);
                    }
                    catch(ArrayIndexOutOfBoundsException e)
                    {
                        System.out.println("ArrayIndexOutOfBounds Exception occurs");
                    }
                }
                catch(Exception e)
                {
                    System.out.println("Parent Exception occurs");
                }
                System.out.println("rest of the code");
        }
}
```

Output :
Arithmetic Exception occurs
ArrayIndexOutOfBounds Exception occurs
rest of the code

User Defined exceptions : In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as user-defined or custom exceptions.