

5.1 Dealing with Errors

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors such as `ClassNotFoundException`, `IOException`, `SQLException`, `RemoteException`, etc.* so that the normal flow of the application can be maintained.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

```
statement 1;  
statement 2;  
statement 3;  
statement 4;  
statement 5;//exception occurs  
statement 6;  
statement 7;  
statement 8;  
statement 9;  
statement 10;
```

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

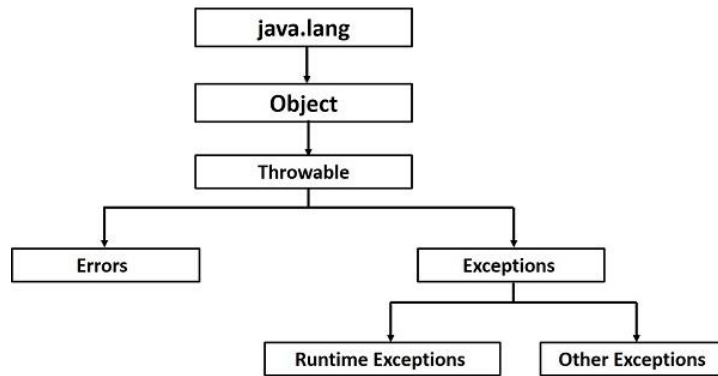
Hierarchy of Java Exception classes

All exception classes are subtypes of the `java.lang.Exception` class. The exception class is a subclass of the `Throwable` class. Other than the exception class there is another subclass called `Error` which is derived from the `Throwable` class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of memory. Normally, programs cannot recover from errors.

The `Exception` class has two main subclasses: `IOException` class and `RuntimeException` Class.

Unit 5: Exception Handling



Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

1) **Checked Exception:** The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) **Unchecked Exception:** The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) **Error:** Error is irrecoverable. Some examples of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

5.2 Catching Exception

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following –

Syntax

```
try {  
    // Protected code  
} catch (ExceptionName e1) {  
    // Catch block  
}
```

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.

Unit 5: Exception Handling

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Example

The following is an array declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
// File Name : ExceptTest.java
import java.io.*;

public class ExceptTest {

    public static void main(String args[]) {
        try {
            int a[] = new int[2];
            System.out.println("Access element three :" + a[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown  :" + e);
        }
        System.out.println("Out of the block");
    }
}
```

This will produce the following result –

Output

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

Multiple Catch Blocks

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following –

Syntax

```
try {
// Protected code
} catch (ExceptionType1 e1) {
// Catch block
} catch (ExceptionType2 e2) {
// Catch block
} catch (ExceptionType3 e3) {
// Catch block
}
```

The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls

Unit 5: Exception Handling

through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Example

```
import java.util.Scanner;
public class Test
{
    public static void main(String args[])
    {
        Scanner scn = new Scanner(System.in);
        try
        {
            int n = Integer.parseInt(scn.nextLine());

            if (99%n == 0)
                System.out.println(n + " is a factor of 99");
        }
        catch (ArithmeticException ex)
        {
            System.out.println("Arithmetic " + ex);
        }
        catch (NumberFormatException ex)
        {
            System.out.println("Number Format Exception " + ex);
        }
    }
}
```

Input 1:

GeeksforGeeks

Output 1:

Exception encountered java.lang.NumberFormatException: For input string: "GeeksforGeeks"

Input 2:

0

Output 2:

Arithmetic Exception encountered java.lang.ArithmeticException: / by zero

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

5.3 try, catch, throw, throws, and finally

Java try-catch block

TryCatchExample.java

```
public class TryCatchExample {  
  
    public static void main(String[] args) {  
        try  
        {  
            int data=50/0; //may throw exception  
        }  
        //handling the exception  
        catch(ArithmeticException e)  
        {  
            System.out.println(e);  
        }  
        System.out.println("rest of the code");  
    }  
}
```

Output:

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

Java throw Exception

In Java, exceptions allows us to write good quality codes where the errors are checked at the compile time instead of runtime and we can create custom exceptions making the code recovery and debugging easier.

The Java throw keyword is used to throw an exception explicitly.

We specify the **exception** object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.

The **syntax** of the Java throw keyword is given below.

throw Instance i.e.,

```
throw new exception_class("error message");
```

Let's see the example of throw IOException.

```
throw new IOException("sorry device error");
```

Where the Instance must be of type Throwable or subclass of Throwable. For example, Exception is the subclass of Throwable and the user-defined exceptions usually extend the Exception class.

Java throw keyword Example

Example 1: Throwing Unchecked Exception

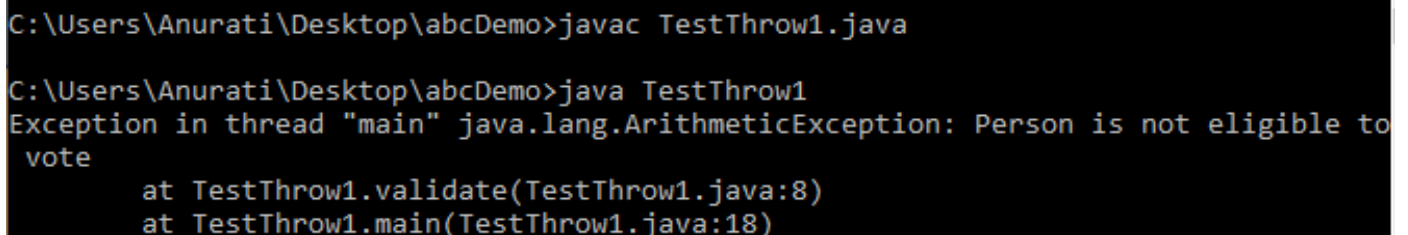
In this example, we have created a method named `validate()` that accepts an integer as a parameter. If the age is less than 18, we are throwing the `ArithmeticException` otherwise print a message welcome to vote.

TestThrow1.java

In this example, we have created the `validate` method that takes integer value as a parameter. If the age is less than 18, we are throwing the `ArithmeticException` otherwise print a message welcome to vote.

```
public class TestThrow1 {  
    //function to check if person is eligible to vote or not  
    public static void validate(int age) {  
        if(age<18) {  
            //throw Arithmetic exception if not eligible to vote  
            throw new ArithmeticException("Person is not eligible to vote");  
        }  
        else {  
            System.out.println("Person is eligible to vote!!");  
        }  
    }  
    //main method  
    public static void main(String args[]){  
        //calling the function  
        validate(13);  
        System.out.println("rest of the code...");  
    }  
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow1.java  
  
C:\Users\Anurati\Desktop\abcDemo>java TestThrow1  
Exception in thread "main" java.lang.ArithmeticException: Person is not eligible to  
vote  
    at TestThrow1.validate(TestThrow1.java:8)  
    at TestThrow1.main(TestThrow1.java:18)
```

The above code throw an unchecked exception. Similarly, we can also throw unchecked and user defined exceptions.

Unit 5: Exception Handling

Note: If we throw unchecked exception from a method, it is must to handle the exception or declare in throws clause.

If we throw a checked exception using throw keyword, it is must to handle the exception using catch block or the method must declare it using throws declaration.

Example 2: [TestThrow2.java](#)

```
import java.io.*;
public class TestThrow2 {

    //function to check if person is eligible to vote or not
    public static void method() throws FileNotFoundException {
        FileReader file = new FileReader("C:\abc.txt");
        BufferedReader fileInput = new BufferedReader(file);
        throw new FileNotFoundException();
    }
    //main method
    public static void main(String args[]){
        try
        {
            method();
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        System.out.println("rest of the code...");
    }
}
```

Output:

```
C:\Users\Anurati\Desktop\abcDemo>javac TestThrow2.java
C:\Users\Anurati\Desktop\abcDemo>java TestThrow2
java.io.FileNotFoundException
    at TestThrow2.method(TestThrow2.java:12)
    at TestThrow2.main(TestThrow2.java:22)
rest of the code...
```

Unit 5: Exception Handling

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as `NullPointerException`, it is programmers' fault that he is not checking the code before it being used.

Syntax of Java throws

```
return_type method_name() throws exception_class_name{  
  
    //method code  
  
}
```

Java throws Example

Let's see the example of Java throws clause which describes that checked exceptions can be propagated by throws keyword.

Testthrows1.java

```
import java.io.IOException;  
class Testthrows1{  
    void m() throws IOException{  
        throw new IOException("device error");//checked exception  
    }  
    void n() throws IOException{  
        m();  
    }  
    void p(){  
        try{  
            n();  
        }catch(Exception e){System.out.println("exception handled");}  
    }  
    public static void main(String args[]){  
        Testthrows1 obj=new Testthrows1();  
        obj.p();  
        System.out.println("normal flow...");  
    }  
}
```

Output:

```
exception handled  
normal flow...
```


Unit 5: Exception Handling

Rule: If we are calling a method that declares an exception, we must either caught or declare the exception.

The Finally Block

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax –

Syntax

```
try {  
    // Protected code  
} catch (ExceptionType1 e1) {  
    // Catch block  
} catch (ExceptionType2 e2) {  
    // Catch block  
} catch (ExceptionType3 e3) {  
    // Catch block  
}finally {  
    // The finally block always executes.  
}
```

Example

```
public class ExcepTest {  
  
    public static void main(String args[]) {  
        int a[] = new int[2];  
        try {  
            System.out.println("Access element three :" + a[3]);  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("Exception thrown  :" + e);  
        }finally {  
            a[0] = 6;  
            System.out.println("First element value: " + a[0]);  
            System.out.println("The finally statement is executed");  
        }  
    }  
}
```

Output

Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3

First element value: 6

The finally statement is executed

Note the following –

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.