Exception Handling

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. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

Major reasons why an exception Occurs

- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file

What is an Error in Java

In Java, an error is a subclass of Throwable that tells that something serious problem is existing and a reasonable Java application should not try to catch that error. Generally, it has been noticed that most of the occurring errors are abnormal conditions and cannot be resolved by normal conditions.

Example of Error

```
public class ErrorExample {
public static void recursiveMethod(int i){
    while(i!=0){
      i=i+1;
      recursiveMethod(i);
    }
}
  public static void main(String[] args){
    recursiveMethod(10);



  }
```

```
}
```

Output

```
Exception in thread "main" java.lang.StackOverflowError
  at ErrorExample.ErrorExample(Main.java:42)
```

Example of Exception

```
public class ExceptionExample {
  public static void main(String[] args){
    int x = 100;
    int y = 0;
    int z = x / y;
  }
}
```

Output

```
java.lang.ArithmeticException: / by zero
  at ExceptionExample.main(ExceptionExample.java:7)
```

here are the below points that differentiate between both terms:

Exception	Error
Can be handled	Cannot be handled.
Can be either checked type or unchecked type	Errors are of unchecked type
Thrown at runtime only, but the checked exceptions known by the compiler and the unchecked are not.	Occurs at the runtime of the code and is not known to the compiler.
They are defined in Java.lang.Exception package.	They are defined in Java.lang.Error package
Program implementation mistakes cause exceptions.	Errors are mainly caused because of the environment of the program where it is executing.

Exception Handling in Java

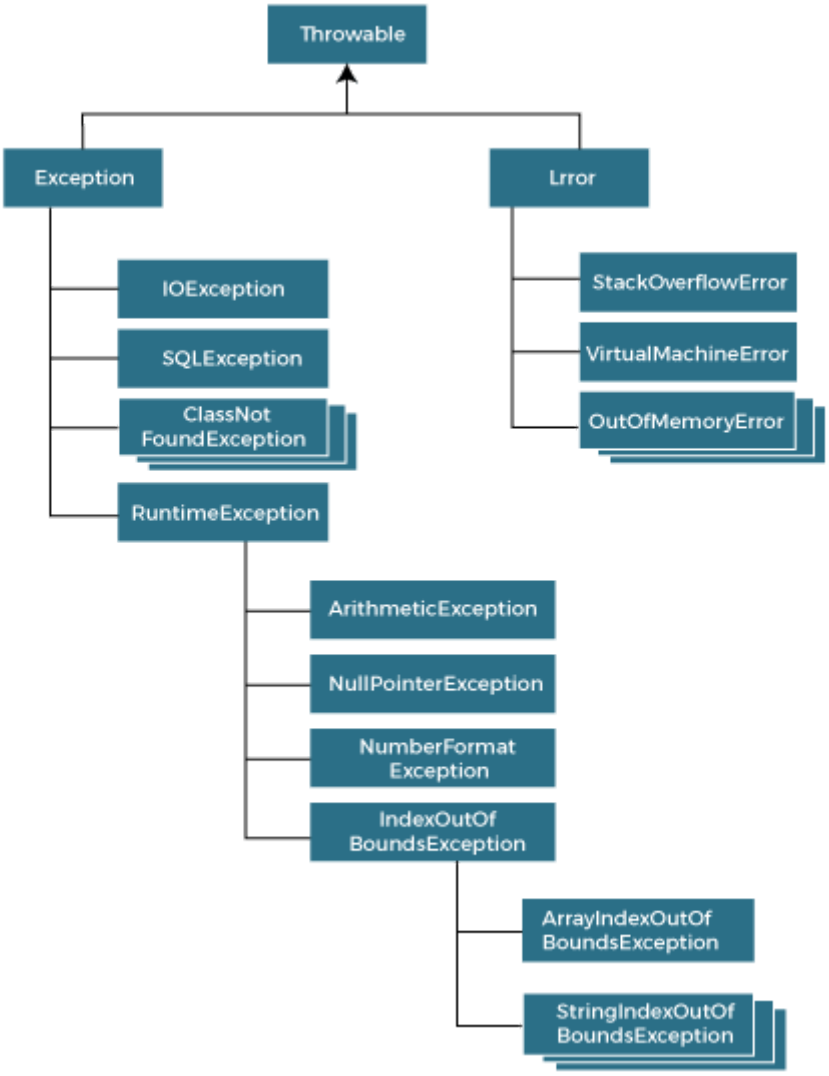
The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained. Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

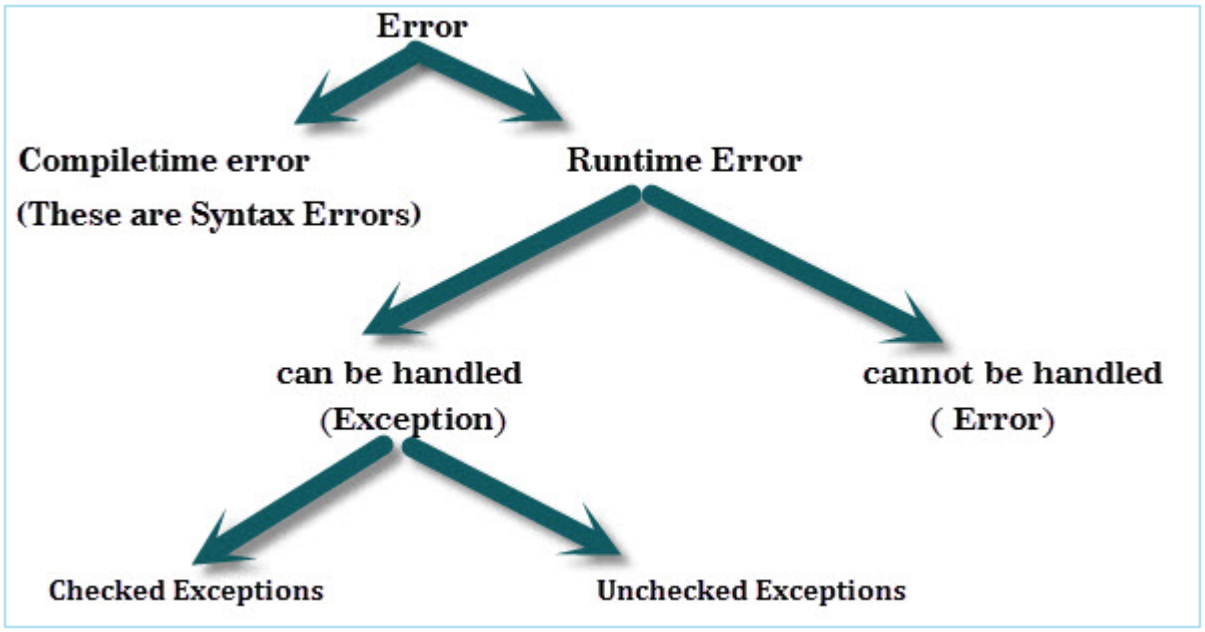
Advantage of Exception Handling

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.

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:





Types of Exceptions

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.

Exceptions can be categorized in two ways:

- Checked Exception
- Unchecked Exception

.

- **Checked Exceptions:** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
- **Eg.** ClassNotFoundException,IOException,SQLException,FileNotFoundException etc.

```
import java.io.*;
public class Checked {
public static void main(String[] args) {
FileReader file = new FileReader("D:\\balmiki\\data.txt");
BufferedReader input = new BufferedReader(file);
for (int c = 0; c < 3; c++)
System.out.println(input.readLine());
input.close();
}
}
```

After handling

```
import java.io.*;

public class Checked {

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

FileReader file = new FileReader("D:\\balmiki\\data.txt");

BufferedReader input = new BufferedReader(file);

for (int c = 0; c < 3; c++)

System.out.println(input.readLine());

input.close();
}
catch(Exception e)
{
System.out.println(e);
}
}

}
```

- **Unchecked Exceptions:** The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

ArrayIndexOutOfBoundsException

```
int a[]=new int[10];

System.out.println(a[20]);
public class Unchecked {

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

```
int Ary[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

System.out.println(Ary[11]);

}

}
```

Errors

An unrecoverable event that collapses the entire application program is called as an error.

Checked Exceptions	Unchecked Exceptions
Can be checked and handled during Compile-time	Cannot be checked nor be handled during Compile-time
Direct subclasses of exception class but do not inherit run-time exception	Direct subclasses of exception class but only inherits run-time exception
The compiler catches these exceptions in the compilation stage	The compiler cannot recognize and catch them during the compilation stage
Checked Exceptions are predictable failures	Uncheck exceptions are unpredictable failures, mostly caused by improper programming logic
Examples: SQL Exception IOException	Examples: ArithmeticException NullPointerException

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

Java Exception Keywords

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.

Without try catch block

```
public class TryCatchExample1 {

   public static void main(String[] args) {

      int data=50/0;

      System.out.println("Helllo World");

   }

}
```

With try catch block

```
public class JavaExceptionExample{
  public static void main(String args[]){
   int a=2,b=0,data;
   try{

data=a/b;
```

```
System.out.println(data);
   }catch(ArithmeticException e){System.out.println(e);}

   System.out.println("Hello World");
  }
}
```

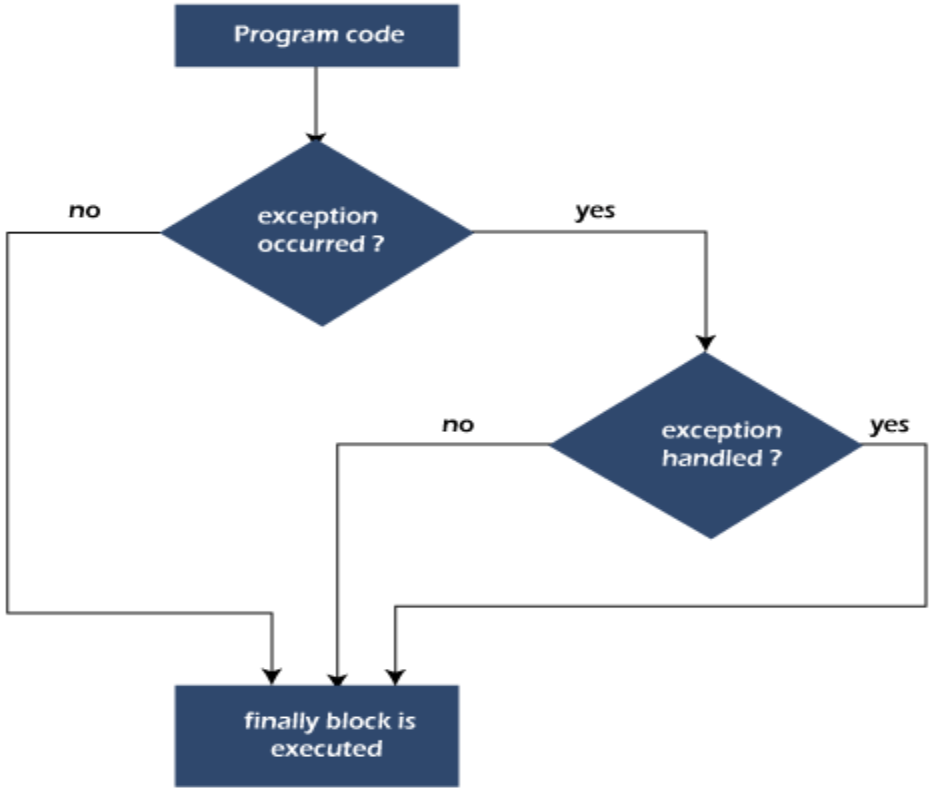
Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

The finally block follows the try-catch block.

Flowchart of finally block




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

```
        int data=25/0;
        System.out.println(data);
       }
       catch(NullPointerException e){
        System.out.println(e);
       }
        finally {
        System.out.println("finally block is always executed");
       }
       System.out.println("rest of the code...");
       }
      }
```

maathiko program ma ArithmeticException generate hunchha tara NullPointerException handle gareko chha teso garda ni finally block execute vayeko chha. aba tyo NullPointerException lai hatayera ArithmeticException rakhera herne.

Java throw keyword

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.

We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception. We will discuss custom exceptions later in this section.

```
public class TestThrow1
   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!!");
      }
   }
   public static void main(String args[]){
      validate(13);
      System.out.println("rest of the code...");
  }
}
```

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. **The throws keyword is used to handle checked exceptions.**

```
import java.io.*;
public class Checked {
public static void main(String[] args)throws FileNotFoundException,IOException
 {
FileReader file = new FileReader("D:\\balmiki\\data.txt");
BufferedReader input = new BufferedReader(file);
for (int c = 0; c < 3; c++)
System.out.println(input.readLine());
input.close();
}
}
```

Java Custom Exception

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

```
class InvalidAgeException  extends Exception
{
   public InvalidAgeException (String str)
   {
      super(str);
   }
}
public class TestCustomException1
{
   static void validate (int age) throws InvalidAgeException{
     if(age < 18){
      throw new InvalidAgeException("age is not valid to vote");
   }
     else {
      System.out.println("welcome to vote");
      }
    }
   public static void main(String args[])
   {
      try
      {
         validate(13);
```

```
        }
        catch (InvalidAgeException ex)
        {
            System.out.println("Caught the exception");
            System.out.println("Exception occured: " + ex);
        }
        System.out.println("rest of the code...");
    }
}
```