JAVA

Class 35

Agenda

Exception and types of Exception

Exception Handling in JAVA

final block

final vs finally

throw vs throws

What is Exception?

An **Exception** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

Exception is an object which is thrown at runtime.

When Exception occurs system generates an error message. We can manage this using exception handler.

The process of converting system error messages into user friendly error message is known as Exception handling.

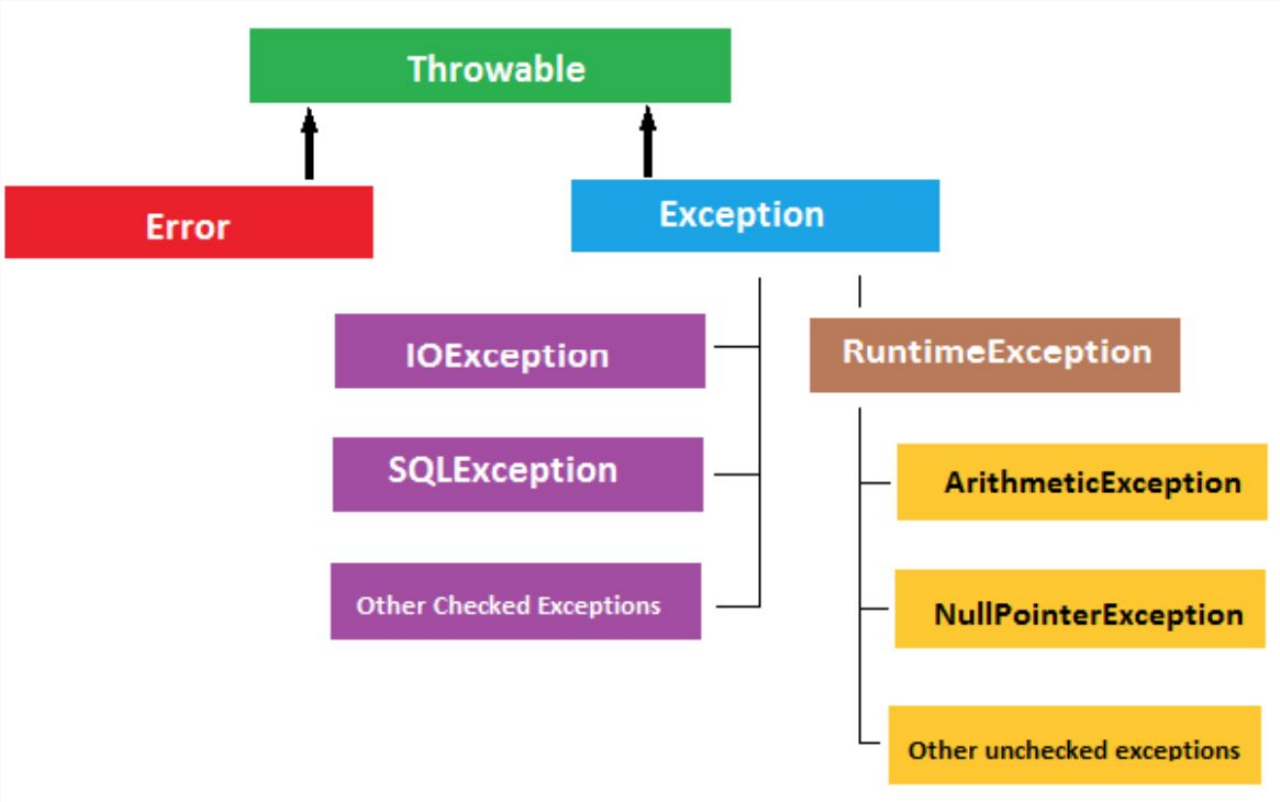
What is Exception?

An exception can occur for many different reasons:

- A user has entered an invalid data, such as division by zero
- A file that needs to be opened cannot be found.
- In the middle of communications the JVM has run out of memory.

Exception in Java

Exceptions are objects, and objects are defined using classes. The root class for exceptions is java.lang.Throwable



Difference in Error & Exception

Both Error and Exception are derived from **java.lang.Throwable** in Java but the main difference between Error and Exception is the kind of error they represent.

Errors are "serious issues and abnormal conditions that most applications should not try to handle". Error defines problems that are not expected to be caught under normal circumstances by the program. For example memory error, hardware error, JVM error etc.

Exceptions are "conditions that a reasonable application might want to catch or conditions within the code". A developer can handle such conditions and take necessary corrective actions.

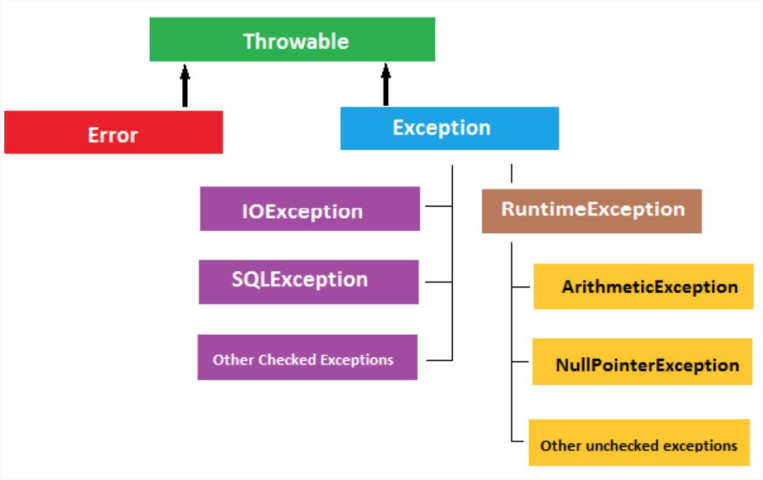
Error	Exception
Can't be handle.	Can be handle.
Example: ***OutOfMemoryError***	Example: **ClassNotFoundException**

Why use Exception Handling

We can convert system error message into user friendly error message by using exception handling feature of java.

When we divide any number by zero then system generate / by zero exception, so this is not understandable by user and we can convert this message into user friendly error message like “Don't enter zero for denominator”

Exception in Java



Exception class is for exceptional conditions that program should catch. This class is extended to create user specific exception classes

RuntimeException is a subclass of Exception. Exceptions under this class are automatically defined for programs.

Errors are typically ignored in code because you can rarely do anything about an error.

Example : if OutOfMemoryError will arise. This type of error is not possible to handle in code.

Types of Exception

The exception that can be predicted by the programmer or we can say all exceptions other than Runtime Exceptions are known as Checked exceptions. As the compiler checks them during compilation to see whether the programmer has handled them or not.	The unchecked exception is ignored at compile time. These exceptions need not be included in any method's throw list because the compiler does not check to see if a method handles or throws these exceptions Unchecked Exceptions comprise of run time exceptions (of type ***RuntimeException*** or its subclasses) and errors (of type Error or its subclasses).

Checked Exception

- ***ClassNotFoundException*** Class not found
- ***InstantiationException*** Attempt to create an object of an abstract class or interface
- ***IllegalAccessException*** Access to a class is denied.
- ***SQLException*** Database access error
- ***FileNotFoundException*** Attempt to open file at specified path has failed

Unchecked Exceptions

- ***ArithmeticException*** Arithmetic error, such as divide-by-zero.
- ***NullPointerException*** Invalid use of a null reference.
- ***ArrayIndexOutOfBoundsException*** Array index is out-of-bounds.

- ***StringIndexOutOfBoundsException*** an index is either negative or greater than the size of the string.
- ***NegativeArraySizeException*** Array created with a negative size.

Scenarios where unchecked exceptions can occur.

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

```
int a=50/0;//ArithmeticException
```

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Java Exception Handling Keywords

There are 5 keywords used in java exception handling

1) try
2) catch
3) finally
4) throws
5) throw

Java Exception Handling Keywords

try-catch block is used to handle the exceptions in Java.

Inside **try block** we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

Java Exception Handling Keywords

try Block, is used to enclose the code that might throw an exception.
A **try block** is **always** followed by a catch block, which handles the exception that occurs in associated try block.
A **try block must** follow by a **catch** block or **finally** block or both. try must be used within the method.

catch Block is used to handle the Exception.
It must be associated with a try block.
The corresponding **catch block** executes if an exception of a particular type occurs within the try block

try block

Important points about try block

- If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.
- After executing appropriate catch block, even through we use run time statement, CPU control never goes to try block to execute the rest of the statements.
- Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.
- Each and every try block must contains at least one catch block. But it is highly recommended to write multiple catch blocks for generating multiple user friendly error messages.
- One try block can contains another try block that is nested or inner try block can be possible.

try catch block

try-catch block is used to handle the exceptions in Java.

Inside **catch block** we write the block of statements which will generates user friendly error messages

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

catch block

catch block important points

- Catch block will execute exception occurs in try block.
- We can write multiple catch blocks for generating multiple user friendly error messages to make your application strong.
- At a time only one catch block will execute out of multiple catch blocks.
- In catch block we declare an object of subclass and it will be internally referenced by JVM.

catch block

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

        int i = 10;
        int j;
        j = i/0 ; // It will throw exception

        SOP("Value of j is " +j);
        SOP("Unhandled Exception");
    }
}
```

catch block

Suppose we have a code of thousand lines and some exceptions occur in the middle of code somewhere. So if the exception is not handled then the rest of the code after exception will not execute.

So if the exception is not handled then the execution abrupt forcefully.

If you handle the exception then an exception will be reported and next code will be executed in normal flow.

We use try catch block to handle the exception.

catch block

```
public class SampleClass {
    public static void main(String[] args) {
        try{
            int i = 10;
            int j;
            j = i/0 ; // It will throw exception
        }catch (Throwable t){
            // Print the exception
            System.out.println(t);
            SOP("Unhandled Exception");
        }
    }
}
```

catch block

A method can **throw more than one** exceptions. To handle these exceptions we need multiple catch blocks associated with the single try block.

A single **try block** can have multiple catch blocks. This is required when the try block has statements that generate different types of exceptions.

If the first catch block contains the Exception class object then the subsequent catch blocks are never executed.

The last catch block in multiple catch blocks must contain the Exception class object.

This is because the java compiler gives an error saying that the subsequent catch blocks haven't been reached.

catch block

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

Rule: At a time only one Exception is occurred and at a time only one catch block is executed.

Rule: All catch blocks must be ordered from most specific to most general i.e. catch for *ArithmeticException* must come before catch for *Exception* .

```
public class MultiCatch {

    public static void main(String argv[]) {

     int[] array = { 17, 99, 8 };

        try {

                System.out.println(array[3]);

                } catch (ArithmeticException e) {
                        System.out.println(e.getMessage());
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                }

                System.out.println("Code after an exception");

    }
}
```

Task

1. How would handle InputMismatchException? Input Mismatch Exception when user enters mismatch value then programmer expected.

2. Create a static method that will return a List of Exceptions.

- Inside your method create objects of 4 exception classes using try and catch blocks and store them inside your list.
- Call your method inside main and print name and details of all Exception objects.

finally block

- The code inside the finally clause will always be executed, even if an exception is thrown from within the try or catch block.
- If your code has a return statement inside the try or catch block, the code inside the finally-block will get executed before returning from the method.
- The main usage of finally block is to do clean up job. Keeping cleanup code in a finally block is always a good practice, even when no exceptions are occurred.

finally block

- Writing finally block is optional.
- You can write finally block for the entire java program
- For each try block there can be zero or more catch blocks, but only one finally block.

Note ： The finally block will not be executed if program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort).

finally block

```
public class TestFinallyBlock {
  public static void main(String args[]) {
     try  {
          int data=25/0;
          System.out.println(data);
      }  catch(ArithmeticException e)   {
           System.out.println(e);
      }  finally     {
          SOP("finally block is always executed");
       }
     SOP("rest of the code...");
  }
}
```

final vs finally

Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.
Final is a keyword.	Finally is a block.

Throws keyword

throws is a keyword in java language which is used to throw the exception which is raised in the called method to it's calling method

throws keyword always followed by method signature

“throws keyword” is mainly used for handling checked exception as using throws we can declare multiple exceptions.

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 normal flow can be maintained.

```
returnType methodName() throws

exceptionClassName {
//method code
}
```

Throws keyword

```
import java.io.IOException;
public class DemoThrows {
    public static void main(String[] args) {

        try {
            displayFile();
        } catch (IOException ioExp) {
            ioExp.printStackTrace();
        }
    }

    static void displayFile() throws IOException {
        SOP("In displayFile method");
        throw new IOException();
    }
}
```

Throw keyword

- In java throw keyword is used to throw an exception explicitly.
- Using “throw keyword” we can throw checked, unchecked and user -defined exceptions.
- The throw keyword is mainly used to throw the custom exception.
- The only object of the Throwable class or its subclasses can be thrown.
- Program execution stops on encountering throw statement, and the closest catch statement is checked for a matching type of exception.
- Note: throw keyword always should exist within method body.
- whenever method body contain throw keyword than the call method should be followed by throws keyword.

Syntax

```
class className {
    returntype method(...) throws Exception_class {
        throw(Exception obj)
   }
}
```

Throw keyword

```
class AgeValidator{


   static void validateStudent(int age) {

        if(age<5) {
              throw new ArithmeticException("age to play more");
       } else
              SOP("welcome to school");
       }
   }

   public static void main(String args[]) {
        validateStudent(4);
        SOP("rest of the code...");
    }
}
```

throw vs throws

throw is used to explicitly throw an exception.	throws is used to declare an exception.
throw is followed by an instance.	throws is followed by class.
throw is used within the method body.	throws is used with the method signature.
You cannot throw multiple exception	You can declare multiple exception e.g. public void method()throws IOException,SQLException.
throw keyword can be used in switch case in Java	throws keyword can not be used anywhere except on method declaration line.