Exceptional handling, Multiple catch, Nested try

Exceptional handling

What is an Exception?

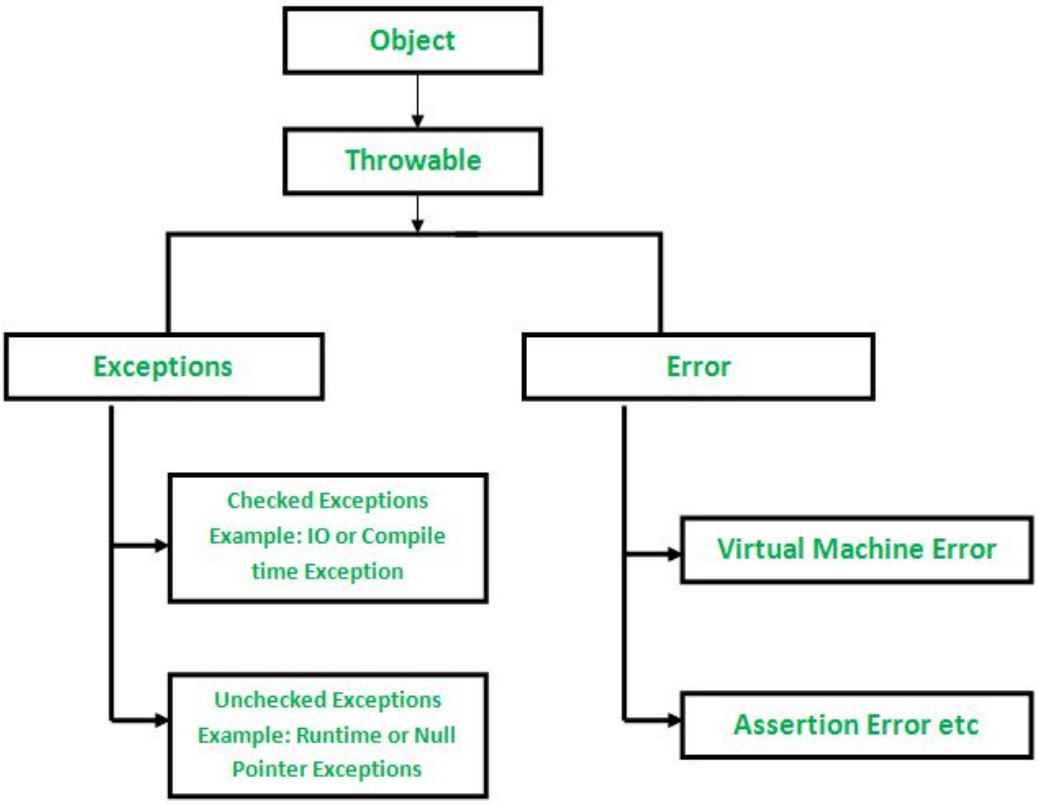
An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions.

Error vs Exception

Error: An Error indicates a serious problem that a reasonable application should not try to catch.

Exception: Exception indicates conditions that a reasonable application might try to catch.

All exception and error types are subclasses of class **Throwable**, which is the base class of hierarchy.One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception.Another branch,**Error** is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.



Difference between Checked and Unchecked 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.

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 use exception handling.

```
statement 1;

statement 2;

statement 3;//exception occurs

statement 4;

statement 5;
```

Suppose there are 5 statements in your program and there occurs an exception at statement 3, the rest of the code will not be executed i.e. statement 4 to 5 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Multiple catch block

Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use the java multi-catch block.

Points to remember

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Example 1

```
Class Main
{
public static void main(String args[])
{
        try
{
int a[]=new int[6];
a[5]=30/0;
}
catch(ArithmeticException e)
{
  System.out.print("Arithmetic exception occurs");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.print("Array out of bound exception occurs");
}
```

```
catch(Exception e)
{
System.out.print("Exception occurs");
}
System.out.print("\nRest of the code will be executed");
}
}
```

output

Arithmetic exception occurs
Rest of the code will be executed

Explanation

In the above example,in try block we generate ArithmeticException,the corresponding
arithmetic exception catch block is invoked.

Example 2

```
Class Main
{
public static void main(String args[])
{
try
{
int a[]=new int[6];
System.out.print(a[10]);
}
catch(ArithmeticException e)
{
  System.out.print("Arithmetic exception occurs");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.print("Array index out of bound exception occurs");
}
catch(Exception e)
{
```

```
System.out.print("Exception occurs");
}
System.out.print("\nRest of the code will be executed");


}
}
```

output

Array index out of bound exception occurs
Rest of the code will be executed

Explanation

In the above example,in the try block we generate an ArrayOutOfBound exception,the corresponding exception catch block is invoked.

Example 3

```
Class Main
{
public static void main(String args[])
{
try
{
String s=null;
System.out.print(s.length()); //NullPointerException
}
catch(ArithmeticException e)
{
  System.out.print("Arithmetic exception occurs");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.print("Array index out of bound exception occurs");
}
catch(Exception e)
{
System.out.print("Exception occurs");
```

```
}
System.out.print("\nRest of the code will be executed");


}
}
```

output

Exception occurs
Rest of the code will be executed

Explanation

In the above example, we generate NullPointerException, but didn't provide the corresponding exception type. In such a case, the catch block containing the parent exception class **Exception** will be invoked.

Example 4

```
Class Main
{
public static void main(String args[])
{
        try
{
int a[]=new int[6];
a[2]=10/0;
System.out.print(a[10]);
}
catch(ArithmeticException e)
{
  System.out.print("Arithmetic exception occurs");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.print("Array index out of bound exception occurs");
}
catch(Exception e)
{
System.out.print("Exception occurs");
```

```
}
System.out.print("\nRest of the code will be executed");


}
}
```

output

```
Arithmetic exception occurs
Rest of the code will be executed
```

Explanation

In the above example, the try block contains two exceptions. But at a time only one exception occurs and its corresponding catch block is invoked.

Example 5

```
Class Main
{
public static void main(String args[])
{
try
{
int a[]=new int[6];
a[2]=10/0;
System.out.print(a[10]);
}
catch(Exception e)
{
System.out.print("Exception occurs");
}
catch(ArithmeticException e)
{
  System.out.print("Arithmetic exception occurs");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.print("Array index out of bound exception occurs");
}
```

```
System.out.print("\nRest of the code will be executed");

}
}
```

output

Compile-time error

Explanation

In the above example, to handle the exception without maintaining the order of exceptions (i.e. from most specific to most general)

Nested try block

The try block within a try block is known as a nested try block in java.

Why use nested try block

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.

Syntax

```
try
{statement 1;
  statement 2;
    try
     { statement 1;
        statement 2;
      }
    catch(Exception e)
     {
```

```
        }
}
catch(Exception e)
{
}
```

Example

```
class Main
{public static void main(String args[])
{
try
{
try
{System.out.print("going to divide");
 int a=39/0;
}
catch(ArithmeticException e)
{
System.out.print(" "+e);
}
try
{int a[]=new int[5];
 a[6]=10;
  System.out. println(a[6]);
}
catch(ArrayIndexOutOfBoundsException e)
{
   System.out.print(" "+e);
}
System.out.println("\nother statement");
}
catch(Exception e)
{
   System.out.print(e);
}
System.out.print("normal flow");
}
}
```

Output

```
going to divide
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: 6
other statement
normal flow
```