

EXCEPTION HANDLING IN JAVA

Errors

1. Compile time errors or Syntax errors

missing;, misspelling, using variables without declaration, assigning values, underflow, overflow, data type mismatch, etc.

2. Runtime errors

Division by zero, invalid conversion, accessing an invalid index of array etc.

Exception

- Runtime error results are an Exception.
- The exception is a representation of an error condition.
- Exception leads to abnormal termination of the program, giving some garbage message on the screen.
- **In Java, an exception is an event that disrupts the normal flow of the program. It is an object, which is thrown at runtime.**
- Whenever exceptions occur, java creates an object for the exception class or its deriving class.

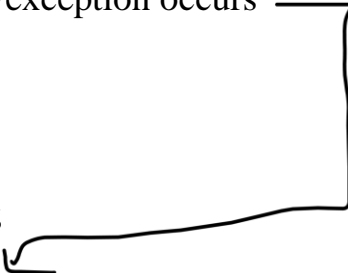
Exception Handling:

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IO, SQL, Remote etc.

The advantage of exception handling is **that it maintains the normal flow of the application/program**. Exception normally disrupts the normal flow of the application that is why we use exception handling.

Example:

//Program

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

Suppose there are 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

Common Example of Exceptions:

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

```
int a=50;
```

```
int b=0;
```

```
int c=a/b;
```

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

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

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

The wrong formatting of any value may occur `NumberFormatException`. Suppose I have a string variable that have characters, converting this variable into digit will occur **`NumberFormatException`**.

1. `String s="abc";`
2. `int i=Integer.parseInt(s);//NumberFormatException`

If you are inserting any value in the wrong index, it would result **`ArrayIndexOutOfBoundsException`** as shown below:

1. `int a[]=new int[5]; index 0-4`
2. `a[10]=50; //ArrayIndexOutOfBoundsException`

There are 5 keywords used in Java exception handling.

1. `try`
2. `catch`
3. `finally`
4. `throw`
5. `throws`

Java try block { }

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Example:

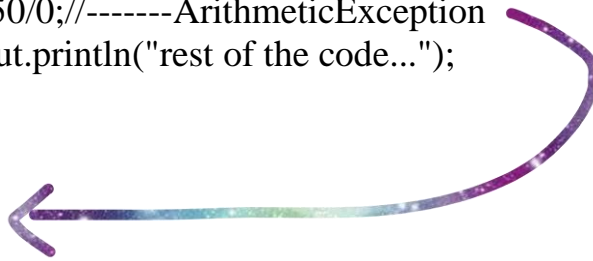
1. try{
2. //code that may throw an exception -----object of Exception class or it derive a class
3. }catch(Exception_class_Name ref){ }

Example:

1. try{
2. //code that may throw exception
3. }finally{ }

Program: Problem without Try Catch

1. public class Testtrycatch1 {
2. public static void main(String args[]){
3. int data=50/0;//-----ArithmeticException
4. System.out.println("rest of the code...");
5. }
6. }



Output:

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac Testtrycatch1.java


D:\1 Java\Programs>java Testtrycatch1
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Testtrycatch1.main(Testtrycatch1.java:3)
```

The code after the exception will not be executed.

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

Solution by exception handling

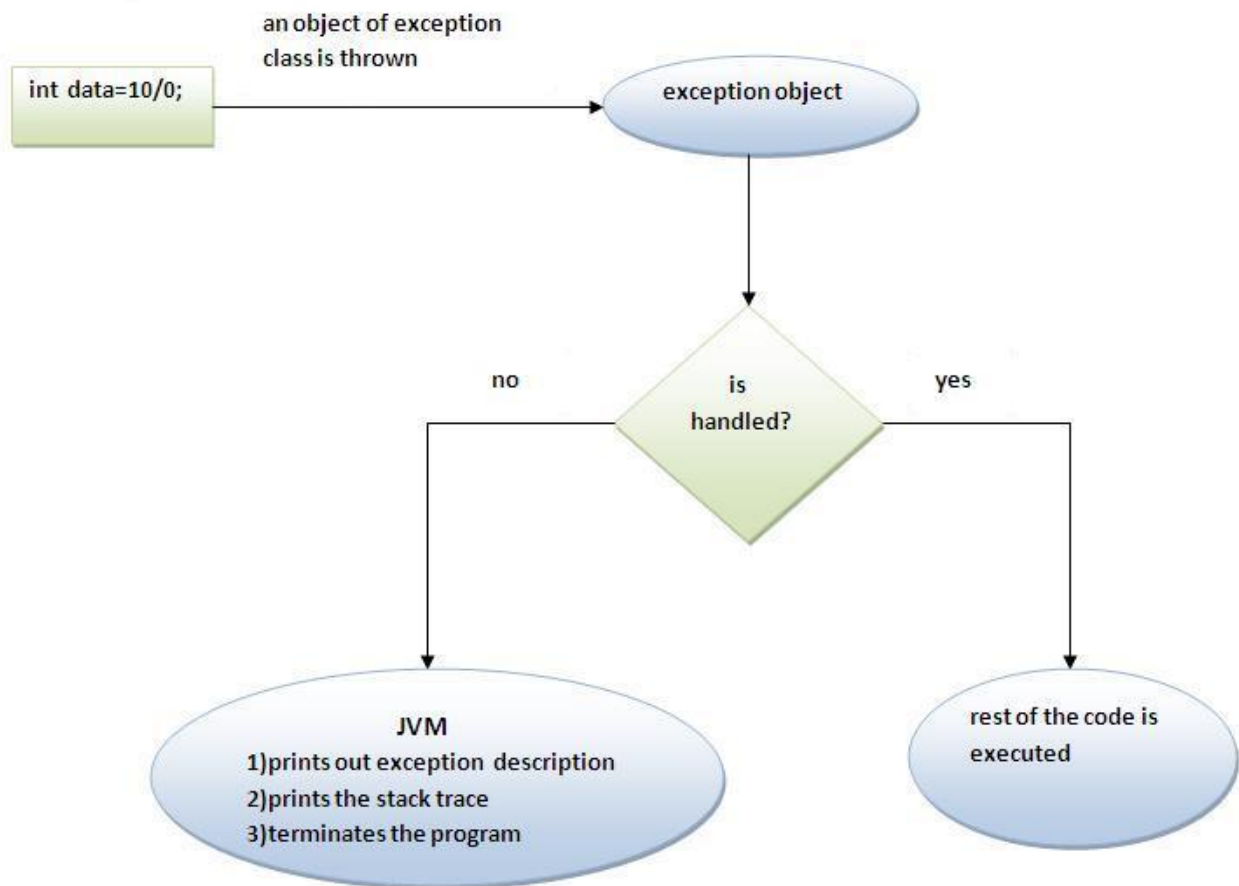
```
1. public class Testtrycatch2{
2.     public static void main(String args[]){
3.         try
4.         {
5.             int data=50/0; //it will throw an object of class ArithmeticException
6.         }
7.         catch(ArithmeticException e)// or catch(Exception e)
8.         {
9.             System.out.println(e);}
10.    }
11.    System.out.println("rest of the code...");
12.    }
```

 C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>javac Testtrycatch2.java

D:\1 Java\Programs>java Testtrycatch2
java.lang.ArithmeticException: / by zero
rest of the code...
```

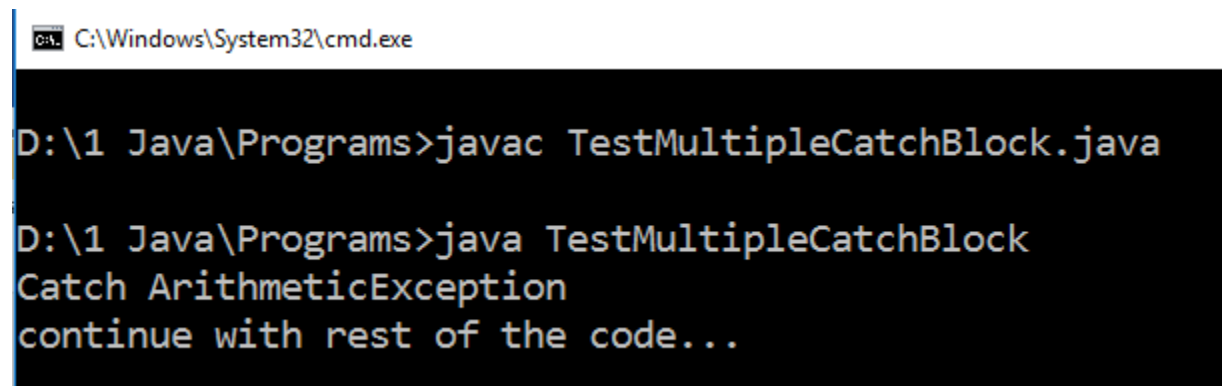
Working of java try-catch block



Java Multi catch block

```
public class TestMultipleCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5]; //index 0-4
            a[5]=30/0; //throw xxxxxxxxxx Exception
        }
        catch(ArithmeticException e)
        {
            System.out.println("Catch ArithmeticException");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

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



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe". The command prompt displays the following sequence of commands and output:

```
D:\1 Java\Programs>javac TestMultipleCatchBlock.java
D:\1 Java\Programs>java TestMultipleCatchBlock
Catch ArithmeticException
continue with rest of the code...
```

Note:

- At a time only one Exception occurs and at a time only one catch block is executed.

```
public class OrderofCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5]; //index 0-4
            a[5]=30/0;
        }

        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }
        catch(ArithmeticException e)
        {
            System.out.println("Catch ArithmeticException");
        }

        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }
        System.out.println("continue with rest of the code...");
    }
}
```



```
C:\Windows\System32\cmd.exe
OrderofCatchBlock.java:11: error: exception ArithmeticException
has already been caught
        catch(ArithmeticException e)
        ^
OrderofCatchBlock.java:15: error: exception ArrayIndexOutOfBoundsExcep
tion has already been caught
        catch(ArrayIndexOutOfBoundsException e)
        ^
2 errors
```

```
public class OrderofCatchBlock1 {
    public static void main(String args[]){
        try{
            int a[]=new int[5];
            a[5]=30/0;
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }
        System.out.println("continue with rest of the code...");
    }
}
```

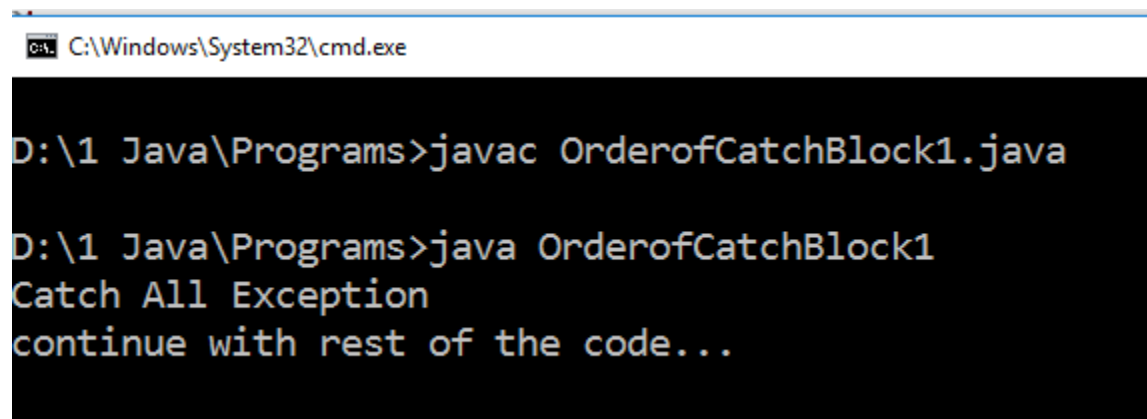
```
D:\1 Java\Programs>javac OrderofCatchBlock1.java
OrderofCatchBlock1.java:15: error: exception ArrayIndexOutOfBoundsExcep
tion has already been caught
        catch(ArrayIndexOutOfBoundsException e)
        ^
1 error
```

```

class OrderofCatchBlock1 {
    public static void main(String args[]){
        try{
            int a[]=new int[5]; ----→one exception object
            a[5]=30/0;
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

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

```



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe". The command prompt is open at the directory "D:\1 Java\Programs". The user has entered the command "javac OrderofCatchBlock1.java" to compile the program. Then, the user has entered the command "java OrderofCatchBlock1" to run the program. The output of the program is displayed as "Catch All Exception" followed by "continue with rest of the code..." on the next line.

```

C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac OrderofCatchBlock1.java

D:\1 Java\Programs>java OrderofCatchBlock1
Catch All Exception
continue with rest of the code...

```

Note:

All catch blocks must be ordered from most specific (derive classes) to most general (base class) i.e. catch for `ArithmeticException` must come before catch for `Exception`.

```

public class TestMultipleCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5]; //index 0-4
            a[5]=5; //throw xxxxxxxxxx Exception
        }
        catch(ArithmeticException e)
        {
            System.out.println("Catch ArithmeticException");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

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

```

```

F:\Java Code>java TestMultipleCatchBlock
Catch ArrayIndexOutOfBoundsException
continue with rest of the code...

```

```

public class TestMultipleCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5];
            a[5]=30/0; //throw xxxxxxxxxx Exception

        }
        catch(ArithmeticException e)
        {
            System.out.println("Catch ArithmeticException");
        }
    }
}

```

```
    }  
    catch(ArrayIndexOutOfBoundsException e)  
    {  
        System.out.println("Catch ArrayIndexOutOfBoundsException");  
    }  
    catch(Exception e)  
    {  
        System.out.println("Catch All Exception");  
    }  
  
    System.out.println("continue with rest of the code...");  
}  
}
```

```
F:\Java Code>java TestMultipleCatchBlock  
Catch ArithmeticException  
continue with rest of the code...
```

Nested Try Blocks

```
class NestedTry{

public static void main(String args[]){
    try {
        try{
            System.out.println("Division of number occur");
            int b =39/0;
        }
        catch(ArithmeticException e)
            {
                System.out.println(e);
            }
    }
    try
    {
        int a[]=new int[5];
        a[5]=4;
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
        System.out.println(e);
    }
}
catch(Exception e)
{
    System.out.println("It handle all type of exceptions");
    System.out.println(e);
}
System.out.println("Program Continued..");
}
}
```

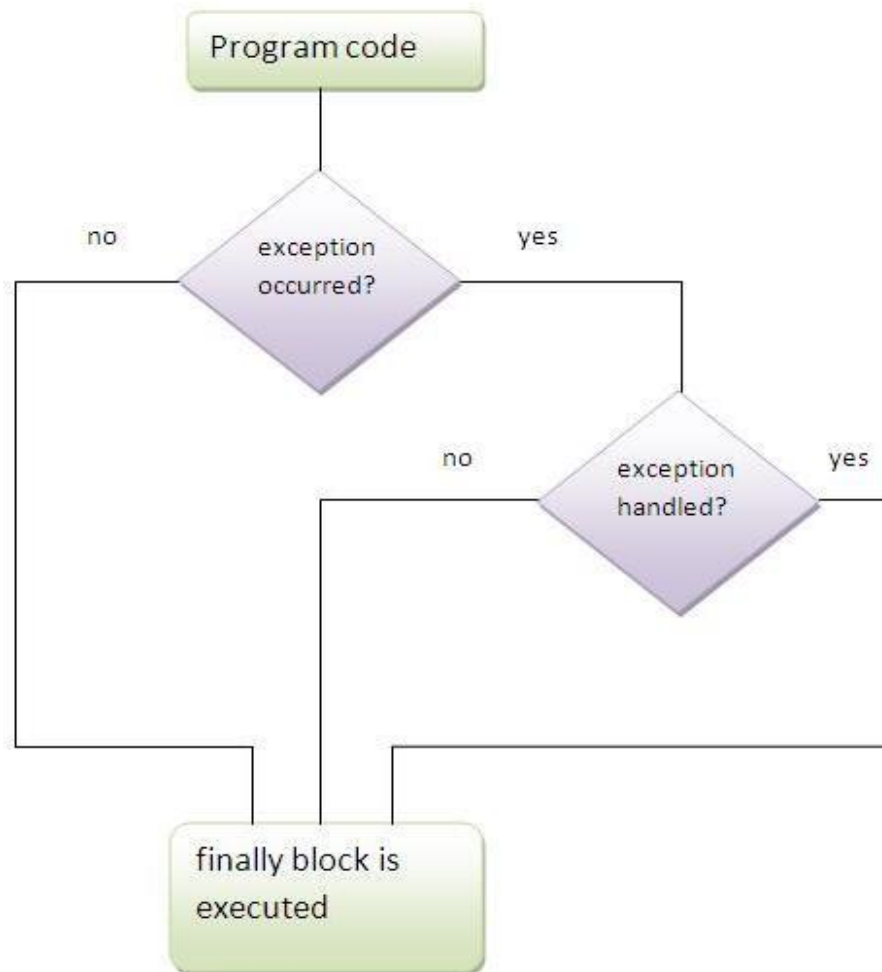
```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac NestedTry.java

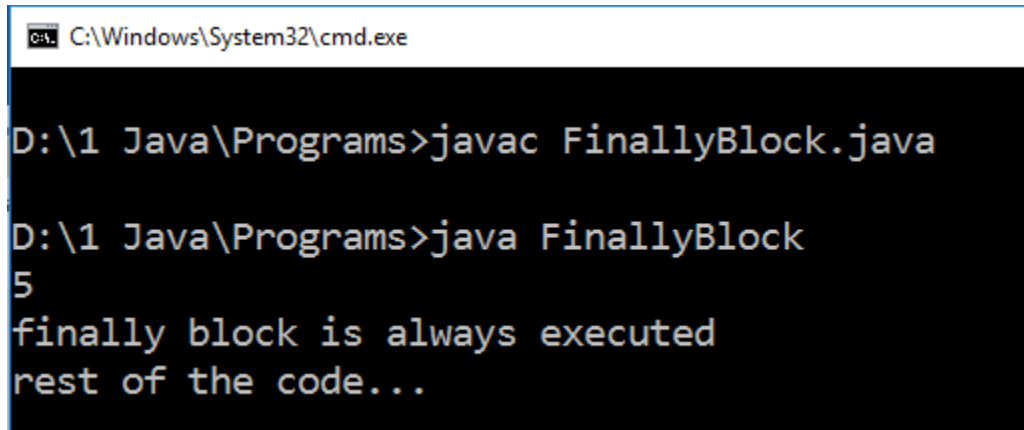
D:\1 Java\Programs>java NestedTry
Division of number occur
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: 5
Program Continued..
```

Java Finally Block

- Java finally block is a block that is used to execute important code such as closing connection, stream, etc.
- Finally, can be used as a substitute or along with a catch.
- There can be only one finally.
- Finally, does not receive any argument/object.
- Java finally block is always executed whether an exception is handled or not.
- If you don't handle the exception, before terminating the program, JVM executes the finally block (if any).
- The finally block will not be executed if the program exits (either by calling `System.exit()` or by causing a fatal error that causes the process to abort).



```
class FinallyBlock{
    public static void main(String args[]){
        try{
            int data=25/5; //
            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...");
    }
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe". The command prompt is at the directory "D:\1 Java\Programs". The user has entered the command "javac FinallyBlock.java" to compile the program. Then, they entered "java FinallyBlock" to run it. The output of the program is displayed on the next two lines: "5" and "finally block is always executed", followed by a blank line and "rest of the code..." on the final line.

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac FinallyBlock.java

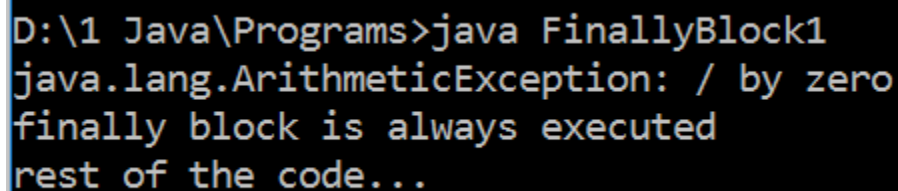
D:\1 Java\Programs>java FinallyBlock
5
finally block is always executed
rest of the code...
```



```

class FinallyBlock1{
    public static void main(String args[]){
        try{
            int data=25/0; // //throw
            System.out.println(data);
        }
        catch(ArithmeticException e)
        {
            System.out.println(e);}
        finally{
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}

```



```

D:\1 Java\Programs>java FinallyBlock1
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...

```

```

class FinallyBlock1{
    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...");
    }
}

```

```
}  
C:\Windows\System32\cmd.exe  
D:\1 Java\Programs>javac FinallyBlock1.java  
D:\1 Java\Programs>java FinallyBlock1  
finally block is always executed  
Exception in thread "main" java.lang.ArithmeticException: / by  
zero  
    at FinallyBlock1.main(FinallyBlock1.java:4)
```

finally block without try block ❌

Multiple finally block ❌

throw Keyword in Java

Basically, the exceptions are thrown by Java runtime systems. It is not all the time possible for a Java runtime system to throw the exception. This burden may come on the programmer's shoulders. That means sometimes we must explicitly mention the throw of exception. The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.

Syntax: throw exception

Program:

```
import java.util.*;
class TestThrow
{
    public static void main(String args[])
    {
        System.out.println("Enter a even number");
        Scanner s=new Scanner(System.in);
        int n=s.nextInt();
        if(n%2==0)
        {
            System.out.println("Number " +n + " is even");
        }
        else
        {
            throw new ArithmeticException("Invalid Number");
        }
    }
}
```

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac TestThrow.java

D:\1 Java\Programs>java TestThrow
Enter a even number
24
Number 24 is even

D:\1 Java\Programs>java TestThrow
Enter a even number
23
Exception in thread "main" java.lang.ArithmeticException: Invalid Number
    at TestThrow.main(TestThrow.java:15)
```

Note: After the throw statement, no other statement should be placed in that block. If any statement is placed after the throw statement in that block, Java will give an **unreachable statement**, compile time error.

```
import java.util.*;
class TestThrow
{
    public static void main(String args[])
    {
        System.out.println("Enter an even number");
        Scanner s=new Scanner (System.in);
        int n=s.nextInt();
        if(n%2==0)
        {
            System.out.println("Number " +n + " is even");
        }
        else
        {
            throw new ArithmeticException("Invalid Number");
            System.out.println("rest of program");
        }
    }
}
```

```
D:\1 Java\Programs>javac TestThrow.java
TestThrow.java:16: error: unreachable statement
        System.out.println("rest of program");
        ^
1 error
```

throws:

To prevent this compile time error, we can handle the exception in two ways:

1. By using try catch (Already Discussed)
 2. By using the **throws** keyword
- throws is a keyword in Java that is used in the signature of the method to indicate that this method might throw one of the listed type exceptions.
 - The caller to these methods must handle the exception using a try-catch block.
 - throws keyword is required only to convince the compiler and usage of the throws keyword does not prevent abnormal termination of the program.
 - With the help of the throws keyword, we can provide information to the caller about the method of the exception.
 - throws keyword is required only for checked exceptions and usage of the throws keyword for unchecked exceptions (run time exception) is meaningless.

Syntax:

Return type method_name (parameters) throws exception_list

An exception list is a comma-separated list of all the exceptions that a method might throw.

We can use the throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

```

import java.util.*;
class ExThrows0
{
    static void divideNum()
    {
        int a=50;
        System.out.println("Enter the value of b");
        Scanner s=new Scanner(System.in);
        int b=s.nextInt();
        int c=a/b;
        System.out.println("Value of c is:"+c);
    }
    public static void main(String args[])
    {
        divideNum();
        System.out.println("Rest of the code");
    }
}

```

C:\Windows\System32\cmd.exe

```

D:\1 Java\Programs>javac ExThrows0.java

D:\1 Java\Programs>java ExThrows0
Enter the value of b
5
Value of c is:10
Rest of the code

D:\1 Java\Programs>java ExThrows0
Enter the value of b
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExThrows0.divideNum(ExThrows0.java:10)
    at ExThrows0.main(ExThrows0.java:16)

```

```
import java.util.*;
class ExThrows1
{
    static void divideNum() throws ArithmeticException
    {
        int a=50;
        System.out.println("Enter the value of b");
        Scanner s=new Scanner(System.in);
        int b=s.nextInt();
        int c=a/b;
        System.out.println("Value of c is:"+c);
    }

    public static void main(String args[])
    {
        try{
            divideNum();
        }
        catch(ArithmeticException e)
        {
            System.out.println("caught Exception"+e);
        }
        System.out.println("Rest of the code");
    }
}
```


C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>javac ExThrows1.java
```

```
D:\1 Java\Programs>java ExThrows1
```

```
Enter the value of b
```

```
5
```

```
Value of c is:10
```

```
Rest of the code
```

```
D:\1 Java\Programs>java ExThrows1
```

```
Enter the value of b
```

```
0
```

```
caught Exceptionjava.lang.ArithmeticException: / by zero
```

```
Rest of the code
```

Java Custom/User Define Exception

If you are creating your own Exception that is known as a custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user needs. With the help of custom exceptions, you can have your own exception and message.

The general form of creating an exception is:
Throwable t = new Throwable("my Exception");
Exception t=new Exception("my Exception");

Constructors:

- Throwable();
- Throwable(String s);
- Exception();
- Exception (String s);

```
class CustomException
{
public static void main(String args[])
{
Throwable t = new Throwable("CustomException");
try
{
    System.out.println("\n A new exception is thrown");
    throw t;
}
catch(Throwable e)
{
    System.out.println("\n The exception is caught here.");
    System.out.println("\n The exception is:"+e);
} }}
```

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac CustomException.java

D:\1 Java\Programs>java CustomException

A new exception is thrown

The exception is caught here.

The exception is:java.lang.Throwable: CustomException
```

```
class CustomException
{
public static void main(String args[])
{
Exception t = new Exception("CustomException");
try
{
    System.out.println("\n A new exception is thrown");
    throw t;
}
catch(Exception e)
{
    System.out.println("\n The exception is caught here.");
    System.out.println("\n The exception is:"+e);
}
}
```

```
D:\1 Java\Programs>javac CustomException.java
```

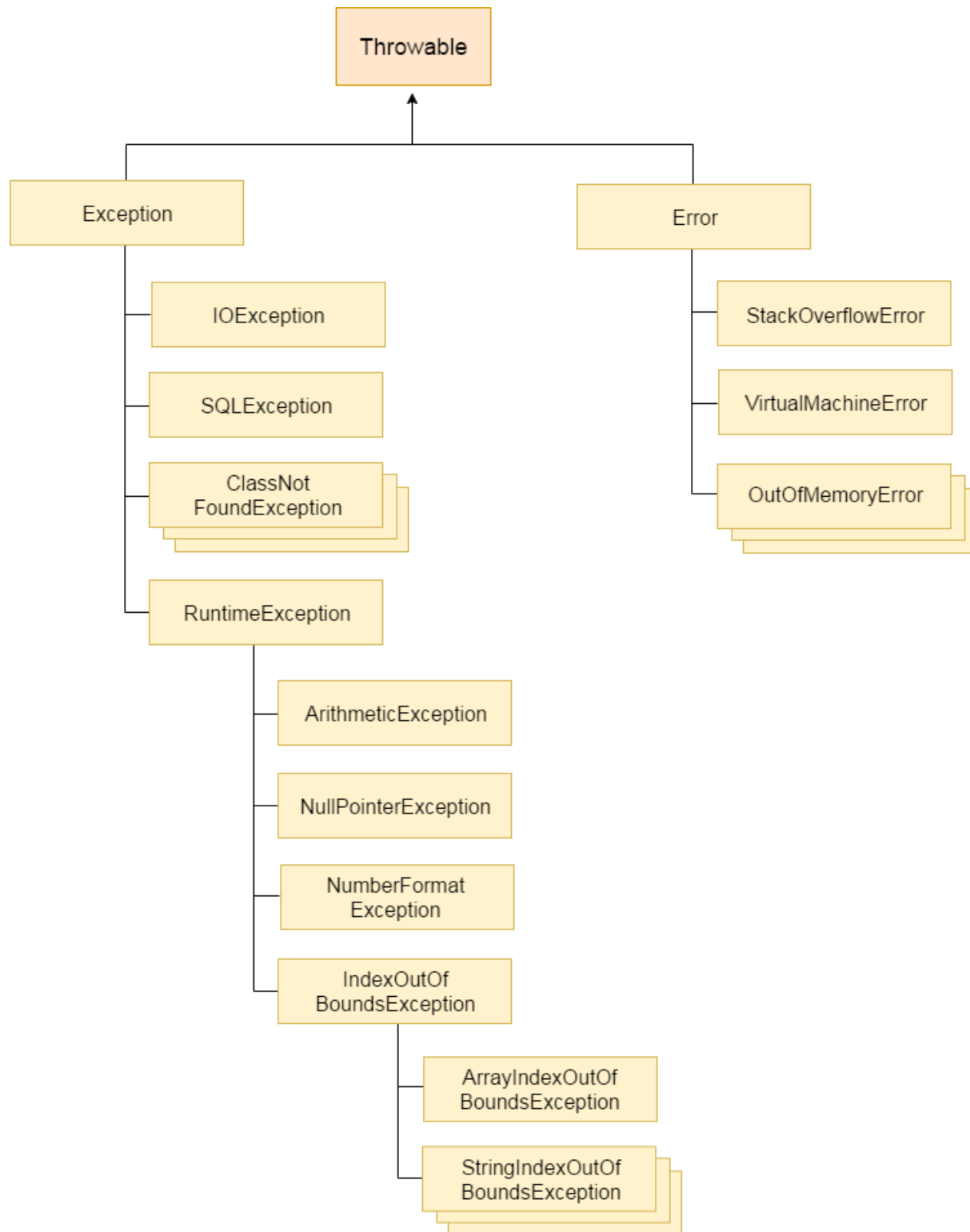
```
D:\1 Java\Programs>java CustomException
```

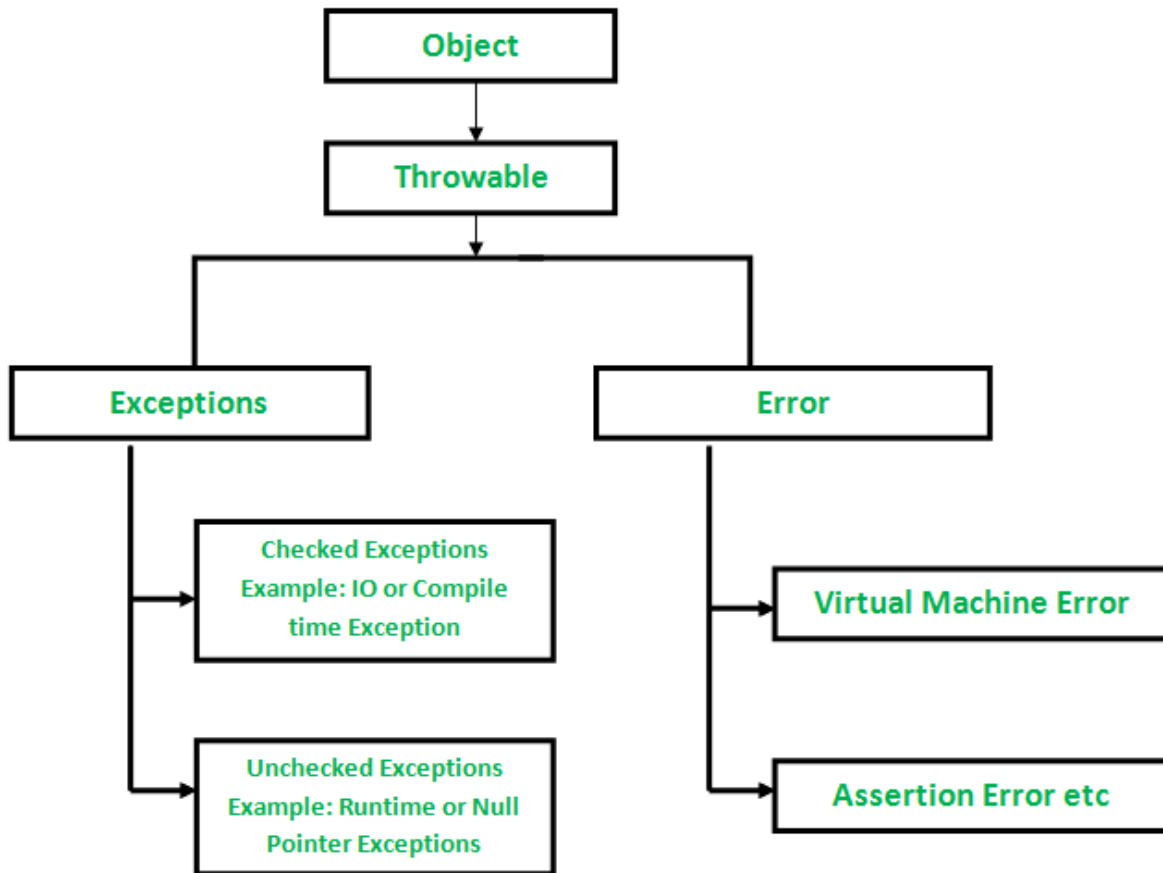
```
A new exception is thrown
```

```
The exception is caught here.
```

```
The exception is:java.lang.Exception: CustomException
```

Hierarchy of Java Exception classes

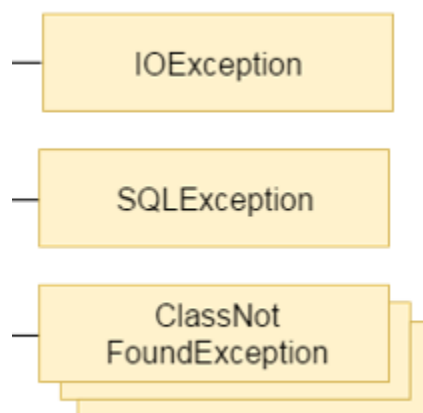




Checked vs. Unchecked Exceptions in Java

In Java, there are two types of exceptions:

1) Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.



2) Unchecked are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

