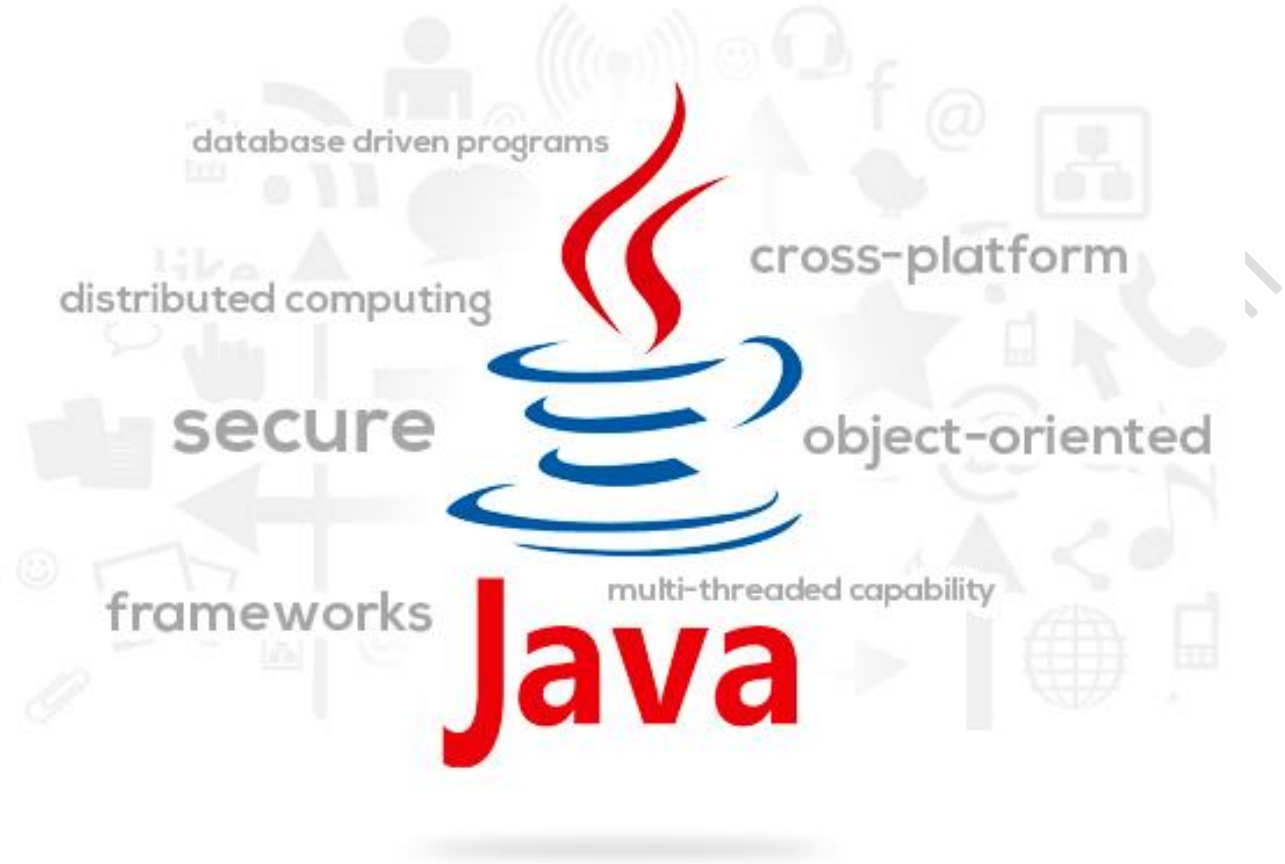




# Yadab Raj Ojha

**Sr. Java Developer / Lecture**

Email: yadabrajojha@gmail.com

Blog: http://yro-tech.blogspot.com/

Java Tutorial: https://github.com/yrojha4ever/JavaStud

LinkedIn: https://www.linkedin.com/in/yrojha

Twitter: https://twitter.com/yrojha4ever


Part 3

## Generics & Exceptions

- Java Generics Basic
- Class and method level Generic
- Handling Exceptions
- throws and throw
- try with resource
- Effective Exception Hierarchy

Java Date

- Date Format
- Simple Date Format
- Joda Time Library

Java Reflection API

instance of operator

# Exception Handling in Java

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

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

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

## Common scenarios where exceptions may occur

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

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

3.
```
String s="java";
int i=Integer.parseInt(s);//NumberFormatException
```

4.
```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

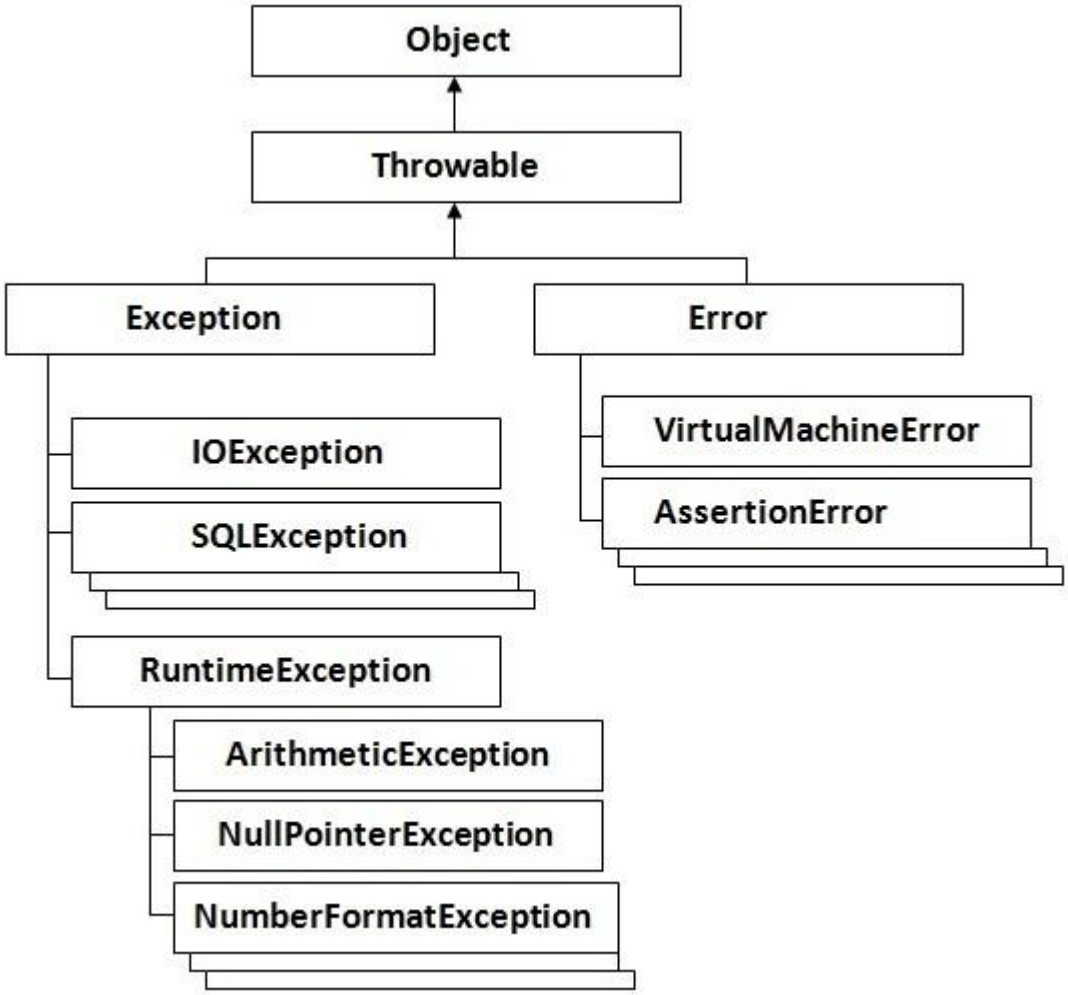
PgmWithoutException.java

```
1  package exc;
2
3  public class PgmWithoutException {
4
5      public static void main( String[] args ) {
6          int no1 = 100;
7          int no2 = 0;
8
9          int value = no1 / no2;
10         System.out.println( value );
11
12         double deposit = no1 + 1000.0;
13         System.out.println( "Deposit Amont: " + deposit );
14     }
15 }
16
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets

```
<terminated> PgmWithoutException [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at exc.PgmWithoutException.main(PgmWithoutException.java:9)
```

# Hierarchy of Java Exception classes




# Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception(Compile Time Exception)
2. Unchecked Exception(Runtime Exception)
3. Error

# Difference between checked and unchecked exceptions

### 1) Checked (Compile Time) Exception

The classes that extend 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 (Runtime) Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

---

## Java Exception Handling Keywords

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.

### *Syntax of java try-catch*

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

### *Syntax of try-finally block*

```
try{
//code that may throw exception
}finally{}
```

# 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.

CompileTimeException.java

Checked Exception

```
1  package exc;
2
3  import java.io.FileInputStream;
5
6  public class CompileTimeException {
7      public static void main( String[] args ) {
8          try {
9              FileInputStream fi = new FileInputStream( "C:\\temp.text" );
10         } catch ( FileNotFoundException e ) {
11             e.printStackTrace( );
12         }
13
14         System.out.println( "Continue other operation here." );
15     }
16 }
17
```

FileInputStream.class

```
98      * @see        java.lang.SecurityManager#checkRead(java.lang.String)
99      */
100     public FileInputStream(String name) throws FileNotFoundException {
101         this(name != null ? new File(name) : null);
102     }
103
104     /**
105      * Creates a <code>FileInputStream</code> by
106      * opening a connection to an actual file,
107      * the file named by the <code>File</code>
108      * object <code>file</code> in the file system.
109      * A new <code>FileDescriptor</code> object
110      * is created to represent this file connection.
111      * <p>
112      * First, if there is a security manager,
113      * its <code>checkRead</code> method  is called
114      * with the path represented by the <code>file</code>
115      * argument as its argument.
116      * <p>
117      * If the named file does not exist, is a directory rather than a regu
118      * file, or for some other reason cannot be opened for reading then a
119      * <code>FileNotFoundException</code> is thrown.
120      *
121      * @param      file   the file to be opened for reading.
122      * @exception  FileNotFoundException  if the file does not exist,
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Log

```
<terminated> CompileTimeException [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 13, 2015, 4:12:05 PM)
java.io.FileNotFoundException: C:\temp.text (The system cannot find the file specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:146)
        at java.io.FileInputStream.<init>(FileInputStream.java:101)
        at exc.CompileTimeException.main(CompileTimeException.java:9)
Continue other operation here.
```

# Java catch multiple exceptions

MultiCatchException.java

```java
package exc;

import java.io.FileInputStream;
import java.io.FileNotFoundException;

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

            int val = 45 / 0; // ERROR: so below line will be not executed.
            FileInputStream fi = new FileInputStream( "C:\\temp.text" ); //

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

        } catch ( FileNotFoundException e ) {
            System.out.println( "FileNotFoundException" );
            e.printStackTrace( );
        }

        System.out.println( "Continue program here!" );
    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Log

```
<terminated> MultiCatchException [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 13, 2015, 4:31:12 P
ArithmeticException
Continue program here!
java.lang.ArithmeticException: / by zero
        at exc.MultiCatchException.main(MultiCatchException.java:10)
```

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 .

```java
class TestMultipleCatchBlock1{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;
   }
   catch(Exception e){System.out.println("common task completed");}
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   System.out.println("rest of the code...");
 }
}
```

**Test it Now**

Output:

```
Compile-time error
```

# Java Nested try block

The try block within a try block is known as 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)
{
}
....
```

# Java nested try example

MultiTryCatch.java

```java
package exhand;

public class MultiTryCatch {

    public static void main( String[] args ) {

        try {

            MultiTryCatch obj = new MultiTryCatch( );
            int divideResult = 0;

            try {
                divideResult = obj.divide( 10, 0 ); //Error:divide by 0
            } catch ( ArithmeticException e ) {
                System.out.println( "ERROR: 10/0" );
                e.printStackTrace( );
            }

            try {
                int[] arr = new int[ 5 ];
                arr[ 10 ] = 50; //Error: index is 10, array size is only 5
            } catch ( ArrayIndexOutOfBoundsException e ) {
                System.out.println( "ERROR: Array Size: 5, item insert index:10" );
                e.printStackTrace( );
            }

            System.out.println( divideResult );

            String str = null;
            System.out.println( str.length( ) ); //Error: method call in null value

        } catch ( Exception e ) {
            System.out.println( "Exception Root." );
            e.printStackTrace( );
        }

    }

    public int divide( int a, int b ) {
        return a / b;
    }

}
```

Console

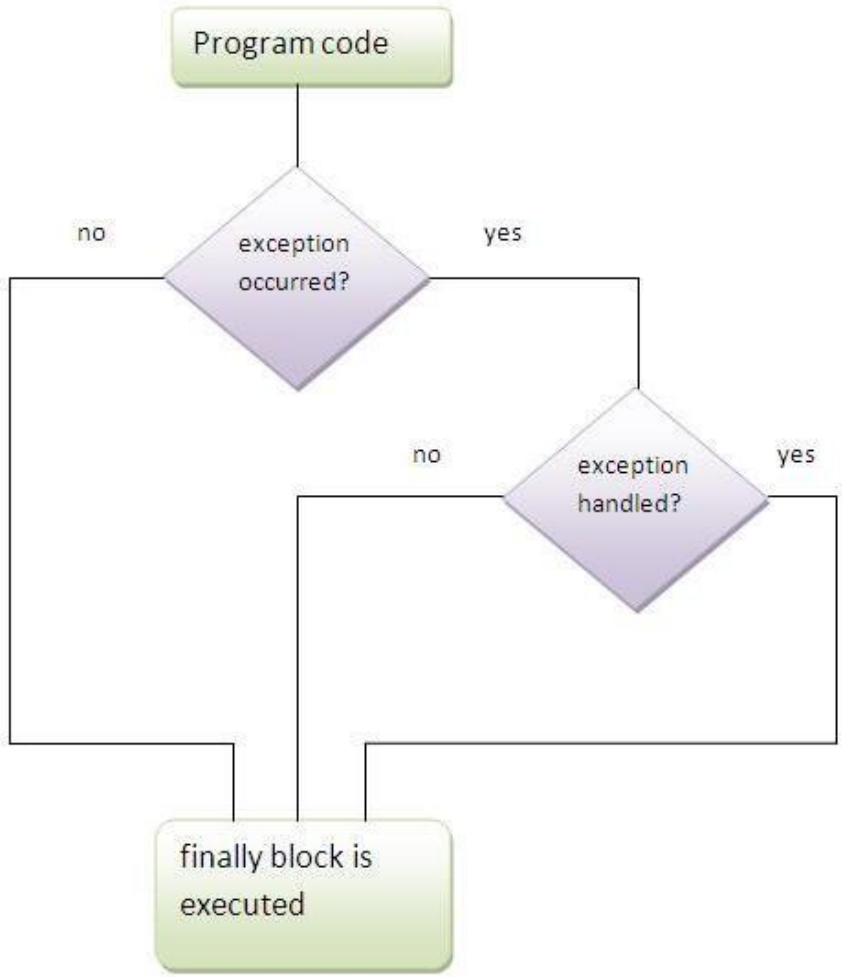
```
<terminated> MultiTryCatch [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\
ERROR: 10/0
java.lang.ArithmeticException: / by zero
        at exhand.MultiTryCatch.divide(MultiTryCatch.java:40)
        at exhand.MultiTryCatch.main(MultiTryCatch.java:13)
ERROR: Array has size 5, you are adding item at index:10
java.lang.ArrayIndexOutOfBoundsException: 10
        at exhand.MultiTryCatch.main(MultiTryCatch.java:21)
0
Exception Root.
java.lang.NullPointerException
        at exhand.MultiTryCatch.main(MultiTryCatch.java:30)
```

# Java finally block

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

Java finally block is always executed whether exception is handled or not.

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




Note: If you don't handle exception, before terminating the program, JVM executes finally block(if any).

## Why use java finally

- Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

```java
package exhand;

import java.util.Scanner;

public class FinallyTest {

    public static void main( String[] args ) {
        FinallyTest obj = new FinallyTest( );
        obj.finallyCallTest( );
    }

    public void finallyCallTest( ) {

        Scanner sc = null;
        try {
            sc = new Scanner( System.in );

            System.out.println( "Input a int value." );
            int val = sc.nextInt( ); // Error if you input String value.
            System.out.println( val );

        } catch ( ClassCastException e ) {
            e.printStackTrace( );

        } finally { // Always executed.
            sc.close( );
            System.out.println( "Scanner resource Closed." );
        }

    }
}
```

```
<terminated> FinallyTest [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw
Input a int value.
100
100
Scanner resource Closed.
```

```
<terminated> FinallyTest [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 14, 2015, 4:0
Input a int value.
JAVA
Scanner resource Closed.
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:909)
        at java.util.Scanner.next(Scanner.java:1530)
        at java.util.Scanner.nextInt(Scanner.java:2160)
        at java.util.Scanner.nextInt(Scanner.java:2119)
        at exhand.FinallyTest.finallyCallTest(FinallyTest.java:19)
        at exhand.FinallyTest.main(FinallyTest.java:9)
```

Class work: Make your own example for NullpointerException, ArithmenticException.

Rule: 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).

# Java throw/throws exception

## Java throw keyword

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

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

ThrowTest.java

1

```java
1  package exhand;
2
3  public class ThrowTest {
4
5      public static void main( String[] args ) {
6
7          validateAge( 17 ); // Exception came here.
8
9          System.out.println( "Your Other logic..." );
10     }
11
12     public static void validateAge( int age ) {
13         if ( age < 18 ) {
14             throw new ArithmeticException( "Age: " + age + " is not valid." );
15         } else {
16             System.out.println( "You are eligible to vote." );
17         }
18     }
19 }
20
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Log

```
<terminated> ThrowTest [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 14, 2015, 5:14:32 PM)
Exception in thread "main" java.lang.ArithmeticException: Age: 17 is not valid.
        at exhand.ThrowTest.validateAge(ThrowTest.java:14)
        at exhand.ThrowTest.main(ThrowTest.java:7)
```

ThrowExampleWithTryCatch.java

2

```java
package exhand;

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

        try {
            validateAge( 17 ); // Exception came here.

        } catch ( ArithmeticException e ) {
            e.printStackTrace( );
        }

        System.out.println( "Your Other logic..." );
    }

    public static void validateAge( int age ) {
        if ( age < 18 ) {
            throw new ArithmeticException( "Age: " + age + " is not valid." );
        } else {
            System.out.println( "You are eligible to vote." );
        }
    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Log

```
<terminated> ThrowExampleWithTryCatch [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 14, 2015, 5:19:09 PM)
java.lang.ArithmeticException: Age: 17 is not valid.
        at exhand.ThrowExampleWithTryCatch.validateAge(ThrowExampleWithTryCatch.java:18)
        at exhand.ThrowExampleWithTryCatch.main(ThrowExampleWithTryCatch.java:7)
Your Other logic...
```

# Java Exception propagation

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method,If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack.This is called exception propagation.

Rule: By default Unchecked Exceptions are forwarded in calling chain (propagated).

RuntimeExceptionAutoPropagate.java

```java
1  package exhand;
2
3  public class RuntimeExceptionAutoPropagate {
4
5      public static void main( String[] args ) {
6
7          RuntimeExceptionAutoPropagate obj = new RuntimeExceptionAutoPropagate( );
8          obj.method1( );
9
10         System.out.println( "Write your logic here..." );
11     }
12
13     public void method1( ) {
14         try {
15             method2( );
16         } catch ( ArithmeticException e ) {
17             System.out.println( "****ArithmeticException handled. ****" );
18             e.printStackTrace( );
19         }
20     }
21
22     public void method2( ) {
23         method3( );
24     }
25
26     public void method3( ) {
27         int a = 75 / 0; // Runtime(Unchecked) Exception
28         // It propagate error to caller method.
29     }
30
31 }
32
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Log

```
<terminated> RuntimeExceptionAutoPropagate [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 14, 2015, 6:03:40 PM)
java.lang.ArithmeticException: / by zero
****ArithmeticException handled. ****
Write your logic here...
        at exhand.RuntimeExceptionAutoPropagate.method3(RuntimeExceptionAutoPropagate.java:27)
        at exhand.RuntimeExceptionAutoPropagate.method2(RuntimeExceptionAutoPropagate.java:23)
        at exhand.RuntimeExceptionAutoPropagate.method1(RuntimeExceptionAutoPropagate.java:15)
        at exhand.RuntimeExceptionAutoPropagate.main(RuntimeExceptionAutoPropagate.java:8)
```

Rule: By default, Checked Exceptions are not forwarded in calling chain (propagated). We can propagate it using throws keyword

***Program which describes that checked exceptions are not propagated***

ThrowCheckExceptionCompileError.java

```java
package exhand;

import java.io.FileInputStream;
import java.io.FileNotFoundException;

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

        ThrowCheckExceptionCompileError obj = new ThrowCheckExceptionCompileError( );
        obj.method1( );

        System.out.println( "Write your logic here..." );
    }

    public void method1( ) {
        try {
            method2( );
        } catch ( ArithmeticException e ) {
            System.out.println( "****ArithmeticException handled. ****" );
            e.printStackTrace( );
        }
    }

    public void method2( ) {
        method3( );
    }

    public void method3( ) {
        try {
            FileInputStream ios = new FileInputStream( "C:\\temp.txt" );
        } catch ( FileNotFoundException e ) {
            throw new FileNotFoundException( "File Not Found." );
        }
    }

}
```

Compile Time Error

Unhandled exception type FileNotFoundException

2 quick fixes available:

Add throws declaration

Surround with try/catch

Press 'F2' for focus

# 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 normal flow 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 performing check up before the code being used.

## Which exception should be declared?

**Ans)** checked exception only, because:

- **unchecked Exception:** under your control so correct your code.
- **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

## Advantage of Java throws keyword

ThrowsUncheckPropagation.java

```java
package exhand;

import java.io.FileInputStream;

public class ThrowsUncheckPropagation {

    public static void main( String[] args ) {

        ThrowsUncheckPropagation obj = new ThrowsUncheckPropagation( );
        obj.method1( );

        System.out.println( "Write your logic here..." );
    }

    public void method1( ) {

        try {
            method2( );

        } catch ( FileNotFoundException e ) {
            System.out.println( "****FileNotFoundException is handled. ****" );
            e.printStackTrace( );
        }
    }

    public void method2( ) throws FileNotFoundException {
        method3( );
    }

    public void method3( ) throws FileNotFoundException {
        FileInputStream ios = new FileInputStream( "C:\\temp.txt" );
    }

}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Log

```
<terminated> ThrowsUncheckPropagation [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 14, 2015, 6:21
****FileNotFoundException is handled. ****
java.io.FileNotFoundException: C:\temp.txt (The system cannot find the file specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:146)
        at java.io.FileInputStream.<init>(FileInputStream.java:101)
        at exhand.ThrowsUncheckPropagation.method3(ThrowsUncheckPropagation.java:32)
        at exhand.ThrowsUncheckPropagation.method2(ThrowsUncheckPropagation.java:28)
        at exhand.ThrowsUncheckPropagation.method1(ThrowsUncheckPropagation.java:19)
        at exhand.ThrowsUncheckPropagation.main(ThrowsUncheckPropagation.java:11)
Write your logic here...
```

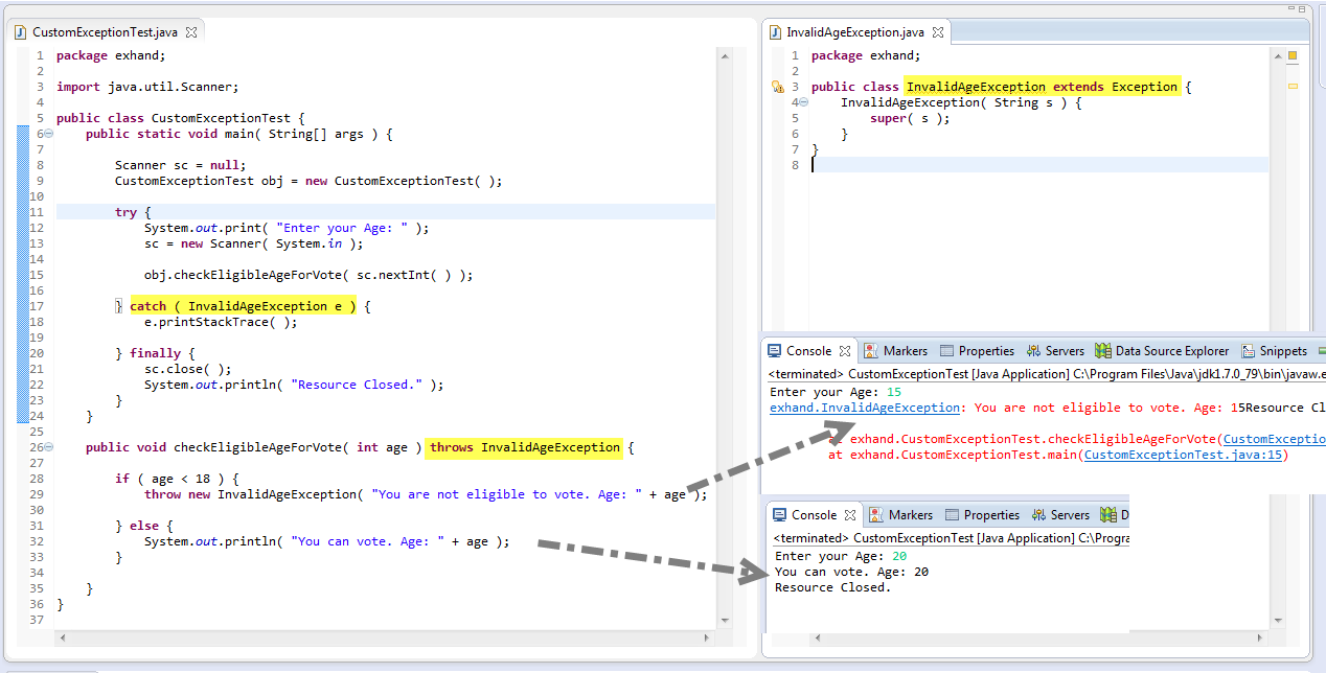
***Rule: If you are calling a method that declares an exception, you must either 1.*** **You caught the exception i.e. handle the exception using try/catch. 2. You declare the exception i.e. specifying throws with the method.**

## Que) Can we rethrow an exception?

Yes, by throwing same exception in catch block.

# Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.



# Java Inner Class

**Java inner class** or nested class is a class i.e. declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

### *Syntax of Inner class*

```
1. class Java_Outer_class{
2.  //code
3.  class Java_Inner_class{
4.   //code
5.  }
6. }
```

# Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.

| | |
|---|---|
| Member Inner Class | A class created within class and outside method. |
| Anonymous Inner Class | A class created for implementing interface or extending class. Its name is decided by the java compiler. |
| Static Nested Class | A static class created within class. |

# Java Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

**Inner class can access all the data members of Outer class including private.**

EmployeeOuter.java

```java
package innerclass;

public class EmployeeOuter {
    private int     id;
    private String  name;

    public EmployeeOuter( int id, String name ) {
        this.id = id;
        this.name = name;
    }

    public static void main( String[] args ) {
        EmployeeOuter employee = new EmployeeOuter( 100, "Bill Gate" );

        AddressInner address = employee.new AddressInner( "New York City", "New York", "00501", "USA" );
        address.showProperties( );
    }

    private class AddressInner {
        private String  city;
        private String  state;
        private String  zip;
        private String  country;

        public AddressInner( String city, String state, String zip, String country ) {
            this.city = city;
            this.state = state;
            this.zip = zip;
            this.country = country;
        }

        private void showProperties( ) {
            // Inner class can access outer class all data members including private.
            System.out.println( id );
            System.out.println( name );

            System.out.println( city );
            System.out.println( state );
            System.out.println( zip );
            System.out.println( country );
        }
    }
}
```

Co... Mar... Pro

```
<terminated> EmployeeOuter [Ja
100
Bill Gate
New York City
New York
00501
USA
```

# Java static nested class

```
StaticInnerO.java
1  package innerclass;
2
3  public class StaticInnerO {
4      static int  count   = 100;
5
6      public static void main( String[] args ) {
7          StaticInnerO.StaticInnerI inn = new StaticInnerO.StaticInnerI( );
8          inn.increaseAndDisplay( );
9          inn.increaseAndDisplay( );
10     }
11
12     private static class StaticInnerI {
13         public void increaseAndDisplay( ) {
14             System.out.println( ++count );
15         }
16     }
17 }
18
```

```
Console
<terminated> StaticI
101
102
```

# Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

## Java anonymous inner class example using class

```
abstract class Person{
  abstract void eat();
}
class TestAnonymousInner{
 public static void main(String args[]){
  Person p=new Person(){
  void eat(){System.out.println("nice fruits");}
  };
  p.eat();
 }
}
```

Test it Now

Output:

```
nice fruits
```

How it can be used in Application: Complex Example

ButtonClick.java

```java
package innerclass;

public class ButtonClick {

    public void addActionEvent( ActionEvent e ) {
        e.actionPerformed( );
    }

    public static void main( String[] args ) {
        button1Clicked( );
        button2Clicked( );
    }

    public static void button1Clicked( ) {
        // No1:way
        ActionEvent aEvent = new ActionEvent( ) {
            @Override
            public void actionPerformed( ) {
                System.out.println( "HELLO From actionPerformed: 1" );
            }
        };
        ButtonClick button2 = new ButtonClick( );
        button2.addActionEvent( aEvent );

    }

    public static void button2Clicked( ) {
        // NO2:way
        ButtonClick button = new ButtonClick( );
        button.addActionEvent( new ActionEvent( ) {
            @Override
            public void actionPerformed( ) {
                System.out.println( "HELLO From actionPerformed: 2" );
            }
        } );
    }
}
```

ActionEvent.java

```java
package innerclass;

public abstract class ActionEvent {
    public abstract void actionPerformed( );
}
```

Console

```
<terminated> ButtonClick [Java Application] C:\Program Files\Java\jdk1
HELLO From actionPerformed: 1
HELLO From actionPerformed: 2
```

# Java anonymous inner class example using interface

```java
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
 public static void main(String args[]){
 Eatable e=new Eatable(){
  public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
}
}
```

**Test it Now**

Output:

```
nice fruits
```

How it can be used in Application: Complex Example

ButtonClickActionListener.java

```java
package innerclass;

public class ButtonClickActionListener {
    public void addActionListener( ActionListener e ) {
        e.actionPerformed( );
    }

    public static void main( String[] args ) {
        button1Clicked( );
        button2Clicked( );
    }

    public static void button1Clicked( ) {
        // No1:way
        ActionListener aEvent = new ActionListener( ) {
            @Override
            public void actionPerformed( ) {
                System.out.println( "HELLO From actionPerformed: 1" );
            }
        };
        ButtonClickActionListener button2 = new ButtonClickActionListener( );
        button2.addActionListener( aEvent );

    }

    public static void button2Clicked( ) {
        // NO2:way
        ButtonClickActionListener button = new ButtonClickActionListener( );
        button.addActionListener( new ActionListener( ) {
            @Override
            public void actionPerformed( ) {
                System.out.println( "HELLO From actionPerformed: 2" );
            }
        } );
    }
}
```

ActionListener.java

```java
package innerclass;

public interface ActionListener {
    void actionPerformed( );
}
```

Console

```
<terminated> ButtonClickActionListener [Java Application] C:\Progra
HELLO From actionPerformed: 1
HELLO From actionPerformed: 2
```

# Java Date

DateTest.java

```java
package datereflection;

import java.util.Calendar;
import java.util.Date;

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

        Date date1 = new Date( );
        System.out.println( date1 );

        long millis = System.currentTimeMillis( );
        Date date2 = new Date( millis );
        System.out.println( date2 );

        Date date3 = Calendar.getInstance( ).getTime( );
        System.out.println( date3 );

    }
}
```

1

2

3

# Java Date Format

DateFormatTest.java

```java
package datereflection;

import java.text.DateFormat;
import java.util.Date;

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

        Date currentDate = new Date( );
        System.out.println( "Date: " + currentDate );

        String dateToStr = DateFormat.getInstance( ).format( currentDate );
        System.out.println( "Date: getInstance(): " + dateToStr );

        dateToStr = DateFormat.getDateInstance( ).format( currentDate );
        System.out.println( "Date: getDateInstance(): " + dateToStr );

        dateToStr = DateFormat.getTimeInstance( ).format( currentDate );
        System.out.println( "Date: getTimeInstance(): " + dateToStr );

        dateToStr = DateFormat.getDateTimeInstance( ).format( currentDate );
        System.out.println( "Date: getDateTimeInstance(): " + dateToStr );

    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Er

```
<terminated> DateFormatTest [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 17, 2015, 2:04:59
Date: Thu Sep 17 14:04:59 NPT 2015
Date: getInstance(): 9/17/15 2:04 PM
Date: getDateInstance(): Sep 17, 2015
Date: getTimeInstance(): 2:04:59 PM
Date: getDateTimeInstance(): Sep 17, 2015 2:04:59 PM
```

# Simple Date Format:

SimpleDateFormatTest.java

```java
package datereflection;

import java.text.SimpleDateFormat;
import java.util.Date;

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

        Date date = new Date( );

        SimpleDateFormat formatter = new SimpleDateFormat( "MM/dd/yyyy" );
        String strDate = formatter.format( date );
        System.out.println( "DateFormat: MM/dd/yyyy : " + strDate );

        formatter = new SimpleDateFormat( "dd-M-yyyy hh:mm:ss" );
        strDate = formatter.format( date );
        System.out.println( "DateFormat: dd-M-yyyy hh:mm:ss : " + strDate );

        formatter = new SimpleDateFormat( "dd MMMM yyyy" );
        strDate = formatter.format( date );
        System.out.println( "DateFormat: dd MMMM yyyy : " + strDate );
    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Lo

```
<terminated> SimpleDateFormatTest [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 17, 2015, 2:08:48
DateFormat: MM/dd/yyyy : 09/17/2015
DateFormat: dd-M-yyyy hh:mm:ss : 17-9-2015 02:08:48
DateFormat: dd MMMM yyyy : 17 September 2015
```

## Java SimpleDateFormat Example: String to Date (**Joda-time**)

DateStrToDateObject.java

```java
package datereflection;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

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

        try {

            SimpleDateFormat formatter = new SimpleDateFormat( "dd/MM/yyyy" );
            Date date = formatter.parse( "31/03/2015" );

            System.out.println( "Date is: " + date );

        } catch ( ParseException e ) {
            e.printStackTrace( );
        }
    }
}
```

**Joda Time Library:**

**Joda-Time** provides a quality replacement for the Java date and time classes.

Joda-Time is the *de facto* standard date and time library for Java.

# Features

A selection of key features:

- `LocalDate` - date without time
- `LocalTime` - time without date
- `Instant` - an instantaneous point on the time-line
- `DateTime` - full date and time with time-zone
- `DateTimeZone` - a better time-zone
- `Duration` and `Period` - amounts of time
- `Interval` - the time between two instants
- A comprehensive and flexible formatter-parser

# Get Date:

JodaTimeExample.java

```
1  package datereflection;
2
3  import java.util.Date;
4
5  import org.joda.time.DateTime;
6
7  public class JodaTimeExample {
8      public static void main( String[] args ) {
9
10         DateTime dateTime = new DateTime( );
11
12         Date date = dateTime.toDate( );
13
14         System.out.println( date );
15     }
16 }
17
```

Console | Markers | Properties | Servers | Data Source Explorer

```
<terminated> JodaTimeExample [Java Application] C:\Program Files\Java\jdk1.7.0_79
Thu Sep 17 15:55:36 NPT 2015
```

# Date example using Joda Time:

JodaTimeExamples.java

```java
package datereflection;

import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;

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

        DateTime dt = new DateTime( ); // Joda Date
        System.out.println( "Date:" + dt.toDate( ) ); // Java Date

        int month = dt.getMonthOfYear( );
        System.out.println( "MonthOfYear: " + month );

        DateTime.Property pDoW = dt.dayOfWeek( );// Monday:1 to Sunday:7
        System.out.println( "dayOfWeek: " + pDoW.getAsText( ) ); // print:Monday/Tuesday

        System.out.println( "getDayOfMonth: " + dt.getDayOfMonth( ) );
        int maxDay = dt.dayOfMonth( ).getMaximumValue( );
        System.out.println( "Last day of this month: " + maxDay + " day" );

        boolean leapYear = dt.yearOfEra( ).isLeap( );
        System.out.println( "Leap Year: " + leapYear );

        DateTime datePlus20 = dt.plusDays( 20 );
        DateTimeFormatter formattedDate = DateTimeFormat.forPattern( "dd/MM/yyyy" );
        System.out.println( dt.toString( formattedDate ) + " + 20 day = " + datePlus20.toString( formattedDate ) );

    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress | Error Log

```
<terminated> JodaTimeExamples [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 17, 2015, 4:19:38 PM)
Date:Thu Sep 17 16:19:38 NPT 2015
MonthOfYear: 9
dayOfWeek: Thursday
getDayOfMonth: 17
Last day of this month: 30day
Leap Yearfalse
17/09/2015 + 20 day = 07/10/2015
```

# Java Reflection API

**Java Reflection** is a *process of examining or modifying the run time behavior of a class at run time*.

The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The java.lang and java.lang.reflect packages provide classes for java reflection.

### Where it is used

The Reflection API is mainly used in:

- IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
- Debugger
- Test Tools etc.

## java.lang.Class class

The java.lang.Class class performs mainly two tasks:

- provides methods to get the metadata of a class at run time.
- provides methods to examine and change the run time behavior of a class.

## Commonly used methods of Class class:

| Method | Description |
|---|---|
| 1) public String getName() | returns the class name |
| 2) public static Class forName(String className)throws ClassNotFoundException | loads the class and returns the reference of Class class. |
| 3) public Object newInstance()throws InstantiationException,IllegalAccessException | creates new instance. |
| 4) public boolean isInterface() | checks if it is interface. |
| 5) public boolean isArray() | checks if it is array. |

| | |
|---|---|
| 6) public boolean isPrimitive() | checks if it is primitive. |
| 7) public Class getSuperclass() | returns the superclass class reference. |
| 8) public Field[] getDeclaredFields()throws SecurityException | returns the total number of fields of this class. |
| 9) public Method[] getDeclaredMethods()throws SecurityException | returns the total number of methods of this class. |
| 10) public Constructor[] getDeclaredConstructors()throws SecurityException | returns the total number of constructors of this class. |
| 11) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException | returns the method class instance. |

## How to get the object of Class class?

There are 3 ways to get the instance of Class class. They are as follows:

- forName() method of Class class
- getClass() method of Object class
- the .class syntax

ReflectionClassName.java

```java
package datereflection;

public class ReflectionClassName {
    public static void main( String[] args ) throws ClassNotFoundException {

        Class c = Class.forName( "java.util.Date" );
        System.out.println( c.getName( ) );

        Integer intObj = new Integer( "25" );
        Class cInt = intObj.getClass( );
        System.out.println( cInt.getName( ) );

        Class c3 = String.class;
        System.out.println( c3.getName( ) );

    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | Progress

```
<terminated> ReflectionClassName [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 17, 2015
java.util.Date
java.lang.Integer
java.lang.String
```

# Java instanceof operator:

InstanceOfUses.java

```java
package datereflection;

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

        displayObjectType( new Integer( "25" ) );

        displayObjectType( "25" );
    }

    public static void displayObjectType( Object o ) {

        if ( o instanceof Integer ) {
            System.out.println( "Parameter is Integer: " + o );

        } else if ( o instanceof String ) {
            System.out.println( "Parameter is String: " + o );
        }
    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer | Snippets | P

```
<terminated> InstanceOfUses [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 17
Parameter is Integer: 25
Parameter is String: 25
```