Java Generics

Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

Generic Methods

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods −

- All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type (< E > in the next example).
- Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
- The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
- A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

Example

Following example illustrates how we can print an array of different type using a single Generic method:

```
public class GenericMethodTest {
   // generic method printArray
   public static < E > void printArray( E[] inputArray ) {
      // Display array elements
      for(E element : inputArray) {
         System.out.printf("%s ", element);
      }
      System.out.println();
   }

   public static void main(String args[]) {
      // Create arrays of Integer, Double and Character
      Integer[] intArray = { 1, 2, 3, 4, 5 };
      Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
      Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

      System.out.println("Array integerArray contains:");
      printArray(intArray);   // pass an Integer array

      System.out.println("\nArray doubleArray contains:");
      printArray(doubleArray);   // pass a Double array

      System.out.println("\nArray characterArray contains:");
      printArray(charArray);   // pass a Character array
   }
}
```

Output

```
Array integerArray contains:

1 2 3 4 5

Array doubleArray contains:

1.1 2.2 3.3 4.4

Array characterArray contains:

H E L L O
```

Generic Classes

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Example

Following example illustrates how we can define a generic class

```
public class Box<T> {
   private T t;

   public void add(T t) {
      this.t = t;
   }

   public T get() {
      return t;
   }

   public static void main(String[] args) {
      Box<Integer> integerBox = new Box<Integer>();
      Box<String> stringBox = new Box<String>();

      integerBox.add(new Integer(10));
      stringBox.add(new String("Hello World"));

      System.out.printf("Integer Value :%d\n\n", integerBox.get());
      System.out.printf("String Value :%s\n", stringBox.get());
   }
}
```

Output

Integer Value: 10

String Value: Hello World

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.

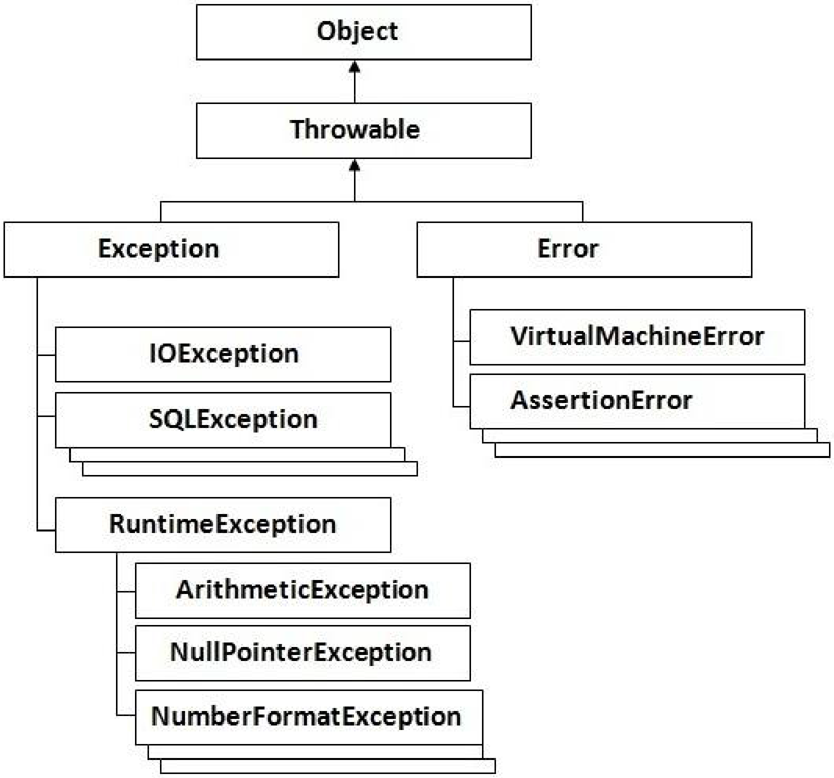
In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

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

Common scenarios where exceptions may occur

- int a=50/0;//ArithmeticException
- String s=null; System.out.println(s.length());//NullPointerException
- String s="java"; int i=Integer.parseInt(s);//NumberFormatException
- int a[]=new int[5]; a[10]=50; //ArrayIndexOutOfBoundsException

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 (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. ArrayIndexOutOfBoundsException, ArithmeticException, NullPointerException, 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 {

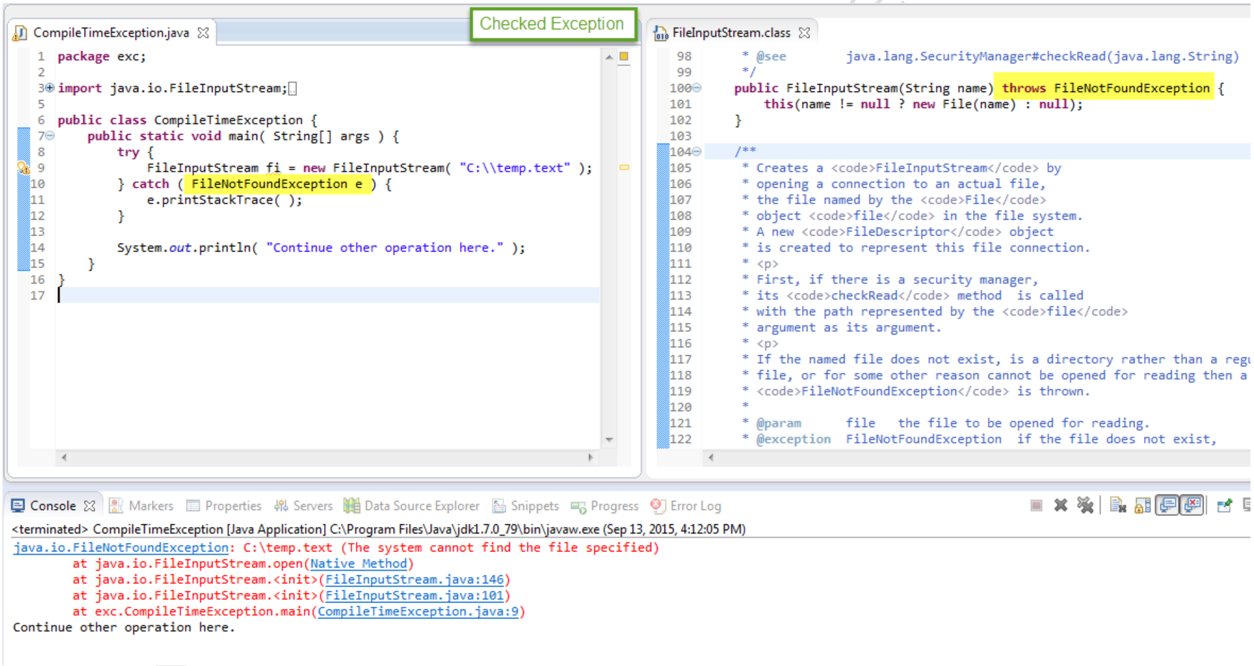
//code that may throw exception

} 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.

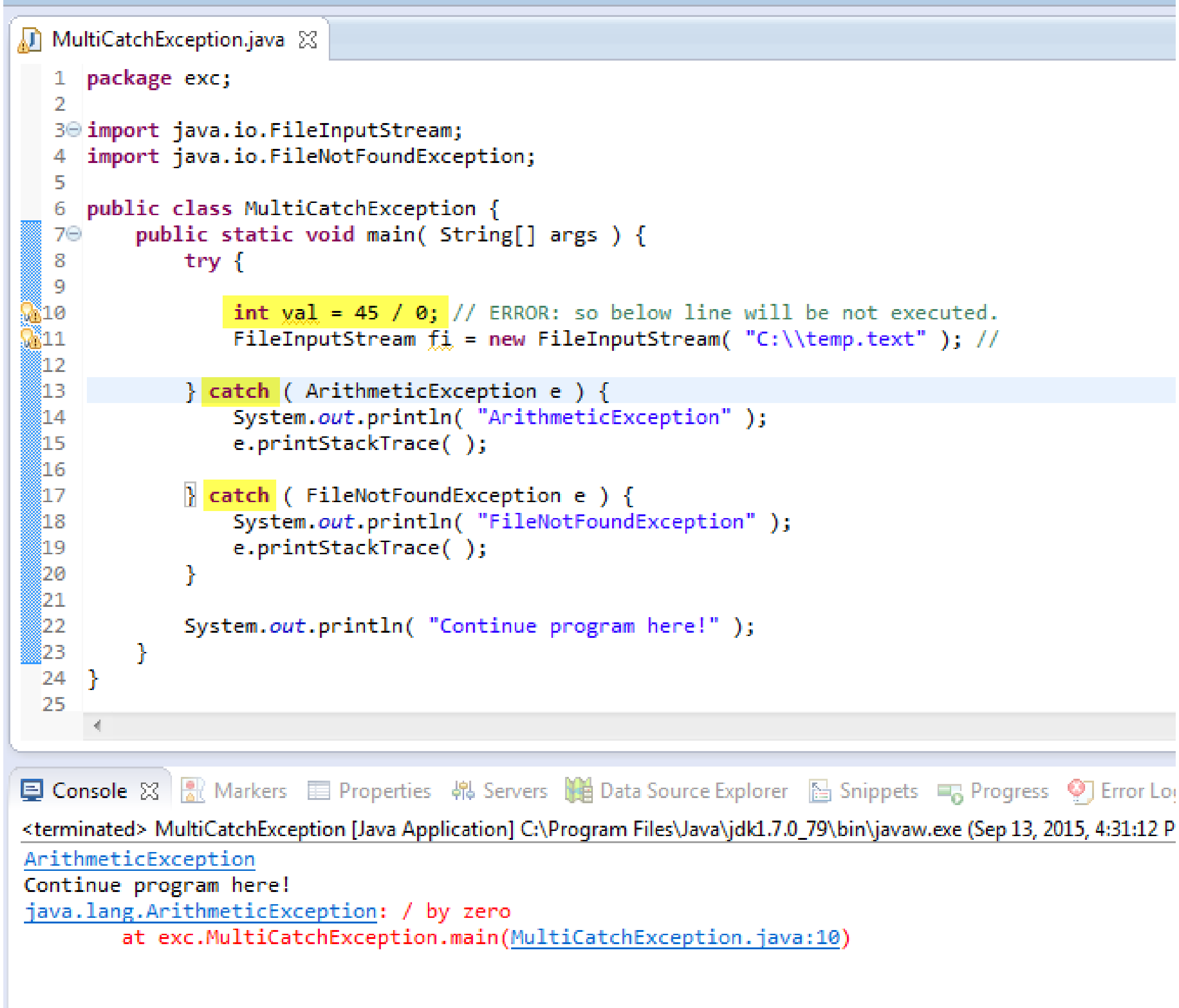


```java
package exc;

import java.io.FileInputStream;

public class CompileTimeException {
    public static void main( String[] args ) {
        try {
            FileInputStream fi = new FileInputStream( "C:\\temp.text" );
        } catch ( FileNotFoundException e ) {
            e.printStackTrace( );
        }

        System.out.println( "Continue other operation here." );
    }
}
```

```java
     * @see        java.lang.SecurityManager#checkRead(java.lang.String)
     */
    public FileInputStream(String name) throws FileNotFoundException {
        this(name != null ? new File(name) : null);
    }

    /**
     * Creates a <code>FileInputStream</code> by
     * opening a connection to an actual file,
     * the file named by the <code>File</code>
     * object <code>file</code> in the file system.
     * A new <code>FileDescriptor</code> object
     * is created to represent this file connection.
     * <p>
     * First, if there is a security manager,
     * its <code>checkRead</code> method  is called
     * with the path represented by the <code>file</code>
     * argument as its argument.
     * <p>
     * If the named file does not exist, is a directory rather than a regu
     * file, or for some other reason cannot be opened for reading then a
     * <code>FileNotFoundException</code> is thrown.
     *
     * @param      file   the file to be opened for reading.
     * @exception  FileNotFoundException  if the file does not exist,
```

```
<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



```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!" );
    }
}
```

```
<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)
```

```
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...");
    }
}
Output: Compile-time error
```

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.

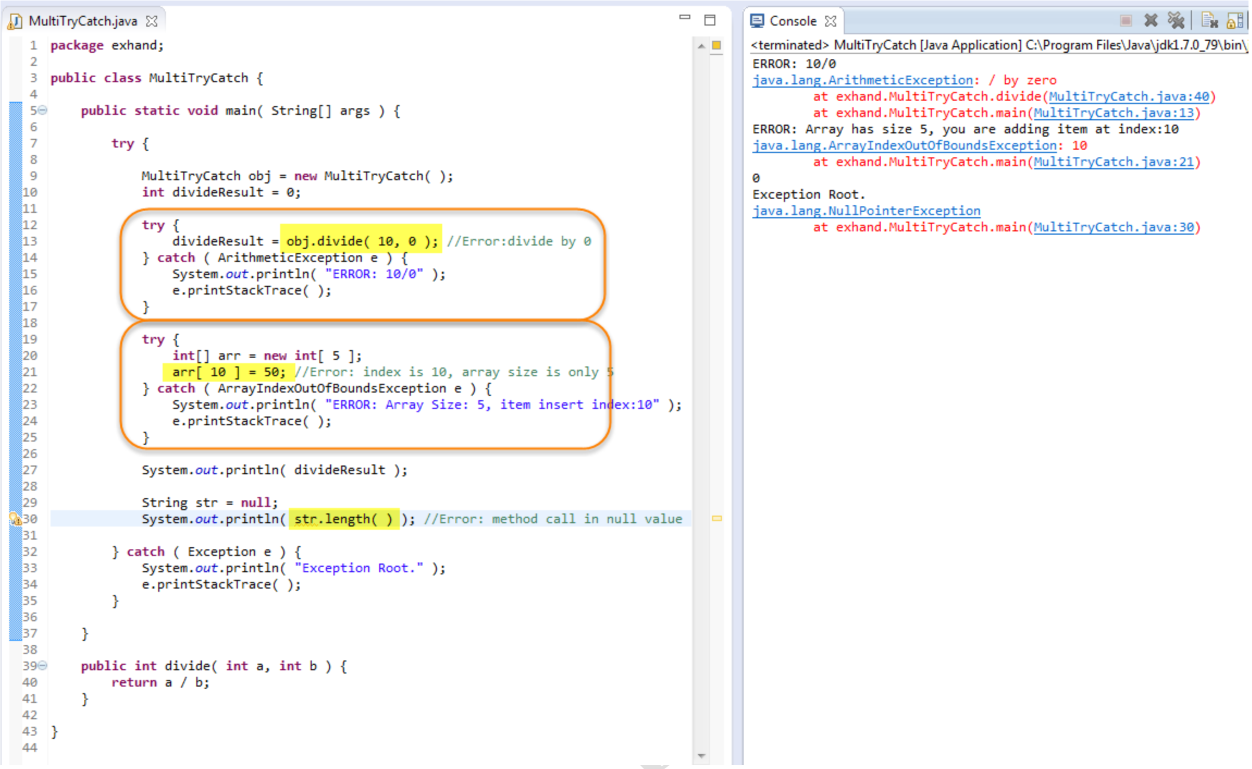
Java Nested try block

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) { }
```

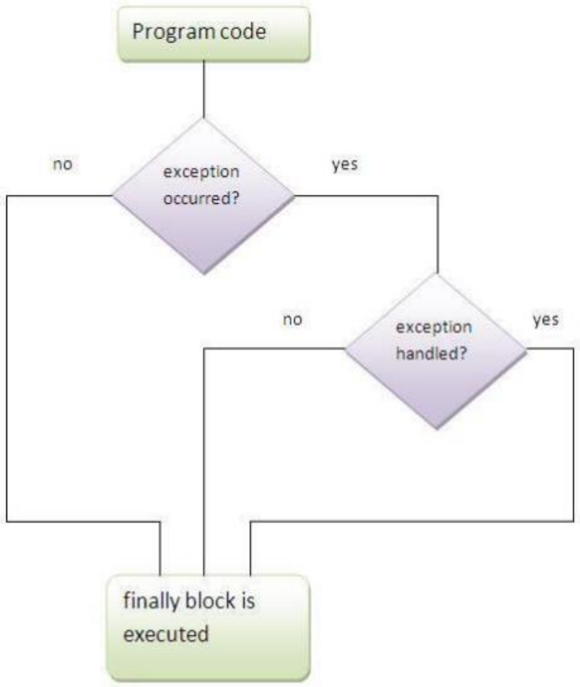
Java nested try example



```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;
    }

}
```

```
<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.

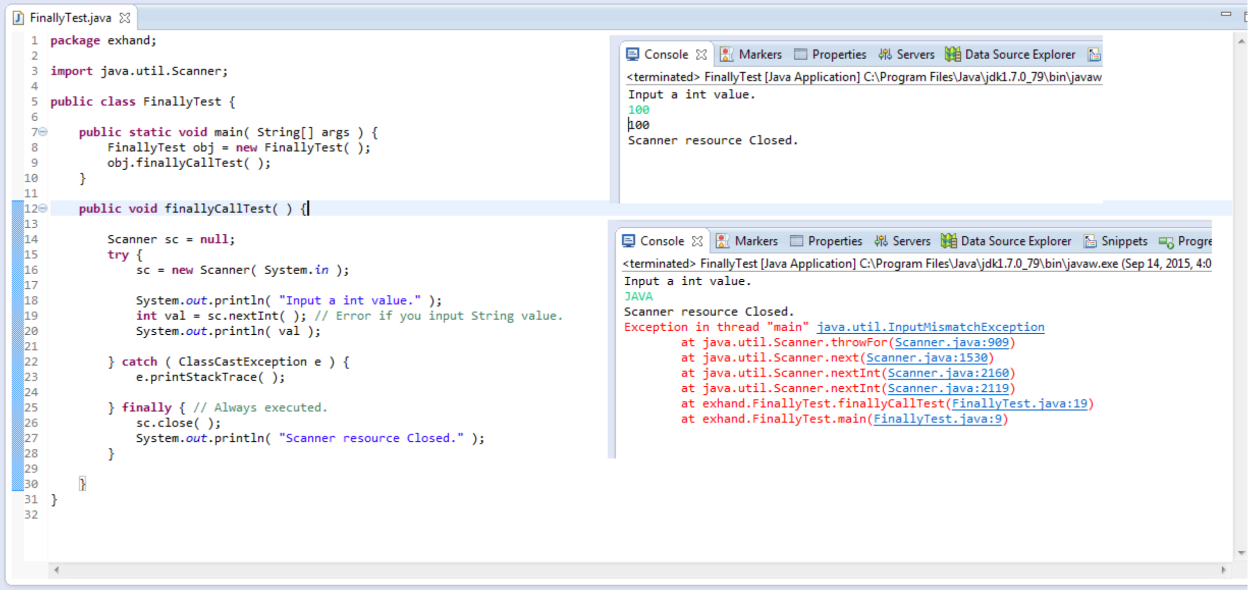
Java finally block is always executed whether exception is handled or not.

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)
```

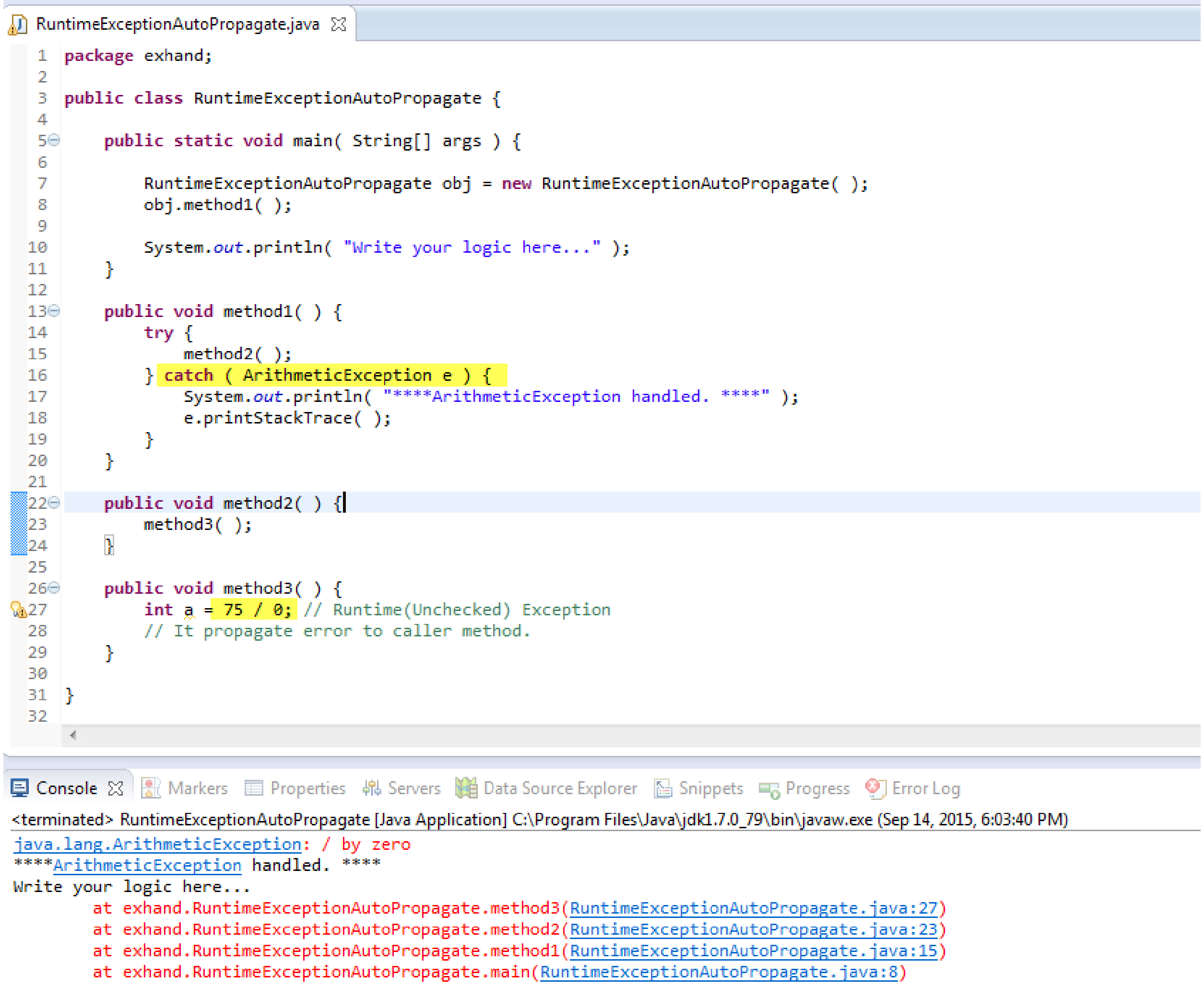
Rule: For each try block there can be zero or more catch blocks, but only one finally block. Note: The finally block will not be executed if program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort).

Java 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.

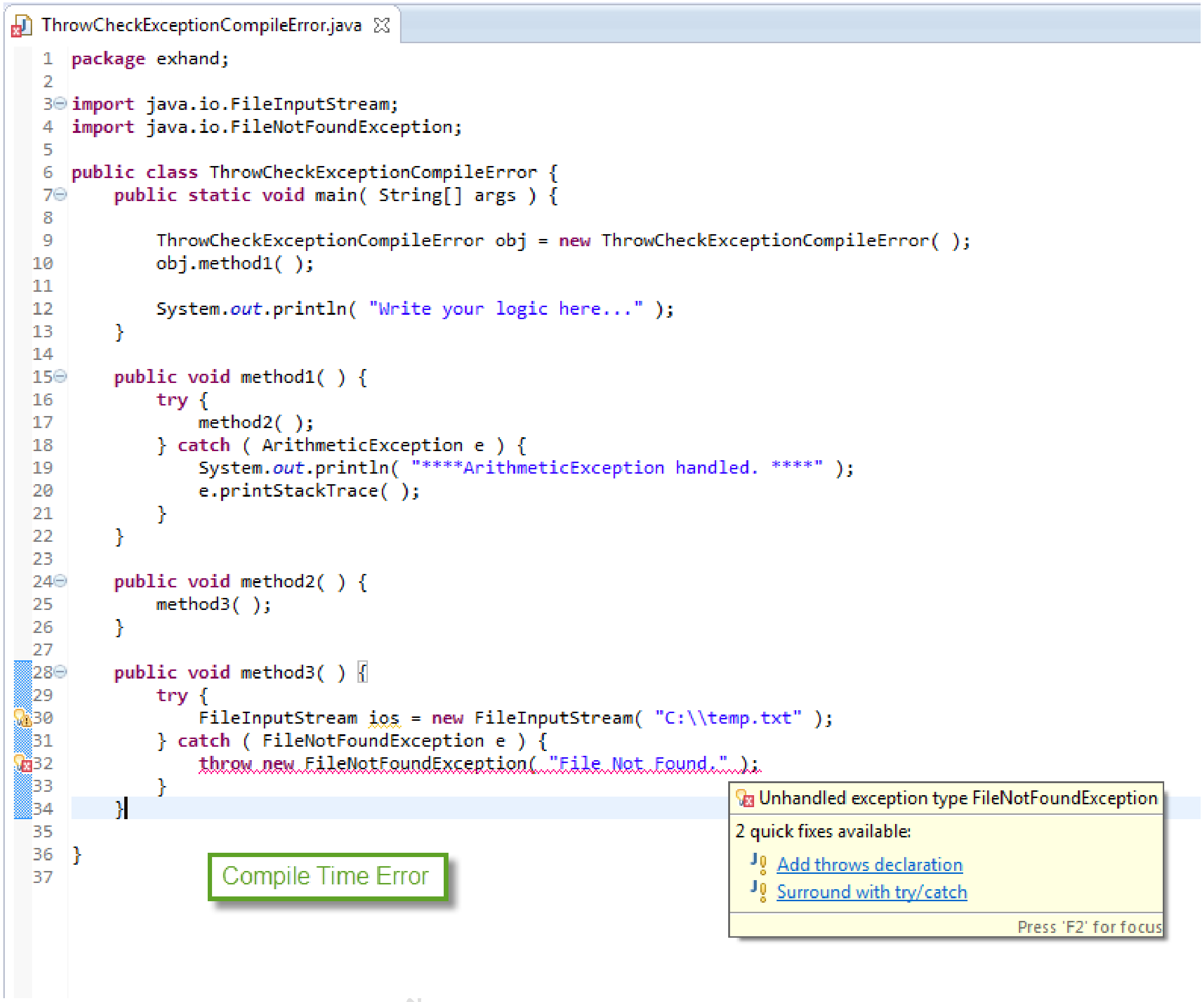
Rule: By default Unchecked Exceptions are forwarded in calling chain (propagated).

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



```java
package exhand;

public class RuntimeExceptionAutoPropagate {

    public static void main( String[] args ) {

        RuntimeExceptionAutoPropagate obj = new RuntimeExceptionAutoPropagate( );
        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( ) {
        int a = 75 / 0; // Runtime(Unchecked) Exception
        // It propagate error to caller method.
    }

}
```

```
<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)
```



```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." );
        }
    }

}
```

Unhandled exception type FileNotFoundException

2 quick fixes available:

Add throws declaration

Surround with try/catch

Press 'F2' for focus

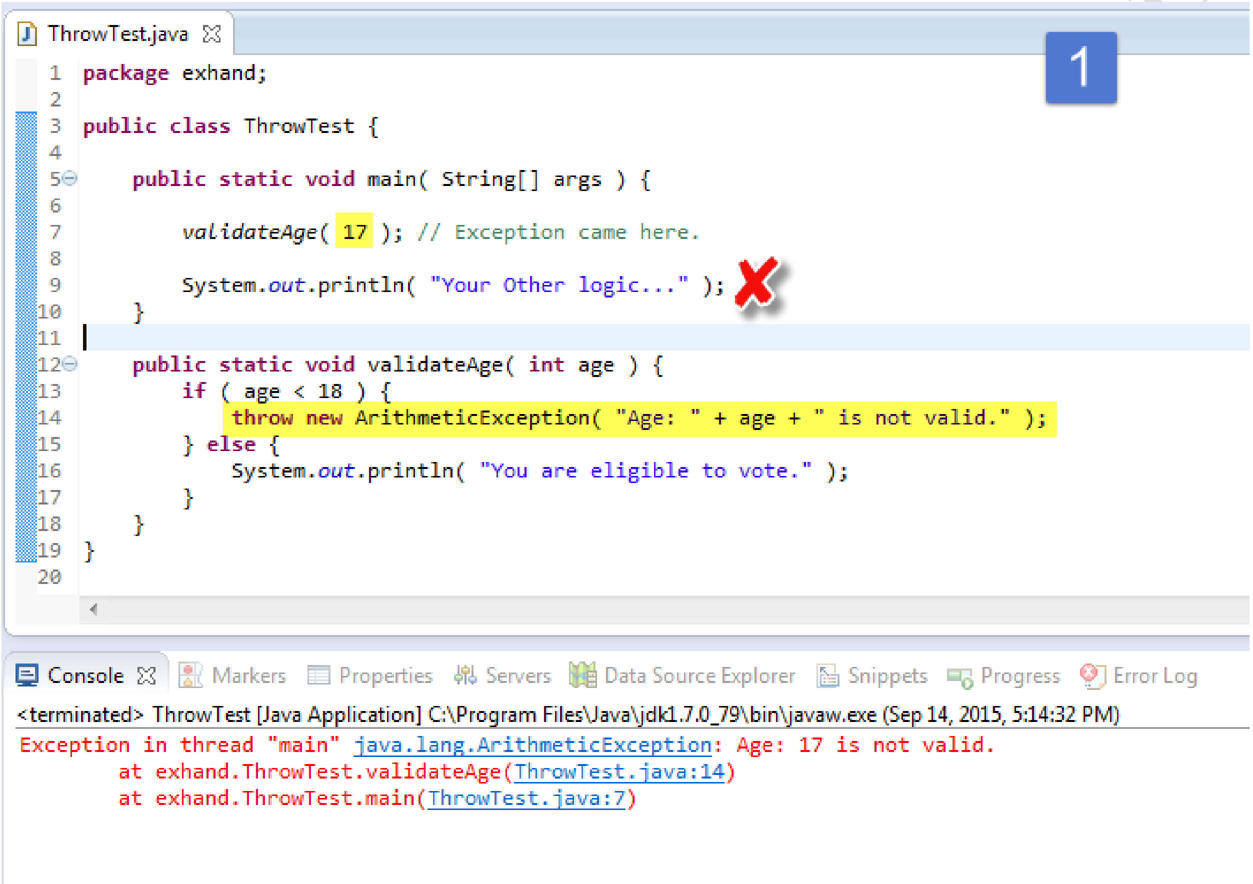
Compile Time Error

Java throw/throws exception

Java throw keyword

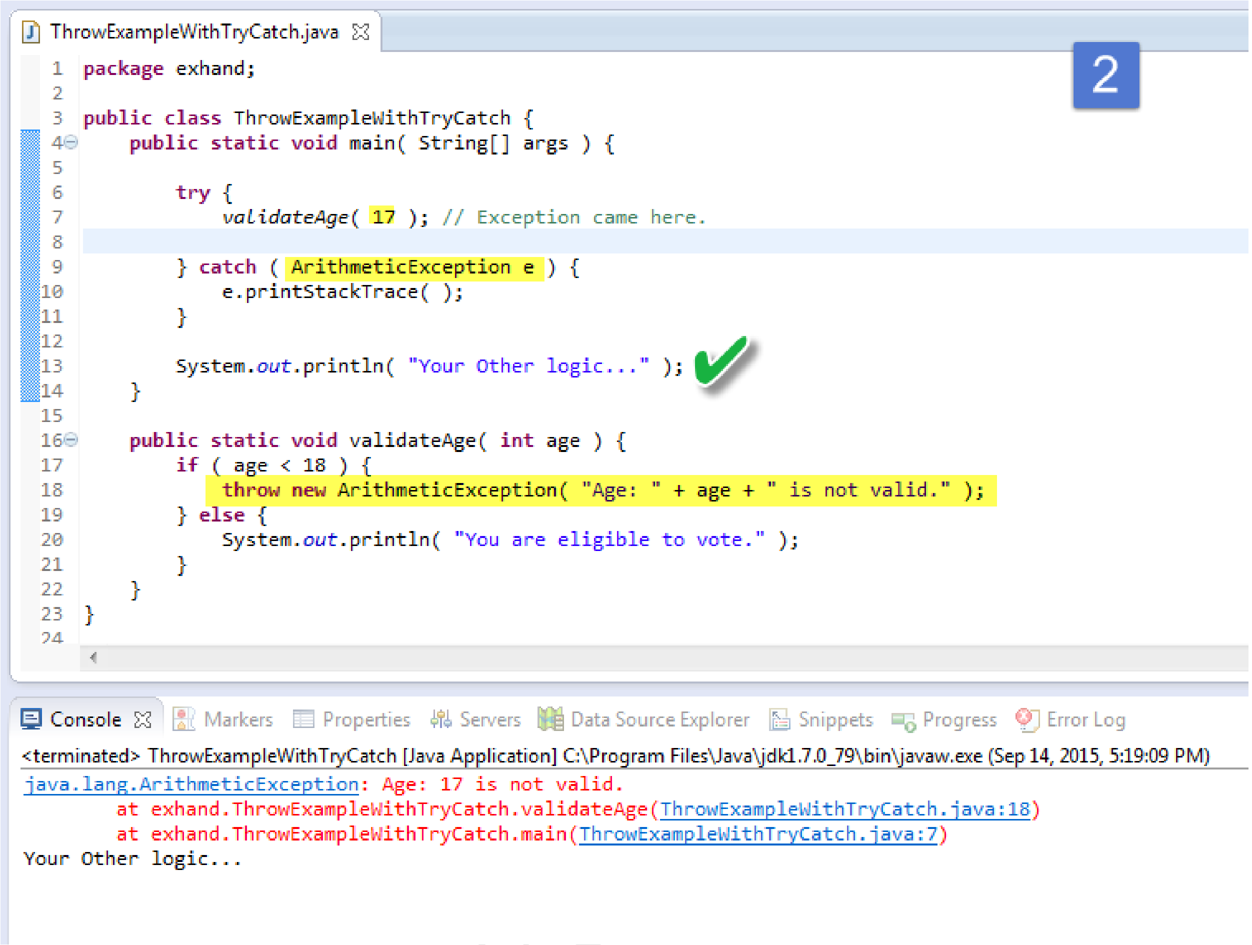
The Java throw keyword is used to **explicitly throw an exception**.

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



```java
package exhand;

public class ThrowTest {

    public static void main( String[] args ) {

        validateAge( 17 ); // Exception came here.

        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." );
        }
    }
}
```

```
<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)
```



```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." );
        }
    }
}
```

```
<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 throws keyword

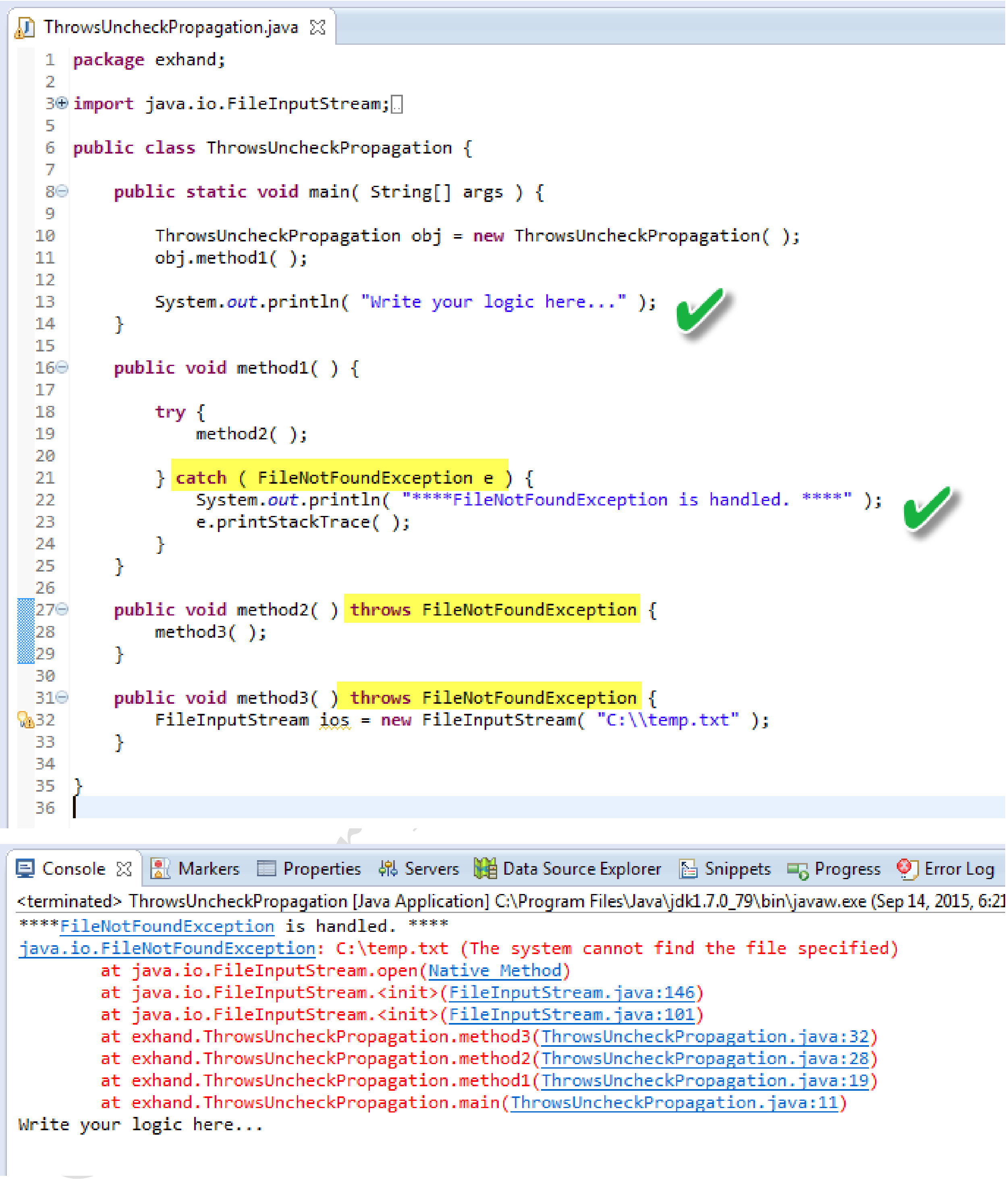
The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

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 checkup 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.



```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" );
    }

}
```

```
<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...
```

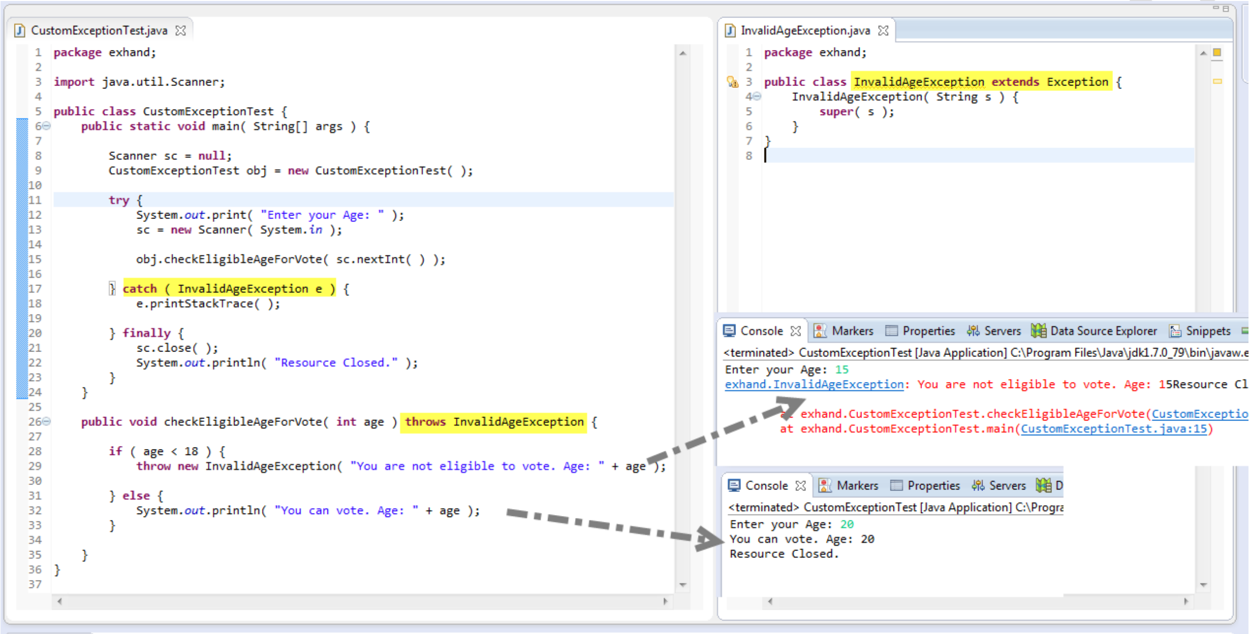
Rule: If you are calling a method that declares an exception, you must either 1. You caught the exception i.e. handle the exception using try/catch. 2. You declare the exception i.e. specifying throws with the method.

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
package exhand;

import java.util.Scanner;

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

        Scanner sc = null;
        CustomExceptionTest obj = new CustomExceptionTest( );

        try {
            System.out.print( "Enter your Age: " );
            sc = new Scanner( System.in );

            obj.checkEligibleAgeForVote( sc.nextInt( ) );

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

        } finally {
            sc.close( );
            System.out.println( "Resource Closed." );
        }
    }

    public void checkEligibleAgeForVote( int age ) throws InvalidAgeException {

        if ( age < 18 ) {
            throw new InvalidAgeException( "You are not eligible to vote. Age: " + age );

        } else {
            System.out.println( "You can vote. Age: " + age );
        }

    }
}
```

```java
package exhand;

public class InvalidAgeException extends Exception {
    InvalidAgeException( String s ) {
        super( s );
    }
}
```

```
<terminated> CustomExceptionTest [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.e
Enter your Age: 15
exhand.InvalidAgeException: You are not eligible to vote. Age: 15Resource Cl

    at exhand.CustomExceptionTest.checkEligibleAgeForVote(CustomExceptio
    at exhand.CustomExceptionTest.main(CustomExceptionTest.java:15)
```

```
<terminated> CustomExceptionTest [Java Application] C:\Progra
Enter your Age: 20
You can vote. Age: 20
Resource Closed.
```