Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II

SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 3

EXCEPTION HANDLING AND I/O

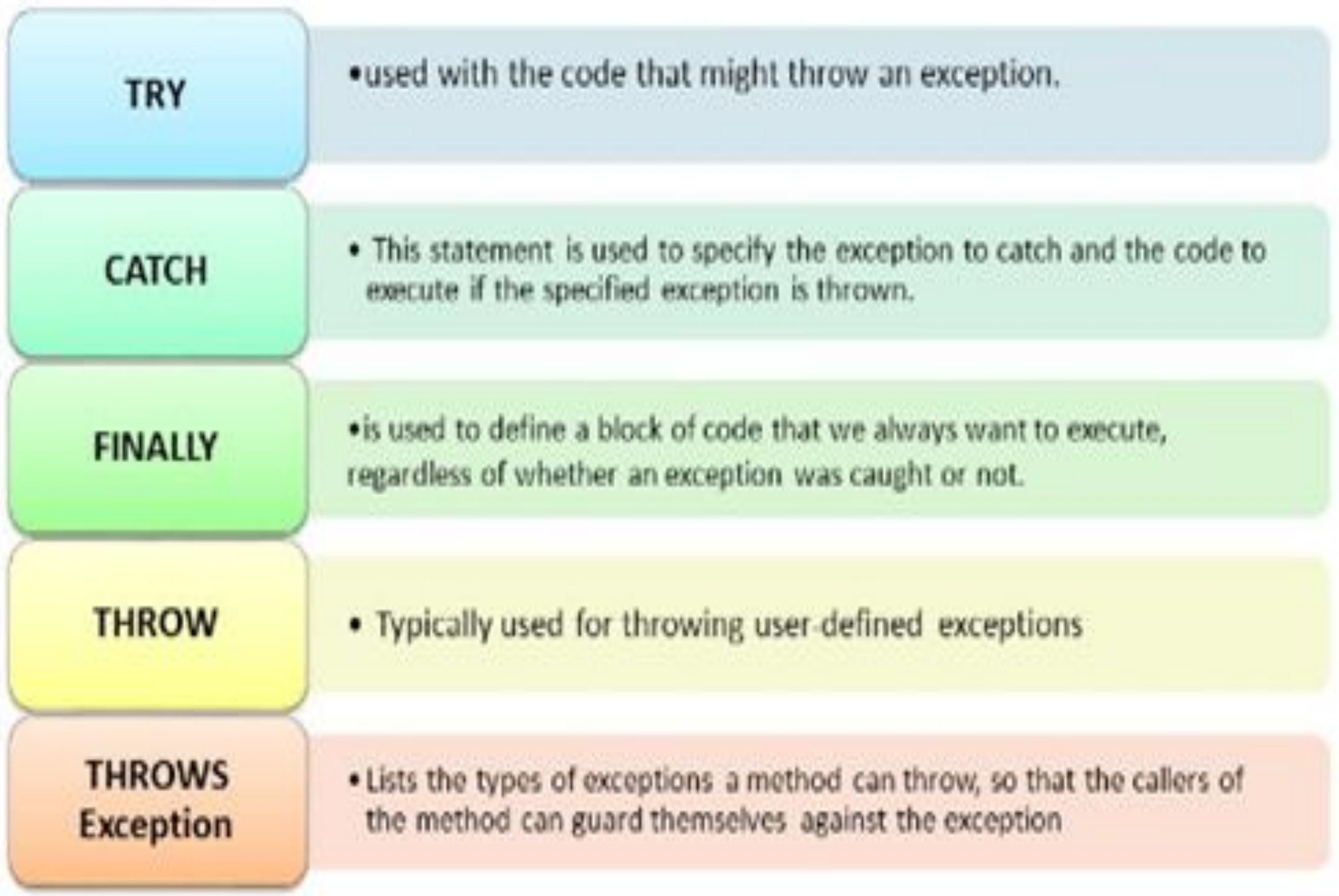
3.2 Throwing and Catching Exceptions

COMPUTER SCIENCE & ENGINEERING

Throwing Exceptions

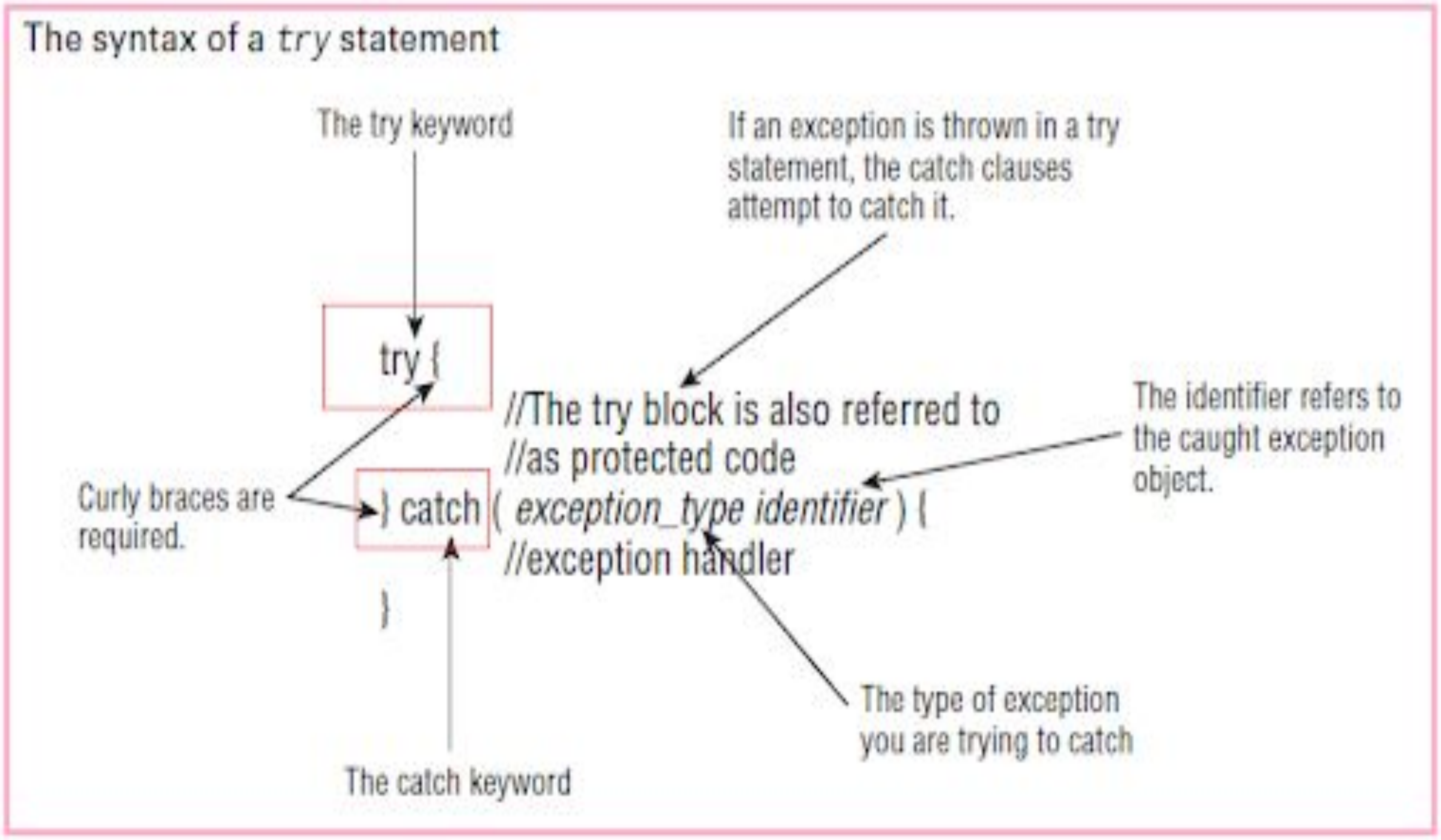
- Before, catch an exception, some code must throw one.
- Any code can throw an exception
- Java platform provides numerous exception classes. All the classes are descendants (family) of the Throwable class, and all allow programs to differentiate among the various types of exceptions that can occur during the execution of a program.
- We can create our own exception classes to represent problems that can occur within the classes we create.

Exceptions in java



TRY	•used with the code that might throw an exception.
CATCH	• This statement is used to specify the exception to catch and the code to execute if the specified exception is thrown.
FINALLY	•is used to define a block of code that we always want to execute, regardless of whether an exception was caught or not.
THROW	• Typically used for throwing user-defined exceptions
THROWS Exception	•Lists the types of exceptions a method can throw, so that the callers of the method can guard themselves against the exception

Syntax


The syntax of a *try* statement

```
try {
    //The try block is also referred to
    //as protected code
} catch ( exception_type identifier ) {
    //exception handler
}
```

The try keyword
If an exception is thrown in a try statement, the catch clauses attempt to catch it.
The identifier refers to the caught exception object.
Curly braces are required.
The type of exception you are trying to catch
The catch keyword

Syntax

try – throw – catch Mechanism

The basic way of handling exceptions in java consists of the try-throw-catch.

The try block contains the code for the basic algorithm that tells what to do when everything goes smoothly. It is called a try block because it "tries" to execute the case where all goes as planned.

It can also contain code that throws an exception if something unusual happens.

```
try
{
Code throws….
}
catch(Exception e)
```

Throwing Exceptions

A catch statement involves declaring type of exception you are trying to catch.

If an exception occurs in protected code, catch block (or blocks) that follows try is checked.

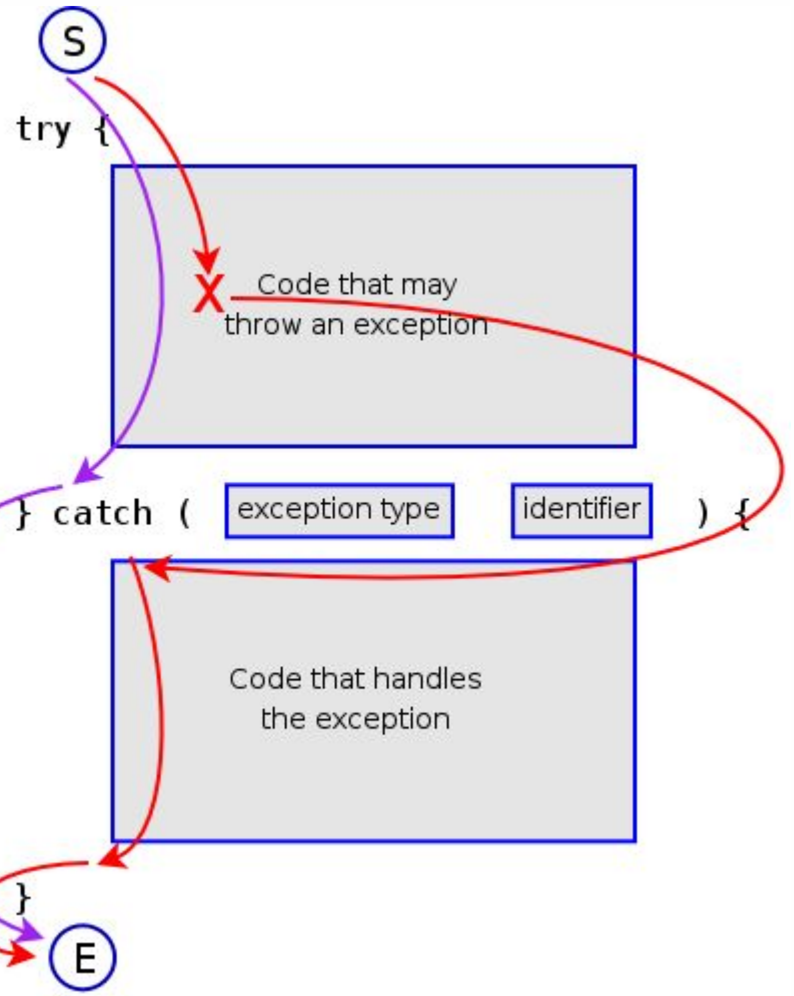
If type of exception that occurred is listed in a catch block, exception is passed to the catch block much as an argument is passed into a method parameter.

The throw statement is used together with an **exception type,** also allows you to create a custom error.

There are many exception types available in Java:

Arithmetic Exception,
FileNotFoundException,
ArrayIndexOutOfBoundsException,
Security Exception,

Flow of Throwing Exceptions


S
try {
X
Code that may throw an exception
} catch (
exception type
identifier
) {
Code that handles the exception
}
E

Example Program

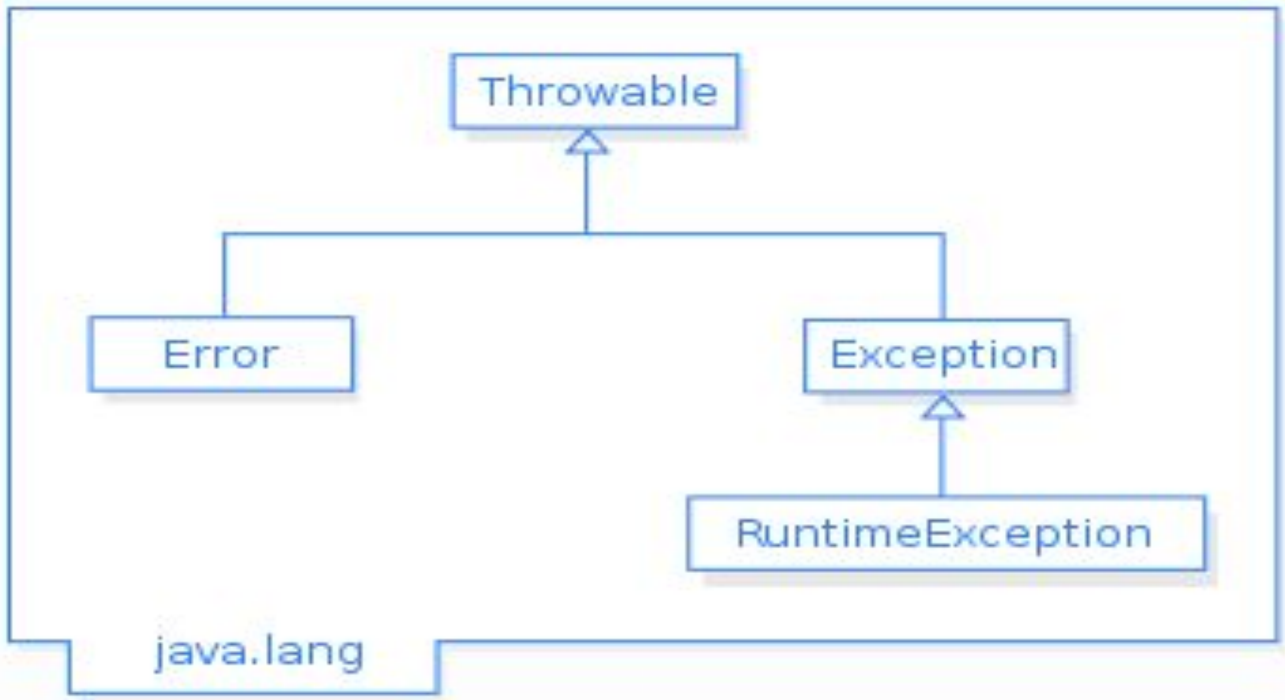
Coding :

```
import java.io.*;
 public class Demo
 {
 public static void main(String args[])
 {
 try              {
                  int a[] = new int[5];
                 System.out.println("Access element eighth :" + a[7]);
         }
 catch (ArrayIndexOutOfBoundsException e)
         {
                 System.out.println("Exception thrown :" + e);
         }
                 System.out.println("Out of the block");
  }
}
```

Output :

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 7
     Out of the block
```

Exceptions classes and their Inheritance



throw ,throws and finally in Java

A method does not handle a checked exception, method must declare it using throws keyword.

throw: Throw keyword is used to transfer control from try block to catch block.

throws: Throws keyword is used for exception handling without try & catch block.
It specifies exceptions that a method can throw to the caller and does not handle itself.

finally: It is executed after catch block. We basically use it to put some common code when there are multiple catch blocks.

throw in Java

throw
The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

Syntax:
throw *Instance*

Example:
throw new ArithmeticException("/ by zero");

But this exception i.e, *Instance* must be of type **Throwable** or a subclass of **Throwable**. For example Exception is a sub-class of Throwable and user defined exceptions typically extend Exception class

Sample Program using throw

```
 public class MyClass
{
static void checkAge(int age)
 {
 if (age < 18)
 {
throw new ArithmeticException("Access denied - You must be at least 18 years old.");
}
else {
 System.out.println("Access granted - You are old enough!");
 }
}
 public static void main(String[] args)
{
 checkAge(15); // Set age to 15 (which is below 18...)
 }
 }
```

continuation of Program using throw

Output:

```
Exception in thread "main" java.lang.ArithmeticException: Access denied – You
must be at least 18 years old.
        at MyClass.checkAge(MyClass.java:4)
        at MyClass.main(MyClass.java:12)
```

Note:
If **age** was 20, you would **not** get an exception:

throws in java

throws
throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

Syntax:

type method_name(parameters) throws exception_list

exception_list ,is a comma separated list of all exceptions which method might throw.

In a program, if there is a chance of rising an exception then compiler always warn us about it and compulsorily we should handle that checked exception.

Otherwise we will get compile time error saying **unreported exception must be caught or declared to be thrown**

throws in java

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

1)By using try catch

2)By using **throws** keyword

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

Sample coding

```
 // Java program to illustrate throws
class test
{
    public static void main(String[] args)throws InterruptedException
    {
        Thread.sleep(10000);
        System.out.println("Hello Geeks");
    }
        }
```

Output:

```
Hello Geeks
```

Program using throws

```
  class ThrowsExecp
 {
          static void fun()  throws IllegalAccessException
    {
        System.out.println("Inside fun(). ");
        throw new IllegalAccessException("demo");
    }
    public static void main(String args[])
    {
          try
        {
           fun();
        }
        catch(IllegalAccessException e)
        {
           System.out.println("caught in main.");
         }               }
          }
         catch(IllegalAccessException e)
          {
                 System.out.println("caught in main.");
                        }
                        }
                        }
```

Program using throws

Important points to remember about throws keyword:

throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.

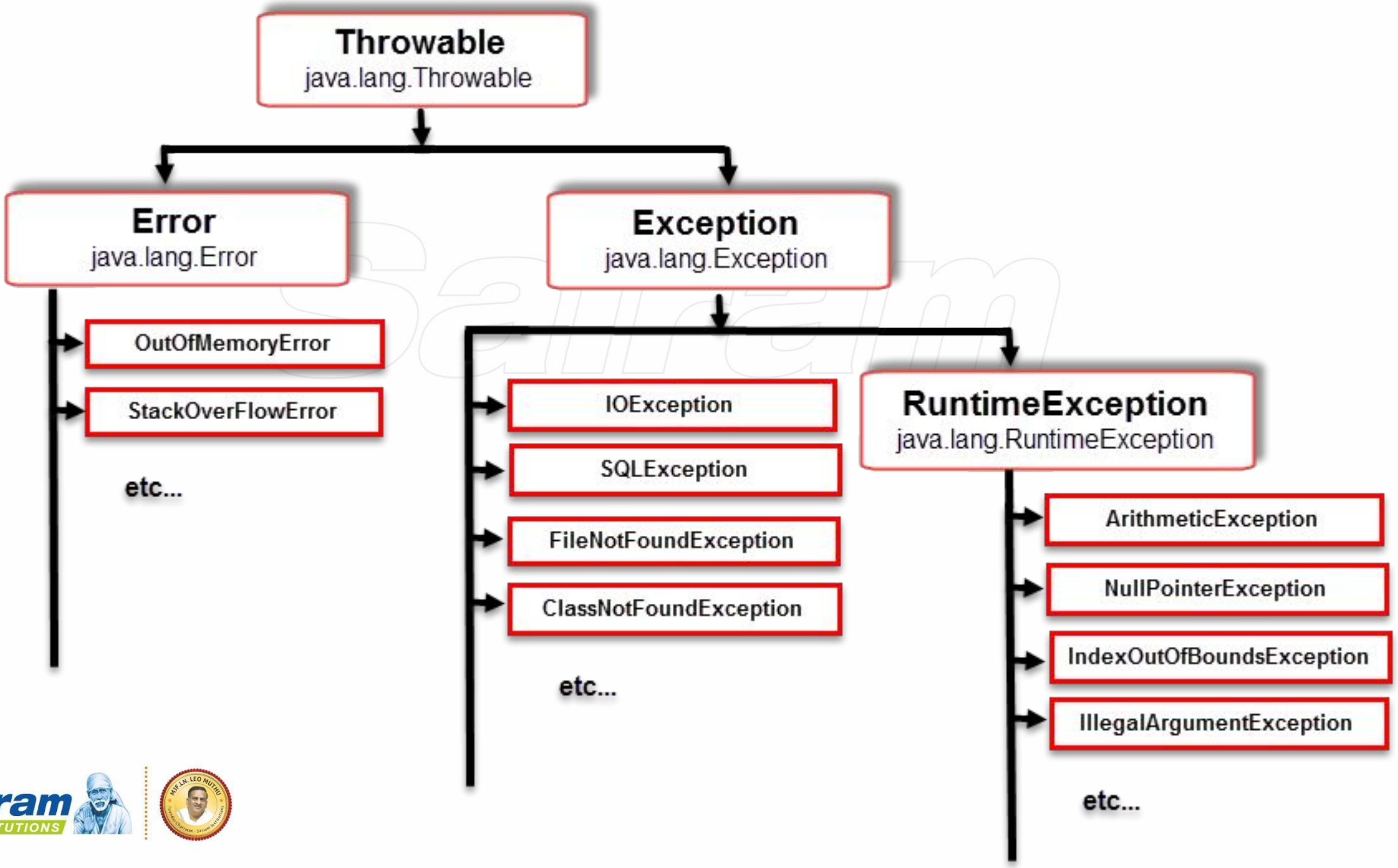
throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.

By the help of throws keyword we can provide information to the caller of the method about the exception.

Difference between using throw , throws

S. No.	Throw	Throws
1	throw keyword is used to throw an exception explicitly.	throws keyword is used to declare one or more exceptions, separated by commas.
2.	Only single exception is thrown by using throw.	Multiple exceptions can be thrown by using throws.
3.	throw keyword is used within the method	throws keyword is used with the method signature.
4.	Syntax wise throw keyword is followed by the instance variable.	Syntax wise throws keyword is followed by exception class names.
5.	Checked exception cannot be propagated using throw only. Unchecked exception can be propagated using throw.	For the propagation checked exception must use throws keyword followed by specific exception class name.

Exception Types



Exception Types

Checked versus 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.

For example, consider the following Java program that opens file at location "C:\test\a.txt" and prints the first three lines of it. The program doesn't compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

Program for Checked Exceptions in Java

```
import java.io.*;
  class Main
{
    public static void main(String[] args) {
        FileReader file = new FileReader("C:\\test\\a.txt");
        BufferedReader fileInput = new BufferedReader(file);
            // Print first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());
          fileInput.close();
    }
}
```

Output:

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - unreported
exception java.io.FileNotFoundException; must be            caught or declared to be thrown at
Main.main(Main.java:5)
```

Program for Checked Exceptions in Java

2) Unchecked

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 specify or catch the exceptions.

In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

Program for UnChecked Exceptions in Java

```
class Main {
   public static void main(String args[]) {
      int x = 0;
      int y = 10;
      int z = y/x;
  }
}
```

Output :

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at Main.main(Main.java:5)
Java Result: 1
```

Sample Program using throw

```
class ThrowsExcep
 {
   // This method throws an exception
   // to be handled
   // by caller or caller
   // of caller and so on.

static void fun() throws IllegalAccessException
   {
       System.out.println("Inside fun(). ");
       throw new IllegalAccessException("demo");
   }
```

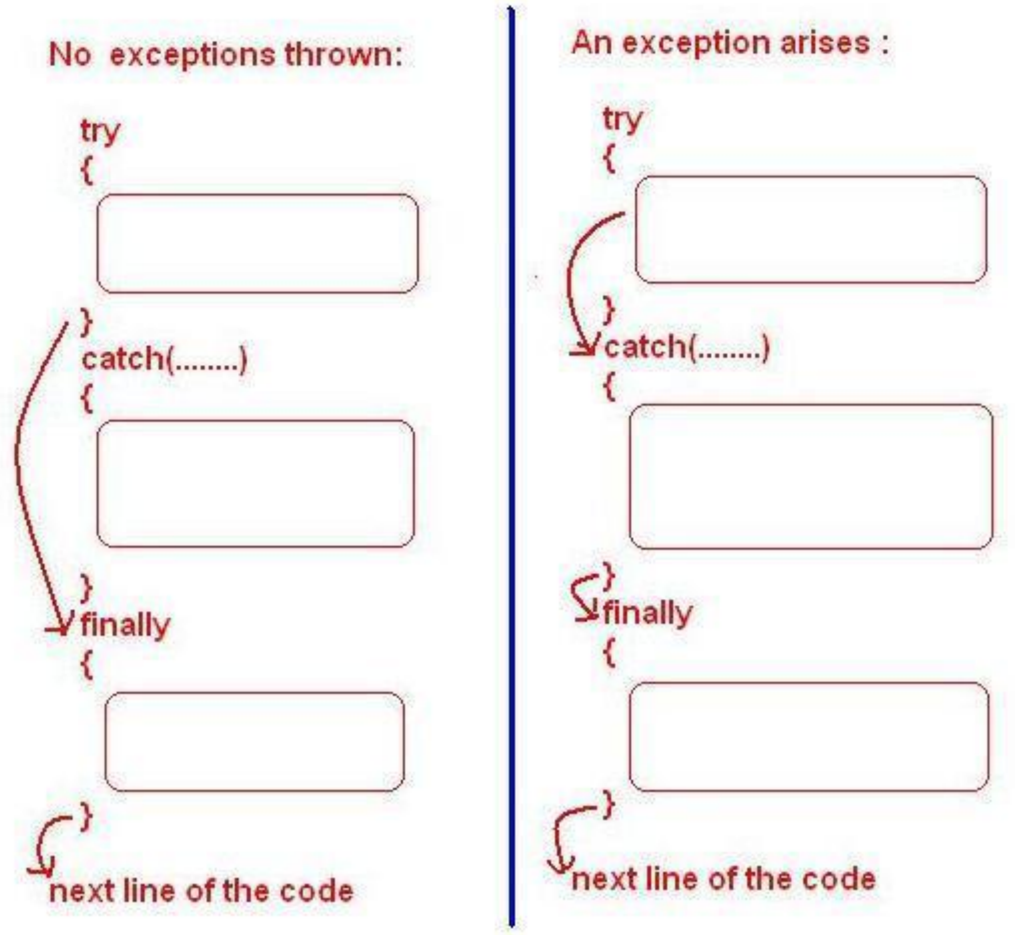
Sample program using throw

```
  // This is a caller function
   public static void main(String args[])
   {
       try {
           fun();
       }
       catch (IllegalAccessException e) {
           System.out.println("caught in main.");
       }
   }
}
```

OUTPUT:

```
                 Inside fun().
                caught in main.
```

With finally block



Sample Program using finally

```
class Division
{
   public static void main(String[] args)
   {    int a = 10, b = 5, c = 5, result;
       try {
           result = a / (b - c);
           System.out.println("result" + result);
       }
         catch (Arithmetic Exception e) {
           System.out.println("Exception caught:Division by zero");
       }
         finally
         {
                 System.out.println("I am in final block");
         }
        }
        }
```

OUTPUT:

```
Exception caught:Division by zero
                    I am in final block
```

Video Link

https://www.youtube.com/watch?v=z0q61egYRzs