SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23CSB101- OBJECT ORIENTED PROGRAMMING

I YEAR /II SEMESTER

Unit III – EXCEPTION HANDLING AND MULTITHREADING

Topic : EXCEPTION HANDLING

EXCEPTION HANDLING

•An **Exception** is an event that occurs during program execution which disrupts the normal flow of a program. It is an object which is thrown at runtime

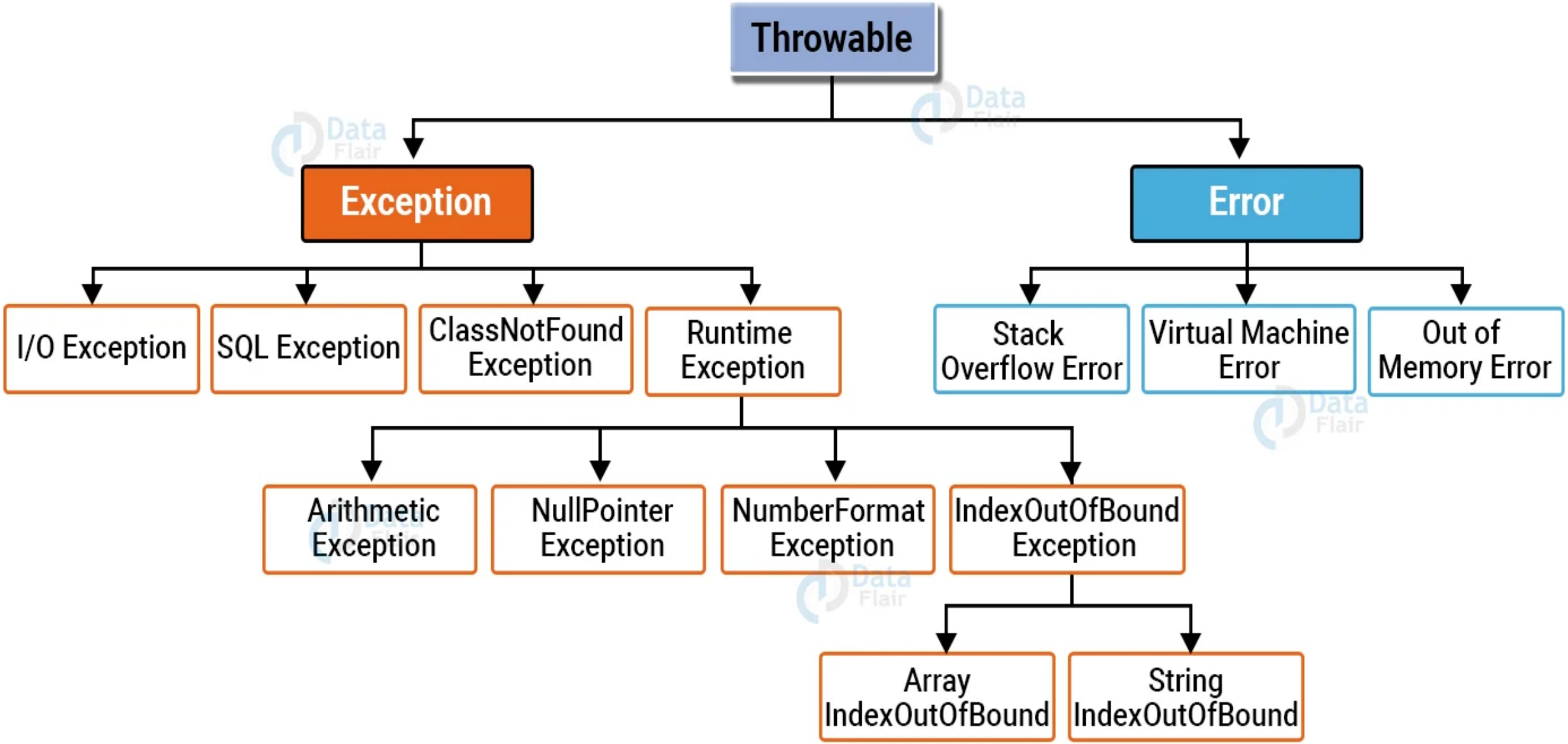
•**Exception handling** is a mechanism that allows you to handle runtime errors and exceptional conditions in a graceful manner, preventing the program from crashing. This is done using a combination of try, catch, throw, throws, and finally blocks.

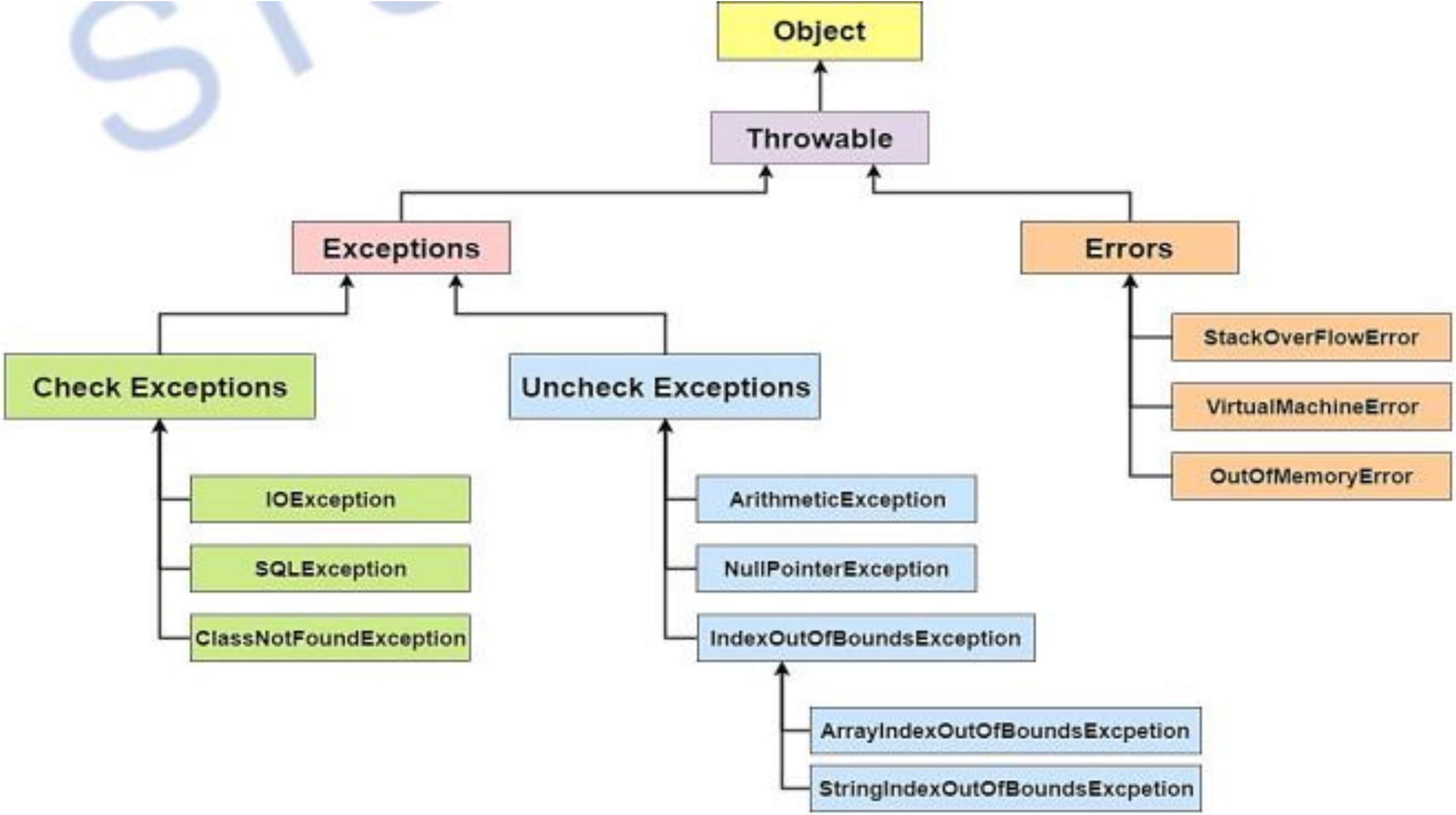
•All exceptions and errors extend from a common java.lang.Throwable parent class.

•The Throwable class is further divided into two classes:

1. Exceptions and
2. Errors.

Hierarchy of Java Exceptions




Object
Throwable
Exceptions
Errors
Check Exceptions
Uncheck Exceptions
StackOverFlowError
VirtualMachineError
OutOfMemoryError
IOException
SQLException
ClassNotFoundException
ArithmeticException
NullPointerException
IndexOutOfBoundsException
ArrayIndexOutOfBoundsExcpetion
StringIndexOutOfBoundsExcpetion

EXCEPTION HANDLING

Some scenarios where an exception occurs.

• A user has entered an invalid data.

• A file that needs to be opened cannot be found.

• A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are **caused by user error,** others **by programmer** error, and others **by physical resources** that have failed in some manner

Errors: Errors represent internal errors of the Java run-time system which could not be handled easily. Eg. OutOfMemoryError.

DIFFERENCE BETWEEN EXCEPTION AND ERROR

S.No.	Exception	Error
1	Exceptions can be recovered	Errors cannot be recovered
2	Exceptions are of type java.lang.Exception	Errors are of type java.lang.Error
3	Exceptions can be classified into two types: a) Checked Exceptions b) Unchecked Exceptions	There is no such classification for errors. Errors are always unchecked.
4	In case of Checked Exceptions, compiler will have knowledge of checked exceptions and force to keep try…catch block. Unchecked Exceptions are not known to compiler because they occur at run time.	In case of Errors, compiler won't have knowledge of errors. Because they happen at run time.
5	Exceptions are mainly caused by the application itself.	Errors are mostly caused by the environment in which application is running.
6	**Examples**: Checked Exceptions: SQLException, IOException Unchecked Exceptions: ArrayIndexOutOfBoundsException, NullPointerException	**Examples**: Java.lang.StackOverFlowError, java.lang.OutOfMemoryError

EXCEPTION HANDLING

Advantage of using Exceptions:

• Maintains the normal flow of execution of the application.

•Exceptions separate error handling code from regular code.

o **Benefit**:

• Cleaner algorithms, less clutter

• Meaningful Error reporting.

• Exceptions standardize error handling.

EXCEPTION HANDLING KEYWORDS

S.No.	Keyword	Description
1	try	A block of code that is to be monitored for exception
2	catch	The catch block handles the specific type of exception along with the try block. For each corresponding try block there exists the catch block.
3	finally	It specifies the code that must be executed even though exception may or may not occur.
4	throw	This keyword is used to explicitly throw specific exception from the program code.
5	throws	It specifies the exceptions that can be thrown by a particular method.

EXCEPTION HANDLING

try…catch block

- The try block is used to wrap the code that might throw an exception. If an exception occurs, the control is transferred to the catch block.
- If an exception is generated within the try block, the remaining statements in the try block are not executed
- The catch block is used to handle the exception.

```
try {
    int result = 10 / 0;  // This will throw ArithmeticException
} catch (ArithmeticException e) {
    System.out.println("Error: Division by zero!");    }
```

EXCEPTION HANDLING

catch Block:

•Exceptions thrown during execution of the try block can be caught and handled in a catch block.

•On exit from a catch block, normal execution continues and the finally block is executed.

finally Block:

•A finally block is always executed, regardless of the cause of exit from the try block, or whether any catch block was executed.

•Generally finally block is used for freeing resources, cleaning up, closing connections etc.

•Even though there is any exception in the try block, the statements assured by finally block are sure to execute

EXCEPTION HANDLING

Syntax:

```
try {
 // Code block
}
catch (ExceptionType1 e1) {
 // Handle ExceptionType1 exceptions
}
catch (ExceptionType2 e2) {
 // Handle ExceptionType2 exceptions
}
// ...
finally {
 // Code always executed after the
 // try and any catch block
}
```

Example:

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

```
java.lang.ArithmeticException: / by zero
finally block is always executed
```

Multiple catch blocks

- **Multiple catch** is used to handle many different kind of exceptions that may be generated while running the program. i.e more than one catch clause in a single try block can be used.

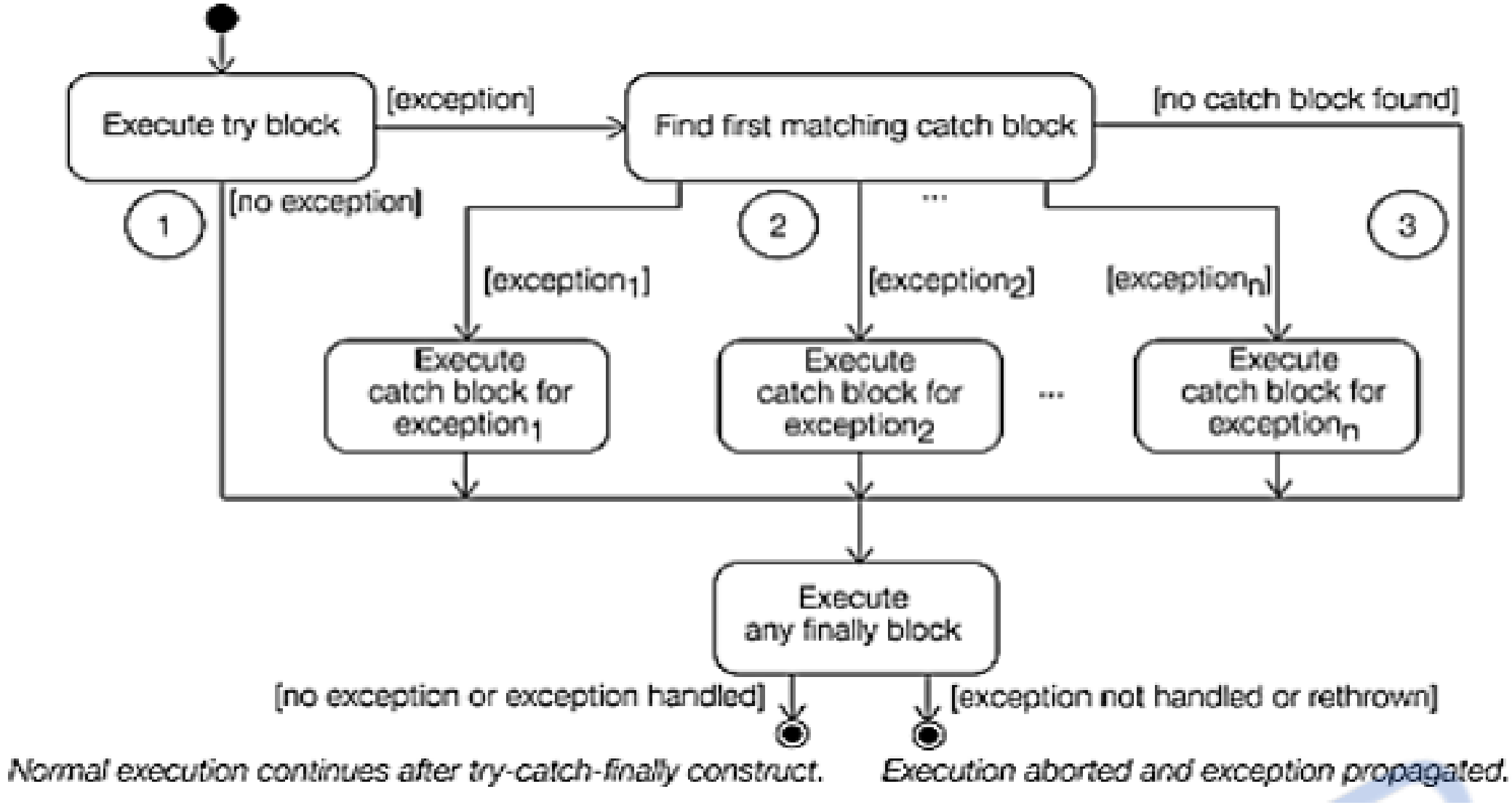
Rules:

1. At a time only one Exception can occur and at a time only one catch block is executed.
2. All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception.

Multiple catch blocks

```
try {
 // Code block
}
catch (ExceptionType1 e1) {
 // Handle ExceptionType1 exceptions
}
catch (ExceptionType2 e2) {
 // Handle ExceptionType2 exceptions
}
```

Multiple catch blocks



Multiple catch - Example

```
public class MultipleCatchBlock2 {
public static void main(String[] args) {
 try
{
 int a[]= {1,5,10,15,16};
 System.out.println("a[1] = "+a[1]);
 System.out.println("a[2]/a[3] = "+a[2]/a[3]);
 System.out.println("a[5] = "+a[5]);
 }
 catch(ArithmeticException e)
 {
 System.out.println("Arithmetic Exception occurs");
 }
```

```
catch(ArrayIndexOutOfBoundsException e)
 {
 System.out.println("ArrayIndexOutOfBounds
   Exception occurs");
 }
 catch(Exception e)
 {
 System.out.println("Parent Exception occurs");
 }
System.out.println("rest of the code"); }
 }
```

```
a[1] = 5
a[2]/a[3] = 0
ArrayIndexOutOfBounds Exception occurs
rest of the code `
```

NESTED try BLOCK

try block within a try block is known as nested try block

Syntax:

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

```
class NestedExcep
{
 public static void main(String[] args)
 {
  try
  {
   int arr[]={1,5,4,10};
   try
   {
    int x=arr[3]/arr[1];
    System.out.println("Quotient = "+x);
   }
   catch(ArithmeticException ae)
   {
    System.out.println("divide by zero");
   }
   arr[4]=3;
  }
  catch(ArrayIndexOutOfBoundsException e)
  {
   System.out.println("array index out of bound exception");
  }
  System.out.println("...End of Program...");
 }
}
```

```
Quotient = 2
array index out of bound exception
...End of Program...
```

THROWING AND CATCHING EXCEPTIONS

throw statement:

- The throw statement is used to explicitly throw an exception. User can create their own exceptions or re-throw an existing one. An exception can be thrown explicitly

    1. Using the **throw** statement
    2. Using the **throws** statement

The general format :

throw <exception reference>;

Example:

```
if (x < 18) {
    throw new ArithmeticException(“Not eligible");
}
```

THROWING AND CATCHING EXCEPTIONS

throws keyword:

- The **throws** keyword is used in method declarations to indicate that a method can throw one or more exceptions. It does not handle the exception but delegates the responsibility to the caller of the method.

Syntax:

```
public void someMethod() throws IOException {
    // Code that might throw an IOException }
```

Or

```
Return-type method_name(arg_list) throws exception_list
{ // method body }
```

EXCEPTION HANDLING

```
public class ThrowsDemo
{
static void divide(int num, int din) throws ArithmeticException
{
int result=num/din;
System.out.println("Result : "+result);
}
public static void main(String args[])
{
int n = 10 ,d =0;
try
{
    divide(n,d);
}
catch(Exception e)
{
System.out.println(" Can't Handle : divide by zero ERROR");
}
}
}
```

```
Can't Handle : divide by zero ERROR **
```

Example: try...catch...finally

```
public class TrycatchExample
{
    public static void main(String[] args)
    {
        try {
            int[] numbers = {1, 2, 3};
            int result = numbers[5]; // This will throw ArrayIndexOutOfBoundsException
            int division = 10 / 0; // This will throw ArithmeticException
            }
      catch (ArrayIndexOutOfBoundsException e)
       {     System.out.println("Array index out of bounds!");        }
      catch (ArithmeticException e)
       {      System.out.println("Arithmetic error!");         }
      finally
       {     System.out.println("This block runs no matter what");
        }
     }
 }
```

Example: throw

```
class ThrowExample {
    static void checkAge(int age) {
        if (age < 18) {
            throw new IllegalArgumentException("Age must be 18 or older.");
        } else {
            System.out.println("Access granted.");
        }
    }
    public static void main(String[] args) {
        checkAge(16);  // This will throw an exception
    }
}
```

```
Exception in thread "main" java.lang.IllegalArgumentException: Age must be 18 or older.
```

EXCEPTION HANDLING

```
class AgeValidation {
  // Method that declares it may throw an exception
static void checkAge(int age) throws
IllegalArgumentException
 {
        if (age < 18)
       {
 throw new IllegalArgumentException("Age
                                must be 18 or older.");
        }
else {
            System.out.println("Access granted.");
        }
    }
```

```
public static void main(String[] args) {
        try {
                checkAge(16);
// This will cause an exception
        }
catch (IllegalArgumentException e)
{
 System.out.println("Exception caught: " +
e.getMessage());
        }
    }
}
```

Difference between throw and throws

throw keyword	throws keyword
throw is used to explicitly throw an exception	throws is used to declare an exception.
checked exception cannot be propagated without throws.	checked exception can be propagated with throws.
throw is followed by an instance	throws is followed by class.
throw is used within the method.	throws is used with the method signature
You cannot throw multiple exception	You can declare multiple exception e.g. public void method()throws IOException,SQLException.

EXCEPTION HANDLING

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
class IOExceptionExample {
    public static void main(String[] args) {
        try {
            // Attempt to read a non-existent file
            BufferedReader reader = new BufferedReader(new FileReader("non_existent_file.txt"));
            String line = reader.readLine();
            System.out.println(line);
            reader.close();
        } catch (IOException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

```
Exception caught: non_existent_file.txt (No such file or directory)
```

EXCEPTION HANDLING

```
import java.io.*;

public class CheckedExceptionExample {
    public static void main(String[] args) {
        try (FileReader file = new FileReader("src/somefile.java")) {
// Try-with-resources (Auto-closes the file)
      System.out.println(file.toString());
// Note: This prints object reference, not file content
        } catch (FileNotFoundException e) {
            System.out.println("Sorry. File not found.");
        } catch (IOException e) {
// Handles any reading errors
            System.out.println("Error while reading the file.");
        }
    }
}
```

Sorry. File not found.


Thank
you