



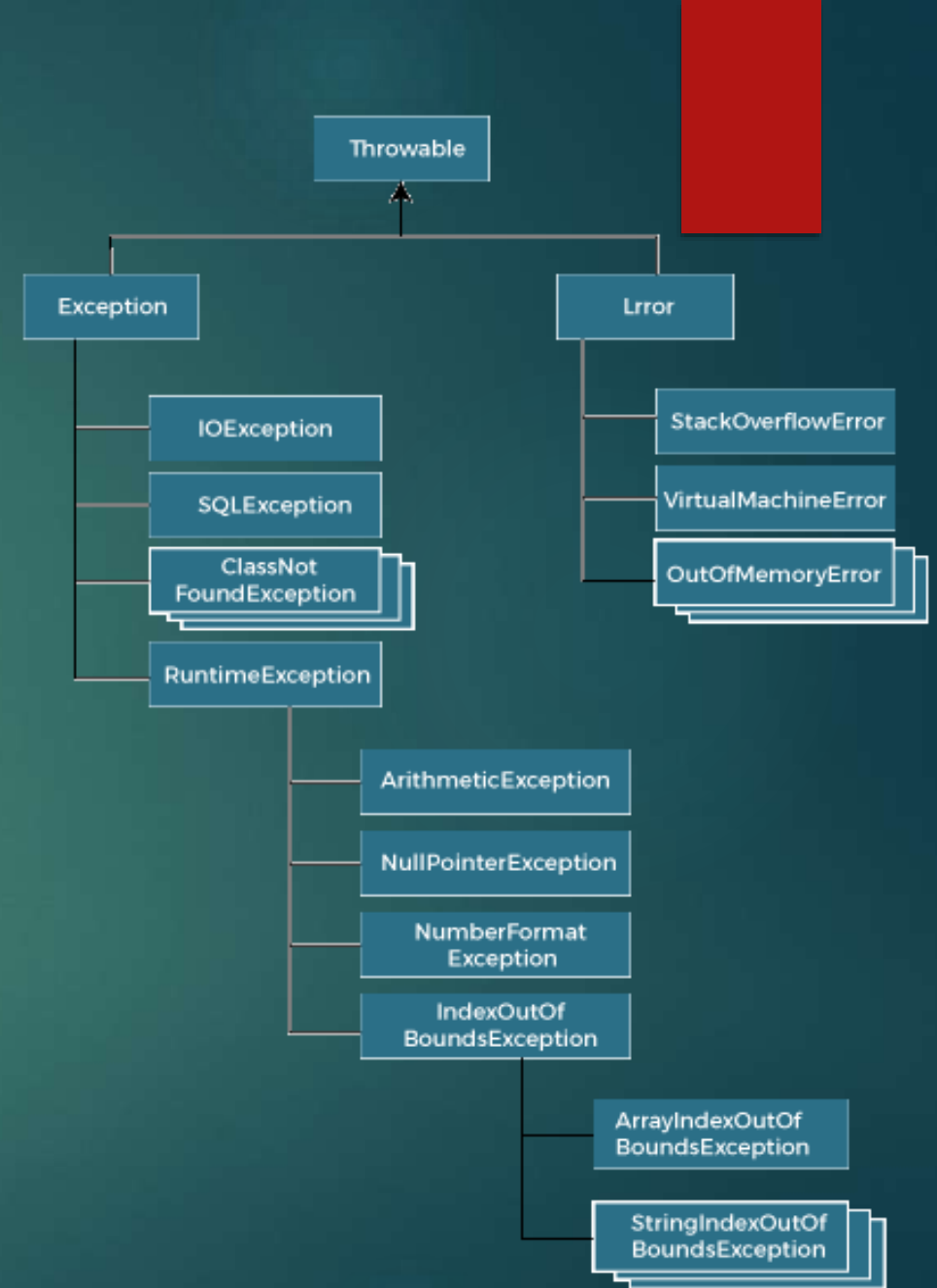
# JAVA PROGRAMMING

## Chap 6 : Exception Handling

- ▶ Exception handling refers to handling of abnormal or unexpected events.
- ▶ Some of these exceptions are known to the compiler while some other occur during runtime and are unknown to the compiler.
- ▶ The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc. so that the normal flow of the application can be maintained.
- ▶ Exceptions can be caught and handled by the program.
- ▶ When an exception occurs within a method, it creates an object.
- ▶ This object is called the exception object.
- ▶ It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.
- ▶ Major reasons why an exception Occurs
  - ▶ Invalid user input
  - ▶ Device failure
  - ▶ Loss of network connection
  - ▶ Physical limitations (out of disk memory)
  - ▶ Code errors
  - ▶ Opening an unavailable file

- ▶ **Errors** represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.
- ▶ Errors are usually beyond the control of the programmer, and we should not try to handle errors.
- ▶ When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.
- ▶ When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).
- ▶ “Exception” is a class and has subclasses according to the exceptions possible in Java.
- ▶ Eg: If a integer is expected while taking input and instead if a character is entered, then while parsing, error will occur. This error is called NumberFormatException. It is one of the subclass of the base class Exception.
- ▶ **Exceptions are classified as :**
- ▶ **Built-in Exceptions**
  - ▶ Checked Exception
  - ▶ Unchecked Exception
- ▶ **User-Defined Exceptions**

- ▶ The `java.lang.Throwable` class is the root class of Java Exception hierarchy inherited by two subclasses: `Exception` and `Error`.
- ▶ All exception and error types are subclasses of class **Throwable**, which is the base class of the hierarchy.
- ▶ One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. `NullPointerException` is an example of such an exception.
- ▶ Another branch, **Error** is used by the Java run-time system([JVM](#)) to indicate errors having to do with the run-time environment itself(JRE). `StackOverflowError` is an example of such an error.



# Built-in Exceptions

- ▶ Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.
- ▶ **Checked Exception** : The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.
- ▶ Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
- ▶ Checked Exceptions make the programmers to handle the exception that may be thrown.
- ▶ There are two ways to deal with an exception :
  - ▶ Indicate that the method throws an exception
  - ▶ Method must catch the exception and take appropriate action using the try catch block
- ▶ **Unchecked Exception** : The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- ▶ The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time, but they are checked at runtime.
- ▶ In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.
- ▶ Hence the programmers may not even know that such an exception would be thrown

# Built-in Exceptions

- ▶ There are given some scenarios where unchecked exceptions may occur. They are as follows:

- ▶ If we divide any number by zero, there occurs an `ArithmeticException`.

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

- ▶ If we have a null value in any variable, performing any operation on the variable throws a `NullPointerException`.

```
String s=null;
```

```
System.out.println(s.length());//NullPointerException
```

- ▶ If the formatting of any variable or number is mismatched, it may result into `NumberFormatException`. Suppose we have a string variable that has characters; converting this variable into digit will cause `NumberFormatException`.

```
String s="abc";
```

```
int i=Integer.parseInt(s);//NumberFormatException
```

- ▶ When an array exceeds to its size, the `ArrayIndexOutOfBoundsException` occurs

```
int a[]=new int[5];
```

```
a[10]=50; //ArrayIndexOutOfBoundsException
```



# User defined Exceptions

- ▶ Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called 'user-defined Exceptions'.
- ▶ The advantages of Exception Handling in Java are as follows:
  - ▶ Provision to Complete Program Execution
  - ▶ Easy Identification of Program Code and Error-Handling Code
  - ▶ Propagation of Errors
  - ▶ Meaningful Error Reporting
  - ▶ Identifying Error Types

# Java's Unchecked Exceptions

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

**Table 10-1** Java's Unchecked **RuntimeException** Subclasses Defined in **java.lang**



# Java's Checked Exceptions

Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the <b>Cloneable</b> interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.
ReflectiveOperationException	Superclass of reflection-related exceptions.

**Table 10-2** Java's Checked Exceptions Defined in **java.lang**

WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException.

```
import java.io.*;
class Main{
    public static void main(String args[])throws IOException{
        int a,b,res;
        BufferedReader sc=new BufferedReader (new InputStreamReader (System.in));
        System.out.println("Enter 2 nos.");
        a=Integer.parseInt(sc.readLine());
        b=Integer.parseInt(sc.re adLine());
        res=a/b;
        System.out.println("The Result is : "+res);
    }
}
```

ArithmeticException →

```
Enter 2 nos.
4
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Main.main(Main.java:9)
```

NumberFormatException →

```
Enter 2 nos.
5
d
Exception in thread "main" java.lang.NumberFormatException: For input string: "d"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at Main.main(Main.java:11)
```

# Java Exception Keywords

- ▶ Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

# try-catch-finally

## ▶ Try Block :

- ▶ The statements that are prone to generate errors or exception are placed in try block.
- ▶ If an exception occurs then the catch block will be executed and then the finally block will be executed.
- ▶ Else if no exception occurs, then the control will directly go to the finally block i.e. the catch block will not be executed.
- ▶ The java code that you think can generate an exception is placed in the try catch block to handle the error.
- ▶ If an exception occurs but a matching catch block is not found then it reaches to the finally block.
- ▶ In any case, when the exception occurs in a particular statement of try block, the remaining statements of the try block after this statement are not executed i.e. no statements of the try block are executed after the exception generating statement.

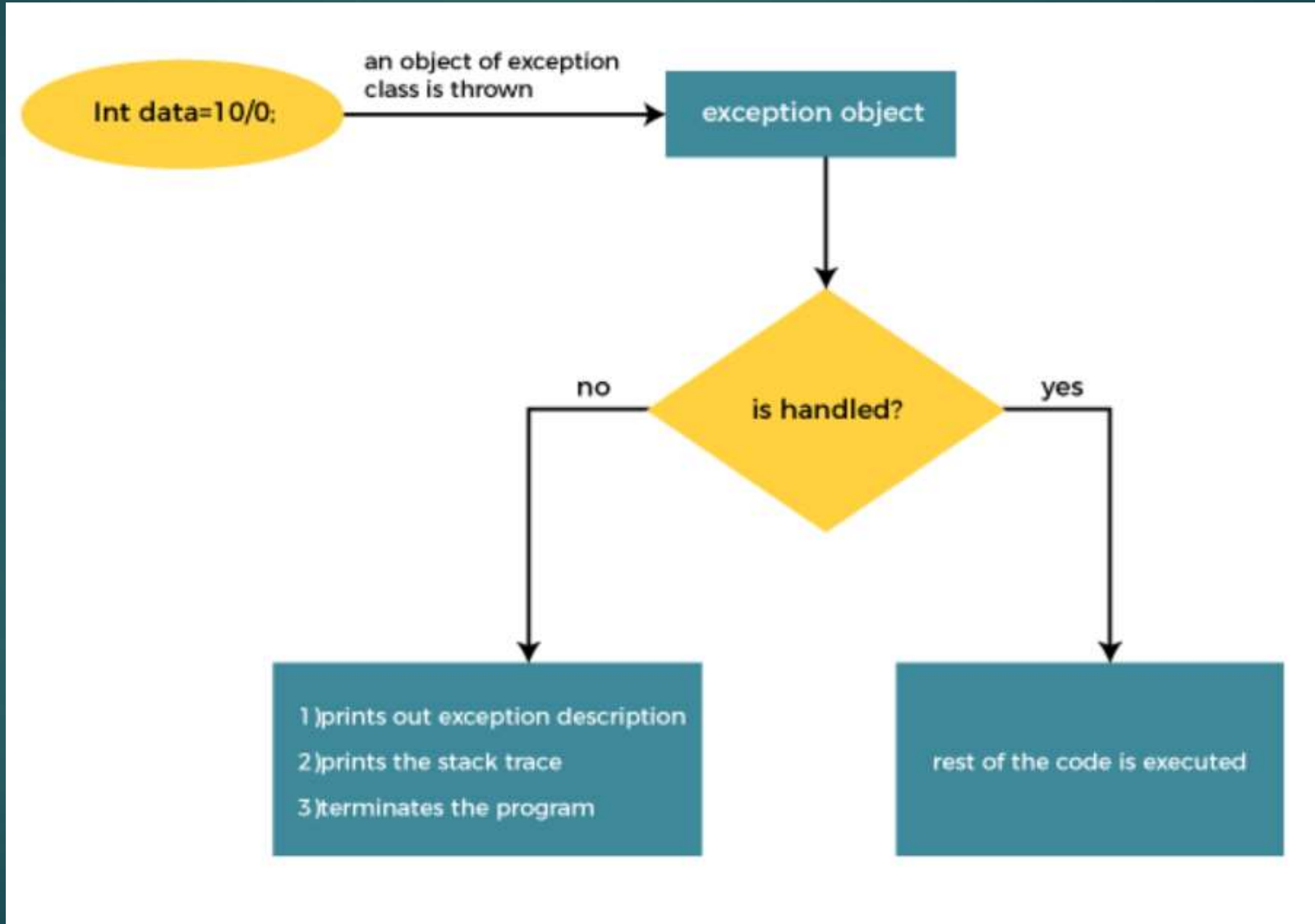
## ▶ Catch Block :

- ▶ The exception thrown by the try block are caught by the catch block.
- ▶ The type of the exception occurred must match with the exception mentioned in the brackets of the catch block.

## ▶ Finally Block :

- ▶ A finally block is always executed irrespective of whether the exception occurred or not
- ▶ It is an optional block. It is not necessary to have a finally block i.e. you may just have try catch blocks.

# Internal Working of Java try-catch block





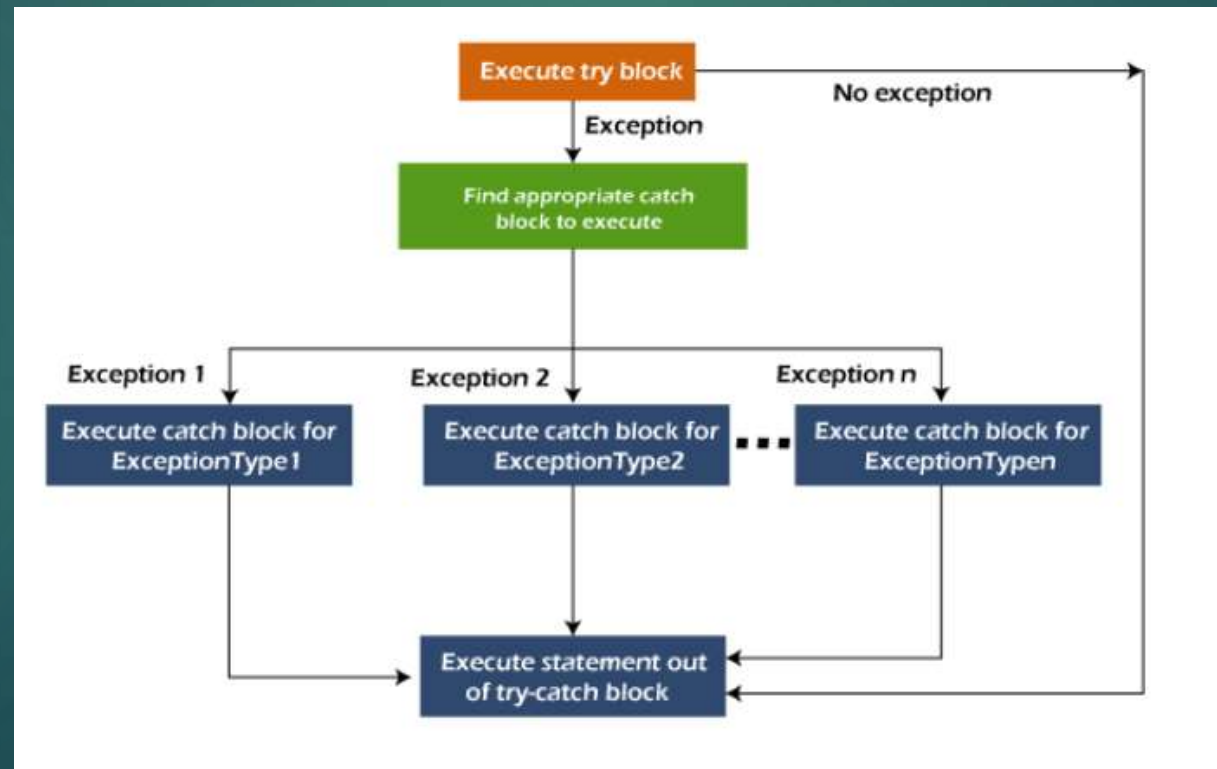
WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException using try catch block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a,b,res;
        BufferedReader sc=new BufferedReader (new InputStreamReader (System.in));
        System.out.println("Enter 2 nos.");
        a=Integer.parseInt(sc.readLine());
        b=Integer.parseInt(sc.readLine());
        try {
            res=a/b;
            System.out.println("The Result is : "+res);
        }
        catch (ArithmeticException ae)
        {System.out.println("Exception has occurred as divisor entered is zero");
        }
        System.out.println("Remaining code");
    }
}
```

```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Remaining code
```

# Multiple catch blocks

- ▶ A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- ▶ **Points to remember**
- ▶ At a time only one exception occurs and at a time only one catch block is executed.
- ▶ All catch blocks must be ordered from most specific to most general, i.e. catch for `ArithmeticException` must come before catch for `Exception`.



WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException using try catch block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a=0,b=0,res;
        BufferedReader sc=new BufferedReader (new
InputStreamReader (System.in));
        try {
            System.out.println("Enter 2 nos.");
            a=Integer.parseInt(sc.readLine());
            b=Integer.parseInt(sc.readLine());
            res=a/b;
            System.out.println("The Result is : "+res);
            String s=null;
            System.out.println(s.length());
        }
        catch (ArithmeticException ae) {
            System.out.println("Exception has occurred as divisor entered
is zero");
        }
        catch (NumberFormatException ne) {
            System.out.println("Invalid Input. Enter Integer Number.");
        }
        catch(Exception e) {
            System.out.println(e);
        }
        System.out.println("Remaining Code Continues");
    }
}
```

```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Remaining Code Continues
```

```
Enter 2 nos.
5
d
Invalid Input. Enter Integer Number.
Remaining Code Continues
```

```
Enter 2 nos.
5
2
The Result is : 2
java.lang.NullPointerException
Remaining Code Continues
```

# Nested try blocks

- ▶ 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.
- ▶ In Java, using a try block inside another try block is permitted. It is called as nested try block.
- ▶ Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.
- ▶ For example, the inner try block can be used to handle `ArrayIndexOutOfBoundsException` while the outer try block can handle the `ArithmeticException` (division by zero).

```
....  
//main try block  
try  
{  
    statement 1;  
    statement 2;  
//try catch block within another try block  
    try  
    {  
        statement 3;  
        statement 4;  
//try catch block within nested try block  
        try  
        {  
            statement 5;  
            statement 6;  
        }  
    }  
}
```

```
catch(Exception e2)  
{  
    //exception message of 2nd nested try block  
}  
catch(Exception e1)  
{  
    //exception message of 1st nested try block  
}  
catch(Exception e3)  
{  
    //exception message of parent (outer) try block  
}  
....
```

WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException using nested try catch block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a=0,b=0,res;
        BufferedReader sc=new BufferedReader (new
InputStreamReader (System.in));
        try {
            System.out.println("Enter 2 nos.");
            a=Integer.parseInt(sc.readLine());
            b=Integer.parseInt(sc.readLine());
            try {
                res=a/b;
                System.out.println("The Result is : "+res);
            }
            catch (ArithmeticException ae) {
                System.out.println("Exception has occurred as divisor
entered is zero");
            }
        }
        catch (NumberFormatException ne) {
            System.out.println("Invalid Input. Enter Integer Number.");
        }
        System.out.println("Remaining Code Continues");
    }
}
```

```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Remaining Code Continues
```

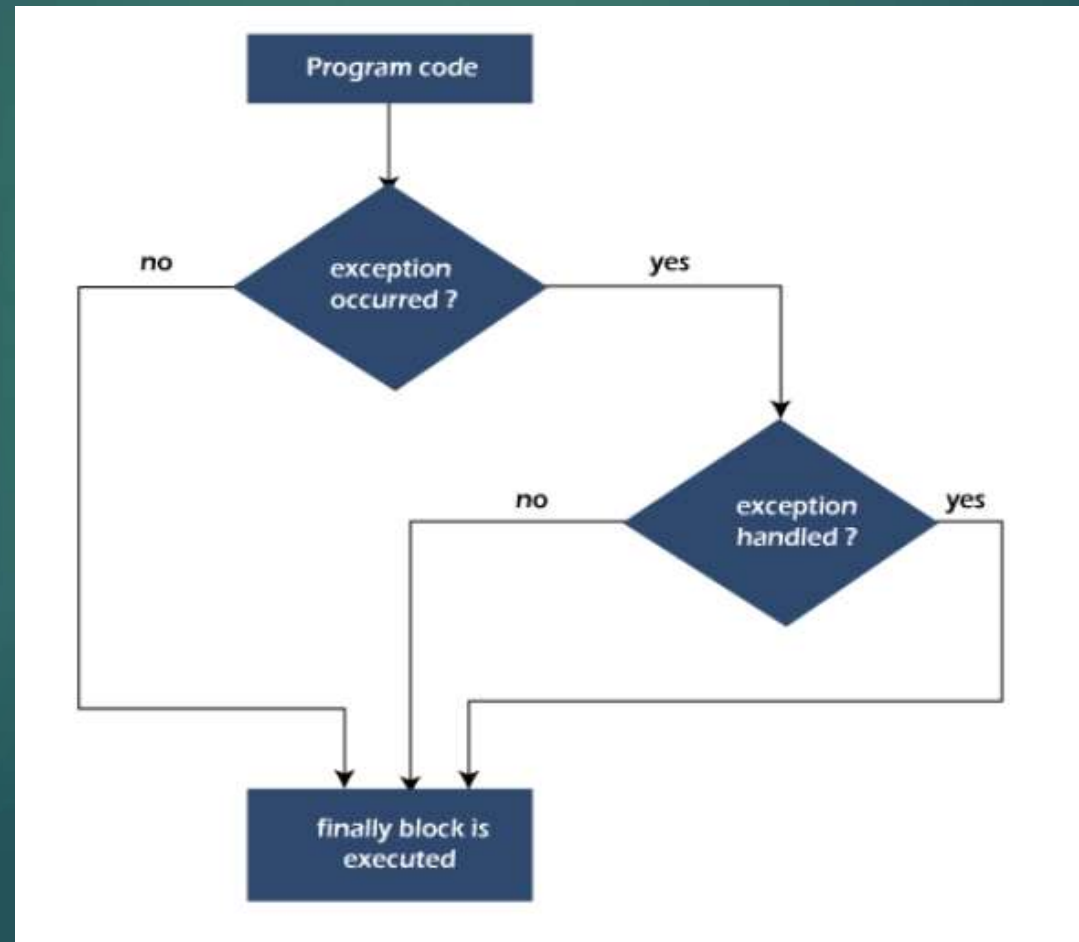
```
Enter 2 nos.
5
d
Invalid Input. Enter Integer Number.
Remaining Code Continues
```

```
Enter 2 nos.
10
2
The Result is : 5
Remaining Code Continues
```



# Internal Working of Java try-catch-finally block

- ▶ A finally block is always executed irrespective of whether the exception occurred or not
- ▶ It is an optional block. It is not necessary to have a finally block i.e. you may just have try catch blocks.
- ▶ finally block in Java can be used to put "cleanup" code such as closing a file, closing connection, etc.
- ▶ The important statements to be printed can be placed in the finally block.



WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException using try-catch-finally block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a=0,b=0,res;
        BufferedReader sc=new BufferedReader (new
InputStreamReader (System.in));
        try {
            System.out.println("Enter 2 nos.");
            a=Integer.parseInt(sc.readLine());
            b=Integer.parseInt(sc.readLine());
            try {
                res=a/b;
                System.out.println("The Result is : "+res);
            }
            catch (ArithmeticException ae)
            { System.out.println("Exception has occurred as divisor entered
is zero");
            }
        }
        catch (NumberFormatException ne)
        { System.out.println("Invalid Input. Enter Integer Number.");
        }
        finally {
            System.out.println("Finally block executed");
        }
    }
}
```

```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Finally block executed
```

```
Enter 2 nos.
5
d
Invalid Input. Enter Integer Number.
Finally block executed
```

```
Enter 2 nos.
10
2
The Result is : 5
Finally block executed
```

# “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 the normal flow of the program can be maintained.
- ▶ Exception Handling is mainly used to handle the checked exceptions.
- ▶ Syntax of Java throws

```
return_type method_name() throws exception_class_name{  
    //method code  
}
```
- ▶ throws keyword in Java is used in the signature of method to indicate that this method might throw one of the listed type (i.e. checked) exceptions. The caller to these methods has to handle the exception using a try-catch block.
- ▶ 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.

WAP to demonstrate throws keyword.

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        int a;
        System.out.println("Enter age :");
        Scanner sc=new Scanner(System.in);
        a=sc.nextInt();
        checkAge(a);
    }
    static void checkAge(int age) throws ArithmeticException
    {
        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!");
        }
    }
}
```

```
Enter age :
20
Access granted - You are old enough!
```

```
Enter age :
15
Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.
    at Main.checkAge(Main.java:13)
    at Main.main(Main.java:8)
```

# “throw” keyword

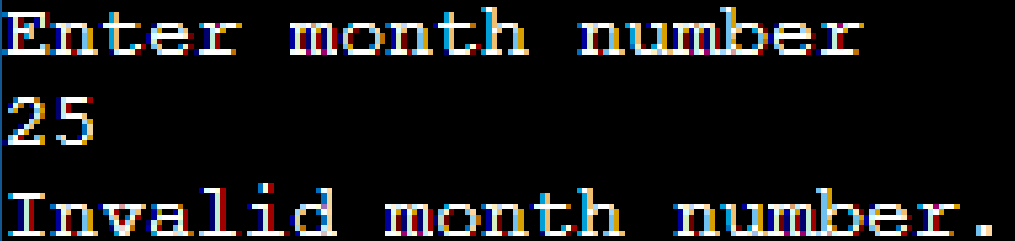
- ▶ The Java throw keyword is used to throw an exception explicitly.
- ▶ We specify the exception object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.
- ▶ We can throw either checked or unchecked exceptions in Java by throw keyword.
- ▶ It is mainly used to throw a custom exception i.e. you can make your own conditions to throw exceptions.
- ▶ It will not be a built in exception but it will be your user defined exception.



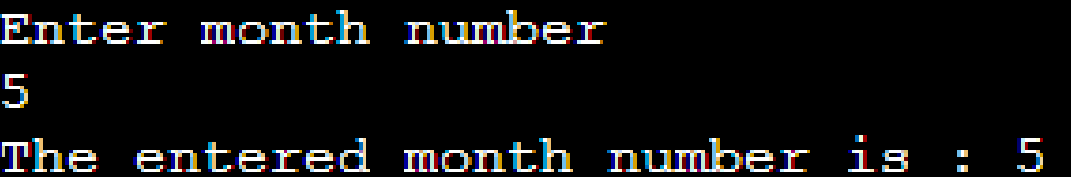
WAP to accept and display month number. Throw a NumberFormatException if month number exceeds 12 or is less than 0.

```
import java.util.*;
class Main{
    public static void main(String args[]){
        int m;
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter month number");
        m=sc.nextInt();
        try {
            if(m<1 || m>12)
                throw new NumberFormatException();
            System.out.println("The entered month number is :
"+m);
        }
        catch (NumberFormatException ne)
        { System.out.println("Invalid month number.");
        }
    }
}
```

A screenshot of a terminal window with a black background and white text. The text shows the prompt "Enter month number", the user input "25", and the program output "Invalid month number.".

```
Enter month number
25
Invalid month number.
```

A screenshot of a terminal window with a black background and white text. The text shows the prompt "Enter month number", the user input "5", and the program output "The entered month number is : 5".

```
Enter month number
5
The entered month number is : 5
```

WAP to accept and display month number. Throw an Exception if month number exceeds 12 or is less than 0. Make your own exception class to handle this exception.

```
import java.util.*;
class MonthNumberException extends Exception {
    MonthNumberException()
    {
        System.out.println("Invalid Month Number");
    }
}
class Main{
    public static void main(String args[]){
        int m;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter month number");
        m=sc.nextInt();
        try {
            if(m<1 || m>12)
                throw new MonthNumberException();
            System.out.println("The entered month number is :
"+m);
        }
        catch (MonthNumberException me)
        {
        }
    }
}
```

```
Enter month number
25
Invalid month number.
```

```
Enter month number
5
The entered month number is : 5
```

# Programming Practice Questions

- ▶ Write a Java program that simulates a simple banking system. Implement nested try-catch blocks to handle:
  1. `ArithmeticException` when dividing by zero in calculating interest.
  2. `NumberFormatException` if the user inputs invalid account details.
- ▶ Create a custom exception class `InvalidAgeException` that extends `Exception`. Write a Java program that uses this custom exception to validate a user's age. If the user enters an age less than 0 or greater than 100, throw and handle `InvalidAgeException`.
- ▶ Create a Java application that prompts the user to enter a number and performs a calculation with it. Implement exception handling to manage invalid inputs and arithmetic errors, and ensure that any resources (e.g., scanners) used for input are closed in the finally block.
- ▶ Develop a program that prompts the user for two integers and performs division. Use multiple catch blocks to handle `InputMismatchException` (if the user inputs non-numeric values), `ArithmeticException` (if there is a division by zero), and `Exception` (for any other unexpected errors).
- ▶ Develop a library management system where you can borrow and return books. Define custom exceptions for scenarios such as when a book is already borrowed (`BookAlreadyBorrowedException`), when a book is not available in the library (`BookNotFoundException`), and when a user tries to return a book that they haven't borrowed (`InvalidReturnOperationException`).