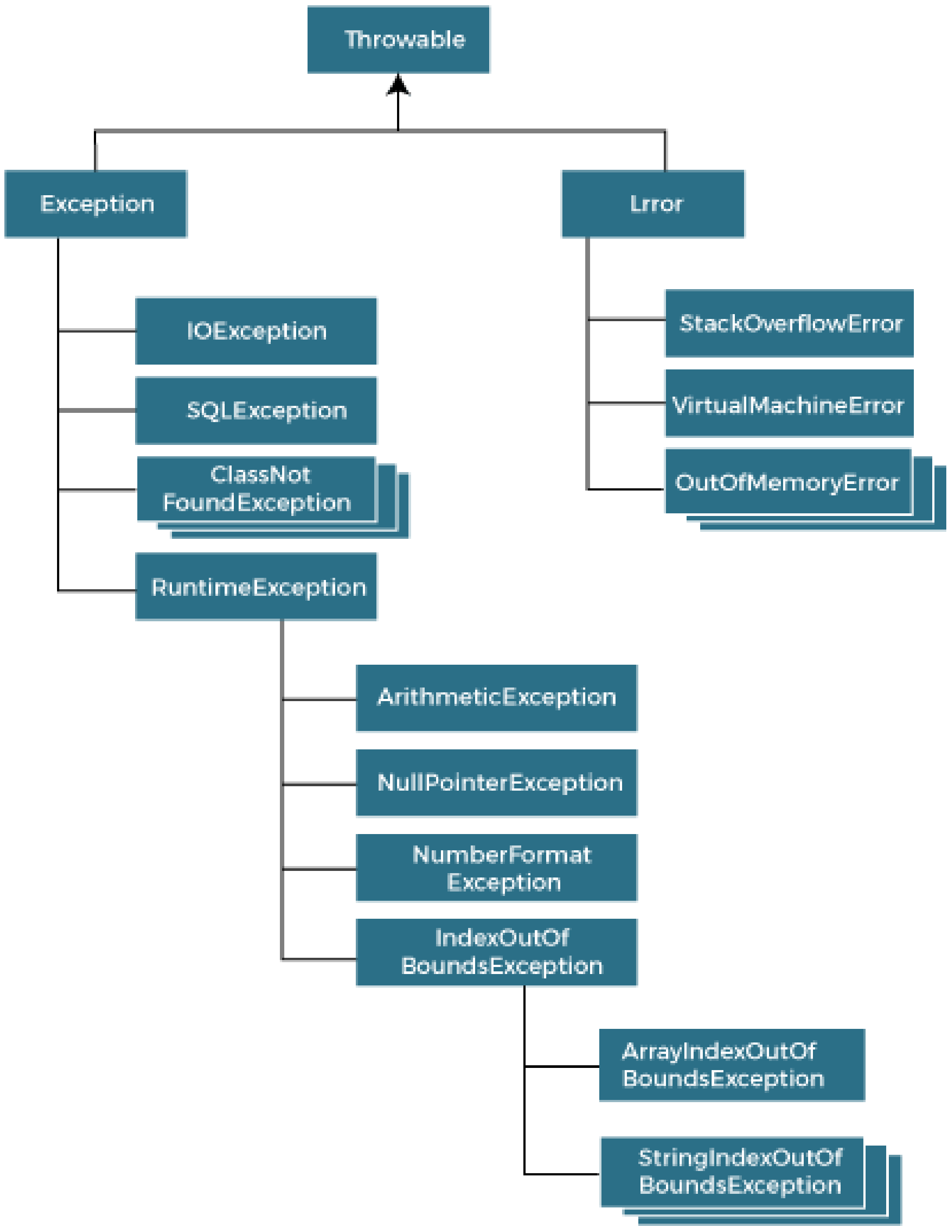
Exception Handling

The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error.



Types of Java Exceptions
There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception.

Checked Exception
Unchecked Exception

1) Checked Exception
The classes that directly inherit the Throwable class except RuntimeException class and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception
The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

ArithmeticException: It is thrown when an exceptional condition has occurred in an arithmetic operation.

ArrayIndexOutOfBoundsException: It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

ClassNotFoundException: This Exception is raised when we try to access a class whose definition is not found

FileNotFoundException: This Exception is raised when a file is not accessible or does not open.
IOException: It is thrown when an input-output operation failed or interrupted
InterruptedException: It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

NoSuchFieldException: It is thrown when a class does not contain the field (or variable) specified
NoSuchMethodException: It is thrown when accessing a method that is not found.

NullPointerException: This exception is raised when referring to the members of a null object. Null represents nothing

NumberFormatException: This exception is raised when a method could not convert a string into a numeric format.

RuntimeException: This represents an exception that occurs during runtime.

StringIndexOutOfBoundsException: It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string.
IllegalArgumentException : This exception will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.

IllegalStateException : This exception will throw an error or error message when the method is not accessed for the particular operation in the application. It comes under the unchecked exception.

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

Keywords used
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 may or may not 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.

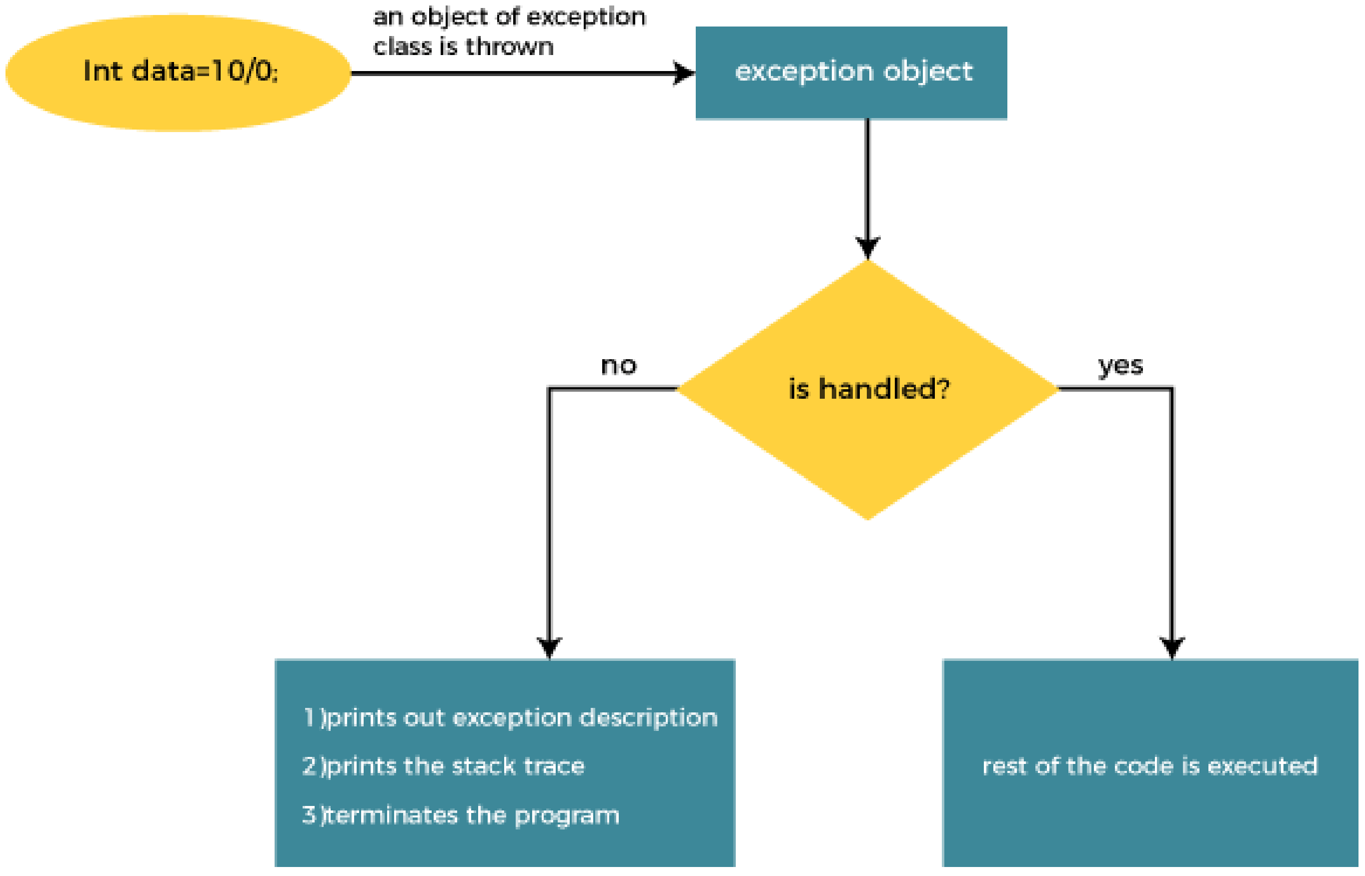
throw The "throw" keyword is used to throw an exception.

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.

Syntax of try block:
```
try
{
// code that might cause an exception
}
```

Syntax of try-finally block:
```
try
{
// code that might cause an exception
}
```

```
finally
{
// code that needs to be executed at all costs
}
```


Int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

```
// Java program to without try-catch Exception
class ArithmeticException_Demo
{
    public static void main(String args[])
    {
            int a = 30, b = 0;
            int c = a/b;  // cannot divide by zero
            System.out.println ("Result = " + c);
            }
}


// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
    public static void main(String args[])
    {
        try
          {
            int a = 30, b = 0;
            int c = a/b;  // cannot divide by zero
            System.out.println ("Result = " + c);
        }
        catch(ArithmeticException e) {
            System.out.println ("Can't divide a number by 0");
        }
    }
}
//Java program to demonstrate NullPointerException
class NullPointer_Demo
{
    public static void main(String args[])
    {
        try
            {
            String a = null; //null value
            System.out.println(a.charAt(0));
        }
catch(NullPointerException e) {
            System.out.println("NullPointerException..");
        }
    }
}
```

```java
// Java program to demonstrate StringIndexOutOfBoundsException
class StringIndexOutOfBound_Demo
{
    public static void main(String args[])
    {
        try {
            String a = "This is like chipping "; // length is 22
            char c = a.charAt(24); // accessing 25th element
            System.out.println(c);
        }
        catch(StringIndexOutOfBoundsException e) {
            System.out.println("No character is available on this index");
        }
    }
}


// Java program to demonstrate NumberFormatException
class  NumberFormat_Demo
{
    public static void main(String args[])
    {
        try {
            // "string" is not a number
            int num = Integer.parseInt ("string") ;

            System.out.println(num);
        } catch(NumberFormatException e) {
            System.out.println("Number format exception");
        }
    }
}


// Java program to demonstrate ArrayIndexOutOfBoundException
class ArrayIndexOutOfBound_Demo
{
    public static void main(String args[])
    {
        try{
            int a[] = new int[5];
            a[6] = 9; // accessing 7th element in an array of
                    // size 5
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println ("Array Index is Out Of Bounds");
        }
    }
}


// Java program to demonstrate IOException
class IOException_Demo {

    public static void main(String[] args)
    {

        // Create a new scanner with the specified String
        // Object
        Scanner scan = new Scanner("Hello Geek!");

        // Print the line
        System.out.println("" + scan.nextLine());

        // Check if there is an IO exception
        System.out.println("Exception Output: "+ scan.ioException());

        scan.close();
    }
}
```

User-Defined Exceptions

To create a user-defined exception:

1. The user should create an exception class as a subclass of the Exception class. Since all the exceptions are subclasses of the Exception class, the user should also make his class a subclass of it. This is done as:
```
        class MyException extends Exception
```
2. We can write a default constructor in its own exception class.
```
MyException()
{
        //Empty Body
}
```
3. We We can also create a parameterized constructor with a string as a parameter. We can use this to store exception details. We can call the superclass (Exception) constructor from this and send the string there.
```
MyException(String str)
{
  super(str);
}
```

4. To raise an exception of a user-defined type, we need to create an object to his exception class and throw it using the throw clause, as:
```
        MyException me = new MyException(“Exception details”);
        throw me;
```

```
// Java program to demonstrate user-defined exception

// This program throws an exception whenever balance
// amount is below Rs 1000
class MyException extends Exception
{
        //store account information
        private static int accno[] = {1001, 1002, 1003, 1004};

        private static String name[] =
                                {"Nish", "Shubh", "Sush", "Abhi", "Akash"};

        private static double bal[] =
                {10000.00, 12000.00, 5600.0, 999.00, 1100.55};
        // default constructor
        MyException() { }

        // parameterized constructor
        MyException(String str) { super(str); }

        // write main()
        public static void main(String[] args)
        {
                try {
                        // display the heading for the table
                        System.out.println("ACCNO" + "\t" + "CUSTOMER" +
                                                                            "\t" + "BALANCE");

                        // display the actual account information
                        for (int i = 0; i < 5 ; i++)
                        {
                                System.out.println(accno[i] + "\t" + name[i] +
                                                                            "\t" + bal[i]);

                                // display own exception if balance < 1000
                                if (bal[i] < 1000)
                                {
                                        MyException me =
                                        new MyException("Balance is less than 1000");
                                        throw me;
                                }
                        }
                } //end of try

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