Java Exceptions

* Exception is a run-time error which arises during the execution of java program.
* If the exception object is not handled properly, the interpreter will display the error and will terminate 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.
* All exception are subclasses of the class **Throwable**

# Reasons for Exceptions:

* Invalid user input
* Device failure
* Loss of network connection
* Physical limitations (out-of-disk memory)
* Code errors
* Out of bound
* Null reference
* Type mismatch
* Opening an unavailable file
* Database errors
* Arithmetic errors

# Exception Types

Examples of Checked exception – classNotFound, IO Exception, SQLExceptions, FileNotFound Exception. 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 the [throws keyword](https://www.geeksforgeeks.org/throw-throws-java/).

Example of handling checked exception:

*// Reading file from path in local directory*

FileReader file = **new** FileReader("C:\\test\\a.txt");

*// Creating object as one of ways of taking input*

BufferedReader fileInput = **new** BufferedReader(file);

*// Printing first 3 lines of file "C:\test\a.txt"*

**for** (int counter = 0; counter < 3; counter++)

System.out.println(fileInput.readLine());

*// Closing file connections*

*// using close() method*

fileInput.close();

Examples of Unchecked exception – Arithmetic exception, NullPointer Exception, ArrayIndexOutOfBound etc.

Unchecked exceptions are not required to be caught or declared in a throws clause. These exceptions are usually caused by programming errors

# Methods to print exceptions

1. **printStackTrace()**

int a=5;

int b=0;

try{

System.out.println(a/b);

}

catch(ArithmeticException e){

e.printStackTrace();

2. toString()

int a=5;

int b=0;

**try**{

System.out.println(a/b);

}

**catch**(ArithmeticException e){

System.out.println(e.toString());

}

3. **getMessage()**

This will print only the description of the exception

int a=5;

int b=0;

**try**{

System.out.println(a/b);

}

**catch**(ArithmeticException e){

System.out.println(e.getMessage());

}

# How to handle Exceptions?

Java exception handling is managed via five keywords: **try**, **catch**, [**throw**](https://www.geeksforgeeks.org/throw-throws-java/), [**throws**](https://www.geeksforgeeks.org/throw-throws-java/), and **finally**.

**try**{

…………

}

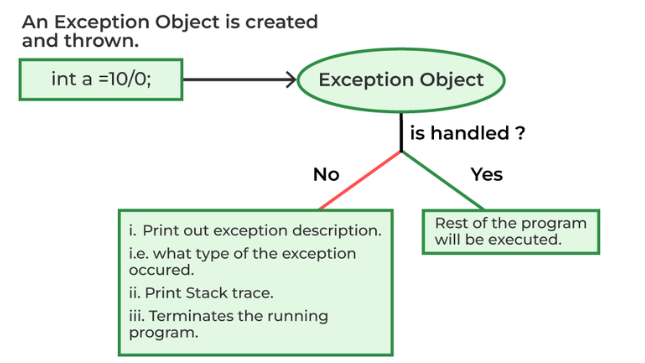
**catch**(Exception ex){

…………

}

**finally**{

}



# Throws and 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](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/). The throw keyword is mainly used to throw custom exceptions.

**static** **void** fun()

    {

**try** {

**throw** **new** NullPointerException("demo");

        }

**catch** (NullPointerException e) {

            System.out.println("Caught inside fun().");

**throw** e; // rethrowing the exception

        }

    }

**public** **static** **void** main(String args[])

    {

**try** {

            fun();

        }

**catch** (NullPointerException e) {

            System.out.println("Caught in main.");

        }

    }

Example 2:

**public** **static** **void** main(String[] args)

    {

        System.out.println(1 / 0);

    }

## Throws keyword

* **throws** is a keyword in Java that is used in the signature of a 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.
* **throws** keyword is required only for checked exceptions and usage of the throws keyword for unchecked exceptions is meaningless.
* **throws** keyword is required only to tell the compiler and usage of the throws keyword does not prevent abnormal termination of the program.
* With the help of the **throws** keyword, we can provide information to the caller of the method about the exception.

Example thread.sleep(5000);

Here we will get compile time error because there is a chance of exception if the main thread is going to sleep, other threads get the chance to execute the main() method which will cause InterruptedException.

# try with resources

* Resource is an object that must be closed after the program is executed.
* The try-with-resources statement ensures that each resource is closed at the end of the statement execution.

**public** **static** **void** main(String args[]) **throws** FileNotFoundException{

**try** (PrintWriter writer = **new** PrintWriter(**new** File("D:\\Training\\test.txt"))) {

writer.println("Hello World");

}

}

### With try—catch—finally and with try-resources

**With Try-catch-finally**

Scanner scanner = **null**;

**try** {

scanner = **new** Scanner(**new** File("test.txt"));

**while** (scanner.hasNext()) {

System.***out***.println(scanner.nextLine());

}

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **finally** {

**if** (scanner != **null**) {

scanner.close();

}

}

**With try – resources**

try (Scanner scanner = new Scanner(new File("test.txt"))) {

while (scanner.hasNext()) {

System.*out*.println(scanner.nextLine());

}

} catch (FileNotFoundException fnfe) {

fnfe.printStackTrace();

}

## Trying with multiple resources

**try** (Scanner scanner = **new** Scanner(**new** File("testRead.txt"));

PrintWriter writer = **new** PrintWriter(**new** File("testWrite.txt"))) {

**while** (scanner.hasNext()) {

writer.print(scanner.nextLine());

}

}

# Creating custom exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, the user can also create exceptions which are called ‘user-defined Exceptions’.

User created exception class should extend the Java **Exception** class.

Creating a Java program which throws a custom exception if the account balance is less than 1000.

**class** MyException **extends** Exception

{

    //store account information

**private** **static** **int** accno[] = {1001, 1002, 1003, 1004, 1005};

**private** **static** String name[] =

                 {"Nita", "Shubh", "Ravi", "Abhi", "Akash"};

**private** **static** **double** bal[] =

         {10000.00, 12000.00, 5600.0, 999.00, 1100.55};

    // 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();

        }

    }

}

# Common exception classes and categories