

# Exception Handling in Java

## Exception

- Exception is an event that disrupts the normal flow of the program.
- Exception means any abnormal, unexpected events or extraordinary condition that may occur at runtime.
- When exception occurs in your program then java environment print some error message onto the screen and program gets terminate their only.

Let's take a scenario:

```
statement 1;  
statement 2;  
statement 3;  
statement 4;  
statement 5;//exception occurs  
statement 6;  
statement 7;  
statement 8;  
statement 9;  
statement 10;
```

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

## Default exception situation

- i. If you are trying to divide a number by zero.
- ii. If you declare the object and memory is not allocated for that object.
- iii. If you are trying to access the element of array which exceeds the array limit.
- iv. If you are accessing the file which is not on hard disk.
- v. If you are converting invalid String to a number. etc

## Some Java Built in Exception

### 1. **ArithmeticException**

It is caused by math errors such as division by zero.

### 2. **NullPointerException**

If object is declared but memory is not allocated.

### 3. **ArrayIndexOutOfBoundsException**

It is caused by bad array index.

### 4. **FileNotFoundException**

It is caused by an attempt to access a non existing file.

### 5. **IOException**

It is caused by general Input/Output failure, such as inability to read from file.

### 6. **NumberFormatException**

It is caused when a conversion between String and number fails.

**etc.**

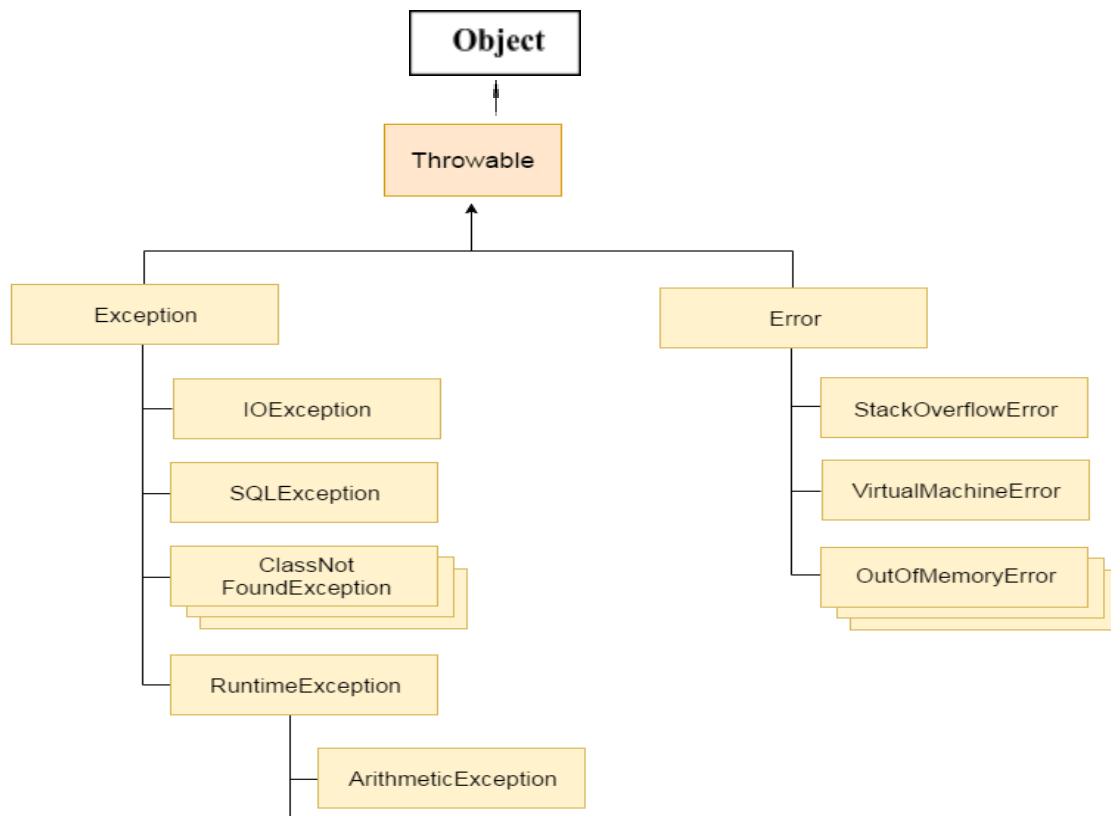
## Exception Handling

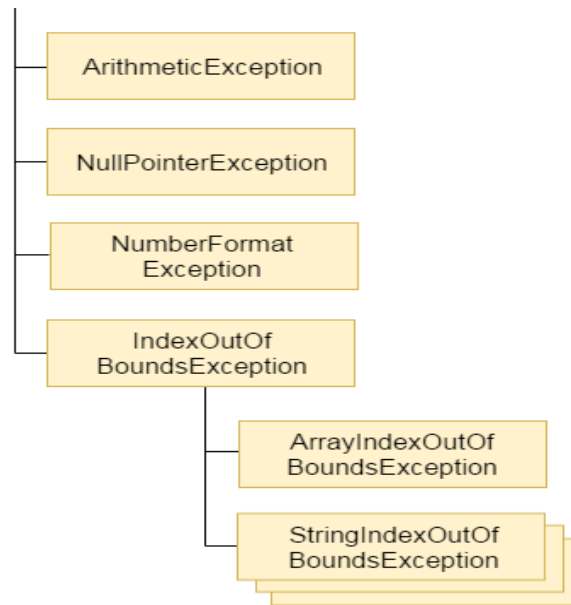
- Exception handling means if exception occur in your program, handle that situation and continue the program execution by taking some necessary action.
- Exception is a predefined class available in “**java.lang**” package.

## Hierarchy of Java Exception classes

Object class is the parent class of all the classes in java by default. In other words, it is the topmost class in java.

Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:





## Types of Java Exceptions

There are mainly two types of exceptions: **checked** and **unchecked**. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

### 1. Checked Exception

- An exception that is checked by the compiler at the compilation time is called checked exception.
- These exceptions cannot be simply be ignored, the programmer should handle these exception
- IOException, SQLException, IllegalStateException etc are checked exception.
- Checked exception forces program to deal with the exception that may be throws or with the help of try – catch block.

### 2. Unchecked Exception

- An exception that occurs at the time of execution is called unchecked exception.
- Unchecked exceptions are also called Runtime exception.
- Runtime exceptions are ignored at the time of compilation.
- ArrayIndexOutOfBoundsException, NullPointerException and so on are all subclasses of the java.lang.RuntimeException class, which is subclass of Exception class.
- Unchecked exceptions are basically built in exception in java.

## Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

1. try	In try block you have to write those statements in which there is a possibility of exception. The try block must be followed by either catch or finally. It means, we can't use try block alone.
2. 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.
3. finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
4. throw	The "throw" keyword is used to throw an exception.
5. throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

**Example of Java Exception Handling where we using a try-catch statement to handle the exception.**

**Example of “ArithmeticException” in java**

(i)

```
import java.util.*;
class Test
{
    public static void main(String args[])
    {

        System.out.println(3/0);

        System.out.println("Thank you.");
    }
}
```

(ii)

```
import java.util.*;
class Test
{
    public static void main(String args[])
    {
        try
        {
            System.out.println(3/0);
        }
    }
}
```

```

        catch(ArithmeticException ex)
        {
            System.out.println("Exception:\t"+ex.getMessage());
        }

        System.out.println("Thank you.");
    }
}

```

(iii)

```

import java.util.*;
class Test
{
    public static void main(String args[])
    {
        int a,b,rs=0;
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter the First No:");
        a=scan.nextInt();
        System.out.println("Enter the Second No:");
        b=scan.nextInt();
        try
        {
            rs=a/b;
        }catch(ArithmeticException ex){ex.printStackTrace();}
        System.out.println("Result:\t"+rs);

    }
}

```

## **MULTIPLE CATCH STATEMENT**

### **Syntax:**

```

try{
    statement;
}

```

```

catch(Exception-Type1 ex)
{
    Statement;
}
catch(Exception-Type2 ex)
{
    Statement;
}
catch(Exception-Type3 ex)
{
    Statement;}

```

Eg:

```

import java.util.*;
class Test
{
    public static void main(String args[])
    {
        int a,b,rs=0;
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter the First No:");
        a=scan.nextInt();
        System.out.println("Enter the Second No:");
        b=scan.nextInt();
        try
        {
            rs=a/b;
        }catch(ArithmeticException ex){ex.printStackTrace();}
        catch(ArrayIndexOutOfBoundsException ex){ex.printStackTrace();}
        catch(Exception ex){ex.printStackTrace();}

        System.out.println("Result:\t"+rs);

    }
}

```

### **Example of "ArrayIndexOutOfBoundsException" in java**

**(i)**

```
import java.util.*;
class Test
{
    public static void main(String args[])
    {
        int a[]={5,10};
        int b=5;
        int x=0;
        try
        {
            x=a[5];
        }

        catch(ArrayIndexOutOfBoundsException ex)
        {
            //System.out.println("Array Index Error");
            ex.printStackTrace();
        }
        System.out.println("X=\t"+x);
    } //main close
} //class close
```

**(ii)**

```
import java.util.*;
class Test
{
    public static void main(String args[])
    {
        int a[]={5,10};
        int b=5;
        int x=0;
        try
```



```

    {

        x=a[2]/(a[1]-b);

    }
    catch(ArithmeticException ex)
    {
        System.out.println("Division by zero");
    }
    catch(ArrayIndexOutOfBoundsException ex)
    {
        System.out.println("Array Index Error");
    }

    //int y=a[1]/a[0];
    System.out.println("X=\t"+x);

} //main close
} //class close

```

(iii)

```

import java.util.*;
class Test
{
    public static void main(String args[])
    {

        int a[]={5,10};
        int b=5;
        int x;
        try
        {
            x=a[2]/a[1]-b;

        }
        catch(ArithmeticException ex)
        {
            System.out.println("Division by zero");
        }
        catch(ArrayIndexOutOfBoundsException ex)
        {
            System.out.println("Array Index Error");
        }

        System.out.println();
        System.out.println("X=\t"+x);

    }

} //main close
} //class close

```

## Example of "NullPointerException" in java

(i)

```
class Stud
{
    int rno;
    String name;

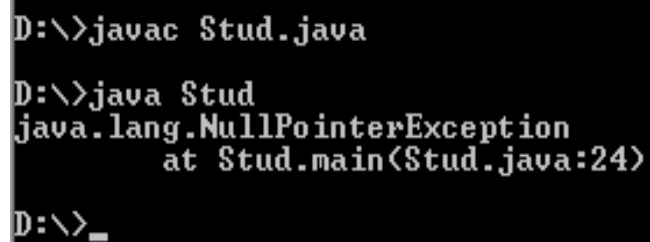
    void set_data(int r,String n)
    {
        rno=r;
        name=n;
    }

    void display()
    {
        System.out.println(rno+"\t"+name);
    }

    public static void main(String args[])
    {

        try
        {
            Stud s=null;
            s.set_data(1001,"manjeet");
            s.display();
        }
        catch(NullPointerException ex){ex.printStackTrace();}

    }
}
```



```
D:\>javac Stud.java
D:\>java Stud
java.lang.NullPointerException
    at Stud.main(Stud.java:24)
D:\>_
```

(ii)

```
class Stud
{
    int rno;
    String name;
```

```
void set_data(int r,String n)
{
    rno=r;
    name=n;
}

void display()
{
    System.out.println(rno+"\t"+name);
}

public static void main(String args[])
{
    try
    {
        Stud s=new Stud();
        s.set_data(1001,"manjeet");
        s.display();
    }
    catch(NullPointerException ex){ex.printStackTrace();}

}

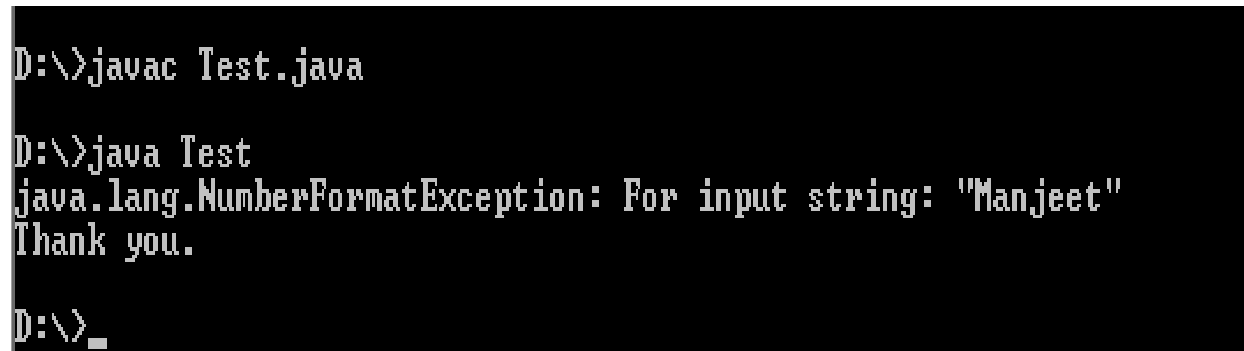
}
```

### Example of "NumberFormatException" in java

```
import java.util.*;
class Test
{
    public static void main(String args[])
    {
        int num;
        try
        {
            num=Integer.parseInt("Manjeet");

        }
        catch(NumberFormatException ex)
        {
            System.out.println(ex);
        }

        System.out.println("Thank you.");
    }
} //main close
} //class close
```



```
D:\>javac Test.java

D:\>java Test
java.lang.NumberFormatException: For input string: "Manjeet"
Thank you.

D:\>_
```

(ii)

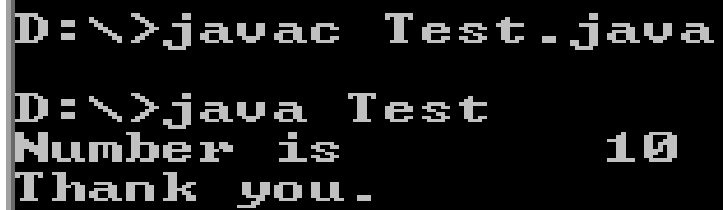
```
import java.util.*;
class Test
{
    public static void main(String args[])
    {
        int num=0;
        try
        {
```

```
        num=Integer.parseInt("10");

    }
    catch(NumberFormatException ex)
    {
        System.out.println(ex);
    }

    System.out.println("Number is\t"+num);
    System.out.println("Thank you.");

} //main close
} //class close
```

A screenshot of a Windows command prompt window with a black background and white text. It shows the compilation of a Java file named 'Test.java' using 'javac' and its execution using 'java Test'. The output of the program is displayed on the next two lines.

```
D:\>javac Test.java
D:\>java Test
Number is      10
Thank you.
```

## Throw in java

- In java, throw keyword is used to explicitly throw an exception.
- We can throw unchecked exception in java by throw keyword.
- The throw keyword is mainly used to throw **custom exception/user defined** exception.

### Syntax:

```
throw exception;
```

### Eg:

```
throw new ArithmeticException(" divide by zero ");
```

### Example:

```
import java.util.Scanner;

class TestThrow
{
    public static void main(String args[])
    {
        try{

            int age;
            Scanner scan=new Scanner(System.in);
            System.out.println("Enter the age:");
            age=scan.nextInt();
            if(age<18)
                throw new ArithmeticException("Invalid age");

            System.out.println("Your age is:\t"+age);

        }
        catch(ArithmeticException ex)
        {
            System.out.println("Exception:\t"+ex.getMessage());
        }
    }
}
```

}

## **Custom exception or User defined exception**

**Custom exception typically extend “Exception” class.**

**Q: Write a java program to create AgeException class, raise the exception when a user input age less than 18 or greater than 40.**

**Or**

**Write a java program to create Employee class with age as a attribute. If age is less than 18 or greater than 40 then throw AgeException.**

**Ans:**

```
class AgeException extends Exception
{
    public void disp()
    {
        System.out.println("Invalid Age.");
    }
}

class Employee
{
    int age;

    void set_data(int a)
    {
        age=a;
    }

    void display()
    {
        try{
            if(age<18||age>40)
                throw new AgeException();

            System.out.println(age);
        }
        catch(AgeException ex)
        {
            ex.disp();
        }
    }

    public static void main(String args[])
    {
    }
```

```

        {
            Employee e=new Employee();
            e.set_data(80);
            e.display();
        }
    }

```

**Eg:**

```

class DateException extends Exception
{
    public void disp()
    {
        System.out.println("Please enter valid Date.");
    }
}

class Employee
{
    int eid;
    String name,dateOfBirth;
    String ss[];

    void set_data(int id,String n,String dob)
    {
        eid=id;
        name=n;
        dateOfBirth=dob;
        ss=dateOfBirth.split("/");
    }

    void display()
    {
        try{
            if((Integer.parseInt(ss[0])<1) || (Integer.parseInt(ss[0])>31))
                throw new DateException();
            else if((Integer.parseInt(ss[1])<1) || (Integer.parseInt(ss[1])>12))
                throw new DateException();
            else if((Integer.parseInt(ss[2])<=2004))
                throw new DateException();

            System.out.println(eid+"\t"+name+"\t"+dateOfBirth);
        }
        catch(DateException ex)

```



```

    {
        ex.disp();
    }

    public static void main(String args[])
    {
        Employee e=new Employee();
        e.set_data(1001,"Anu","12/01/201");
        e.display();
    }
}

```

### Throws Keyword

If any method is able to raise or generate an exception, but if it is not handled there and if you want to handle that exception in calling method then in-front of method name mention the possible exception list by using “**throws**” keyword.

Eg:

```

class AgeException extends Exception
{
    public void disp()
    {
        System.out.println("Invalid Age.");
    }
}

class Employee
{
    int age;

    void set_data(int a)
    {
        age=a;
    }

    void display() throws AgeException
    {

```

```

        if(age<18||age>60)
            throw new AgeException();

        System.out.println(age);

    }

    public static void main(String args[])
    {

        try{
            Employee e=new Employee();
            e.set_data(50);
            e.display();
        }
        catch(AgeException ex)
        {
            ex.disp();
        }

    }

}

```

### Difference between Throw and Throws

	<b>Throw</b>	<b>Throws</b>
1.	Used to explicitly throw an exception.	Used to declare an exception.
2.	Checked exceptions cannot be propagated using throw.	Checked exceptions can be propagated using throws
3.	Followed by an instance.	Followed by a class.
4.	Used within a method.	Used with a method signature.
5.	Cannot throw multiple exception.	Can declare multiple exception.

## Finally keyword

In finally block you can write those statements which you want to execute in both situation, exception generated or exception not generated. Finally block should be immediately after the try block or catch block.

```
import java.util.Scanner;
class Test1
{
    public static void main(String args[])
    {
        int a,b,rs=0;
        Scanner scan=new Scanner(System.in);
        try
        {
            a=scan.nextInt();
            b=scan.nextInt();
            rs=a/b;

        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            System.out.println("Result:"+rs);
        }
    }
};
```



```
import java.util.Scanner;
class Test1
{
public static void main(String args[])
{
int a,b,rs=0;
Scanner scan=new Scanner(System.in);
try
{
a=scan.nextInt();
b=scan.nextInt();
rs=a/b;
}
finally
{
System.out.println("Result:"+rs);
}

}

};
```