



# CORE JAVA

By : Kalpesh Chauhan



# EXCEPTION HANDLING IN JAVA



The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

# WHAT IS EXCEPTION IN JAVA

**Dictionary Meaning:** Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

# WHAT IS EXCEPTION HANDLING

Exception Handling is a mechanism to handle runtime errors such as `ClassNotFoundException`, `IOException`, `SQLException`, `RemoteException`, etc.

# ADVANTAGE OF EXCEPTION HANDLING

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

statement 1;


statement 2;

statement 3; **//exception occurs**

statement 4;

statement 5;

statement 6;



Suppose there are 6 statements in your program and there occurs an exception at statement 3, the rest of the code will not be executed i.e. statement 4 to 6 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.

# HIERARCHY OF JAVA EXCEPTION CLASSES

The `java.lang.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:



# Throwable

- Exception
  - IO
  - SQL
  - Class not Found
  - Runtime
    - Arithmetic
    - Class not found
    - Array index out of Bound
- Error
  - Stack overflow error
  - Virtual machine error
  - Out of memory error

# 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:

# DIFFERENCE BETWEEN CHECKED AND UNCHECKED EXCEPTIONS

## Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

## Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

## Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.



# JAVA EXCEPTION KEYWORDS

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

Keyword	Description
<b>try</b>	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
<b>catch</b>	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.
<b>finally</b>	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
<b>throw</b>	The "throw" keyword is used to throw an exception.
<b>throws</b>	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.

# COMMON SCENARIOS OF JAVA EXCEPTIONS

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

A scenario where `ArithmeticException` occurs

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

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



A scenario where NullPointerException occurs

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
```



A scenario where `NumberFormatException` occurs

The wrong formatting of any value may occur `NumberFormatException`. Suppose I have a string variable that has characters, converting this variable into digit will occur `NumberFormatException`.

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

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



A scenario where `ArrayIndexOutOfBoundsException` occurs

If you are inserting any value in the wrong index, it would result in `ArrayIndexOutOfBoundsException` as shown below:

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

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


# JAVA TRY-CATCH

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

## *Syntax of Java try-catch*

```
Try {  
    //code that may throw exception  
}catch (Exception_class_Name ref) {  
  
}
```



*Syntax of try-finally block*

**try {**

//code that may throw exception

**} finally {**

**}**

# JAVA CATCH BLOCK

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

# PROBLEM WITHOUT EXCEPTION HANDLING

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1 {  
    public static void main(String args[]){  
        int data=50/0;//may throw exception  
        System.out.println("rest of the code...");  
    }  
}
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

# SOLUTION BY EXCEPTION HANDLING

Let's see the solution of above problem by java try-catch block.

```
public class Testtrycatch2
    public static void main(String args[]){
        try{
            int data=50/0;
        }catch(ArithmeticException e){
            System.out.println(e);
        }
        System.out.println("rest of the code...");
    }
}
```



The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

1. Prints out exception description.
2. Prints the stack trace (Hierarchy of methods where the exception occurred).
3. Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

# JAVA MULTI CATCH BLOCK

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block. Let's see a simple example of java multi-catch block.

***Rule: At a time only one Exception is occurred and at a time only one catch block is executed.***

***Rule: All catch blocks must be ordered from most specific to most general i.e. catch for `ArithmeticException` must come before catch for `Exception` .***



# JAVA NESTED TRY BLOCK

The try block within a try block is known as nested try block in java.

## **Why use nested try block**

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.

# JAVA FINALLY BLOCK

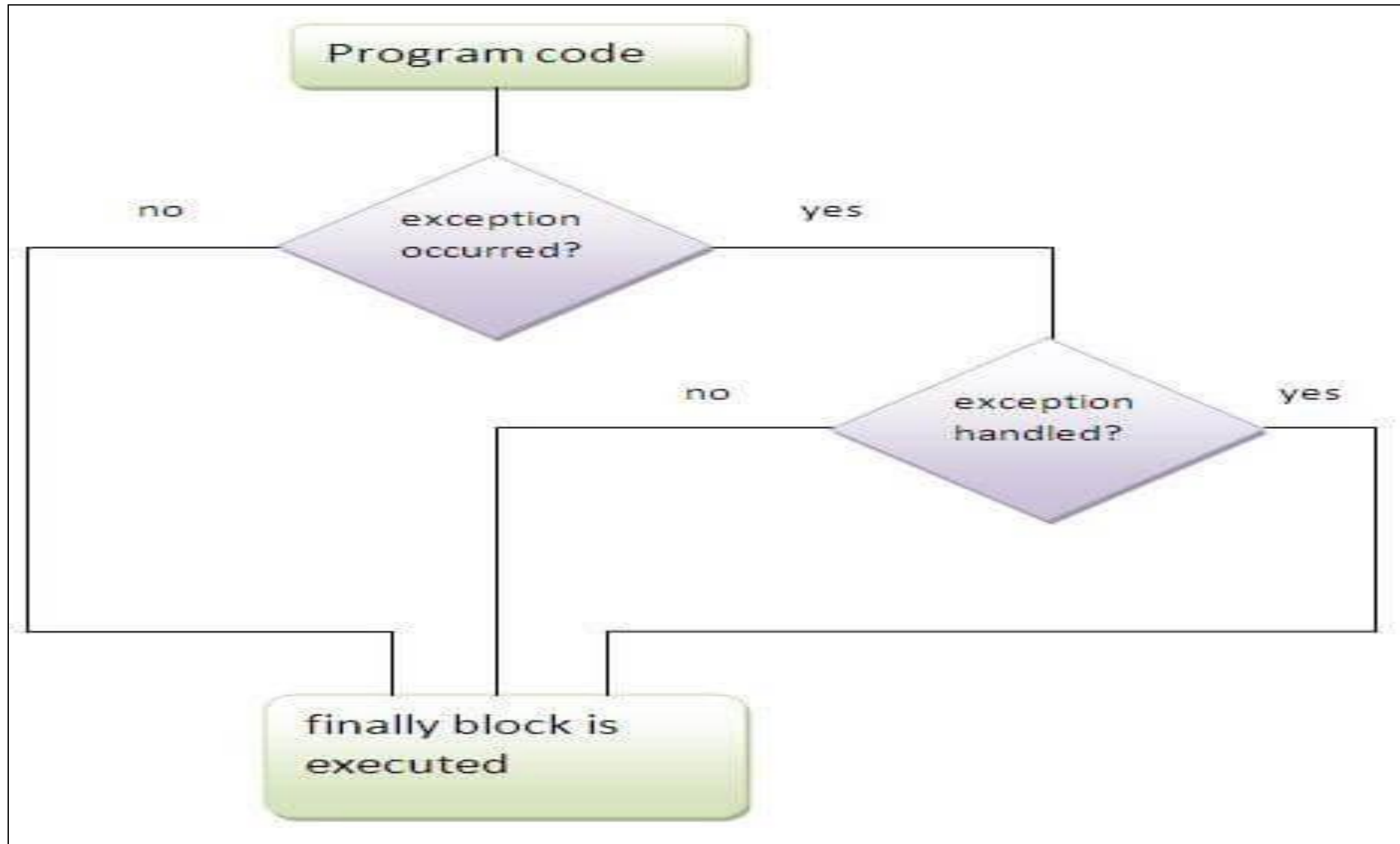
**Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

## **Why use java finally**

Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.



# JAVA THROW EXCEPTION

Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

**throw** exception;

Let's see the example of throw IOException.

**throw new** IOException("sorry device error);

# JAVA EXCEPTION PROPAGATION

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

***Rule: By default Unchecked Exceptions are forwarded in calling chain (propagated).***

```
class TestExceptionPropagation1
{
    void m(){ int data=50/0; }
    void n(){ m(); }
    void p(){
        try{ n(); }
        catch(Exception e){
            System.out.println("exception handled");}
    }

    public static void main(String args[]){
        TestExceptionPropagation1 obj=new TestExceptionPropagation1 ();
        obj.p();
        System.out.println("normal flow..."); }
}
```

# JAVA 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 normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as `NullPointerException`, it is programmers fault that he is not performing check up before the code being used.

# SYNTAX OF JAVA THROWS

```
return_type method_name() throws exception_class_name  
{  
    //method code  
}
```



# WHICH EXCEPTION SHOULD BE DECLARED

**Ans)** checked exception only, because:

**unchecked Exception:** under your control so correct your code.

**error:** beyond your control e.g. you are unable to do anything if there occurs `VirtualMachineError` or `StackOverflowError`.

# ADVANTAGE OF JAVA THROWS KEYWORD

Now Checked Exception can be propagated (forwarded in call stack).  
It provides information to the caller of the method about the exception.

# JAVA THROWS EXAMPLE

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

**Rule: If you are calling a method that declares an exception, you must either caught or declare the exception.**



There are two cases:

**Case1:** You caught the exception i.e. handle the exception using try/catch.

**Case2:** You declare the exception i.e. specifying throws with the method.

# CASE1: YOU HANDLE THE EXCEPTION

In case you handle the exception, the code will be executed fine whether exception occurs during the program or not.

```
import java.io.*;
class M{
    void method()throws IOException{
        throw new IOException("device error");
    }
}

public class Testthrows2{
    public static void main(String args[]){
        try{
            M m=new M();
            m.method();
        }
        catch(Exception e){
            System.out.println("exception handled");
        }
        System.out.println("normal flow...");
    }
}
```

## CASE2: YOU DECLARE THE EXCEPTION

In case you declare the exception, if exception does not occur, the code will be executed fine.

In case you declare the exception if exception occurs, an exception will be thrown at runtime because throws does not handle the exception.

```
import java.io.*;

class M{
    void method()throws IOException{
        System.out.println("device operation performed");
    }
}

class Testthrows3{
    public static void main(String args[])throws IOException{//declare exception
        M m=new M();
        m.method();
        System.out.println("normal flow...");
    }
}
```



# DIFFERENCE BETWEEN THROW AND THROWS IN JAVA

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

# DIFFERENCE BETWEEN FINAL, FINALLY AND FINALIZE

There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

# JAVA FINAL EXAMPLE

```
class FinalExample
{
    public static void main(String[] args){
        final int x=100;
        x=200;//Compile Time Error
    }
}
```

# JAVA FINALLY EXAMPLE

```
class FinallyExample{
    public static void main(String[] args){
        try{
            int x=300;
        }catch(Exception e){
            System.out.println(e);
        }
        finally{
            System.out.println("finally block is executed");
        }
    }
}
```

# JAVA FINALIZE EXAMPLE

```
class FinalizeExample{  
    public void finalize(){  
        System.out.println("finalize called");  
    }  
    public static void main(String[] args){  
        FinalizeExample f1=new FinalizeExample();  
        FinalizeExample f2=new FinalizeExample();  
  
        f1=null;  
        f2=null;  
        System.gc();  
    }  
}
```

# JAVA CUSTOM EXCEPTION

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

```
class InvalidAgeException extends Exception {  
    InvalidAgeException(String s){  
        super(s);  
    }  
}
```

```
class TestCustomException1 {  
    static void validate(int age) throws InvalidAgeException {  
        if (age < 18)  
            throw new InvalidAgeException("not valid");  
        else  
            System.out.println("welcome to vote");  
    }  
  
    public static void main(String args[]) {  
        try {  
            validate(13);  
        } catch (Exception m) {  
            System.out.println("Exception occurred: " + m);  
        }  
        System.out.println("rest of the code...");  
    }  
}
```



**CONTINUE IN NEXT UNIT.....**