**OBJECT ORIENTED PROGRAMMING**

***Department of Software Engineering***

###### Public Relations, MUET, SZAB Campus Khairpur Mir's - College & University | Facebook - 9,847 PhotosMentor:

Engr. Abdul Hafeez Babar [hafeez.babar@muetkhp.edu.pk](mailto:hafeez.babar@muetkhp.edu.pk)

***Mehran University of Engineering and Technology, SZAB Campus Khairpur Mir’s***

# Exception Handling



##### Exception Handling



* Exception handling in java is a powerful mechanism or technique that allows us to handle runtime errors in a program so that the normal flow of the program can be maintained.
* In general, an exception means a problem or an abnormal condition that stops a computer program from processing information in a normal way.
* An exception in java is an object representing an error or an abnormal condition that occurs at runtime

execution and interrupts (disrupts) the normal execution flow of the program.

* In other words, unwanted and unexpected behavior/event that interrupts the normal execution flow of the program is called exception in java. It is thrown from a method. The caller of the method can catch and handle the exception.
* An exception can be identified only at runtime, not at compile time. Therefore, it is also called runtime errors that are thrown as exceptions in Java. They occur while a program is running.



Example:

if we access an array using an index that is out of bounds, we will get a runtime error named ArrayIndexOutOfBoundsException.

If we enter a double value while the program expecting an integer value, we will get a runtime error called

InputMismatchException.

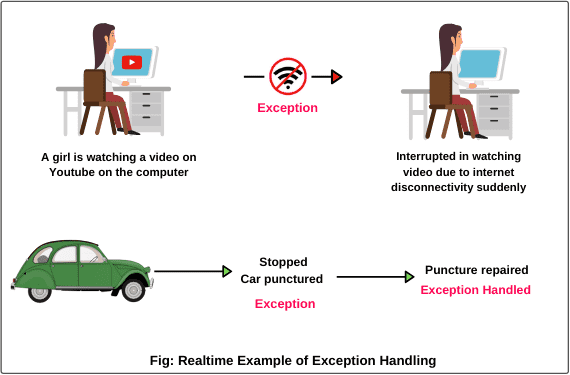
If the exception object is not caught and handled properly, JVM will display an error message and will terminate the rest of the program abnormally.

If we want to continue the execution of remaining code in the program, we will have to handle exception object thrown by error condition and then display a user-friendly message for taking corrective actions. This task is known as exception handling in java.



##### Realtime Example of Exception





###### Example:



import java.util.Scanner; public class Test

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter two integer numbers");

// Read two integer numbers. int num1 = sc.nextInt();

int num2 = sc.nextInt();

System.out.println(num1 + "/" + num2 + " = " + (num1/num2));

}

Output 1:

Enter two integer numbers 4

2

4/2 = 2

Output 2:

Enter two integer numbers 2

0

Exception in thread "main"

java.lang.ArithmeticException: / by zero

}





###### Why Exception occurs in Program?



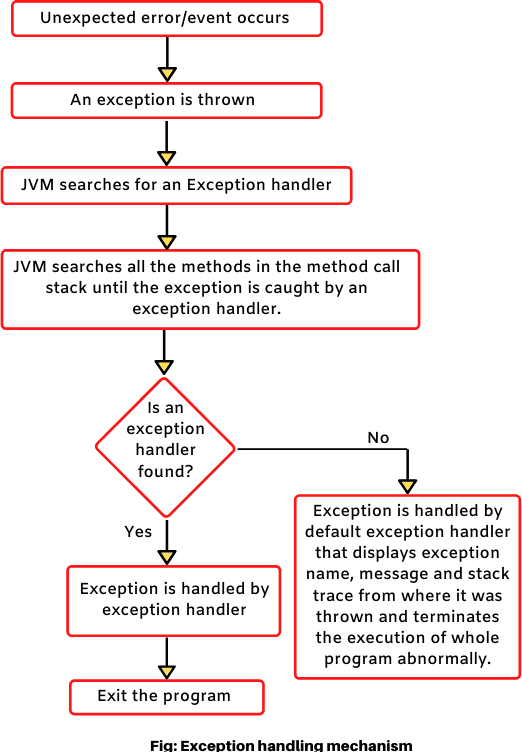
* 1. Opening a non-existing file in your program.
  2. Reading a file from a disk but the file does exist there.
  3. Writing data to a disk but the disk is full or unformatted.
  4. When the program asks for user input and the user enters invalid data.
  5. When a user attempts to divide an integer value by zero, an exception occurs.
  6. When a data stream is in an invalid format, etc.

##### What is Exception Handler in Java?



* The code that catches the exception thrown by JVM is called exception handler in Java. It is responsible for receiving information about the exception/error.
* When an exception occurs, exception handling transfers the control of execution of the program to an appropriate exception handler.
* For example, suppose we call a method that opens a file but the file does not open. In this case, the execution of that method will stop and the code that we wrote to handle with this situation, will be run.





###### Example:

package exceptionHandling;

public class ExceptionWithoutError

{

public static void main(String[] args)

{

System.out.println("One"); System.out.println("Two"); System.out.println("Three"); System.out.println("Four");

}

}

Output:

One Two

Output:

One

Two

###### Example:

package exceptionHandling;



public class ExceptionDivideByZero

{

public static void main(String[] args)

{

System.out.println("One"); System.out.println("Two");

int a = 1/0; // Exceptional case.

System.out.println("Three");

System.out.println("Four");

}

}

Three

Four

Exception in thread "main" java.lang.ArithmeticException: / by zero at exceptionHandling.ExceptionDivideByZero.main(ExceptionDivideByZero.java:9)

###### Example

package exceptionHandling;

public class ExceptionHandlingwithTryCatch

{

public static void main(String[] args)

{

System.out.println("One"); System.out.println("Two");

// Declare a try-catch block.

try // Error handling code starts here.

{

System.out.println("Before divide");

int a = 1/0; // Exceptional case (Exception has occurred). System.out.println("After divide");

}

Output:



One Two

Before divide

A number cannot be divided by zero.

Three Four

catch(ArithmeticException e) // Exception handled. Here, catch block is exception handler.

{

System.out.println("A number cannot be divided by zero."); // User- friendly message

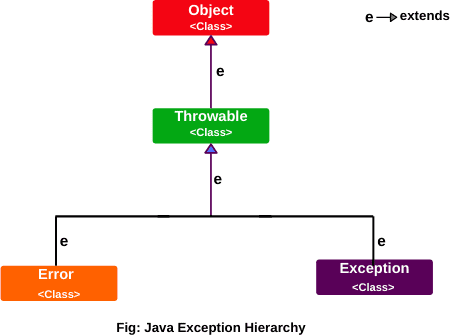
}

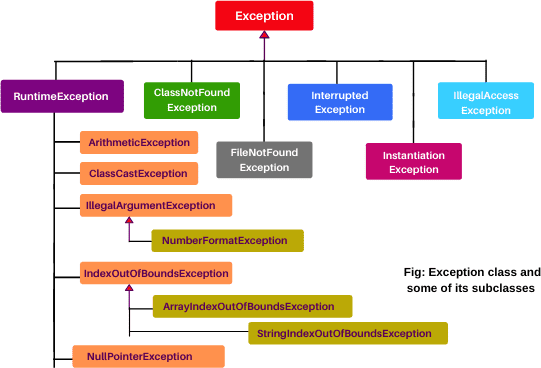
System.out.println("Three"); System.out.println("Four");

}

}

**Exception Hierarchy in Java**

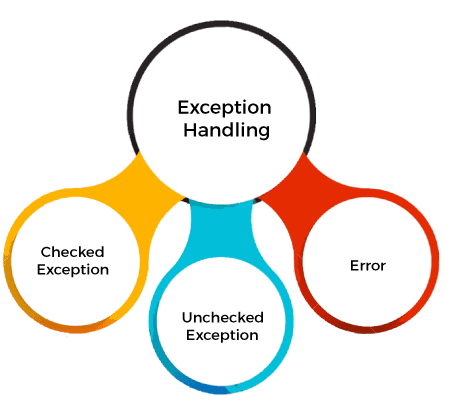




**Types of Java Exceptions**

##### Types of Java Exceptions



* + There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. 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.

1. 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.
2. Error
   * Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

##### Checked Exceptions in Java



* Checked exceptions are those exceptions that are checked by the java compiler itself at compilation time and are not under runtime exception class hierarchy. If a method throws a checked exception in a program, the method must either handle the exception or pass it to a caller method.
* Checked exceptions must be handled either by using try and catch block or by using throws clause in the method declaration. If not handles properly, it will give a compile- time error.

*List of some important checked exceptions*

* + ClassNotFoundException
  + InterruptedException
  + InstantiationException
  + IOException
  + SQLException
  + IllegalAccessException
  + FileNotFoundException, etc

##### Unchecked Exceptions (Runtime Exceptions) in Java



* Unchecked exceptions in Java are those exceptions that are checked by JVM, not by

java compiler. They occur during the runtime of a program. All exceptions under runtime exception class are called unchecked exceptions or runtime exceptions in Java.

* We can write a Java program and compile it. But we cannot see the effect of unchecked exceptions and errors until we run the program. This is because Java compiler allows us to write a Java program without handling unchecked exceptions and errors.
* Java compiler does not check runtime exception at compile time whether programmer handles them or not. If a runtime exception occurs in a method and programmer does not handle it, JVM terminates the program without the execution of rest of the code.

**Some important examples of runtime exceptions**

* ArithmeticException
* ClassCastException
* NullPointerException
* ArrayIndexOutOfBoundsException
* NegativeArraySizeException
* ArrayStoreException
* IllegalThreadStateException
* SecurityException, etc.



#### Java Try Catch Block

##### Java provides five essential keywords to handle an exception.



* + try
  + **catch**
  + **finally**
  + **throw**
  + **throws**

**Try Block**



* A keyword “try” is a block of code or statements that might throw an exception. That’s why a try block is also known as exception generated block.
* The Java code that may generate an exception during the execution of program, must be placed within a try block.
* That is, we should place exception generated code (risky code) inside try block. We should not keep normal code inside try block.

**Example**

* + Suppose there are three statements inside try block. First statement may occur exception and is on top inside try block. The other two statements are normal and are below the first statement.
  + If exception occurred in statement 1, other two normal statements will not be executed. Therefore, the length of code inside the try block should be as much as less.
    - **try-catch:** A try block is always followed by one or more catch blocks.
    - **try-finally:** A try block followed by a finally block.
    - **try-catch-finally:** A try block followed by one or more catch blocks followed by a finally block.
* A keyword “catch” is a block of code that handles the exception thrown by the try block. That’s why it is also known as exception handler block. A catch block that catches an exception, must be followed by try block that generates an exception.

*General syntax of try-catch block (exception handling block)*

**Syntax:**

try

{

// A block of code; // generates an exception

}

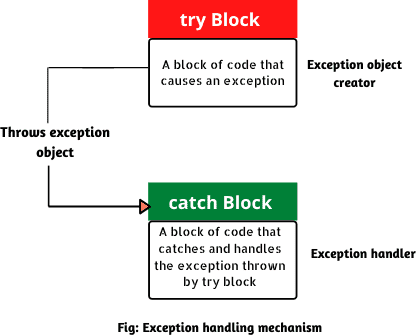
catch(exception\_class var)

{

// Code to be executed when an exception is thrown.

}





1. Java try-catch block must be within a method.
2. A try block can not be used without a catch or finally block. It must be followed by at least one catch block otherwise, the compilation error will occur.
3. A catch block must be followed by try block. There should not be any statement between the end of try

block and the beginning of catch block.

1. A finally block cannot come before catch block.

###### Example:

try

{

statement 1;

statement 2;

statement 3;

}

catch(exception\_class var)

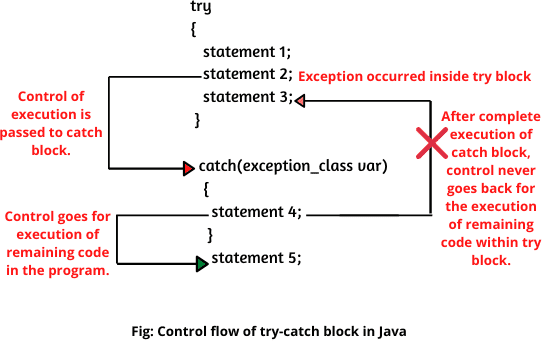
{

statement 4;

}

statement 5;





##### Example: Java Exception Handling

public class TryCatchEx

{

public static void main(String[] args)

{

System.out.println("11"); System.out.println("Before divide"); int x = 1/0; System.out.println("After divide"); System.out.println("22");

}

}

public class TryCatchEx1

{

public static void main(String[] args)

{

System.out.println("11"); System.out.println("Before divide"); try

{

int x = 1/0; System.out.println("After divide");

}

Output:



11

Before divide

A number cannot be divided by zero

22

Output:

11

Before divide

Exception in thread "main" java.lang.ArithmeticException: / by zero

catch(ArithmeticException ae) // Here, ae is a reference variable of exception object.

{

System.out.println("A number cannot be divided by zero");

}

System.out.println("22");

}

}

public class TryCatchEx2

{

public static void main(String[] args)

{

System.out.println("111"); try

{

int x = 12/0; System.out.println("Result of x: " +x); System.out.println("333");

}

catch(ArithmeticException ae)

{

System.out.println("Hello world");

}

System.out.println("444");

}

}

public class TryCatchEx3

{

public static void main(String[] args)

{

int x = 100, y = 0; try

{

System.out.println("111");

int z = x/y; System.out.println("Result of z: " +z);

}

catch(ArithmeticException ae)

{

System.out.println("Hello Java");

}

System.out.println("333");

}

}

Example

public class TryCatchEx4

{



int x = 30, y = 0;

void divide(){

System.out.println("I am in method"); try {

System.out.println("I am in try block"); int z = x/y;

System.out.println("Result of z: " +z);

}

catch(NullPointerException np)

{

System.out.println("I am in catch block");

}

}

public static void main(String[] args)

{

TryCatchEx4 obj = new TryCatchEx4(); System.out.println("I am in main method"); obj.divide();

}

}

**Example**

public class TryCatchEx7



{

public static void main(String[] args)

{

try

{

int a[] = {20, 30, 40, 50};

a[10] = 5;

}

catch(ArrayIndexOutOfBoundsException a)

{

System.out.println("Array Index Out Of Bounds Exception");

}

}

}

Output:

Array Index Out Of Bounds Exception

Example



public class TryCatchEx9

{

public static void main(String[] args)

{

try {

String input = “Software Engineering"; int a = Integer.parseInt(input); System.out.println("Value of a: " +a);

}

catch(NumberFormatException n)

{

System.out.println(n.getMessage()+ " is not an integer.");

}

} }

Output:

For input string: " Software Engineering " is not an integer.

##### Multiple Catch Block in Java



* In Java, a single try block can have multiple catch blocks. When statements in a single try block generate multiple exceptions, we require multiple catch blocks to handle different types of exceptions. This mechanism is called multi-catch block in java.
* Each catch block is capable of catching a different exception. That is each catch block must contain a different exception handler.

Syntax:

try



{

statements;

}

catch(ExceptionType1 e1)

{

statements;

}

catch(ExceptionType2 e2)

{

statements;

}

catch(ExceptionType3 e3)

{

statements;

}

. . .

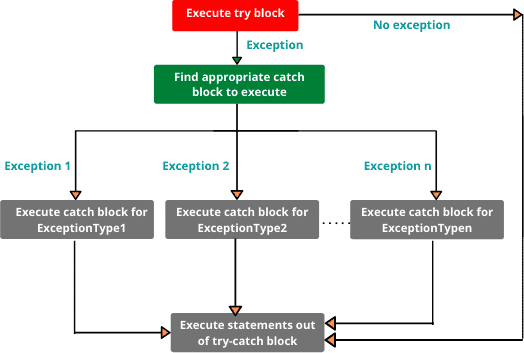
catch(ExceptionTypen en)

{

statements;

}





**Syntax**

try

{



statements;

}

catch(NullPointerException np)

{

statements;

}

catch(RuntimeException re)

{

statements;

}

catch(IOException ieo)

{

statements;

}

catch(Exception e)

{

statements;

}

**Example** public class MultiCatchEx1

{

public static void main(String[] args)

{

try

{

int arr[] = new int[6];

arr[3] = 20/0; // Exception occurred.

System.out.println("I am in try block");

}

catch(ArithmeticException ae)

{

Output:



A number cannot be divided by zero, Illegal operation in java I am out of try-catch block

System.out.println("A number cannot be divided by zero, Illegal operation in java");

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("Accessing array element outside of specified limit");

}

catch(Exception e)

{

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

}

System.out.println("I am out of try-catch block");

}

}



###### Example

public class MultiCatchEx2

{

public static void main(String[] args)

{

String s = “Software Engineering";

int a[] = {0, 1, 2, 3, 4, 5};

try

{

s = null;

int sLength = s.length(); System.out.println("String length: " +sLength); int b = 6;

System.out.println(a[b]);

}

catch(NullPointerException npe)

{

System.out.println("Exception is caught");

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

}

catch(ArrayIndexOutOfBoundsException aie)

{

System.out.println("Exception is caught"); System.out.println(aie.toString());

}

}

}

Output:

Exception is caught java.lang.NullPointerException

##### Unreachable Catch Block Error?



* When we are using multiple catch blocks for a single try block, the order of exceptions in the catch block must be placed in such a way that first, subclass exception comes and later on superclass exception.
* If we place superclass exception first and later on subclass exception, all the exceptions thrown for subclass exception will be caught by the first catch block.
* In this case, the second catch block will not be used in the program and java compiler will generate unreachable catch block error.

Example

package exceptionHandling; import java.util.Scanner; public class MultiCatchEx4

{

public static void main(String[] args)

{

int x, y;

Scanner sc = new Scanner(System.in);

try

{

System.out.println("Enter your first number"); x = Integer.parseInt(sc.nextLine());

System.out.println("Enter your second number"); y = Integer.parseInt(sc.nextLine());

int z = x / y; System.out.println("z = " +z);

}

catch(RuntimeException re)



{

System.out.println("Exception thrown: " +re);

}

catch(ArithmeticException ae) // Unreachable catch block error.

{

System.out.println("Exception thrown: " +ae);

}

catch(NumberFormatException nfe) // Unreachable catch block error.

{

System.out.println("Exception thrown: " +nfe);

}

}

}



#### Nested Try in Java

##### Nested Try in Java

* + When a try block is defined

Syntax:

try // Outer try block



{

within another try, it is called

nested try block in java.

* + The try block which encloses another try block is called outer try block and the enclosed try

block is called inner try block.

statements;

statements;

try // Inner try block

{

statements; statements;

}

catch(Exception1 e1) // Inner catch block

{

statements;

statements;

}

}

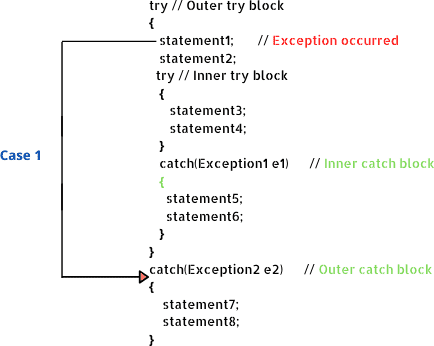
catch(Exception2 e2) // Outer catch block

{

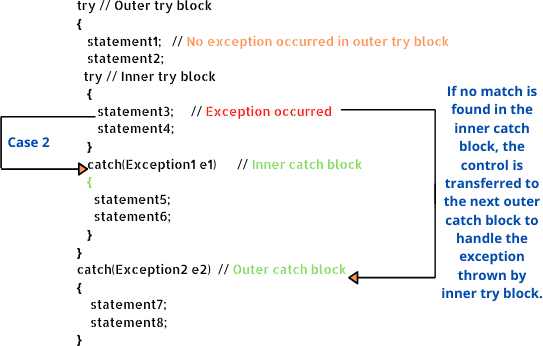
statements; statements;

}









Example

package nestedTryExample;

public class NestedTryBlockEx1



{

public static void main(String[] args)

{

String str = "Scientech Easy"; int x[ ] = {0, 1, 2, 3, 4};

try // Outer try block

{

str = null; // Exception occurred. int slength = str.length();

System.out.println("String length: " +slength);

try // Inner try block

{

int y = 6;

System.out.println(x[y]);

}

catch(ArrayIndexOutOfBoundsException aie)

{

System.out.println("Exception is thrown"); System.out.println(aie.toString());

}

}

catch(NullPointerException npe)

{

Output:

Exception is thrown java.lang.NullPointerException

System.out.println("Exception is thrown "); System.out.println(npe.toString());

}

}

Example



public class NestedTryBlockEx5 { public static void main(String[] args) { try {

int x[] = {0, 1, 2};

try {

int y[] = {0, 10};

int z = x[2]/y[0];

System.out.println("Division of two numbers: " +z);

}

catch(ArrayIndexOutOfBoundsException aie)

{

System.out.println("Inside inner try catch block"); System.out.println(aie.toString());

}

}

catch(ArithmeticException ae) // No match is found.

{

System.out.println("Inside outer try catch block "); System.out.println(ae.toString());

}

System.out.println("I am out of outer catch block");

}

}

Output:

Inside outer try catch block java.lang.ArithmeticException: / by zero I am out of outer catch block



**Finally Block in Java**

##### Finally Block in Java



* + - A “finally” is a keyword used to create a block of code that follows a try or catch block.
    - A finally block contains all the crucial codes such as closing connections, stream, etc that is always executed whether an exception occurs within a try block or not.
    - When finally block is attached with a try-catch block, it is always executed whether the catch block has

handled the exception thrown by try block or not.

##### Syntax: try-finally block

try

{

statement1; statement2;

}

finally // finally block

{

statement3;

}

##### Syntax: try-catch-finally block

try



{

statement1; statement2;

}

catch(Exceptiontype e1)

{

statement3;

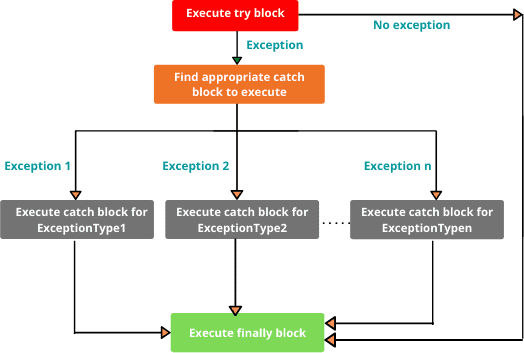
}

statement4; finally

{

statement5;

}



1. Generally, finally block or clause is used for freeing up resources, cleaning up code, db closing connection, io stream, etc.
2. A java finally block is used to prevent resource leak. While closing a file or recovering resources, the

code is put inside the finally block to ensure that the resource is always recovered.

1. Finally clause is used for terminating threads.

##### Example: try-catch-finally block



public class finallyBlockExample1

{

public static void main(String[] args)

{

int a = 20, b = 30;

try

{

int sum = a + b; System.out.println("Sum: " +sum);

}

catch(Exception e)

{

System.out.println(e);

}

finally

{

###### Output:

Sum: 50

finally block must be executed Hello Java

System.out.println("finally block must be executed");

}

System.out.println("Hello Java");

}

}

Example: exception will occur inside try block and it will be handled by catch block.



public class finallyBlockExample2

{

public static void main(String[] args)

{

int a = 20, b = 0;

try

{

System.out.println("Value of a: " +a); System.out.println("Value of b: " +b); int div = a/b; System.out.println("Division: " +div);

}

catch(ArithmeticException ae)

{

Output:

Value of a: 20 Value of b: 0

java.lang.ArithmeticException: / by zero Denominator cannot be zero

Hello Java

System.out.println(ae); // prints corresponding exception.

}

finally

{

System.out.println("Denominator cannot be zero");

}

System.out.println("Hello Java");

}

}

**Example** public class finallyBlockExample3

{

public static void main(String[] args)

{

int a = 20, b = 0; try

{

System.out.println("Value of a: " +a); System.out.println("Value of b: " +b); int div = a/b; System.out.println("Division: " +div);

}

catch(NullPointerException npe)

{

Output:



Value of a: 20 Value of b: 0

Exception in thread "main" Denominator cannot be zero java.lang.ArithmeticException: / by zero

at finallyBlockExample.finallyBlockExample1.main(finallyBlockExa mple1.java:11)

System.out.println(npe); // prints corresponding exception.

}

finally

{

System.out.println("Denominator cannot be zero");

}

System.out.println("Hello Java");

}

}

###### Example

public class finallyBlockExample4

{

public static void main(String[] args)

{

try

{

System.out.println("111"); System.out.println("222");

}

catch(Exception ae)

{

Output:



111

222

444

555

System.out.println(10/0);

}

finally

{

System.out.println("444");

}

System.out.println("555");

}

}

try

{

System.out.println("111"); System.out.println(20/0); System.out.println("222");

}

catch(Exception ae)

{

System.out.println(10/0);

}

finally

{

Output:

111

Exception in thread "main" 444 java.lang.ArithmeticException: / by zero

at finallyBlockExample.finallyBlockExample5.main(finallyBlockExa mple5.java:14)

System.out.println("444");

}

System.out.println("555");

}

}

try

{

System.out.println("111"); System.out.println(20/0); System.out.println("222");

}

catch(ArithmeticException ae)

{

Output:

111

Exception in thread "main" 333

java.lang.ArithmeticException: / by zero at

finallyBlockExample.finallyBlockExample6.main(finallyBlockExa mple6.java:18)

System.out.println("333");

}

finally

{

System.out.println(10/0); // Exception occurred in finally block.

}

System.out.println("555");

}

}

###### Example



public class finallyBlockExample7

{

public static void main(String[] args)

{

try

{

System.out.println("111"); System.out.println(20/0); System.out.println("222");

}

catch(NullPointerException npe)

{

System.out.println("333");

Output:

111

Exception in thread "main" java.lang.ArithmeticException: / by zero

at finallyBlockExample.finallyBlockExample7.main(finallyBlockExa mple7.java:18)

}

finally

{

System.out.println(10/0); // Exception occurred in finally block.

}

System.out.println("555");

}

} O

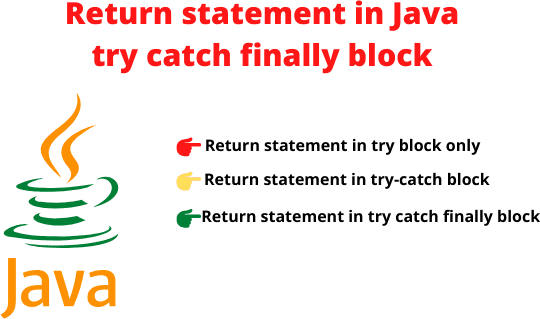


**Return statement in try catch finally block in Java**

##### Two famous questions arise in the topic “try catch finally block”

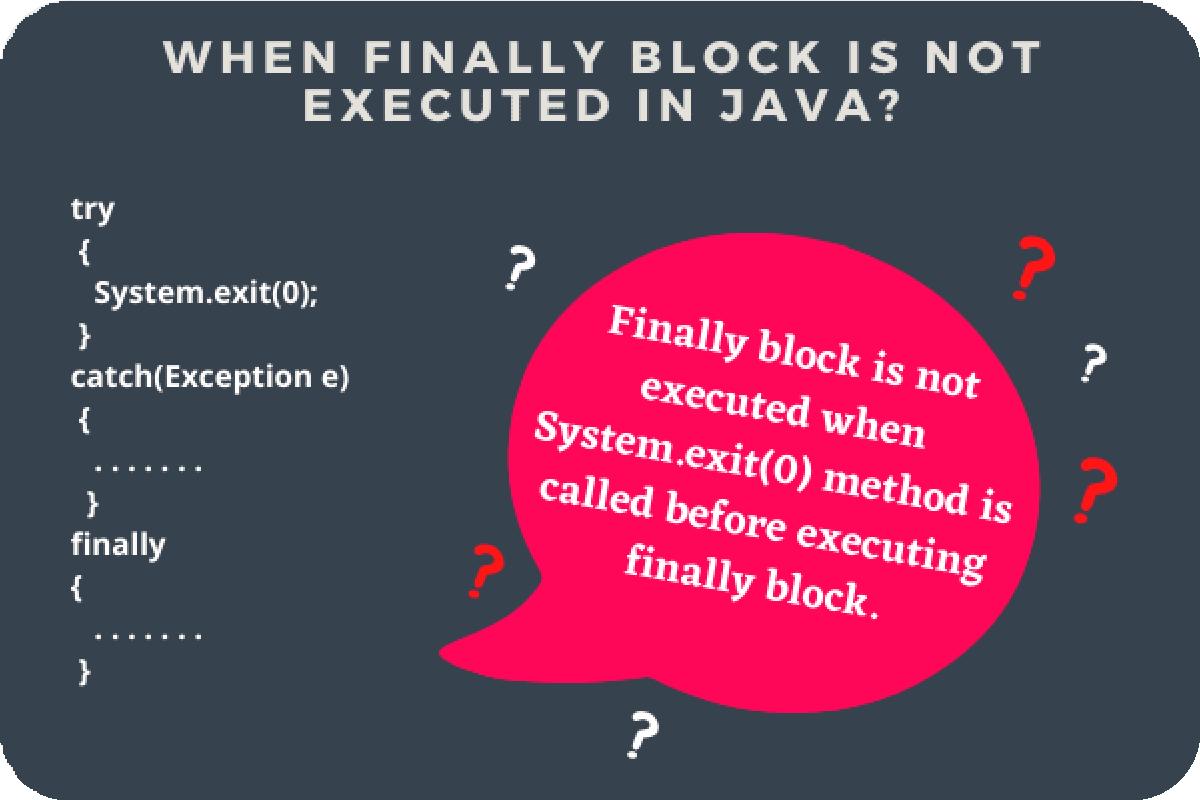


* 1. Can we define return statement in try block or catch block or finally block in Java?
  2. **If we return a value in try block or catch block or finally block, what will happen?**



**When Finally Block is not**

**executed in Java?**



**Condition where finally block is not executed in Java**



* When the System.exit() method is called in the try block before the execution of finally block, finally block will not be executed.

**Engr. A. Hafeez Babar (MUET-SZAB)** [**hafeez.babar@muetkhp.edu.pk**](mailto:hafeez.babar@muetkhp.edu.pk)

**Example** package finallyProgram;

public class FinallyBlock1

{

void m1()

{

try {

System.out.println("I am in try block"); System.exit(0);

}

finally {

System.out.println("I am in finally block");

}

System.out.println("Statement after finally block");

}

public static void main(String[] args)

{

FinallyBlock1 obj = new FinallyBlock1(); obj.m1();

}

Output:

I am in try block

} O



Example

package finallyProgram;

public class FinallyBlock2

{

void m1(){

int a = 20, b = 0; try {

System.out.println("I am in try block"); System.exit(0);

int c = a/b; System.out.println("Result: " +c);

}

catch(ArithmeticException ae)

{

Output:



I am in try block

System.out.println("I am in catch block");

}

finally {

System.out.println("I am in finally block");

}

System.out.println("Statement after finally block");

}

public static void main(String[] args) { FinallyBlock2 obj = new FinallyBlock2(); obj.m1();

}

}

**Example** package finallyProgram;

public class FinallyBlock3

{

void m1()

{

int a = 20, b = 0;

try {

System.out.println("I am in try block"); int c = a/b;

System.exit(0); System.out.println("Result: " +c);

}

catch(ArithmeticException ae)

{

System.out.println("I am in catch block");

}

finally

{

Output:

I am in try block

I am in catch block

I am in finally block

Statement after finally block

System.out.println("I am in finally block");



}

System.out.println("Statement after finally block");

}

public static void main(String[] args)

{

FinallyBlock3 obj = new FinallyBlock3(); obj.m1();

}

}

### Conditions when statements in finally block are not executed in Java



##### Conditions when statements in finally block are not executed in Java



* *Two conditions in which statements in the finally block will not be executed in Java.*

1. When the return statement is defined in the finally block, the control of execution is transferred to calling routine, and statements after the return statement in finally block are not executed.
2. When an exception occurs in the code written in the finally block. In this case, finally block does not complete normally.

**Example**



package finallyProgram; public class FinallyBlock4

{

int m1()

{

int a = 20, b = 0; try {

System.out.println("I am in try block"); int c = a/b;

System.exit(0);

System.out.println("Result: " +c);

}

catch(ArithmeticException ae)

{

System.out.println("I am in catch block"); return 20;

}

finally

{

System.out.println("I am in finally block"); return 50;

System.out.println("Statement after return statement"); //

Unreachable code.

}

}

public static void main(String[] args)

{

FinallyBlock4 obj = new FinallyBlock4();

System.out.println(obj.m1());

}

}

Output:

Exception in thread "main" java.lang.Error: Unresolved

compilation problem: Unreachable code

Example:



package finallyProgram; public class FinallyBlock5

{

int m1()

{

int a = 20, b = 0; try

{

System.out.println("I am in try block"); int c = a/b;

System.exit(0); System.out.println("Result: " +c);

}

catch(ArithmeticException ae)

{

System.out.println("I am in catch block"); return 20;

}

finally

{

System.out.println("I am in finally block");

return 50;

}

}

public static void main(String[] args)

{

FinallyBlock5 obj = new FinallyBlock5();

System.out.println(obj.m1());

}

}

Output:

I am in try block

I am in catch block I am in finally block 50



### Throw Keyword

##### Throw Keyword in Java



* + JVM automatically throws system-generated exceptions. All those exceptions are called implicit exceptions.
  + If we want to throw an exception manually or explicitly, for this, Java provides a keyword throw.
  + Throw in Java is a keyword that is used to throw a built-in exception or a custom exception explicitly or manually.

Using throw keyword, we can throw either checked or unchecked exceptions in java programming.

* + When an exception occurs in the try block, throw keyword transfers the control of execution to the caller by throwing an object of exception.
  + Only one object of exception type can be thrown by using throw keyword at a time. Throw keyword can be used inside a method or static block provided that exception handling is present.

Syntax:

**throw exception\_name;**

**Example:**



throw new ArithmeticException(); or,

ArithmeticException ae = new ArithmeticException();

throw ae;

throw new NumberFormatException();

1. The object of Throwable class or its subclasses can be created using new keyword or using a parameter inside catch clause.
2. Instances of classes other than Throwable class or its subclasses cannot be used as exception objects.



### Control flow of try-catch block with

**throw Statement in Java**

##### Control flow of try-catch block with throw Statement in Java



* + - When a throw statement is encountered in a program, the flow of execution of subsequent statements stops immediately in the try block and the corresponding catch block is searched.

Example



public class ThrowTest1

{

public static void main(String[] args)

{

try

{

ArithmeticException a = new ArithmeticException("Hello from throw"); // Line 7

throw a; // Exception thrown explicitly. // Line 8

// Line 7 and 8 can be written also in one line like this:

// throw new ArithmeticException("Hello from throw");

}

catch(ArithmeticException ae){ System.out.println("ArithmeticException caught: \n" +ae); System.out.println(ae.getMessage());

}

}

}

Output:

ArithmeticException caught: java.lang.ArithmeticException: Hello from throw Hello from throw

public class ThrowTest2

{

public static void main(String[] args)

{

int x = 20; int y = 0; try

{

int z = x/y; // Exception occurred. // Line 9 System.out.println("Result: " +z); // Line 10 throw new ArithmeticException();

}

catch(ArithmeticException ae){ System.out.println("Exception caught: \n" +ae);

}

}

}

Output:

Exception caught:

java.lang.ArithmeticException: / by zero

Example



public class ThrowTest3

{

public static void main(String[] args)

{

int x = 20; int y = 0; try

{

int z = x/y;

throw new ArithmeticException(); System.out.println("Result: " +z); // Unreachable code.

}

catch(ArithmeticException ae){

System.out.println("Exception caught: \n" +ae);

}

}

}

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation problem: Unreachable code



Example

public class ThrowTest4

{

public static void main(String[] args)

{

int num = 1;

for(num = 1; num <= 10; num++){ try

{

if(num == 5)

throw new ArithmeticException("ArithmeticException"); else if(num < 2)

throw new RuntimeException("RuntimeException");

else if(num > 9)

throw new NullPointerException("NullPointerException");

}

catch(Exception e)

{

System.out.println("Caught an exception");

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

}

}

}

}

Output:

Caught an exception RuntimeException Caught an exception ArithmeticException Caught an exception NullPointerException

Example

class Test1 extends Exception {

}

class Test2 extends Exception {

}

class Test3 extends Exception {

}

public class ThrowTest5

{

public static void main(String[] args)

{

int num = 1;

for(num = 1; num <= 10; num++){ try

{

if(num == 5)

throw new Test1(); else if(num < 2) throw new Test2(); else if(num > 9) throw new Test3();

}



catch(Exception e)

{

System.out.println("Caught an exception");

}

}

}}

Output:

Caught an exception Caught an exception Caught an exception



**Throws Keyword in Java**

Throws Keyword in Java



* In Java, sometimes a method may throw an exception in a program but cannot handle it due to not have an appropriate exception handling mechanism.
* In such a case, the programmer has to throw that exception to the caller of the method using throws clause.
* Throws clause consists of throws keyword followed by a comma-separated by the list of all exceptions thrown by that method.

Syntax

access\_specifier return\_type method\_name(parameter list)



throws exception

{

// body of the method.

}

* + Java throws keyword can be used to throw multiple exceptions thrown by a method at a time. Multiple exceptions thrown by a method can be declared by separating them in comma with the help of throws keyword.

Syntax

access\_specifier return\_type method\_name(parameter\_list) throws exception1, exception2, exceptionN

{

// body of the method.

}

* + When throws keyword is used with a method declaration, the method calling a method with throws keyword

must be enclosed within try-catch block.

Example

public class ThrowsTest1

{

public static void main(String[] args)

{

Thread.sleep(1000);

System.out.println("Hello Java");

}

}

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Unhandled exception type InterruptedException

Example

public class ThrowsTest1



{

public static void main(String[] args) throws InterruptedException

{

Thread.sleep(1000); System.out.println("Hello Java");

}

}

Output:

Hello Java

Example



package throwsProgram; import java.io.BufferedReader; import java.io.IOException;

import java.io.InputStreamReader;

public class ThrowsTest2

{

private String firstName, lastName;

void accept() throws IOException

{

// Reading data from keyboard.

InputStreamReader sr = new InputStreamReader(System.in); BufferedReader br = new BufferedReader(sr);

System.out.println("Enter your first name"); firstName = br.readLine(); System.out.println("Enter your last name"); lastName = br.readLine();

}

void display(){

System.out.println("Full Name: " +firstName+ " " +lastName);

}

public static void main(String[] args) throws IOException

{

ThrowsTest2 obj = new ThrowsTest2(); obj.accept();

obj.display();

}

} **Output:**

Enter your first name

Muzammil

Enter your last name Hussain

Full Name: Muzammil Hussain

## Difference between final, finally and finalize

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S#** | **Key** | **final** | **finally** | **finalize** |
| 1. | Definition | final is the keyword and access modifier which is used to apply restrictions on a class, method or variable. | finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not. | finalize is the method in Java which is used to perform clean up processing just before object is garbage collected. |
| 2. | Applicable to | Final keyword is used with the classes, methods and variables. | Finally block is always related to the try and catch block in exception handling. | finalize() method is used with the objects. |
| 3. | Functionality | 1. Once declared, final variable becomes constant and cannot be modified. 2. final method cannot be   overridden by sub class.   1. final class cannot be inherited. | 1. finally block runs the important code even if exception occurs or   not.   1. finally block cleans up all the   resources used in try block | finalize method performs the cleaning activities with respect to the object before its destruction. |
| 4. | Execution | Final method is executed only when we call it. | Finally block is executed as soon as the try-catch block is executed.It's execution is not dependant on the exception. | finalize method is executed just before the object is destroyed. |

**User defined Exception in Java |Custom Exception**

* + - Custom exceptions in Java are those exceptions that are created by a programmer to meet the specific requirements of the application.

Example

* + - A banking application, a customer whose age is lower than 18 years, the program throws a custom exception

indicating “needs to open joint account”.

* + - Voting age in Pakistan: If a person’s age entered is less than 18 years, the program throws “invalid age” as a

custom exception.

**Create your own User-defined**

**Exception**

###### Create your own User-defined Exception



**Step 1:** User-defined exceptions can be created simply by extending Exception class. This is done as:

class OwnException extends Exception

**Step 2:** If you do not want to store any exception details, define a default constructor in your own exception class. This can be

done as follows:

OwnException(){ }

**Step 3:** If you want to store exception details, define a parameterized constructor with string as a parameter, call superclass

(Exception) constructor from this, and store variable “str”. This can be done as follows:

OwnException(String str)

**{**

**super(str); // Call super class exception constructor and store variable "str" in it.**

**}**

**Step 4:** In the last step, we need to create an object of user-defined exception class and throw it using throw clause.

OwnException obj = new OwnException("Exception details"); throw obj;

**or,**

**throw new OwnException("Exception details");**

**Example**



package customExceptionProgram;

public class OwnException extends Exception

{

// Declare default constructor. OwnException()

{

}

}

public class MyClass {

public static void main(String[] args)

{

try

{

// Create an object of user defined exception and throw it using throw clause.

OwnException obj = new OwnException(); throw obj;

}

catch (OwnException ex)

{

System.out.println("Caught a user defined exception");

}

}

}

Output:

Caught a user defined exception

Example



public class OwnException extends Exception

{

// Declare parameterized constructor with String as a parameter. OwnException(String str)

{

super(str); // Call super exception class constructor.

}

}

public class MyClass {

public static void main(String[] args)

{

try

{

// Create an object of user defined exception and throw it using throw clause.

OwnException obj = new OwnException("Creating user defined exception");

throw obj;

// or, throw new OwnException("Creating user defined exception");

}

catch (OwnException ex)

{

Output:

Caught a user defined exception Creating user defined exception

System.out.println("Caught a user defined exception"); System.out.println(ex.getMessage());

}

}

}

Example

public class InvalidAgeException extends Exception

{

// Declare a parameterized exception with string str as a parameter.

InvalidAgeException(String str)

{

super(str);

}

}

import java.util.Scanner; public class TestClass

{

private static int age;

static void validate() throws InvalidAgeException

{

Scanner sc = new Scanner(System.in); System.out.println("Enter your age"); age = sc.nextInt();

if(age < 18)

throw new InvalidAgeException("Invalid Age, You are not



eligible to vote");

else

System.out.println("Welcome to vote");

}

public static void main(String[] args)

{

try

{

validate();

}

catch(Exception e)

{

System.out.println("Caught an Exception: \n "+e);

}

**Output:**

}

First Execution:

} Enter your age 7

Caught an Exception:

customExceptionProgram.InvalidAgeException: Invalid Age, You are not eligible to vote Second Execution:

Enter your age 40

Welcome to vote



### Chained Exceptions in Java

* Java 2, version 1.4 added a new feature chained exceptions. The chained exceptions feature relates one exception with another exception. The second exception explains the cause of the first exception.



* Chained exception in Java is a technique to handle exceptions that occur one after another. This technique

helps us to know when one exception causes another.

Example

* + let us assume that a, b, and c are objects of three different exception types A, B, and C respectively. The object a of type A causes an exception of type B to occur and an object of B type also causes an exception of C type.
  1. **Constructors added to Throwable class to support Chained Exceptions**
  + To implement chained exceptions feature in JDK 1.4 version, Java added two constructors and two methods to Throwable class. The constructors that support chained exception in java are as follows:

1. **Throwable(Throwable causeExc)** // causeExc is exception that causes current exception.
2. **Throwable(String msg, causeExc)** // msg is an exception message and causeExc is exception that causes the current exception.

## Errors in Java | Runtime &

**Compile Time Errors**

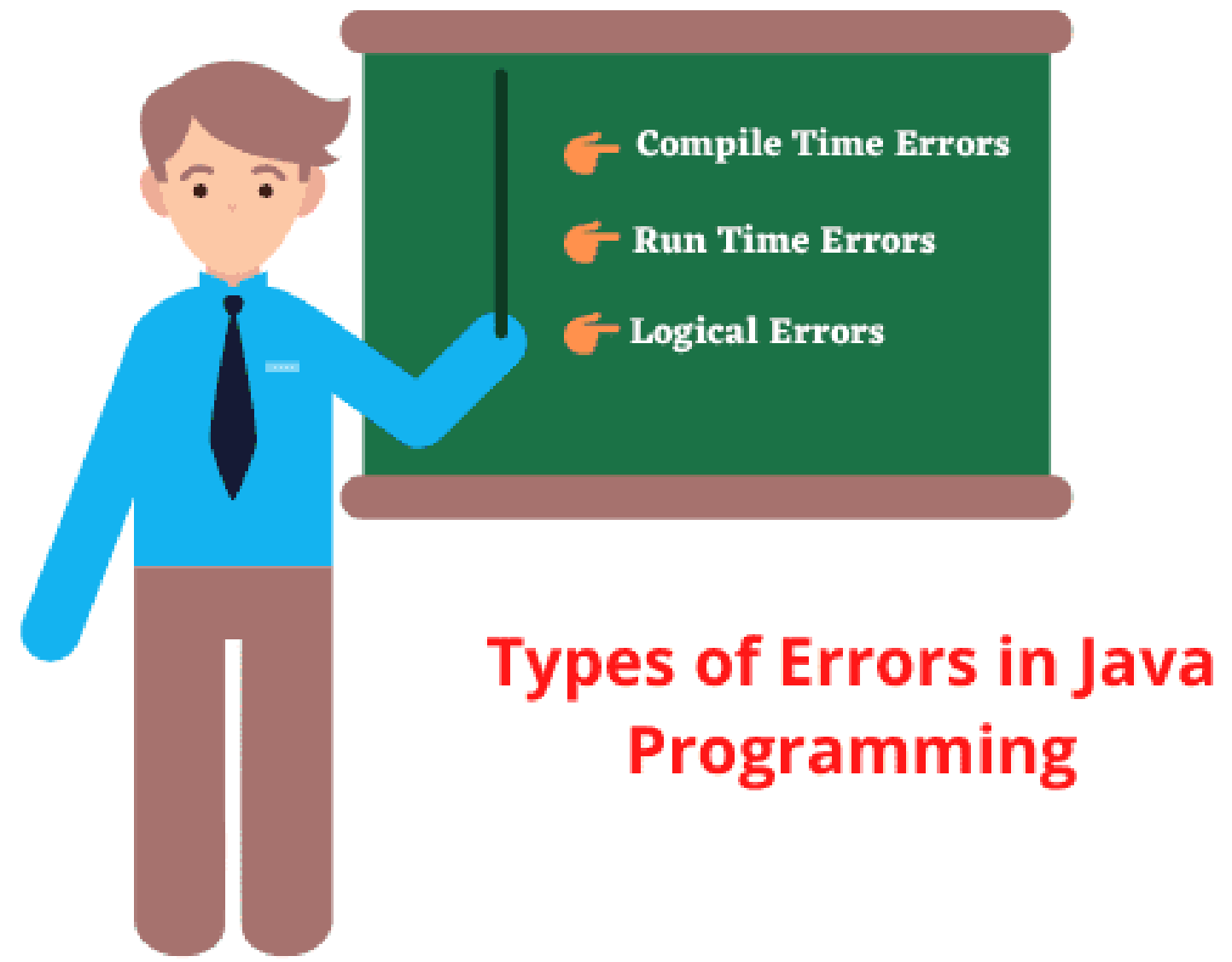
* + - Errors in Java occur when a programmer violates the rules of Java programming language.



* + - It might be due to programmer’s typing mistakes while developing a program. It may produce incorrect output or may terminate the execution of the program abnormally.







##### Java Compile Time Errors



* + - * Compile-time errors occur when syntactical problems occur in a java program due to incorrect use of Java syntax.
      * These syntactical problems may be missing semicolons, missing brackets, misspelled keywords, use of undeclared variables, class not found, missing double-quote in Strings, and so on.

Example

public class CompileTimeErrorEx

{

public static void main(String[] args)

{

System.out.println("a") // Syntax error. Semicolon missing.

}

}

Compile time error in Java code:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Syntax error, insert ";" to complete BlockStatements

public class MisspelledVar

{

public static void main(String[] args)

{

int x = 20, y = 30;

// Declare variable sum. int sum = x + y;

// Call variable Sum with Capital S.

System.out.println("Sum of two numbers: " + Sum); // Calling

of undeclared variable.

}

}

Compile time error in Java code:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Sum cannot be resolved to a variable

public class MissingBracket

{

public static void main(String[] args)

{

int i;

int sum = 0;

// Missing bracket in for statement.

for (i = 1; i <= 5; i++ // insert " ) Statement" to complete For Statement.

{

sum = sum + i;

}

System.out.println("Sum of 1 to 5 \n"); System.out.println(sum);

}

}

Compile time error in Java code:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Syntax error, insert ") Statement" to complete ForStatement

##### Runtime Errors in Java



* + - * + Runtime errors occur when a program is successfully compiled creating the .class file but does not run properly. It

is detected at run time (i.e. during the execution of the program).

* + - * + Such a program that contains runtime errors, may produce wrong results due to wrong logic or terminate the program. These runtime errors are usually known as exceptions.

Example:

if a user inputs a value of string type in a program but the computer is expecting an integer value, a runtime error will be generated.

##### Most common runtime errors are as follows:



* 1. Dividing an integer by zero.
  2. Accessing an element that is out of range of the array.
  3. Trying to store a value into an array that is not compatible type.
  4. Passing an argument that is not in a valid range or valid value for a method.
  5. Striving to use a negative size for an array.
  6. Attempting to convert an invalid string into a number.
  7. and many more.

Example



public class DivisionByZeroError

{

public static void main(String[] args)

{

int a = 20, b = 5, c = 5;

int z = a/(b-c); // Division by zero.

System.out.println("Result: " +z);

}

}

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at errorsProgram.DivisionByZeroError.main(DivisionByZeroError.java:8)

##### Logical Errors in Java Program



* + - * + Logical errors in Java are the most critical errors in a program and they are difficult to detect. These errors occur when the programmer uses incorrect logic or wrong formula in the coding.
        + The program will be compiled and executed successfully but does not return the expected output.
        + Logical errors are not detected either by Java compiler or JVM (Java runtime system). The programmer is entirely responsible for them. They can be detected by application testers when they compare the actual result with its expected result.

For example:

A programmer wants to print even numbers from an array but he uses division (/) operator instead of modulus

(%) operator to get the remainder of each number. Due to which he got the wrong results.

Example

public class LogicalErrorEx



{

public static void main(String[] args)

{

int a[]={1, 2 , 5, 6, 3, 10, 12, 13, 14};

System.out.println("Even Numbers:");

for(int i = 0; i <a.length; i++)

{

if(a[i] / 2 == 0) // Using wrong operator.

{

System.out.println(a[i]);

}

}

}

} **Output:**

Even Numbers:

1

* + - * Two types of exceptions in Java: Predefined and Custom exceptions.
      * The root class for all the exceptions in the hierarchy of exception classes is java.lang.Throwable.
      * Exception classes are mainly divided into three types: system errors, exceptions, and runtime exceptions.
      * System errors are represented by Error class and thrown by JVM.
      * Exceptions are represented by Exception classes that describe errors in your program.
        + The main advantage of exception handling technique is to maintain the normal flow of the program.
        + It provides flexibility in handling situations of errors.
        + It allows us to define a user-friendly message to handle the exception.
        + The exception handling technique helps to separate “Error-Handling code” from “Regular code.”

