# Access Modifiers in Java:

There are two types of modifiers in Java: ****access modifiers**** and ****non-access modifiers****.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. ****Private****: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. ****Default****: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. ****Protected****: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. ****Public****: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

### Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

### 1) Private

The private access modifier is accessible only within the class.

****Simple example of private access modifier****

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

1. **class** A{
2. **private** **int** data=40;
3. **private** **void** msg(){System.out.println("Hello java");}
4. }
6. **public** **class** Simple{
7. **public** **static** **void** main(String args[]){
8. A obj=**new** A();
9. System.out.println(obj.data);//Compile Time Error
10. obj.msg();//Compile Time Error
11. }
12. }

### Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

1. **class** A{
2. **private** A(){}//private constructor
3. **void** msg(){System.out.println("Hello java");}
4. }
5. **public** **class** Simple{
6. **public** **static** **void** main(String args[]){
7. A obj=**new** A();//Compile Time Error
8. }
9. }

### 2) Default

If you don't use any modifier, it is treated as ****default**** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

****Example of default access modifier****

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

1. //save by A.java
2. **package** pack;
3. **class** A{
4. **void** msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. **package** mypack;
8. **import** pack.\*;
9. **class** B{
10. **public** **static** **void** main(String args[]){
11. A obj = **new** A();//Compile Time Error
12. obj.msg();//Compile Time Error
13. }
14. }

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

### 3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

****Example of protected access modifier****

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

1. //save by A.java
2. **package** pack;
3. **public** **class** A{
4. **protected** **void** msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. **package** mypack;
8. **import** pack.\*;
10. **class** B **extends** A{
11. **public** **static** **void** main(String args[]){
12. B obj = **new** B();
13. obj.msg();
14. }
15. }

### 4) Public

The ****public access modifier**** is accessible everywhere. It has the widest scope among all other modifiers.

****Example of public access modifier****

1. //save by A.java
3. **package** pack;
4. **public** **class** A{
5. **public** **void** msg(){System.out.println("Hello");}
6. }
7. //save by B.java
9. **package** mypack;
10. **import** pack.\*;
12. **class** B{
13. **public** **static** **void** main(String args[]){
14. A obj = **new** A();
15. obj.msg();
16. }
17. }

# Exception Handling in Java

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

In this tutorial, we will learn about Java exceptions, it's types, and the difference between checked and unchecked exceptions.

## 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 need to handle exceptions. Let's consider a scenario:

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

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in [Java](https://www.javatpoint.com/java-tutorial)

.

## Hierarchy of Java Exception classes

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



### 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
2. Unchecked Exception
3. Error



## Difference between Checked and Unchecked Exceptions

### 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.

### 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.

### 3) Error

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

## java Exception Keywords

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

Try

Catch

Finally

Throw

Throws

\* 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.

\* 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.

\* The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

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

\* 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.

## Java Exception Handling Example

Let's see an example of Java Exception Handling in which we are using a try-catch statement to handle the exception.

****JavaExceptionExample.java****

1. **public** **class** JavaExceptionExample{
2. **public** **static** **void** main(String args[]){
3. **try**{
4. //code that may raise exception
5. **int** data=100/0;
6. }**catch**(ArithmeticException e){System.out.println(e);}
7. //rest code of the program
8. System.out.println("rest of the code...");
9. }
10. }

## Common Scenarios of Java Exceptions

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

### 1) A scenario where ArithmeticException occurs

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

1. **int** a=50/0;//ArithmeticException

### 2) A scenario where NullPointerException occurs

If we have a null value in any [variable](https://www.javatpoint.com/java-variables)

, performing any operation on the variable throws a NullPointerException.

1. String s=**null**;
2. System.out.println(s.length());//NullPointerException

### 3) A scenario where NumberFormatException occurs

If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a [string](https://www.javatpoint.com/java-string)

variable that has characters; converting this variable into digit will cause NumberFormatException.

1. String s="abc";
2. **int** i=Integer.parseInt(s);//NumberFormatException

### 4) A scenario where ArrayIndexOutOfBoundsException occurs

When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

1. **int** a[]=**new** **int**[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException