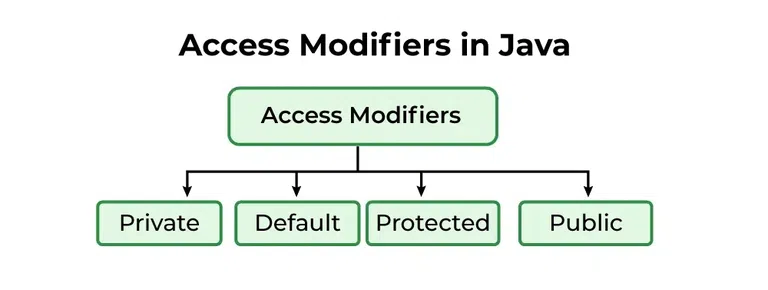
**Q1 .Access Modifiers in Java**

In Java, Access modifiers help to restrict the scope of a class, constructor, variable, method, or data member. It provides security, accessibility, etc. to the user depending upon the access modifier used with the element. Let us learn about Java Access Modifiers, their types, and the uses of access modifiers in this article.

**Types of Access Modifiers in Java**

There are four types of access modifiers available in Java:

1. Default – No keyword required
2. Private
3. Protected
4. Public



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.

### **Understanding Java Access Modifiers**

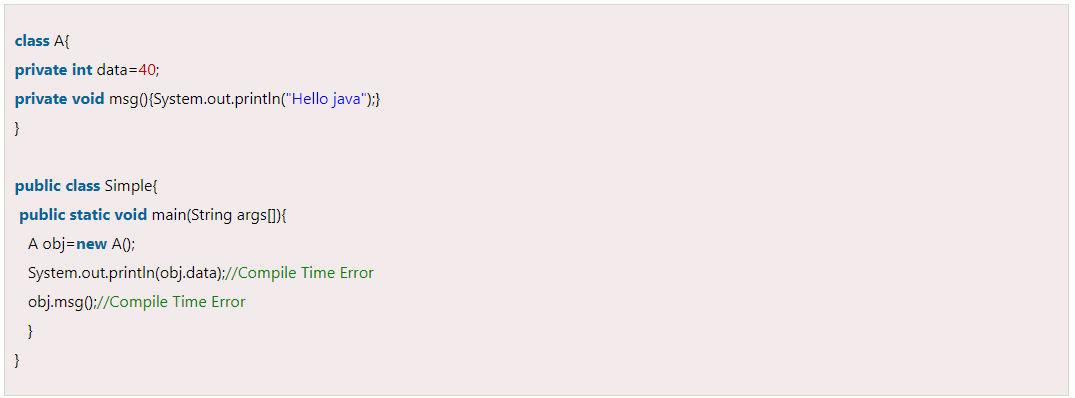
Let's understand the access modifiers in Java by a simple table.

### **1) Private**

The private access modifier is accessible only within the class.

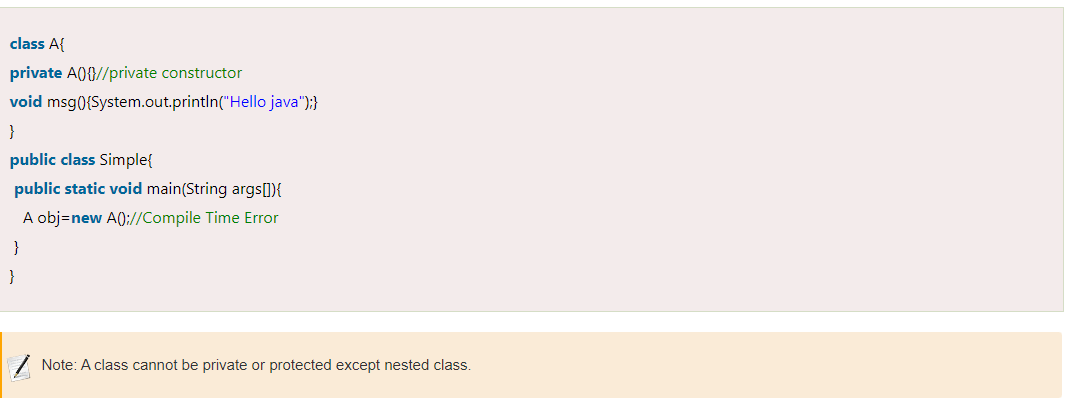
**Simple example of private access modifier**

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.



### **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:

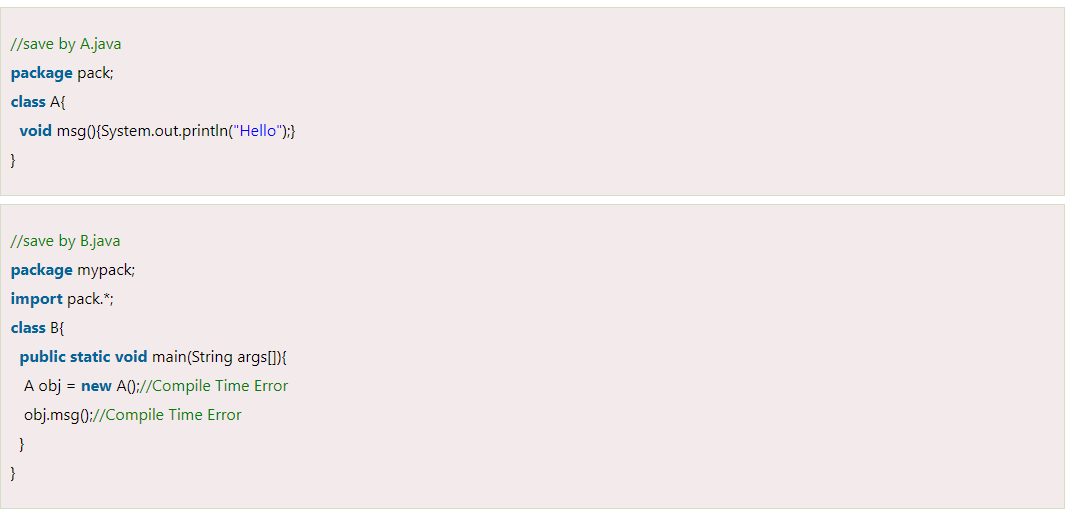


### **2) Default**

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 my pack. 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.



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



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 modifier.

**Example of protected access modifier**

In this example, we have created the two packages pack and my pack. 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.



### **4) Public**

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

**Example of public access modifier**



## Algorithm to use access modifier in Java

#### Here’s a basic algorithm for using access modifiers in Java:

* **Define a class:** Create a class that represents the object you want to manage.
* **Define instance variables:**Within the class, define instance variables that represent the data you want to manage.
* Specify an access modifier: For each instance variable, specify an access modifier that determines the visibility of the variable. The three main access modifiers in Java are private, protected, and public.
* **Use private for variables that should only be accessible within the class:** If you want to prevent access to a variable from outside the class, use the private access modifier. This is the most restrictive access modifier and provides the greatest level of encapsulation.
* **Use protected for variables that should be accessible within the class and its subclasses**: If you want to allow access to a variable from within the class and its subclasses, use the protected access modifier. This is less restrictive than private and provides some level of inheritance.
* **Use public for variables that should be accessible from anywhere**: If you want to allow access to a variable from anywhere, use the public access modifier. This is the least restrictive access modifier and provides the least amount of encapsulation.
* **Use accessor and mutator methods to manage access to the variables:**In order to access and modify the variables, use accessor (getter) and mutator (setter) methods, even if the variables have a public access modifier. This provides a level of abstraction and makes your code more maintainable and testable.

**Q2. Errors and Exceptions in Java**

In Java, errors and exceptions are both types of throwable objects, but they represent different types of problems that can occur during the execution of a program.

Errors are usually caused by serious problems that are outside the control of the program, such as running out of memory or a system crash. Errors are represented by the Error class and its subclasses. Some common examples of errors in Java include:

* Out of Memory Error: Thrown when the Java Virtual Machine (JVM) runs out of memory.
* Stack Overflow Error: Thrown when the call stack overflows due to too many method invocations.
* No Class Def Found Error: Thrown when a required class cannot be found.

Since errors are generally caused by problems that cannot be recovered from, it’s usually not appropriate for a program to catch errors. Instead, the best course of action is usually to log the error and exit the program.

Exceptions, on the other hand, are used to handle errors that can be recovered from within the program. Exceptions are represented by the Exception class and its subclasses. Some common examples of exceptions in Java include:

* Null Pointer Exception: Thrown when a null reference is accessed.
* Illegal Argument Exception: Thrown when an illegal argument is passed to a method.
* IO Exception: Thrown when an I/O operation fails.

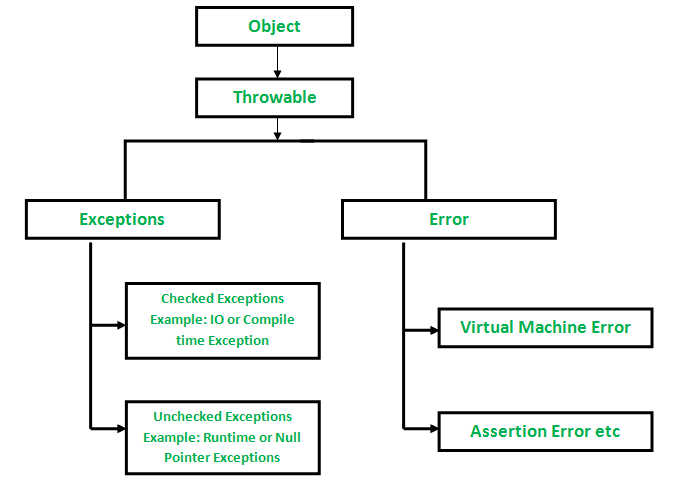
Since exceptions can be caught and handled within a program, it’s common to include code to catch and handle exceptions in Java programs. By handling exceptions, you can provide more informative error messages to users and prevent the program from crashing.

In summary, errors and exceptions represent different types of problems that can occur during program execution. Errors are usually caused by serious problems that cannot be recovered from, while exceptions are used to handle recoverable errors within a program.

In java, both Errors and Exceptions are the subclasses of java lang. throwable class. [Error](https://www.geeksforgeeks.org/types-of-errors-in-java-with-examples/) refers to an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus, errors should be removed before compiling and executing. It is of three types:

* Compile-time
* Run-time
* Logical

Whereas [exceptions in java](https://www.geeksforgeeks.org/exceptions-in-java/) refer to an unwanted or unexpected event, which occurs during the execution of a program i.e., at run time, that disrupts the normal flow of the program’s instructions.



## Exception

The term [**exception**](https://www.javatpoint.com/exception-handling-in-java) is shorthand for the phrase **exception event**. It is an event that occurs during the execution of the program and interrupts the normal flow of program instructions. These are the errors that occur at compile time and run time. It occurs in the code written by the developers. It can be recovered by using the try-catch block and throws keyword. There are two types of exceptions i.e., **checked** and **unchecked**.

There are some important points that should be kept in mind while dealing with the exception:

* When an error is detected, an exception is thrown.
* Any exception that is thrown must be caught by the exception handler.
* If the programmer has forgotten to provide an exception handler, the exception will be caught by the catch-all exception handler provided by the system.
* Exception may be rethrown if exception handler is failure to handle it.

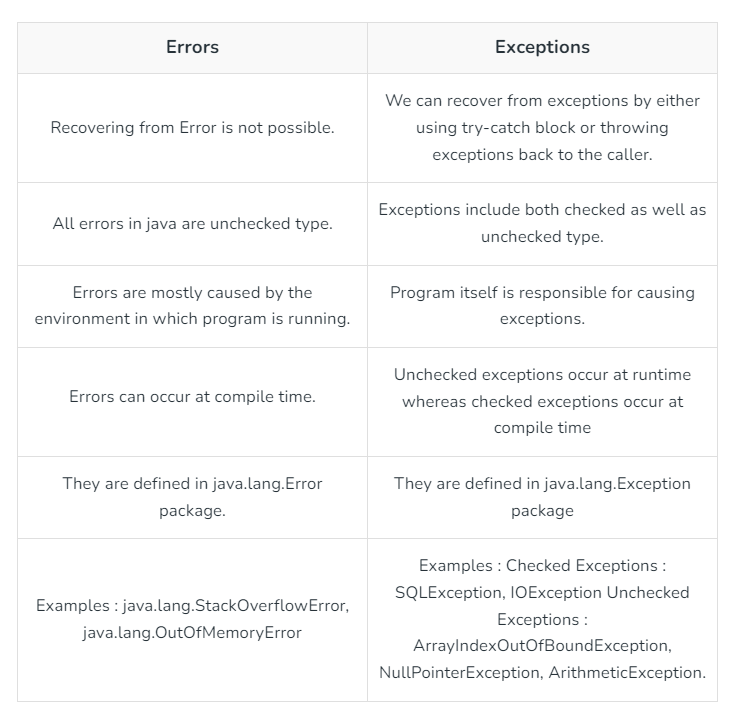
### Advantages of Exceptions

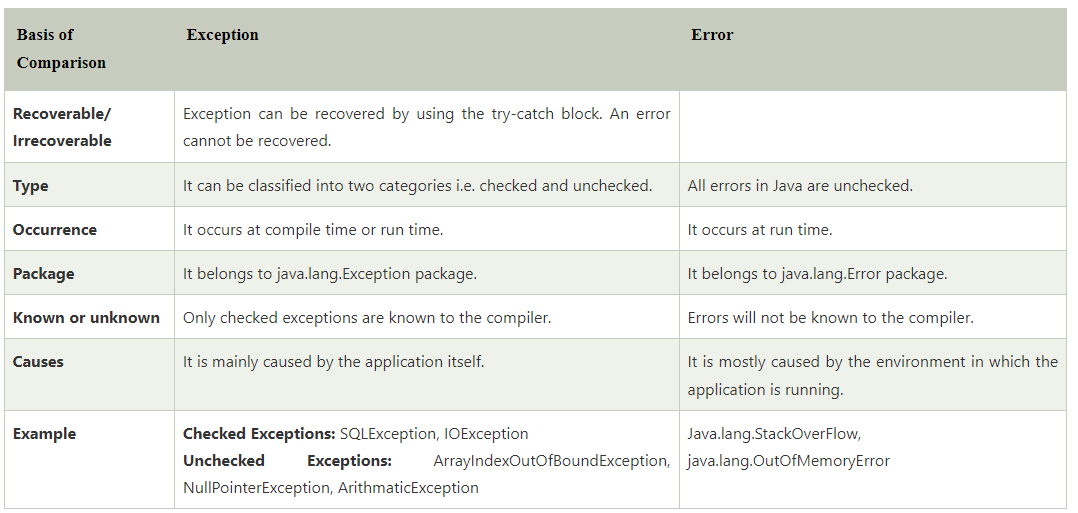
* It separates error handling code from regular code.
* It has the ability to propagate error reporting up the call stack of methods.
* The grouping or categorizing of exceptions is a natural outcome of the class hierarchy.

## Error

[Errors](https://www.javatpoint.com/java-error) are problems that mainly occur due to the lack of system resources. It cannot be caught or handled. It indicates a serious problem. It occurs at run time. These are always unchecked. An example of errors is **Out of Memory Error, Linkage Error, Assertion Error**, etc. are the subclasses of the Error class.

## Difference Between Exception and Error



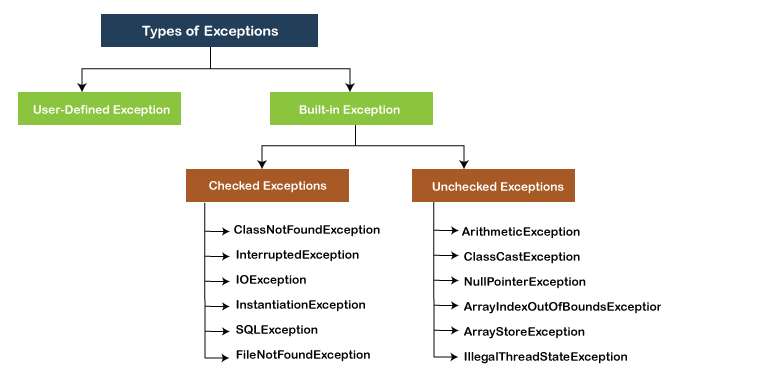


**Q3 . Types of Exception in Java**

In Java, exception is an event that occurs during the execution of a program and disrupts the normal flow of the program's instructions. Bugs or errors that we don't want and restrict our program's normal execution of code are referred to as exceptions. In this section, we will focus on the types of exceptions in Java and the differences between the two.

Exceptions can be categorized into two ways:

1. Built-in Exceptions
   * Checked Exception
   * Unchecked Exception
2. User-Defined Exceptions



## Built-in Exception

[Exceptions](https://www.javatpoint.com/exception-handling-in-java) that are already available in **Java libraries** are referred to as **built-in exception**. These exceptions are able to define the error situation so that we can understand the reason of getting this error. It can be categorized into two broad categories, i.e., **checked exceptions** and **unchecked exception**.

### **Checked Exception**

**Checked** exceptions are called **compile-time** exceptions because these exceptions are checked at compile-time by the compiler. The compiler ensures whether the programmer handles the exception or not. The programmer should have to handle the exception; otherwise, the system has shown a compilation error.

Theseare the exceptions that are checked at compile time. 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/). In checked exceptions, there are two types: fully checked and partially checked exceptions. A fully checked exception is a checked exception where all its child classes are also checked, like IO Exception, and Interrupted Exception. A partially checked exception is a checked exception where some of its child classes are unchecked, like an Exception.

## Unchecked Exceptions

These are the exceptions that are not checked at compile time. In C++, all exceptions are unchecked, so it is not forced by the compiler’s to either handle or specify the exception. It is up to the programmers to be civilized and specify or catch the exceptions. In Java, exceptions under Error and Runtime Exception classes are unchecked exceptions, everything else under throwable is checked.

Difference between checked and unchecked exceptions in Java.

## Checked Exceptions

* They occur at compile time.
* The compiler checks for a checked exception.
* These exceptions can be handled at the compilation time.
* It is a sub-class of the exception class.
* The JVM requires that the exception be caught and handled.
* Example of Checked exception- ‘File Not Found Exception’

## Unchecked Exceptions

* These exceptions occur at runtime.
* The compiler doesn’t check for these kinds of exceptions.
* These kinds of exceptions can’t be caught or handled during compilation time.
* This is because the exceptions are generated due to the mistakes in the program.
* These are not a part of the ‘Exception’ class since they are runtime exceptions.
* The JVM doesn’t require the exception to be caught and handled.
* Example of Unchecked Exceptions- ‘No Such Element Exception’

Output screenshot of code:

