**Access Modifiers:**

Access modifiers are object-oriented programming that is used to set the accessibility of classes, constructors, methods, and other members of Java.

Using the access modifiers we can set the scope or accessibility of these classes, methods, constructors, and other members.

Access modifiers are keywords that can be used to control the visibility of fields, methods, and constructors in a class. The four access modifiers in Java are public, protected, default, and private.

**Four Types of Access Modifiers**

* Default
* Private
* Protected
* Public

**Private Access Modifier:**

The private access modifier is specified when any member of a class is prefixed with the private keyword. In comparison with the other access modifiers, this is the most restricted access modifier.

When the methods or data members are prefixed with a private access modifier, the visibility of these methods and data members are restricted so, they can be accessed only within the same class where they have been declared, they will not be visible to the outside world.

If we have another class from the same package still, we will not be able to access these methods or data members. So usually, we keep the class variables and methods as private, which are intended to be used inside the same class where declared.

Let us consider an example where we will consider two classes and within the same package. We will declare a variable and a method as private in one class and then try to access these methods and variables from another class, then we get error as we cannot access the methods and variable declared as private.

**Default Access Modifier:**

It is not a keyword. Any Java members such as class or methods or data members when not specified with any access modifier they are by default considered as default access modifiers. These methods or data members are only accessible within the same package and they cannot be accessed from outside the package. It provides more visibility than a private access modifier. But this access modifier is more restricted than protected and public access modifiers.

Example:

we have two different packages p1 and p2. In the p1 package, we have class A1 where we declared a default variable and a default method. Now we are trying to access this variable and method from outside the package that is from package p2 which has a class A2. When we try to access these variables and methods from outside the package we get a Compile time error.

**Protected Access Modifier:**

It is a keyword. This access modifier is used to access the methods or data members of a class within the same package as well as outside the package but only through inheritance. The protected access modifier has more accessibility than private and defaults access modifiers. But it has less visibility than the public access modifier.

Example:

we have two packages p1 and p2. In package p1 we have class A1 where we have declared a protected method. In package p2 we are inheriting the members of class A1 inside class A2 with help of extending keywords and creating a relationship between the two classes. We can also say that class A1 is the parent class or the superclass and class A2 is the child class or the subclass respectively. When we inherit the members of class A1 inside class A2, with the help of a protected access modifier we can access the members of class A1 of package p1 from class A2 of the different package p2.

**Public Access Modifier:**

It is a keyword. If a class member like variable, method, or data members are prefixed with a public access modifier, then they can be accessed from anywhere inside the program. That is, they can be accessed within the same class as well as from outside the different classes.

It also includes access within the same package and also from outside the package. The members like variables, methods, and other data members can be accessed globally.

Using public access modifiers we can provide access to the members most simply. There are no restrictions on public access modifier members. Hence, it has the widest accessibility or visibility scope as compared to the rest of the access modifiers.

Example:

we have two different packages p1 and p2. In p1 we have a class a1 where we have declared a variable and a method prefixed public keyword. And in the p2 package, we have a class A2 from where we are trying to access the members of class A1 without inheritance.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accessibility** | **Private** | **Default** | **Protected** | **Public** |
| **Same Package** | **Same Class** | Yes | Yes | Yes | Yes |
| **Without Inheritance** | No | Yes | Yes | Yes |  |
| **With Inheritance** | No | Yes | Yes | Yes |  |
| **Different Package** | **Without Inheritance** | No | No | No | Yes |
| **With Inheritance** | No | No | Yes | Yes |  |

Difference between Exception and Error

**Error:**

Errors occur at compile time and run time, which can terminate the compilation or execution. 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 OutOfMemoryError, LinkageError, AssertionError, etc

Types of Errors:

* Syntax Errors
* Runtime Errors
* Logical Errors

**Syntax Errors:**

Syntax Errors are those errors detected during the compilation phase by the compiler when your code does not follow the syntactical rules of the programming language you are using. e.g, missing semi-colon(s), missing parenthesis, using else if() block directly without using if block first, returning nothing from the function when the return type is some data type, say, int, etc.

**Runtime Error:**

Runtime Errors occur during the execution of a program, due to lack of system resources, or due to irrelevant input by the user. The compiler has no idea whatsoever how to detect these kinds of errors. For example, dividing a number by 0, accessing an element from an array that is out of range, trying to convert an invalid string to an integer, out of memory error, etc.

**Logical Error:**

Logical Errors are those errors where the program returned incorrect results when you were expecting the desired result. These occur due to some mistake in the code logic made by the programmer. The compiler cannot detect these errors. The user can just understand them after seeing the output. These are also known as Semantic Errors.

For example, When the programmer writes mistakenly, if(i = 1) instead of, if(i == 1): This will change the program's narrative.

**Exception**

An Exception is the occurrence of an event that can disrupt the normal flow of the program instructions. Exceptions can be caught and handled in order to keep the program working for the exceptional situation as well, instead of halting the program flow. If the exception is not handled, then it can result in the termination of the program. Exceptions can be used to indicate that an error has occurred in the program.

**Types of Exception:**

* Checked Exception
* Unchecked Exception

Difference Between Checked and Unchecked Exception

**Checked Exceptions**

Exceptions that occur and can be detected at compile time are known as Checked Exceptions. These exceptions prevent the program from running, and so they must be handled by the programmer. Otherwise, the program will not get compiled.

For example, ClassNotFoundException, FileNotFoundException, IOException, SQLException, etc.

**Unchecked Exceptions**

Exceptions that occur at the time of the program execution, i.e., at runtime, are known as Unchecked Exceptions. These exceptions are ignored at the time of compilation.

They occur due to the following main reasons:

* Invalid User Input
* Bugs in the program
* Improper usage of an API
* Memory limit exceeded

Examples of unchecked exceptions : ArrayIndexOutOfBoundsException, NullPointerException, IllegalArgumentException,

ArithmeticException, NumberFormatException,StringOutOfBoundsException etc.