ACCESS SPECIFIERS/ MODIFIERS:

- The keywords which define accessibility permissions are called as Access Modifiers.
- The different levels of Accessibility modifier levels are:
    - Public
    - Protected
    - Private
    - Default

1. Public:

- A class,method,constructor,an interface etc declared public can be accessed from any other class. Therfore fields,methods,blocks declared inside a public class can be accessed from any class belonging to the java universe.
- However if the public class we are trying to access is in a different package, then the public class still needs to be imported.
- Because of class inheritance , all public methods and variables of a class are inherited by its subclasses.

Example of public access modifier

```
//save by A.java

package pack;

public class A{

public void msg(){

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

}


//save by B.java

package mypack;

import pack.*;
```

```
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

2. **Protected:**

   - Variables,methods and constructors which are declared protected in a super class can be accessed only by the subclasses in other package or any class with in the package the protected members class.
   - The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in an interface cannot be declared protected.
   - Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class which is trying to use it.

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.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
```

```
class B extends A{
public static void main(String args[]){
 B obj = new B();
 obj.msg();
}
}
```

3. **Private:**
    - Variables, Methods and constructors that declared as private can only be accessed with in the declared and class itself.
    - Class and interface cannot be private.variables that are declared private can be accessed outside the class , if the public getter methods are present in the class.
    - Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

Example of private access modifier

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

4.default:

- It means that explicity an access modifier for a class, field, method, etc is not declared.
- A variable or method declared without any access modifier is available to any other class in the same package.
- The fields in an interface are implicity public static final and the methods in an interface are by default 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.

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}


//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
    A obj = new A();//Compile Time Error
    obj.msg();//Compile Time Error
  }
}
```

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

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:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
```

statement 6;

statement 7;

statement 8;

statement 9;

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

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

```
public class JavaExceptionExample{
   public static void main(String args[]){
    try{
        //code that may raise exception
        int data=100/0;
    }catch(ArithmeticException e){System.out.println(e);}
    //rest code of the program
```

```
        System.out.println("rest of the code...");
      }
}
```

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.

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

2) 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
```

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

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

```
String s="abc";
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.

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```