**Assignment 11**

**Title of Assignment**

Any application defining scope of Formal parameter, Global parameter, Local parameter accessing mechanism and also relevance to private, public and protected access. Write a Java program which demonstrates the scope rules of the programming mechanism

**Relevant Theory**

**Scope and Lifetime of Variables**

The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it is destroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it. We have written a few classes till now. You might have observed that not all variables are the same. The ones declared in the body of a method were different from those that were declared in the class itself. There are there types of variables: instance variables, formal parameters or local variables and local variables.

**Instance variables**

Instance variables are those that are defined within a class itself and not in any method or constructor of the class. They are known as instance variables because every instance of the class (object) contains a copy of these variables. The scope of instance variables is determined by the access specifier that is applied to these variables. We have already seen about it earlier. The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by the garbage collector of Java when there are no more reference to that object. We shall see about Java's automatic garbage collector later on.

**Argument variables**

These are the variables that are defined in the header oaf constructor or a method. The scope of these variables is the method or constructor in which they are defined. The lifetime is limited to the time for which the method keeps executing. Once the method finishes execution, these variables are destroyed.

**Local variables**

A local variable is the one that is declared within a method or a constructor (not in the header). The scope and lifetime are limited to the method itself.

One important distinction between these three types of variables is that access specifiers can be applied to instance variables only and not to argument or local variables.

Global variables can be accessed from anywhere in a program, so variables that do not change are generally global in scope. A program that relies on a value of pi, for example, might use a global variable. They are often placed at the beginning of a program where they are easy to access, and constant global variables are often spelled with all capital letters.

**Local**  
These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program. As such, they are normally implemented using a stack. Local variables cease to exist once the function that created them is completed. They are recreated each time a function is executed or called.

**Global**  
These variables can be accessed (ie known) by any function comprising the program. They are implemented by associating memory locations with variable names. They do not get recreated if the function is recalled.

**Access Modifier**

Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors. The four access levels are −

* Visible to the package, the default. No modifiers are needed.
* Visible to the class only (private).
* Visible to the world (public).
* Visible to the package and all subclasses (protected).

## Default Access Modifier - No Keyword

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

### Example

Variables and methods can be declared without any modifiers, as in the following examples −

String version = "1.5.1";

boolean processOrder() {

return true;

}

## Private Access Modifier - Private

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Variables that are declared private can be accessed outside the class, if 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

The following class uses private access control −

public class Logger {

private String format;

public String getFormat() {

return this.format;

}

public void setFormat(String format) {

this.format = format;

}

}

Here, the *format* variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.

So, to make this variable available to the outside world, we defined two public methods: *getFormat()*, which returns the value of format, and *setFormat(String)*, which sets its value.

## Public Access Modifier - Public

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, 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

The following function uses public access control −

public static void main(String[] arguments) {

// ...

}

The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

## Protected Access Modifier - Protected

Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of 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 a interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

### Example

The following parent class uses protected access control, to allow its child class override *openSpeaker()* method −

class AudioPlayer {

protected boolean openSpeaker(Speaker sp) {

// implementation details

}

}

class StreamingAudioPlayer {

boolean openSpeaker(Speaker sp) {

// implementation details

}

}

Here, if we define openSpeaker() method as private, then it would not be accessible from any other class other than *AudioPlayer*. If we define it as public, then it would become accessible to all the outside world. But our intention is to expose this method to its subclass only, that’s why we have used protected modifier.

**Conclusion: Understood and performed application of scope of variable and Access specifier**