Scope of Variables in Java with Example

-Siva Yannam

Scope of Variables in Java | Scope is that area of the program where the variable is visible to a program and can be used (accessible). i.e. the scope of variable determines its accessibility for other parts of program. Java allows declaring variables within any block. A block defines a scope that starts with an opening curly brace and ends with a closing curly brace.

There are three types of variables in java, depending on their scope:

- local variables
- instance variables
- class variables (static variables).

Scope of Local Variables in Java

1. When the local variable is created inside a method, constructor, or block, their scope only remains within the method, block, or constructor. They are visible only within the method, constructor, or block. As you exit from the method or block then the scope of a local variable is destroyed.

2. We cannot access local variables from outside the method, constructor, or block.

3. We can not change their values from outside of the block.

Program source code 1:

```
package variablePrograms;

public class School

{
```

```
// Declaration of instance variables.

    public String name = "John";

// Declaration of constructor.

   School()

   {

     int id = 1234;

     System.out.println("Id of Student: " +id);

   }

// Declaration of user-defined method in instance area.

   public void mySchool()

   {

// Declaration of local variable.

     String schoolName = "RSVM";

     System.out.println("Name of School: " +schoolName);

   }

public void mySchool1()

{

  System.out.println("Name of School: " +schoolName)// Not possible because local variable cannot access from outside the method, constructor, or block.

}
```

```
public static void main(String[] args)

{

// Create the object of class 'School'.

    School sc = new School();

     sc.mySchool();

  }

 }

Output:

          Id of Student: 1234

          Name of School: RSVM
```

In the preceding example program, we cannot access the local variable "schoolName" from outside the method mySchool(). If we try to access it, we will get compile-time error.

Let's take one more example program to understand better.

Program source code 2:

```
package variablePrograms;

 public class School

 {

// Declaration of instance variable.

    public String name = "John";

// Declaration of local variable in constructor.
```

```
    School()
    {
     int id  =1234;
    }
// Declaration of a instance method.
     public void mySchool()
     {
       String schoolName = "RSVM";
     }
public static void main(String[] args)
{
// Create the object of class 'School'.
     School sc = new School();
// Calling the local Variables from outside the method and constructor.
     System.out.println("Name of School: " +schoolName); // compilation error.
     System.out.println("Id of Student: " +id); // compilation error.
   }
}
```

Scope of Instance variables in Java

➲ The scope of instance variables is inside the class. They are visible inside all the methods, constructors, and from the beginning of its program block to the end of program block in the class.

Therefore, All the methods, constructors, and blocks inside the class can access instance variables. Normally, it is recommended to make these variables private in the class.

However, the visibility of instance variables for the sub-classes can be given with the use of access modifiers.

➲ In the user-defined method (instance method), the instance variables can be accessed directly by calling the variable name inside the class.

➲ Within static methods and different classes, instance variables should be called using object reference variable. It has the following general form:

Syntax:

```
ObjectReference.VariableName;
```

Let's take an example program related to scope of instance variables.

Program source code 3:

```
package variablePrograms;

 public class Calculation

 { // Block 1

// Declaration of instance variables.

    int a = 20;

    int b = 30;

Calculation()
```

```
{ // Block 2.

   int c = 50; // Local variable.

}

void addition()

{ // Block 3.

    int x = 100;

    int add = a + b + x; // Here, variable a and b declared in block 1 are available to block
3.



    System.out.println("Sum: " +add);

}

void subtraction()

{ // Block 4.

     int sub = a + b + c; // Here, variables a and b are available to block 4 but variable c is
not available to block 4 because c is local variable.

     System.out.println("Sub: " +sub);

}

public static void main(String[] args)

{

   Calculation c = new Calculation();
```

```
   c.addition();

   c.subtraction();

 }

}
```

Output:

```
        Sum: 150

        Exception in thread "main" java.lang.Error: Unresolved compilation problem: c
cannot be resolved to a variable
```

The variable a and b declared in block 1 are visible to all blocks but variable c is visible only for block 2 because of local variable. That's why we cannot access it in block 4.

Variables are created when their scope is started, and destroyed when their scope is ended. It means that a variable defined within the block loses its value when scope is ended. Thus, the lifetime of any variable is confined to its scope.

Let's create a program where we will declare the instance and local variables with the same name and definition. Consider the following source code.

Program source code 4:

```
package variablePrograms;

 public class ScopeTest

 {

   int num = 20;

   void m1()
```

```
  {
    int num = 30;
    System.out.println("Number: " +num);
    System.out.println("Number: " +this.num);
  }
public static void main(String[] args)
{
  ScopeTest st = new ScopeTest();
  st.m1();
 }
}
Output:
          Number: 30
          Number: 20
```

In the preceding class, we have declared two variables instance and local with the same name and definition. First, instance variable num is declared with value 20, and second is local variable with the same name but with value 30. The local variable num within m1() method hides instance variable num.

Therefore, when we will call m1() method from the main() method, the output will be displayed num equal to 20, even though there is also a

num instance variable that equals to 30. In this case, we can use this.num to call instance variable. It will print the output 20.

Scope of Static variables

The scope of a static variable is within the class. All the methods, constructors, and blocks inside the class can access static variables by using the class name. It has the following general form:

Syntax:

```
ClassName.VariableName;
```

The visibility of the static variable is similar to the instance variable. However, you can declare the static variable as public so that it can be available for users of the class. You will learn more in detail in the static chapter.

Let's see an example program based on the scope of static variables.

Program source code 5:

```
package staticVariable;

 public class StaticTest

 {

// Declaration of static variable.

   static int a = 20;

   void m1()

   {

     int a = 30;
```

```
    System.out.println("a: " +a);

    System.out.println("a: " +StaticTest.a); // Accessing static variable using class name
within instance method.

}

 public static void main(String[] args)

 {

   StaticTest st = new StaticTest();

   st.m1();

 }

}

Output:

          a: 30

          a: 20
```

Hope that this tutorial has covered all the important points related to scope of variables in java with example programs. Memorize the following points related to variable scope.

Key Points:

1. Scope of a variable means simply the region of the program where a variable is accessible.

2. The scope of local variables always remains inside constructor, method, and block. It can not be accessible from outside the constructor, method, and block.

3. The scope of instance variables is within the class. They can be accessed from all the methods, constructors, and from the beginning of its program block to the end of program block in the class.

4. The scope of static variables is also within the class. All the methods, constructors, and blocks within class can access static variables by using the class name.