**SCOPE**

The *scope* of a declaration is its range within the written computer program. Inside its scope, the declared identifier represents the declared entity. Outside the scope, the identifier is either unknown or represents something different.

Scope is a concept analogous to a broadcasting station’s range. Suppose you turn your car’s FM radio on during a long road trip. Your radio is set to some frequency (say 95.5) at which a radio station is broadcasting. Drive far enough and you will eventually move out of range of that station and no longer receive it. You may even move into the range of another station which is broadcasting at the same operating frequency. In this analogy, the program is the highway, some declaration statement is the broadcasting station and the declared identifier is station’s operating frequency. As you move through the program, you can move out of range of the declaration so that the identifier becomes unknown or represents something entirely different.

Scope is typically a static property determined by the compiler at compile-time. Furthermore, in block-structured computer languages scope is generally coextensive with the block in which the declaration resides. Within that block, the entity declared is said to be *local* to it and is unknown outside it. Entities that are accessible to the entire program are said to be *global*.

---

**Example**

```java
public class ABC {
    public static void main( String args[] ) {
        while ( not done ) {
            int x;
        }
    }
}
```

Local to this block; accessible within it

Not accessible outside the block
Java is such a statically scoped, block-structured computer language. Even so, its scope rules differ subtly from those of very similar languages. Here’s a summary of Java’s scope rules.

**Program Level Scope**

A complete Java program is a collection of classes. There is no scoping structure larger than a class block.

**Example**

In a Java program, all public classes can access each other.
**Class-Wide Scope**
The following diagram shows the hierarchy of permissible Java declarations.

![Hierarchy of Permissible Java Declarations](image)

The diagram shows, for instance, that a class block can contain methods, variables and nested (inner) classes but a method block can only contain statements and variables.

In the absence of shadowing, all entities declared within a class block have scope that extends throughout the class block.

**Example**
Class field declarations are accessible to all methods within the class and any inner classes, subject to the shadowing rule.

Every method in the class is accessible to every other method in the class and to code within the inner class.

The inner class can be used by methods and field declarations in the outer class.
Java’s class-wide scope allows methods and inner classes within the same class block to share data declared as fields of the class.

Example

In application `PayWithMethods`, the three methods `inputData, calculatePay` and `printResults` share the variables `name, wage, hours` and `pay` that are declared on lines 8, 9 and 10.

```java
import static javax.swing.JOptionPane.*;
import java.util.Scanner;
import java.text.DecimalFormat;

public class PayWithMethods {
    public DecimalFormat df = new DecimalFormat( "$#,###.00" );
    public String name;       // worker's name
    public double wage, pay;  // worker's hourly wage and pay
    public int hours;         // worker's hours worked

    public static void main( String[] args )
    {
        new PayWithMethods(); // call app constructor
    }

    public PayWithMethods( ) // app constructor
    {
        inputData( );
        calculatePay( );
        printResults( );
    }

    public void inputData( )
    {
        String prompt, input;
        prompt = "Enter worker's name, wage, hours (comma separated)";
        input = showInputDialog( prompt );
        Scanner scan = new Scanner( input );
        scan.useDelimiter( "," );
        name = scan.next( );
        wage = scan.nextDouble( );
        hours = scan.nextInt( );
    }

    public void calculatePay( )
    {
        pay = hours * wage;
    }

    public void printResults( )
    {
        String output = "Pay " + name + " " + df.format( pay );
        showMessageDialog( null, output );
    }
}
```
Method and Statement Level Scope
A variable declared within a method or statement block is called a *local variable*. Its scope is restricted to the block in which it is declared. The variable is unknown outside its local block. Furthermore, within the entire method block the variable must be declared before it is used and cannot be declared more than once.

### Example
```java
public class ABC {
    public void methodOne() {
        int x, y;
    }

    public void methodTwo() {
        double x;
        while (not done) {
            double y;
        }
    }
}
```

Java’s method-level scope insures that data declarations in different methods do not conflict, even though they may be given the same identifiers.
Shadowing
In Java, a class field can be *shadowed out* of a method or inner class by declaring the same identifier within the inner block. By doing so, you create a local variable or inner-class field that has the same name as the one in the outer block. Code inside the inner block assumes the local variable by default.

**Example**

```java
public class TestFraction {
  
  public TestFraction() {
    Fraction f = new Fraction(-1, 4);
    System.out.println(f.toString() + " = " + f.toDecimal());
  }

  public class Fraction { // inner class
    private int numerator, denominator;

    public Fraction(int n, int d) {
      numerator = n;
      denominator = d;
    }

    public String toString() {
      return "(" + numerator + "/" + denominator + ")";
    }

    public double toDecimal() {
      return (double)numerator / denominator;
    }
  }

  public static void main(String[] args) {
    new TestFraction();
  }
}
```