NAMING AND SCOPE RULES IN JAVA

When a variable declaration is executed, memory is allocated for that variable. The variable name can be used in at least some part of the program source code to refer to that memory or to the data that is stored in the memory. The portion of the program source code where the variable name is valid is called the scope of the variable. Similarly, we can refer to the scope of subroutine names and formal parameter names.

For static member subroutines, scope is straightforward. The scope of a static subroutine is the entire source code of the class in which it is defined. That is, it is possible to call the subroutine from any point in the class, including at a point in the source code before the point where the definition of the subroutine appears. It is even possible to call a subroutine from within itself. This is an example of something called "recursion," a fairly advanced topic that we will return to in Chapter 9.

For a variable that is declared as a static member variable in a class, the situation is similar, but with one complication. It is legal to have a local variable or a formal parameter that has the same name as a member variable. In that case, within the scope of the local variable or parameter, the member variable is hidden. Consider, for example, a class named `Game` that has the form:

```
public class Game {

    static int count;  // member variable

    static void playGame() {
        int count;  // local variable
          .
          .   // Some statements to define playGame()
          .
    }
```

```
    .
    .   // More variables and subroutines.
    .

}  // end Game
```

In the statements that make up the body of the `playGame()` subroutine, the name "`count`" refers to the local variable. In the rest of the `Game` class, "`count`" refers to the member variable (unless hidden by other local variables or parameters named `count`). However, there is one further complication. The member variable named `count` can also be referred to by the full name `Game.count`. Usually, the full name is only used outside the class where `count` is defined. However, there is no rule against using it inside the class. The full name, `Game.count`, can be used inside the `playGame()` subroutine to refer to the member variable instead of the local variable. So, the full scope rule is that the scope of a static member variable includes the entire class in which it is defined, but where the simple name of the member variable is hidden by a local variable or formal parameter name, the member variable must be referred to by its full name of the form **className**.**variableName**. (Scope rules for non-static members are similar to those for static members, except that, as we shall see, non-static members cannot be used in static subroutines.)

The scope of a formal parameter of a subroutine is the block that makes up the body of the subroutine. The scope of a local variable extends from the declaration statement that defines the variable to the end of the block in which the declaration occurs. As noted above, it is possible to declare a loop control variable of a `for` loop in the `for` statement, as in "`for (int i=0; i < 10; i++)`". The scope of such a declaration is considered as a special case: It is valid only within the `for` statement and does not extend to the remainder of the block that contains the `for` statement.

It is not legal to redefine the name of a formal parameter or local variable within its scope, even in a nested block. For example, this is not allowed:

```
void  badSub(int y) {
    int x;
    while (y > 0) {
       int x;  // ERROR:  x is already defined.
         .
         .
         .
    }
 }
```

In many languages, this would be legal; the declaration of `x` in the `while` loop would hide the original declaration. It is not legal in Java; however, once the block in which a variable is declared ends, its name does become available for reuse in Java. For example:

```
void goodSub(int y) {
   while (y > 10) {
      int x;
        .
        .
        .
      // The scope of x ends here.
   }
   while (y > 0) {
      int x;  // OK: Previous declaration of x has expired.
       .
       .
       .
   }
}
```

You might wonder whether local variable names can hide subroutine names. This can't happen, for a reason that might be surprising. There is no rule that variables and

subroutines have to have different names. The computer can always tell whether a name refers to a variable or to a subroutine, because a subroutine name is always followed by a left parenthesis. It's perfectly legal to have a variable called `count` and a subroutine called `count` in the same class. (This is one reason why I often write subroutine names with parentheses, as when I talk about the `main()` routine. It's a good idea to think of the parentheses as part of the name.) Even more is true: It's legal to reuse class names to name variables and subroutines. The syntax rules of Java guarantee that the computer can always tell when a name is being used as a class name. A class name is a type, and so it can be used to declare variables and formal parameters and to specify the return type of a function. This means that you could legally have a class called `Insanity` in which you declare a function

```
static  Insanity  Insanity( Insanity Insanity ) { ... }
```

The first `Insanity` is the return type of the function. The second is the function name, the third is the type of the formal parameter, and the fourth is the name of the formal parameter. However, please remember that not everything that is possible is a good idea!

Source : http://math.hws.edu/javanotes/c4/s7.html