**Scope Rules for Variables**

The idea of scope in programming languages is all about the access and lifetimes of variable names and definitions.

In Java, as in many languages based on C syntax, any variable declaration only has the extent of the immediately enclosing block, where a block is indicated by opening and closing curly braces "{" and "}". Informally, we use the terms "local" and "global" to give the relationship between a variable and a block, i.e. a variable is **local** to a block if it is declared in that block and it is **global** to a block if it is declared outside that block.

So a class definition is a block and any variable declared here can be used throughout the class:

```
public class MyClass
{
  // field variables
  int number;

  // rest of class is the scope of variables

}
```

A method definition is a block and any variable declared within the definition can be used throughout the method, but not outside the method.

```
public void MyMethod ()
{
  // variables local to this method
  int number;

  // rest of method is the scope of variables
}
```

And any block introduced in the code by curly braces can have the same effect.

```
while (condition)
{
  // variables local to the body of the while loop
  int number;

}
```

This is also true for the index variable declared in the header of the for loop:

```
for (int i;  i < length;  i++ )
{
  // use the index variable i inside this loop, but not outside
}
```

If you have already declared a variable at a certain scope level, for example, at the class level, and you make another declaration of the same name locally, then the first variable is no longer accessible. It is covered up by the second local declaration.

**Controlling Access to Class Variables and Methods**

In a class definition, we sometimes want to make variables and methods accessible outside the normal scope of the class definition. But this can be controlled by the scope keywords, also called the access keywords.

- **public** – the variable or method can be used by any other program outside the class.
- **<no keyword>** - when the keyword is omitted, this is "package-private" and the variable or method can be used anywhere inside this package, except for subclasses
- **protected** – the variable or method can be used anywhere inside the package, including subclasses
- **private** – the variable or method can only be used inside this class