All identifiers have an associated scope

- The **scope** of a name is the region of program code where that name is visible
- A name **cannot be accessed** outside its scope
- Within a **method**:
  - Braces ({}) mark closed regions of program statements
  - A **local variable** is only visible from the point of its declaration until the closing brace enclosing it

```java
public void thisMethod()
{
    int variable1 = ...;
    while ( notDone )
    {
        int variable2 = ...;
    }
}
```
Variables with the same name can be confusing

Let’s look at a code example . . .
Class-level declarations are visible more widely

- Fields and methods within a class are visible everywhere in the class (regardless of order of declaration)

- They can also be given a visibility modifier
  - Members of a class that are declared with public visibility can be accessed from anywhere
  - Public fields should be avoided (violates encapsulation)
  -Members of a class that are declared with private visibility can only be accessed from inside the class
There are two more visibility modifiers

- We’ll use **public** and **private** in this course, but ...

  - Members of a class that are declared with **protected** visibility can only be accessed within subclasses or by other classes in the same package
    - Used for designing flexible inheritance hierarchies

  - Members declared without a visibility modifier have “default” visibility and are accessible in any class in the same package
    - Used for designing collections of inter-related classes in a package
Public vs. private determines what a class exports

- We use visibility modifiers to determine which parts of a class are part of the external interface other objects can access, and which are for internal use only

  - **public** methods are services that other objects can call on

  - **private** methods can only be accessed within the class itself, not from outside

- Let’s look at a code example . . .
Expressions are the way we combine values

- An **expression** is a combination of one or more operands and their operators
- **Arithmetic expressions** compute numeric results and make use of the arithmetic operators:

  - Addition +
  - Subtraction -
  - Multiplication *
  - Division /
  - Remainder %

- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point value
Using the division and remainder operations

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

  \[
  14 \div 3 \quad \text{equals?} \quad 4 \\
  8 \div 12 \quad \text{equals?} \quad 0
  \]

- The remainder operator (%) returns the remainder after dividing the second operand into the first

  \[
  14 \% 3 \quad \text{equals?} \quad 2 \\
  8 \% 12 \quad \text{equals?} \quad 8
  \]
Operator precedence governs which is performed 1st

- Operators can be combined into complex expressions

  \[ \text{result} = \text{total} + \frac{\text{count}}{\text{max}} - \text{offset}; \]

- Operators have a well-defined **precedence** which determines the order in which they are evaluated

- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation

- Arithmetic operators with the same precedence are evaluated from **left to right** (op. associativity)

- Parentheses can be used to force the evaluation order
The assignment operator has a lower precedence than the arithmetic operators.

First the expression on the right hand side of the = operator is evaluated:

```
answer = sum / 4 + MAX * lowest;
```

Then the result is stored in the variable on the left hand side.
Execution flows from one method to another

- The called method can be within the same class, in which case only the method name is needed.
When designing methods, think about their use

- Public methods describe the services that this class provides to other classes.
  - Public methods are often called **service methods**.

- Methods that simply return information about an object are called **accessor methods** (they allow other objects to access information about this object’s state).

- Methods that modify an object are called **mutator methods** (they allow other objects to change or mutate this one).

- Private methods are often called **support methods** or **helper methods**, because they help perform the work provided by service methods.
Control also flows between classes and back

- The called method can be part of another class or object
Method Decomposition

- A method should be **relatively small**, so that it can be understood as a single entity
  - No longer than a page (about 40 lines)
  - A better goal: fits completely on the screen at once (about 20 lines)
- A potentially large method should be decomposed into several smaller methods as needed for clarity
- A service method of an object may call one or more support methods to accomplish its goal
- Support methods could call other support methods if appropriate