Assertions

An *assertion* is a true-false statement, a boolean expression, about the values of the variables in a program. In the sequence to the left below, an assertion has been placed in a comment on the second line. The assertion is true at that point. In the middle is the same sequence of statements but with the assertion slightly different. This assertion is also true at that point. The assertion in the rightmost sequence is false; we shouldn't write such things.

```
x= 5;              x= 5;              x= 5;
// x  =  5         // x ≥ 3           // x < 5
x= x + 2;          x= x + 2;          x= x+1;
```

We put assertions within a program at various places to help the reader understand the program. When we place an assertion in a program, we are *asserting* that the assertion is true at that point. Hence, the name *assertion*.

What language do we use for assertions?

The assertions in the examples above are written in mathematics. When placed in "assert statements" in Java, they *have* to be written in Java. But often, when we write them as comments in a program, they may be written in math, English, or a combination of both. The important point is not the language but the clarity and precision with which assertions are written. Here are some assertions we might place in programs.

```
x is the sum of 1..k
array segment b[i..j] is sorted (in ascending order)
b[0..h-1] ≤ 0       (meaning that every element of b[0..h-1] is at most zero)
b[0..h-1] ≤ 0  &&  b[h..] > 0
```

Later on, we will show how we can use pictures of arrays as assertions.

Preconditions and postconditions

The *precondition* of a statement is the assertion that precedes it. The *postcondition* of a statement is the assertion that follows it. In the sequence:

```
x= 5;
// x  =  5
x= x + 2;
```

the statement x= 5; does not have a precondition and has the postcondition x = 5. The statement x= x+2; has the precondition x = 5 and has no postcondition.

The Hoare triple

In 1969, Sir Tony Hoare (Hoare was knighted by the queen of England for his many contributions to computer science and especially to his work on correctness of programs) introduced the following notation, now called a *Hoare-triple*, where P (the precondition) and Q (the postcondition) are assertions and S is a statement (or a sequence of statements):

```
{P}   S   {Q}
```

This notation is itself a true-false statement with this meaning: If S is executed in a state in which P is true, then S is guaranteed to terminate, and when it does, Q will be true.

For example, you can see that the following Hoare-triple is true:

```
{x = 5}   y = 2*x;   {x = 5  &&  y == 10}
```

Some people put braces around an assertion in a comment in order to distinguish assertions from other kinds of comments. We will not follow that practice, writing the Hoare triple instead as shown to the right.

```
// {x = sum of 1..k}          // x = sum of 1..k
k= k+1;  x= x + k;            k= k+1;  x= x + k;
//{x = sum of 1..k}           //x = sum of 1..k
```

An important point about the Hoare triple

If precondition P is false, then the Hoare-triple guarantees nothing about execution, so execution of S can do anything! It can get into an infinite loop; it can stop immediately, with no guarantees about what state it is in; it can abort in some fashion, perhaps by dividing by 0. That's because of its meaning: "if S is executed in a state in which P is true, then …" That sentence says nothing about what will happen if P is not true.

Preconditions of methods

In Java, we put preconditions in method specifications. For example, in this specification, array b should not contain 0 values because the average of 0 values is undefined.

```
/** Return the average value of array b.
  * Precondition: b is not empty —it contains at least one value.  */
public static int average(int[] b) {
   Store the average of b in local variable av;
   return av;
}
```

The precondition is actually the precondition in this Hoare-triple:

```
{b is not empty}
Store the average of b in local variable av;
{av is the average of array b}

return av;
```

As we have stated earlier in this course, if a method is called with the precondition being false, the method can do whatever it wants. This goes right along with the notion of the Hoare-triple.

However, in many cases, we can help debug Java programs more easily by placing assert statements at the beginning of methods to catch false preconditions. In this case, we can put at the beginning of S the statement

```
assert b.length != 0
```

In summary, if the precondition is false, we can do anything we want in the method body, and, often, we decide to do something to help us catch errors more quickly.