CS100J 12 March 2006

More on the loops and assertions

Start reading chapter 7 on loops.

The lectures on the ProgramLive CD can be a big help.

Also: drawing frames for method calls: see pp. 93-94

Course website contains an "assignment" that you can do to get practice, with answers. You need not hand it in.

"O! Thou hast damnable iteration and art, indeed, able to corrupt a saint." Shakespeare, *Henry IV*, Pt I, 1 ii

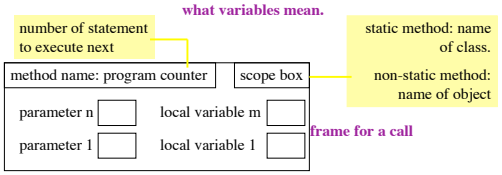
"Use not vain repetition, as the heathen do."
Matthew V, 48

Your "if" is the only peacemaker; much virtue if "if".
Shakespeare, *As You Like It*.

1

Executing method calls, pp 93-94

Understanding this not only prepares you for prelim 2, it helps you understand how recursion can work and how a method determines what variables mean.



Execution of method call
1. Draw a frame for the call.
2. Assignment arg values to the pars
3. Execute method body
4. Erase frame, and, for a function, return value of the **return** expression.

In step 3, look in frame for variables/methods. If not there, look in place given by scope box.

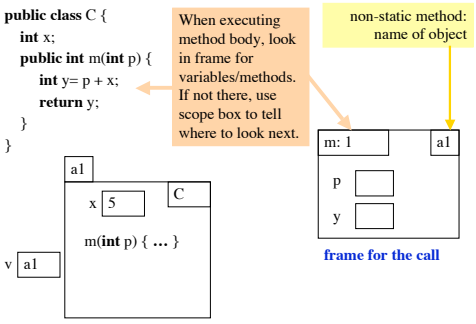
2

Executing method calls, pp 93-94

```
public class C {
   int x;
   public int m(int p) {
      int y= p + x;
      return y;
   }
}
```

When executing method body, look in frame for variables/methods. If not there, use scope box to tell where to look next.

non-static method: name of object



v.m(6) evaluate this expression

3

The for loop: syntax. See top of page 78

for (<*initialization*> ; <*condition*> ; <increment>)
<*repetend*>

```
// Loop to process a range b..c of integers
for (int k= b ; k <= c ; k= k+1  ) {
   Process k
}
// Print k*k for k in the range 5..19
for (int k= 5;  k <= 19; k= k+1) {
    System.out.println(k*k);
}
```

When faced with a problem that might require a loop that processes a range of integers:

1. Identify the range of integers (say, b..c).
2. Write the loop as shown to the left.
3. Then figure out how to process k.

Separate your concerns. Focus on one thing at a time.

4

Understanding assertions. Read p. 75 and look at style notes.

An assertion is a true-false statement about the variables used in a program. It is usually placed in the program at places where we expect it to be true.

```
int x= 0;
// { x is the sum of 0..0 }
x= x + 1;
// { x is the sum of 0..1 }
x= x + 2;
// { x is the sum of 0..2 }
x= x + 3;
// { x is the sum of 0..3 }
```

```
int x= 0;  int k= 1;
// { x is the sum of 0..k–1 }
x= x + k;  k= k + 1;
// { x is the sum of 0..k–1 }
x= x + k;  k= k + 1;
// { x is the sum of 0..k–1 }
x= x + k;  k= k+1;
// { x is the sum of 0..k–1 }
```

x ☐ k ☐

The assertion "x is the sum of 0..k–1" is invariantly true.

"invariant" means "unchanging".

5

Understanding assertions

An assertion is a true-false statement about the variables used in a program. It is usually placed in the program at places where we expect it to be true.

```
int x= 0;  int k= 1;
// { x is the sum of 0..k–1 }
x= x + k;  k= k + 1;
// { x is the sum of 0..k–1 }
x= x + k;  k= k + 1;
// { x is the sum of 0..k–1 }
x= x + k;  k= k + 1;
// { x is the sum of 0..k–1 }
```

```
// Set x to sum of 1..3
// { invariant: x is sum of 0..k–1}
for (int k= 1; k <= 3; k= k + 1) {
        x= x + k;
}
```

x ☐ k ☐

The assertion "x is the sum of 0..k–1" is invariantly true.

"invariant" means "unchanging".

6

The invariant of a loop that processes a range

```
// Set x to sum of 1..3
 // { invariant: x is sum of 0..k–1}
for (int k= 1; k <= 3; k= k + 1) {
        x= x + k;
}
```

The invariant tells you something about the integers 0..k–1 that have been processed. It is true before and after each iteration of the loop —just before and after the loop condition is evaluated.

```
// { inv: b..k-1 have been processed }
for (int k= b; k <= c; k= k+1) {
   Process k;
}
```



```
// Print squares of ints in range m..n

// { inv:                                  }
for (int k= m; k <= n; k= k + 1) {
      // Process k;

}
// {squares of ints in range m..n have been printed }
```

1. What is the invariant?
2. Is any initialization needed?
3. How is k to be processed?

```
// Store in t a copy of string s but with a blank inserted after each char

// { inv:                                                                  }
for (int k= 0; k < s.length(); k= k + 1) {
      // Process k;

}
// { t = s[0..s.length()–1] but with a blank inserted after each char }
```



```
// { n >= 2 }    —we take this as true at this point
b= true;
// Store false in b if some integer in 2..n–1 divides n

// { inv:                                                  }
for (int k= 2;  k <= n–1;  k= k + 1) {
      // Process k;


}
// { b is false iff some integer in 2..n–1 divides n }
```

1. What is the invariant?
2. Is any initialization needed?
3. How is k to be processed?


```
// { String s has at least 1 character }
// Set c to the largest character in String s


// { inv:                                                  }

for (int k= 0;  k < s.length();  k= k + 1) {
      // Process k;


}
// { c is the largest character in s[0..s.length()–1] }
```

1. What is the invariant?
2. Is any initialization needed?
3. How is k to be processed?