Definite Loops, Nested Loops, Managing Complexity

CSCI 161 – Introduction to Programming I

Professor Thomas Rogers

Overview

- Reading: Chapter 2 - sections 2.3, 2.4
- Topics:
  - Definite Loops
  - Nested Loops
  - Managing Complexity

Definite Loops

- Programming often requires redundant tasks.
  The **for** loop helps to avoid such redundancy by repeatedly executing a sequence of statements over a particular range of values.
- The general syntax for the **for** loop is:

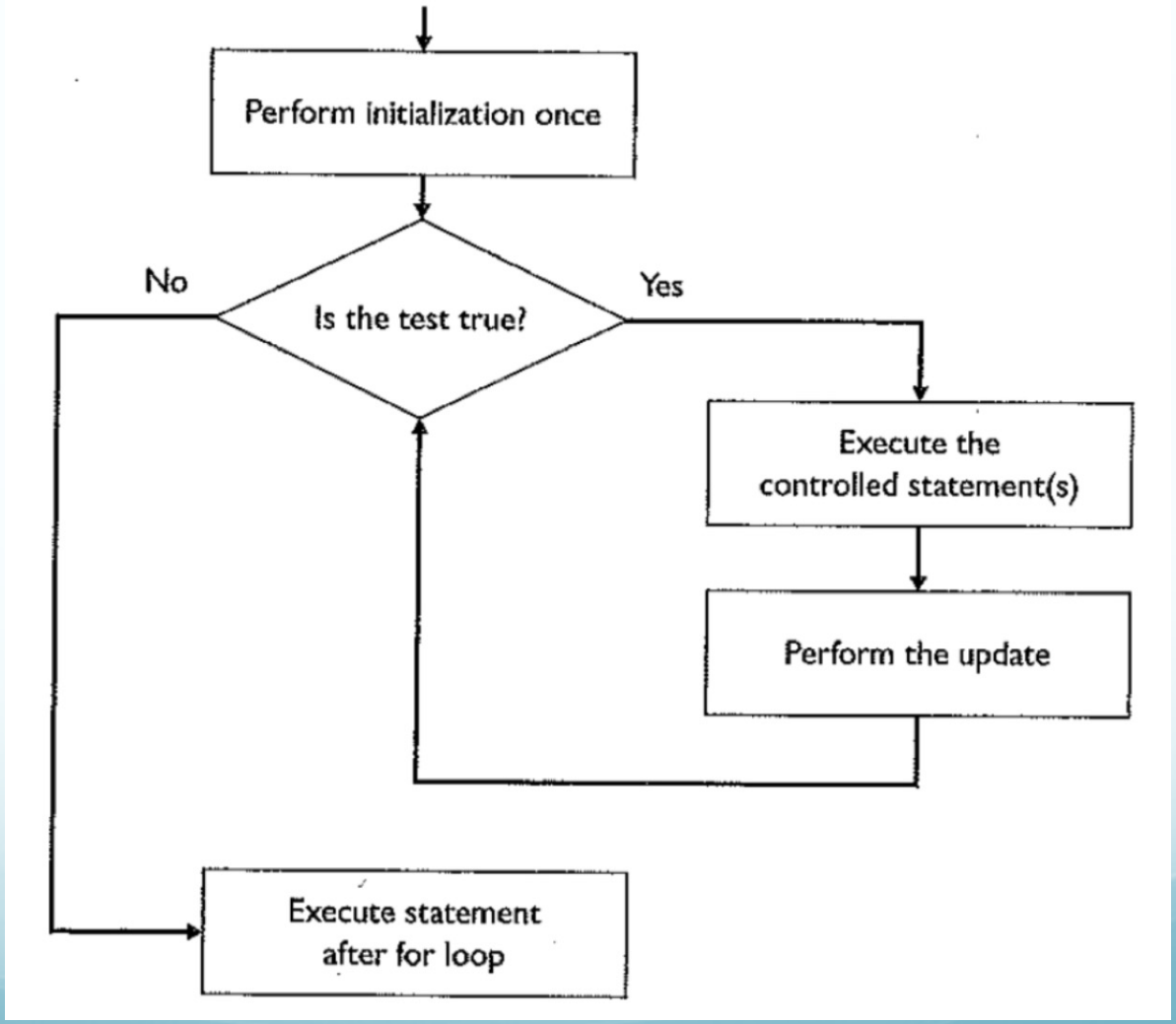
```
for (<initialization>; <continuation test>; <update>) {
    <statement>
    <statement>
    ...
    <statement>
}
```

Definite Loops (continued)

- Loop Constructs:
  - ***initialization*** – The statement that declares and initialized the loop variable and is executed only once before the loop starts.
  - ***continuation test*** – The logical expression that is evaluated *before* each execution of the loop allowing the loop to execute only if it evaluates to *true*.
  - ***update*** – The statement that updates the loop variable, executed after each loop execution and before the next ”continuation test.”

Definite Loops (continued)

- Diagram:



Nested Loops

- For loops can be nested and such is often the case when dealing with things in two dimensions.

```
for (int i=1; i<=10; i++) {
    for (int j=1; j<=10; j++) {
        System.out.printf("%3d ", i*j);
    }
   System.out.println();
}
```

- **Warning:** - Be careful not to mix up the loop variables in nested loop code (i, j not so great choices).
- **Note:** - Note the use of the **printf** for formatted output. A very good tutorial here

Managing Complexity

- Brian Kernighan, co-author of *The C Programming Language* said that "Controlling complexity is the essence of computer programming."
- **Scope** - The part of a program where a particular declaration is valid.
- **Local Variable** - A variable declared inside a method that is accessible only in that method.
- **Localizing Variables** - The action of declaring variables in the innermost (most local) scope possible.
- **Global Variable** - (aka Class Variable) In the case of Java class, these are public static variables declared within the class (not within methods), typically at the top of your program.

Managing Complexity (continued)

- **Class Constant** - (aka Constant) Like a global variable, but with the keyword **final** included and conforming to the **ALL_CAPS** identifier convention.

```
public static final int MAX_LEN = 255; // Max line length
```

Managing Complexity (continued)

- **Pseudocode** - English-like descriptions of algorithms. Use pseudocode to describe your program then break down each step into further pseudocode that maps to methods that then break down into more pseudocode then into actual code.

- For example, pseudocode start with a general problem statement then gets successively broken down into an increasing number of more detailed steps:
  1. Figure out the amount of liner material needed for a swimming pool.

1. First becomes:
   1. Calculate the linear material needed for *different types* of swimming pools.

Managing Complexity (continued)

- Then becomes:
  1. Determine the type of pool: round above-ground, or rectangular in-ground.
  2. Depending on the type ask for dimensions.
  3. Calculate the linear material needed based on type and dimensions.

- Then becomes:
  1. Prompt user for type of pool: A. for round above-ground, B. for rectangular in-ground.
  2. If user entered A then ask for diameter and height, each in feet with fractional feet in lieu inches (4' 6" = 4.5 feet).
  3. Else if user entered B then ask for length, width and height of in feet with fractional feet in lieu of inches.
  4. If A then calculate material needed as: wall area + floor area
  5. Else if B then calculate material needs as area of each wall + floor area

Managing Complexity (continued)

- Then becomes:
  1. declare needed variables
  2. poolType = getPoolType()
  3. if (poolType == 'A') then
     i. height = getPoolHeight()
     ii. radius = getPoolDiameter() / 2.0
     iii. material = calcPoolMaterial(height, radius)
  4. else
     i. height = getPoolHeight()
     ii. length = getPoolWallLength()
     iii. width = getPoolWallWidth()
     iv. material = calcPoolMaterial(height, length, width)
  5. printMaterialNeeded(material)

Sample Program

- Sample is here (click on page, cut and paste)