Iteration

Lecture 8 - A
Object-Oriented Programming

Agenda

- While Loop
- Infinite Loops
- Block Statements in Loops
- While Loop for Input Validation
- Do-while Loop
- For Loop
- Sentinel Values
- Nested Loops
- Break Statement
- Continue Statement
- Deciding Which Loops to Use
- Boolean Flags
- Empty Intervals
- Common Loop Errors
- Recursion vs. Iteration

`while` Loop

- Java provides three different looping structures.
- The while loop has the form:

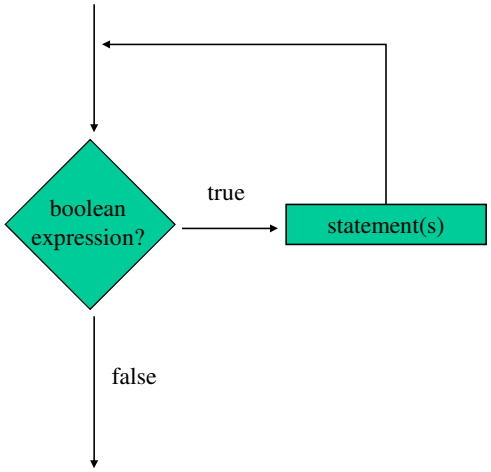
```
while(condition){
    statements;
}
```

- While the condition is true, the statements will execute repeatedly.
- The while loop is a ***pretest*** loop, which means that it will test the value of the condition prior to executing the loop.

`while` Loop

- Care must be taken to set the condition to false somewhere in the loop so the loop will end.
- Loops that do not end are called *infinite loops.*
- A while loop executes 0 or more times since if the condition is false, the loop will not execute.

`while` loop Flowchart



Infinite Loops

- In order for a while loop to end, the condition must become false.

```
{
    int x = 20;
    while(x > 0){
        System.out.println("x is greater than 0");

    }
}
```

- The variable x never gets decremented so it will always be greater than 0.

Infinite Loops

- In order for a while loop to end, the condition must become false.

```
{
    int x = 20;
    while(x > 0){
        System.out.println("x is greater than 0");
        x--;
    }
}
```

- The variable x never gets decremented so it will always be greater than 0.
- Adding the **x--** above fixes the problem.

Block Statements in Loops

- Although not required for single statement while loops, convention holds that while loops always use curly braces.
- Curly braces are required to enclose block statement while loops. (like block if statements)

```
while(condition){
  single or block statements;
}
```

`while` Loop for Input Validation

- *Input validation* is the process of ensuring that user input is valid.

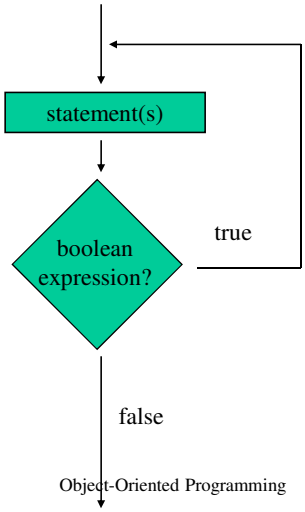
```
System.out.print("Enter a number in the "
                 + "range of 1 through 100: ");
number = Keyboard.readInt();
// Validate the input.
while (number < 1 || number > 100)
{
  System.out.println("That number is invalid.");
  System.out.print("Enter a number in the "
                   + "range of 1 through 100: ");
  number = Keyboard.readInt();
}
```


The `do-while` Loop

- The do-while loop is a *post-test* loop, which means it will execute the loop prior to testing the condition.
- The do-while loop, more commonly called a do loop, takes the form:

```
do{
  statements
}while(condition);
```

`do-while` Loop Flowchart

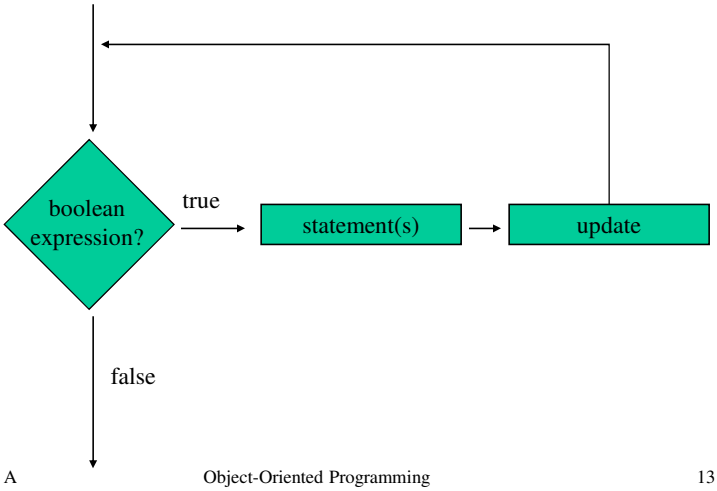


`for` Loop

- The for loop is a specialized form of the while loop, meaning it is a pre-test loop.
- The for loop allows the programmer to initialize a control variable, test a condition, and modify the control variable all in one line of code.
- The for loop takes the form:

```
for(initialization; test; update)
{
    loop statements;
}
```

`for` Loop Flowchart



Sections of The `for` Loop

- The initialization section of the for loop allows the loop to initialize its own control variable.
- The test section of the for statement acts in the same manner as the condition section of a while loop.
- The update section of the for loop is the last thing to execute at the end of each loop.

`for` Loop Initialization

- The initialization section of a for loop is optional; however, it is usually provided.
- Typically, for loops initialize a counting variable that will be tested by the test section of the loop and updated by the update section.
- The initialization section can initialize multiple variables.
- Variables declared in this section have scope only for the for loop.

Update Expression

- The update expression is usually used to increment or decrement the counting variable(s) declared in the initialization section of the for loop.
- The update section of the loop executes last in the loop.
- The update section may update multiple variables.
- Each variable updated is executed as if it were on a line by itself.

Modifying The Control Variable

- It is bad programming style to update the control variable of a for loop within the body of the loop.
- The update section should be used to update the control variable.
- Updating the control variable in the for loop body leads to hard to maintain code and difficult debugging.

Multiple Initializations and Updates

- The for loop may initialize and update multiple variables.

```
for(int i = 5, int j = 0; i < 10 || j < 20; i++, j+=2){
    loop statements;
}
```

- Note that the only parts of a for loop that are mandatory are the semicolons.

```
for(;;){
   loop statements;
}//infinite loop.
```

- If left out, the test section defaults to true.

Sentinel Values

- Sometimes (usually) the end point of input data is not known.
- A *sentinel value* can be used to notify the program to stop acquiring input.
- If it is a user input, the user could be prompted to input data that is not normally in the input data range (i.e. –1 where normal input would be positive.)
- Programs that get file input typically use the end-of-file marker to stop acquiring input data.

Nested Loops

- Like if statements, loops can be nested.
- If a loop is nested, the inner loop will execute all of its iterations for each time the outer loop executes once.

```
for(int i = 0; i < 10; i++)
    for(int j = 0; j < 10; j++)
        loop statements;
```

- The loop statements in this example will execute 100 times.

`break` Statement

- The break statement can be used to abnormally terminate a loop.
- The use of the break statement in loops bypasses the normal mechanisms and makes the code hard to read and maintain.
- It is considered bad form to use the break statement in this manner.

`continue` Statement

- The continue statement will cause the currently executing iteration of a loop to terminate and the next iteration will begin.
- The continue statement will cause the evaluation of the condition in while and for loops.
- Like the break statement, the continue statement should be avoided because it makes the code hard to read and debug.

Deciding Which Loops to Use

- The while loop:
  - Pretest loop
  - Use it where you do not want the statements to execute if the condition is false in the beginning.
- The do-while loop:
  - Post-test loop
  - Use it where you want the statements to execute at least one time.
- The for loop:
  - Pretest loop
  - Use it where there is some type of counting variable that can be evaluated.

`boolean` Flags

- A boolean flag is a boolean variable that denotes a condition (e.g., `done`, `working`, `available`)
  - set in one place, tested in another
- Boolean flags can also be used as the loop condition
- Example (implementing a `for` loop, using `while`):

```
boolean done = false;
int i = 0;
while (!done) {
  i++;
  if (i == 5)
   done = true;
}
```

- Notice that boolean flag is set within loop
  - in previous slides all checking was done through delegation, here we do it ourselves

Empty Intervals

```
public int sum() {
     int temp_sum = 0;
     for (int i = 1; i < 1; i++)
       temp_sum += i;

     return temp_sum;
   }
```

- Answer: Body of loop is not executed
- Why?
  - boolean is **`false`** for initial value of counter

Empty Intervals

- Correct example:

```
/* This method sums all numbers from 1 up to
   and including the number specified */

public int sum(int number) {
  int temp_sum = 0;
  for (int i = 1; i <= number; i++)
    temp_sum += i;

  return temp_sum;
}
```

Off by One Errors

- Occur when loop executes one too many or one too few times
- Example: Add even integers from 2 to number, inclusive

```
...
count = 2;
result = 0;
while (count < number) {
  result += count;
  count += 2;
}
```


Off by One Errors

- Produces incorrect result if **`number`** is assigned an even value. Values from *2* to **`number`**-*2* will be added (i.e., **`number`** excluded)
- Should be:

```
while (count <= number) {
...
}
```

- Now, value of **`number`** is included in summation

Other Loop Errors

- Make sure test variables have proper values before loop is entered

```
...
product = 0;
do {
  product *= 2;
} while (product < 100);

/* what will happen here? */
```

- Make sure tests check proper conditions

```
...
for (int i = 1; i != 100; i += 2) {
  // do something here
}

/* will we ever get here? */
```


Loops and Recursion

- Loops and Recursion Could be Used in Conjunction with Each Other.
- Koch's Snowflake is one example

Recursion vs. Iteration

- Choice between simple recursion and iteration is one of modeling
  - recursion is often less efficient because of more method calls (each activation record takes up some of computer's memory)
  - recursion is more concise and more elegant for tasks that are "naturally" self-similar
    - like Towers of Hanoi!

Reading

Book Name: Object Oriented Programming in Java™
Author: Richard L.Halterman
Content: Chapter # 17 & 18

Acknowledgements

- While preparing this course I have greatly benefited from the material developed by the following people:
  - Andy Van Dam (Brown University)
  - Mark Sheldon (Wellesley College)
  - Robert Sedgewick and Kevin Wayne (Princeton University)
  - Mark Guzdial and Barbara Ericsson (Georgia Tech)
  - Richard Halterman (Southern Adventist University)