Programming 1

Week 07 – Looping

Java Loops

- When programming, it is often necessary to repeat a selected portion of code a specific number of times, or until some condition occurs.
- Loops is part of a program that is repeated over and over, until a specific goal is reached
- We will look here at the **`FOR`** , **`WHILE`** and **`DO WHILE`** loop constructs that are available in most programming languages.

The `while` Loop

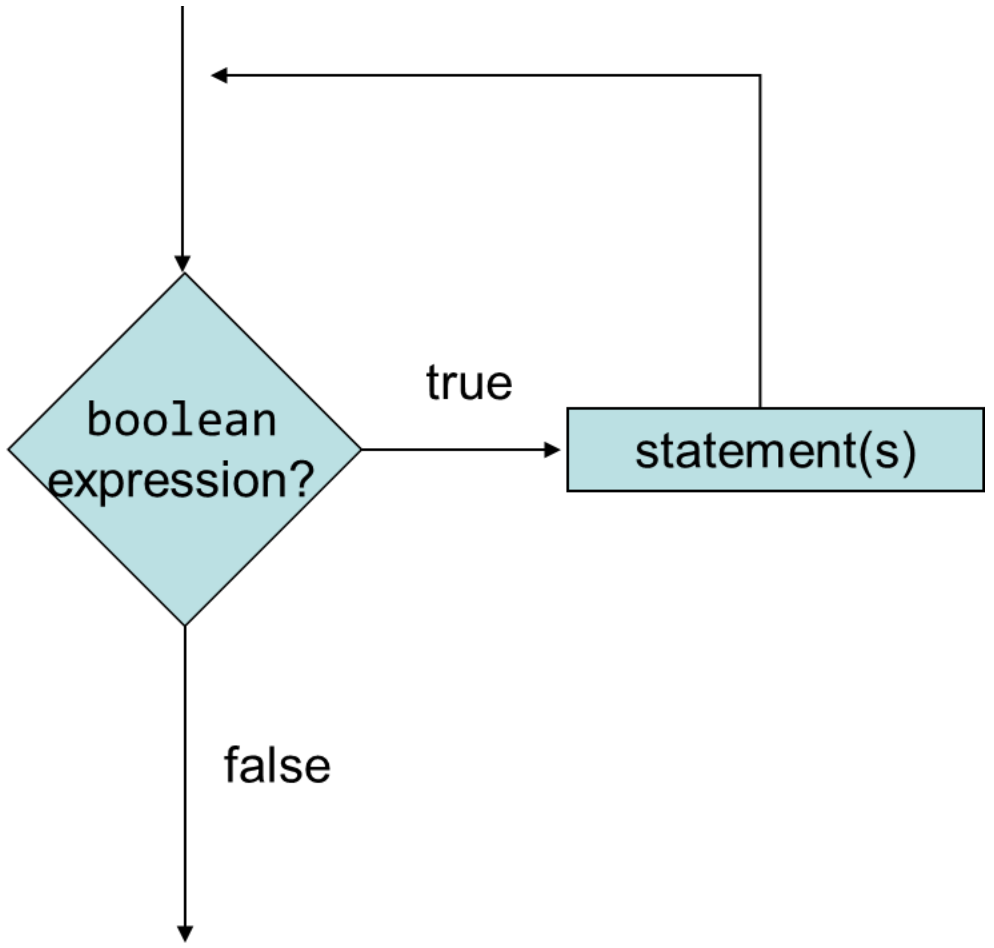
- The ***while*** statement is the most important because it is the most general-purpose statement. It can be used in *any* situation where a loop is needed.
- The `while` loop has the form:

```
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.

The `while` Loop Flowchart

- ✓ The process flow involves the verification of the Boolean expression.
- ✓ If the condition is true, the next step includes statements, which again involves the decision-making step.
- ✓ If the condition is false, the loop is exited.



The **`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. If the condition is false, the loop will not execute.

```
int number = 1;
while (number <= 5)
{
        System.out.println("Hello");
        number++;
}
```

Infinite Loops (1 of 2)

- In order for a `while` loop to end, the condition must become false. The following loop will not end:

```
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.
- Adding the **`x--`** above fixes the problem.

Infinite Loops (2 of 2)

- This version of the loop decrements x during each iteration:

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

Block Statements in Loops

- Curly braces are required to enclose block statement while loops. (like block `if` statements)

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

Number of Iterations

- What is the minimum/ maximum number of iterations for the following while loops?

  - The minimum number of iterations is 0. The loop never runs. The test condition is False right away on the very first test.

```
int num = 4;
while (num < 0)
     num = num + 1;
```

- The test condition never becomes False, so the loop keeps running. This is known as an *infinite loop*. The maximum number of iterations is infinity.

```
     int  num = 4;
     While( num  > 0)
        num = num  + 1;
```

The `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.nextInt();
// 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.nextInt();
}
```

Demo

- A class of 10 students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Write a java program to calculate the class average on the quiz. The class average is equal to the sum of the grades divided by the number of students. The algorithm for solving this problem on a computer must input each of the grades, calculate the average and print the result. Validate each grade entered so that it is between 0 and 100

```
 Scanner input = new Scanner(System.in);
 int numberOfStudent = 10;// number of student
 int grade; //student's grade
 int sum = 0; // to get the sum of the grades
 double average; // to calculate the average
 int counter = 1;
 while (counter <= numberOfStudent) {// begin of while
     System.out.print("Please enter studemt's grade: ");
     grade = input.nextInt();
     if (grade >= 0 && grade <= 100) {
         sum = sum + grade;
         counter++;
     } else {
         System.out.println("Please insert a valid grade!");
     }

 }//end of while
 average = sum / numberOfStudent;
System.out.println("The sum of the grades is: " + sum);
System.out.println("The Average is: " + average);
```

The `do-while` Loop

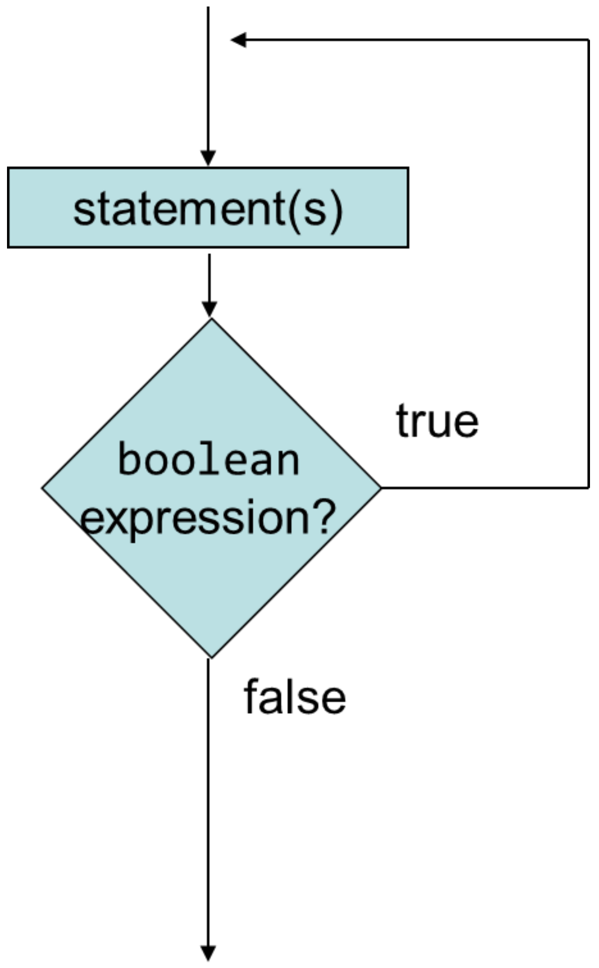
- The `do-while` loop is a **post-test** loop, which means it will execute the loop prior to testing the condition.
- The `do-while` loop (sometimes called called a do loop) takes the form:

```
do
{
  statement(s);
} while (condition);
```

The parentheses are required, and a semi-colon *is* required after the closing parenthesis (since this ends the statement)

The `do-while` Loop Flowchart

1. The loop body is executed.
2. The boolean expression is evaluated.
3. If it is true, then the process is repeated from step 1 (i.e., the loop body is executed and the boolean expression is evaluated ; if true, then the process is repeated, etc.)
4. Otherwise (if it is false), the loop is *terminated*, and control passes to the statement following the *while*.
5. Since the boolean expression is tested only *after* the loop body has been executed, the loop body will always be done at least one time in a do-while loop.



Demo

- Write a do-while loop that asks the user to enter two numbers. The numbers should be added, and the sum displayed. The loop should ask the user whether he or she wishes to perform the operation again. If so, the loop should repeat; otherwise, it should terminate.

```java
int num1, num2, sum = 0;
       char again;
       do {
           System.out.print("Enter number 1: ");
           num1 = input.nextInt();
           System.out.print("Enter number 2: ");
           num2 = input.nextInt();
           sum = num1 + num2;
           System.out.println("The sum is: " + sum);
           System.out.print("Do you want to do this again? (Y/N):");
           again = input.next().charAt(0);
       }while(again == 'Y' || again == 'y');
```

So far…

- We have seen the “while” and “do while” loop, in which we usually declare a variable “*i*” (stands for iterator) before the loop as the counter to control the loop.
- Inside of the loop, we have to modify the ***i*** to make sure that the loop can be existed successfully.

```
int i = 1;                 // declare a counter (before the loop)
while (i <= num) {         // check the counter (in the condition)
   System.out.print(i);
   i++;                    // increase the counter (in the loop)
}
```

- We must declare the counter, check the counter, and change the value of the counter in three different places (one before the loop, one in the (), one in the loop body), which is, sometimes, inconvenient.

The `for` Loop

for loop is one convenient statement indicates:

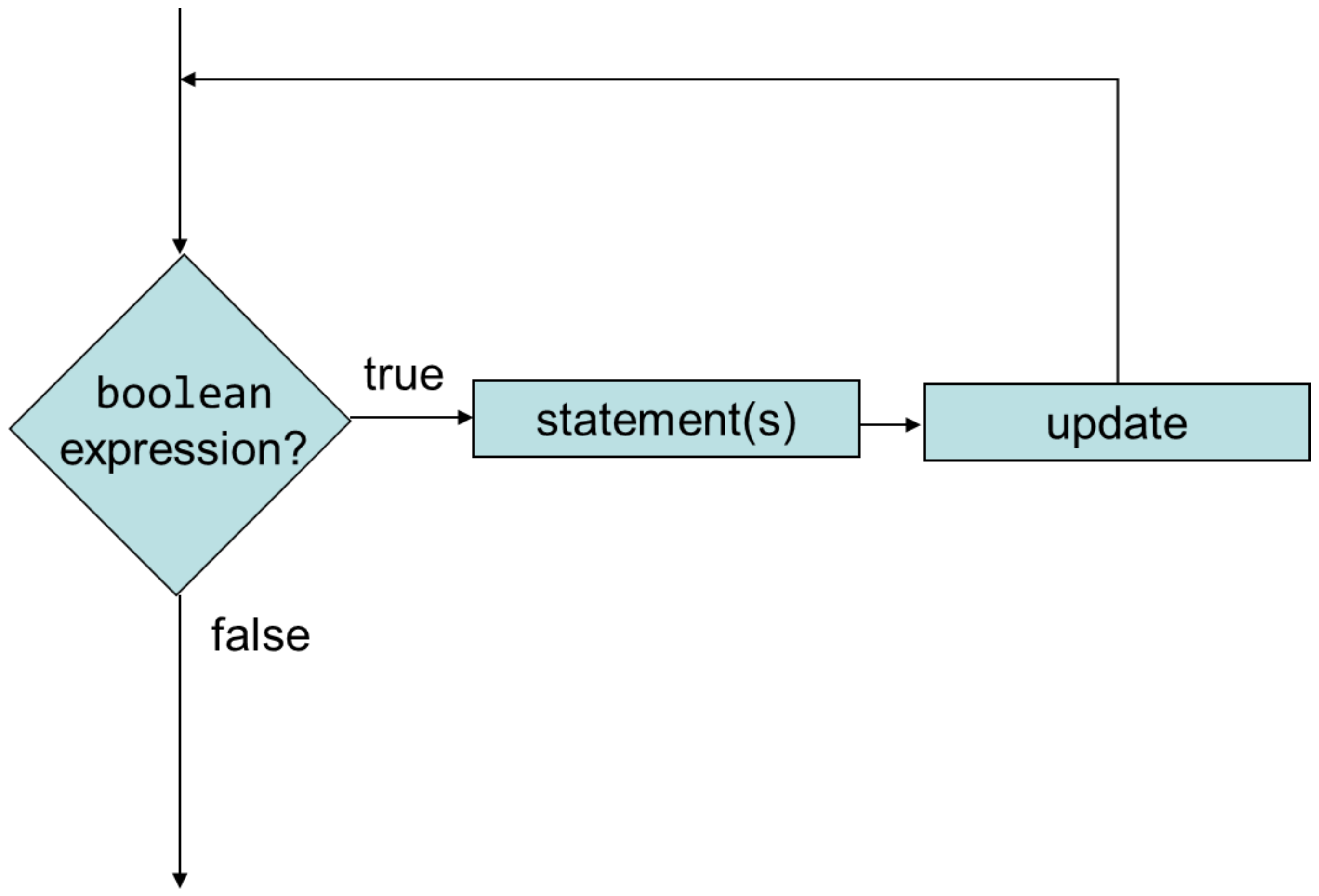
- The starting value for the loop control variable
- The test condition that controls loop entry
- The expression that alters the loop control variable

The **`for`** Loop

- The `for` loop 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:

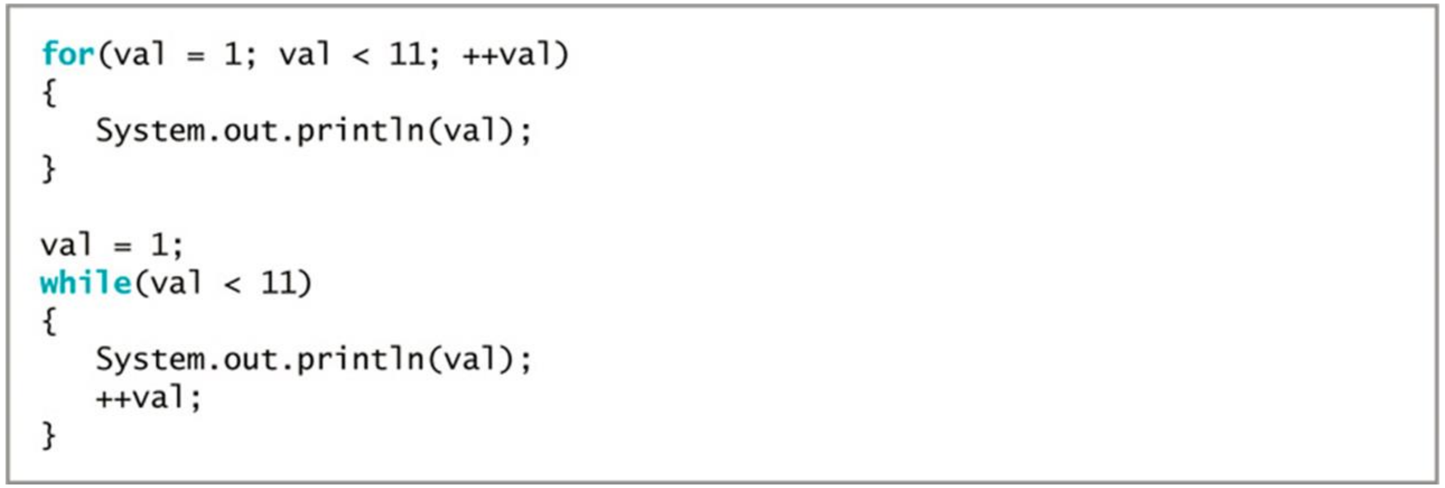
```
for (initialization; test; update)
{
    statement(s);
}
```

The `for Loop` Flowchart



The 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(val = 1; val < 11; ++val)
{
    System.out.println(val);
}

val = 1;
while(val < 11)
{
    System.out.println(val);
    ++val;
}
```

The `for` Loop Initialization

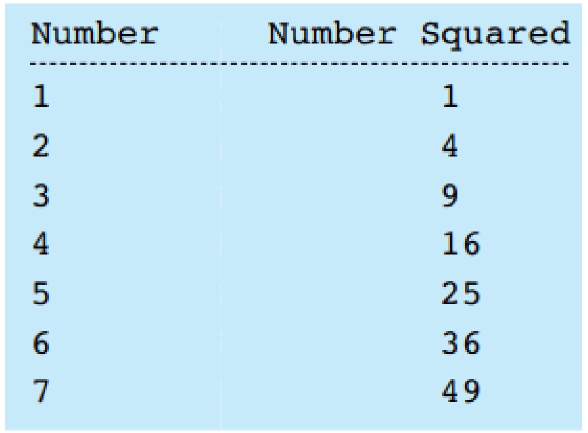
- The initialization section of a `for` loop is optional; however, it is usually provided.
- Typically, `for` loops initialize a counter variable that will be tested by the test section of the loop and updated by the update section.
- The initialization section can initialize multiple variables.
- Variables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

The Update Expression

- The update expression is usually used to increment or decrement the counter 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.

Demo

- Demonstrate a user controlled for loop by displaying the following table:



```
Number      Number  Squared
---------------------------
1                1
2                4
3                9
4                16
5                25
6                36
7                49
```

- Write a program to classify N Numbers and calculate number of 0s, number of even integers, number of odd integers

Modifying the Control Variable

- You should avoid updating 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.

The **`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.

The `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.

sentinel-controlled loop

- Sometimes 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.)

```
while (inValue != -99)
{
Statements..
}
```

Nested Loops

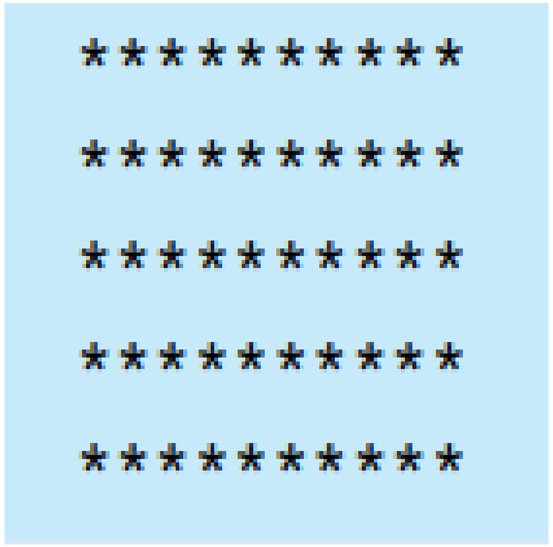
- Like `if` statements, loops can be nested.
- If a loop is nested, the inner loop will execute all of its iterations for each time the outer loop executes once.

```
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.

Demo

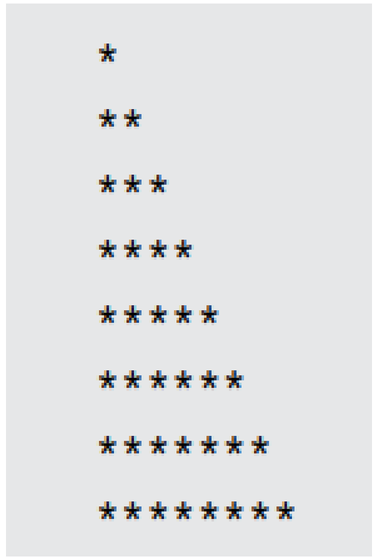
- One interesting way to learn about nested loops is to use them to display patterns on the screen. Let’s look at a simple example. Suppose we want to print asterisks on the screen in the following rectangular pattern:



```
**********
**********
**********
**********
**********
```

```
for (int i = 1; i <= 5; i++) {
      for (int j = 1; j <= 10; j++) {
        System.out.print("*");
      }
      System.out.println();
   }
```

Try it yourself



```
*
**
***
****
*****
******
*******
********
```

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.