Java Chapter 06 – Loops

**Loop Structure & Function:**

* W/in the looping structure, a bool expression is evaluated
* If the expression is true, the statement(s) in the loop body are executed
* Expression is evaluated again and again until the expression returns false

**WHILE Loop:**

* Used when you need to perform tasks an uncertain number of times
* The loop-controlling bool expression is the 1st statement in the loop
* Syntax : while(condition) { }
* Writing a definite(counter-controlled) while loop:
* Initialize a loop control variable
* In loop body, write statement that alters the L.C.V. by incrementing or decrementing
* Writing an indefinite(event-controlled) while loop:
* Not required to alter the L.C.V by ++ or –
* Value of the L.C.V. is often altered by user input rather than arithmetic

**INFINITE LOOPS:**

* Bad to write intentionally
* How to exit 🡪 CTRL, C or the Break Key
* Preventing an infinite loop:

**VALIDATING DATA:**

* Indefinite loops are commonly used when validating input data

\*\*Mistake Tip: don’t check for invalid data using IF statements; USE LOOPS!

**SHORTCUT ARITHMETIC OPERATORS:** perform operation & assigns the result in one step

* += 🡪 count += 1; increments count by 1
* --= 🡪 count --= 1; decrements count by 1
* \*= 🡪 count \*= 2; multiplies count by 2
* /= 🡪 count /= 2; divides count by 2
* %= 🡪 count %= 2; divides count by 2 & assigns the remainder
* Prefix and Postfix operators:
* ++count & count++ 🡪 increments count by 1
* -- --count & count -- -- 🡪 decrements count by 1

\*\*When prefix & postfix operators are used as part of a larger expression it makes a difference which one you use because they function differently in terms of what they return

\*\*Cannot use prefix and postfix operators w/ constants

* ++5 is illegal; instead 🡪 assign 5 to val & use ++val

**FOR Loop:**

* Used when you need to perform tasks a definite number of times
* More convenient way to create counter-controlled loop than a while loop
* Syntax: for (val= 1 ; val < 10 ; val++) { statement(s) }
* Initializing 🡪 Indicate the starting value for the L.C.V.
* Testing 🡪 The test condition that controls loop entry
* Incrementing 🡪 The expression that alters the L.C.V.
* Unconventional For Loops:

The 3 sections most commonly used to hold single expressions, however: other tasks can be done

* Initializing more than 1 variable using commas between
* Declare a variable w/in the for statement
* Performing more than 1 test using compound conditions
* Performing more than one task in Incrementing
* Using methods calls in any of the 3 sections
* Leaving 1 or more sections of the for statement empty – ‘;’ still required as placeholders
* Tips Do’s & Dont’s:
* Generally, should use the same L.C.V. in all 3 sections
* Avoid altering the L.C.V. in the body of the loop
* Java contains a built-in method to pause program execution – sleep()

**DO... WHILE Loop:**

* Used when you need a loop to execute at least once
* The loop-controlling bool expression is the last statement
* Syntax: do { statement(s) } while (condition)
* \*\*This loop is never required: display the needed info before, then proceed with a while loop

**NESTED LOOPS:**

* When loops are nested, each pair contains an Inner loop & an Outer loop
* Inner loop must be entirely contained w/in Outer loop; Loops can never overlap
* Always think of the Outer loop as the all-encompassing loop
* Sometimes it doesn’t matter which variable controls the outer & which variable controls the inner, but often it does make a difference. (Pay Attention To Logic)

**IMPROVING LOOP PERFORMANCE:** more efficient techniques in bold

* Avoid Unnecessary Operations: w/in the tested expression or the body
* while (x < a + b) 🡪 inefficient to recalculate every iteration
* **int sum = a + b;**

**while (x < sum)** 🡪 more efficient as long as the sum of a & b is fixed prior to the loop

* while (count < getNumberOfEmployees()) 🡪 calls method each time
* **numEmployees = getNumberOfEmployees();**

**while (count < numEmployees)** 🡪 if method returns the same value during program

* Consider the Oder of Evaluation: of short-circuit operators AND, OR
* Especially when the loop might execute many times
* Enter the more likely option first to eliminate testing the other(s)
* Comparing To **0:**
* **Saves time**; loops using the 0 comparison will never be slower than ones not using it
* Will vary for subsequent executions on the machine depending on events occurring elsewhere on the machine during the same time period

for (int x = 0; x <= REPEAT; ++x) 🡪 faster

for (int x = REPEAT; x <= 0; ++x) 🡪 slower

* Employing Loop Fusion:

for (int x = 0; x < TIMES; ++x) { method1(); }

for (int x = 0; x < TIMES; ++x) { method2(); }

* If the 2 methods don’t depend on each other

for (int x = 0; x < TIMES; x++) {

method1();

method2();

}

* Using Prefix Rather than Postfix Incrementing:
* Prefix Incrementing produces a faster loop
* **Prefix:** method receives reference, increases, & returns
* **Postfix:** method receives reference, **makes copy & stores it,** increases, & returns
* Final Note:
* Always try to find ways to enhance & make code more efficient

However, almost always err in favor of programs that are more readable & easier to maintain even if they execute more slowly

\*\*\*Might Be Useful\*\*\*

Using WHILE with a FOR in cases where you need both a Counter-Controlled Loop and a Boolean

* Use the FOR w/in the WHILE

Enhanced For Loop/Statement 🡪 goes w/ Arrays