```
class Lecture4 {

                    "Loops"

}

/* References
[1] Ch. 5 in YDL
*/
```

Loops

A loop can be used to make a program execute statements repeatedly without having to code the same statements.

- For example, a program prints "Hello, Java." for 100 times.

```
System.out.println("Hello, Java.");
System.out.println("Hello, Java.");
.
. // Copy and paste for 100 times
.
System.out.println("Hello, Java.");
```

```java
int cnt = 0;
while (cnt < 100) {
    System.out.println("Hello, Java.");
    cnt++;
}
```

- This is a simple example to show the power of loops.
- In practice, any routine which repeats couples of times[1] can be done by folding them into a loop.

[1] I'd like to call them "patterns."

Flow Controls (Recap)

The concept of looping, which is one of elements in algorithms, is fundamental to programming.

- Loops provide significant computation power.
- Loops bring an efficient way of programming.
- Loops could consume a lot of time.[2]

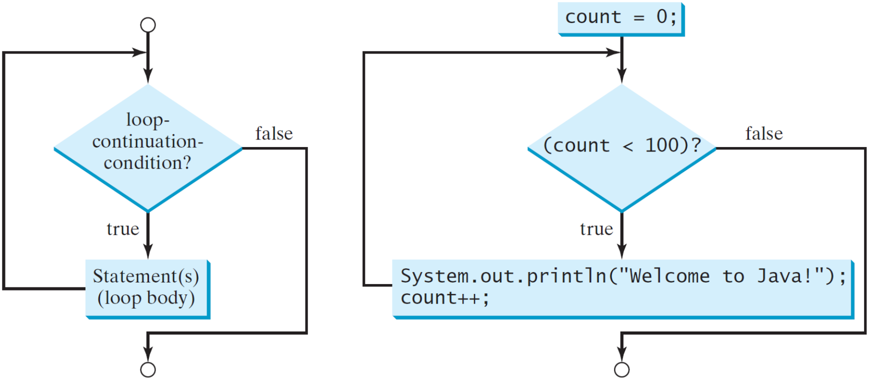
[2] We will visit the analysis of algorithms soon.

while Loops

A while loop executes statements repeatedly while the condition is true.

```
...
        while (condition) {
            // loop body
        }
...
```

- The condition is a Boolean expression which controls the execution of the body.
- It is evaluated each time to determine if the loop body is executed.
- If true, the loop body is executed.
- Otherwise, the entire loop terminates.


loop-
continuation-
condition?
false
true
Statement(s)
(loop body)
count = 0;
(count < 100)?
false
true
System.out.println("Welcome to Java!");
count++;

Example

> Write a program which sums up all integers from 1 to 100.

- In math, the question can be:

$$\text{sum} = 1 + 2 + \cdots + 100.$$

- But the form is not doable in a computer.
  - What is $\cdots$?!
- Think in a programmer's way.

- Normally, the computer executes the instructions sequentially.[3]
- So, one needs to decompose the math equation into several lines, like:

```
int sum = 0;
sum = sum + 1;
sum = sum + 2;
.
.
.
sum = sum + 100;
```

- Cons: Not efficient, not general (what if sum up to 10^{10}?)

[3] If we are talking about the parallel computing, then it is a different world.

- Using a while loop, the program looks like this:

```java
    ...
    int sum = 0;
    int i = 1;
    while (i <= 100) {
        sum = sum + i;
        ++i;
    }
    ...
```

- Make sure that the condition eventually becomes false so that the loop will terminate.
- It is really easy to make an infinite loop.

```java
...
    while(true);
...
```

- Besides, replacing 100 by *n* determined by the user makes this program more general.

```
...
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        int sum = 0;
        int i = 1;
        while (i <= n) {
            sum = sum + i;
            i = i + 1;
        }
...
```

- In practice, the number of loop steps is unknown until the input data is given.
- For example, the bisection algorithm[4] provides a numerical solution to root-finding problems.

[4] http://en.wikipedia.org/wiki/Bisection_method

Example

Write a program which sums two random integers and lets the user repeatedly enter a new answer until it is correct.

```
...
    public static void main (String[] args) {
        // random integer generation
        int number1 = (int)(Math.random() % 10);
        int number2 = (int)(Math.random() % 10);

        Scanner input = new Scanner(Sysmte.in);
        System.out.println(number1 + " + " + number2 + " = ?");
        int ans = input.nextInt();

        while (number1 + number2 != ans) {
            System.out.println("Wrong answer. Try again?");
            ans = input.nextInt();
        }
        System.out.println("Congrats! You are smart!");
    }
...
```

Exercise[5]

Write a program which runs for a predetermined time, say, one minute, and terminates itself.

- You may use **System.currentTimeMillis**() to produce a time stamp.

[5] Contribution by Ms. Hsu, Tzu-Hen (JB25318) on June 6, 2015.

```java
...
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        long t0 = System.currentTimeMillis();
        System.out.println("Duration? (ms) ");
        long interval = input.nextInt();
        input.close();
        while (System.currentTimeMillis() <= t0 + interval)
            System.out.println("Running...");
        System.out.println("Time's up.");
    }
...
```

Loop Design Strategy

Writing a correct loop is not an easy task for novice programmers.
Consider 3 steps when writing a loop:

- Find the pattern: identify the statements that need to be repeated.
- Wrap: put these statements in a loop.
- Set the continuation condition: translate the criteria from the real world problem into computational conditions.[6]

[6] Not unique.

Sentinel-Controlled Loop

Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special input value, known as a sentinel value, signifies the end of the loop.

Example

Write a program which sums real numbers from the input except for -1 to exit, and displays the sum.

```
1  ...
2          Scanner in = new Scanner(System.in);
3          double sum = 0;
4          System.out.println("Enter a positive integer (-1 to exit): ");
5          double x = in.nextDouble();
6          while (x != -1) {
7              sum += x;
8              System.out.println("Enter a positive integer (-1 to exit):
                   ");
9              x = in.nextDouble();
10         }
11         System.out.println("Sum = " + sum);
12         in.close();
13 ...
```

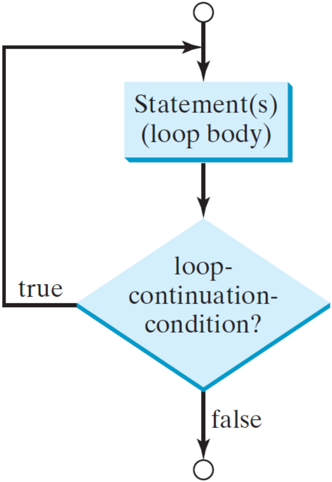
- Line 8 and 9 are the recurrence of Line 4 and 5?!

do-while Loops

A do-while loop is the same as a while loop except that it does execute the loop body first and then checks the loop continuation condition.

```
...
        do {
            // loop body
        } while (condition); // Do not miss the semicolon!
...
```

- Note that there is a semicolon at the end the do-while loop.
- do-while loops are also called posttest loop, in contrast to while loops, which are pretest loops.


Statement(s)
(loop body)
true
loop-
continuation-
condition?
false

Example

Write a program which allows the user to enter positive integers except for -1 to exit, and displays the maximum.

```
Please enter a real number (-1 to exit):
5
Max = 5.0
Please enter a real number (-1 to exit):
2
Max = 5.0
Please enter a real number (-1 to exit):
7
Max = 7.0
Please enter a real number (-1 to exit):
-1
```

```
...
        Scanner in = new Scanner(System.in);
        int max = 0, x;
        do{
            System.out.println("Please enter a positive integer (-1 to exit): ");
            x = in.nextInt();
            if(max < x) {
                max = x;
            }
            System.out.println("Max = " + max);
        } while(x != -1);
        in.close();
...
```

for Loops

A for loop generally uses a variable to control how many times the loop body is executed and when the loop terminates.

```
...
        for(init; condition; increment) {
            // loop body
        }
...
```

- *init*: initialize a variable
- *condition*: a criteria that a loop continues
- *increment*: how the variable changes after each iteration
- Note that the three terms are separated by a semicolon.

Example

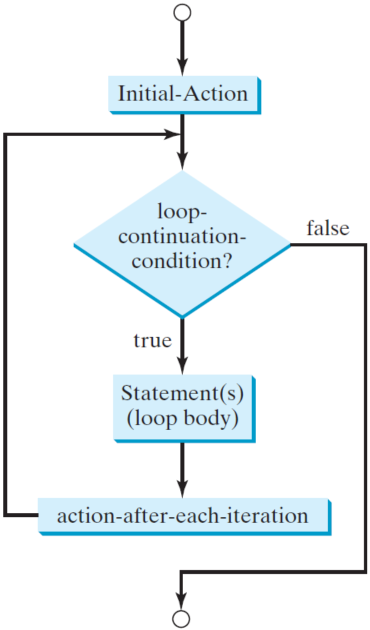
Sum from 1 to 100

Write a program which sums from 1 to 100.

```
...
        int sum = 0;
        for (int i = 1; i <= 100; ++i)
            sum = sum + i;
...
```

- Compared to the while version,

```
        ...
        int sum = 0;
        int i = 1;
        while (i <= 100) {
            sum = sum + i;
            ++i;
        }
        ...
```


Initial-Action
loop-continuation-condition?
false
true
Statement(s) (loop body)
action-after-each-iteration

Example: Selection Resided in Loop

Display all even numbers

Write a program which displays all even numbers smaller than 100.

- An even number is an integer of the form $x = 2k$, where k is an integer.
- You may use modular operator (%).

```
...
        for (int i = 1; i <= 100; i++) {
            if (i % 2 == 0)
                System.out.println(i);
        }
...
```

- You may consider this alternative:

```
...
        for (int i = 2; i <= 100; i += 2) {
                System.out.println(i);
        }
...
```

- How about odd numbers?

for Loops: Multiple Loop Variables

The loop variable *init* can be a list of zero or more comma-separated variable declaration statements or assignment expressions.

```
...
    for (int i = 0, j = 0; i + j < 10; i++, j += 2) {
        System.out.println("(i, j) = " + "(" + i + ", " + j + ")");
    }
...
```

- The last line on display is?

Jump Statements

The keywords, break and continue, are often used in loop structures to provide additional controls.

- break: The loop is terminated right after a break statement is executed.
- continue: The loop skips this iteration right after a continue statement is executed.
- Normally, jump statements are placed within selection structures in loops.

Example

isPrime problem

Write a program which determines if the input integer is a prime number.

- Recall that a prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
- Let x be any natural number.
- The most naive approach is to divide x by all natural numbers smaller than x.
- A better approach is to divide x by all natural numbers smaller than $\sqrt{x}$. (Why?)

```java
...
        Scanner input = new Scanner(System.in);
        System.out.println("Please enter an integer: ");
        int x = input.nextInt();
        in.close();
        for (int i = 2; i <= (int) Math.sqrt(x); i++) {
            if ((x % i) == 0) {
                System.out.println("Composite");
                break;
            }
            if (i == (int) Math.sqrt(x))
                System.out.println("Prime");
        }
...
```

Equivalence: while and for Loops

Compounding problem

Assume that the initial amount of saving is $10,000$ NTD. Write a program which determines the number of years n such that the compounding amount of saving exceeds $15,000$ NTD.

- Suppose the following variables:
  - *amount*: the money at time 0
  - *goal*: the money at time n
  - r: the annual interest rate during time 0 to n
- Compounding?
- Stopping criteria? Continuation criteria?

```java
...
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Interest rate (%) = ?");
        double r = in.nextDouble();
        System.out.println("Amount = ?");
        double amount = in.nextDouble();
        System.out.println("Goal = ?");
        double goal = in.nextDouble();
        int n = 0;
        while (amount <= goal) { // Continuation criteria
            amount *= (1 + r / 100);
            n = n + 1;
        }
        System.out.println("Years = " + n);
        System.out.println("Amount = " + amount);
        in.close();
    }
...
```

- Note that the listing is a generalized version.
- Try to extend your work as general as possible.
- It could be hard, but worth.

```java
...
        int n; // Should declare n here.
        for (n = 1; true; n++) {
            amount *= (1 + r);
            if (amount >= goal) // stopping criteria
                break;
        }
...
```

- A for loop can be an infinite loop by setting true or simply leaving empty in the *condition* in for.
- An infinite for loop with a break statement is equivalent to a normal while loop.

Equivalence: while and for Loops (Concluded)

You can use a for loop, a while loop, or a do-while loop, whichever is convenient.

- In general, a for loop may be used if the number of repetitions is known in advance.
- If not, then a while loop is preferred.
  - Recall the bisection algorithm for root finding.

Nested Loops

A loop can be nested inside another loop.

- Nested loops consist of an outer loop and one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered, and started anew.

Example

Multiplication table

Write a program which displays the multiplication table.

```
              Multiplication Table
        1    2    3    4    5    6    7    8    9
-----------------------------------------------------
1 |     1    2    3    4    5    6    7    8    9
2 |     2    4    6    8   10   12   14   16   18
3 |     3    6    9   12   15   18   21   24   27
4 |     4    8   12   16   20   24   28   32   36
5 |     5   10   15   20   25   30   35   40   45
6 |     6   12   18   24   30   36   42   48   54
7 |     7   14   21   28   35   42   49   56   63
8 |     8   16   24   32   40   48   56   64   72
9 |     9   18   27   36   45   54   63   72   81
```

```java
...
    public static void main(String[] args) {
        for (int i = 1; i <= 9; ++i){
            for (int j = 1; j <= 9; ++j)
                System.out.printf("%3d", i * j);
            System.out.println();
        }
    }
...
```

- You can try to make exactly the same table like this:

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Exercise: Coupled Loops

```
*          *****        *   *****
**         ****        **    ****
***        ***        ***     ***
****       **        ****      **
*****      *        *****       *
 (a)        (b)       (c)      (d)
```

```java
// Case (a)
public class printStars {
    public static void main(String[] args){
        for(int i = 1; i <= 5; i++) {
            for(int j = 1; j <= i; j++){
                System.out.printf("*");
                if (j == i)
                    System.out.printf("\n");
            }
        }
    }
}
```

Analysis of Algorithm In A Nutshell

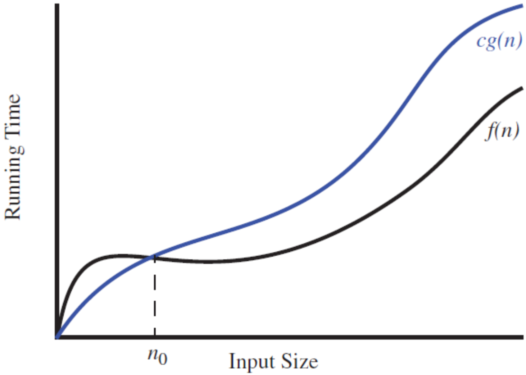
- First, the algorithms for the same problem are supposed to be correct.
- Then we compare these algorithms.
- The first question is, Which one is more efficient? (Why?)
- We focus on the growth rate of the running time or space requirement as a function of the input size n, denoted by $f(n)$.

O-notation

- In math, O-notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions.
- **Definition** (O-notation) $f(x) \in O(g(x))$ as $x \to \infty$ if and only if there is a positive constant c and a real number x_0 such that

$$|f(x)| \leq c|g(x)| \quad \forall x \geq x_0. \tag{1}$$

- So, $O(g(x))$ is a set featured by some $g(x)$.
- $f(x) \in O(g(x))$ means $f(x)$ is one element of $O(g(x))$.



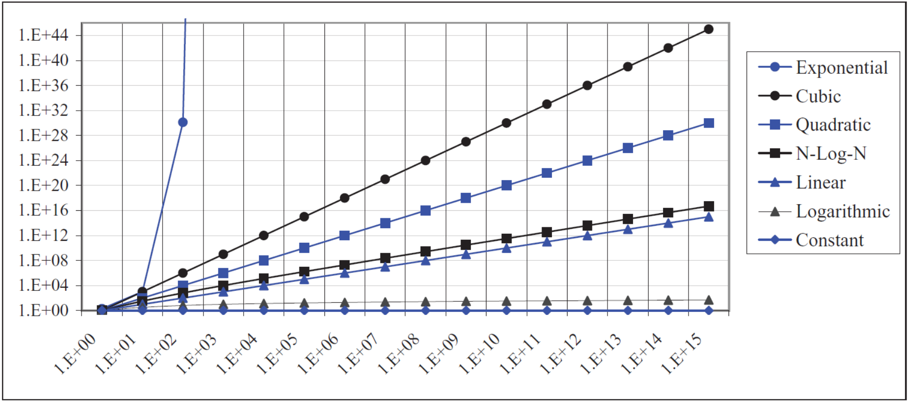
- For example, $8n^2 - 3n + 4 \in O(n^2)$ (tight bound).
- Note that $8n^2 - 3n + 4 \in O(n^3)$ (loose) but $8n^2 - 3n + 4 \notin O(n)$ (false).

- O-notation is used to classify algorithms by how they respond to changes in input size.[7]
  - Time complexity
  - Space complexity
- In short, O-notation describes the asymptotic[8] upper bound of the algorithm.
- That is, the worst case we can expect as $n \to \infty$.

[7] Actually, there are $\Theta, \theta, o, \Omega$, and ω which classify algorithms.

[8] The asymptotic sense is that the input size n grows toward infinity.

Growth Rates for Fundamental Functions[9]



constant	*logarithm*	*linear*	*n-log-n*	*quadratic*	*cubic*	*exponential*
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

[9]See Table 4.1 and Figure 4.2 in Goodrich and etc, p. 161.

Problem Set

Exercise 4.1 (Greatest common divisor)

Write a program which receives two positive integers and returns the greatest common divisor which is the largest positive integer that divides the numbers without a remainder.

Exercise 4.2 (Find all prime numbers smaller than 1000)

Write a program which displays all prime numbers smaller than 1000.

Exercise 4.3 (π estimated by Monte Carlo)

Write a program which estimates π by Monte Carlo Simulation.

Exercise 4.4 (Find the two highest scores)

Write a program that prompts the user to enter the number of students and each student's name and score, and finally displays the student with the highest score and the student with the second-highest score.

Exercise 4.5 (Find numbers divisible by 5 and 6)

Write a program that displays all the numbers from 100 to 1,000, ten per line, that are divisible by 5 and 6. Numbers are separated by exactly one space.

Exercise 4.6 (Continued from 4.6)

Write a program that displays all the numbers from 100 to 200, ten per line, that are divisible by 5 or 6, but not both. Numbers are separated by exactly one space.

Exercise 4.7

Write a program that finds the smallest n such that $n^2 > 12000$ using a while loop.

Exercise 4.8

Write a program which finds the largest n such that $n^3 < 12000$ using a while loop.

Exercise 4.9 (Display pyramid)

Write a program that prompts the user to enter an integer from 1 to 15 and displays a pyramid, as shown in the following sample run:

```
Enter the number of lines: 7 ↵Enter
                        1
                     2  1  2
                  3  2  1  2  3
               4  3  2  1  2  3  4
            5  4  3  2  1  2  3  4  5
         6  5  4  3  2  1  2  3  4  5  6
      7  6  5  4  3  2  1  2  3  4  5  6  7
```

Exercise 4.10 (Display numbers in a pyramid pattern)

Write a nested for loop that prints the following output:

```
                              1
                          1   2   1
                      1   2   4   2   1
                  1   2   4   8   4   2   1
              1   2   4   8  16   8   4   2   1
          1   2   4   8  16  32  16   8   4   2   1
      1   2   4   8  16  32  64  32  16   8   4   2   1
  1   2   4   8  16  32  64 128  64  32  16   8   4   2   1
```

Exercise 4.11 (Display four patterns using loops)

Use nested loops that display the following patterns in four separate programs:

```
Pattern A      Pattern B      Pattern C      Pattern D
1              1 2 3 4 5 6              1    1 2 3 4 5 6
1 2            1 2 3 4 5              2 1      1 2 3 4 5
1 2 3          1 2 3 4              3 2 1        1 2 3 4
1 2 3 4        1 2 3              4 3 2 1          1 2 3
1 2 3 4 5      1 2              5 4 3 2 1            1 2
1 2 3 4 5 6    1              6 5 4 3 2 1              1
```

Exercise 4.12 (Display leap years)

Write a program that displays all the leap years, ten per line, in the twenty-first century (from 2001 to 2100), separated by exactly one space.

Exercise 4.13 (Compute π)

You can approximate by using the following series:

$$\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{(-1)^{i+1}}{2i-1})$$

Write a program that displays the value for $i = 10000, 20000, \ldots,$ and 100000.

Exercise 4.14 (Perfect number)

A positive integer is called a perfect number if it is equal to the sum of all of its positive divisors, excluding itself. For example, 6 is the first perfect number because $6 = 3 + 2 + 1$. The next is $28 = 14 + 7 + 4 + 2 + 1$. There are four perfect numbers less than 10,000. Write a program to find all these four numbers.

Exercise 4.15 (Game: scissor, rock, paper)

Exercise 3.17 gives a program that plays the scissor-rock-paper game. Revise the program to let the user continuously play until either the user or the computer wins more than two times.

Exercise 4.16 Histogram for math grades

Write a program which allows the user to enter the math grades in order (-1 to exit), and makes a histogram.

```
Enter: 50
Enter: 60
Enter: 70
Enter: 80
Enter: 90
Enter: 100
Enter: 85
Enter: -1
```

```
Total: 7
100 ~  90: **
 89 ~  80: **
 79 ~  70: *
 69 ~  60: *
 59 ~   0: *
```