Lecture Notes
Chapter #4
Loops

1. Switch Statement

```
int n;
input a value into n, e.g., 3
switch (n)
{
    case 0: { System.out.println(n); break; }
    case 1: { System.out.println(n*n); break; }
    case 2: { System.out.println(n * n);}
    case 3: { System.out.println(" + n / n "); break; }
    case 4: { System.out.println(n * n + n / n); break; }
    default: System.out.println( "Value is out of bounds ");
}
```

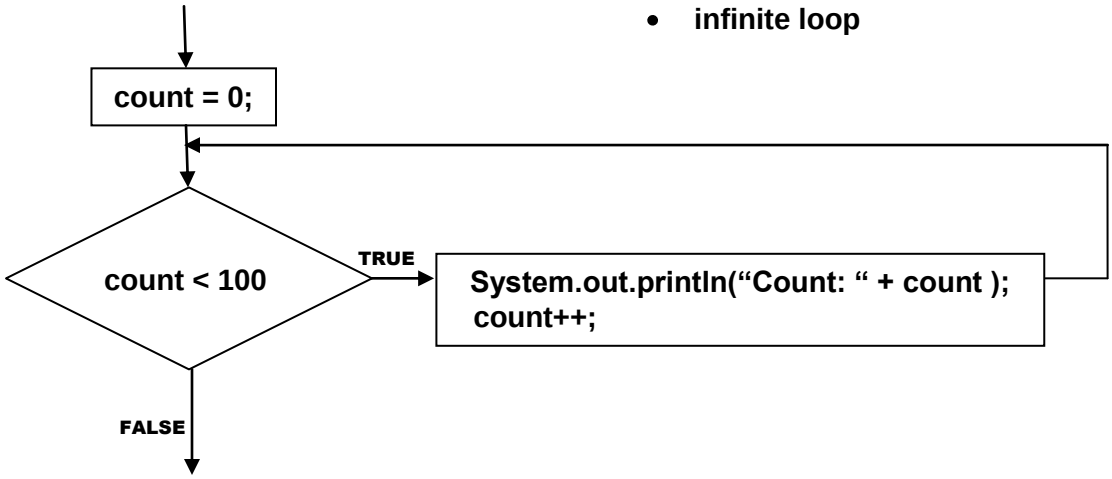
2. While Loop

a. Syntax

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

```
int count = 0;
while (count < 100)
{
   System.out.println("Count: " + count );
}
```

- loop iteration
- loop body
- infinite loop



b. Liang Listing 4.2 pages 106-107

```
int number = (int)(Math.random( ) * 101);

Scanner input = new Scanner(System.in);

int guess = -1;
while (guess != number)     // exit the loop if guess == number
{
  System.out.print( "\n Enter guess between 0 and 100 ");
  guess = input.nextInt( );
  if (guess == number)
        System.out.println( guess + " is correct!");
  else if (guess > number)
        System.out.println( guess + " is too high!");
  else
        System.out.println( guess + " is too low!");
}
```

$0.0 \leq r < 101.0 \rightarrow 0.0 \leq r \leq 100.9999\ldots.$

c.

A variable can only be created once! ← Partially TRUE !!

<u>In any block</u>, a variable can only be created once! E.g., the variable n can only be created once in the following block

```
{
  int n:
}
```

but another variable n can be created in another distinct block such as

```
{
  int n:
}
.....
{
  int n:
}
```

or in a block which encompasses another block, such as

```
{
  int n:
  ...
 {
  int n:
  ...
 }
...
}
```

Further Discussion Chapter 5 Variable Scoping Rules

d. Liang Listing 4.3 pages 107-108

```
final int NUMBER_OF_QUESTIONS = 5;            // line 5
int correctCount = 0;                         // line 6
int count = 0;                                // line 7
long startTime = System.currentTimeMillis( );    // line 8
String output = " ";
Scanner input = new Scanner(System.in);

while (count < NUMBER_OF_QUESTIONS)           // line 12
{
    int number1 = (int)(math.random( ) * 10);
    int number2 = (int)(math.random( ) * 10);

    if (number1 < number2)
    {
      int temp = number1;
      number1 = number2;
      number2 = temp;
    }

   System.out.print(number1 + " + " + number2 + " = " );
   int answer = input.nextInt( );

  if (number1 – number2 == answer)
  {
    System.out.println("Correct!");
    correctCount++;
  }
  else
  {
    System.out.println("Wrong! \nCorrect Answer: " + number1 – number2);
    count ++;                                                    // line 39
    output += "\n" + number1 + " – " + number2 + " = " + answer +
              ((number1 – number2 == answer) ?   "  Correct!" : "  Wrong!" );
  }
}
long endTime = System.currentTimeMillis( );     // line 45
long testTime = endTime – startTime;            // line 46

long seconds = testTime/1000;
long minutes = seconds/60;
seconds =  seconds % 60;  // seconds %= 60;
long hours = minutes/60;
minutes = minutes % 60;    // minutes %= 60;

// output correctCount
// output time required to complete the math quiz
```

e. Sentinel Values Liang Listing 4.4 page 109

```
int data = input.nextInt( );
int sum = 0:

while(data != 0)
{
  sum += data;
  data = input.nextInt( );
}

System.out.println("Total: " + sum);
```

f. Loop Variables – Caution!

If at all possible, don't use floating-point values for equality checking in loop statements. For example, given

```
double p = 2.0;
```

the test (p/3)*3 == p may fail because p/3 may return 0.6666667 which when multiplied by 3 returns 2.0000001. Thus the test (2.0000001 == 2.0) will fail.

if it is necessary to use floating-point values to control loop statements, use a bounded inequality such as the following:

```
double p, q;
while ( Math.abs(p – q) < 0.00000000001 )
{
}
```

Math.abs(p – q) is a function that returns the absolute value of p – q;

by restricting the returned value to be less than some small value, e.g., 0.00000000001, the while loop can be made to perform within some predefined limits.

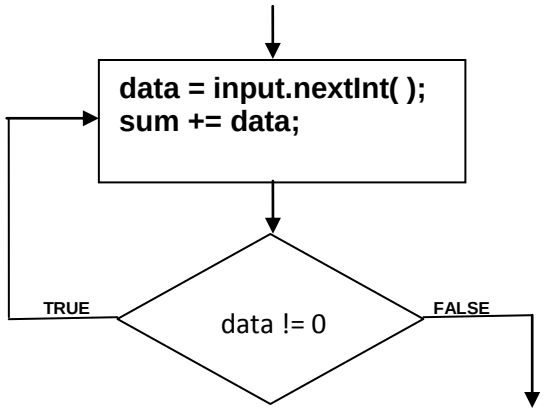
3. Do-While Loop Liang Listing 4.5 page 110-111

a. Syntax

```
do
{
 …
} while ( loop control
statement);
```

```
do
{
   data = input.nextInt( );
   sum += data;
} while ( data != 0 );
```

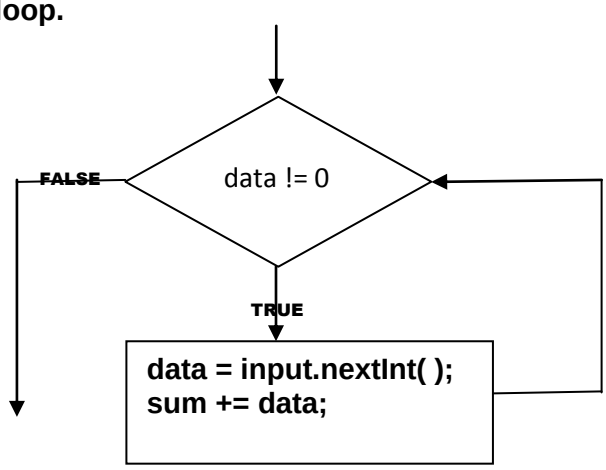
The Do-While Loop always <u>executes the body of the loop</u> before it reaches the control statement.




```
do
{
   data = input.nextInt( );
   sum += data;
} while ( data != 0 );
```

semicolon required

The While Loop always <u>executes the control statement</u> before executes the body of the loop.




```
data = input.nextInt( );
while(data != 0)
{
  sum += data;
  data = input.nextInt( );
}
```

semicolon prohibited

4. **For Loop** **Liang page 111-113**

a. **Syntax**

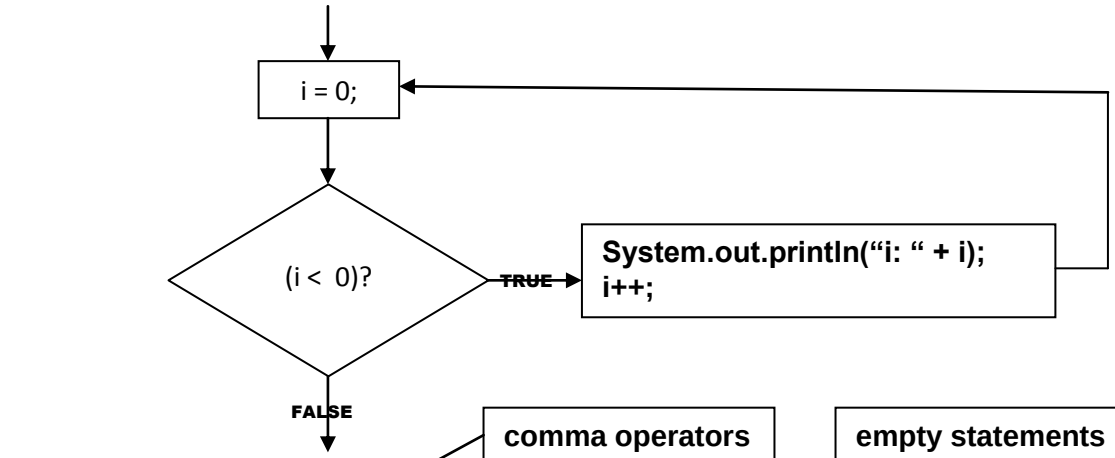
```
for( initial-stmts; loop-termination-tests; increment/decrement-stmts )
{
    …
}
```

```
for( int n = 1; n < 100; n++)
{
System.out.println(n);
}
```

```
for( … ; … ; … )
{
  …

}
```

b. **Actions (Semantics)**

i. Create loop control variable; this action may occur before entering the loop, but is usually done as part of executing the initial-stmts.
ii. Execute initial-stmts
iii. Execute loop-termination-tests on the control variable (which results in a boolean value);
    if the test is successful, continue to next step else terminate the loop
iv. Execute body of the loop
v. Execute increment/decrement-stmts on the loop control variable
vi. Go back to step iii above



comma operators

empty statements -- TRUE
infinite loops

```
for( int i = 1, j = 0, m = 27; i <= j + m; i++, j++ )
{
   // input m
   m %= (i + j);
  // output i, j, m
}
```

referencing the control loop variable outside of the loop

```
for(int n =0; ...; ...){  }

int n;
for( n =0; ...; ...){  }
```

5. **Comparative Loops**

   a. **Pre-test loops**
      - **while loop**
      - **for loop**

   b. **Post-test loops**
      - **do while loops**

   c. **Indefinite loops**
      - **while loop**
      - **do while loops**

   d. **Definite loops**
      - **for loop**

```
for(  ;  ;  )
{
   // actions
}
```

↔

```
for(  ; TRUE ;  )
{
   // actions
}
```

↔

```
while( TRUE )
{
   // actions
}
```

```
for( int i = 0; i < 10; i++); ← empty stmt
{
   System.out.println("i: " + i); } unrelated block
}
```

```
for( int i = 0; i < 10; i++){ }
                              ↑
                        empty body
```

```
int i = 0;
while( i < 0 ); ← empty stmt
{
   System.out.println("i: " + i++); } unrelated block
}
```

```
int i = 0;
while( i < 0 ){ } ← empty body
{
   ...    } unrelated block
}
```

6. Nested Loops

Listing 4.6 Multiplication Table page 129

```
System.out.println(“                Multiplication Table  “);
System.out.print(“        “);
for(int j = 1; j <=  9; j++) system.out.println(“    “ + j);
System.out.println(“\n_____________________________________”);

for( int i = 1; i <= 9; i++ )
{
    System.out.print(“    “ + “ | ”);
    for( int j = 1; j <= 9; j++ )
    {
        System.out.printf(“%4d”, i * j);
    }
    System.out.println(output);
}
```

Outer Loop

Inner Loop
Use %4d to reserve four spaces for each number, which allows any number up to four digits to be properly aligned without having to make adjustments for each number.

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

7. Numeric Errors in Loops (Numeric Analysis)

floating point numbers ➔ numeric errors

```
float sum = 0.0;

for(  float i = 0.01f;  i <= 1.0f;   i +=  0.01f )
   sum += 1;
System.out.println("Sum: " + sum);          ➔ Sum: 50.499985
                                               Should be 50.50
```

```
double sum = 0.0;

for(  double i = 0.01f;  i <= 1.0f;   i +=  0.01f )
   sum += 1;
System.out.println("Sum: " + sum);          ➔ Sum: 49.50000000000003
                                               Should be 50.50
```

```
double currentValue = 0.01;

for(  int count i = 0;  count < 100;   count++ )
{
   sum +=  currentValue;
   currentValue += 0.01;
}
System.out.println("Sum: " + sum);          ➔ Sum: 50.50000000000003
                                               Should be 50.50
```

```
double currentValue = 1.0;

for(  int count i = 0;  count < 100;   count++ )
{
   sum +=  currentValue;
   currentValue -= 0.01;       // currentValue = currentValue - 0.01;
}
System.out.println("Sum: " + sum);          ➔ Sum: 50.49999999999995
                                               Should be 50.50
```

Adding small amounts to large amounts is more accurate than adding large amounts to small amounts
Result(Small + Large) "more accurate than" Result(Large + Small)
Computer addition is NOT commutative
a + b b + a

8. Greatest Common Divisor

GCD(n1, n2)
n1 = 12;
n2 = 27;
gcd == 3

Listing 4.8 GCD

```
int gcd = 1;
int k = 2;

int n1 = input.nextInt( );
int n2 = input.nextInt( );

while ( k <= n1 && k <= n2 )
{
   if ( n1 % k == 0 && n2 % k == 0 )
      gcd = k;
   k++;
}
```

http://www.cut-the-knot.org/blue/Euclid.shtml

Think before you type!
Design potential algorithms
Evaluate the multiple solutions suggested in the algorithms
Erroneous solutions – see page 118

9. Sales Amount Listing 4.9

Off-by-One Error

10. Pyramid of Numbers Listing 4.10

```
int numberOfLines = input.nextInt( );

if ( numberOfLines < 1 || numberOfLines > 15)
{
  System.exit(0);
}

for ( int row = 1; row <= numberOfLines; row++)
{
   for ( int column = 1; column <= numberOfLines – row; column++)
   System.out.print("   ");

  for (  int num = row; num >= 1; num-- )
  System.out.print( (num >= 10) ? " " + num : "  " + num );     // one space, two spaces

  for (  int num = 2; num <= row; num++ )
  System.out.print( (num >= 10) ? " " + num : "  " + num );     // one space, two spaces

  System.out.println( );
}
```

11. Break vs. Continue

- **break exits the innermost enclosing block;**
- **execution continues in the succeeding block**
  - **break exits a single loop**
  - **break exits the innermost loop, but continues execution in the next outer loop**
- **continue ends/exits the current iteration of the block**
- **execution continues in the next iteration of the same block**
  - **continue exits the current iteration of the loop**
  - **continue exits the current iteration of the loop, but continues execution in the next iteration of the same loop**

- **continue remains in the current block**
- **continue is only used in loops, i.e., continue only makes sense in loops**
- **break exits the current block**
- **breaks are used in loops and in other blocks, e.g., switch statements**

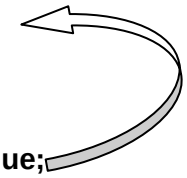
Listing 4.11

```
int sum = 0;
int number = 0;

while ( number < 20 )
{
  number++;
  sum += number;

  if( sum >= 100) break;              // if( sum == 100) break; ➔ infinite loop
}
```

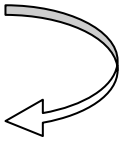
Listing 4.12

```
int sum = 0;
int number = 0;

while ( number < 20 )
{
  number++;
  if ( number == 10 || number == 11 ) continue;
  sum += number;
}
```

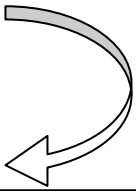
```
while ( loop-continuation-statement )
 {
      ….
      ….
    if ( continue-evaluation-statement) continue;
      ….
      ….
}
```



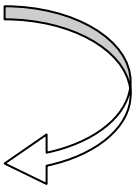
```
 do
 {
     ….
      ….
    if ( continue-evaluation-statement) continue;
      ….
      ….
} while ( loop-continuation-statement );
```



```
while ( loop-continuation-statement )
 {
      ….
      ….
    if ( continue-evaluation-statement) break;
      ….
      ….
}
```



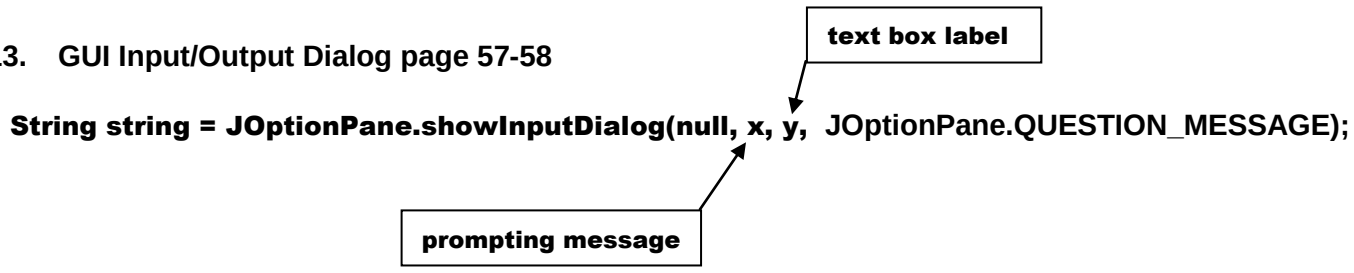
```
 do
 {
     ….
      ….
    if ( continue-evaluation-statement) continue;
      ….
      ….
} while ( loop-continuation-statement );
```



12. Prime Numbers Listing 4.14

```
for all divisors :
if (2 <= divisor && divisor <= number/2)
     if (number % divisor == 0))
     then number is not Prime;

final int NUMBER_OF_PRIMES = 50;
final int NUMBER_OF_PRIMES_PER_LINE = 10;
int count = 0;
int number = 2;

while( count < NUMBER_OF_PRIMES )
{
   boolean isPrime = true;
   for ( int divisor = 2; divisor <= number/2; divisor++ )
   {
     if ( number % divisor == 0 )
     {
 isPrime = false;
 break;
     }
   }
   if (isPrime)
   {
          count++;
 if ( count % NUMBER_OF_PRIMES_PER_LINE == 0 )
    System.out.println (number);
 else
    System.out.println (number + " ");
   }
   number++;
}
```

13. GUI Input/Output Dialog page 57-58




```
String annualInterestRateString =
        JOptionPane.showInputDialog( "Enter Yearly Interest Rate ");       // text label == "Input"

String output = "Monthly Payment: " + monthlyPayment + "\tTotal Payment: " + totalPayment;
JOptionPane.showMessageDialog(null, output);
```

14. GUI Confirmation Dialog page 92-93

```
String set1 =
“  1    3   5    7\n” +
“  9  11 13  15\n” +
“17  19  21  23\n” +
“25  27  29  31”;

String question = “Is your birthdate in this set of numbers?\n”
int answer == JOptionPane.showConfirmDialog(null, question + set1);
if (answer == JOptionPane.YES_OPTION) date +=1;
```

15. Loop Control with Confirmation Dialog Listing 4.15 pages 127-128

```
int sum = 0;
int option = 0;

while( option == JOptionPane.YES_OPTION)
{
  String dataString = JOptionPane.showInputdialog(“Enter Value: “);
  int data = Integer.parseInt(dataString);
  sum += data;
  option = JOptionPane.showConfirmDialog(null, “Continue?”);
}
JOptionPane.showMessageDialog(null, “Sum: “ + sum);
```

16. Command Line Input/Output

>java Test

Test written using System.out and Scanner Input/Output Statements
Uses Input from the Keyboard
Outputs Results to the Monitor

> java Test < file1 >file2
or
> java Test >file2 < file1

Test written using regular System.out and Scanner Input/Output Statements but all <u>input prompting statements</u> have been removed
Uses Input from the ASCII Text File labeled file1

<u>file1.txt</u> is created with text editor with date formatted to meet the requirements of the Test program, i.e., a single space between all numeric data items and quotation marks around all string data items

All Output is written to file2