1. Switch Statement

```java
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

```
int count = 0;
while (count < 100)
{
    System.out.println(“Count: “ + 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 ≤ r < 101.0  →  0.0 ≤ r ≤ 100.9999....

A variable can only be created once! ← Partially TRUE !!
In any block, 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
```java
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!
Correct 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

```java
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

```java
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:

```java
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);
```

The Do-While Loop always **executes the body of the loop** before it reaches the control statement.

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

```
true

semicolons
required
```

The While Loop always **executes the control statement** before executes the body of the loop.

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

```
false

semicolons
prohibited
```

TRUE

FALSE
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**
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

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

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

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

```java
for( int i = 0; i < 10; i++) { }
```

```java
while( i < 0 ){
  System.out.println("i: " + i);
} unrelated block
```

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

```java
for( int i = 0; i < 10; i++){
}
```

```java
empty stmt
```

```java
int i = 0;
while( i < 0 ){
  …
} unrelated block
```

```java
empty body
```
6. Nested Loops

Listing 4.6  Multiplication Table  page 129
System.out.println("Multiplication Table ");
System.out.println(" ");
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);
}

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

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.
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: 50.49999999999495
    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

\[ a + b \neq b + a \]
8. Greatest Common Divisor

\[
\text{GCD}(n_1, n_2) \\
\quad n_1 = 12; \\
\quad n_2 = 27; \\
\quad \text{gcd} == 3
\]

Listing 4.8 GCD
\[
\text{int gcd} = 1; \\
\text{int } k = 2;
\]
\[
\text{int } n_1 = \text{input.nextInt( );} \\
\text{int } n_2 = \text{input.nextInt( );}
\]
\[
\text{while } ( k <= n_1 && k <= n_2 ) \\
\quad \{
\quad \quad \text{if } ( n_1 \% k == 0 && n_2 \% k == 0 ) \\
\quad \quad \quad \text{gcd} = k; \\
\quad \quad \quad k++; \\
\quad \}
\]

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

\[
\text{int numberOfLines} = \text{input.nextInt( );}
\]
\[
\text{if } ( \text{numberOfLines} < 1 || \text{numberOfLines} > 15) \\
\quad \{
\quad \quad \text{System.exit(0);}
\quad \}
\]
\[
\text{for } ( \text{int row} = 1; \text{row} <= \text{numberOfLines}; \text{row}++) \\
\quad \{
\quad \quad \text{for } ( \text{int column} = 1; \text{column} <= \text{numberOfLines} - \text{row}; \text{column}++)
\quad \quad \quad \text{System.out.print(" ");}
\quad \quad \quad \text{for } ( \text{int num} = \text{row}; \text{num} >= 1; \text{num}-- )
\quad \quad \quad \quad \text{System.out.print( (num >= 10) ? " " + num : " " + num );} // one space, two spaces
\quad \quad \text{for } ( \text{int num} = 2; \text{num} <= \text{row}; \text{num}++ )
\quad \quad \quad \text{System.out.print( (num >= 10) ? " " + num : " " + num );} // one space, two spaces
\quad \quad \text{System.out.println( );}
\quad \}
\]
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**

```c
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**

```c
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 string = JOptionPane.showInputDialog(null, x, y, JOptionPane.QUESTION_MESSAGE);

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

String output = "Monthly Payment: " + monthlyPayment + "\nTotal Payment: " + totalPayment;
JOptionPane.showMessageDialog(null, output);
14. GUI Confirmation Dialog  page 92-93

String set1 =
"  1  3  5  7
"  9 11 13 15
"17 19 21 23
"25 27 29 31"

String question = “Is your birthdate in this set of numbers?
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: “);
in t 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  
G or
> java Test > file2 < file1

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

file1.txt 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