Java Basic Grammar 2

Java Programming

Byeongjoon Noh

powernoh@sch.ac.kr

SCH SOON CHUN HYANG UNIVERSITY

Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

Contents

1. Input and output

Input and output in java

- Input and output (I/O) mechanism
  - designed to handle a wide range of data processing needs
  - from reading user inputs from the keyboard to writing complex data structure to files

- Standard output: the console output in the monitor

```
System.out.println("Hello, World!");
```

```
Hello, World!
```

  - other output: writing to files the results

Keyboard inputs by ‘Scanner’ Class

- Standard input: keyboard
  - ‘Scanner’ class in the ‘java.util’ package in a versatile tool for reading input of various types
    - including strings, numbers, and other data, from different sources like keyboard, files, etc.

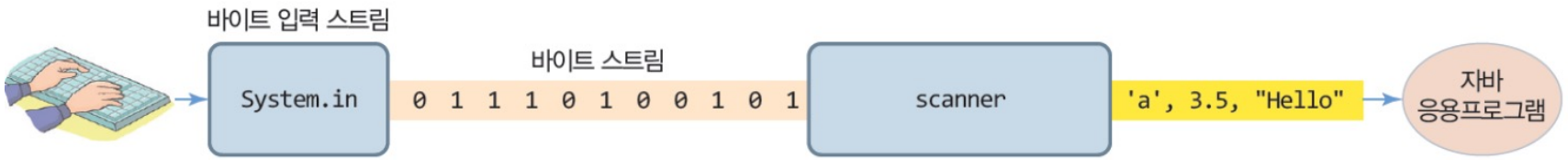
```
package Chap03Example;

import java.util.Scanner;

public class ScannerExample01 {
  public static void main(String[] args) { ... }
}
```

Usage of ‘Scanner’ class

- Create a new instance of ‘Scanner’ class

```
import java.util.Scanner;

public class ScannerExample01 {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
        ...
  }
}
```

  - simply… to declare to the computer that we are going to get keyboard input
    - we can use ‘`scanner`’ as variable from now on
  - note
    - `System.in`: a standard input (keyboard input)



Usage of 'Scanner' class

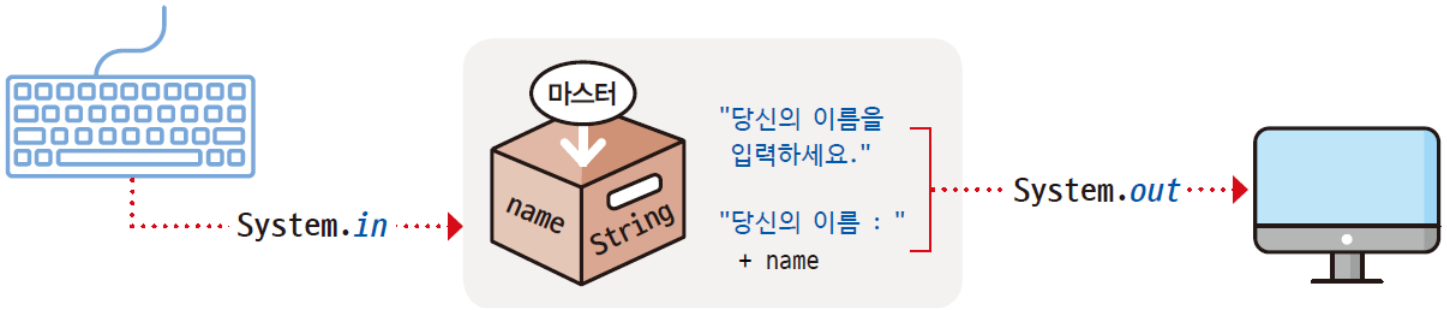
- Read one line by keyboard

```
import java.util.Scanner;

public class ScannerExample01 {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter your name: ");
    String name = scanner.nextLine(); // Reads a line of text

        ...
  }
}
```

Usage of ‘Scanner’ class

- Close the scanner (no longer use this ‘scanner’ variable):

```
import java.util.Scanner;

public class ScannerExample01 {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter your name: ");
    String name = scanner.nextLine(); // Reads a line of text
    System.out.print("Your name is " + name);

    scanner.close(); // Closes the scanner
  }
}
```



Usage of 'Scanner' class

- Read one line by keyboard; but only for integer value

```
import java.util.Scanner;

public class ScannerExample02 {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter your name: ");
    String name = scanner.nextLine(); // Reads a line of text

    System.out.print("Enter your age: ");
    int age = scanner.nextInt(); // Reads an integer

        ...
  }
}
```

Usage of 'Scanner' class

- Print the values from the keyboard input

```
import java.util.Scanner;

public class ScannerExample02 {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter your name: ");
    String name = scanner.nextLine(); // Reads a line of text

    System.out.print("Enter your age: ");
    int age = scanner.nextInt(); // Reads an integer

    System.out.println("Hello, " + name + ". You are " + age + " years old.");

        ...
  }
}
```

Usage of ‘Scanner’ class

- Close the scanner (no longer use this ‘scanner’ variable):

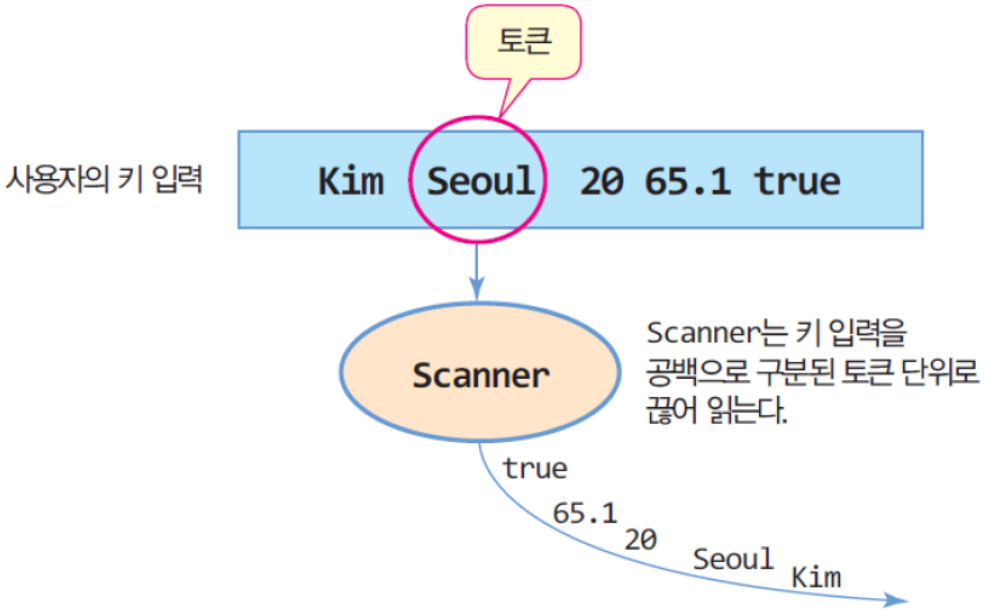
```
import java.util.Scanner;

public class ScannerExample02 {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter your name: ");
    String name = scanner.nextLine(); // Reads a line of text

    System.out.print("Enter your age: ");
    int age = scanner.nextInt(); // Reads an integer

    System.out.println("Hello, " + name + ". You are " + age + " years old.");

    scanner.close(); // Closes the scanner
  }
}
```

Methods of ‘Scanner’ class in Java

Method	Description
`next()`	Finds and returns the next complete token from this scanner.
`nextLine()`	Advances this scanner past the current line and returns the input that was skipped.
`nextDouble()`	Scans the next token of the input as a `double`.
`nextFloat()`	Scans the next token of the input as a `float`.
`nextBoolean()`	Scans the next token of the input as a `boolean`.
`hasNext()`	Returns `true` if this scanner has another token in its input.

Input tokens

- Scanner input example
  - scanner can get the values by ‘token’
    - token: based on a delimiter, which by default is whitespace (space, tabs, newlines, etc.)




```
Scanner scanner = new Scanner(System.in);

String name = scanner.next(); // "Kim"
String city = scanner.next(); // "Seoul"
int age = scanner.nextInt(); // 20
double weight = scanner.nextDouble(); // 65.1
boolean single = scanner.nextBoolean(); // true
```

Examples and practices for scanner

- 사용자로부터 이름, 나이, 거주도시, MBTI를 입력받아서 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/ScannerPractice01.java
  - inputs and outputs

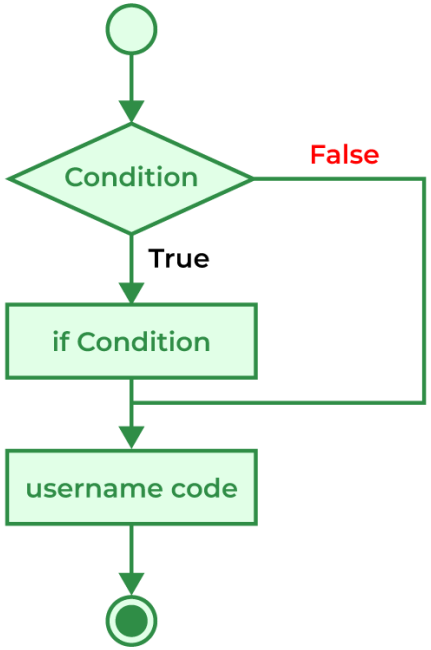
```
Please enter name, age, city, MBTI information with space
ByeongjoonNoh 31 Sejong ENTJ
Name:ByeongjoonNoh
Age:31
City:Sejong
MBTI:ENTJ
```

2. Conditional statement

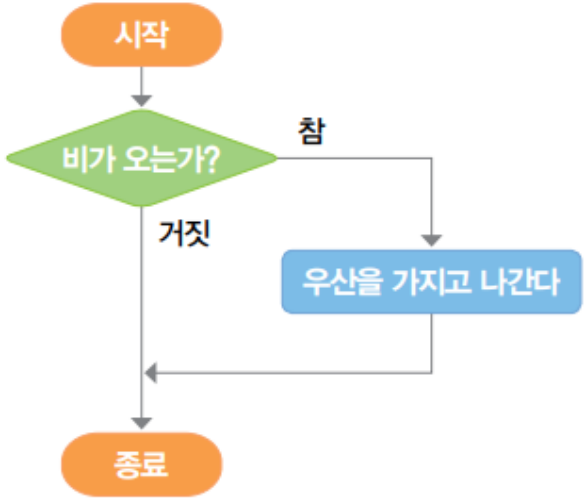
Concept of conditional statement

- A conditional statement lets you check a condition and perform an action accordingly
  - if the condition is true ➔ the block of code inside the conditional statement will execute
  - if the condition is false ➔ the code block is skipped, and the program continues with the next of the code

- Flowchart of the simple conditional statement



- Types of conditional statement
  - ternary operator (: ?)
  - if-else statement
  - switch-case statement



Simple ‘if’ statement

- The ‘if’ statement is the simplest form of control flow statement that decides whether a certain statement or block of code will be executed
  - syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

    - ‘condition’: a boolean expression that evaluates to true or false
      - the condition must always be enclosed in parentheses
    - block of code: if the condition is true, the block of code inside the curly braces { } will be executed. If the condition is false, the block of code will be skipped

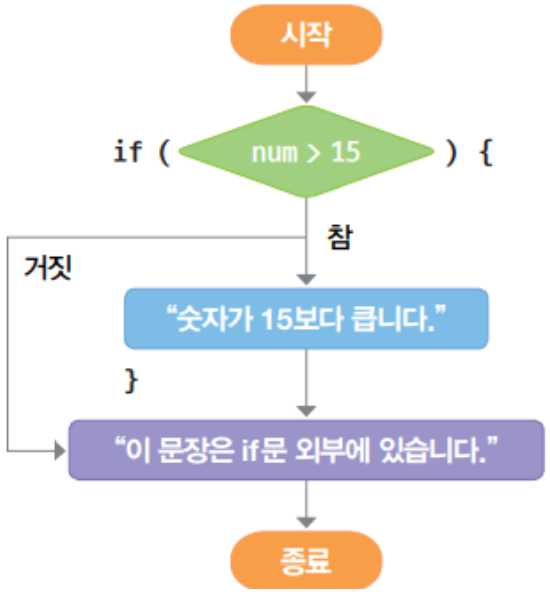
Simple ‘if’ statement

- Conditions within parentheses
  - the condition inside the parentheses can be any expression that results in a boolean value
  - examples
    - comparisons: ==, !=, <, >, <=, >=
    - logical operations: &&, ||, !
    - boolean values and literals: true, false
    - method that return boolean values

Simple ‘if’ statement

- Examples of simple ’if’ statement
  - with the flow diagram

```
int num = 10;

if (num > 15) {
  System.out.println("숫자가 15보다 큽니다.");
}
System.out.println("이 문장은 if 문 외부에 있습니다. ");
```



Simple ‘if’ statement

- Examples of simple ’if’ statement
  - expect the results

```
int a = 4;
int b = 5;

if (a < b) {
  System.out.println("a is less than b");
}
```

```
boolean isEligible = true;

if (isEligible) {
  System.out.println("You can apply for the position");
}
```

Simple ‘if’ statement

- Examples of simple ’if’ statement
  - expect the results

```
boolean isRaining = true;
boolean hasUmbrella = false;

if (isRaining && !hasUmbrella) {
  System.out.println("Take an umbrella");
}
```

Simple ‘if’ statement

- Examples of simple ’if’ statement
  - expect the results

```
int hour = 10;
boolean morningCoffee = false;

// if (hour < 14 && morningCoffee == false) {
if (hour < 14 && !morningCoffee) {
  System.out.println("Please iced americano");
}
System.out.println("Coffee order completed");
```

Simple 'if' statement

- Examples of simple 'if' statement
  - expect the results

```
int score = 85;
if (score > 75)
  System.out.println("Passed");
```

    - when the 'if' statement's body consists of only one statement ➔ without the curly braces

```
int score = 85;
if (score > 75) {
  System.out.println("Passed");
  System.out.println("Congratulations!");
}
```

Simple ‘if’ statement

- Examples of simple ’if’ statement
  - expect the results

```
int number = 10;

if (number > 0) {
  System.out.println("The number is positive.");
}
```

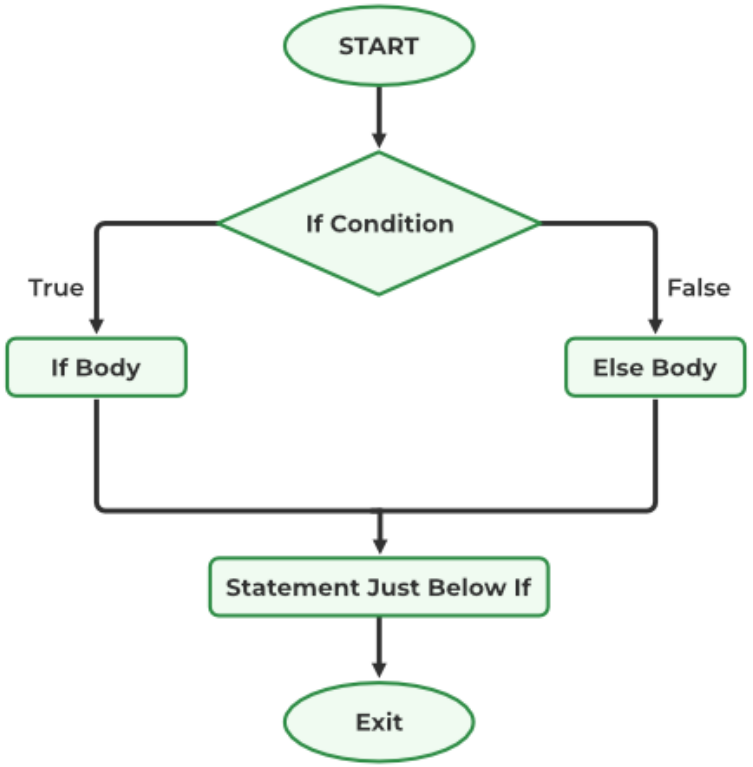
```
int number = 10;

if (number > 0) {
  System.out.println("The number is positive.");
}

System.out.println("The number is negative.");
```

‘if-else’ statement

- The ‘if-else’ statement provide two paths of execution
  - one if the condition evaluates to true and another if the condition evaluate to false
  - syntax

```
if (condition) {
 // Block of code executes if the condition is true
} else {
 // Block of code executes if the condition is false
}
```

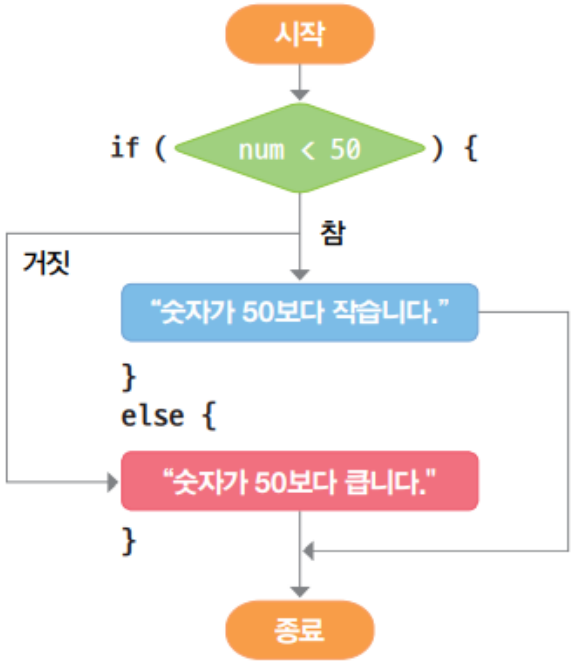
- Flow diagram of ‘if-else’ statement



‘if-else’ statement

- Example of ‘if-else’ statement
  - with the flow diagram

```
int num = 120;

if (num < 50) {
  System.out.println("숫자가 50보다 작습니다");
} else {
  System.out.println(" 숫자가 50보다 큽니다.");
}
```



‘if-else’ statement

- Example of ‘if-else’ statement

```
int temperature = 20;

if (temperature > 25) {
  System.out.println("It's warm outside.");
} else {
  System.out.println("It's cool outside.");
}
```

```
int hour = 10;

if (hour < 14) {
  System.out.println("Please iced americano");
} else {
  System.out.println("Please iced americano (Decaffeination)");
}
System.out.println("Coffee order completed");
```

Examples and practices for condition statement

- 사용자로부터 나이를 입력받아서 20대인지 여부를 판단하고 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/ConditionPractice01.java
  - requirements
    - input (age) is user input by keyboard
    - range of twenties: 20<=age<30
  - input and output examples

```
Enter the age: 27
Yes
```

```
Enter the age: 35
No
```

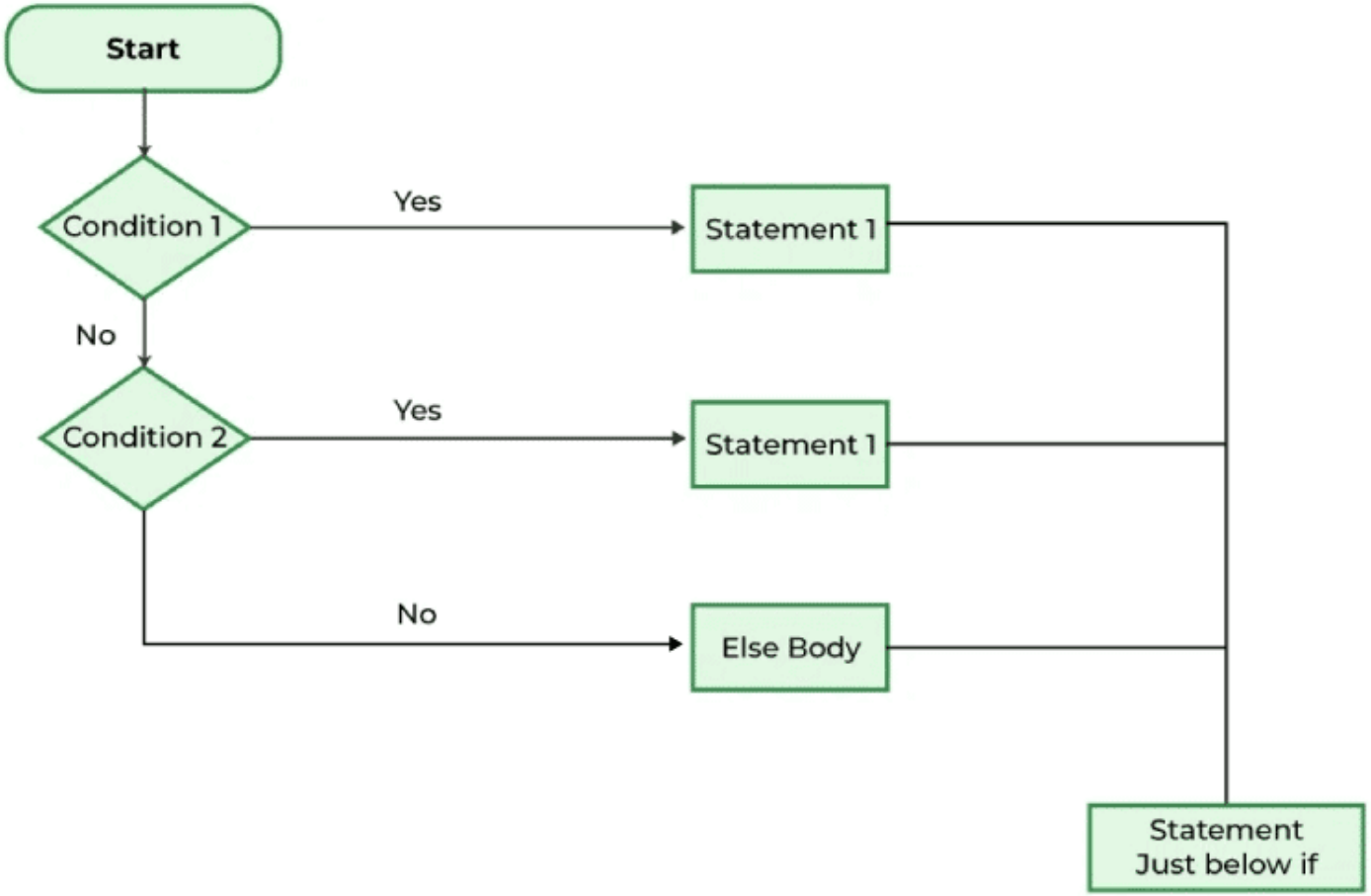
‘if-else if-else’ statement

- The ‘if-else if-else’ statement allows for multiple conditions to be evaluated in sequence
  - if the first ‘if’ condition is false, the program checks the condition in the next ‘else if’, and so on
  - if non of the conditions is true, the final ‘else’ block is executed
    - the ‘else’ block is optional
  - syntax

```
if (condition1) {
  // Block of code executes if condition1 is true
} else if (condition2) {
  // Block of code executes if condition1 is false and condition2 is true
} else if (condition3) {
  // Block of code executes if condition1 & condition2 are false and condition3 is true
} else {
  // Block of code executes if none of the above conditions is true
}
```

‘if-else if-else’ statement

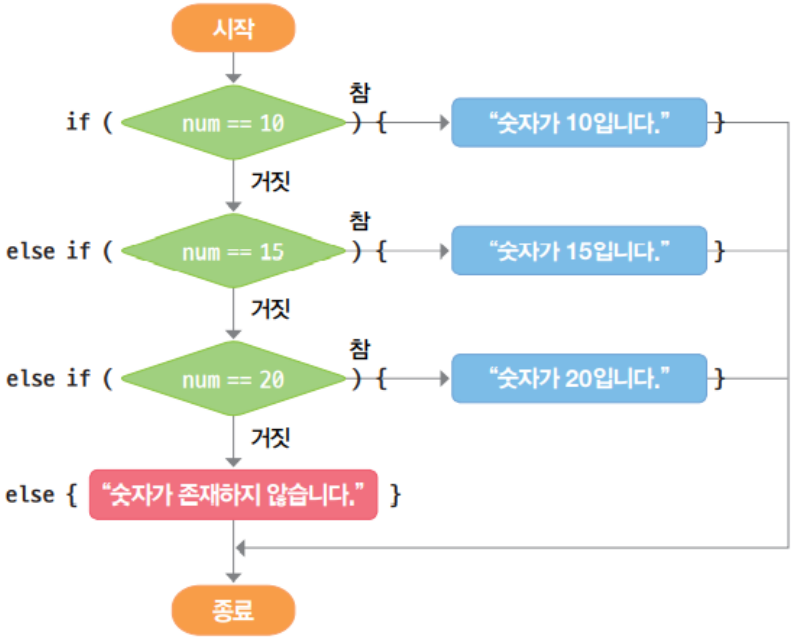
- Flow diagram of ’if-else if-else’ statement



‘if-else if-else’ statement

- Example of ‘if-else if-else’ statement
  - with the flow diagram

```
int num = 20;

if (num == 10) {
  System.out.println("숫자가 10입니다.");
} else if (num == 15) {
  System.out.println ("숫자가 15입니다 ");
} else if (num == 20) {
  System.out.println ("숫자가 20입니다 ");
} else {
  System.out.println ("숫자가 존재하지 않습니다 ");
}
```



‘if-else if-else’ statement

- Example of ‘if-else if-else’ statement

```
int score = 75;

if (score >= 90) {
  System.out.println("Grade A");
} else if (score >= 80) {
  System.out.println("Grade B");
} else if (score >= 70) {
  System.out.println("Grade C");
} else if (score >= 60) {
  System.out.println("Grade D");
} else {
  System.out.println("Grade F");
}
```

```
Grade C
```

Examples and practices for condition statement

- 사용자로부터 나이를 입력받아서 그 사람의 연령대를 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/ConditionPractice02.java
  - requirements
    - input (age) is user input by keyboard
    - age groups
      - kid (ages less than 13), teenager (ages 13-19), twenties (ages 20-29), thirties (ages 30-39), forties (ages 40-49), fifties (ages 50-59), sixties and over (age 60 and above)
  - input and output examples

```
Enter your age: 50
You are in your Fifties.
```

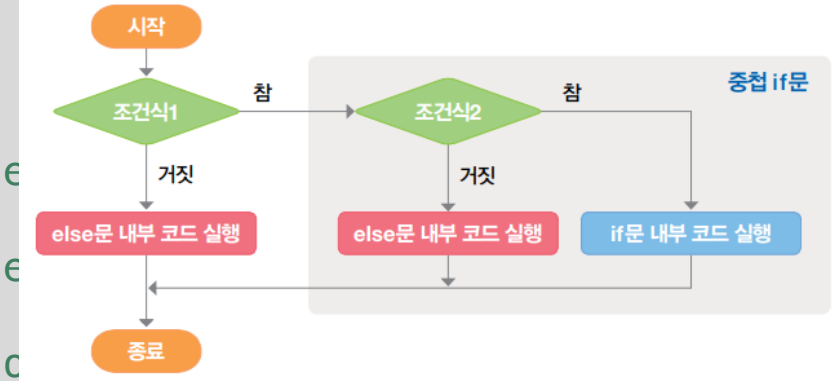
```
Enter your age: 21
You are in your Twenties.
```

```
Enter your age: 2
You are a Kid.
```

Nested ‘if’ statement

- A nested ‘if’ statement is an ‘if’ statement within another ‘if’ statement
  - useful to perform a series of checks that depend on the previous condition being true
  - syntax

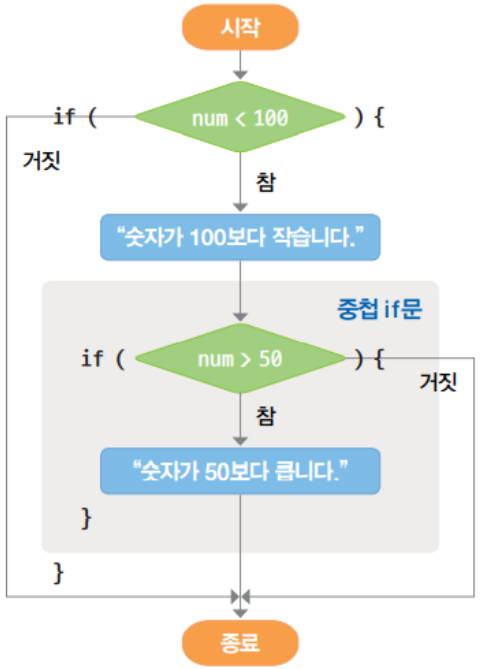
```
if (condition1) {
  // Block of code to execute if condition1 is true
  if (condition2) {
    // Block of code to execute if both condition1 & 2 are true
  } else if (condition3) {
    // Block of code to execute if both condition1 & 3 are true
  } else {
    // Block of code executes if condition1 is true, but none c
  }
  ...
}
else {
  if (condition k) {
    // Block of code to execute if condition1 is false, but condition k is true
  }
  ...
}
```



Nested ‘if’ statement

- Example with the flow diagram

```
int num = 70;

if (num < 100) {
  System.out.println("숫자가 100보다 작습니다.");
  if (num < 50) {
    System.out.println( " 숫자가 50보다 큽니다.");
  }
}
```



Nested ‘if’ statement

- Example

```
int a = 10;
int b = 20;
int c = 30;

if (a < b) {
  if (b < c) {
    System.out.println("a is less than b and b is less than c");
  }
}
```

Nested ‘if’ statement

- Example

```
int score = 85;

if (score >= 50) {
  System.out.println("You passed.");
  if (score >= 90) {
    System.out.println("Grade: A");
  } else if (score >= 75) {
    System.out.println("Grade: B");
  } else {
    System.out.println("Grade: C");
  }
} else {
  System.out.println("You failed.");
}
```

  - first checks if the score is 50 or more. If true, it prints “You passed.”
  - then, within the first if block, it further categorizes the score into grades A, B, or C
  - if the score is less than 50, it skips all the nested checks and directly prints “You failed.”

Nested ‘if’ statement

- Example

```
boolean isMember = true;
int purchaseAmount = 750;

if (isMember) {
  if (purchaseAmount > 500) {
    System.out.println("Eligible for 20% discount");
  } else if (purchaseAmount > 200) {
    System.out.println("Eligible for 10% discount");
  } else {
    System.out.println("Eligible for 5% discount");
  }
} else {
  System.out.println("Please sign up for membership for discount eligibility");
}
```

```
Eligible for 20% discount
```

Examples and practices for condition statement

- 사용자로부터 시간을 입력받고, 시간대에 따라 적절한 인사말을 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/ConditionPractice03.java
  - requirements
    - the greeting should be “Good Morning” for times before noon
      “Good Afternoon” for times for noon until 6 PM
      “Good Evening” for times after 6 PM
    - should also validate that the provided hour is within a 24-hour day
  - input and output examples

```
Enter the time of day: 15
Good Afternoon
```

```
Enter the time of day: 18
Good Evening
```

```
Enter the time of day: 25
Invalid hour of the day
```

Nested ‘if’ statement

- Practical use of the nested ‘if’ statement
  - limit nesting depth
    - deeply nested ‘if’ statements can make the code hard to read and maintain
    - consider using logical operators to combine conditions or breaking down complex conditions into functions or methods for clarity
  - use comments
    - when nesting becomes necessary, use comments to explain the logic, especially if the conditions are not immediately clear
  - consider alternatives
    - for some cases, using a ‘switch’ statement or polymorphism might be a cleaner alternative to deeply nested ‘if’ structure

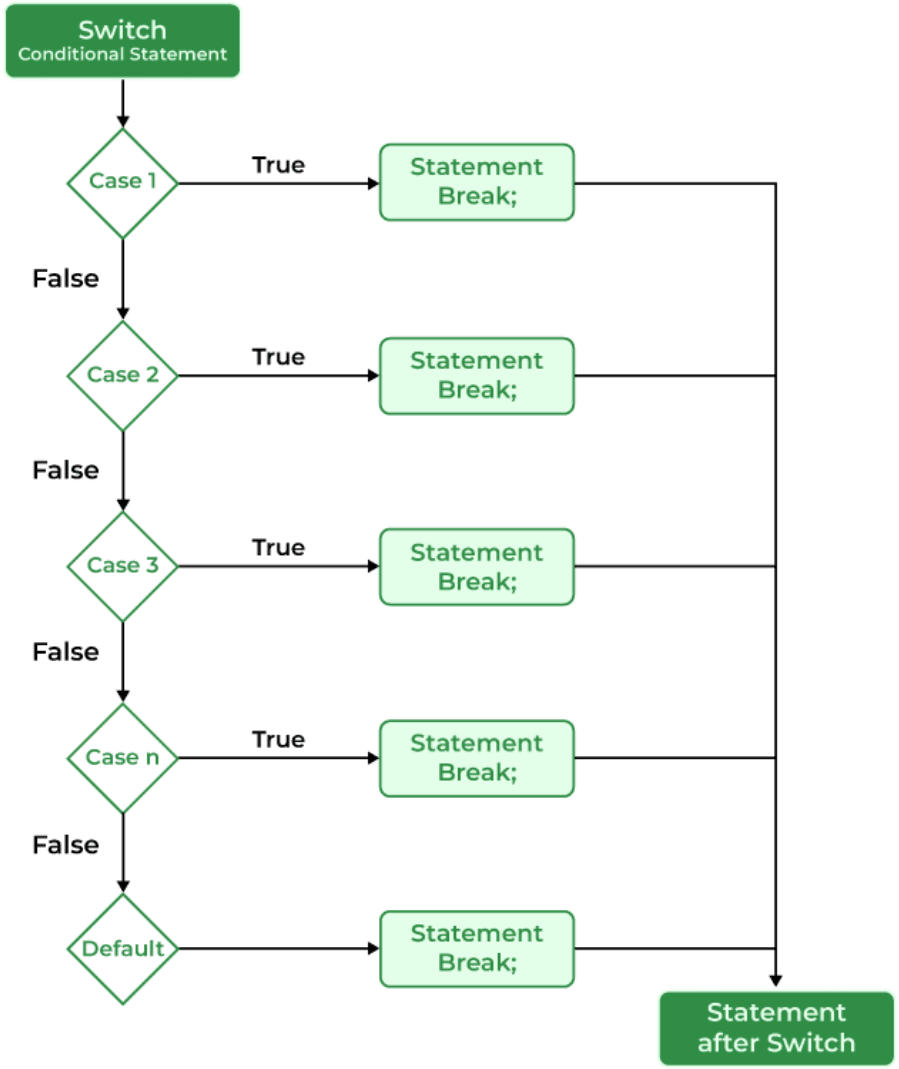
‘switch-case’ statement

- ‘Switch-case’ statement provides a way to execute different parts of code based on the value of an expression
  - a more efficient and cleaner way to write a sequence of ’if-else if-else’ statements
  - syntax

```
switch (expression) {
  case value1:
      // Code to execute when expression equals value1
    break; // Optional

  case value2:
    // Code to execute when expression equals value2
    break; // Optional
    ...

  default:
   // Code to execute if expression does not match any case
}
```

‘switch-case’ statement

- Flow diagram of ‘switch-case’ statement
  - ‘expression’: this is evaluated once, and its result is compared with the values of each ‘case’
  - ‘case’: if the expression matches a ‘case’ value, the block of code following that ‘case’ is executed
  - ‘break’: this statement is used to exit the switch block
  - ‘default’: this block is optional and executes if none of the ‘case’ values match the expression



‘switch-case’ statement

- Example

```
int day = 4;

switch (day) {
  case 1: System.out.println("Monday");
    break;
  case 2: System.out.println("Tuesday");
    break;
  case 3: System.out.println("Wednesday");
    break;
  case 4: System.out.println("Thursday");
    break;
  case 5: System.out.println("Friday");
    break;
  case 6: System.out.println("Saturday");
    break;
  case 7: System.out.println("Sunday");
    break;
  default: System.out.println("Invalid day");
}
```

‘switch-case’ statement

- Example

```
String month = "JANUARY";

switch (month) {
  case "JANUARY":
    System.out.println("It's the first month of the year.");
    break;
  case "JUNE":
    System.out.println("It's the sixth month of the year.");
    break;
  default:
    System.out.println("It's some other month.");
}
```

‘switch-case’ statement

- The type of expression that can be used in a ‘switch’ statement includes
  - primitive data types: byte, short, char, and int
  - wrapper classes: Byte, Short, Character, and Integer
  - enumerated types: enum
  - String classes
- Examples

```
int day = 3;

switch (day) {
  case 1:
    ...
  case 2:
    ...
  case 3:
    ...
}
```

```
String month = "JANUARY";

switch (month) {
  case "JANUARY":
    ...
  case "JUNE":
    ...
}
```

‘switch-case’ statement

- The ‘break’ keyword
  - to terminate a statement sequence
  - to exit the ‘switch’ statement once a matching case has been executed
  - if ‘break’ is omitted after a case, execution will continue into the next case regardless of whether it matches the expression (fall through)

‘switch-case’ statement

- Example

```
int number = 2;

switch (number) {
  case 1:
    System.out.println("One");
    break; // Exits the switch block
  case 2:
    System.out.println("Two");
    // Missing break, so execution would continue to the next case if present
  case 3:
    System.out.println("Three");
    break;
  default:
    System.out.println("Something else");
}
```

```
Two
Three
```

‘switch-case’ statement

- Example

```
int stage = 1;

switch (stage) {
  case 1:
    System.out.println("Start water");
    // Fall through
  case 2:
    System.out.println("Shampoo hair");
    // Fall through
  case 3:
    System.out.println("Rinse hair");
    // No break, so execution continues
  case 4:
    System.out.println("Stop water");
    break;
  default:
    System.out.println("Invalid stage");
}
```

```
Start water
Shampoo hair
Rinse hair
Stop water
```

‘switch-case’ statement

- The ‘default’ case in a ‘switch’ statement is optional and serves as a “catch-all” when none of the case values match the switch expression
  - similar with ‘else’ in ‘if-else’ statements
  - it can appear anywhere within the ‘switch’ block but typically is used at the end

```
int day = 5;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  default:
    System.out.println("Another day");
    // Executed if none of the above cases match
}
```

```
Another day
```

‘switch-case’ statement

- Converting ‘if-else’ statement to ‘switch-case’ statements
  - ‘switch-case’ statements can make some types of logical more readable
  - however, less flexible than ‘if-else’; only allowing equality; (‘if-else’ can handle ranges or conditions)
  - all ‘switch-case’ statement can be converted into ‘if-else’ statement, but the opposite conversion is difficult

‘switch-case’ statement

- Example

```
int day = 3;

switch (day) {
  case 1:
    System.out.println("Monday");
  break;
  case 2:
    System.out.println("Tuesday");
  break;
  case 3:
    System.out.println("Wednesday");
  break;
  // Additional cases...
  default:
    System.out.println("Some other day");
}
```

```
int day = 3;

if (day == 1) {
  System.out.println("Monday");
} else if (day == 2) {
  System.out.println("Tuesday");
} else if (day == 3) {
  System.out.println("Wednesday");
  // Additional conditions...
} else {
  System.out.println("Some other day");
}
```

‘switch-case’ statement

- How can we convert into ‘switch-case’ statements?

```
int temperature = 41;

if (temperature > 30) {
  System.out.println("It's a hot day.");
} else if (temperature > 20) {
  System.out.println("It's a warm day.");
} else {
  System.out.println("It's a cold day.");
}
```

Examples and practices for condition statement

- 사용자로부터 숫자 두 개와 하나의 연산(operator)를 입력받고, 그 연산의 결과를 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/ConditionPractice04.java
  - requirements
    - 0으로 나눌 경우 적절한 에러 메시지를 출력해야한다.
  - input and output

```
Enter the first number: 3
Enter the second number: 5
Enter an operator (+, -, *, /): *
Result: 15.00
```

```
Enter the first number: 9
Enter the second number: 0
Enter an operator (+, -, *, /): /
Error: Division by zero is not allowed.
```

```
Enter the first number: 9
Enter the second number: 0
Enter an operator (+, -, *, /): +
Result: 9.00
```

3. Loop statement

Concept of loop

- Loops are fundamental control flow statements
  - allows code to be executed repeatedly based on a condition
  - example to print the sentences repeatedly without loop statement

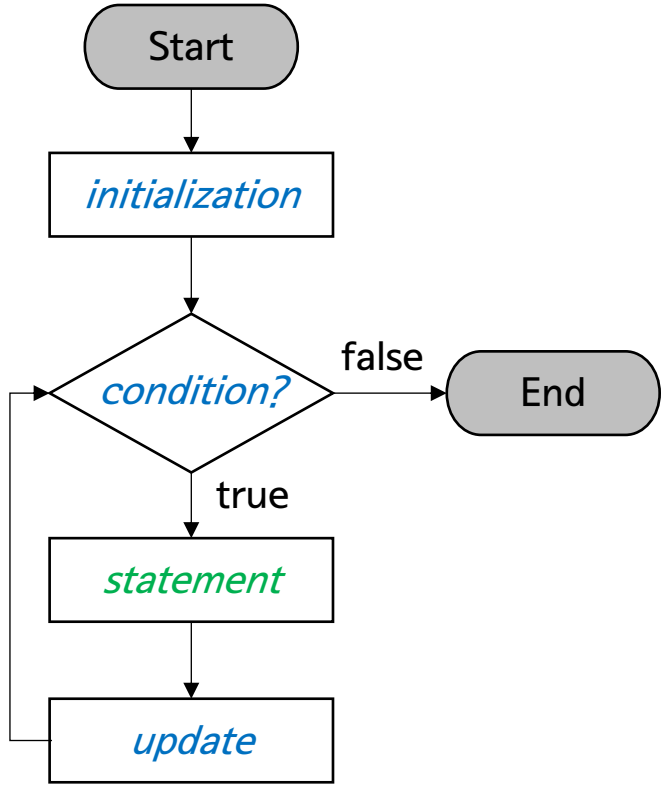
```
System.out.println("Thank you, 1");
System.out.println("Thank you, 2");
System.out.println("Thank you, 3");
System.out.println("Thank you, 4");
System.out.println("Thank you, 5");
System.out.println("Thank you, 6");
System.out.println("Thank you, 7");
System.out.println("Thank you, 8");
System.out.println("Thank you, 9");
System.out.println("Thank you, 10");
```

  - the primary loop constructs in
    - ‘for’, ‘while’, and ‘do-while’ loop statements

‘for’ loop statement

- ‘for’ loop is used to execute a block of code a specific number of times
  - particularly useful when the number of iterations is known before entering the loop
  - syntax

```
for (initialization; condition; update) {
   // Block of code to be executed
}
```



    - ‘initialization’
      - this step is executed only once, setting up the loop variable
    - ‘condition’
      - before each iteration, this condition is checked
      - if it evaluate to true, the loop continues; if false, the loop ends
    - ‘update’
      - executes after each iteration, typically used to update the loop variable

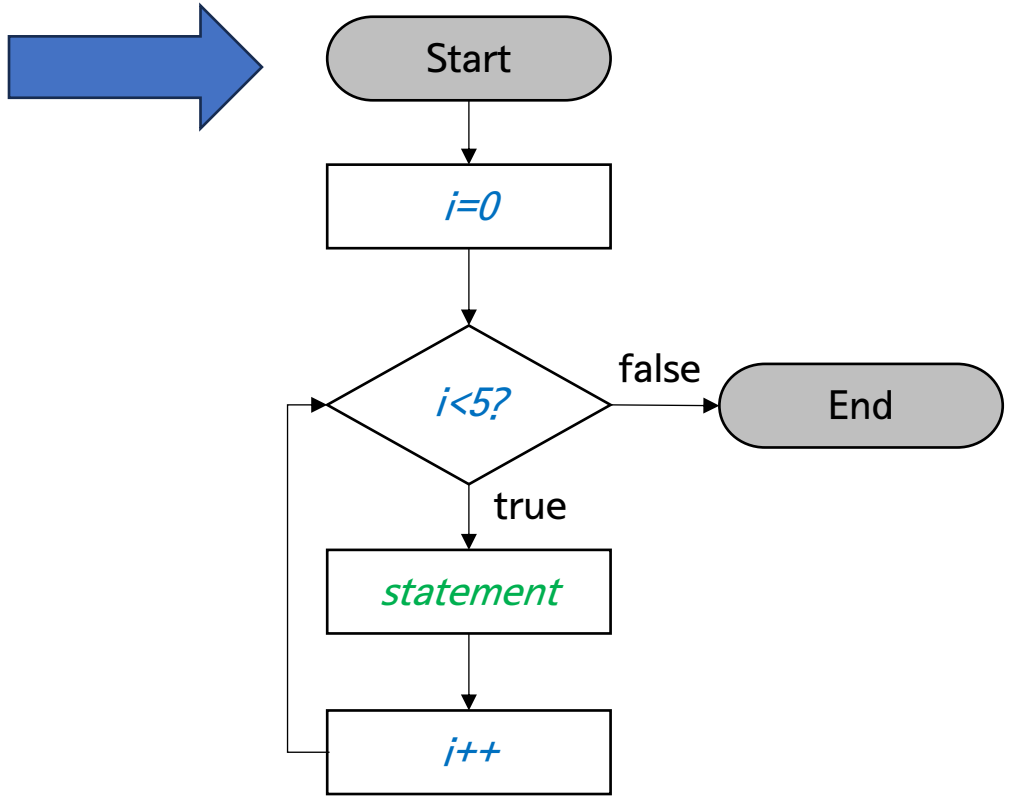
‘for’ loop statement

- Simple examples and flow diagram

```
for (int i = 0; i < 5; i++) {
  System.out.println("i is: " + i);
}
```

```
i is: 0
i is: 1
i is: 2
i is: 3
i is: 4
```

loop variable




```
for (int i = 0; i <= 10; i++) {
  System.out.println("i is: " + i);
}
```

```
for (int i = 1; i <= 10; i++) {
  System.out.println("i is: " + i);
}
```

```
for (int i = 0; i <= 10; ++i) {
  System.out.println("i is: " + i);
}
```

‘for’ loop statement

- Simple examples and flow diagram

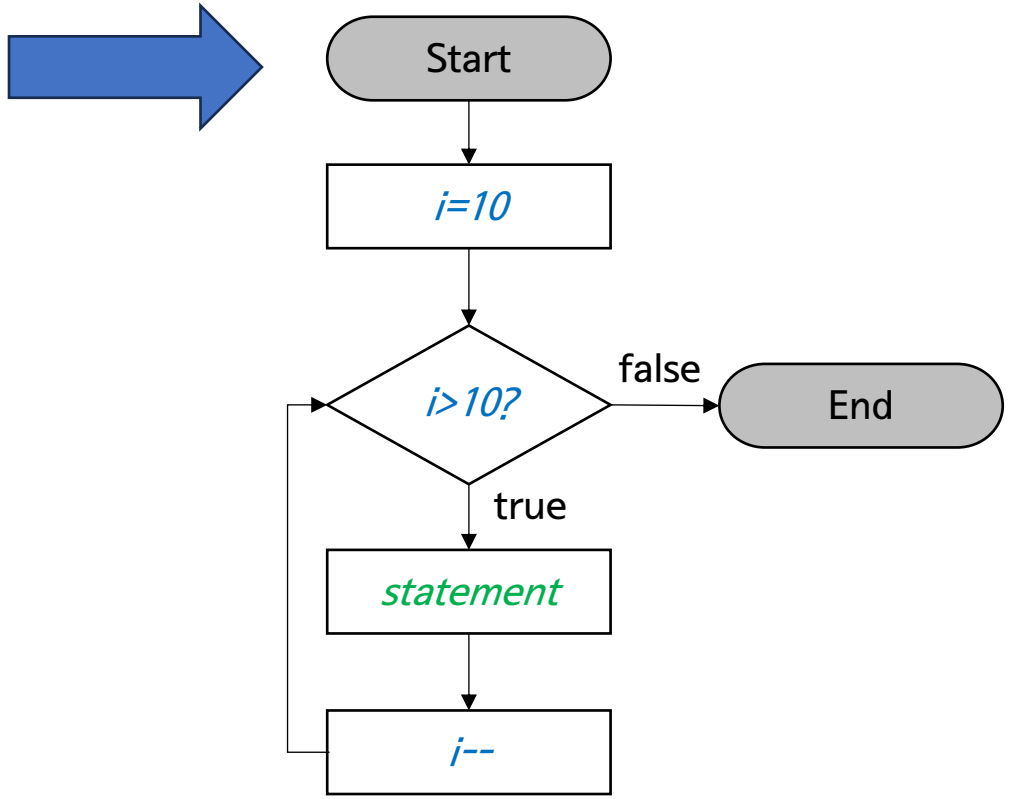
```
for (int i = 10; i >= 0; i--) {
  System.out.println("i is: " + i);
}
```

```
for (int i = 10; i > 0; i--) {
  System.out.println("i is: " + i);
}
```

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

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

```
i is: 1
i is: 2
i is: 4
i is: 8
```



‘for’ loop statement

- Example for calculating the sum of the first 10 natural numbers

```
int result=0;
for (int i = 1; i <= 10; i++) {
  result += i;
}
System.out.println("Sum of the first 10 natural numbers is: " + result);
```

```
Sum of the first 10 natural numbers is: 55
```

  - if we use `result` variable without initialization as 0, we got error below

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
The local variable result may not have been initialized
The local variable result may not have been initialized

at Chap03Example.LoopExample01.main(LoopExample01.java:12)
```

- How about factorial? (e.g., 1*2*3*....*(n-1)*n)

‘for’ loop statement

- Example for printing the multiplication table for a specific number

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

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

‘for’ loop statement

- Example for printing the even numbers between 1 and 20

```
for (int i = 1; i <= 20; i++) {
  if (i % 2 == 0) {
    System.out.println(i + " is even.");
  }
}
```

```
2 is even.
4 is even.
6 is even.
8 is even.
10 is even.
12 is even.
14 is even.
16 is even.
18 is even.
20 is even.
```

Examples and practices for loop statement

- 사용자로부터 양의 정수 1개(N)를 입력받고, 1부터 N까지의 정수의 합과 그 식을 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/LoopPractice01.java
  - requirements
    - use a loop statement to calculate the sum of numbers from 1 to N
    - display the sequence of the numbers from 1 to N, each separated by a '+', and ends with '='
  - input and output examples

```
Enter a number: 10
1+2+3+4+5+6+7+8+9+10=55
```

```
Enter a number: 15
1+2+3+4+5+6+7+8+9+10+11+12+13+14+15=120
```

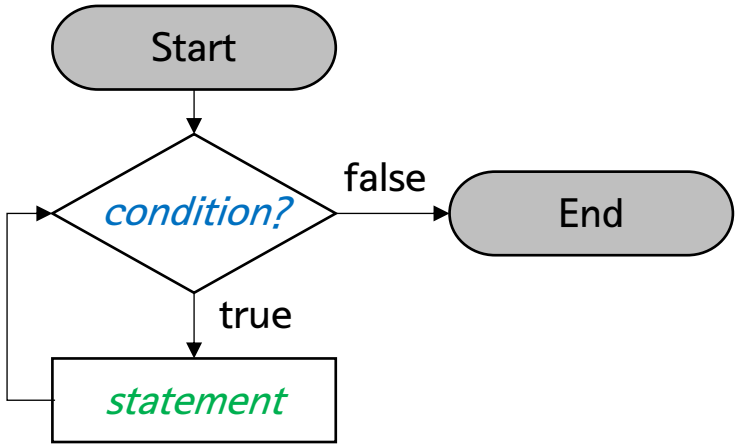
  - HINT: use if-else statement; print '+' or '=' depending on current number

‘while’ loop statement

- The ‘while’ loop is a control flow statement that allows code to be executed repeatedly based on given boolean condition
  - the loop will continue executing until the condition evaluates to false
  - ‘while’ loop is useful when the number of iterations is not known before entering the loop
  - syntax

```
while (condition) {
    // Block of code to be executed
}
```

  - ‘condition’: boolean expression that must be true for the loop to execute
  - after each iteration, the condition is evaluated, and if it is true, the loop executes again



‘while’ loop statement

- Example

```
int i = 1; // Initialization
while (i <= 10) { // Condition
  System.out.print(i + " "); // Loop body
  i++; // Increment
}
```

```
1 2 3 4 5 6 7 8 9 10
```

  - the loop starts with ‘i’ equal to 1 (similar with loop variable in ‘for’ statement)
  - after each iteration, ‘i’ is incremented by 1
  - the loop continues until ‘i’ is greater than 10, at which point the condition ‘i<=10’ evaluates to false and the loop exists

- Note: always ensure that initialization and increment parts when you use the ‘while’ loop

‘while’ loop statement

- Example for calculating average of a sequence of integer numbers entered by the user
  - the user can enter the numbers until the user input 0

```
Scanner scanner = new Scanner(System.in);
int count = 0, n = 0;
double sum = 0.0;
System.out.println("Enter the integer numbers, and enter the 0 at the last");
while ( (n = scanner.nextInt()) != 0 ) {
  sum += n;
  count++;
}
System.out.println("Average = " + sum/count);
```

```
Enter the integer numbers, and enter the 0 at the last
1
2
3
4
5
0
Average = 3.0
```

‘while’ loop statement

- Infinite loop

```
while (true) {
  // This code runs forever unless interrupted by a break statement or an external event.
}
```

  - infinite loops are useful when a program should run continuously until it is manually stopped or when it is waiting for an event that will terminate the loop internally, such as a user input
  - important points
    - always ensure that the loop

‘while’ loop statement

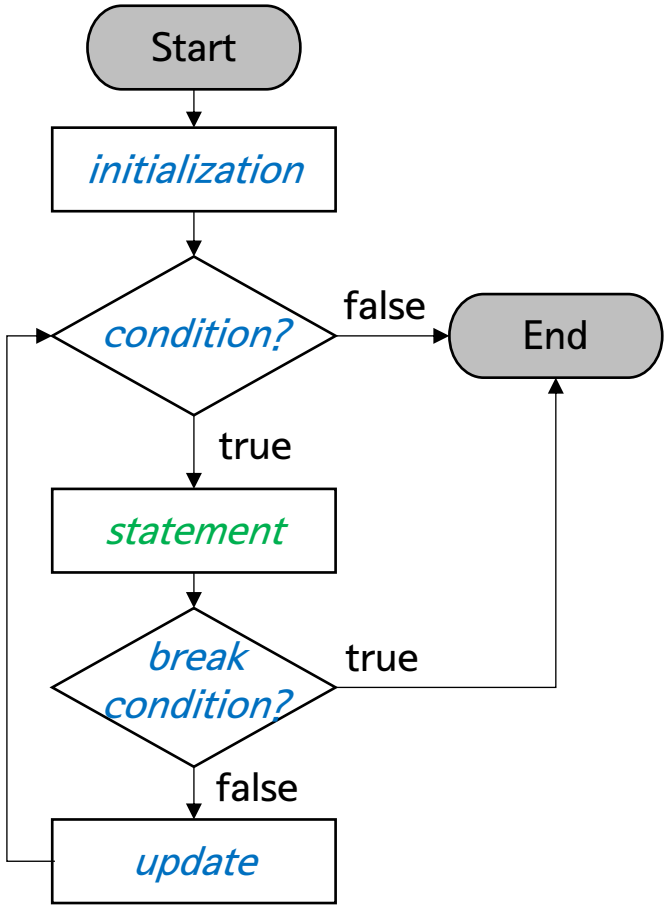
- Example of infinite loop

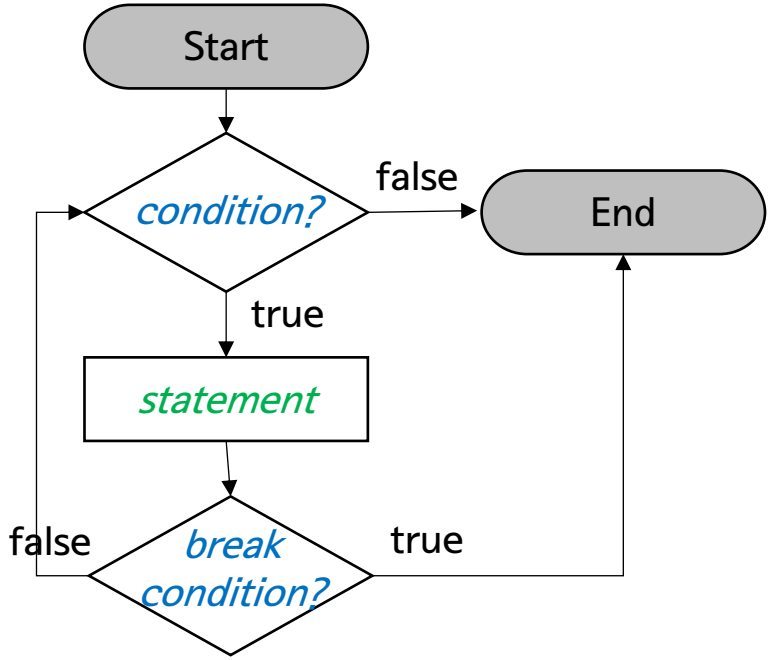
```
Scanner scanner = new Scanner(System.in);
int inputNumber;

while (true) {
  System.out.println("Enter a positive number to
continue or -1 to exit:");
  inputNumber = scanner.nextInt();

  if (inputNumber == -1) {
    break;
  }

  System.out.println("You entered: " +
inputNumber);
}
System.out.println("Loop terminated by user
request.");

scanner.close();
```

```
Enter a positive number to continue or -1 to
exit:
5
You entered: 5
Enter a positive number to continue or -1 to
exit:
13
You entered: 13
Enter a positive number to continue or -1 to
exit:
-4
You entered: -4
Enter a positive number to continue or -1 to
exit:
-1
Loop terminated by user request.
```

‘break’ statement

- ‘break’ statement exists the loop immediately, regardless of the loop’s condition
  - after a ’break’ statement is executed, control moves to the line of code immediately following the loops’ block





‘break’ statement

- Example – for loop

```
for (int i = 1; i <= 10; i++) {
  if (i == 5) {
    break; // Exit the loop when i is 5
  }
  System.out.println(i);
}
```

```
1
2
3
4
```

‘break’ statement

- Example – while loop

```
int count = 0;
while (true) {
  if (count == 5) {
    break;
  }
  System.out.println("Count: " + count);
  count++;
}
```

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
```

‘continue’ statement

- ‘continue’ statement skips the remaining code in the current iteration and proceeds with the next iteration of the loop
  - the loop’s condition is tested immediately after a ’continue’ statement is executed

‘continue’ statement

- Example – for loop

```
for (int i = 1; i <= 10; i++) {
  if (i % 2 == 0) {
    continue; // Skip even numbers
  }
  System.out.println(i);
}
```

```
1
3
5
7
9
```

‘continue’ statement

- Example – while loop

```
int i = 0;
int max = 10;

while (i < max) {
  i++;
  if (i % 2 == 0) {
    continue;
}

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

```
1
3
5
7
9
```

Examples and practices for while loop statement

- 사용자로부터 3의 배수가 나올 때까지 숫자를 입력받고, 숫자의 합을 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/LoopPractice02.java
  - requirements
    - 'while' statement를 활용하여라
    - 입력받은 수가 3의 배수인 경우 덧셈을 수행하지 않고, 최종결과를 출력하고 프로그램을 종료한다.
  - input and output examples

```
Enter a number: 1
Current sum = 1
Enter a number: 2
Current sum = 3
Enter a number: 4
Current sum = 7
Enter a number: 5
Current sum = 12
Enter a number: 9
Total sum = 12
```

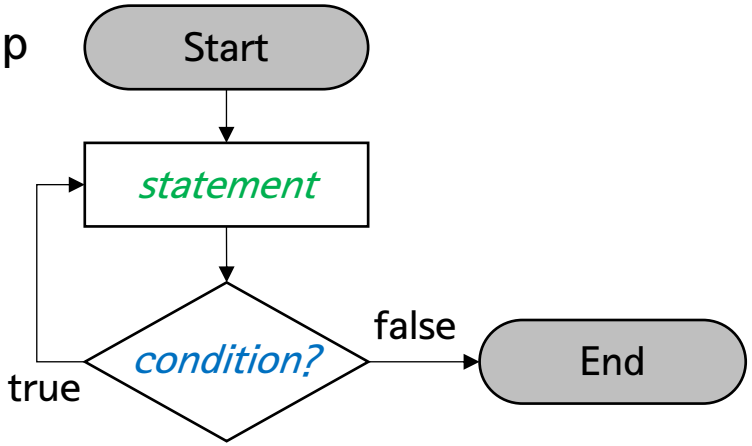
‘do-while’ loop statement

- A ‘do-while’ loop is post-test loop that will execute its body at least once before any condition is tested
  - differentiates it from a regular ‘while’ loop that tests its condition at the beginning
  - useful when the code inside the loop needs to be executed at least once regardless of the condition
  - syntax

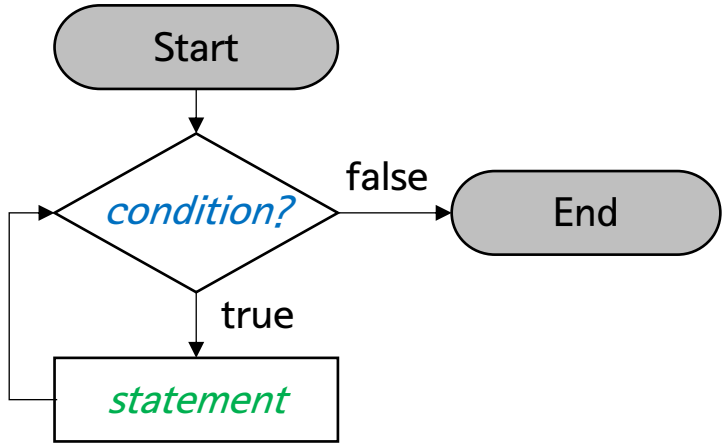
```
do {
    // Block of code to be executed
} while (condition);
```

  - flow diagram (vs. ‘while’ loop)

‘do-while’ loop



‘while’ loop



‘do-while’ loop statement

- Example

```
char a = 'a';
do {
  System.out.print(a);
  a = (char) (a+1);
} while(a <= 'z');
```

```
abcdefghijklmnopqrstuvwxyz
```

  - RECALL: “Casting”

‘do-while’ loop statement

- Example

```
Scanner scanner = new Scanner(System.in);
int number;
int sum = 0;

do {
  System.out.print("Enter a number (or enter 0 to finish): ");
  number = scanner.nextInt();
  sum += number;
} while (number != 0);

System.out.println("Sum of entered numbers is: " + sum);

scanner.close();
```

```
Enter a number (or enter 0 to finish): 50
Enter a number (or enter 0 to finish): 12
Enter a number (or enter 0 to finish): -12
Enter a number (or enter 0 to finish): 0
Sum of entered numbers is: 50
```

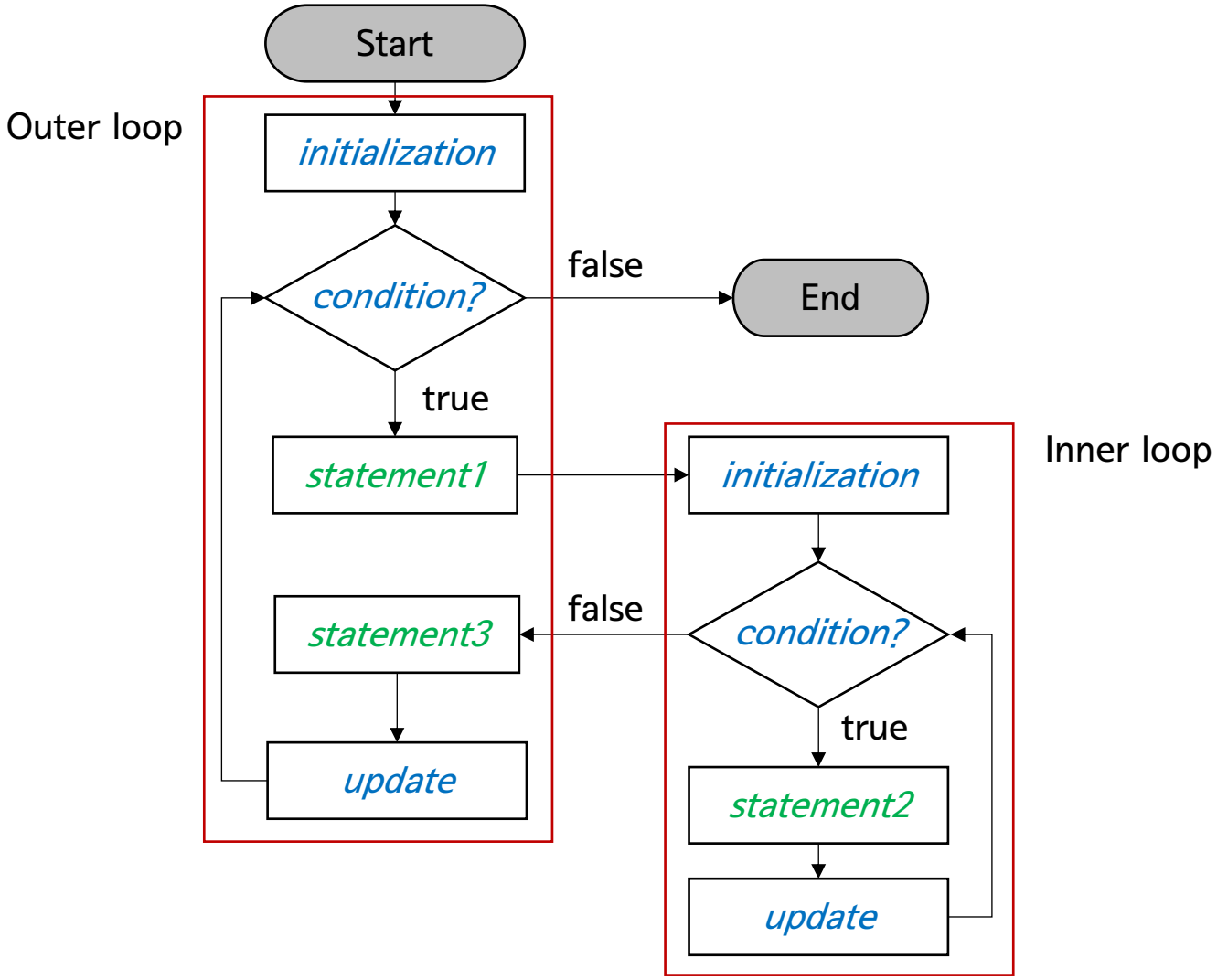
Nested loop statement

- Nested loops are a series of loops where one loop is situated inside the body of another loop
  - commonly used when dealing with multi-dimensional data structure
    - e.g., array, matrices,
  - or when you need to perform operations that require multiple levels op looping
  - syntax

```
for (initialization1; condition1; update1) {
  // Outer loop statements (statement1)
  for (initialization2; condition2; update2) {
    // Inner loop statements (statement2)
  }
  // Outer loop statements (statement3)
}
```

Nested loop statement

- Flow diagram of the nested loop statement (nested for loop)



Nested loop statement

- Example

```
for(int i=0; i<3; i++) {
  for (int j=0; j<5; j++) {
    System.out.println("i = " + i + ", j = " + j);
  }
}
```

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 0, j = 3
i = 0, j = 4
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 1, j = 4
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 2, j = 4
```

Nested loop statement

- Example

```
int width = 5;
int height = 3;

for (int i = 0; i < height; i++) {
  for (int j = 0; j < width; j++) {
    System.out.print("*");
  }
  System.out.println(); // New line after each row
}
```

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

Nested loop statement

- Example

```
for (int i = 1; i <= 10; i++) {
  for (int j = 1; j <= 10; j++) {
    System.out.printf("%4d", i * j);
  }
  System.out.println();
}
```

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

Nested loop statement

- Example

```
for (int i = 1; i <= 9; i++) {
  for (int j = 1; j <= 9; j++) {
    System.out.printf("%d * %d = %d\t", i, j, i * j);
  }
  System.out.println();
}
```

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

Nested loop statement

- Example

```
for (int i = 1; i <= 10; i++) {
  for (int j = 1; j <= 10; j++) {
    System.out.printf("%4d", i * j);
  }
  System.out.println();
}
```

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

‘break’ statement in the nested loop

- Using a ‘break’ statement in a nested loop allows to exit one or more loops prematurely
  - by default, a ‘break’ statement exists only the innermost loop in which in is placed
- Example

```
for (int i = 1; i <= 3; i++) {
  for (int j = 1; j <= 3; j++) {
    if (j == 2) {
      break;
    }
    System.out.println("i = " + i + ", j = " + j);
  }
}
```

```
i = 1, j = 1
i = 2, j = 1
i = 3, j = 1
```

Examples and practices for nested loop statements

- 사용자로부터 양의 정수 하나를 입력받고, 아래와 같은 크기의 숫자 사각형을 출력하는 프로그램을 작성해보세요.
  - file path and name: Chap03Example/LoopPractice03.java
  - input and output examples

```
Enter the number as size of rectangle: 3
  1  2  3
  4  5  6
  7  8  9
```

```
Enter the number as size of rectangle: 5
  1   2   3   4   5
  6   7   8   9  10
 11  12  13  14  15
 16  17  18  19  20
 21  22  23  24  25
```

End of slide