Object – Oriented Programming

Lab #2

Contents

- **Flow Control**
  - Branching Statements
  - Looping Statements

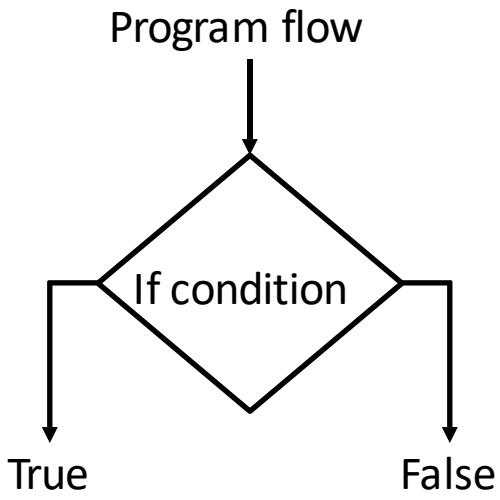
Flow Control

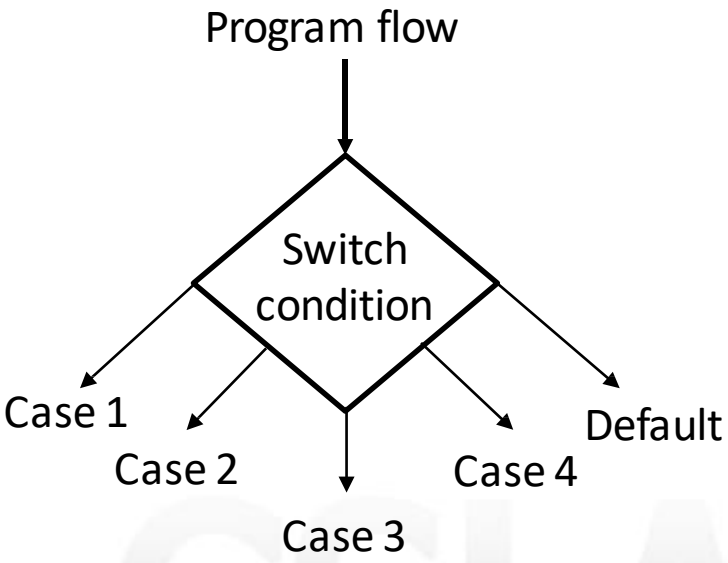
- **An application may use various flow control statements to direct the logic of its execution**

- **Two form of flow control statements are:**
  - Branching Statements
    - if, if – else, if else-if;
    - switch
  - Looping Statements
    - while
    - do-while
    - for

Branching Statements

- **Branch statements are used to divert the execution based on the *Boolean* results of a condition**





if Statements

- **If statements allow one of two branch direction**
  - *if (boolean condition)*
    statement;
- **Multiple statements can be executed by including them within { }**
  - *if (boolean condition)*
    {
    statement1;
    statement2;
    statement3;
    }
- **The statements within the if statement are executed if the condition is true**
- **If the condition is false, the statements are ignored and the flow of the program continues after the statements.**

if else-if Statements

- **It is sometimes necessary to combine multiple if statements.**
- **Use if else-if statements to combine if conditions**

  – *if (boolean condition)*
  statement1;
  else if (boolean condition2)
  statement2;
  else
  statement3;

- **Using a combination of if else-if and else statements it is possible to code complex conditions**

switch Statements

- **Switch statements allow for multiway branching**
- **Switch statements comprise of specific parts**
  - The switch: switch and control statements
  - The cases: each case that must be considered by the switch
  - The default case: if none of the case fit then the default case will match

- **switch-case blocks should have break statements within each case**

switch Statements (Contd)

```
switch (numberOfFlavors)
{
    case 32:
        System.out.println("Case 32");
        break;
    case 1:
        System.out.println("Case 1");
        break;
    case 2: break;
    case 3: break;
    case 4:
        System.out.println("Case 4");
        break;
    default:
        System.out.println("Default Case");
        break;
}
```

Controlling Expression

Case Labels

switch Statements (Contd)

- **There are special considerations when using switch statements.**
  - The controlling expression must evaluate to one of the following types
    - Char
    - Int
    - Short
    - Byte
    - Enum
    - Strings (As of Java 7. Not available if you use an older version of java)
  - The case labels must all be of the same type as the controlling expression
  - The case labels are followed by colons :

switch Statements (Contd)

- **Omitting the breaks;**
  - Execution "falls through" to the next case statement after the end of the statement is reached

```
switch (numberOfFlavors)
{
    case 32:
        System.out.println("Case 32");
        break;
    case 1:
        System.out.println("Case 1");
        break;
    case 2:
        System.out.println("Case 2");
    case 3:
        System.out.println("Case 3");
    case 4:
        System.out.println("Case 4");
        break;
    default:
        System.out.println("Default Case");
        break;
}
```

Case 2

Case 3

Case 4

Self-Test (1)

- **int type 변수 n의 값을 키보드를 통해 입력 받은 후, 입력 받은 n의 값이 어느 범위에 속하는지 판단하는 프로그램을 작성할 것**
  - $n < 0$
    - 출력: n is less than 0
  - $0 \leq n < 100$
    - 출력: n is greater than or equal to 0 and less than 100
  - $n \geq 100$
    - 출력: n is greater than or equal to 100

Loop Statements

- Loops are control structures that allow groups of code to be repeated a number of times
- The statement or group of statements to be repeated is called *body* of the loop
- Each time a loop repeats is called an *iteration* of the loop

Loop Statements (Contd)

- **Control of loop: *ICU***
  1. *Initialization*
  2. *Condition for termination (continuing)*
  3. *Updating the condition*
- **Body of loop**

Loop Statements (Contd)

- the *do-while* loop
- the *while* loop
- the *for* loop

while Statement

- **Also called a *while* loop**

- **A controlling boolean expression**
  - True -> repeats the statements in the loop body
  - False -> stops the loop
  - Initially false (the very first time)
    - loop body will not even execute once

while Statement (Contd)

- **Syntax**

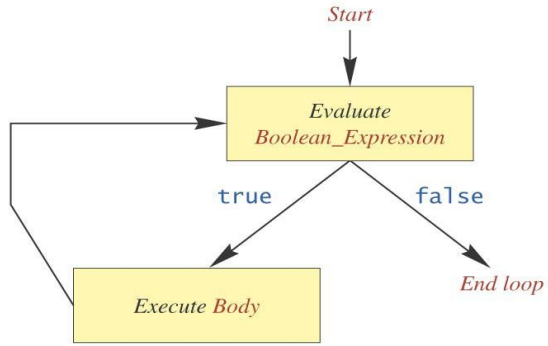
```
while (boolean_expression)
        body_statement

/*-----------------------------------------------------------------------*/

while (boolean_expression)
{
        first_statement
        second_statement
        ...
}
```

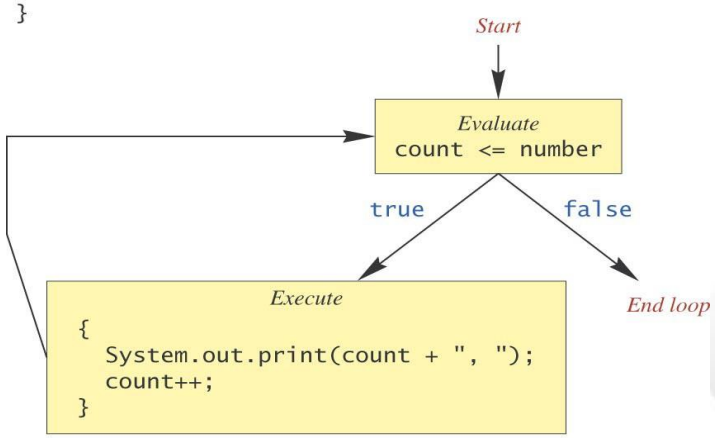
while Statement (Contd)

```
while (Boolean_Expression)
    Body
```



Example:

```
while (count <= number)
{
   System.out.print(count + ", ");
   count++;
}
```



Display 3.7

Semantics of the while Statement[2]

while Statement (Contd)

- ***class WhileDemo***

```java
import java.util.*;

public class WhileDemo
{
    public static void main(String[] args)
    {
        int count, number;

        System.out.println("Enter a number");
        Scanner keyboard = new Scanner(System.in);
        number = keyboard.nextInt();

        count = 1;
        while (count <= number)
        {
            System.out.print(count + ", ");
            count++;
        }

        System.out.println();
        System.out.println("Buckle my shoe.");
    }
}
```

Sample Screen Dialog 1

```
Enter a number:
2
1, 2,
Buckle my shoe.
```

Sample Screen Dialog 2

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

Sample Screen Dialog 3

```
Enter a number:
0

Buckle my shoe.
```

The loop body is iterated zero times.

Display 3.6
A while Loop

do-while Statement

- **Also called *do-while* loop (repeat-until loop)**
- **similar to a *while* statement**
  - except that the loop body is executed **at least once**
- **syntax**

```
do
    body_statement
while (boolean_expression);
```

  - **don't forget the semicolon at the end**

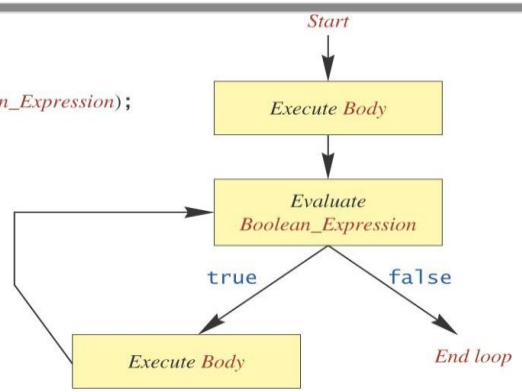
do-while Statement (Contd)

- **First, the loop body is executed**
- **Then the boolean expression is checked**
  - As long as it is true, the loop is executed again
  - If it is false, the loop exits
- **Equivalent *while* statement**

```
body_statement(s)
while (boolean_condition)
        body_statement(s)
```

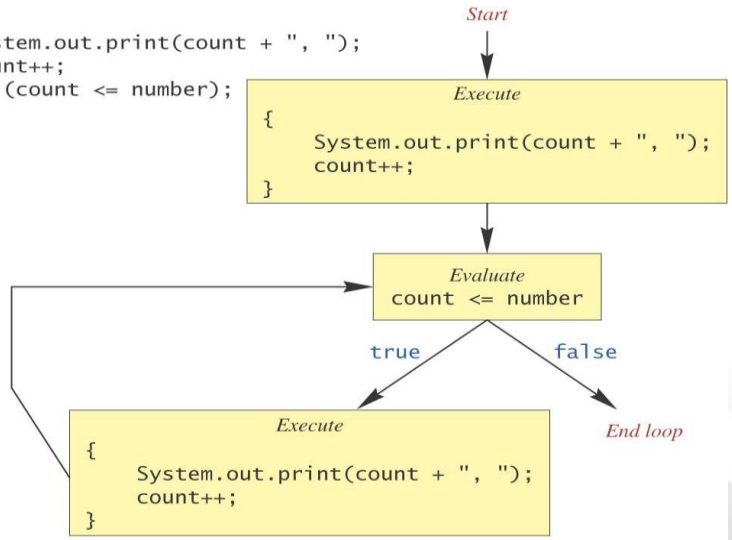
do-while Statement (Contd)

```
do
    Body
while (Boolean_Expression);
```



Example:

```
do
{
    System.out.print(count + ", ");
    count++;
}while (count <= number);
```



Display 3.9

Semantics of the do–while Statement[3]

do-while Statement (Contd)

- ***class DoWhileDemo***

```
import java.util.*;

public class DoWhileDemo
{
    public static void main(String[] args)
    {
        int count, number;

        System.out.println("Enter a number");
        Scanner keyboard = new Scanner(System.in);
        number = keyboard.nextInt();

        count = 1;
        do
        {
            System.out.print(count + ", ");
            count++;
        }while (count <= number);

        System.out.println();
        System.out.println("Buckle my shoe.");
    }
}
```

Sample Screen Dialog 1

```
Enter a number:
2
1, 2,
Buckle my shoe.
```

Sample Screen Dialog 2

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

Sample Screen Dialog 3

```
Enter a number:
0
1,
Buckle my shoe.
```

The loop body is always executed at least one time.

Display 3.8
A do–while Loop

Infinite Loops

- **A loop which repeats without ever ending**
- **The controlling boolean expression (condition to continue)**
  - **never** becomes false

for Statement

- **A *for* statement executes the body of a loop a fixed number of times**

- **example**

```
for (count = 1; count < 3; count++)
        System.out.println(count);
System.out.println("Done");
```

for Statement (Contd)

- **Syntax**

  for (Initialization; Condition; Update)

  body_statement

  body_statement

  - a simple statement or
  - a compound statement in {}

- **Corresponding *while* statement**

  Initialization

  while (Condition)

  body_statement_including_update

for Statement (Contd)

- ***class ForDemo***

```
public class ForDemo
{
    public static void main(String[] args)
    {
        int countDown;

        for (countDown = 3; countDown >= 0; countDown--)
        {
            System.out.println(countDown);
            System.out.println("and counting.");
        }

        System.out.println("Blast off!");
    }
}
```

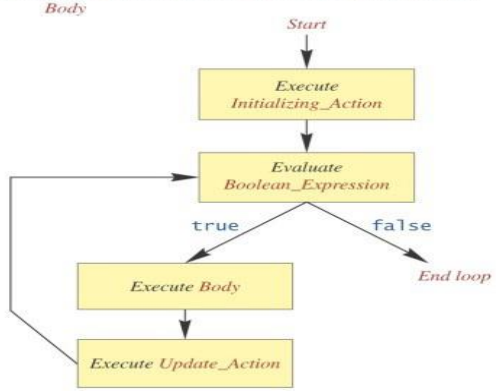
Screen Output

```
3
and counting.
2
and counting.
1
and counting.
0
and counting.
Blast off!
```

Display 3.11
A for Statement

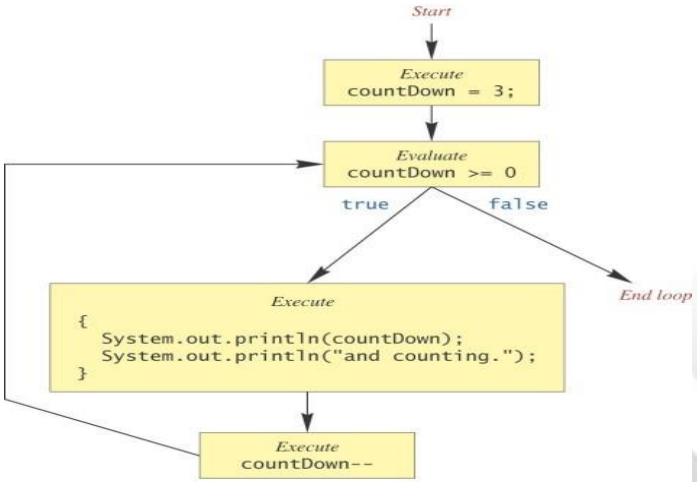
for Statement (Contd)

```
for (Initializing_Action; Boolean_Expression; Update_Action)
    Body
```



Example:

```
for (countDown = 3; countDown >= 0; countDown--)
{
    System.out.println(countDown);
    System.out.println("and counting.");
}
```



Display 3.12
Semantics of the for Statement

Choosing a Loop Statement

- **If you know how many times the loop will be iterated, use a *for* loop**
- **If you don't know how many times the loop will be iterated, but**
  - It could be zero, use a *while* loop
  - It will be at least once, use a *do-while* loop
- **Generally, a *while* loop is a safe choice**

Programming with Loops: Outline

- The Loop Body
- Initializing Statements
- Ending a Loop

Loop Body

- **To design the loop body, write out the actions the code must accomplish**

- **Then look for a repeated pattern**
  - The pattern need not start with the first action
  - The repeated pattern will form the body of the loop
  - Some actions may need to be done after the pattern stops repeating

Initializing Statements

- **Some variables need to have a value before the loop begins**
  - Sometimes this is determined by what is supposed to happen after one loop iteration
  - Often variables have an initial value of zero or one, but not always
- **Other variables get values only while the loop is iterating**

Ending a Loop

- **If the number of iterations is known before the loop starts, the loop is called a *count-controlled loop***
  - Use a *for* loop

- **Asking the user before each iteration if it is time to end the loop is called the *ask-before-iterating technique***
  - Appropriate for a small number of iterations
  - Use a *while* loop or a *do-while* loop

Ending a Loop (Contd)

- **For large input lists, a sentinel value can be used to signal the end of the list**
  - The sentinel value must be different from all the other possible inputs
  - A negative number following a long list of non-negative exam score could be suitable
    - 90
    - 0
    - 10
    - -1 <= exam's score cannot be a negative, so this score is sentinel value

Ending a Loop (Contd)

- **Example – reading a list of scores followed by a sentinel value**

```
int next = keyboard.nextInt();
while (next >= 0)
{
    Process_The_Score
    next = keyboard.nextInt();
}
```

Ending a Loop (Contd)

- *class ExamAverager*

```java
import java.util.*;
/**
 Determines the average of a list of (nonnegative) exam scores.
 Repeats for more exams until the user says she/he is finished.
*/
public class ExamAverager
{
    public static void main(String[] args)
    {
        System.out.println("This program computes the average of");
        System.out.println("a list of (nonnegative) exam scores.");
        double sum;
        int numberOfStudents;
        double next;
        String answer;
        Scanner keyboard = new Scanner(System.in);
        do
        {
            System.out.println();
            System.out.println("Enter all the scores to be averaged.");
            System.out.println("Enter a negative number after");
            System.out.println("you have entered all the scores.");
            sum = 0;
            numberOfStudents = 0;
            next = keyboard.nextDouble();
            while (next >= 0)
            {
                sum = sum + next;
                numberOfStudents++;
                next = keyboard.nextDouble();
            }
            if (numberOfStudents > 0)
                System.out.println("The average is "
                                   + (sum/numberOfStudents));
            else
                System.out.println("No scores to average.");
            System.out.println("Want to average another exam?");
            System.out.println("Enter yes or no.");
            answer = keyboard.next();
        }while (answer.equalsIgnoreCase("yes"));
    }
}
```

Sample Screen Dialog

```
This program computes the average of
a list of (nonnegative) exam scores.

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
100
90
100
90
-1
The average is 95.0
Want to average another exam?
Enter yes or no.
yes

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
90
70
80
-1
The average is 80.0
Want to average another exam?
Enter yes or no.
no
```

Display 3.14
Nested Loops

Nested Loops

- **The body of a loop can contain any kind of statements, including another loop**

- **In the previous example**
  - The average score was computed using a *while* loop
  - This *while* loop was placed inside a *do-while* loop so the process could be repeated for other sets of exam scores.

Nested Loops (Contd)

```
for (line = 0; line < 4; line++)
{
    for (star = 0; star < 5; star++)
        System.out.print('*');
    System.out.println();
}
```

body of outer loop

body of inner loop

- Each time the outer loop body is executed, the inner loop body will execute 5 times
- 20 times total

Output:

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

Programming Example:

```
public class Pyramid {

    public static void main(String args[]) {

        for (int x = 1; x < 10; x++) {

            for (int y = 0; y < x; y++) {
                System.out.print("*");
            }

            System.out.println();
        }

    }

}
```

Output:

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

Self-Test (2)

- **다음과 같은 프로그램을 작성할 것**
  - 2개의 int type 변수 intVal1, intVal2를 선언할 것
  - 두 수를 다음과 같이 곱하여 출력할 것
    - 1 multiplied by 5 = 5
    - 1 multiplied by 4 = 4
    - 1 multiplied by 3 = 3
    - 1 multiplied by 2 = 2
    - 1 multiplied by 1 = 1

    …

    - 5 multiplied by 5 = 25
    - 5 multiplied by 4 = 20
    - 5 multiplied by 3 = 15
    - 5 multiplied by 2 = 10
    - 5 multiplied by 1 = 5
  - 곱셈의 왼쪽 피연산자는 intVal1이며 1부터 5까지의 정수형 값을 가짐
  - 곱셈의 오른쪽 피연산자는 intVal2이며 1부터 5까지의 정수형 값을 가짐