Object Oriented Programming

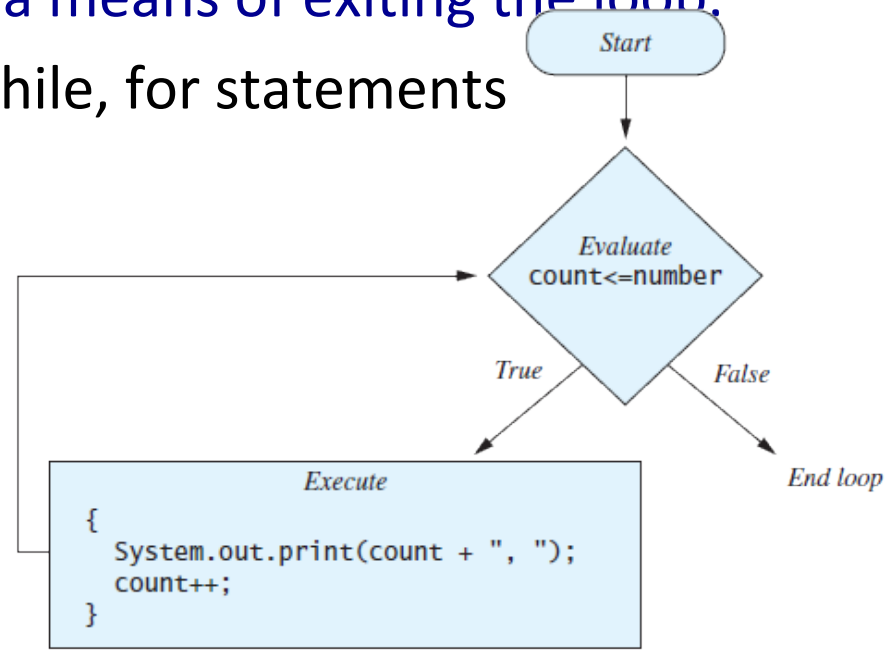
Introduction to Java

Ch. 4. Flow of Control : Loops

School of AI.Software, Gachon University
Ahyoung Choi, Spring

Java Loop Statements

- A portion of a program that repeats a statement or a group of statements is called a ***loop***.
  - E.g. A loop could be used to compute grades for each student in a class.
- There must be a means of exiting the loop.
  - while, do-while, for statements



while Statement

- A `while` statement repeats while a controlling boolean expression remains true
  - Start from expression evaluation
  - As long as it is true, repeat instructions in brackets
- Syntax

  while (*Boolean_expression*) {

  Statements;

  }

- Example

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

while Statement

- You have to do **some initialization before** the statement
- The loop body typically contains an action that ultimately causes the controlling boolean expression to become false.

```
number = keyboard.nextInt();
count = 1;
while (count <= number) {
  System.out.println(count);
  count++;
}
```

Lab: `while` Statement

- View sample program, Listing 4.1
  `class WhileDemo`
  - Write a program to count the number to buckle the shoe.
  - Write a program to count the odd number to buckle the shoe.

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

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

```
Enter a number:
0

Buckle my shoe.
```

The loop body is iterated zero times.

LISTING 4.1 A while Loop

```
import java.util.Scanner;
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.");
    }
}
```

do-while Statement

- Similar to a **`while`** statement, except that the loop body is executed at least once
- Syntax

  do {

  Statements;

  } while (*Boolean_Expression*);

- Don't forget the semicolon!

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

`do-while` Statement

- 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 is exited.
- Equivalent `while` statement

```
Statement(s)_S1
while (Boolean_Condition)
    Statement(s)_S1
```

Difference?

- do-while statement will execute the body statements at least once
- while statement may output nothing
  - Consider: number == 0

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

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

Lab: `do-while` Statement

- View sample program, listing 4.2
  `class DoWhileDemo`

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

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

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

The loop body always executes at least once.

LISTING 4.2 A do-while Loop

```
import java.util.Scanner;
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.");
    }
}
```

Infinite loops

- A loop which repeats without ever ending
  - If the controlling Boolean expression never becomes *false*, a while/do-while loop will repeat without ending
- Always make sure that your loop will end
  - Never forget to change the count

```
int count = 0;
while (count < 5)
{
   System.out.println(count);

}
System.out.println("count after loop = " + count);
```

Infinite Loops

- Infinite loop is not a syntax error. It's a logical error
- eclipse will not help you in this case
- Write pseudo code, think, and rethink before coding

Nested loops

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

Nested Loops

- Computes the average of a list of (**nonnegative) exam scores**. Repeats computation for more exams until the **user says to stop**
- View listing 4.4 `class ExamAverager`

```
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
```

```java
import java.util.Scanner;
/**
Computes the average of a list of (nonnegative) exam scores.
Repeats computation for more exams until the user says to stop.
*/
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"));
```

for Statement

- `for` statement (or usually called *`for` loop*)
  - Used to executes the body of a loop a **fixed number** of times
- While vs for

```
number = keyboard.nextInt();
count = 1;
while (count <= number) {
  // all the actions
  count++;
}
```

```
number = keyboard.nextInt();
int count;
for (count = 1;
   count<=number; count++) {
  // all the actions
}
```

for Statement

- Syntax:

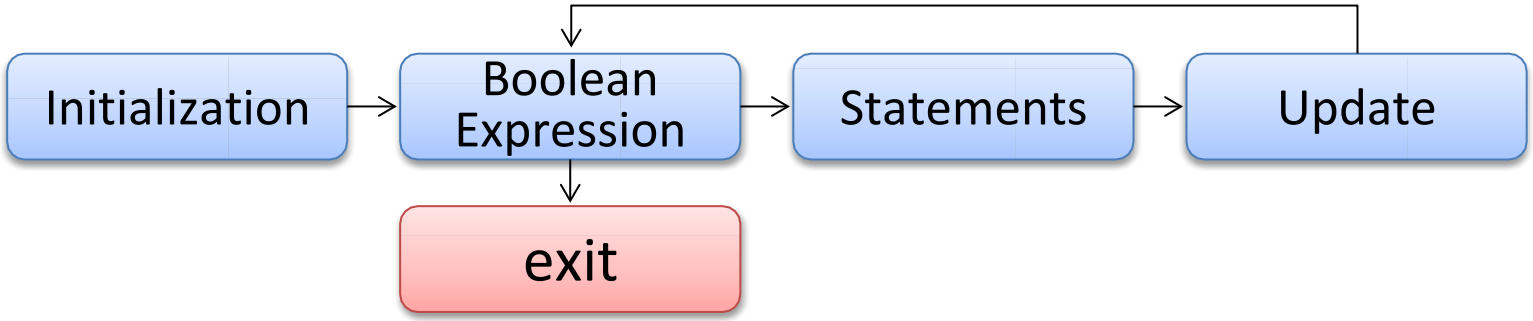
  for (*Initialization*; *Boolean_Expression*; *Update*) {

  Statements;

  }

- Example

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



Local variables

- The counter can be defined in the for loop
  - Only available in this loop
  - Undefined outside of the loop
- It is really a bad idea to name the count variable the same as the one outside of the loop
- Local variables can also be defined in if and while statements

```
for (int i = 0; i < 100; i++) {
    // all the actions
}
System.out.println(i);
// WRONG! i is not visible outside of for loop
```

```
if (true) {
     int temp = 0;
}

do {
     int temp2 = 2;
} while (false);
```

For Loop: Don't Overcount

- Repeat 3 times

```
for (int count = 1; count <= 3; count++) {
  // all the actions
}
```

- Repeat 3 times

```
for (int count = 0; count < 3; count++) {
  // all the actions
}
```

- Repeat **4 times**!

```
for (int count = 0; count <= 3; count++) {
  // all the actions
}
```

Using a comma (,)

- Multiple initializations and updates can be performed in a for statement
- Actions are simply separated with commas
- Only one boolean expression is allowed, but it can consist of `&&`s, `||`s, and `!`s.

```
for (n = 1, product = 1; n <= 10; n++)
    product = product * n;
```

```
for (n = 1, product = 1; n <= 10; product = product * n, n++);
```

For Loop: Case Study

- Let the user input 10 numbers, then output the sum of those numbers
- What's wrong with this piece of code?

```
import java.util.Scanner;
public class input {
  public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    for (int i = 1; i <= 10; i++) {
      int sum = 0;
      System.out.println("Please enter a new number (" + i + " of 10):");
      int input = keyboard.nextInt();
      sum += input;
    }
    System.out.println("Total sum is: " + sum);
  }
}
```

For Loop: Case Study

- Let the user input 10 numbers, then output the product of those numbers
- What's wrong with this piece of code?

```
import java.util.Scanner;
public class input {
  public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    int product = 0;
    for (int i = 1; i <= 10; i++) {
      System.out.println("Please enter a new number (" + i + " of 10):");
      int input = keyboard.nextInt();
      product *= input;
    }
    System.out.println("Total product is: " + product);
  }
}
```

break and continue Statements

- break statement
  - A break statement can be used to end a loop immediately
  - The break statement ends only the *innermost* loop or switch statement that contains the break statement
  - break statements may make loops more difficult to understand

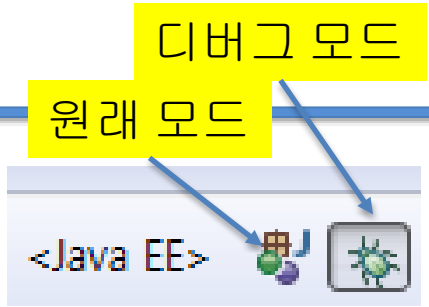
```
count = 10;
for (int i = 0; i < count; i ++) {
    if (i == 3) break;
    System.out.println(i);
}
System.out.println("Loop complete");
```

break and continue Statements

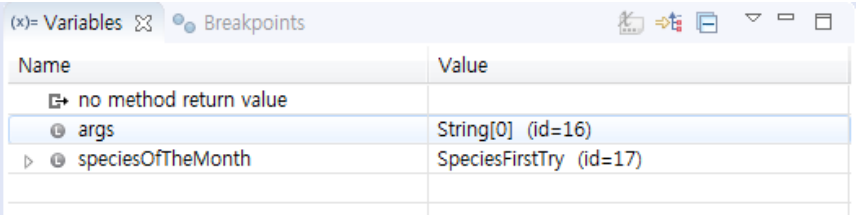
- continue statement
  - **Ends current loop iteration**
  - Begins the next iteration
  - May introduce some complications

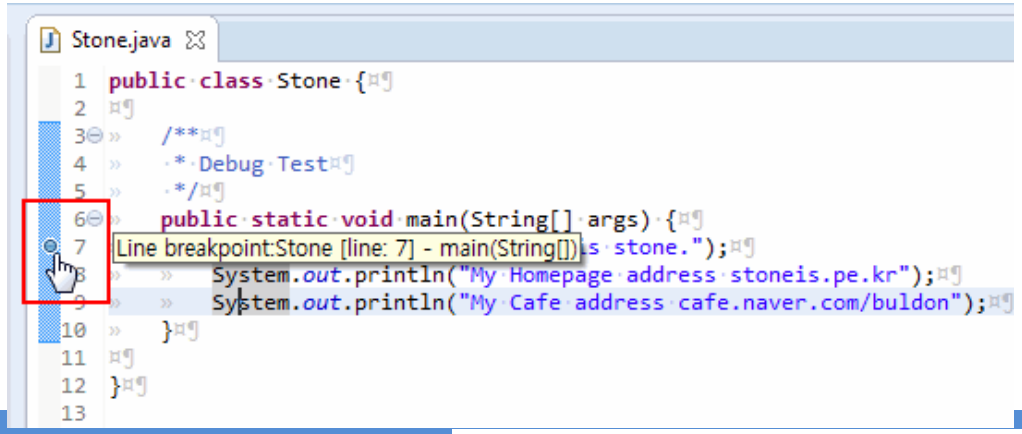
```
count = 10;
for (int i = 0; i < count; i ++) {
    if (i == 3) continue;
    System.out.println(i);
}
System.out.println("Loop complete");
```

Debugging : finding errors



- Tracing variables
  - Watching the variables change while the program is running
  - Simply insert temporary output statements in your program to print the values of variables of interest or, learn to use the debugger that may be provided by your system
  - Setting the breakpoint: click twice on the left side line





File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access <Java EE>

Debug

- Thread [main] (Suspended (breakpoint at line 9 in SpeciesFirstTryDemo))
  - SpeciesFirstTryDemo.main(String[]) line: 9
- C:₩Program Files₩Java₩jre1.8.0_191₩bin₩javaw.exe (2019. 3. 20. 오후 3:15:19)
- input [Java Application]
  - C:₩Program Files₩Java₩jre1.8.0_191₩bin₩javaw.exe (2019. 3. 20. 오후 3:24:03)
- <terminated>input [Java Application]
  - <disconnected>multiplyTest.input at localhost:53322
  - <terminated, exit value: 0>C:₩Program Files₩Java₩jre1.8.0_191₩bin₩javaw.exe (2019. 3. 20. 오후 3:26:11)

(x)= Variables Breakpoints

RectangleDem... SpeciesFirs... enumtest.java Time.java input.java »94

```
1  package multiplyTest;
2  import java.util.Scanner;
3
4  public class input {
5      public static void main(String[] args) {
6              Scanner keyboard = new Scanner(System.in);
7              int product = 1;
8              for (int i = 1; i <= 5; i++) {
9                System.out.println("Please enter a new number (" + i + " of 5):");
10               int input = keyboard.nextInt();
11               product *= input;
12             }
13             System.out.println("Total product is: " + product);
14             keyboard.close();
15     }
16 }
17
```

Outline

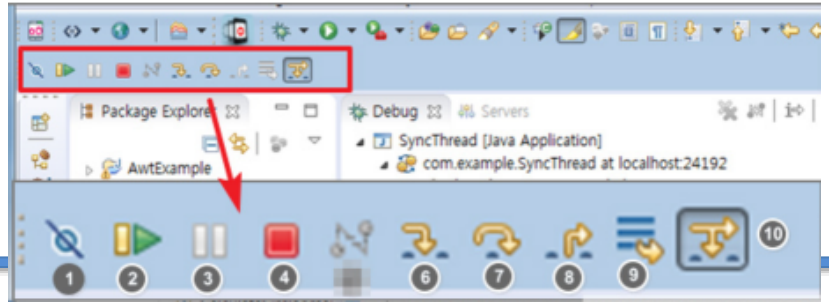
- multiplyTest
- input
  - main(String[]) : void

Console Tasks

<terminated> input [Java Application] C:₩Program Files₩Java₩jre1.8.0_191₩bin₩javaw.exe (2019. 3. 20. 오후 3:26:11)

```
Please enter a new number (1 of 5):
2
Please enter a new number (2 of 5):
3
Please enter a new number (3 of 5):
4
Please enter a new number (4 of 5):
10
Please enter a new number (5 of 5):
5
Total product is: 1200
```

Debugging



- **1) Skip All Breakpoints** 모든 Break Point를 무시
- **2) Resume(F8) :** Continue to the next breakpoint 다음브레이크까지 실행
- **3) Suspend :** Pauses the thread and is the same as that specified for the current statement.
- **4) Terminate :** End thread
- **6) Step Into(F5) :** If the next line is in a function, it goes into the function. 함수 안으로 들어감
- **7) Step Over(F6) :** Pass a function call and go one step at the current position 한 스텝씩 진행, 현재 소스창에서 한 스텝씩 진행
- **8) Step Return(F7) :** Go back to the end of the current function, return to the function call 함수 밖으로 빠져 나옴
- **9) Drop to Frame :** Moves to the first row of the selected stack frame. When you want to start from scratch
- **10) Use Step Filters(Shift+F5)**

Debugging: Assertion check

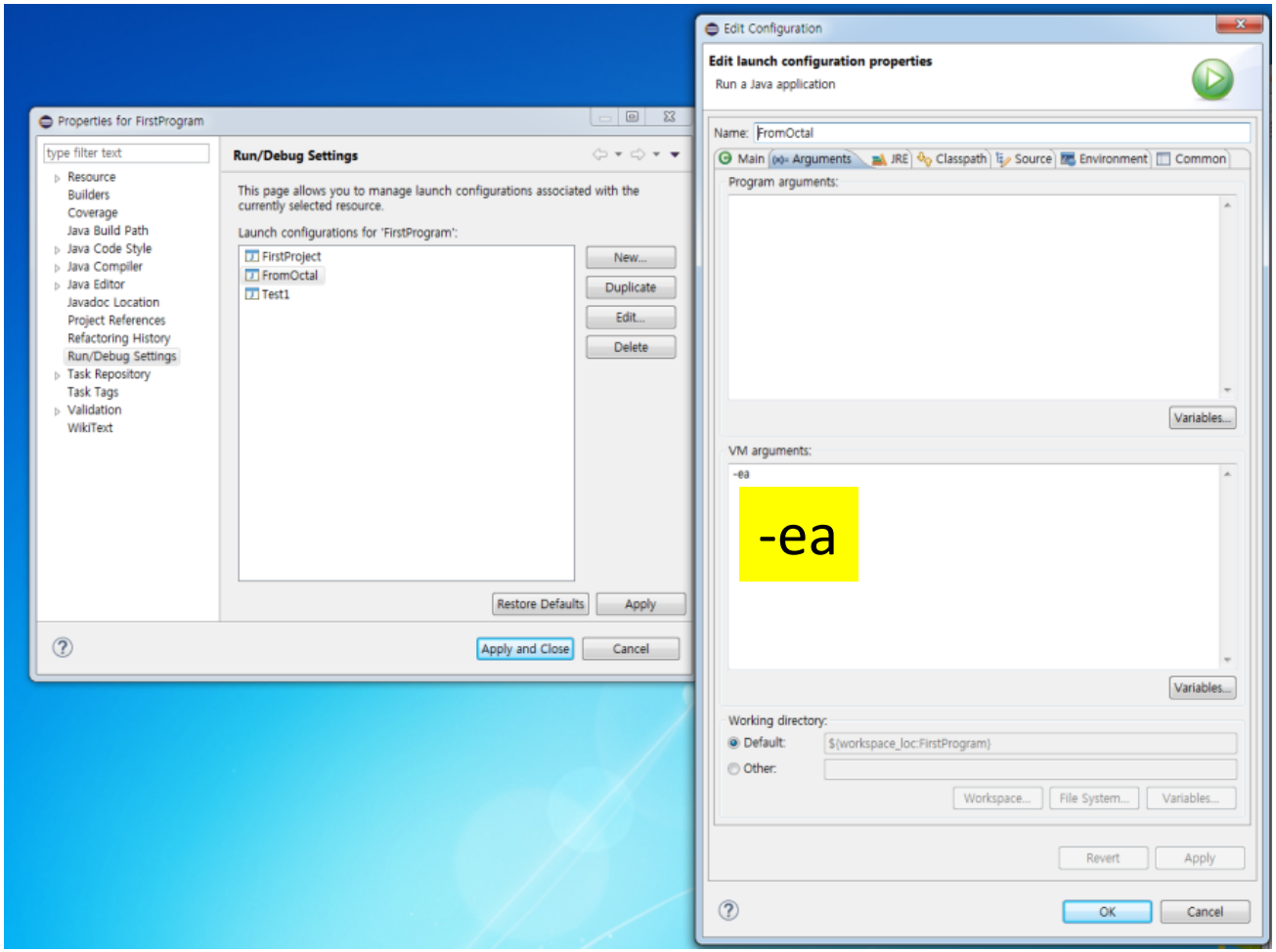
- Assertion
  - Something that says the current state of the program
  - Can be true or false
  - Should be true when no mistakes in running program
- Syntax for assertion check

  Assert Boolean_Expression;
- Assertion check

```
assert n == 1;
while (n < limit)
{
    n = 2 * n;
}
assert n >= limit;
//n is the smallest power of 2 >= limit.
```

Debugging : Assertion check

- Assertion on
- Right click on project →Run/Debug Settings → Choose resources → click “edit” → Set VM argument “–ea”



```
<terminated> movieRating [Java Application] C:₩Program Files₩Java₩jre1.8.0_191₩bin₩javaw.exe (2019.
Exception in thread "main" java.lang.AssertionError
        at movieRating.main(movieRating.java:12)
```

Cipher.java | CipherInterf... | movieRating.... | »98

```
1
2  public class movieRating {
3
       2 references | 1 reference | 1 reference | 2 references
4      enum Rating {a, b, c}
5
6      public static void main(String[] args) {
7
8          Rating r;
9          String a = "Hello";
10         r= Rating.a;
11
12         assert a == "a";
13
14         switch(r) {
15         case a:
16             System.out.println("aaaaaaaaa");
17             break;
```

Practice 4

- Ex4_1a. Write a following program
  - Read a list of non-negative integers: the end is indicated by any negative value (not included)
  - Print the max, min, average (double) of the integers
- Ex4_1b. Extend Ex4_1b
  - Read integer percentage (0 ~ 100): ignore values > 100
  - Set grade for each value: A (90~100), B (80~89), C (70~79), D (60~69), F (0~59)
  - Print total count of grades and the count for each grade
  - E.g., 98 87 86 78 -1 → total count: 4
    A count: 1, B count: 2, C count: 1, D count: 0, F count: 0