



УПРАВЛІННЯ ХОДОМ ВИКОНАННЯ JAVA-ПРОГРАМИ

Питання 1.4.

Оператори МП (statements, інструкції)

- Оператори мови програмування – «робочі лошадки».
 - Присвоюють значення, контролюють хід виконання програми (виконують вибір або повторно виконують інші оператори МП, та ін.)
- Оператор МП може бути простим та складеним:
 - *Простий оператор* – одна окрема інструкція вихідного коду для виконання певного завдання; закінчується ‘;’
 - *Складений оператор* – послідовність (можливо, порожня) простих та інших складених операторів МП, що знаходяться між роздільниками (delimiters) {...}.
 - Приклад – тіло методу.
 - Ще складені оператори називають блоками

Оператор присвоєння (Assignment Statements)

- Присвоює значення змінній.
- Розрізняють оператор присвоєння (=) та складений оператор присвоєння (наприклад, +=)

```
x = 10;  
ages[0] = 25;  
counter += 10;
```

- Ініціалізація змінної теж може розглядатись як особлива форма оператору присвоєння

```
int counter = 1;
```

Умовні оператори (Decision Statements)

- Вже згадували про оператор (?:)
- Виділяють конструкції if, if-else та switch.

- Оператор if:

```
if (Boolean expression)
    statement
```

```
if (numMonthlySales > 100)
    wage += bonus;
```

```
if (numMonthlySales > 100) {
    wage += bonus;
}
```

- Оператор if-else:

```
if ((n & 1) == 1)
    System.out.println("odd");
else
    System.out.println("even");
```

```
if (Boolean expression)
    statement1
else
    statement2
```

- Передбачається існування цілочисельної змінної n.
- Через пріоритет операцій потрібні дужки (== має вищий пріоритет за &)
- Запис за допомогою тернарного оператора

```
System.out.println((n & 1) == 1 ? "odd" : "even");
```

Множинный if-else

```
if (Boolean expression1)
    statement1
else
    if (Boolean expression2)
        statement2
    else
        ...
    else
        statementN
```

```
Your test result suggests that you need a tutor.
Your grade is D.
```

14.02.2020 10:57

```
public class GradeLetters
{
    public static void main(String[] args)
    {
        int testMark = 69;
        char gradeLetter;

        if (testMark >= 90)
        {
            gradeLetter = 'A';
            System.out.println("You aced the test.");
        }
        else
            if (testMark >= 80)
            {
                gradeLetter = 'B';
                System.out.println("You did very well on this test.");
            }
            else
                if (testMark >= 70)
                {
                    gradeLetter = 'C';
                    System.out.println("You'll need to study more for future tests.");
                }
                else
                    if (testMark >= 60)
                    {
                        gradeLetter = 'D';
                        System.out.println("Your test result suggests that you need a tutor.");
                    }
                    else
                    {
                        gradeLetter = 'F';
                        System.out.println("Your fail and need to attend summer school.");
                    }

                System.out.println("Your grade is " + gradeLetter + ".");
            }
    }
}
```

DANGLING-ELSE PROBLEM

- Одночасне використання if та if-else може призвести до непорозумінь

```
if (car.door.isOpen())  
    if (car.key.isPresent())  
        car.start();  
else car.door.open();
```

```
if (car.door.isOpen())  
    if (car.key.isPresent())  
        car.start();  
else  
    car.door.open();
```

```
if (car.door.isOpen())  
{  
    if (car.key.isPresent())  
        car.start();  
}  
else  
    car.door.open();
```

Забування того, що оператор else відноситься до найближчого if при погано форматovanому коді відоме як dangling-else problem.

Оператор Switch

- Дозволяє більш зручно обирати з декількох шляхів виконання (execution paths)

```
switch (selector expression)
{
    case value1: statement1 [break;]
    case value2: statement2 [break;]
    ...
    case valueN: statementN [break;]
    [default: statement]
}
```

```
switch (direction)
{
    case 0: System.out.println("You are travelling north."); break;
    case 1: System.out.println("You are travelling east."); break;
    case 2: System.out.println("You are travelling south."); break;
    case 3: System.out.println("You are travelling west."); break;
    default: System.out.println("You are lost.");
}
```

- direction – ціле число.

Оператори циклу (Loop Statements)

- Виділяють три види: for, while, do-while

```
for ([initialize]; [test]; [update])  
    statement
```

```
public static void main(String[] args)  
{  
    for (int i = 0; i < args.length; i++)  
        System.out.println(args[i]);  
}
```

- Секції опційні.
- Цикл for ідеальний для ітерування (*iterating*, *looping* – проходження по) масиву

Прохід по 2D-масиву

```
public class DumpMatrix
{
    public static void main(String[] args)
    {
        float[][] matrix = { { 1.0F, 2.0F, 3.0F }, { 4.0F, 5.0F, 6.0F } };
        for (int row = 0; row < matrix.length; row++)
        {
            for (int col = 0; col < matrix[row].length; col++)
                System.out.print(matrix[row][col] + " ");
            System.out.print("\n");
        }
    }
}
```

- Вираз `matrix.length` повертає кількість рядків у даному табульованому масиві.
 - Для кожного рядка вираз `matrix[row].length` повертає кількість стовпців у ньому.

```
1.0 2.0 3.0
4.0 5.0 6.0
```

Цикл while

```
while (Boolean expression)  
    statement
```

```
int ch = 0;  
while (ch != 'C' && ch != 'c')  
{  
    System.out.println("Press C or c to continue.");  
    ch = System.in.read();  
}
```

```
for (int i = 0; i < 10; i++)  
    System.out.println(i);
```

=

```
int i = 0;  
while (i < 10)  
{  
    System.out.println(i);  
    i++;  
}
```

Оператор циклу do-while

```
do
    statement
while (Boolean expression);
```

- Спочатку виконує *statement* і лише потім перевіряє умову
- Аналогічний приклад: потреби ініціалізації *ch* немає

```
int ch;
do
{
    System.out.println("Press C or c to continue.");
    ch = System.in.read();
}
while (ch != 'C' && ch != 'c');
```

Порожні цикли

```
for (String line; (line = readLine()) != null; System.out.println(line));
```

- Тут `for` представляє програмну ідіому для копіювання рядків тексту, зчитаних з певного джерела за допомогою фіктивного методу `readLine()` з подальшим виводом за допомогою `System.out.println()`.
 - Копіювання продовжується, поки `readLine()` не поверне `null`.
 - Зверніть увагу на порожній оператор МП «;» в кінці рядка.
 - Будьте уважні з ;

```
for (int i = 0; i < 10; i++); // this ; represents the empty statement  
System.out.println("Hello");
```

Оператори переходу break, continue, return

- Що спільного між операторами
 - for (;;) ;
 - while (true) ;
 - do;while (true) ; ?

Вони представляють нескінченний цикл, з якого неможливо вийти

Оператори переходу break, continue, return

- Проте такі цикли можуть бути корисними.
 - Можна використовувати while (true), який буде весь час просити ввести якийсь символ, доки не буде введено коректний. Тоді цикл має закінчитись.
 - Для такої задачі використовують оператор break.

```
int ch;
while (true)
{
    System.out.println("Press C or c to continue.");
    ch = System.in.read();
    if (ch == 'C' || ch == 'c')
        break;
}
```

Оператор break

- Корисний і в скінченних циклах.
- Задача: пошук в масиві заданого значення та вихід з циклу при його знаходженні.

```
public class EmployeeSearch
{
    public static void main(String[] args)
    {
        int[] employeeIDs = { 123, 854, 567, 912, 224 };
        int employeeSearchID = 912;
        boolean found = false;
        for (int i = 0; i < employeeIDs.length; i++)
            if (employeeSearchID == employeeIDs[i])
            {
                found = true;
                break;
            }
        System.out.println((found) ? "employee " + employeeSearchID + " exists"
                                : "no employee ID matches " + employeeSearchID);
    }
}
```

Вивід

```
employee 912 exists
```

Break як заміна goto

- Використовується оператор break з міткою (labeled break statement).
- Корисний для вкладених циклів

```
outer:
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        if (i == 1 && j == 1)
            break outer;
        else
            System.out.println("i=" + i + ", j=" + j);
System.out.println("Both loops terminated.");
```

```
i=0, j=0
i=0, j=1
i=0, j=2
i=1, j=0
Both loops terminated.
```


Оператори Continue та Labeled Continue

- Оператор continue пропускає решту ітерації циклу та переходить до нової (якщо умова циклу – true).
- Приклад: хочемо якимось чином обробляти текст, проте пропускати порожні рядки.

```
String line;  
while ((line = readLine()) != null)  
{  
    if (isBlank(line))  
        continue;  
    processLine(line);  
}
```

- Метод isBlank() фіктивний: повертає true, якщо зчитує порожній рядок.
- Можна зробити рефакторинг!

Оператор continue

- Скоротимо код

```
String line;
while ((line = readLine()) != null)
{
    if (!isBlank(line))
        processLine(line);
}
```

- Існує також оператор continue з міткою (*labeled continue statement*)
 - Корисний для переривання вкладених циклів, при цьому все ще продовжуючи виконання циклу з міткою.

```
outer:
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        if (i == 1 && j == 1)
            continue outer;
        else
            System.out.println("i=" + i + ", j=" + j);
System.out.println("Both loops terminated.");
```

```
i=0, j=0
i=0, j=1
i=0, j=2
i=1, j=0
i=2, j=0
i=2, j=1
i=2, j=2
Both loops terminated.
```

Оператор return

- Оператор переходу, котрий здійснює вихід із поточного методу і повертає потік управління до точки програми, з якої було викликано даний метод.
 - Має дві форми: одна повертає значення, інша – ні.
- Для того, щоб повернути значення, достатньо записати його (або вираз, що обчислює це значення) після ключового слова `return`:
 - `return i;`
 - `return a+b;`
- Тип даних, що повертає оператор, повинен співпадати із оголошеним типом методу, якого стосується даний оператор `return`.
- Якщо метод оголошено як `void`, використовують форму оператора `return`, котра не повертає жодного значення:
 - `return;`



ДЯКУЮ ЗА УВАГУ!

Наступне запитання: управління ходом виконання Java-програми