Програмування-1

Лекція 3

Agenda

- Структурне програмування
- Поняття про блоки
- if, if - else, if - else if – else if ...
- ?:
- switch
- while, do - while, for
- break, continue, return

Структурне програмування (СП)

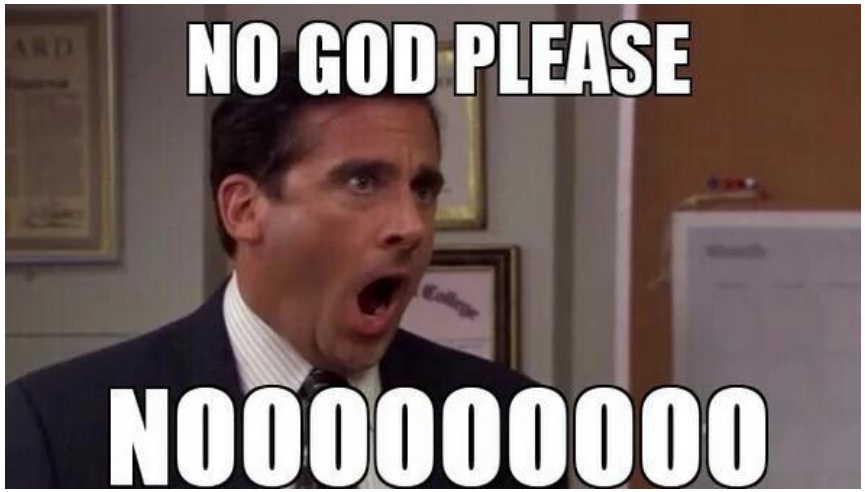
- **Структурне програмування** - методологія розробки програмного забезпечення на основі представлення програми у вигляді ієрархічної структури блоків
- **Основні конструкції**
  - послідовне виконання
  - розгалуження
  - цикл

Принципи СП
NO GOTO!

- No “**GOTO**” !!!

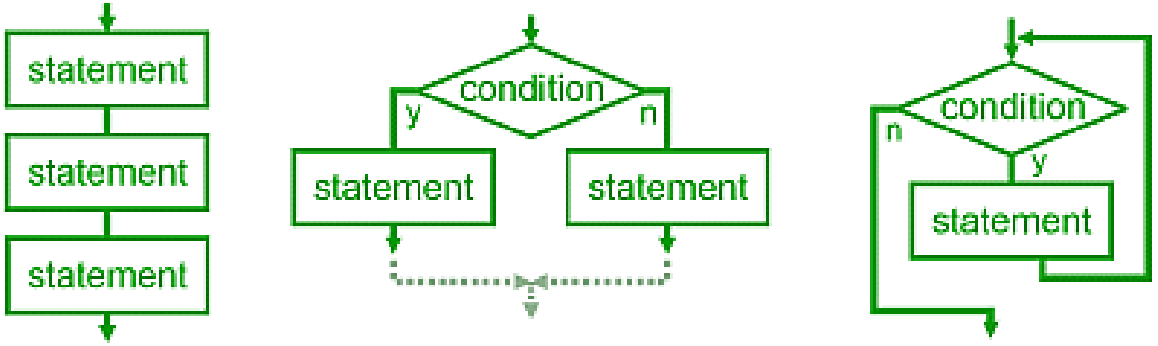
- GOTO в програмі = спагеті-код

```
10 i = 10
20 Print "Hello World"
30 i = i+1
40 IF i<20 THEN GOTO 20 ENDIF
50 PRINT "DONE"
...
...
100 GOTO 30
```



Принципи СП
sequence-selection-repetition

- Базові конструкції СП:
  - послідовне виконання
  - розгалуження
  - цикл



Принципи СП
nesting

- Базові конструкції можуть бути вкладені одна в одну

Принципи СП
DRY

- DRY = Don’t repeat yourself
- Повторювані фрагменти програми можна оформити у вигляді підпрограм (процедури, функції, методи)

- Ctrl-C + Ctrl-V = погано, не робіть так! ☺

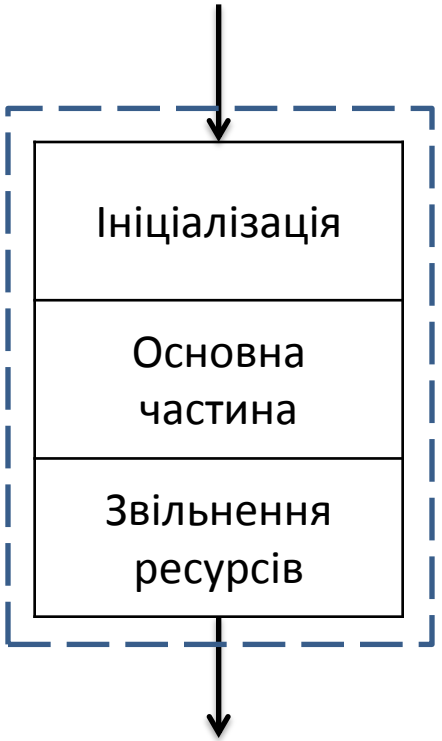
Принципи СП
Блок

- Блок - це логічно згрупована частина початкового коду
- До блоку можна звертатись як до окремої інструкції
- Блоки обмежують область видимості ідентифікаторів
- Блоки можуть бути пустими
- Блоки можна вкладати один в інший

Принципи СП

Один вхід, один вихід

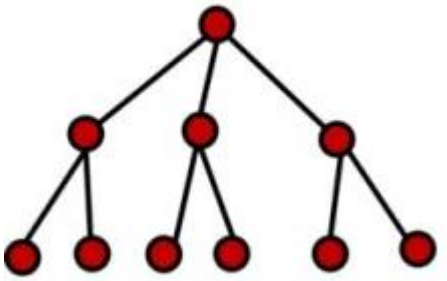
- В Java завжди є лише один вхід у блок
  - no “GOTO” in Java
- Може бути декілька виходів з блоку, якщо це підвищую читабельність коду
  - break, return, throw
- Якщо потрібно вивільняти ресурси, рекомендується використовувати try-finally або try-with-resources



Принципи СП

Проектування згори-вниз

- Від загального до деталей (декомпозиція)
- Не потрібно відразу писати код для деталей, доки не досягнутий відповідний рівень конкретизації



Управління потоком виконання

Блоки

- Блок в Java позначається { }
- Область видимості змінних
  - з середини блоку можна «побачити» те, що знаходиться навколо нього
  - ззовні неможливо побачити те, що знаходиться всередині блоку

- Перевизначення змінних
  - всередині блоку можна перевизначити змінну рівня класу або об’єкту
  - всередині блоку неможливо перевизначити локальну змінну

Управління потоком виконання

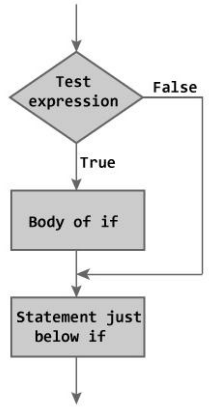
Блоки

```
public class BlockDemo{

        static int a=0; // Змінна на рівні класу

        public static void main(String[] args) {
                int a=1; // Локальна змінна. Перекриває зовнішню
                // Зовнішня змінна доступна як BlockDemo.a
                {
                        a=2; // Ok
                        int a=3; // Error: already defined
                        int b=4; // Ok
                }
                b=5; // Error: cannot find
                int b=6; // Ok
        }
}
```

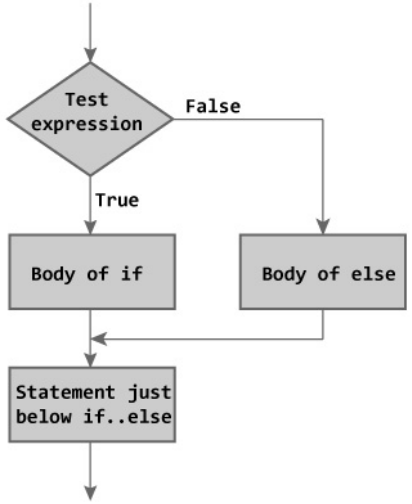
Управління потоком виконання

if

```
if ( boolean_expression ) {
   ...
}
```




```
if ( boolean_expression ) {
   ...
} else {
   ...
}
```



Управління потоком виконання

if

```
if (d<0) {
      System.out.println("Немає коренів!");
      return;
}
```

```
if (d<0) {
      System.out.println("Немає коренів!");
      return;
} else {
      // x1 = ....
      // x2 = ....
}
```

Управління потоком виконання

if

- Можна (але не варто) не використовувати {}, якщо блок складається лише з однієї команди

```
hours++;
if (hours == 24)
    hours=0;
```

- Краще завжди використовувати {}

```
hours++;
if (hours == 24) {
    hours=0;
}
```

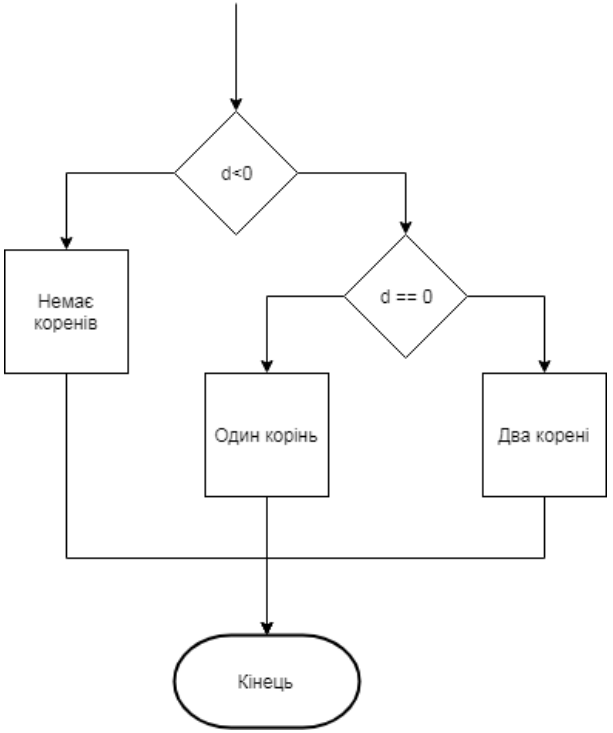
- Виняток: ланцюжки **if - else if** (дивись наступний слайд)

Управління потоком виконання

if

- Ланцюжок if – else if – else if – … – else

```
if (d < 0) {
    System.out.println("Немає коренів!")
    // ...
} else if (d == 0) {
    System.out.println("Один корень");
    // x = ...
} else {
    System.out.println("Два корені");
    // x1 = ...
    // x2 = ...
}
```



Управління потоком виконання оператор ?:

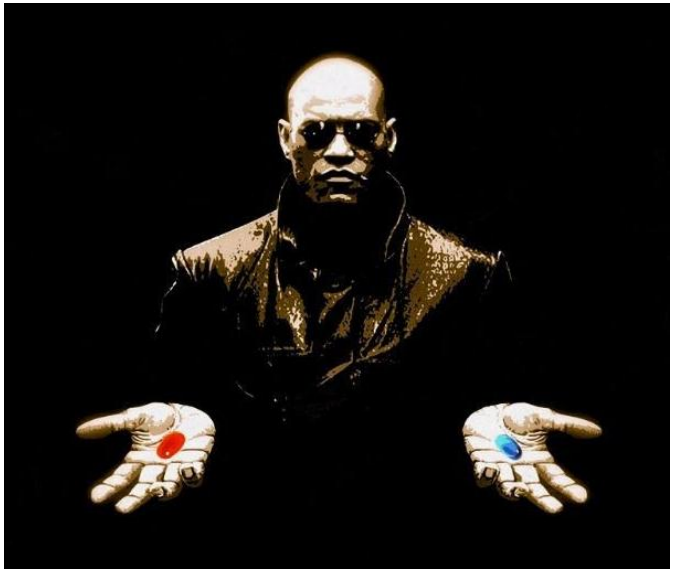
- (*умова*) **?** (*результат_якщо_true*) **:** (*результат_якщо_false*)

- Приклад: замість

```
if (a<b) {
   min = a;
} else {
   min = b;
}
```

можна написати

```
min = (a<b) ? a : b;
```



Управління потоком виконання **switch**

```
switch (expression) {
    case constant1:
       ...
       break;
    case constant2:
       ...
       break;

     ...

    default:
       ...
       break;
}
```

- В залежності від значення **expression** управління передається на відповідний **case**
- **expression/constant** можуть бути **byte**, **short**, **int**, **char**
  - починаючи з Java 5 також **enum**
  - починаючи з Java 7 також **String**
- Розділ **default** – необов'язковий
- Не забувайте про **break** **!**

Управління потоком виконання
switch

Приклад

```
switch (colorNum) {
        case 0:
                setBackground(Color.RED);
                break;
        case 1:
                setBackground(Color.GREEN);
                break;
        default:
                setBackground(Color.BLACK);
                break;
}
```

Управління потоком виконання

for

```
for (ініціалізація; умова; інкремент) {
   ...
}
```

- Кожен з виразів «*ініціалізація*», «*умова*», «*інкремент*» може бути відсутнім
- **for(;;) {}** – «вічний» цикл
- якщо в «*ініціалізації*» задекларована змінна, її область видимості: від **for** до **}**
  - іншими словами, після циклу її нема

- «*умова*» перевіряється **перед** виконанням кожної ітерації
  - тобто може бути, що тіло циклу не буде виконано жодного разу
- «*інкремент*» виконується **після** виконання ітерації
- в «*інкременті*» може бути записано кілька виразів через кому (наприклад: «i++, j++»)

Управління потоком виконання

for

Приклад:

```
for (int i = 0; i < 10; i++) {
      System.out.println("Hello, World!");
}
```

```
for (int i=0, j=10; i<=10; i++, j--){
      System.out.println(i + " " + j);
}
```

Управління потоком виконання

while

```
while (умова) {
    ...
}
```

- «*умова*» перевіряється **перед** виконанням кожної ітерації
  - тобто може бути, що тіло циклу не буде виконано жодного разу
- **while** (true) { } – «вічний» цикл

Управління потоком виконання
while

Приклад:

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

Управління потоком виконання

do-while

```
do {
      ...
} while (умова);
```

- «*умова*» перевіряється **після** виконання кожної ітерації
  - тобто, тіло циклу буде виконано щонайменше один раз
- **do {} while (true);** – «вічний» цикл
- якщо **«умова» == true**, то виконується наступна ітерація
  - Не плутати з repeat-until у Pascal, де «*умова*» == true призводить до виходу з циклу
- не забувайте про «**;**» у кінці

Управління потоком виконання
do-while

Приклад

```
int i=0;
do {
    System.out.println("Hello");
    i++;
} while (i > 10000);
```

Управління потоком виконання
break

- використовується для передчасного виходу із циклів
- разом з мітками дозволяє вийти із вкладених циклів
- разом з мітками дозволяє вийти із будь-якого блоку (майже як “goto”)

```
do {
      ...
      if (умова_передчасного_виходу)
           break;
      ...
} while (умова_продовження_циклу);
```

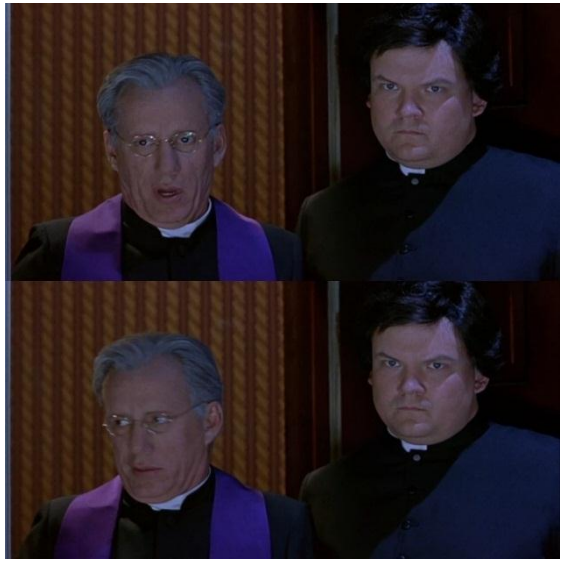
Управління потоком виконання
break

Приклад:

```
int i=0;
while (true) { // Вічный цикл
   System.out.println("Hello, World!");
   i++;
   if (i=13)
      break; // Передчасний вихід
}
```

Приклад 2 (не робіть так):

```
label:{
   System.out.println(1);
   if (omg) {
      break label;
   }
   System.out.println(2);
}
```



Управління потоком виконання **continue**

- Використовується для передчасного переходу на наступну ітерацію циклу

- Пример:

```
// розрахунок суми перших 20 чисел, крім 13
      int s=0;
      for (int i=1; i<=20; i++) {
            if (i == 13)
                  continue;
            s = s + i;
      }
```

Управління потоком виконання

мітки

- На відміну від інших мов програмування у Java міткою помічається не рядок, на який буде здійснено перехід, а блок, з якого буде здійснено вихід
- **break** та **continue** впливають на поточний цикл. Якщо потрібно вийти з вкладеного циклу, використовуйте мітки:

```
outer: do {    // outer loop
   …
   do {        // inner loop
    …
    if (умова)
       break outer;
   } while (умова);
   …
} while (умова);
```