Pętle

Spis treści

Zapiski uczestnika Bootcampu Zajavka.pl w 12 tygodni by Bartek Borowczyk aka Samuraj Programowania. Dopiski na zielono od Karola Rogowskiego.

Rodzaje pętli

W przypadku instrukcji `if` i `switch` mówimy o instrukcji **sterującej**[1], a konkretnie instrukcji warunkowej, czyli zrób to i to jeśli jest taki i taki warunek spełniony. W przypadku pętli (`for`, `while` i `do-while`) również mówimy o instrukcji sterującej, ale już nie o warunkowej a **iteracyjnej**[2]. Pętle bowiem pozwalają wykonać dany kod wielokrotnie.

[1] - instrukcje sterujące kierują przepływem programu.

[2] - iterować znaczy powtarzać. W przypadku programowania i pętli powtarzanie oznacza wielokrotne wykonywanie tych samych instrukcji.

Pętla while

Pętla `while` pozwala wykonywać coś (iterować), dopóki spełniony jest warunek. Może się zdarzyć, oczywiście, że dana pętla nie wykona się ani razu (i to jest ok, jeśli właśnie taka jest logika programu) lub coś może wykonywać się nieskończoną ilość razy (i to zazwyczaj oznacza błąd, ale nie zawsze, bo np. czasami pętla trwa w nieskończoność, jeśli np. zakłada próbę pobrania jakichś danych i będzie się wykonywała tak długo, aż te dane pobierze np. user wprowadzi prawidłowe hasło).

Zobaczmy najpierw schemat, a potem przykład, by stało się jasne, o co chodzi:

```
while (warunek) {
    // ciało pętli (instrukcje do wykonania)
}
```

Instrukcja `while` wykona się tylko wtedy, jeśli warunek będzie prawdziwy (zwróci `true`) i będzie

iterować, dopóki warunek nie stanie się `false`. Warunek to (podobnie jak przy instrukcjach warunkowych) wyrażenie, które zwraca typ `boolean`, a więc `true` lub `false`. Oczywiście, warunek może być prosty lub bardziej rozbudowany (zawierać alternatywę, negację itd.).

Najprostszy przykład:

```
int licznik = 0;
while (licznik < 10) {
    // kod (ciało pętli)
    licznik++; // za każdą iteracją podnosi wartość licznik o 1
}
System.out.println(licznik); // 10
```

Przy każdej iteracji za pomocą inkrementacji zwiększyliśmy wartość licznika. Przy 10. iteracji licznik miał już wartość 10 i sprawdzenie `licznik < 10` zwróci wartość `false`. Na tym etapie pętla `while` zakończyła swoje działanie i program przeszedł dalej, w naszym wypadku wydrukował wartość licznika, czyli `10`.

Pętla do while (do-while)

Do while jest odmianą pętli while i odróżnia się od niej jedną cechą. Zawsze wykona się co najmniej raz. Zobaczmy przykład:

```
int licznik = 0;
while (licznik > 10) {
    // kod
} // nie wykona się ani razu, bo nie jest spełniony warunek.
```

```
int licznik = 0;
do {
    // kod
} while (licznik > 10);
// wykona się przynajmniej raz. W naszym przypadku właśnie raz.
```

Pętla for

Jest najczęściej używaną pętlą, choć potrafi przerazić wielu z tych, którzy zobaczą ją pierwszy raz.

Zobaczmy najpierw schemat pętli `for`:

```
for (inicjalizacja iteratora; warunek; iteracja) {
    // kod do wykonania (ciało pętli)
}
```

I teraz przykład, na którym lepiej zobaczymy, na czym to polega.

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

```
zostanie wydrukowane:
1 5
2 5
3 5
4 5
```

Przeanalizujmy teraz każdy fragment z pętli `for`:

- `int i = 1;` - pierwszy element inicjalizacji naszej pętli to iterator (zmienna sterująca pętlą), nie mamy obowiązku go tutaj inicjalizować (możemy wcześniej, przed pętlą), ale tak robimy najczęściej. Bardzo często używamy tutaj jako nazwy zmiennej literki 'i', jednak to tylko konwencja. Konwencja mówi także, że iteracje zaczynamy od 0, chyba że jest jakiś powód, by zacząć od innej wartości - w naszym przykładzie użyliśmy 1 i choć nie jest to błąd, to właściwe byłoby napisanie 0. Wynika to z tego, że tablice, do których wkrótce przejdziemy, które są główną strukturą, na której pracujemy pętlami, zaczynają się od zera (ich indeks zaczyna się od zera), ale o tym w niedalekiej przyszłości ☺. Zwróć też uwagę, że inicjalizację kończymy średnikiem a nie przecinkiem.
- `i < x;` - drugi element pętli to warunek, który mówi, jak długo wykonuje się pętla. Najczęściej liczba wykonań pętli jest uzależniona od wielkości obiektu, po którym iterujemy. Jeśli np. nasza tablica ma 10 elementów, to chcemy by warunek był taki: i < długość tablicy i wtedy, jeżeli tablica ma 10 elementów, otrzymalibyśmy zapis i < 10. Iterator wykona się wtedy 10 razy (o ile liczymy od zera czyli 0-9).
- `i++;` - trzeci element pętli to zwiększenie iteratora. Tu również nie jest wymagane, by była to inkrementacja, ale tak robimy najczęściej. To, czy użyjemy postinkrementacji (najczęściej i tak jak w naszym przykładzie), czy preinkrementacji, nie ma znaczenia.
- `System.out.println(i + " " + x);` - Wreszcie ciało funkcji składające się z instrukcji, w naszym przypadku z jednej. Kod wykona się tyle razy, ile pozwoli warunek.

Przeanalizujmy sobie jeszcze wszystkie kroki.

Kolejne omawiane kroki są zaznaczone we fragmentach kodu.

Krok 1 - Pętla zaczyna się od inicjalizacji, która wykonuje się tylko raz.

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

Krok 2 - następuje sprawdzenie warunku. Jeżeli warunek będzie `false`, pętla się kończy i program idzie dalej do kolejnej instrukcji poza pętlą. Jeśli warunek zwróci `true`, to program wchodzi do ciała pętli.

```
for (int i = 1; i < x; i++) {
```

```
    System.out.println(i + " " + x);
}
```

Krok 3 - program wykonuje ciało pętli

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

Krok 4 - program przechodzi do iteratora i w ten sposób zamyka jedną iterację. (w naszym przykładzie podnosi wartość zmiennej i o 1, czyli ma ona już wartość 2 na tym etapie)

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

krok 5 - program sprawdza warunek, zwróć uwagę, że nie wykonuje już inicjalizacji zmiennej, to wydarzyło się tylko raz, na samym początku wykonania pętli. Jeśli warunek zwraca `true`, przechodzi do ciała funkcji, jeśli nie, kończy pętlę. To już wiesz. I tak dalej ☺.

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

Sterowanie przebiegiem pętli

break

`break`, to słowo kluczowe, które przerywa pętle (i przechodzi do kolejnej po pętli instrukcji) w momencie, gdy program na nie natrafi. Zobaczmy przykład:

```
int x = 10;
for (int i = 0; i < x; i++) {
    System.out.println(i + " " + x);
    if (x - i == 7) // czyli gdy 10 - 3 się pojawi w naszym przykładzie
        break; // zakończ działanie pętli
}
```

```
Wydrukuje się:
0 10
1 10
2 10
3 10
```

Gdyby nie było słowa `break`, mielibyśmy wdrukowane także `4 10`, `5 10`, `6 10`, `7 10`, `8 10` i `9 10`. Wyrażenie `break` pozwoliło nam zakończyć pętlę po spełnieniu jakiegoś dodatkowego warunku. W tym przykładzie widać też połączenie pętli z instrukcją warunkową. To zupełnie normalne w programowaniu, podobnie jak instrukcja warunkowa w instrukcji warunkowej czy pętla w pętli.

continue

Słowo kluczowe `continue` użyte w pętli oznacza zakończ w tym momencie i przejdź do kolejnej iteracji.

```
int x = 4;
for (int i = 0; i < x; i++) {
    System.out.println(i);
    if (x - i < 3) {
        continue;
    }
    System.out.println(i);
}
```

```
Co zostanie wydrukowane?
0
0
1
1
2
3
```

Czy wiesz, dlaczego? Za 1. i 2. iteracją warunek w `if` nie był prawdziwy, dlatego wykonały się jedynie dwie instrukcje `System.out.println()`, które wydrukowały to samo. Za 3. i 4. iteracją warunek w `if` był już spełniony i program wszedł do `if`, gdzie napotkał słowo kluczowe `continue`. Słowo continue nakazało przerwać tę iterację i przejść do kolejnej (czyli do licznika, a następnie sprawdzenia warunku). W ten sposób to, co po `if`, nie zostało w tych iteracjach uwzględnione, dlatego 2 i 3 pokazały się na ekranie tylko raz.

Etykiety (loop labels)

Możemy nazwać pętle używając etykiety. Nie korzystamy z tego raczej, zarówno w "życiu" jak i w tym bootcampie. Bardzo szybko więc o co chodzi na konkretnym przykładzie.

Przykład 1 - pętla w pętli - obie nazwane za pomocą etykiety

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

W tym przykładzie widzimy zastosowanie nazw dla pętli. Oczywiście nie mamy obowiązku nazywać wszystkich, a tylko te, które rzeczywiście chcemy wykorzystać. Etykietowanie (nazywanie) pętli może się przydać, gdy chcemy skorzystać z instrukcji break lub continue, o czym zaraz się przekonamy. Teraz

skupmy się na samej konstrukcji nazwy. Zasady są jak przy nazwach zmiennych. Dodajemy także dwukropek, a potem już deklaracja. Proste. Zanim przejdziemy dalej zobaczymy co zostanie wydrukowane:

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

W tym przypadku pętla wykona się tak samo, jakby się wykonała, gdybyśmy nie napisali etykiet. Etykiety nie mają tutaj nic do rzeczy. A teraz sprawdźmy pewien przykład ze słowem kluczowym break.

Tu widzimy rozwiązanie, które kończy wykonanie danej pętli i zaczyna nową iteracje (w naszym przypadku zwiększa j o 1 i sprawdza warunek `j < 3`).

Przykład 2 - umieszczenie `break` i pętli

```
first: for (int i = 0; i < 3; i++) {
    second: for (int j = 0; j < 3; j++) {
        System.out.println("i = " + i + ", j = " + j);
        if (j == 1) {
            break;
        }
    }
}
```

```
Wydrukuje się:
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
```

Jak widzimy także w tym 2 przykładzie etykiety nie mają nic do tego jak pętla się wykonuje. Break zaś kończy pętle, w której break wystąpił (a więc w naszym przypadku wewnętrzną, a nie zewnętrzną). Ale za chwilę wreszcie zobaczymy ich, etykiet, znaczenie ☺.

Przykład 3 - połączenie `break` *i etykiety*

```
first: for (int i = 0; i < 3; i++) {
    second: for (int j = 0; j < 3; j++) {
        System.out.println("i = " + i + ", j = " + j);
        if (j == 1) {
            break first;
        }
    }
}
```

```
Wydrukuje się:
i = 0, j = 0
i = 0, j = 1
```

Zwróć uwagę na to, że nazwa pętli (jej etykieta) występuje po słowie break. Co się tutaj stało? Break zatrzymałby domyślnie działanie tylko pętli w której występuje. W tym wypadku wskazaliśmy przerwanie pętli first, a pamiętajmy, że wykonanie drugiej, wewnętrznej pętli (tej z etykietą second) jest częścią iteracji pętli z etykietą first. Pisząc `break first` wskazujemy pętlę, którą zatrzymujemy (kończymy), a kończąc zewnętrzną (`first`), kończymy także wewnętrzną (`second`).

Podobnie etykietę możemy zastosować przy słowie `continue`.

Słówko "return" w pętlach

Pętlę występują w metodach, a metody mogą być przerwane (kończone) przez słowo kluczowe **return**.

Pamiętajmy, że w ten sposób kończymy nie tylko działanie pętli, ale i całej metody. Kiedy program natknie się na `return,` kończy metodę i wychodzi z niej.

```
public static void example() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            System.out.println("i = " + i + ", j = " + j);
            if (i == 1) {
                return;
            }
            /* Instrukcja return kończy działanie metody na etapie,
            na którym program natrafi na nią.
            Po return zazwyczaj przekazujemy zwracaną wartość,
            ale jeśli korzystamy ze słowa void w definicji metody,
            to nic nie zwracamy (void znaczy, że metoda nic nie zwraca).
            Gdybyśmy mieli zamiast void np int, to moglibyśmy zwrócić,
            w naszym przykładzie oczywiście, return i;,
            bo i jest integerem. */
        }
    }
}
```

W naszym przykładzie uzyskalibyśmy efekt, w postaci wydrukowania na ekranie:

```
i = 0, j = 0
```

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

Jeśli masz problem ze zrozumieniem tego przykładu to śmiało przekopiuj kod od `for` np. do metody `main()` w swoim IDE i przeanalizuj go krok po kroku.