Pętle

Następnym elementem, który warto poznać są pętle. To jedne z podstawowych konstrukcji programowania. Dzięki nim program wielokrotnie wykonuje określoną instrukcję – do momentu, w którym zostanie spełniony podany warunek.

Dobra wiadomość jest taka, że coraz częściej się z nich rezygnuje na rzecz lambd.
Dlaczego dobra?
Problem z pętlami polega na tym, że jest w nich zawarta podwójna logika. Warto je jednak poznać, by zaznajomić się ze składnią Javy. To zaprocentuje w przyszłości.

Na następnej stronie proste zobrazowanie pętli w programie:



Jak widać program w trzech krokach wykonał pięć instrukcji – w tym trzy razy instrukcję 'B' – dzięki użyciu pętli w kroku drugim. I to pomimo tego, że fizycznie instrukcja została napisana tylko raz!

Pętle są dość trudne do zrozumienia dla osób początkujących, ale nie martwcie się, jeszcze dziś poznacie cztery rodzaje pętli używanych w języku Java.

1. Pętla while

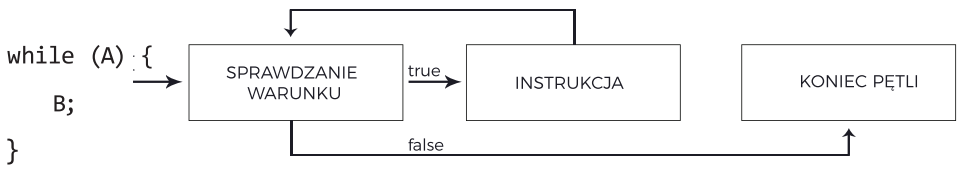
Pierwszą, najprostszą, i zarazem najrzadziej używaną, pętlą w Javie jest *while*.
Można ją zastosować, gdy znamy warunek, który musi zostać spełniony, ale nie wiemy, ile ma być powtórzeń pętli. Przyjrzyjmy się temu zjawisku na prostym przykładzie:

```
int x = 10;
while (x < 15){
x  = x+2;
System.out.println(x);
```

Nasz warunek mówi, że liczba musi być mniejsza niż 15.
Zmienna x początkowo wynosi 10.

Przy pierwszym obrocie pętli otrzymamy 12 – liczbę mniejszą niż 15, więc pętla ponownie się wykona. Za drugim podejściem uzyskamy 14. Znów warunek zostaje spełniony i pętla wykona kolejny obrót. Tyle, że tym razem wynik nie spełni warunku (16>15) i pętla się zakończy. Gdyby od początku warunek był nieprawdziwy, np. *while* (x<11){},

pętla nie wykonałaby się ani razu. Można też doprowadzić do sytuacji, w której pętla będzie obracać się w nieskończoność.
Poniżej proste zobrazowanie pętli:



Już na samym początku sprawdzany jest warunek (A). Jak widać na grafice w zależności od wyniku albo wykona się instrukcja (B) i obrócimy pętlą, albo nastąpi od razu koniec pętli i program wykona następny krok. Widać to na tym przykładzie:

```
int i = 2;
while (i<6){
   i = i + 3;
}
```

Krok 1:

`i = 2,`

sprawdzany jest warunek, czy `i<6`, czyli `2<6` —>TAK.

`i = i+3 , i=2+3, i = 5`

Krok 2:

```
i = 5
i<6, tak 5<6
i = i+3, i =5+3, i=8
```

Krok 3:

```
i<6, nie 8<6
```

Wyjście z pętli.

Widzimy, że program wykonując pętle *while*, musi sprawdzić warunek (A), czyli ten podany w nawiasach 'i<6'. Podstawiamy wartości za niewiadomą zmienną 'i', która przed wykonaniem pętli posiada wartość '2'.
Warunek '2<6' zwraca wynik prawdziwy, więc dojdzie do wykonania instrukcji (B), która brzmi: zmienna 'i' otrzyma nową wartość, równą dotychczasowej wartości dodanej do liczby '3'. Czyli: 'i=2+3', nasza zmienna 'i' przyjmie wartość '5' i pętla się obróci. Program znowu zatrzyma się przy sprawdzaniu warunku (A). Warunek znowu zwróci prawdę, ponieważ "5<6"; wykona się instrukcja, czyli: 'i=5+3' oraz dokona się kolejny obrót. Podczas ponownego sprawdzania warunku '8<6', wynik będzie nieprawdą i nastąpi koniec pętli. Program przejdzie do następnego kroku.

2. Pętla do while

Następną omawianą pętlą jest *do while*.

Od pętli *while* różni się tym, że najpierw wykonuje się instrukcja podana w pętli, a dopiero potem sprawdzany jest warunek.

Jak zauważyliście, używając pętli *while*, program może nigdy nie wykonać instrukcji (gdy warunek nie zostanie spełniony).

Przy pętli *do while* mamy pewność, że instrukcja zostanie wykonana przynajmniej raz.

Poniżej ogólna konstrukcja pętli:

```
do{
   B;
}while (A);
```



Jak widzimy, pierwsza jest wykonywana instrukcja (B), a dopiero później sprawdzany jest warunek (A). W zależności od wyniku, pętla obraca się i znowu wykonywana jest instrukcja (B). Zwróćcie uwagę na 'średnik' przy konstrukcji pętli przy słowie *while*.
Poniżej przykładowe zastosowanie pętli *do while*:

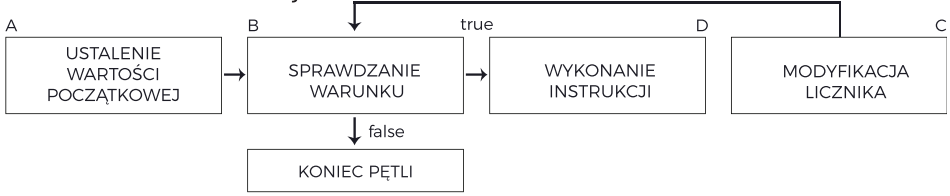
```
Scanner odczyt = new Scanner(System.in);
String password;
do {
   System.out.println("Podaj hasło (minimum 5 znaków)");
   password = odczyt.nextLine();
} while (password.isEmpty() || password.length() < 5);
System.out.println("Brawo! Podane hasło
spełnia nasze wymogi!");
```

W tym przykładzie program prosi nas o wymyślenie i podanie hasła, które odczyta nasz skaner 'odczyt' i zapisze pod zmienną 'password'.
Wykona to przynajmniej raz, bo to jest instrukcja (B).
Następnie dojdzie do sprawdzenia warunku (A), który brzmi: "jeżeli hasło jest puste albo hasło nie posiada długości pięciu znaków, to zwróć fałsz".
Jeżeli hasło, czyli nasza zmienna 'password', nie spełni warunku i zwróci 'false', to znów wykona się instrukcja (B) i program ponownie poprosi nas o podanie hasła. Znacie może skądś ten przykład?

3. Pętla for

Nadszedł czas na omówienie pętli *for*. Jest to najczęściej używana pętla, gdy znamy zakładaną liczbę powtórzeń. Na przykład, gdy chcemy policzyć do 10. Największą zaletą tej pętli jest kontrolowanie licznika. Wiemy dokładnie, podczas którego obrotu pętli się znajdujemy.
Poniżej ogólna konstrukcja pętli:

```
for (A; B; C;){
  D
}
```



Dla pętli *for* zawsze na początku ustanawiana jest wartość początkowa (A) – inaczej wartość sterująca. Gdy nic nie wpiszemy, domyślnie będzie to zero.
Następnym krokiem jest ustalenie warunku pętli (B). Identycznie jak ma to miejsce w zwykłej instrukcji warunkowej, na przykład: "jeżeli x>5". Gdy pominiemy wartość początkową oraz warunek w pętli *for*, to będzie ona pętlą nieskończoną. Do pełnego zapisu pętli *for* potrzeba nam modyfikacji licznika (C), czyli określenia, co ma się dziać z naszym licznikiem po każdym obrocie pętli. Gdy warunek jest prawdziwy, to dojdzie do wykonania instrukcji (D), modyfikacji licznika (D) i obrotu pętli.
Poniżej przykład:

```
for (int i = 0; i < 5; i++) {
   System.out.println(i);
}
```

Powyżej widać zainicjowanie wartości początkowej (A) prostym *intem* o wartości '0'. Następnie ustaliliśmy warunek (B), który brzmi: jeżeli 'i' jest mniejsze od '5'.
Na koniec modyfikator licznika (C), który zwiększamy z każdym obrotem o '+1', ponieważ zapis '++' oznacza '+1'.

Zróbmy symulację naszej pętli:

1. Program ustali wartość 'i' na wartość '0' (A);
2. sprawdzanie warunku (B): '0<5'. Wynik jest prawdziwy, więc zwróci nam true, dlatego program może wykonać instrukcję (D) zawartą w pętli;
3. instrukcja brzmi: "System drukuj wartość 'i'.";
4. teraz czas na modyfikację licznika (C), czyli: 'i++'. Wystarczy

podstawić i wykonać kalkulację '0+1=1', co da nam 'i=1';

5. w tym momencie pętla się obraca i wracamy do punktu sprawdzenia warunku: '1<5'. Wynik znowu jest prawdziwy, więc powtarzamy całą procedurę od nowa.

Taka sytuacja będzie miała miejsce do momentu piątego obiegu pętli. Wtedy to nasze 'i', po modyfikacji licznika, uzyska wartość 'i=5'. W tej sytuacji podczas sprawdzania warunku (B), czyli '5<5', wynik nie będzie prawdziwy i zwróci nam 'false'. Dlatego nastąpi koniec pętli i program przejdzie dalej.

4. Continue oraz break

Przy omawianiu pętli nie może zabraknąć dwóch słów kluczowych: *continue* – odpowiedzialne za wymuszenie obrotu pętli oraz *break* – za przerwanie aktualnej pętli.

Przykład continue:

```
for (int i = 0; i < 5; i++) {
   if (i % 2 == 0) {
      continue;
   }
   System.out.println(i);
}
```

Program wypisze nam tylko liczby 1 oraz 3, a to dlatego, że instrukcja w pętli zawiera warunek logiczny: reszta z dzielenia, który spełniają liczby parzyste. Przy tym warunku zadziała magiczne słówko *continue*, które wymusi obrót pętli i nie wykonają się instrukcje poniżej słówka *continue*, w naszym przykładzie "system drukuj".

Przykład break:

```
int i = 0;
while (true){
   if (i == 3){
      break;
   }
   System.out.println(i);
   i++;
}
```

Każda pętla bywa czasami pułapką, w której warunek będzie zawsze zwracał prawdę. A to będzie skutkować nieskończoną pętlą. Pętla może mieć też wiele warunków do zakończenia. Jednym z nich może być słówko kluczowe *break*. W przykładzie powyżej widzimy warunek dla 'i' równego '3' — koniec pętli. System wydrukuje nam kolejne liczby: 0, 1, 2.

Zadanie 1

Napisz program, który wyświetla wszystkie liczby od 5 do 10.

Rozwiązanie:

```
public class Zad1 {

   public static void main(String[] args) {
      for (int i = 5; i <= 10; i++) {
         System.out.println(i);
      }
   }

}
```

Zadanie 2

Napisz program, który wyświetla na ekranie pięć razy napis Hello.

Rozwiązanie:

```
public class Zad2 {

   public static void main(String[] args) {
      for (int i=1; i<=5; i++) {
         System.out.println("Hello");
      }
   }

}
```