Pętle

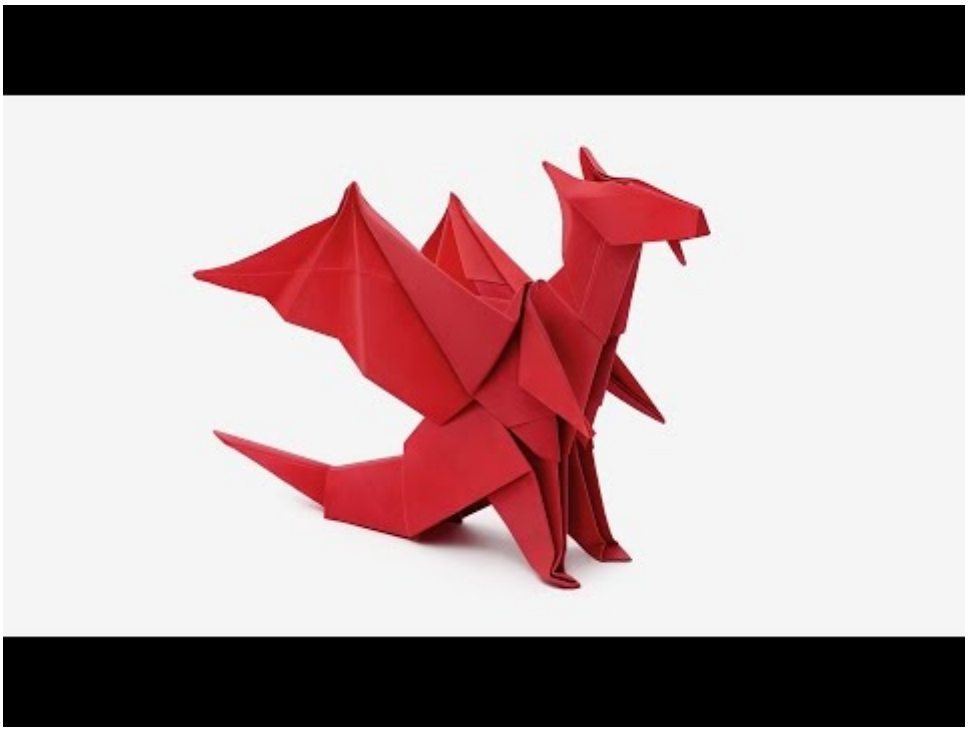
#loop #for #do #while #break #continue

Author: Piotr Niemczuk
Nickname: Pyoneru

Teoria

Wyobraź sobie że żyjesz w średniowiecznym mieście (takim w świecie fantasy), oprócz przygód potrzebujesz jeszcze mieć za co jeść. Masz talent do tworzenia rozmaitych figurek z papieru - origami. Postanawiasz że ta pasja zostanie Twoim sposobem na zarobienie na chlebek (i EQ).

Żeby móc się z tego utrzymac potrzebujesz zrobić wiele kopii tej samej figurki. Postanawiasz zacząć od smoków.



Jak będzie wyglądać Twoja praca?

Potrzebujesz instrukcji czy to w głowie czy to na papierze, jak złożyć takiego smoka z kartki papieru. Musisz w odpowiedniej kolejności i w odpowiedni sposób zaginać i uginać fragmenty kartki papieru.

Aby zrobić wiele figurek smoków, musisz powtórzyć te kroki wiele razy.

Gdy wyprodukujesz 100 figurek możesz otworzyć swój sklep i zobaczyć czy one się przyjmą.

Pętle to nic innego jak potwarzanie tego samego kroku, wiele razy.

Praktyka

W języku java (i innych również), mamy do dyspozycji trzy rodzaje pętli. Każda pętla powtarza instrukcje ale robię to troszeczkę inaczej. Różne pętle przydają się w różnych sytuacjach.

While

```
while(CONDITION){
 // instructions
}

while(CONDITION)
        // instruction
```

Pętle typu while tworzymy podobnie do instrukcji **if**, z tym że zamiast słowa kluczowego **if**, używamy słowa kluczowego **while**.

Do póki **warunek** (*ang.* ***condition***) jest prawdziwy, czyli zwróci nam wartość **true**, dopóty pętla sie będzie wykonywać. Warunek jest najpierw sprawdzany, a później są wykonywane instrukcje.

Podobnie jak w if'e, możemy opuścić klamary `{}` jeżeli pętla wykonuje tylko jedną instrukcje

Przykład, program wypisze liczby od 1 do 10. Wypróbuj!

```
int i = 1;
while(i <= 10){
        System.out.println(i);
        i++;
}
```

Do pracy z pętlami mamy dodatkowe słowa kluczowe - **break** i **continue**.

break

break - przerywa działanie pętli

```
int x = 10;
while(x > 0){
        System.out.println(x);
        if(x % 4 == 0) break; // jeżeli x będzie podzielne przez 4, to pętla
się zakończy
        x--;
}
```

Output

```
10
9
8
```

Wypisze najpierw (*do something*), a dopiero potem przerywa pętle.

✓ **Inkrementacja i Dekrementacja | Nowe operatory arytmetyczne**

Mamy jeszcze dwa operatory arytmetyczne, inkrementacji i dekrementacji. Zwiększają (inkrementacja) lub zmniejszają (dekrementacja) wartość zmiennej o 1.

```
x++; // zwieksz wartość x o 1
// to samo można osiągnąć w ten sposób
x += 1;
```

Analogicznie z dekrementacją.

```
x--;
x -= 1;
```

continue

Jeżeli chcemy przerwać ale tylko jeden obieg pętli, a nie całą pętlę to możemy użyć słowa kluczowego **continue**.

```
int x = 10;
while(x > 0){
        if(x % 4 == 0) {
                x--;
                continue;
        } // jeżeli x będzie przez 4, to przerwij ten obieg pętli
        System.out.println(x)
        x--;
}
```

Teraz przerwamy obieg pętli przed wypisaniem wartości.

Output

```
10
9
7
6
5
3
2
1
```

8 i 4 są podzielne przez 4 więc nie zostaną wypisane.

Do...While

```
do{
 // instructions
}while(CONDITION);
```

W przeciwieństwie do pętli **while**, w pętli **do...while** instrukcje są najpierw wykonywane, a warunek jest sprawdzany później. Oznacza to że pętla **do...while** wykona się co najmniej raz, zawsze.

> ⓘ **Do czego to może się przydać?**

Pętle do while możesz wykorzystać do sprawdzania hasła. Hasło musisz podać co najmniej raz, jeżeli hasło jest poprawne to pętla zakończy się od razu. Jeżeli hasło jest złe, to pętla zapyta ponownie o hasło.

```
Scanner in = new Scanner(System.in);
String correctPassword = "admin1";
String password; // zmienną musimy stworzyć poza pętlą żeby móc ją wykorzystać
w warunku. Sprawdź blok kodu w poprzednim kroku.
do{
    System.out.print("Podaj hasło: ");
    password = in.nextLine();
        if(!correctPassword.equals(password)) // powtórzenie warunku aby
wypisać komunikat o błędzie
        System.out.println("Złe hasło!");
}while(!correctPassword.equals(password)); // dopóty password jest różne od
correctPassword, dopóty powtarzaj pętle
```

for

Ostatnią pętlą jest pętla **for**, która pozwala nam w prosty sposób śledzić kroki pętli.

```
for(INIT;CONDITION;NEXT_STEP){
 // instructions
}

for(INIT;CONDITION;NEXT_STEP)
 // instruction
```

Ta pętla na pierwszy rzut oka jest bardziej skomplikowana. Ale tak na prawdę jest to skrócony zapis czegoś co robiliśmy w pętli **while**.

```
int i = 0; // INIT
while(i < 10){// CONDITION
        // do something
        i++; // NEXT_STEP
}
```

```
for(int i = 0; i < 10; i++){
        // do something
}
```

> ⚠ **INIT jest niewidoczny poza forem**
>
> Sekcja INIT w pętli for jest dostępna tylko w pętli for, czyli miedzy nawiasami `{}`. Czyli nie możesz użyć zmiennej `i` utworzonej w pętli for, poza nią.

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

Zagnieżdzanie pętli

Pętle również możemy zagnieżdżać, czyli umieszczać jedną pętlę wewnątrz drugiej.

```
for(int x = 0; x < 10; x++){
        for (int y = 0; y < 10; y++){
                // do something or more loops :)
        }
}
```

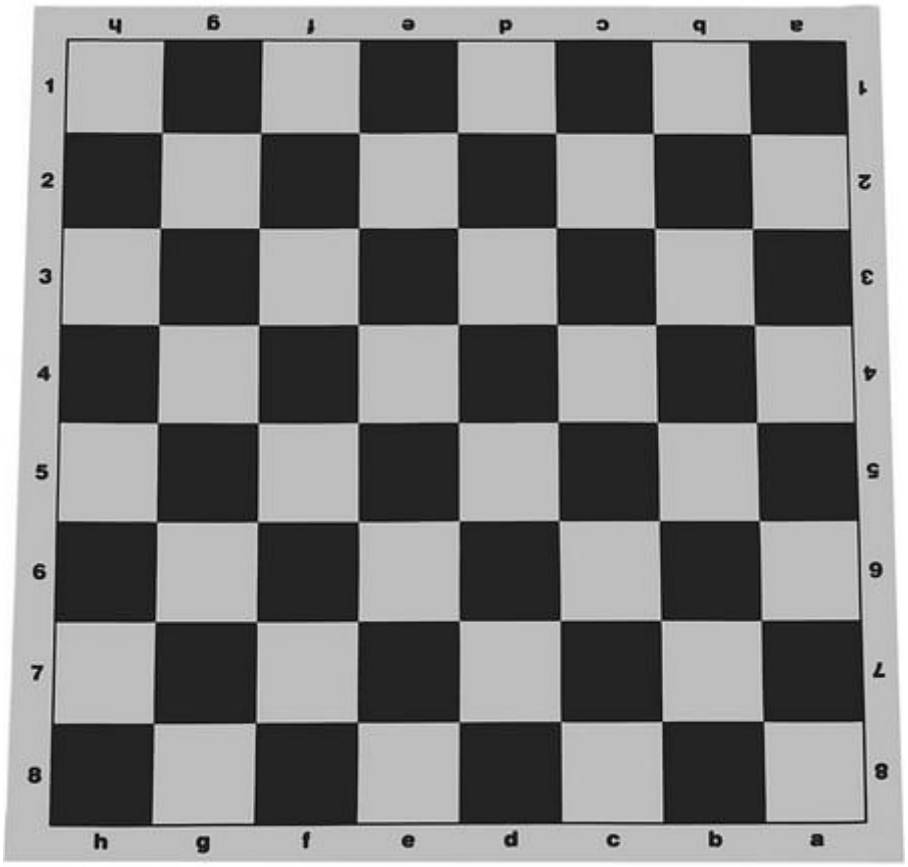
Możemy umieszczać więcej niż jedną pętle wewnątrz drugiej.

```
for(int row = 0; row < 100; row++){
        for(int column = 0; column < 10; column++){
                // do something or more loops :)
        }

        for(int time = 0; time < 100; time++){
                // do something or more loops :)
        }
}
```

Pamietaj o konsekwencjach zagnieżdzania pętli.

Wyobraź sobie szachownice. Czyli siatke o rozmiarze 8×8. Aby wypisać lub zrobić coś na podstawie każdego punktu musimy przejść przez wszystkie wiersze i kolumny.



Możesz sobie to wyobrazić tak że zaczynasz od literki `a` na szachownicy, a następnie przeskakujesz po polach w pionie po liczbach `1..8`. Następnie przechodzisz do kolejnej literki, i znów skaczesz w polach w pionie po liczbach `1..8`. I tak aż do literki `h`. W rezultacie przeszedłeś po 64 polach → 8 razy po 8 razy.

```
for(int x = 0; x < 8; x++){ // literki możemy zastąpić kolejnymi liczbami 1..8
    for (int y = 0; y < 8; y++){
        System.out.println(x + ":" + y);
    }
}
```

Zewnętrza pętla X wykona się 8 razy, co jest oczywiste.

Wewnętrzna pętla Y sama w sobie wykona się również 8 razy, jednak jest ona wewnątrz innej i jej pełen obieg wykona się tyle razy ile wykona sie zewnętrzna pętla.

Pełen obieg w tym przypadku to 8.
Zewnętrzna pętla X wykonuje się 8 raz, pętla Y również.
8 * 8 = 64.

Podsumowanie

Dzisiaj poznałeś/aś potężne narzędzie, pętle ! Każdy potężny program czy gra korzysta z pętli do działania.

Pamiętaj żeby dać znać czy coś jest nie jasne :) Staram sie pisać coraz zwieźlej, więc jeżeli czegoś nie zrozumiałeś/aś to masz obowiązek mnie o tym poinformować!

Fajny smok, co nie?