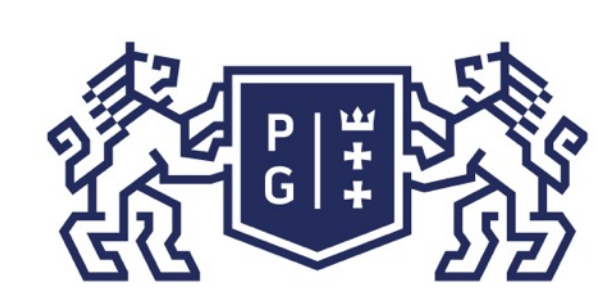




Język Java podstawy

Jacek Rumiński



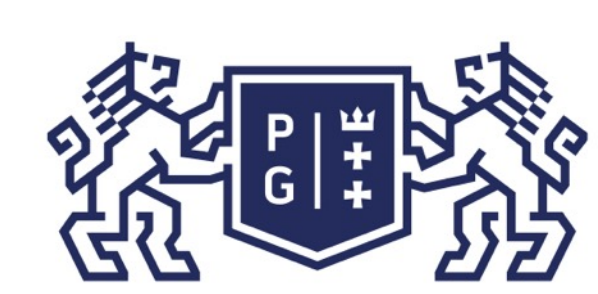
Język Java podstawy

Jacek Rumiński



Katedra Inżynierii Biomedycznej,
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska





1. Instrukcje sterujące

2. Eksperymenty praktyczne z instrukcjami sterującymi

Składnia

```
while(wyrażenie_logicze)  
    wyrażenie
```

Pętla ("tak długo aż ... wykonuj...") powoduje wykonywanie wyrażenia tak długo, dopóki rezultat wyrażenia logicznego jest true. Wyrażenie stanowi zazwyczaj blok programu wyróżnialny klamrami.

```
int i=3;  
while(i>1) { // (3>1) true, następnie (2>1) true, ale (1>1) false -> koniec pętli  
    System.out.println("Java jest super!"); //wydruk napisu 2 razy  
    i--; //zmniejsz i o 1, inaczej i=i-1;  
}
```

Wyrażenia (zawarte instrukcje) w pętli `while(){}` mogą się ani razu nie wykonać jeśli na samym początku nie spełniony jest warunek (wyrażenie logiczne prowadzi do `false`).

Składnia

do

wyrażenia

while(wyrażenie_logiczne);

Podobnie jak pętla ("wykonuj ... tak długo aż ... ") **while** sprawdzany jest tu rezultat wyrażenia logicznego. Jeżeli rezultat ten jest **false** wówczas przerywana jest pętla. Różnica pomiędzy pętlami: **while** – warunek na początku, **do-while** – warunek na końcu.

```
int i=3;
do{
    System.out.println("Java jest super!"); //wydruk napisu 2 razy
    i--; //zmniejsz i o 1, inaczej i=i-1;
}while(i>1) { // (3>1) true, następnie (2>1) true, ale (1>1) false -> koniec pętli
```

Wyrażenia (zawarte instrukcje) w pętli wykonane zostaną co najmniej jeden raz (sprawdzanie warunku na końcu).

Składnia

**for(inicjowanie; wyrażenie_logiczne ; krok)
wyrażenia**

Pętla **for** ("powtarzaj z krokiem... tak długo aż...") powoduje wykonanie wyrażenia tyle razy ile to wynika z warunku (wyrażenie_logiczne) przedstawionego w wywołaniu pętli. Warunek ten polega na określeniu wartości początkowej licznik powtórzeń (iteracji), określeniu warunku końca oraz kroku zmiany licznika powtórzeń (iteracji).

```
for(int i=1; i<3;i++) {  
    System.out.println("Java jest super!"); //wydruk napisu 2 razy  
}  
for(;;){ //pętla nigdy się nie kończy  
    System.out.println("Java jest super!"); //wydruk nieskończenie wiele razy  
}
```

Pętla **for** stosowana jest najczęściej wówczas, gdy chcemy wykonać określone instrukcje N (kilka, kilkaset, kilka tysięcy,...) razy.

Zastosowanie

1. wykonuj operacje od wartości początkowej do 0

```
for(int i=10; i>0;i--) //tylko 1 operacji w pętli - nawiasy {} są niepotrzebne  
    System.out.println(i+": Java jest super!"); //wydruk napisu 10 razy
```

2. wykonuj operacje od wartości początkowej do 0, z krokiem 2

```
for(int i=10; i>0;i-=2) //i-=2 to to samo co i=i-2  
    System.out.println(i+":Java jest super!"); //wydruk napisu 5 razy
```

3. zastosowanie dwóch liczników powtórzeń i, j (zawsze 1 warunek)

```
for(int i=10, j=5; i>0;i--,j++) //tylko 1 warunek; dwie zmienne iteracyjne i, j  
    System.out.println("Wartości zmiennych i= "+i+", j= "+j); //sprawdź sam!
```

4. to samo co w punkcie 1, tylko inny zapis

```
int i=10;  
for(; i>0;) { //potrzebne nawiasy {} bo 2 instrukcje  
    System.out.println(i+":Java jest super!"); //wydruk napisu 10 razy  
    i--;  
}
```

Składnia

for(typ zmienna: zbiór) wyrażenia

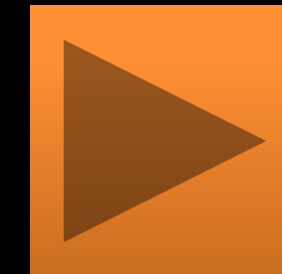
Pętla **for** ("powtarzaj dla każdej możliwej wartości ze zbioru...") powoduje wykonanie wyrażenia tyle razy ile różnych wartości może przyjąć zmienna dla danego zbioru. Inaczej, pętla będzie tyle razy wykonana, ile wynosi liczba elementów zbioru.

```
public static long suma(int ... argumenty){// wywołanie np. suma(2,34,3,987);  
    long suma=0;  
    for(int n : argumenty){//liczba podanych argumentów stanowi zbiór  
        suma+=n;  
    }  
    return suma;  
} //koniec suma()
```

W powyższym przykładzie pojawił się nowy symbol "..." oznaczający początek definicji listy argumentów.

Demo ->

Kod programu: ObliczeniaJedi.java



```
public class ObliczeniaJedi{
```

```
/** Uniwersalny sumator liczba całkowitych*/
```

```
public static long suma(int ... argumenty){
```

```
    long suma=0;
```

```
    for(int n : argumenty){
```

```
        suma+=n;
```

```
    }
```

```
    return suma;
```

```
}//koniec suma()
```

```
public static void main(String args[]){
```

```
    System.out.printf("Suma wartosci 1,2,3,4,5,6,7 to %2d\n", ObliczeniaJedi.suma(1,2,3,4,5,6,7));
```

```
    System.out.printf("Suma wartosci 10,20,30,40 to %2d\n", ObliczeniaJedi.suma(10,20,30,40));
```

```
}//koniec main()
```

```
}//koniec class ObliczeniaJedi
```

Dla wszystkich pętli stosować można polecenia **break** i **continue** umieszczane w ciele pętli. Polecenie **break** przerywa pętlę i ją kończy (następuje wyjście za pętlę do kolejnej instrukcji). Polecenie **continue** przerywa pętlę dla danej iteracji i rozpoczyna następną iterację.

```
for(int i=1; i<10;i++) {  
    if(i%2==0) continue; //jeśli reszta z dzielenia przez 2 wynosi 0 wróc do for  
    System.out.println(i+":Java jest super!");//wydruk napisu 5 razy  
}  
for(;;){//pętla nieskończona, ale  
    System.out.println("Java jest super!");  
    break; //wyjście z pętli po jednym wydruku  
}  
int i=3; //część for(int i=3;;)  
while(true){ //część for(;;)  
    if( (--i)<0 ) break; //część for(;i>0;i--) razem pętla for(int i=3; i>0;i--)!  
    System.out.println(i+":Sen też jest fajny");  
}
```


Składnia

```
switch(wyrażenie_wyboru) {  
    case wartość1 : wyrażenie; break;  
    case wartość2 : wyrażenie; break;  
    case wartość3 : wyrażenie; break;  
    case wartość4 : wyrażenie; break;  
    // ...  
    default: wyrażenie;  
}
```

Instrukcja wyboru **switch** powoduje sprawdzenie stanu wyrażenia_wyboru (zmiennej liczbowej) i w zależności od jej stanu (wartości) wybierane jest zgodne wyrażenie. Słowo **break** oznacza przerwanie działania w instrukcji wyboru (nie jest wykonywane kolejne wyrażenie). Domyślne wyrażenie jest wykonywane dla wszystkich innych stanów niż te, wymienione w ciele instrukcji.

[Demo ->](#)

Kod programu: WyborJedi.java



```
public class WyborJedi{  
//throws Exception deklarujemy brak obsługi wyjątków dla tej metody  
//Obsługa jest delegowana dalej. W tym przypadku do systemu (Maszyna Wirtualna)  
//Konieczność deklaracji jest wymuszone przez użycie metody System.in.read()  
    public static void main(String args[]) throws Exception{  
        System.out.println("Wybierz liczbę mieczy: 1, 2, 3 lub 4 i naciśnij Enter");  
        int test = System.in.read();//czytaj z klawiatury  
        switch(test){  
            case 49: System.out.println("Wybrano 1"); break;  
            case 50: System.out.println("Wybrano 2"); break;  
            case 51: System.out.println("Wybrano 3"); break;  
            case 52: System.out.println("Wybrano 4");  
            /* Brak break; program przejdzie dalej do pola default  
            i wykona odpowiednie instrukcje */  
            default:  
                System.out.println("Wybrano cos");  
            }//koniec switch  
        }//koniec main()  
    }// koniec public class WyborJedi
```

Składnia **return (wyrażenie);**

Instrukcja powrotu **return** kończy metodę, w ciele której się znajduje i powoduje przeniesienie wartości wyrażenia do kodu wywołującego daną metodę. Oznacza to, że typ wartości wyrażenia instrukcji **return** musi być zgodny z typem zadeklarowanym w czasie definicji metody, w ciele której znajduje się instrukcja **return**. Jeśli typ jest **void** nie trzeba stosować **return** (ale można jak **break** dla pętli **return**;))

```
public int suma(int a, int b){
    return (a+b);
}
public boolean test (int wiek) {
    if( wiek < 18) return false;
    else return true;
}
public String toString(){
    return "Ten obiekt reprezentuje rycerza Jedi";
}
```



1. Instrukcje sterujące

2. Eksperymenty praktyczne z instrukcjami sterującymi



Zapraszamy na kolejną część materiału

