

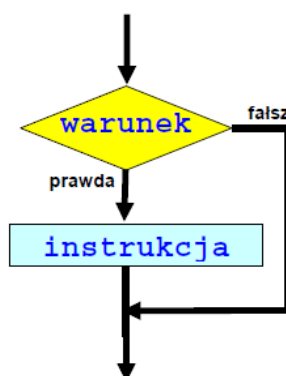
Wykład III

Podstawowe instrukcje w języku Java

Składnia instrukcji warunkowej **if**

Instrukcja warunkowa **if** służy do podjęcia decyzji, gdzie wykonanie instrukcji jest uzależnione od spełnienia jakiegoś warunku.

```
if (warunek)
{
    instrukcja;
}
```



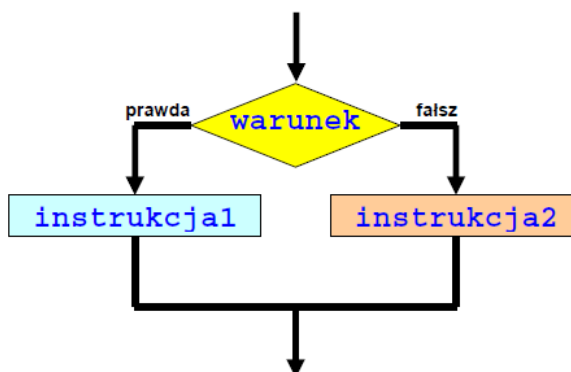
+

Postać instrukcji warunkowej if - else

Instrukcja warunkowa if ... else służy do zapisywania decyzji, gdzie wykonanie jednej z alternatywnych instrukcji zależy od spełnienia jakiegoś warunku.

Jeśli warunek jest prawdziwy to wykonywana jest instrukcja1, w przeciwnym wypadku wykonywana jest instrukcja2.

```
if (warunek)
{
    instrukcja1;
}
else
{
    instrukcja2;
}
```



Przykład programu: Obliczenie pierwiastków równania kwadratowego.

```
import javax.swing.*;
public class Instrukcja_if {

    public static void main(String[] args) {
        double a, b, c, delta, x1, x2;
        a = Double.parseDouble(JOptionPane.showInputDialog(null, "Podaj a:"));
        b = Double.parseDouble(JOptionPane.showInputDialog(null, "Podaj b:"));
        c = Double.parseDouble(JOptionPane.showInputDialog(null, "Podaj c:"));
        System.out.println("\n" + a + " x^2 + " + b + " x + " + c + " = 0");
        if (a==0)
        { System.out.println("\n To nie jest równanie kwadratowe");
          System.exit(0);
        }
        delta = b*b-4*a*c;
        System.out.println("\n delta=" + delta);
        if (delta>0){
            x1 = (-b - Math.sqrt(delta))/(2*a);
            x2 = (-b + Math.sqrt(delta))/(2*a);
            System.out.println("\n x1=" + x1 + " x2=" + x2);
        }
        else if (delta==0){
            x1 = -b/(2*a);
            System.out.println("\n x1=" + x1);
        }
        else System.out.println("\n To równanie nie ma pierwiastków");
    }
}
```

+

Instrukcji wyboru - switch

Instrukcja **if else** przy znacznej liczbie opcji wyboru jest nieefektywna. Instrukcja wyboru **switch** pozwala na podjęcie decyzji, wtedy gdy o wyborze opcji decyduje wartość skalarna jakiegoś wyrażenia testowego lub zmienna. Etykiety **case** mogą być następujących typów :

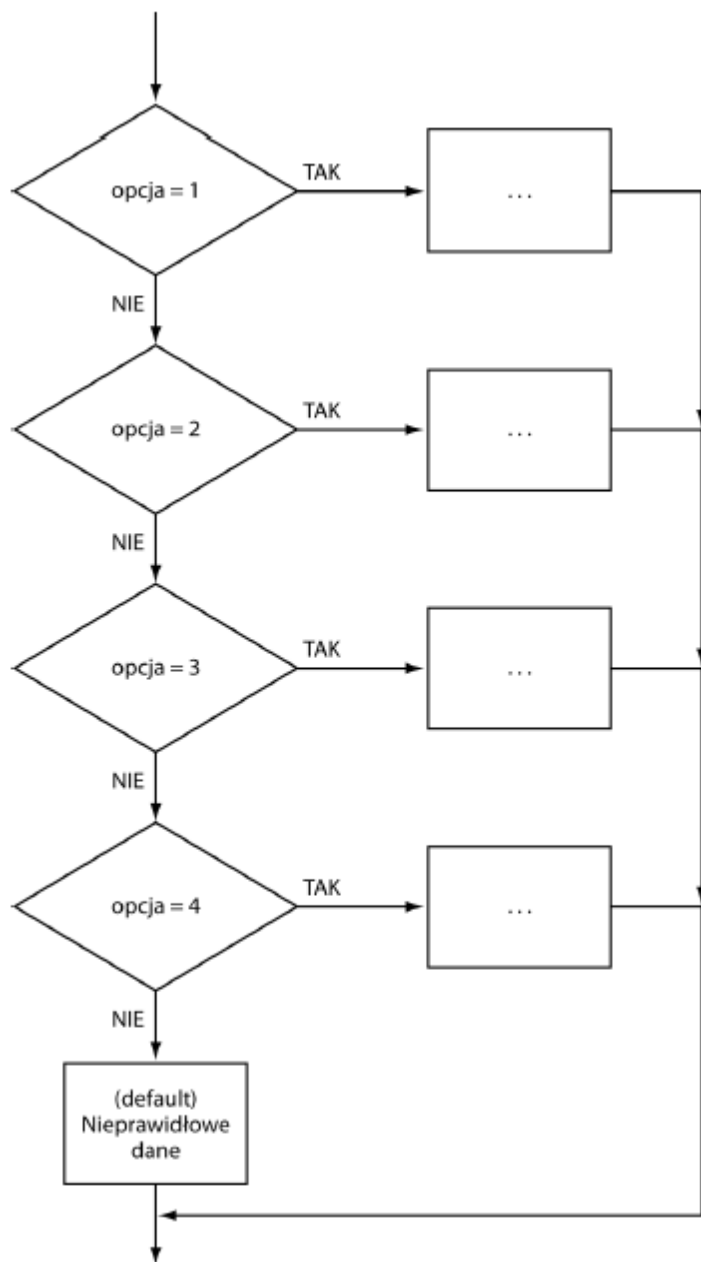
- wyrażeniami stałymi typu char, byte, short lub int (oraz odpowiednich klas opakowujących: Character, Byte, Short i Integer)
- stałymi wyliczeniowymi;
- łańcuchami od wersji Java SE 7.

Wynik wyrażenia jest porównywany z wyrażeniami stałymi (np. literałami) występującymi po słowie kluczowym **case**. W przypadku zgodności wykonywana jest odpowiednia instrukcja po dwukropku i następujące po niej kolejne instrukcje aż do napotkania instrukcji **break** lub **return**.

Jeśli żadna z etykiet po słowie **case** nie jest zgodna z wartością wyrażenia testowego to wykonywana jest instrukcja po klauzuli default jeżeli istnieje.

```
import java.util.Scanner;
public class SW1 {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Wybierz liczbą (1, 2, 3, 4) ");
        int test = in.nextInt();
        switch (test)
        {
            case 1: System.out.println(" Podana wartość to: " + test);
                    break;
            case 2: System.out.println(" Podana wartość to: " + test);
                    break;
            case 3: System.out.println(" Podana wartość to: " + test);
                    break;
            case 4: System.out.println(" Podana wartość to: " + test);
                    break;
            default:
                // Nieprawidłowe dane.
                System.out.println(" Podano nieprawidłową wartość");
                break;
        }
    }
}
```

Algorytm działania instrukcji **switch**

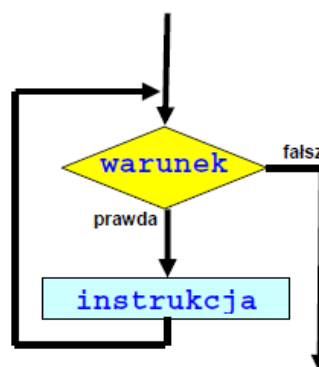
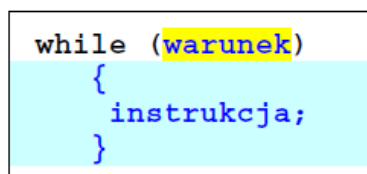
Pętle w języku Java

Pętla while

Pętla **while** wykonuje instrukcję (albo blok instrukcji) tak długo, jak długo warunek ma wartość **true**. Ogólna postać instrukcji **while** jest następująca:

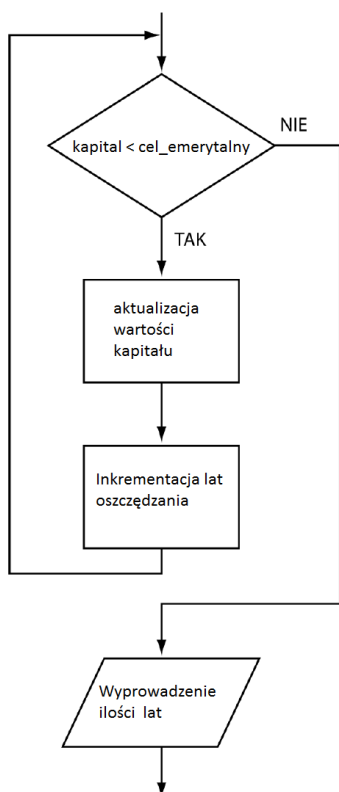
```
while (warunek)
{
    instrukcje;
}
```

Instrukcje pętli **while** nie zostaną nigdy wykonane, jeśli warunek ma wartość **false** na początku wykonywania pętli.



Przykład:

Program oblicza, ile czasu trzeba składać pieniądze, aby uzbierać odpowiedni kapitał emerytalny, przy założeniu, że każdego roku wpłacana jest taka sama kwota, i przy określonej stopie oprocentowania wpłaconych pieniędzy. (procent składany)



Kod źródłowy programu:

```
import java.util.*;
public class Emerytura {

    public static void main(String[] args) {
        // ile lat należy pracować aby zgromadzić kapitał emerytalny
        int srednia_dlugosc_zycia = 75;
        int wiek_emerytalny = 67;
        int srednia_emerytura = 3000;
        int lata_pracy = 0;
        double cel_emerytalny ;
        double wpłata_ZUS ;
        double kapitał = 0;
        double procent ;
        Scanner in = new Scanner(System.in);
        System.out.print("Ile pieniędzy potrzebujesz, aby przejść na emeryturę? ");
        cel_emerytalny = in.nextDouble();
        System.out.print("Ile pieniędzy rocznie będziesz wpłacać? ");
        wpłata_ZUS = in.nextDouble();
        System.out.print("Stopa procentowa w %: ");
        procent = in.nextDouble();

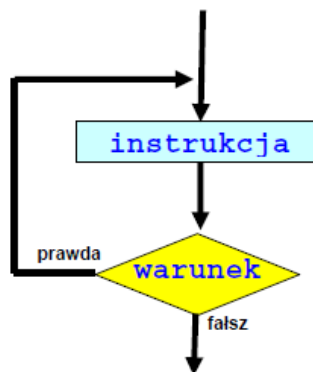
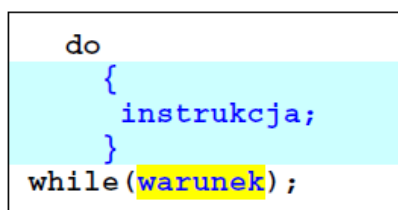
        while (kapitał < cel_emerytalny)
        {
            kapitał += wpłata_ZUS;
            double odsetki = kapitał * procent / 100;
            kapitał += odsetki;
            lata_pracy++;
        }
        System.out.println("Musisz pracować tylko " + lata_pracy + " lat.");
        //Wypłata miesięczna
        double wypłata = kapitał / (srednia_dlugosc_zycia - wiek_emerytalny);
        System.out.println("Wypłata miesięczna będzie wynosiła =" + wypłata + " zł.");
    }
}
```

Pętla **do ... while**

Pętla **while** sprawdza warunek na samym początku działania. W związku z tym jej instrukcje mogą nie zostać wykonane ani razu. Aby mieć pewność, że instrukcje zostaną wykonane co najmniej raz, sprawdzanie warunku trzeba przenieść na sam koniec. Do tego służy pętla **do ... while**. Jej składnia jest następująca:

do instrukcja while (warunek)

Ta instrukcja najpierw wykonuje instrukcję (która zazwyczaj jest blokiem instrukcji), a dopiero potem sprawdza warunek. Następnie znowu wykonuje instrukcję i sprawdza warunek itd.



Algorytm realizacji pętli



Przykładowy kod oblicza nowe saldo na koncie emerytalnym, a następnie pyta, czy jesteśmy gotowi przejść na emeryturę:

```
import java.util.*;

public class Emerytura2 {

    public static void main(String[] args)
    {
        // Wczytanie danych.
        Scanner in = new Scanner(System.in);
        System.out.print("Ile pieniędzy potrzebujesz, aby przejść na emeryturę? ");
        double cel_emerytalny = in.nextDouble();
        System.out.print("Ile pieniędzy rocznie będziesz wpłacać? ");
        double wpłata_ZUS = in.nextDouble();
        System.out.print("Stopa procentowa w %: ");
        double procent = in.nextDouble();
        double kapitał = 0;
        int lata_pracy = 0;
        String input;
        // Aktualizacja stanu konta, kiedy użytkownik nie jest gotowy do przejścia na
        // emeryturę.
        do
        {
            // Dodanie tegorocznych płatności i odsetek.
            kapitał += wpłata_ZUS;
            double odsetki = kapitał * procent / 100;
            kapitał += odsetki;
            lata_pracy++;
            // Drukowanie aktualnego stanu konta.
            System.out.printf("Po upływie %d lat stan twojego konta wyniesie %,2f%n",
                lata_pracy, kapitał);
            // Zapytanie o gotowość do przejścia na emeryturę i pobranie danych.
            System.out.print("Chcesz przejść na emeryturę? (T/N) ");
            input = in.next();
        }
        while (input.equals("N") || input.equals("n") );
    }
}
```


Pętle o określonej liczbie powtórzeń - pętla For

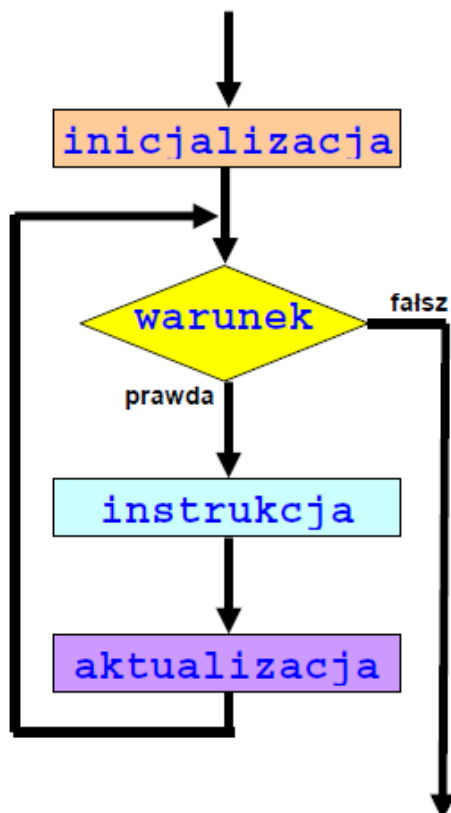
W nagłówku pętli for podawane są następujące elementy : inicjalizacja , warunek powtórzenia oraz aktualizacja. Inicjalizacja jest wykonywana przed rozpoczęciem wykonywania pętli.

Warunek jest sprawdzany przed każdą iteracją i jeśli jest spełniony wykonywana jest instrukcja wewnątrz pętli i następująca po niej aktualizacja. W przeciwnym razie pętla jest przerywana. Liczba iteracji instrukcji for jest kontrolowana za pomocą licznika lub innej zmiennej, której wartość zmienia się po każdym powtórzeniu.

Składnia instrukcji for:

```
for(inicjalizacja; warunek; aktualizacja)
{
    instrukcja;
}
```

Algorytm działania pętli

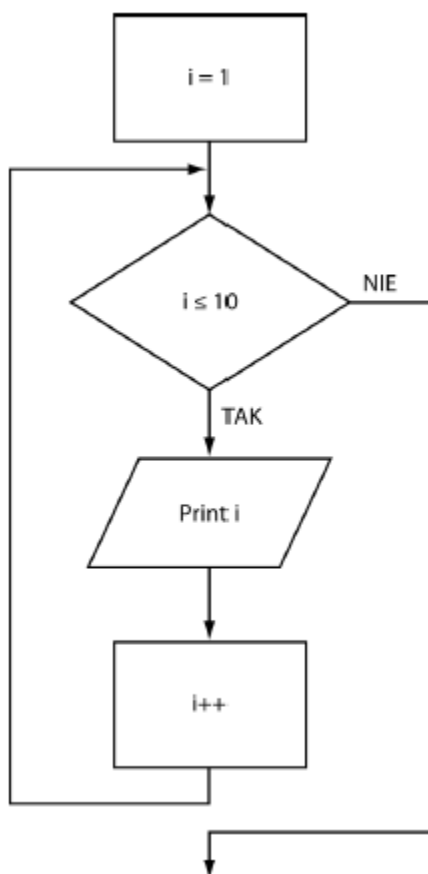


Przykład:

```
public class Petla_for {

    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++)
            System.out.println(i);
    }
}
```

Algorytm działania programu



Zmienna zadeklarowana na pierwszej pozycji w pętli for ma zasięg do końca ciała tej pętli.

```
for (int i = 1; i <= 10; i++)
{
    // Tutaj zmienna i już nie jest dostępna.
}
```

Wartość zmiennej zadeklarowanej w wyrażeniu pętli for nie jest dostępna poza tą pętlą. W związku z tym, aby móc użyć wartości licznika pętli poza tą pętlą, trzeba go zadeklarować poza jej nagłówkiem!

```
int i;
for (i = 1; i <= 10; i++)
{
    // Zmienna i tutaj też jest dostępna.
}
```

W różnych pętlach `for` można zdefiniować zmienną o takiej samej nazwie:

```
for (int i = 1; i <= 10; i++)
{
    // W tym miejscu dozwolona jest ponowna deklaracja zmiennej i.
    for (int i = 11; i <= 20; i++)
    {
    }
```

Porównanie instrukcji iteracyjnych

Pętle `while` oraz `do...while` stosujemy zwykle wtedy, gdy kontynuacja działania pętli zależy od jakiegoś warunku, a liczba iteracji nie jest z góry znana lub jest trudna do określenia. Pętla `for` jest stosowana najczęściej przy organizacji pętli iteracyjnych ze znanym zakresem iteracji.

Pętlę `for` można łatwo przekształcić na pętlę `while`. Ilustruje to poniższe zestawienie:

<pre>for (inicjalizacja; warunek; aktualizacja) { instrukcja; }</pre>	<pre>inicjalizacja; while(warunek) { instrukcja; aktualizacja; }</pre>
---	--

+

Instrukcje przerywające przepływ sterowania

Instrukcja `break` - przerwanie pętli

Instrukcja `break` powoduje przerwanie wykonywania pętli. W przypadku pętli zagnieżdżonych przerywana jest pętla wewnętrzna, w której bezpośrednio znajduje się instrukcja `break`.

Jeśli po instrukcji `break` występuje etykieta, to przerywana jest ta pętla lub blok instrukcji, która jest opatrzona tą etykietą.

Uwaga: etykieta musi być umieszczona bezpośrednio przed pętlą lub blokiem instrukcji, które mają być przerywane.

Instrukcja break - bez etykiety

Składnię zastosowania instrukcji break prześledzimy na następującym przykładzie
Na przykład:

```
public class Instrukca_break {

    public static void main(String[] args)
    {
        System.out.println("POCZATEK 1");
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<100; j++)
            {
                if (j==10) break;
                System.out.print(j + " ");
            }
            System.out.println();
        }
        System.out.println("KONIEC 1\n");
        System.out.println("POCZATEK 2");
    }
}
```

Instrukcja break - z etykietą

```
etykieta:
for(int i=0; i<3; i++)
{
    for(int j=0; j<100; j++)
    {
        if (j==10) break etykieta;
        System.out.print(j + " ");
    }
    System.out.println();
}
System.out.println("\nKONIEC 2\n");
}
}
```

Wynik działania programu;

```
POCZATEK 1
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
KONIEC 1

POCZATEK 2
0 1 2 3 4 5 6 7 8 9
KONIEC 2
```

Instrukcja continue - bez etykiety

Kontynuowanie pętli – instrukcja continue

Instrukcja continue przerywa wykonywanie bieżącego kroku pętli i wznowia wykonanie kolejnej iteracji. W przypadku pętli zagnieżdżonych działanie to dotyczy tej pętli wewnętrznej, w której jest umieszczona instrukcja continue.

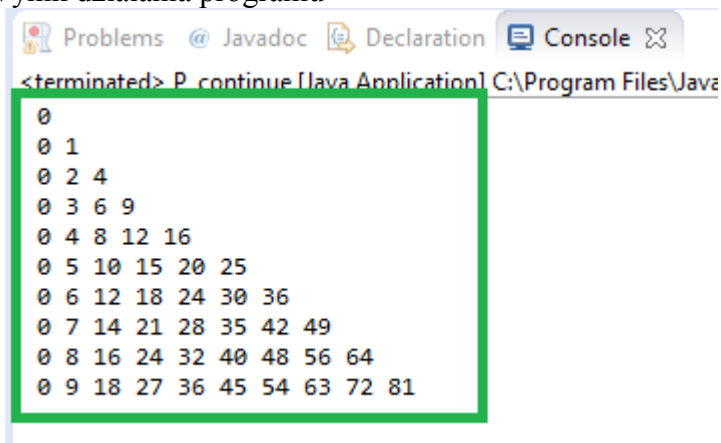
Jeśli po instrukcji continue występuje etykieta, to wznowiana jest iteracja tej pętli, która jest opatrzona tą etykietą.

Przykład:

```
public class P_continue {

    public static void main(String[] args) {
        etykieta:
        for(int i=0; i<10; i++)
        {
            for(int j=0; j<10; j++)
            {
                if (j>i)
                {
                    System.out.println();
                    continue etykieta;
                }
                System.out.print(" " + (i*j));
            }
            System.out.println();
        }
    }
}
```

Wynik działania programu



```
<terminated> P_continue [Java Application] C:\Program Files\Java
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81
```