Z tej lekcji nauczysz się :

-stosować pętlę WHILE

-stosować instrukcję CONTINUE

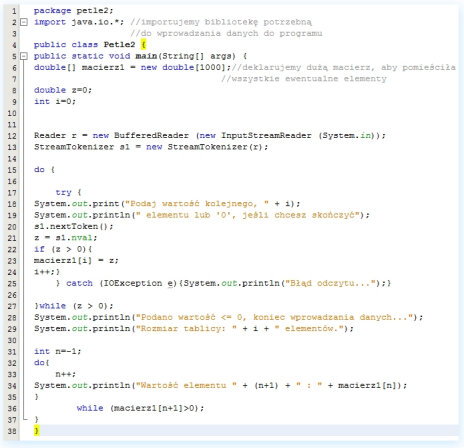
-stosować instrukcję BREAK

PĘTLA WHILE

Przyjrzyjmy się jeszcze raz programowi z poprzedniej lekcji. Przypomnijmy ten program:

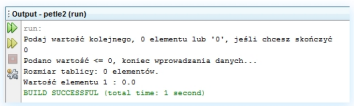
{ Napisz program, który tworzy macierz liczb całkowitych podanych przez użytkownika. O ilości elementów w macierzy (czyli jej rozmiarze) również decyduje użytkownik. Po zakończeniu wprowadzania danych przez użytkownika program wyświetla po kolei wszystkie wartości. }

Listing do programu:



```
package petle2;
import java.io.*; //importujemy bibliotekę potrzebną
                  //do wprowadzania danych do programu
public class Petle2 {
public static void main(String[] args) {
double[] macierz1 = new double[1000];//deklarujemy dużą macierz, aby pomieściła
                                     //wszystkie ewentualne elementy
double z=0;
int i=0;


Reader r = new BufferedReader (new InputStreamReader (System.in));
StreamTokenizer s1 = new StreamTokenizer(r);

do {

    try {
System.out.print("Podaj wartość kolejnego, " + i);
System.out.println(" elementu lub '0', jeśli chcesz skończyć");
s1.nextToken();
z = s1.nval;
if (z > 0){
macierz1[i] = z;
i++;}
    } catch (IOException e){System.out.println("Błąd odczytu...");}

}while (z > 0);
System.out.println("Podano wartość <= 0, koniec wprowadzania danych...");
System.out.println("Rozmiar tablicy: " + i + " elementów.");

int n=-1;
do{
    n++;
System.out.println("Wartość elementu " + (n+1) + " : " + macierz1[n]);
}
       while (macierz1[n+1]>0);
}
}
```

Przypomnijmy, że użytkownik, aby zakończyć wprowadzanie danych musiał podać wartość zero lub mniejszą niż zero., po czym program przechodził do wyświetlania pobranych od użytkownika danych. Jeśli użytkownik podał od razu wartość równą zero lub mniejszą niż zero, program kończył procedurę wprowadzania danych i rozpoczął wyświetlanie.. elementów tablicy pustej:



```
Output - petle2 (run)
run:
Podaj wartość kolejnego, 0 elementu lub '0', jeśli chcesz skończyć
0
Podano wartość <= 0, koniec wprowadzania danych...
Rozmiar tablicy: 0 elementów.
Wartość elementu 1 : 0.0
BUILD SUCCESSFUL (total time: 1 second)
```

Wyraźnie jest napisane, że tablica zawiera 0 elementów, natomiast wyświetlony jest element pierwszy {którego nie ma} równy zero. Skąd ten błąd i dlaczego tak się dzieje?

Odpowiedź jest prosta i kryje się w budowie pętli do..while. Jeszcze raz przypomnijmy sobie jej składnię i działanie:

do{

instrukcje

}while(warunek jest spełniony**);**

Zauważ, że program sprawdza, czy dany warunek jest spełniony dopiero po pierwszym przebiegu instrukcji. Czyli pętla wykona się przynajmniej jeden raz, zanim sprawdzony zostanie warunek, czy powinna się dalej wykonywać. Stąd problem z błędem, który przedstawiono na przykładzie powyżej. Dlaczego jednak program nie zawiesił się, tylko wyświetlił nieistniejący element pustej tablicy? Odpowiedź jest również bardzo prosta: tablica nie jest pusta, tylko zawiera 1000 elementów typu double, które zainicjowane są z wartością zero.

```
6  double[] macierz1 = new double[1000];//deklarujemy dużą macierz, aby pomieściła
7                                       //wszystkie ewentualne elementy
```

Oznacza to, że w momencie tworzenia tablicy zdefiniowano, że ma ona 1000 elementów i od razu przypisano jej wartość zero. Zrobiliśmy tak dlatego,że język Java nie zawiera prostego sposobu na zdefiniowanie tzw. tablic dynamicznych, czyli takich,których rozmiar może być zmieniany w miarę potrzeb. Możliwości takie mają inne języki,natomiast Java, głównie ze względu na kwestie stabilności działania programów w niej napisanych nie daje takiej opcji. Dlatego tworząc tablicę, w której nie wiemy, ile elementów będzie wykorzystanych stworzyliśmy ją bardzo dużą, aby mieć pewność, że nie będzie za mała. Również ze względu na specyfikę języka, zainicjowaliśmy tablicę wypełniając ją "zerami".

Przejdźmy teraz do rozwiązania naszego problemu. W jaki sposób zapobiec wyświetlaniu, w

naszym programie elementów, które nie istnieją?

Jedną z możliwości jest wykorzystanie innej pętli, takiej, która od razu sprawdza, czy w ogóle choć raz powinny się wykonać zawarte w niej instrukcje. Taką pętlą jest iteracja **while()**. Składnia pętli jest taka:

while (warunek jest spełniony**)**

{

instrukcje

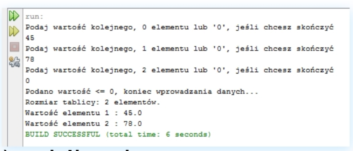
}

w porównaniu do pętli do..while różni się ona właśnie tym, że instrukcja sprawdzająca wystąpienie warunku wykonywania przesunięta jest na początek.

Oto, jak powinniśmy zmodyfikować nasz program, aby wszystko działało jak należy:

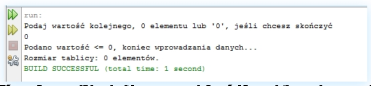
```
package petle2;
import java.io.*; //importujemy bibliotekę potrzebną
                  //do wprowadzania danych do programu
public class Petle2 {
public static void main(String[] args) {
double[] macierz1 = new double[1000];//deklarujemy dużą macierz, aby pomieściła
                                     //wszystkie ewentualne elementy
double z=0;
int i=0;


Reader r = new BufferedReader (new InputStreamReader (System.in));
StreamTokenizer s1 = new StreamTokenizer(r);

do {

    try {
System.out.print("Podaj wartość kolejnego, " + i);
System.out.println(" elementu lub '0', jeśli chcesz skończyć");
s1.nextToken();
z = s1.nval;
if (z > 0){
macierz1[i] = z;
i++;}
    } catch (IOException e){System.out.println("Błąd odczytu...");}

}while (z > 0);
System.out.println("Podano wartość <= 0, koniec wprowadzania danych...");
System.out.println("Rozmiar tablicy: " + i + " elementów.");

int n=0;
while(n<i)
{
    n++;
System.out.println("Wartość elementu " + (n) + " : " + macierz1[n-1]);
}
}
}
```

[illegible] programu dla dwóch wariantów:



```
run:
Podaj wartość kolejnego, 0 elementu lub '0', jeśli chcesz skończyć
45
Podaj wartość kolejnego, 1 elementu lub '0', jeśli chcesz skończyć
78
Podaj wartość kolejnego, 2 elementu lub '0', jeśli chcesz skończyć
0
Podano wartość <= 0, koniec wprowadzania danych...
Rozmiar tablicy: 2 elementów.
Wartość elementu 1 : 45.0
Wartość elementu 2 : 78.0
BUILD SUCCESSFUL (total time: 6 seconds)
```

b) tablica jest pusta



```
run:
Podaj wartość kolejnego, 0 elementu lub '0', jeśli chcesz skończyć
0
Podano wartość <= 0, koniec wprowadzania danych...
Rozmiar tablicy: 0 elementów.
BUILD SUCCESSFUL (total time: 1 second)
```

[illegible] lub

INSTRUKCJA BREAK:

Poznaliśmy trzy rodzaje iteracji:

a) pętla **for** (dla instrukcji powtarzalnych określoną ilość razy)

b) pętla **do..while** (która wykonuje się przynajmniej jeden raz oraz kolejne razy, dopóki warunek podany w nawiasie jest spełniony)

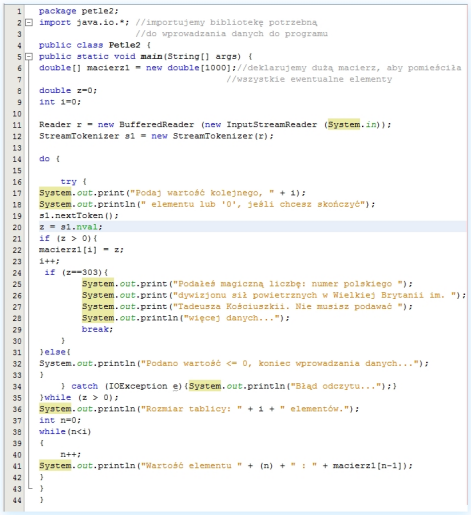
c) pętla **while** (która zaczyna się wykonywać dopiero, kiedy nastąpi sprawdzenie warunku podanego w nawiasie i trwa dopóki jest o spełniony)

Niekiedy jednak, niezależnie od tego, czy warunek działania pętli jest spełniony potrzebna jest

instrukcja przerywająca pętlę. Taką instrukcją jest właśnie BREAK. Zilustrujmy to na przykładzie:

Zadanie 2: Napisz program, który prosi użytkownika o podanie kolejnych liczb dodatnich, które później wyświetla. Wyjątek stanowi podanie przez użytkownika wartości '303', kiedy program przerywa etap podawania danych, wyświetla zdanie "Podałeś magiczną liczbę: numer polskiego dywizjonu sił powietrznych w Wielkiej Brytanii im. Tadeusza Kościuszkii. Nie musisz podawać więcej danych..."

Listing:



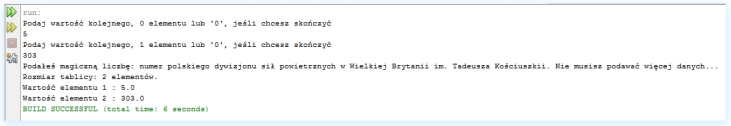
```
1  package petle2;
2  import java.io.*; //importujemy bibliotekę potrzebną
3                    //do wprowadzania danych do programu
4  public class Petle2 {
5  public static void main(String[] args) {
6  double[] macierz1 = new double[1000];//deklarujemy dużą macierz, aby pomieściła
7                                       //wszystkie ewentualne elementy
8  double z=0;
9  int i=0;
10
11 Reader r = new BufferedReader (new InputStreamReader (System.in));
12 StreamTokenizer s1 = new StreamTokenizer(r);
13
14 do {
15
16     try {
17 System.out.print("Podaj wartość kolejnego, " + i);
18 System.out.println(" elementu lub '0', jeśli chcesz skończyć");
19 s1.nextToken();
20 z = s1.nval;
21 if (z > 0){
22 macierz1[i] = z;
23 i++;
24  if (z==303){
25          System.out.print("Podałeś magiczną liczbę: numer polskiego ");
26          System.out.print("dywizjonu sił powietrznych w Wielkiej Brytanii im. ");
27          System.out.print("Tadeusza Kościuszkii. Nie musisz podawać ");
28          System.out.println("więcej danych...");
29          break;
30      }
31 }else{
32 System.out.println("Podano wartość <= 0, koniec wprowadzania danych...");
33 }
34      } catch (IOException e){System.out.println("Błąd odczytu...");}
35 }while (z > 0);
36 System.out.println("Rozmiar tablicy: " + i + " elementów.");
37 int n=0;
38 while(n<i)
39 {
40     n++;
41 System.out.println("Wartość elementu " + (n) + " : " + macierz1[n-1]);
42 }
43 }
44 }
```

W linijkach 24-30 dodaliśmy instrukcję warunkową IF. która sprawdza, czy użytkownik nie podał czasem wartości 303. Z powodów praktycznych długi komunikat rozbiliśmy na kilka instrukcji *System.out.print* (zamiast jednej *System.out.**println***) - aby zrzut z kodem programu nie był zbyt szeroki. Przy okazji przypomnieliśmy sobie o różnicy między tymi dwiema instrukcjami.

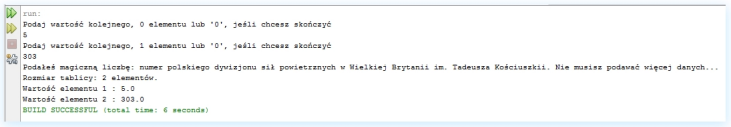
W linijce 29 zastosowaliśmy instrukcję BREAK, która kończy działanie pętli, w której się znajduje.

Zauważ, że instrukcja *else* z linijki 31 odnosi się do instrukcji *(if z>0)* z linijki 21. Wprowadziliśmy ją po to, aby program wyświetlał komunikat o podaniu wartości mniejszej niż zero tylko w przypadku podania takiej wartości, a nie po każdorazowym przerwaniu pętli (gdybyśmy tak nie zrobili, komunikat: "Podano wartość <= 0, koniec wprowadzania danych..." pojawiałby się także po podaniu wartości 303, co byłoby nielogiczne).

Wynik uruchomienia programu:



Dla pewności sprawdzamy, czy nadal program obsługuje "puste" macierze:



Zadanie do samodzielnego wykonania:

Zadanie 3 Zmodyfikuj powyższy program tak, aby, podobnie jak w przypadku podania wartości "303" wyświetlał adekwatne informacje po wprowadzeniu wartości 1410, 966, 1918 (odzyskanie niepodległości).

INSTRUKCJA CONTINUE

Instrukcja BREAK była specyficzną instrukcją, bo dawała nam możliwość ingerencji w wykonywanie pętli - konkretnie możliwość zatrzymania tej pętli.

Inną instrukcją, która umożliwia ingerencję jest instrukcja CONTINUE. Jej funkcja polega na zaprzestaniu wykonywania danego przebiegu pętli i przejściu do kolejnego przeibegu. Zilustrujmy to na przykładzie:

Zadanie 4 Zmodyfikuj powyższy program tak, aby podanie wartości '666' powodowało jej zignorowanie:

Listing:

```
package petle2;
import java.io.*; //importujemy bibliotekę potrzebną
                  //do wprowadzania danych do programu
public class Petle2 {
public static void main(String[] args) {
double[] macierz1 = new double[1000];//deklarujemy dużą macierz, aby pomieściła
                                     //wszystkie ewentualne elementy
double z=0;
int i=0;

Reader r = new BufferedReader (new InputStreamReader (System.in));
StreamTokenizer s1 = new StreamTokenizer(r);

do {

    try {
System.out.print("Podaj wartość kolejnego, " + i);
System.out.println(" elementu lub '0', jeśli chcesz skończyć");
s1.nextToken();
z = s1.nval;
if (z > 0){
    if (z==666) continue;
macierz1[i] = z;
i++;
 if (z==303){
        System.out.print("Podałeś magiczną liczbę: numer polskiego ");
        System.out.print("dywizjonu sił powietrznych w Wielkiej Brytanii im. ");
        System.out.print("Tadeusza Kościuszkii. Nie musisz podawać ");
        System.out.println("więcej danych...");
        break;
    }
}else{
System.out.println("Podano wartość <= 0, koniec wprowadzania danych...");
}
    } catch (IOException e){System.out.println("Błąd odczytu...");}
}while (z > 0);
System.out.println("Rozmiar tablicy: " + i + " elementów.");
int n=0;
while(n<i)
{
    n++;
System.out.println("Wartość elementu " + (n) + " : " + macierz1[n-1]);
}
}
}
```

```
Output - petle2 (run)
run:
Podaj wartość kolejnego, 0 elementu lub '0', jeśli chcesz skończyć
6
Podaj wartość kolejnego, 1 elementu lub '0', jeśli chcesz skończyć
7
Podaj wartość kolejnego, 2 elementu lub '0', jeśli chcesz skończyć
666
Podaj wartość kolejnego, 2 elementu lub '0', jeśli chcesz skończyć
3
Podaj wartość kolejnego, 3 elementu lub '0', jeśli chcesz skończyć
0
Podano wartość <= 0, koniec wprowadzania danych...
Rozmiar tablicy: 3 elementów.
Wartość elementu 1 : 6.0
Wartość elementu 2 : 7.0
Wartość elementu 3 : 3.0
BUILD SUCCESSFUL (total time: 15 seconds)
```