Z tej lekcji

dowiesz się:

- jak działa pętla do...while
- jak działa pętla while
- kiedy stosujemy w/w pętle

nauczysz się:

- stosować pętlę a do...while
- stosować pętlę while

Na ostatniej lekcji uczyliśmy się stosować pętlę for. Jej ogromne możliwości polegały na tym, że wykonywała ona zaprogramowany ciąg instrukcji określoną ilość razy, np:

for(int i=0; i<20; i++){

instrukcje do wykonania przy każdym przebiegu pętli

}

Powyższy przykład ilustruje pętlę, która wykona się równo 20 razy (dla i=0 aż do i=19).

Czasami jednak musimy użyć pętli gdy chcemy, aby jakieś instrukcje wykonywały się wiele razy, ale nie jesteśmy w stanie z góry przewidzieć ile razy.

Dzieje się tak najczęściej wtedy, kiedy nie od nas zależało będzie kiedy dana pętla ma się przestać wykonywać. Zilustrujemy to na poniższym przykładzie:

Zadanie 1: Napisz program, który tworzy macierz liczb całkowitych podanych przez użytkownika. O ilości elementów w macierzy (czyli jej rozmiarze) również decyduje użytkownik.

//tworzymy nowy projekt w programie NetBeans klikając kolejno: **File** >**New Project** > [Java | Java Application]
Next
> {wpisujemy nazwę pakietu: "
petle2
"}
Finish

Oto kod programu: (po znakach "//" występują komentarze)

```
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
12  Reader r = new BufferedReader (new InputStreamReader (System.in));
13  StreamTokenizer s1 = new StreamTokenizer(r);
14
15  do {
16
17      try {
18  System.out.print("Podaj wartość kolejnego, " + i);
19  System.out.println(" elementu lub '0', jeśli chcesz skończyć");
20  s1.nextToken();
21  z = s1.nval;
22  if (z > 0){
23  macierz1[i] = z;
24  i++;}
25      } catch (IOException e){System.out.println("Błąd odczytu...");}
26
27  }while (z > 0);
28  System.out.println("Podano wartość <= 0, koniec wprowadzania danych...");
29  System.out.println("Rozmiar tablicy: " + i + " elementów.");
30  }
31  }
```

Omówienia, poza komentarzami, wymaga też kilka nowych instrukcji:

W linijce 2 wpisaliśmy import java.io.* - najprościej rzecz biorąc chodzi o to, że podstawowy program w javie zawiera tylko najpotrzebniejsze możliwości (biblioteki), dlatego zajmuje mało miejsca i może być używany w różnych warunkach, nawet tam, gdzie wielkość pliku z programem musi być jak najmniejsza (internet, urządzenia przenośne o mocno ograniczonych zasobach). Możliwości języka Java są jednak olbrzymie i można je zwiększać dodając (importując) do kodu programu gotowe moduły (tzw. biblioteki). Np. pisząc program, w którym wykorzystujemy obsługę urządzeń Bluetooth importujemy specjalistyczne biblioteki, ale nie ma sensu, aby doczepiane były one do wszystkich programów. W ten sposób osiągnięto konsensus pomiędzy oszczędnością zasobów a funkcjonalnością.

W linijce 12 i 13 wpisaliśmy instrukcję, które umożliwią użytkownikowi wprowadzenie kolejnych liczb, a programowi poprawne odczytanie tych liczb. Na chwilę obecną nie zagłębiajmy się zbytnio w to, jak one działają, przyjmijmy, że tak będziemy postępować w takich przypadkach. Oczywiście nazwy tych obiektów (r, s1) zależą od nas.

W linijce 18 i 19 podzielono jeden komunikat komputera na 2 instrukcję, aby zrzut ekranu nie był zbyt szeroki i zmieścił się na portalu. (przy okazji przypomnieliśmy sobie różnicę między System.out.println a System.out.print)

W linijce 15 i 27 wpisaliśmy pętlę do. Jej konstrukcja wygląda tak:

do{

instrukcje;

} while (warunek jest spełniony**);**

Działą to tak: po każdorazowym wykonaniu instrukcji zawartych pomiędzy nawiasami klamrowymi {} program sprawdza, czy warunek jest nadal spełniony. Jeśli nie, pętla jest przerywana. Jeśli tak, wykonywanie pętli jest kontynuowane. U nas warunkiem jest wartość zmiennej 'z', którą podaje użytkownik: wartość większa od zera jest dopisywana do macierzy (staje się jej kolejnym elementem), natomiast wprowadzenie zera lub wartości mniejszej od zera interpretowane jest jako chęć zakończenia wpisywania danych (czyli przerwania pętli).

W linijce 20 następuje odczyt wartości wprowadzonej przez użytkownika.

W linijkach 22 do 24 użyto instrukcji warunkowej if, która po stwierdzeniu, że użytkownik podał wartość większą od zera dodaje ją do macierzy i zwiększa licznik i o 1.

Przypomnijmy składnię instrukcji if:

if (warunek jest spełniony**){**

instrukcje

}

W linijkach 17 i 25 użyto czegoś zupełnie nowego: instrukcji try .. catch. Co to jest i po co to wstawiliśmy? Jest to pewne zabezpieczenie programu. Zauważ, że try znaczy po angielsku "próbować" natomiast catch znaczy "łapać". Mówiąc najprościej instrukcja ta zapobiega sytuacjom, kiedy program zawiesza się na wskutek niepożądanych okoliczności. Jeśli przeanalizujemy to, co znajduje się wewnątrz instrukcji try..catch, szybko dojdziemy do wniosku, że głównie chodzi o wprowadzanie danych przez użytkownika. Wszystko działa dobrze, jeśli użytkownik wprowadza poprawne dane (liczby). Jeśli natomiast wprowadzone dane nie będą zrozumiałe jako liczby, program nie mógłby sobie z tym poradzić i najczęściej

zawiesiłby się. Można temu przeciwdziałać na 2 sposoby: przewidzieć wszystkie możliwe błędy użytkownika i zapisać w programie odpowiednie instrukcję obsługi tych błędów (mało skuteczne wyjście, znacznie komplikuje kod a i tak nie wszystko jesteśmy w stanie przewidzieć), albo użyć instrukcji try..catch, która przez komputer rozumiana jest mniej więcej tak: próbuj wykonywać instrukcje między komendami try oraz catch, a gdyby stało się coś niepożądanego to wykonuj instrukcje zawarte w nawiasie klamrowym po komendzie catch (u nas program wyświetla wtedy napis "Błąd odczytu..."). Przypomnijmy jeszcze raz składnię instrukcji try..catch:

try{

instrukcje

}catch (rodzaj wyłapywanych błędów**){**instrukcja na wypadek wystąpienia błędów**}**

W javie zachowanie stabilności działania programu jest szczególnie ważne choćby dlatego, że kod jest przenośny i nieraz nawet nie jesteśmy przewidzieć w jakich warunkach program będzie pracował. Na przykład w telefonie komórkowym zawieszenie aplikacji znacznie utrudniałoby używanie telefonu i dlatego musimy dążyć do wyeliminowania szansy wystąpienia sytuacji, w których program "nie wie co z tym fantem zrobić".

Po uruchomieniu programu przyciskiem:

otrzymujemy np. taki rezultat:

```
Output - petle2 (run)
run:
Podaj wartość kolejnego, 0 elementu lub '0', jeśli chcesz skończyć
1
Podaj wartość kolejnego, 1 elementu lub '0', jeśli chcesz skończyć
45
Podaj wartość kolejnego, 2 elementu lub '0', jeśli chcesz skończyć
0
Podano wartość <= 0, koniec wprowadzania danych...
Rozmiar tablicy: 2 elementów.
BUILD SUCCESSFUL (total time: 9 seconds)
```

```
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
12  Reader r = new BufferedReader (new InputStreamReader (System.in));
13  StreamTokenizer s1 = new StreamTokenizer(r);
14
15  do {
16
17      try {
18  System.out.print("Podaj wartość kolejnego, " + i);
19  System.out.println(" elementu lub '0', jeśli chcesz skończyć");
20  s1.nextToken();
21  z = s1.nval;
22  if (z > 0){
23  macierz1[i] = z;
24  i++;}
25      } catch (IOException e){System.out.println("Błąd odczytu...");}
26
27  }while (z > 0);
28  System.out.println("Podano wartość <= 0, koniec wprowadzania danych...");
29  System.out.println("Rozmiar tablicy: " + i + " elementów.");
30
31  int n=-1;
32  do{
33      n++;
34  System.out.println("Wartość elementu " + (n+1) + " : " + macierz1[n]);
35  }
36          while (macierz1[n+1]>0);
37  }
38  }
```

Przykładowy wynik kompilacji i uruchomienia programu:

```
Output - petle2 (run)
run:
Podaj wartość kolejnego, 0 elementu lub '0', jeśli chcesz skończyć
23.6
Podaj wartość kolejnego, 1 elementu lub '0', jeśli chcesz skończyć
12
Podaj wartość kolejnego, 2 elementu lub '0', jeśli chcesz skończyć
0
Podano wartość <= 0, koniec wprowadzania danych...
Rozmiar tablicy: 2 elementów.
Wartość elementu 1 : 23.6
Wartość elementu 2 : 12.0
BUILD SUCCESSFUL (total time: 9 seconds)
```

```
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)
```