Spis treści

Temat: Typy danych w Java^{TM}
Typy prymitywne. Tablice. Opakowania. Krótkie omówienie z punktu widzenia naszych zamiarów.

1 Podstawowe typy Java^{TM}

Podstawa: Arnold, Gosling;

Java^{TM} dysponuje bogatą kolekcją typów danych. Podstawowe typy proste pokazane są w tablicy.

typ	opis
`boolean`	true lub false
`char`	16 bitowe znaki Unicode 1.1
`byte`	8 bitowe liczby całkowite
`short`	16 bitowe liczby całkowite
`int`	32 bitowe liczby całkowite
`long`	64 bitowe liczby całkowite
`float`	32 bitowe liczby całkowite
`double`	54 bitowe liczby całkowite

Oprócz nich istnieją klasy opakowujące dla typów prostych, np. Float. O tym powiemy później.

Wiemy, że obieky w Java^{TM} są określonego typu. Typem obiektu jest klasa, której obiekt jest egzemplarzem, instancją. Obiekty tworzy się za pomocą operatora `new` w procesie konkretyzacji.

P r z y k ł a d. 1.

```
...
class Punkt {
  public double x, y;
}

Punkt lewyDolny = new Punkt();
Punkt prawyDolny = new Punkt();

lewyDolny.x = 0.0;
lewyDolny.y = 0.0;

Punkt prawyGorny.x = 50d;
Punkt prawyGorny.y = 100d;
...
```

Każdy obiekt klasy `Punkt` jest niepowtarzalny i posiada własną kopię pól `x, y`. Pola obiektów nazywamy też zmiennymi instancyjnymi (przykładowymi).

2 Tablice

Podstawa: Bruce Eckel, Thinking in Java^{TM}

2.1 Tablice jednowymiarowe

W celu zdefiniowania tablicy w Java^{TM} wypisujemy nazwą tablicy wraz z pustymi nawiasami kwadratowymi `[]`:

```
int[] a1;
```

Nawiasy można również postawić po nazwie tablicy:

```
int a1[];
```

W Java^{TM} można używać obu metod deklarowania tablic. Kompilator nie pozwala podać liczby elementów tablicy. Inicjalizacja elementów tablicy odbywa się w dowolnym miejscu programu. Można też podać zawartość tablicy w deklaracji. Np.

```
int[] a1 = {1, 2, 3, 4, 5, 6;}
```

W Java^{TM} można przypisac jedną tablicę drugiej:

```
int[] a2;
a2 = a1;
```

Jest to możliwe dlatego, że tak naprawdę mamy doczynienia z referencjami, odniesieniami do tablic. Ostatnie polecenie jest przypisaniem referencji.

```
// ---------------------------------------------------------------------
/*
Fizyka komputerowa, IV, 2001.  Java(TM).
Program #1.
A. Baran, IFiz UMCS, 2000.
http://tytan/umcs.lublin.pl/baran
*/
```

```
// ---------------------------------------------------------------------
//

public class Tablice {  //// Bruce Eckel
  public static void main(String[] args) {
    int[] a1 = {1, 2, 3, 4, 5};
    int[] a2;
    a2 = a1;
    for(int i = 0; i<a2.length; i++) {
      a2[i]++;
    }
    for(int i = 0; i<a1.length; i++) {
      System.out.println(
        "a1[" + i + "] = "  + a1[i]);
    }
  }
}
```

W powyższym przykładzie wystąpiła składowa `length`, która mierzy długość tablic. Maksymalnym indeksem jst, podobnie jak w `C++` lub `perlu`, `length - 1`. Warto wiedzieć, że Java^{TM} sprawdza zakresy wskaźników tablic.

Elementy tablic kreujemy w dowolnej chwili używajac operatora `new`.

```
// ---------------------------------------------------------------------
/*
Fizyka komputerowa, IV, 2001.  Java(TM).
Program #2.
A. Baran, IFiz UMCS, 2000.
http://tytan/umcs.lublin.pl/baran
*/
// ---------------------------------------------------------------------
//

//// Bruce Eckel; c04
import java.util.*;

public class NTablica {
  static Random rand = new Random();
  static int pRand(int mod) {
    return Math.abs(rand.nextInt()) % mod + 1;
  }
  public static void main(String[] args) {
    int[] a;
    a = new int[pRand(20)];
    System.out.println(
      "length of a = " + a.length);
    for(int i = 0; i<a.length; i++) {
      System.out.println(
```

```
        "a[" + i + "] = "  + a[i]);
    }
  }
}
```

Wynik działania programu mówi, że tablica kreowana jest w czasie wykonywania się programu (*ang.* at runtime). Dodatkowo widzimy, że elementy tablicy prymitywnego typu `int` są inicjowane jako zera.

Z a d a n i e 1.
Wykonać powyższy przykład uruchamiając kilkakrotnie otrzymany kod Java^{TM}

Inicjacji tablicy można dokonac też w jednym poleceniu

```
int[] a = new int[pRand(20)];
```

2.2 Tablice wielowymiarowe

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/*
Fizyka komputerowa, IV, 2001.  Java(TM).
Program #3.
A. Baran, IFiz UMCS, 2000.
http://tytan/umcs.lublin.pl/baran
*/
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//

//// Tablice wielowymiarowe
public class MultiTab {
  static void prt(String s) {
    System.out.println(s);
  }
  public static void main(String[] args) {
    int[][] a = {    //// inicjalizacja tablicy 2-wym
      { 1, 2, 3 },
      { 4, 5, 6 },
    };
    for(int i = 0; i < a.length; i++)
      for(int j = 0; j < a[i].length; j++)
        prt("a[" + i + "][" + j + "] = " + a[i][j]);
  }
}
```

Z a d a n i e 2.
Zdefiniuj tablicę 3-wymiarową. Wydrukuj elementy tablicy.

3 Opakowania typów prostych

Podstawa: Boone Java^{TM} udostępnia klasy opakowujące dla prymitywnych typów danych. Podstawowe typy opakowań to: `Integer`, `Long`, `Float`, `Double`, `Character`, `Boolean`. Są to odpowiedniki typów prostych. Jak łatwo zauważyć ich nazwy zaczynają sie dużą literą i brzmią tak samo jak nazwy typów prostych. Obiekty „opakowane"tworzymy podając reprezentowaną przez nie wartość, np.

```
Integer opakInt = new Integer(123);
Character opakChar = new Character('g');
```

Opakowania dostarczają wielu przydatnych metod, głównie, konwertujących dane. Przykładowo, zmiana obiektu `Integer` na `double` wygląda następująco:

```
double d = opakInt.doubleValue();
```

Aby dostac napis piszemy:

```
String tekst = opakInt.toString();
```

Jeśli zechcemy zamienic znaki wczytywane z klawiatury do postaci liczb zmiennoprzecinkowych `double`, wykonujemy następujące operacje:

1. tworzymy element typu `String`, zawierający wartość zapisaną w zmiennej typu `StringBuffer` (tutaj `sb`).
2. korzystamy z metody `valueOf()` klasy `Double` aby utworzyć nowy obiekt klasy `Double`.
3. metoda `doubleValue()` kończy postawione zadanie i w wyniku dostajemy liczbę.

Wygląda to tak:

```
double d = Double.valueOf(sb.toString()).doubleValue();
```

4 Napisy

Do reprezentowania napisów w Java^{TM} używa się klas `String` i StringBuffer. Obiekty klasy `String` dysponują wieloma użytecznymi metodami do opracowywania napisów. Bez zbędnych komentarzy podamy kilka publicznych metod tej klasy.

4.1 Klasa String

Metody `indexOf()` i `lastIndexOf()` podają wskaźnik, od którego zaczyna się znaleziona w napisie wartość, lub -1 jeśli wartość nie została znaleziona. Oto metody wyszukujące w przód.

```
indexOf(Char ch)
indexOf(Char ch, int start)
indexOf(String str)
indexOf(String str, int start)
```

Podobnie działają metody lastIndexOf() wypisane niżej z tym, że przeszukują napis do tyłu.

```
lastIndexOf(Char ch)
lastIndexOf(Char ch, int str)
lastIndexOf(String str)
lastIndexOf(String str, int start)
```

Z a d a n i e 3.
Napisać metodę, która zlicza wystąpienie wskazanego znaku w napisie.

Inne metody klasy `String`

```
public String replace(Char stary, Char nowy)
public String toLowerCase()
public String toUpperCase()
public String trim() -- tworzy obiekt bez początkowej
                        i końcowej spacji
public String concat(String inny)
```

W operacjach porównywania napisów nie należy używać operatora ==, gdyż porównuje on referencje do napisów, a nie napisy. Do porównywania napisów służy metoda `compareTo(String)`, która porównuje całe napisy oraz metoda `public boolean regionMatches(int start, String inny, int istart, int tyle)`, która daje wynik `true`, gdy wskazane fragmenty napisów pokrywają się. Porównanie rozpoczyna sie na pozycji `start` pierwszego napisu i na pozycji `ostart` napisu `inny`. Porównuje się `tyle` znaków. Z a d a n i e 4.

Napisz program wyszukiwania binarnego (patrz AG, str 166)

Metoda `public boolean regionMatches(boolean ignoruj, int start, String inny, int istart, int tyle)` robi to samo w przypadku gdy `ignoruj == false`, a w przypadku `ignoruj == true` ignoruje wielkość liter.

4.2 Konwersje

Konwersji napisów na wartości innych typów dokonuja metody:

```
boolean String.valueOf(boolean)
int String.valueOf(int)
long String.valueOf(long)
float String.valueOf(float)
double String.valueOf(double)
```

Obiekty różnych typów zamienimy na napis, korzystając z metod:

```
new Boolean(String).booleanValue()
Integer.parseInt(String, int base)
Long parseLong(String, int base)
new Float(String).floatValue()
new Double(String).doubleValue()
```

4.3 Napisy i tablice

Napisy można traktować jako tablice znaków i odwrotnie. Metoda `toCharArray()` zamienia obiekt `String` na tablicę znaków.

4.4 Klasa `StringBuffer`

Klasa `StringBuffer` (różna niż `String`) pozwala modyfikować napisy bez konieczności tworzenia wielu obiektów klasy `String` do przechowywania pośrednich wyników. Poniższa metoda (`replace`) dokonuje modyfikacji "w miejscu", używając metod klasy `StringBuffer`.

```
public static void
    replace(StringBuffer str, char from, char to) {
    // Arnold, Gosling
  for(int i=0; i<str.length; i++)
    if(str.charAt(i) == from)
       str.setCharAt(i,to);
}
```

Metoda `setLength(...)` skraca lub wydłuza napis w buforze. Metody `append(...)` i `insert(...)` zamieniaja swój argument na napis, który dołączony zostaje na końcu danego napisu lub wstawiony we wskazanym miejscu. Argumenty obu metod mogą być nasępujących typów: `Object, boolean`, `String, char[]`, `int, long, float, double`.

P r z y k ł a d. 2.

```
int i = 5;
StringBuffer buf = new StringBuffer();
buf.append("sqrt(").append(i).append(')');
buf.append("=").append(Math.sqrt(i));
System.out.println(buf.toString());
```

Metoda `toString()` zamienia obiekt `StringBuffer` na napis.

Wielkość bufora typu `StringBuffer` można ustawiać w chwili tworzenia obiektu konstruktorem `public StringBuffer(int capacity)`. Metoda `public synchronized void ensureCapacity(int minimum)` zapewnia, że w buforze będzie można zmieścić co najmniej `minimum` znaków. Metoda `public capacity()` informuje o pojemności bufora.

P r z y k ł a d. 3.
Przykład ilustruje sposób wczytywania danych z klawiatury i wypisywania wyników na terminal. Wykorzystano klasy strumieni wejścia//wyjścia, klasy opakowujące, klasę `StringBuffer`, itp. Zilustrowano obsługę wyjątków. (patrz B. Boone, str 65)

```
import java.io.DataInputStream; // do czytania wiersza znaków

class Astronaut {
  Double earthWeight;

  Astronaut (double weight) {  // konstruktor klasy Astronaut
    earthWeight = new Double(weight);
  }

  public double moonWeight() {
    return earthWeight.doubleValue() * 0.166;
```

```
  }
}

class PlanetaryScale {
  Astronaut armstrong;

  // waga
  void calculateWeight() {
    armstrong = new Astronaut(getEarthWeight());
    showMoonWeight(armstrong.moonWeight());
  }

  // pobranie wartości wpisanej przez użytkownika
  double getEarthWeight() {
    double earthWeight;
    DataInputStream stream = new DataInputStream(System.in);
    String strng;

    System.out.println("Podaj wagę ziemską.");
    // próbuj czytać wiersz zklawiatury

    try {
      strng = stream.readLine();
    } catch (java.io.IOException e) {
  strng = "0.0";
    }

    // przekształcanie do liczby
    try {
  earthWeight = Double.valueOf(strng).doubleValue();
    } catch (java.lang.NumberFormatException e) {
  earthWeight = 0.0;
    }

    return earthWeight;
  }

  // wyświetl wynik
  void showMoonWeight(double wt) {
    System.out.println("Na Księżycu ważysz " +
       String.valueOf(wt));
  }

  public static void main (String args[]) {
    PlanetaryScale ps = new PlanetaryScale();
    ps.calculateWeight();
  }

}
```