Język JAVA
podstawy programowania

Jacek Rumiński

Wykład 3, część 2

Plan wykładu:

1. Konstrukcja kodu programów w Javie
2. Identyfikatory, zmienne
3. Typy danych
4. Operatory, instrukcje sterujące – instrukcja warunkowe,
5. Instrukcje sterujące – pętle, instrukcje wyboru, instrukcje powrotu

Podstawowe typy danych

Właściwości:

1. Typy danych są niezależne od platformy sprzętowej – ten sam rozmiar w bajtach.
2. Wszystkie typy liczbowe są przechowywane za znakiem, np. byte: -128..0..127. Nie ma typów oznaczanych w innych językach jako *unsigned*, czyli bez znaku.
3. Wszystkie podstawowe typy danych są oznaczane z małych liter.
4. Nie istnieje w gronie podstawowych typów danych typ łańcucha znaków (ciągu znaków).
5. Klasa – to też typ danych.
6. Istnieją klasy jako odpowiedniki dla typów podstawowych (oznaczane z wielkich liter), umożliwiające konwersję i inne operacje wartości danego typu,
   np. (typ podstawowy) int – (klasa) Integer

Podstawowe typy danych

Typy liczbowe, całkowite:

byte : (1 bajt) typ liczbowy, jednobajtowy za znakiem. Wartości tego typu mieszczą się w przedziale: -128 do 127.

short : (2 bajty) typ liczbowy, dwubajtowy ze znakiem. Wartości tego typu mieszczą się w przedziale: -32,768 do 32,767

int : (4 bajty) typ liczbowy, czterobajtowy ze znakiem. Wartości tego typu mieszczą się w przedziale: -2,147,483,648 do 2,147,483,647.

long : (8 bajtów) typ liczbowy, ośmiobajtowy ze znakiem. Wartości tego typu mieszczą się w przedziale: -9,223,372,036,854,775,808 do +9,223,372,036,854,775,807.

Podstawowe typy danych

Typy liczbowe, liczb rzeczywiste, zmiennoprzecinkowe:

float *:* (4 bajty, spec. IEEE 754) typ liczb rzeczywistych, czterobajtowy ze znakiem (tzw. pojedyncza precyzja). Wartości tego typu mieszczą się w przedziale: 1.40129846432481707e-45 do 3.40282346638528860e+38 (dodatnie lub ujemne).

double *:* (8 bajtów spec. IEEE 754) typ liczb rzeczywistych, ośmiobajtowy ze znakiem (tzw. podwójna precyzja). Wartości tego typu mieszczą się w przedziale: 4.94065645841246544e-324d do 1.79769313486231570e+308d (dodatnie lub ujemne).

Podstawowe typy danych

Inne typy podstawowe:

boolean : (1 bit) typ jednobitowy oznaczający albo true albo false (a nie 1 czy 0). Oznaczanie wartości typów (jak wszystkie w Javie) jest ściśle związane z wielkością liter. Przykładowe oznaczenia TRUE czy False nie mają nic wspólnego z wartościami typu boolean.

void: typ nie jest reprezentowany przez żadną wartość, wskazuje, że dana metoda nic nie zwraca; albo poprawniej, że metoda zwraca nic.

char : (2 bajty), typ znakowy dwubajtowy, dodatni. Kod dwubajtowy umożliwia zapis wszelkich kodów w systemie Unicode, który jest standardem w Javie. Zakres wartości kodu to: 0 to 65,535.

Podstawowe typy danych

Zmienne typów podstawowych

Następujące wartości domyślne są przyjmowane dla zmiennych poszczególnych typów podstawowych:

boolean:	false
char:	‘\u0000’ (null)
byte:	(byte)0
short:	(short)0
int:	0
long:	0L
float:	0.0f
double:	0.0 (lub 0.0d)

Kod programu: AtrybutyJedi.java

```
public class AtrybutyJedi {
   //pola - zmienne obiektu - mają ustawiane wartości domyślne
   int liczbaMieczy;
   short wiek;
   char kodZnakowy;
   double dlugoscMiecza;
   public AtrybutyJedi() {    } //pusty konstruktor
   public static void main(String[] args) {
                        int poziomMocy=0; //zmienne lokalne muszą mieć przypisaną wartość
                        AtrybutyJedi aj = new AtrybutyJedi();
                        System.out.println("Liczba mieczy = "+aj.liczbaMieczy);
                        System.out.println("Wiek = "+aj.wiek);
                        /*operacja (int) oznacza zmianę typu - rzutowanie, wartość domyślna
                          znaku (null, kod "0") zostaje zamieniona na liczbę 0          */
                        System.out.println("Litera kodu = "+(int)aj.kodZnakowy);
                        System.out.println("Długość miecza = "+aj.dlugoscMiecza);
                        System.out.println("Poziom mocy = "+poziomMocy);
   }//koniec main()
}//koniec public class AtrybutyJedi
```

Klasy typów danych

W Javie w bibliotece podstawowej języka: *java.lang.** znajdują się następujące klasy typów danych:

Boolean*:* klasa umożliwiająca stworzenie obiektu przechowującego pole o wartości typu podstawowego boolean. Klasa ta daje liczne możliwości przetwarzania wartości typu boolean na inne (np. łańcuch znaków -> obiekt klasy String). Przykładowa funkcja statyczna: Boolean.valueof("yes"); zwraca wartość true.

Byte : umożliwia stworzenie obiektu przechowującego pole o wartości typu podstawowego byte. Różne metody, np. : intValue() - konwersja wartości typu byte na typ int; floatValue()- konwersja wartości typu byte na typ float i inne. Stałe tej klasy: MIN_VALUE oraz MAX_VALUE, umożliwiają pozyskanie rozmiaru danego typu:

Byte.MAX_VALUE – to wartość maksymalna danego typu.

Podobnie dla klas: Short , Integer, Long , Float , Double.

Klasy typów danych

Character *:* klasa umożliwiająca stworzenie obiektu przechowującego pole o wartości typu podstawowego char. Większość pól i metod tej klasy dotyczy obsługi standardowej strony kodowej platformy Javy czyli Unicodu. W Javie obsługiwana jest większość stron kodowych a liczne klasy i metody umożliwiają konwersje pomiędzy stronami kodowymi.

Znaki specjalne (znak ucieczki \ wskazuje, że kolejny znak ma specjalne znaczenie):
'\n' – wstaw nową linię,
'\t' – wstaw odstęp (tabulator)
'\"' – wstaw znak specjalny "
'\\" – wstaw ukośnik \.

Klasy typów danych

Klasa Character posiada szereg statycznych, przydatnych metod, m.in.:

- Character.isWhitespace(znak) -> zwraca "true" jeśli *znak* jest znakiem nowej linii, tabulacji, spacją, itp.
- Character.isLetter(znak) -> zwraca "true" jeśli *znak* jest literą (a-z)
- Character.isDigit(znak) -> zwraca "true" jeśli *znak* jest cyfrą (0-9)
- Character.isLetterOrDigit(znak) -> zwraca "true" jeśli *znak* jest literą lub cyfrą,
- Character.isUpperCase(znak) {analogicznie isLowerCase(znak)} -> zwraca "true" jeśli *znak* jest wielką {mała} literą
- Character.toUpperCase(znak) {analogicznie toLowerCase(znak)} -> zamienia *znak* na wielką {małą} literę
- Character.digit(znak,baza_systemu_liczbowego) -> zamienia *znak* na liczbę, Character.digit('F',16) daje wartość 15.

Kod programu: ZnakiJedi.java

```
public class ZnakJedi {
   public ZnakJedi() {    }

   public static void main(String[] args) {
      int wartosc=Character.digit('F',16);
      System.out.println("Wartość znaku \"F\" w kodzie szesnastkowym to: \n\t"+wartosc);

      System.out.print("\n");

      wartosc=Character.digit('2',10);
      System.out.println("Wartość znaku \"2\" zamienionego na liczbę to: \n\t"+wartosc);

      wartosc=(char)'2';
      System.out.println("Kod znaku \"2\" w danej stronie kodowej to to: \n\t"+wartosc);
      //Pierwsze 128 znaków kodu Unicode to znaki kodu ASCII

   }//koniec main()

}//koniec public class ZnakJedi
```

Klasy typów danych - String

Jak zapisać ciąg znaków (łańcuch znaków)? Specjalny typ danych String (klasa!)

Obiekty klasy String przechowują sekwencję (ciąg) znaków w Unicodzie. Dana wartość sekwencji obiektu nie może być zmieniona, co oznacza, że przypisując nową wartość tworzymy nowy obiekt !

```
String s = new String("Ala");  //lepiej unikać takiego zapisu dla k. String
```

lepiej

```
String s="Ala";
```

Dlaczego tak?

Żeby to wytłumaczyć muszę wprowadzić znaczenie literałów w Javie.

Klasy typów danych – String

Literały to oznaczenie w kodzie źródłowym wartości typu podstawowego, obiektu klasy String lub wartości nieokreślonej (=null).

Przykładowo 12 oznacza wartość typu int.

boolean - true lub false,

byte – np. (byte) 12;

short – np. (short) 12;

int – np. 12,

long – np. 12L,

float – np. 12.0f

double – np. 12.0

char – np. '2';

String – np. "Ala ma kota" (oznacza obiekt o wartości ciągu ...)

Klasy typów danych – String

Dlatego zapis String s="Ala"; oznacza, że:
-"Ala" – istnieje instancja klasy String (obiekt) o wartości ciągu "Ala",
- String s - tworzona jest referencja do tego obiektu.

Natomiast zapis String s = new String("Ala"); oznacza:
-new String() – utworzenie nowego obiektu (kopia wartości "Ala" z istniejącego już obiektu utworzonego dla literału "Ala").
-reszta jak wyżej.

Tylko dla klasy String można dokonać bezpośredniego przypisania wartości ="Ala", ze względu na interpretację literału (czyli zapis "Ala" to obiekt zawierający wartość ciągu znaków "Ala").

Język JAVA – typy danych

//Kod programu: MocJedi.java

```
public class Moc{
        public static void main(String args[]){
                String dobro = new String("Dobro - jasna strona mocy");
                String zlo =     new String("Dobro - jasna strona mocy");
                /*W celu porównania efektu wykomentować powyższe 2 linii i
                 usunąć komentarze poniżej. Skompilować i uruchomić ponownie*/
                //String dobro = "Dobro - jasna strona mocy";
                //String zlo =      "Dobro - jasna strona mocy";
                System.out.println("Ciemna strona mocy twierdzi:");
                //if - instrukcja warunkowa jeśli wyrażenie jest prawdziwe ...inaczej ...
                if (zlo == dobro){
                        System.out.println("1. Moc to jedno");
                } else
                        System.out.println("1. Dwie moce?");
                //equals() funkcja sprawdza równość wartości w obiektach
                if (zlo.equals(dobro)){
                        System.out.println("2. Moc to jedno");
                } else
                        System.out.println("2. Dwie moce?");
        }//koniec main()
}// koniec public class Moc
```

Klasy typów danych – String

Względem obiektów klasy String można używać operatora dodawania, po to, aby utworzyć nowy obiekt String o wartości ciągu znaków będącym złożeniem dodawanych ciągów, np.:

```
String napis = "To" + " jest"+" napis."; // "To jest napis."
```

W takim dodatku dodawane są 3 obiekty, ale ponieważ nie można nadpisać wartości obiektu to teoretycznie (w starszych JDK) takie złożenie generuje dużo obiektów:

3 obiekty: "To", " jest", " napis."

1 nowy jako złożenie "To jest".

1 nowy jako złożenie "To jest napis".

Dzisiaj dodawanie obiektów String będzie miało następującą realizację

```
StringBufffer t = new StringBuffer("To").append(" jest").apend(" napis.");
String napis = t.toString(); //zamień zbuforowane ciągi na obiekt k. String
```

StringBuffer to klasa umożliwiająca rozbudowywanie ciągów bez tworzenia wielu obiektów!

```
//Kod programu: FunkcjeJedi.java
public class FunkcjeJedi {
   //static - mogę użyć tej zmiennej klasy (a nie obiektu) bez tworzenia obiektu
  static String s = "   Rycerz Luke ma niebieski miecz.";

  public static void main(String[] args) {
    System.out.printf("Liczbę znaków zwraca funkcja length() = %s \n",s.length());
    System.out.printf("Znaki na wielkie, toUpperCase() = %s \n",s.toUpperCase());
    System.out.printf("Znaki na małe, toLowerCase() = %s    \n",s.toLowerCase());
    System.out.printf("Usunięcia znaków pustych, trim() = %s \n",s.trim());

   System.out.printf("Pobranie znaku na pozycji,  charAt(int pozycja) = %c \n",s.charAt(5));
   //numeracja pozycji w ciągu lub tablicy rozpoczyna się od 0, stad s.charAt(5)='c'
   System.out.printf("Wyciąć podciąg,substring(int start,int koniec) = %s \n",s.substring(3,9));
   System.out.printf("Pozycję podciągu, indexOf(String ciag) = %s \n",s.indexOf("Rycerz"));
   System.out.printf("Zamienić podciąg, replace(String stary, String nowy) = %s
                      \n",s.replace("Rycerz", "Senator"));
  }//koniec main()

}//koniec public class FunkcjeJedi
```

Klasy typów danych – String – porównywanie ciągów "s" i "r"

1. s.equals(r) – omówione wcześniej (boolean: true lub false)
2. s.compareTo(r) - zwraca wartość liczbową typu int (-1,0,1)

   String s = "Luke";

   s.compareTo("Vader") → -1 ("Luke" < "Vader")

   s.compareTo("Luke") → 0 ("Luke" == "Luke")

   s.compareTo("Anakin") → +1 ("Luke" > "Anakin")

3. s.matches("wzorzec") – zwraca "true" jeśli ciąg zgodny z wzorcem:

   wzorzec = wyrażenie regularne (regular expression), np.:

   [a-zA-Z] – dowolna 1 litera

   [a-zA-Z]* – wiele dowolnych litera

   [a-zA-Z]{2} – dwie dowolne litery

   [a-z]{2,} – co najmniej dwie małe litery

   \d – cyfra (0-9), w zapisie w Javie "\\d", bo '\' to znak specjalny,

   \d{2}-\d{3} – dwie cyfry, myślnik, trzy cyfry (kod pocztowy)

//Kod programu: PorownajJedi.java

```
public class PorownajJedi {

   public static void main(String[] args) {
            String s = "Luke";
            String kod = "80-952";
            if(s.compareTo("Vader")<0)
                        System.out.println(s+" mniejszy od Vadera");
            if(s.compareTo("Luke")==0)
                        System.out.println(s+" równy Lukowi");
            if(s.compareTo("Anakin")>0)
                        System.out.println(s+" większy od Anakina");

            if(s.matches("[a-zA-Z]*"))
                        System.out.println(s+ " pasuje do wzorca");
            if(kod.matches("\\d{2}-\\d{3}[a-zA-Z]*"))
                        System.out.println(kod+ " pasuje do wzorca");

   }//koniec main()
}//koniec public class PorownajJedi
```

Klasy typów danych – inne

Object - klasa ta jest klasą nadrzędną wszystkich klas w Javie, tak więc tworzenie własnych typów danych będących klasami jest odwołaniem się do obiektu klasy Object. Każdy obiekt jest zawsze również typu Object (typ uniwersalny !).

Void - klasa, odpowiednik typu void. Przechowuje referencje do obiektu klasy Class reprezentującego typ podstawowy void (). Nie będziemy używać !

Class – reprezentuje klasy wykonywanych (wczytanych) aplikacji:
Typowe zastosowanie:
wczytaj klasę:

```
Class.forName("Rycerz"); //Rycerz to klasa z pliku Rycerz.class
```

pobierz nazwę klasy dla danego obiektu obj:

```
obj.getClass().getName(); //getName() to metoda klasy Class
```

Plan wykładu:

1. Konstrukcja kodu programów w Javie
2. Identyfikatory, zmienne
3. Typy danych
4. Operatory, instrukcje sterujące – instrukcja warunkowe
5. Instrukcje sterujące – pętle, instrukcje wyboru, instrukcje powrotu

Co to są operatory?

Operatory to elementy języka służące do generacji nowych wartości na podstawie podanych argumentów (jeden lub więcej). Operator wiąże się więc najczęściej z określonym działaniem na zmiennych. Prawie wszystkie operatory (z wyjątkiem: '=', '==', '!=', '+', '+=') działają na podstawowych typach danych, a nie na obiektach.
Wyróżnia się następujące klasy operatorów podane wedle ich kolejności wykonywania:

- ***operatory negacji,***
- ***operatory matematyczne,***
- ***operatory przesunięcia,***
- ***operatory relacji,***
- ***operatory logiczne i bitowe,***
- ***operatory warunkowe,***
- ***operatory przypisania.***

Operatory negacji i matematyczne

Operator negacji powoduje zmianę wartości zmiennej na przeciwną pod względem znaku, np.: int a =4; x = -a; (to x jest równe -4), itd.

Operatory matematyczne to takie operatory, które służą do wykonywania operacji matematycznych na argumentach. Do operacji matematycznych zalicza się:
mnożenie ‘*’;
dzielenie ‘/’;
modulo - reszta z dzielenia ‘%’,
dodawanie ‘+’,
odejmowanie ‘-‘.

Operatory matematyczne

W wyniku dzielenia liczba całkowitych **Java** nie zaokrągla wyników do najbliższej wartości całkowitej, lecz **obcina** powstałą **liczbę do liczby całkowitej**. Dodatkowym elementem wykonywania operacji matematycznych w Javie (podobnie jak i w C) jest skrócony zapis operacji matematycznych jeśli jest wykonywana operacja na zmiennej, która przechowuje zarazem wynik tej operacji. Wówczas możliwe są następujące skrócone zapisy operacji:

zwiększanie / zmniejszanie o 1 wartości zmiennej:

zapis klasyczny, np.: x = x+1; x= x-1;

zapis skrócony, np.: x++, x--.

operacja na zmiennej:

zapis klasyczny, np.: x = x+4; x= x*6; x= x/9;

zapis skrócony, np.: x+=4; x*=6; x/=9;

Operatory matematyczne

Zwiększanie lub zmniejszanie wartości zmiennej o 1 możliwe jest na dwa sposoby:

- zwiększanie/zmniejszanie przed operacją (najpierw zmniejsz/zwiększ wartość zmiennej, później wykonaj operację na tej zmiennej), wówczas notacja operacji jest następująca np.: `--x; ++x;`

- zwiększanie/zmniejszanie po operacji (najpierw wykonaj operację na tej zmiennej a później zmniejsz/zwiększ wartość zmiennej), wówczas notacja operacji jest następująca np.: `x--; x++;`

Operatory przesunięcia

Operatory przesunięcia działają na bitach w ich reprezentacji poprzez całkowite typy podstawowe danych. Operator „<<” powoduje przesunięcie w lewo o zadaną liczbę bitów, natomiast operator „>>” powoduje przesunięcie w prawo o zadaną liczbę bitów, np.:

```
int liczba = 7; //bitowo 111
int liczbaL =  liczba << 2;  //czyli 11100
int liczbaR = liczba>>2;    //czyli 1
```

Przykładowym zastosowaniem przesunięcia bitowego jest dzielenie przez 2, które można zrealizować poprzez przesunięcie bitowe w prawo o 1: **(8>>1) == (8/2)**.

Operatory relacji

Operatory relacji generują określony rezultat reprezentowany przez typ logiczny boolean w wyniku przeprowadzenia porównania:

‘a > b‘ - a większe od b,
‘a < b’ - a mniejsze od b,
‘a >= b‘ - a większe równe jak b,
‘a < =b’ - a mniejsze równe jak b,
‘a == b‘ - a identyczne z b,
‘a != b’ - a różne od b (! – negacja: nieprawda, że =).

Operatory logiczne i bitowe

Operatory logiczne również generują rezultat reprezentowany przez typ logiczny boolean. Rezultat ten jest tworzony w wyniku działania operacji:

'a **&&** b' - a i b (rezultatem jest true jeśli a i b są true);
'a **||** b ' - a lub b (rezultatem jest true jeśli a lub b jest true).

Operatory bitowe działają podobnie lecz operują na bitach. Ich zapis jest następujący (1001b – oznacza zapis binarny):
'**&**' - bitowy AND, np. 10&9 -> 1010b AND 1001b = 1000b=8
'**|**' - bitowy OR, np. 10|9 -> 1010b OR 1001b = 1011b=11
'**^**' - bitowy XOR, np. 10^9 -> 1010b XOR 1001b = 0011b=3

Operator warunkowy

Operator warunkowy w Javie jest skróconą wersją instrukcji warunkowej *if*:

```
if(wyrażenie_logiczne)
        instrukcje1; //jeżeli wyrażenie_logiczne jest true
else
        instrukcje2; //jeżeli wyrażenie_logiczne jest true
```

Operator warunkowy:

```
wyrażenie_logiczne ? instrukcje1 : instrukcje2;
```

instrukcje1 oznacza działanie, gdy wynik wyrażenia jest *true*;
instrukcje2 oznacza działanie, gdy wynik wyrażenia jest *false*.

DEMO ->

Język JAVA – operatory, instrukcja warunkowa

Kod programu: Relacje.java

```
public class Relacje{

        public static void main(String args[]){

                System.out.println("Kto to Vader?");
                String vader = "Vader";
                String anakin = "Anakin";
                String s;
                //użycie if
                if (vader.equals(anakin))
                        s=vader;
                else
                        s=anakin;
                System.out.println(s);
                //użycie operatora warunkowego
                s = (vader.equals(anakin)) ?  vader  :  anakin;
                System.out.println(s);

        }//koniec main()
}// koniec public class Relacje
```

Plan wykładu:

1. Konstrukcja kodu programów w Javie
2. Identyfikatory, zmienne
3. Typy danych
4. Operatory, instrukcje sterujące – instrukcja warunkowe,
5. Instrukcje sterujące – pętle, instrukcje wyboru, instrukcje powrotu – wykład 3, część 3