Wprowadzenie do Javy dla programistów C++

J. Apelski G. Chimosz D. Górczyńska E. Luty

Wydział Matematyki, Informatyki i Mechaniki

20 listopada 2007

Wprowadzenie

Wprowadzenie

Java różni się od C++:

- narzucono ograniczenia, które ułatwiają testowanie programów i czynią kod przejrzystym,
- usunięto wskaźniki i uproszczono model zarządzania zasobami, np. pamięcią operacyjną,
- usunięto niektóre konstrukcje języka C++ (np.: typedef, #define, goto, struct, union) oraz preprocesor, jako elementy niezgodne z paradygmatem programowania obiektowego oraz utrudniające zrozumienie i modyfikowanie kodu; można je zastąpić poprzez definicje klas,
- usunięto możliwość przeciążania operatorów,
- usunięto wielodziedziczenie (zastępując je mechanizmem implementacji interfejsów),
- uwolniono programistę od konieczności przydzielania i zwalniania pamięci,
- „zlikwidowano" możliwość definiowania procedur i funkcji nie związanych z definicją żadnej klasy.

Źródła wiedzy nt. Javy

http://java.sun.com/developer/onlineTraining/

- Dokumentacja na stronach Suna (http://java.sun.com/docs/books/tutorial/)
- Tutoriale, wiki (http://arturt.republika.pl/java/index.htm)
- Ważniak (http://wazniak.mimuw.edu.pl/index.php?title=PO_Wst%C4%99p_do_Javy)
- Przedmioty monograficzne na wydziale
- Książki (http://helion.pl/online/thinking/index.html#tij2)

Typy danych

- boolean (8 bitów)
- int (32 bity ze znakiem)
- long (64 bity ze znakiem)
- char (16 bitów bez znaku)
- byte (8 bitów, l. całkowita)
- short (16 bitów, l. całkowita)
- float (32 bity, l. zmiennoprzecinkowa)
- double (64 bity, l. zmiennoprzecinkowa)
- void dla null
- typy referencyjne:
  - typy klas
  - typy interfejsów
  - typy tablicowe.

Alfabet

- Unicode , UTF-16
- np. Kłapuś = "K\u0142ap\u015B"
- Nazwy zmiennych mogą się zaczynać od _ , $ lub dowolnej litery alfabetu
- Ważna wielkość liter

i jeszcze a'propos zmiennych:

- zerowane
- lub w przypadku boolean – *false*
- deklaracja, jak w C

i a'propos komentarzy:

- //do końca wiersza
- /* przez kilka wierszy */

Literały

- liczba całkowita ze znakiem - literał typu int lub long (wymuszenie, przez dodanie *l* na końcu)
- liczby większe od 2.147.483.647 , automatycznie long
- rozpoczynające się znakiem '0' są liczbami ósemkowymi
- '0x' lub '0X' stanowią liczby szesnastkowe (wielkość liter nieważna)
- literały rzeczywiste, domyślnie double (wymuszenie float przez *f* na końcu)
- boolean: *true* i *false*
- znakowe np. 'a'
- typu string "Dysia ma kotka"
- literały do znaków specjalnych np. \n, \t jak w C++
- literał *null*

Priorytety operatorów

- x.y f(x) a[x] x++ x−
- + - ! ++x −x (T)x
- * / %
- + - (binarne)
- << >> >>>
- < > <= >= instanceof
- == !=
- &
- ∧
- |
- &&
- ||
- ? :
- = *= /= %= += -= <<= >>= >>>= &= ∧= |=

Jeszcze o operatorach

- leniwe (pojedyncze) i gorliwe (podwójne)
- >>> - przesunięcie bitowe w prawo bez powielania najbardziej znaczącego bitu
- konwersje:
  - char -> int
  - byte -> short -> int -> long -> float -> double
  - boolean nie konwertuje się na int
- rzutowanie : (typ) wartość

Instrukcje C++-identyczne

- pusta ;
- deklaracja
- etykieta
- o.wypisz(i);
- warunkowa
- while (i>0) i–;
- for w wersji podstawowej
- return
- blok
- throw new Wyjątek();

Instrukcje C++-podobne

- deklaracja tablic

```
1 switch (i){
2   default: System.out.println("Wartosc spoza zakresu");
        break;  // Tak, nie musi byc ostatnia!
3   case 1: case 2: System.out.println("Dodatnie"); break;
4   case -1: case -2: System.out.println("Ujemne"); break;
5   case 0: System.out.println("Zero"); break; }
```

- break i continue (bo mogą być z etykietami, a przez to wyskoczyć kilka poziomów)

Instrukcje inne

- „for each”

```
1 int[] tab = new int[10];
2 for(int elt: tab)
3   System.out.print(elt+", ");
```

- assert

```
1 assert i>0;
2 assert i>=0: "i (" + i + ") mniejsze od zera";
```

- try
- do i++ while (i < 0);

Obiektowość

Klasy

- Każda klasa w oddzielnym pliku (dobra praktyka)
- Automatyczne odśmiecanie pamięci – nie ma destruktorów, jest tylko finalize()

```
1  [ abstract | final ] class Nazwa extends Nadklasa
2      implements Interfejs1, I2 {
3  [ public | protected | private ]
4      [ static | final | transient | volatile ]
5      typ nazwa;
6
7  [ public | protected | private ] [ static | final ]
8      [ void | typ ] Nazwa(Argumenty) {
9        /* tresc metody */
10       this.nazwa;
11       super.nazwa;
12     }
13 };
```

Klasy (c.d.)

```
1   class HelloWorld {
2     public static void main(String args[]) {
3       System.out.println("Hello world");
4     }
5   } ;
6
7   /* ... */
8
9   HelloWorld hw; // null
10  hw = new HelloWorld();
11  HelloWorld hw2 = hw; // ten sam obiekt!
12  hw = null; // obiekt nadal istnieje
```

Metody

```
1   class HelloWorld {
2     public HelloWorld() { /* konstruktor */ };
3     int call_me() {
4       System.out.println("Who is speaking?");
5       return 0;
6     }
7     protected void finalize() {};
8   } ;
9
10  /* ... */
11
12  HelloWorld hw = new HelloWorld();
13  hw.call_me();
```

Widoczność zmiennych i metod

Modyfikatory dostępu:

- `public` dostęp dla wszystkich
- `private` dostęp dla swojej klasy
- `protected` dostęp dla pakietu i podklas
- *domyślnie* dostęp dla wszystkich z pakietu

Metody klasowe:

- `static` można korzystać tylko z innych statycznych metod i zmiennych, zabronione użycie `this` oraz `super`

```
1  class HelloWorld {
2    static int i = 1;
3    static { i = 3; /* przy ladowaniu klasy */ };
4  };
```

Dziedziczenie

```
1   class HelloWorld {
2     int i;
3     private int tajne;
4     int give_var() { return i; };
5   } ;
6
7   class SuperHello extends HelloWorld {
8     int j;
9     int give_var() { return i * j; };
10    // nie mamy dostepu do "tajne"
11    void hello() {};
12  }
13
14  /* ... */
15
16  HelloWorld shw = new SuperHello();
17  // shw.hello(); jest niepoprawne
```

Dziedziczenie (c.d.)

```
1   class HelloWorld {
2     int i;
3     HelloWorld(int x) { i = x; };
4     void zero() { i = 0; };
5   };
6
7   class SuperHello extends HelloWorld {
8     int j;
9     SuperHello() { super(3); j = 5; super.zero() };
10  };
```

`final`

Znaczenia słowa kluczowego `final`:

- „stałe” w klasach
- metoda, której nie można przesłonić
- klasa, po której nie można dziedziczyć

Interfejsy

- Klasa bez implementacji
- Bez danych
- Można dziedziczyć
- Klasa może implementować kilka interfejsów

```
1  interface Nazwa {
2    void zrobA();
3    void zrobB();
4  };
5
6  class Klasa implements Nazwa {
7    void zrobA() { ; };
8    void zrobB() { ; };
9    int zmienna;
10   int zm() { return zmienna };
11 };
```

Pakiety

- Sposób na grupowanie klas
- Organizacja w katalogach – $CLASSPATH

```
1  package HelloPack; // w pierwszej linijce pliku!
2
3  import java.util.Stack;
4  import java.awt.*; // wolniejsza (tylko) kompilacja
5
6  class HelloWorld { };
7
8  class SuperHello extends HelloWorld { };
9
10 /* ... */
```

Możliwości

Wyjątki

- Mechanizm obsługi błędów wbudowany w język
- Wyjątki - obiekty
- Możliwość przerwania ciągu zagnieżdżonych wywołań (instrukcja throw)
- Możliwość zaznaczania wyjątków, które mają być zgłoszone na zewnątrz (klauzula throws)
- Nieobsłużone wyjątki - przerwanie działania programu
- Możliwość tworzenia własnych podklas wyjątków

Klasy wyjątków

- Klasa Throwable
- Podklasy:
  - Error - poważne błędy niezwiązane z działaniem aplikacji
  - Exception - błędy, na które aplikacja powinna reagować
    - własne wyjątki tworzy się jako podklasy tej klasy
    - wyróżniona podklasa RuntimeException - typowe błędy powstające podczas działania maszyny wirtualnej

Instrukcja try...catch

- Blok do obsługi wyjątków
- Blok `finally` opcjonalny, wykonywany niezależnie od nastąpienia wyjątku
- Schemat bloku:

```
1  try{
2     /* blok instrukcji mogacych spowodowac wyjatek */
3  }
4  catch(TypWyjatku1 idWyjatku1){
5     /* obsluga wyjatku 1 */
6  }
7  catch(TypWyjatkuN idWyjatkuN){
8     /* obsluga wyjatku N */
9  }
10 finally {
11    /* instrukcje */ // wykonuja sie zawsze
12 }
```

Instrukcja throws

- Wyrzucenie wyjątku poza metodę, w której wystąpił

```
1  SpecyfikatorDostepu typWyniku nazwaMetody (parametry)
2  throws TypWyjatku {
3     /* cialo metody */
4  };
```

Operacje wejścia-wyjścia

- Operacje wejścia-wyjścia opierają się na strumieniach
- Standardowo zdefiniowane strumienie:
  - `System.in` - strumień wejściowy
  - `System.out` - strumień wyjściowy
  - `System.err` - strumień związany z obsługą błędów
- Wypisanie danych na ekran - metoda println:
  `System.out.println("Hello")`

Wczytanie danych z klawiatury

- Potrzebne klasy:
  - `InputStream` - jej instancją jest System.in
  - `InputStreamReader`
  - `BufferedReader`
- Metoda readLine() klasy BufferedReader

```
1 BufferedReader inbr;
2 InputStreamReader inp;
3 inp = new InputStreamReader(System.in);
4 inbr = new BufferedReader(inp);
5
6 String line = inbr.readLine(); // czytanie linii tekstu z
      klawiatury
```

- Można zrezygnować z tworzenia zmiennej referencyjnej inp:

```
1 BufferedReader inbr = new BufferedReader(new
      InputStreamReader(System.in));
2 String line = inbr.readLine(); // czytanie linii tekstu z
      klawiatury
```

Wczytanie liczb z klawiatury

- Potrzebne klasy:
  - `StreamTokenizer`
- metoda `nextToken()` - pobiera kolejną jednostkę leksykalną
- pole `nval` - jeśli obecna jednostka leksykalna jest liczbą, zawiera jej wartość, w przeciwnym przypadku zawiera 0

```
1 int a;
2 Reader r = new BufferedReader(new InputStreamReader(System.
      in));
3 StreamTokenizer inp = new StreamTokenizer(r);
4 inp.nextToken();
5 a = inp.nval;
```

Operacje na plikach

- Potrzebne klasy:
  - `File` - opis pliku
  - `FileInputStream` - do strumieniowych operacji na plikach
  - `FileOutputStream` - do strumieniowych operacji na plikach
  - `BufferedReader`
  - `BufferedWriter`
  - `DataInputStream`
  - `DataOutputStream`

Pisanie do pliku

```
1 String line;
2 FileOutputStream fout = null;
3 File file = newFile("text.txt");
4 fout = new FileOutputStream(file);
5 DataOutputStream out = new DataOutputStream(fout);
6 BufferedReader inbr = new BufferedReader(new
      InputStreamReader(System.in));
7 line = inbr.readLine();
8 fout.writeBytes(line + '\n');
```

Czytanie z pliku

```
1 String line;
2 FileInputStream fin = null;
3 File file = newFile("text.txt");
4 fin = new FileInputStream(file);
5 DataInputStream out = new DataINputStream(fin);
6 BufferedReader inbr = new BufferedReader(new
      InputStreamReader(fin));
7 line = inbr.readLine();
8 System.out.println(line);
```

Typy sparametryzowane

`http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf`

- Typy umożliwiające tworzenie klas, w których typ danych jest parametrem
- Ułatwiają implementację algorytmów działających identycznie niezależnie od typu stosowanych danych
- Ułatwiają tworzenie bezpiecznego, wielokrotnie używanego kodu
- Rozwiązują problem bezpieczeństwa typów
- Różnią się od szablonów z C++ i od typów sparametryzowanych z C#
- Rodzaje typów sparametryzowanych:
  - Klasy sparametryzowane
  - Metody sparametryzowane
  - Interfejsy sparametryzowane

Klasy sparametryzowane

- Składnia klasy sparametryzowanej:

```
class nazwa-klasy <lista-parametrow-typow>{
  //definicja klasy
};
```

- Przykład prostej klasy sparametryzowanej:

```
class MojaKlasa<T>{
  T ob; // deklaracja obiektu typu T

  MojaKlasa(T o){
    ob = o;
  }
  T getob(){
    return ob;
  }
}
```

Typy ograniczone

- Mechanizm zapewnienia, że podawane typy będą wyłącznie podaną klasą bazową lub jej podklasami
- Składnia: `<T extends klasa-bazowa>`
- Przykład zastosowania:

```
1 class MojaKlasaNumeryczna<T extends Number>{
2   T[] nums;
3   MojaKlasaNumeryczna(T[] o){
4     nums = o;
5   }
6   double avg(){
7     double sum = 0.0;
8     for(int i=0; i < nums.length; i++)
9     sum += nums[i].doubleValue(); // metoda klasy Number
10    return sum / nums.length;
11  }
12 }
```

Argumenty wieloznaczne

- Umożliwiają tworzenie metod korzystających z argumentów różnych typów
- Przykład zastosowania:

```
1 boolean sameAvg(MojaKlasaNumeryczna<?> ob){
2   if (avg() == ob.avg() )
3     return true;
4   return false;
5 }
```

Kolekcje

`http://java.sun.com/javase/6/docs/api/java/util/package-summary.html`

- Szkielet Collections Framework - jeden z najbardziej użytecznych podsystemów Javy
- Podobieństwa między kolekcjami Javy a STL z C++
- Konkretne klasy zapewniają różne implementacje standardowych interfejsów
- Kolekcje w pakiecie java.util - klasy i interfejsy pochodzące z różnych epok rozwoju języka
- Interfejsy kolekcji:
  - Collection - praca na grupach obiektów
  - List - obsługa sekwencji
  - Queue - obsługa list, w ktorych elementy usuwa się tylko z początku listy
  - Set - obsługa zbiorów zawierających unikatowe elementy
  - SortedList - Set z obsługą posortowanych zbiorów
- Przykłady klas wbudowanych w pakiecie java.util:
  - Arrays - zawiera metody przydatne w trakcie korzystania z tablic
  - Vector, Stack - klasy starsze, choć jeszcze używane

Klasa Arrays

- Pozwala wykonać typowe operacje na tablicach
- Niektóre metody klasy Arrays:
  - `equals(arr1, arr2)` - porównuje dwie tablice
  - `fill(arr, value)` - wypełnia tablicę zadanym elementem
  - `sort(arr)` - sortuje rosnąco elementy tablicy
  - `asList(arr)` - zwraca listę zawierającą elementy danej tablicy
  - `binarySearch(arr, value)` - zwraca indeks podanego elementu
- Przykłady zastosowania:

```
1 int[] arr1 = {1, 2, 3};
2 int[] arr2 = {6, 5, 4};
3 boolean rowne = Arrays.equals(arr1, arr2); // rowne = false
4 Arrays.fill(arr1, 4); // arr1 = [4, 4, 4]
5 Arrays.sort(arr2); // arr2 = [4, 5, 6]
6 List l = Arrays.asList(arr2);
7 int indeks = Arrays.binarySearch(arr2, 5); // index = 2
```

Klasa Vector

- Implementuje tablicę dynamiczną
- Niektóre metody zdefiniowane w klasie Vector:
  - `addElement(element)` - dodaje obiekt element do wektora
  - `capacity()` - zwraca pojemność wektora
  - `firstElement()` - zwraca pierwszy element wektora
  - `lastElement()` - zwraca ostatni element wektora
  - `contains()` - zwraca true, jeśli element jest w wektorze
- Przykłady zastosowania:

```
1 Vector<Integer> v = new Vector<Integer>(2); //tworzymy
      wektor dlugosci 2
2 v.addElement(1);
3 v.addElement(8);
4 v.addElement(12);
5 int a = v.capacity(); // a = 4, przy dodaniu nowego elementu
      dlugosc wektora zwiekszyla sie dwukrotnie
6 int b = v.firstElement(); // b = 1
7 int c = v.lastElement(); // c = 12
```

Klasa Stack

- Podklasa klasy Vector
- Implementuje „Last In First Out”
- Deklaracja klasy: `class Stack<E>`
- Metody zdefiniowane w klasie Stack:
  - `empty()`
  - `peek()`
  - `pop()`
  - `push(element)`
  - `search(element)`
- Przykłady zastosowania:

```
1 Stack<Integer> st = new Stack<Integer>();
2 st.push(42);
3 st.push(7);
4 Integer a = st.pop();
5 Integer b = st.peek();
```

Część praktyczna

Skąd wziąć Javę

- do uruchomienia programu wystarczy Java Runtime Environment (JRE)
- kompilator (jak i JRE) znajduje się w Java SE Development Kit (JDK)
- `http://java.sun.com/javase/downloads/index.jsp`
- prawdopodobnie trzeba do zmiennej PATH dodać .../jdk6/bin

Kompilacja

- *javac Klasa.java*
- Powstanie plik Klasa.class
- Kompilator tłumaczy program na kod pośredni (bytecode)
- Podatne na reverse engineering - warto użyć obfuskatora jeśli to problem

Uruchamianie

- *java Klasa*
- Uruchamiamy plik Klasa.class
- *java -jar Klasa.jar* - w przypadku pliku jar

Przykładowy program

omówienie i uruchomienie

- Program znajduje się w katalogu src/
- Kompilujemy z poziomu tego katalogu, który zawiera katalog pl
- *javac pl/egarden/ProgramNaPrezentacje.java*
- Uruchamiamy tym poleceniem:
- *java pl.egarden.ProgramNaPrezentacje*

Hello World

czyli zrób pan(i) program w Javie

Prosimy o wykonanie następującego zadania:

- Otwórz notatnik lub inny prosty edytor plików
- Wpisz w nim poniższy kod
- Zapisz jako plik HelloWorld.java
- Skompiluj poleceniem *javac HelloWorld.java*
- Uruchom program poleceniem *java HelloWorld*

```
1  class HelloWorld {
2    public static void main(String args[]) {
3      System.out.println("Hello world");
4    }
5  } ;
```

Hello World

czyli zrób pan(i) program w Javie

Prosimy o wykonanie następującego zadania:

- Otwórz notatnik lub inny prosty edytor plików
- Wpisz w nim poniższy kod
- Zapisz jako plik HelloWorld.java
- Skompiluj poleceniem *javac HelloWorld.java*
- Uruchom program poleceniem *java HelloWorld*

```
1  class HelloWorld {
2    public static void main(String args[]) {
3      System.out.println("Hello world");
4    }
5  } ;
```

```
1  Wersja dla ambitnych ;)
2  ++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++
3  ..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>.
```

Koniec

Dziękujemy za uwagę