Beata Jankowska,
Instytut Automatyki, Robotyki i Inżynierii Informatycznej

X. Java po C++. Wprowadzenie

1. Rodowód Javy

1.1. Java jest powiązana z językiem C++, wywodzącym się w prostej linii od języka C: z języka C zaczerpnęła składnię, a z C++ – większość elementów związanych z obiektowością. Do Javy wprowadzono przy tym szereg innowacji, mających na celu rozwiązanie istotnych problemów, których nie udało się rozwiązać w poprzednich językach.

1.2. Autorami języka są: James Gosling, Patrick Naughton, Chris Warth i Mike Sheridan, z firmy Sun Microsystems. Pierwsza, powstała w 1992 roku wersja języka nosiła nazwę OAK. W 1995 roku przemianowano go na Java. W okresie od 1991 do 1995, nad rozwojem języka pracowały dalsze osoby, w tym: Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin i Tim Lindholm. W zamyśle, głównym celem przyświecającym autorom było zaprojektowanie języka niezależnego od platformy sprzętowej, który nadawałby się do tworzenia oprogramowania urządzeń domowego użytku.

1.3. Gwałtowny rozwój Internetu ujawnił w międzyczasie potrzebę opracowania nowego języka programowania, którego nieodłączną cechą miała być przenośność (brak rozróżniania systemów operacyjnych i procesorów przez Internet). Okazało się, że przy niewielkim nakładzie środków, można było Javę – język przewidziany do oprogramowania elektroniki użytkowej, zmodyfikować na potrzeby Internetu.

1.4. Poza licznymi podobieństwami do C i C++, Java z pewnością nie stanowiła zwykłego rozszerzenia tych języków. Zapełniając lukę związaną z pisaniem programów niezależnych od platformy sprzętowej i udostępnianych za pośrednictwem Internetu, Java wywarła odwrotny wpływ na późniejszą postać Internetu. Przede wszystkim, rozszerzyła zbiór obiektów, dających się bezproblemowo przesyłać w sieci internetowej. I tak, między serwerem a komputerem

klienta można było przesyłać już nie tylko informacje pasywne, lecz także dynamiczne, samowykonujące się programy.

1.5. Aplet (obecnie w odwrocie) to rodzaj programu napisanego w języku Java, który zaprojektowano do przesyłania w Internecie i automatycznego wykonywania przez przeglądarkę WWW obsługującą Javę. Pobierany na żądanie, podobnie jak obrazy, czy dźwięki, był jednak inteligentnym programem (reagującym na dane od użytkownika i podlegającym dynamicznym zmianom), nie zaś tylko animacją lub plikiem multimedialnym. Szerokie stosowanie apletów było możliwe dzięki rozwiązaniu w języku kwestii bezpieczeństwa i przenośności.

2. Kod bajtowy

2.1. Kompilator Javy generuje kod bajtowy, który nie jest kodem wykonywalnym. Kod bajtowy to zoptymalizowany zbiór instrukcji, zaprojektowanych do wykonywania przez system wykonawczy Javy, zwany maszyną wirtualną Javy.

2.2. Przekład programu źródłowego na kod bajtowy ułatwia uruchamianie tego kodu w różnych środowiskach: wystarczy na danej platformie komputerowej zaimplementować maszynę wirtualną. Takie podejście gwarantuje oczekiwaną przenośność aplikacji.

2.3. Fakt wykonywania programu przez maszynę wirtualną Javy jest też gwarancją wysokiego poziomu bezpieczeństwa. Sterująca wszystkim maszyna wirtualna jest w stanie blokować te działania programu, których skutki mogą się okazać negatywne dla systemu.

2.4. Program skompilowany do postaci pośredniej i interpretowany przez maszynę wirtualną działa wolniej od programu skompilowanego do kodu binarnego. Różnica szybkości jest jednak w tym wypadku nieznacząca, bo kod bajtowy jest kodem wysoce zoptymalizowanym. Dla zwiększenia wydajności, można dodatkowo przeprowadzić kompilację „w locie” z kodu bajtowego na kod wykonywalny.

Beata Jankowska,
Instytut Automatyki, Robotyki i Inżynierii Informatycznej

3. Java – podstawowe właściwości języka

3.1. Java jest językiem prostym do przyswojenia przez profesjonalnego programistę i dającym się równocześnie używać w sposób wydajny.

3.2. Java, będąc językiem obiektowym, stosuje racjonalne podejście do obiektowości. Model obiektów w Javie jest prosty i rozszerzalny. Typy proste (np. liczby całkowite) nie są – ze względów wydajnościowych – obiektami, jednak – na życzenie – można ich używać w specjalnie opracowanej wersji obiektowej.

3.3. Java jest językiem solidnym. Ograniczając programistę w pewnych istotnych kwestiach, chroni go przed popełnianiem typowych błędów programistycznych. Nieodłączną cechą języka jest ścisła kontrola typów; dzięki niej, weryfikacja poprawności kodu odbywa się już na etapie kompilacji. Automatyczne zarządzanie alokacją i zwalnianiem pamięci przez maszynę wirtualną Javy chroni programistę przed popełnianiem błędów związanych ze złym zarządzaniem pamięcią. Z kolei, zapewniając restrykcyjną, obiektową obsługę wyjątków, Java pomaga w efektywny sposób obsłużyć potencjalne błędy wykonania.

3.4. Dla spełnienia rzeczywistych wymagań związanych z tworzeniem interaktywnych aplikacji sieciowych, w język wbudowano mechanizmy programowania wielowątkowego.

3.5. W procesie projektowania języka położono nacisk na żywotność i przenośność programów. W efekcie powstał język neutralny względem architektury.

3.6. Kompilacja kodu źródłowego odbywa się do postaci pośredniej, zwanej kodem bajtowym. Kod bajtowy jest z kolei interpretowany przez szybki, działający efektywnie system wykonawczy Javy. Mimo zastosowania interpretacji, przyjęte rozwiązanie cechuje wysoka wydajność.

3.7. Obsługując protokół TCP/IP, Java świetnie nadaje się do użycia w rozproszonym środowisku Internetu. Java obsługuje zdalne wywoływanie metod (ang. RMI – Remote Method Invocation), czyli uruchamianie metod na odległych komputerach.

3.8. Program napisany w Javie zawiera informacje niezbędne do weryfikacji dostępu do obiektów w trakcie uruchamiania aplikacji. Tym samym, dynamiczne dołączenie kodu przebiega w sposób bezpieczny i przewidywalny.

4. Java – język programowania proceduralnego i obiektowego

4.1. W Javie zrealizowano podstawowe paradygmaty programowania proceduralnego i obiektowego (zorientowanie na proces, hermetyzacja, dziedziczenie, polimorfizm).

4.2. Oto krótki, przykładowy program w języku Java:

```
/* prosty program w języku Java*/
class Przyklad
{ public static void main(String args[])
  { int num = 100;
    System.out.println("Witaj uzytkowniku");
    while (num<10000)
     { num *= 2;
       System.out.println
         ("Aktualna wartość num wynosi" + num);
     }
  }
}
```

W celu skompilowania tego programu, należy wydać komendę:
`javac Przyklad.java`

W efekcie uzyskamy kod bajtowy `Przyklad.class`, który można uruchomić przy wykorzystaniu aplikacji uruchomieniowej `java`:

`java Przyklad`

Beata Jankowska,
Instytut Automatyki, Robotyki i Inżynierii Informatycznej

5. Konstrukcja wyrażeń w Javie

5.1. Wyrażenia w Javie konstruuje się z argumentów (literały, zmienne, wywołania metod) i operatorów, przy zastosowaniu tradycyjnej notacji infiksowej. Zmienne mogą być używane tylko w zakresie ich widoczności. Zakresy widoczności można w programach zagnieżdżać. Najszerszy zakres widoczności wyznacza klasa, kolejny – jej metoda (zakres inicjowany nawiasem „(”, należą do niego także parametry metody), następne – bloki kodu, występujące na coraz głębszych poziomach.

5.2. W ogólności, poprawna konstrukcja wyrażenia wymaga, by typy operandów były zgodne z oczekiwanymi. Dla spełnienia tego warunku, w pewnych charakterystycznych przypadkach możliwa jest automatyczna konwersja typów. Konwersja ta ma charakter rozszerzający – może być zrealizowana tylko w sytuacji, gdy typ docelowy jest typem pojemniejszym. Takiej konwersji nie można przeprowadzić, np. pomiędzy niezgodnymi ze sobą typem numerycznym a typem `char`/`boolean` lub pomiędzy typem `int` a `byte`.

5.3. Aby dokonać konwersji między typami niezgodnymi, trzeba użyć jawnej konwersji typu, zwanej rzutowaniem, w postaci:
(*typ_docelowy*) *wyrażenie*
np.

```
int a;
byte b; //...
b = (byte) a;
```

5.4. Wymienione zasady konwersji automatycznej i jawnej obowiązują także w odniesieniu do instrukcji przypisania, np.:

```
double c;
int a = 1;
float b = 5.3; //...
c= a*b;
```

5.5. Spośród operatorów dostępnych w Javie, poza znanymi wcześniej operatorami arytmetycznymi, bitowymi, relacyjnymi,

logicznymi, przypisania i warunkowym, uwagę zwraca rozbudowany operator przesunięcia bitowego w prawo (także z wypełnianiem zerami) oraz operatory logiczne ze skracaniem, zwane też operatorami niepełnymi, np.

```
if ((mianownik != 0) && (licznik/mianownik > 10))
// ...
```

6. Struktury danych w języku Java

6.1. Spośród znanych struktur danych, w języku Java programowo dostępne są jedynie tablice. Dopuszcza się przy tym używanie zarówno tablic jednowymiarowych, jak i wielowymiarowych. Tablice wielowymiarowe są w istocie tablicami tablic. Przy alokacji pamięci dla tablicy wielowymiarowej, obowiązkowe jest podanie tylko pierwszego wymiaru. Pozostałe wymiary można deklarować osobno. Zgodnie z życzeniem, każdy z nich może mieć tę samą lub różne liczby elementów.

```
class TabDwuwymiar {
 public static void main(String args[]) {
  int dwaW[][] = new int[4][];
  dwaW[0] = new int[1];
  dwaW[1] = new int[2];
  dwaW[2] = new int[3];
  dwaW[3] = new int[4];
  int i,j,k = 0;
  for(i=0; i<4; i++)
    for(j=0; j<i+1; j++) {
      dwaW[i][j] = k;
      k++;
    }
  for(i=0; i<4; i++) {
    for(j=0; j<i+1; j++)
      System.out.print(dwaW[i][j] + ” ”);
    System.out.println();
  }
 }
}
```

7. Struktury sterowania w języku Java

7.1. Java oferuje programiście wszystkie struktury sterowania dostępne we wcześniejszych językach proceduralnych i obiektowych. Są wśród nich: bloki kodu (`{...}`), instrukcje wyboru (`if` oraz `switch`), instrukcje iteracji (`while`, `do-while`, `for`) oraz strukturalne instrukcje skoku (`break`, `continue` i `return`).

7.2. Ponadto, w wersjach języka od J2SE 5 poczynając, udostępniono pętlę typu for_each. Pętla ta, zwana inaczej usprawnioną wersją pętli for, ma konstrukcję składniową w postaci:

`for` (*typ zmienna-iteracyjna*`:` *kolekcja*) *blok-instrukcji*

Oto przykład jej użycia:

```
class Foreach {
  public static void main(String args[]) {
    int nums[] = {1,2,3,4,5,6,7,8,9,10};
    int sum = 0;
    for (int x : nums){
      System.out.println("Wartosc – " + x);
      sum += x;
     }
    System.out.println("Suma wartości: " + sum);
   }
  }
```

8. Uwagi końcowe

8.1. Łańcuchy (typ `String`) są zaimplementowane w Javie jako obiekty. Korzystanie z nich jest proste; ułatwia je duża liczba metod wbudowanych.

8.2. Język Java nie obsługuje wskaźników. Jest to spowodowane koniecznością ustawienia zapory między środowiskiem wykonawczym Javy a systemem operacyjnym komputera. Na

szczęście, Java jest tak zaprojektowana, że wskaźniki stają się niepotrzebne, gdy pozostaje się wewnątrz środowiska wykonawczego.