Kurs programowania

Wykład 1

Wojciech Macyna

3 marca 2016

Słowa kluczowe języka Java

```
abstract, break, case, catch, class, const, continue,
default, do, else, extends, final, finally, for, goto, if,
implements, import, instanceof, interface, native, new,
package, private, protected, public, return, static,
super, switch, synchronized, this, throw, throws,
transient, try, volatile, while
```

Typy podstawowe

```
boolean, byte, char, double, float, int, long, short, void
```

Pierwotne typy danych w Javie mają ustaloną długość (w przeciwieństwie do C, gdzie może zależeć to od systemu).
Nie ma typów całkowitych bez znaku. Nie ma także niejawnych konwersji między typami.
Typy podstawowe mają swoje odpowiedniki obiektowe – klasy opakowujące.

Operatory

inkrementacja/dekrementacja	`++, --`
operatory arytmetyczne	`+, -, *, /, %`
operatory przesunięć	`<<, >>, >>>`
porównania	`<, <=, >, >=, ==, !=`
bitowe OR, AND, XOR	`&, |, ^`
logiczne OR, AND	`&&, ||`
przypisanie	`=`

Przypisanie można łączyć z większością operatorów (jak w C).

Klasa String

Łańcuchy tekstowe w Javie są zdefiniowane za pomocą klasy String. Ich obsługa jest mocno uproszczona można je przypisywać (operator =) i konkatenować (operator +). Przy próbie konkatenacji innych obiektów niż String wywoływana jest metoda toString(). Łańcuchy można też wprost porównywać.

Przykład

Dodaj.java

```
1  public class Dodaj {
2    public static void main(String args[]) {
3      int a,b;
4      if( args.length!=2 ) {
5        System.out.println("Zla liczba argumentow!");
6        return;
7      }
8      try {
9        a = Integer.parseInt( args[0] );
10       b = Integer.parseInt( args[1] );
11     }
12     catch( NumberFormatException ex ) {
13       System.out.println("Zly format argumentow!");
14       return;
15     }
16     System.out.println( a+" + "+b+" = "+(a+b) );
17   }
18 }
```

Definiowanie klas

Podstawowe elementy klasy

- Pola – jakie dane będzie przechowywać klasa.
- Metody – jakie operacje będziemy wykonywać na danych w tej klasie.
- Konstruktory – jak inicjować obiekty w danej klasie (konstruktor domyślny).
- Współdziałanie między klasami.

Klasa Temperatura

Mamy trzy powszechnie znane jednostki temperatury: Celsjusz, Kelvin, Fahrenheit. Zdefiniujmy klasę która będzie przechowywała temperaturę Celsjusza, ale miała też metody przystosowane do pozostałych jednostek.

Klasa Temperatura

Temperatura.java

```
1  /** Klasa do operacji na temperaturach */
2  public class Temperatura {
3    private double t = 0.0;
4
5    public double Kelvin() { return t+273.15; }
6    public double Fahrenheit() { return 1.8*t+32; }
7    public double Celsjusz() { return t; }
8
9    public void set(double t) { this.t = t; }
10   public void setF(double t) { this.t = 5.0*(t-32)/9.0; }
11   public void setK(double t) { this.t = t-273.15; }
12
13   // Konstruktory
14   Temperatura() { t = 0.0; }
15   Temperatura(double t) { this.t = t; }
16   // Konwersja do String
17   public String toString() { return Double.toString(t); }
18 }
```

Klasa TemperaturaTest

TemperaturaTest.java

```
1  public class TemperaturaTest {
2    public static void main( String args[] ) {
3      Temperatura a;
4      Temperatura b = new Temperatura();
5      Temperatura c = new Temperatura(100.0);
6      a = b;
7      System.out.println( "C:"+a+";␣K:"+a.Kelvin()+";␣F:"+
8          a.Fahrenheit()+";");
9      System.out.println( "C:"+c+";␣K:"+c.Kelvin()+";␣F:"+
10         c.Fahrenheit()+";");
11     c = b;
12     System.out.println( "C:"+c+";␣K:"+c.Kelvin()+";␣F:"+
13         c.Fahrenheit()+";");
14     c.setF( 0.0 );
15     System.out.println( "C:"+c+";␣K:"+c.Kelvin()+";␣F:"+
16         c.Fahrenheit()+";");
17   }
18 }
```

Ten sam program w C++

Temperatura.cpp

```
1  #include<iostream>
2  using namespace std;
3
4  class Temperatura {
5    private:
6      double t;
7    public:
8      double Kelvin() { return t+273.15; }
9      double Fahrenheit() { return 1.8*t+32; }
10     double Celsjusz() { return t; }
11     void set(double t) { this->t = t; }
12     void setF(double t) { this->t = 5.0*(t-32)/9.0; }
13     void setK(double t) { this->t = t-273.15; }
14   // Konstruktory
15     Temperatura() { t = 0.0; }
16     Temperatura(double t) { this->t = t; }
17
18     friend ostream & operator<<(ostream & out, Temperatura * t) { return out<<(t->t); }
19 };
20 int main(int argc, char *argv[]) {
21   Temperatura * a;
22   Temperatura * b = new Temperatura();
23   Temperatura * c = new Temperatura(100.0);
24
25   a = b;
26   cout << "C:" << a << "; K:" << a->Kelvin() << "; F:" << a->Fahrenheit() << ";" << endl;
27   cout << "C:" << c << "; K:" << c->Kelvin() << "; F:" << c->Fahrenheit() << ";" << endl;
28   c = b;
29   cout << "C:" << c << "; K:" << c->Kelvin() << "; F:" << c->Fahrenheit() << ";" << endl;
30   c->setF( 0.0 );
31   cout << "C:" << c << "; K:" << c->Kelvin() << "; F:" << c->Fahrenheit() << ";" << endl;
32 }
```

Dostęp

Typy dostępu do komponentów klasy

`public` – dostępne zawsze.

`private` – dostępne tylko wewnątrz klasy w której jest zdefiniowane.

`protected` – dostępne tylko wewnątrz pakietu klas, poza nim traktowany jak `private`.

(default) – podobnie jak `protected`.

Dostęp do klas

Zasadniczo klasy są publiczne albo mają domyślny dostęp (w ramach pakietu).

Tworzenie obiektu

Deklaracja zmiennej i przypisanie mu obiektu w języku Java

```
1 Temperatura t;
2 t = new Temperatura();
3 Temperatura s = new Temperatura(20.1);
```

Typ pola	Domyślna wartość
byte, short, int, long	0
float, double	0.0
char	znak o kodzie 0
boolean	false
referencja innego typu	null

Deklaracja zmiennej i przypisanie mu obiektu w języku C++

```
1 Temperatura t;
2 Temperatura & t1 = t; // t ma teraz rownoczesnie nazwe t1
3 Temperatura * t2 = new Temperatura(20.1);
```

Referencje

Referencje są wskaźnikami pamięci (w typowym przypadku 32-bitowymi). Jednak trzeba pamiętać, że w programie są do nich przypisane typy obiektów na które wskazują, więc nie można podstawić pod referencję jednego typu obiektu innego typu (kompilator potraktuje to jako błąd).
Niektóre języki obiektowe umożliwiają takie zachowanie.

Dostęp do pól i metod

Operator .

Dostęp do komponentów jest realizowany za pomocą kropki: nazwa-obiektu(kropka)nazwa-metody.

```
1 Temperatura t = new Temperatura();
2
3 t.set(30.1);
4 System.out.println( t.Kelvin() );
```

Referencja `this` (Java)

W definicji klasy aby odwołać się do pól i metod obiektu wewnątrz jego samego używamy albo bezpośrednio nazw pól albo używamy referencji `this` (w przypadkach kiedy mamy przesłonięte nazwy pól).

Wskaźnik `this` (C++)

W C++ `this` jest wskaźnikiem. Dla wskaźników operator dostępu ma postać `->`, np. `this->t`.

Komponenty statyczne

Pola i metody statyczne nie są przypisane do konkretnego obiektu a do klasy jako całości. Ich wywołanie i użycie nie wymaga stworzenia obiektu, a wszystkie stworzone obiekty widzą tylko jedno (to samo) pole.
Wywołanie komponentu statycznego bez tworzenia obiektu ma postać:

```
nazwa_klasy.nazwa_komponentu_statycznego // Java
nazwa_klasy::nazwa_komponentu_statycznego // C++
```

Maszyna Wirtualna Javy uruchamiana dla danej klasy szuka w niej i uruchamia statyczną metodę `main`.

Wyjątki

Metody w sytuacjach wystąpienia błędu mogą zgłaszać *wyjątki*. Wyjątek przerywa działanie podprogramu i jest zwracany wyżej.

Deklarowanie wyjątku

```
class MojException extends Exception {};
```

Deklarowanie użycia wyjątku przez metodę

```
void mojaMetoda() throws MojException;
```

Tak zadeklarowana metoda może być użyta tylko w środowisku `try{ }` a sam wyjątek może być przechwycony poleceniem `catch`.

Wywołanie wyjątku

```
throw new MojException();
```

Przykład

```
1  class Moj1Exception extends Exception {};
2  class Moj2Exception extends Exception {};
3
4  class TablicaBool {
5    private boolean[] t;
6    public boolean getValue(int i) throws Moj2Exception {
7      if((i<0)||(i>=t.length)) throw new Moj2Exception();
8      return t[i];
9    }
10   public void setValue(int i,boolean b) throws Moj2Exception {
11     if((i<0)||(i>=t.length)) throw new Moj2Exception();
12     t[i] = b;
13   }
14   TablicaBool(int i) throws Moj1Exception {
15     if(i<=0) throw new Moj1Exception();
16     t = new boolean[i];
17   }
18 }
```

Przykład cd

```
1  public class TablicaBoolTest {
2    public static void main(String[] arg) {
3      TablicaBool t;
4      try { t = new TablicaBool(2); }
5      catch( Moj1Exception e ) {
6        System.out.println( "Zle inicjowanie tablicy!" );
7        return;
8      }
9      try { t.setValue(20,true); }
10     catch( Moj2Exception e ) {
11       System.out.println( "Przekroczenie zakresu" ); }
12     try { t = new TablicaBool(0); t.setValue(20,true); }
13     catch( Moj2Exception e ) {
14       System.out.println( "Przekroczenie zakresu" ); }
15     catch( Moj1Exception e ) {
16       System.out.println( "Zle inicjowanie tablicy!" );
17       return; }
18   }
19 }
```

Ten sam program w C++

TablicaBool.cpp (1)

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4  class TablicaBool {
5    private:
6      bool *t;
7    public:
8      bool getValue( int i ) throw(string) {
9        if((i<0)||(i>=sizeof t)) throw (string)"Zly indeks";
10       return t[i];
11     }
12     void setValue( int i, bool b ) throw(string) {
13       if((i<0)||(i>=sizeof t)) throw (string)"Zly indeks";
14       t[i] = b;
15     }
16     TablicaBool( int i ) throw(string) {
17       if(i<=0) throw (string)"Zly rozmiar tablicy";
18       t = new bool(i);
19     }
20     ~TablicaBool() { delete t; }
21 };
```

Ten sam program w C++

TablicaBool.cpp (2)

```
22 int main(int argc, char* argv[] ) {
23   TablicaBool *t;
24   int n = 2;
25
26   try { t = new TablicaBool(n); }
27   catch(string w) { cout << w << endl; return 1; }
28   try { t->setValue(20,true); }
29   catch(string w) { cout << w << endl; }
30   delete t;
31   n = 0;
32   try { t = new TablicaBool(0); t->setValue(20,true); }
33   catch(string w) { cout << w << endl; return 1; }
34 }
```

Konstruktory

Konstruktor domyślny (metoda o nazwie klasy)

Jeśli nie zdefiniujemy w klasie własnych konstruktorów to będzie ona miała konstruktor domyślny, który nada wszystkim polom wartości domyślne.

Zdefiniowanie chociaż jednego własnego konstruktora powoduje, że konstruktor domyślny nie jest już zdefiniowany.

Każdy konstruktor musi się różnić od innych listą argumentów.

Destruktory

Destruktor – `protected void finalize() throws Throwable` (Java)

Ponieważ Java ma Garbage Collector definiowanie destruktorów w większości przypadków nie jest konieczne.
Jeśli jednak potrzebujemy destruktora to nadpisujemy metodę `finalize` która jest wywoływana przez Garbage Collector w chwili zwalniania pamięci obiektu.

Destruktor – `~nazwa_klasy()` (C++)

Destruktor w C++ jest wywoływany przy wyjściu z zakresu widzialności zmiennej obiektowej lub przy zwalnianiu obiektu słowem kluczowym `delete`.

Przykład (Java)

```
1  class Test {
2    public int a;
3    Test(int i) { a=i; System.out.println( "Utworzono␣"+a ); }
4    protected void finalize() throws Throwable {
5      System.out.println( "Zniszczono␣"+a ); }
6  }
7  public class DestruktorTest {
8    public static void main(String[] arg) {
9      Test a = new Test(1), b = new Test(2), c = new Test(3);
10
11     a = b;
12     System.gc();
13     b = c;
14     a = new Test(4);
15     System.gc();
16     b = c = null;
17     System.gc();
18   }
19 }
```

Przykład (C++)

```
1  #include <iostream>
2  using namespace std;
3
4  class Test {
5    public:
6      int a;
7      Test( int a ) {
8        this->a=a;
9        cout << "Utworzono " << a << endl;
10     }
11     ~Test() { cout << "Zniszczono " << a << endl; }
12 };
13 int main(int argc, char* argv[]) {
14   Test a(1);
15   Test* b = new Test(2);
16
17   delete b;
18 }
```