Języki i Paradygmaty Programowania

Programowanie obiektowe i Smalltalk

Marcin Benke

MIM UW

30 maja 2016

Podstawowe klasy

```
Object
  Boolean
    False
    True
  Magnitude
    Character
    Number
      Float
      Fraction
      Integer
    Date
    Time
  Collection
```

```
Collection
  Bag
  Set
    Dictionary
  SequenceableCollection
    ArrayedCollection
      Array
      String
        Symbol
    OrderedCollection
      SortedCollection
```

Klasa Object(operacje na wszystkich obiektach)

Tworzenie obiektów (metody klasowe)

- `new` — daje nowy egzemplarz odbiorcy (który jest klasą)
- `new: anInteger` — daje egzemplarz odbiorcy o podanym rozmiarze. Dopuszczone tylko dla klas indeksowalnych, np.

  ```
  Array new: 10
  String new: 10
  ```

- `new` może być (i często jest) reimplementowane w klasach użytkownika. Zwykle treścią `new` jest
  `^super new initialize`.
  należy pamiętać o umieszczeniu go po stronie klasowej, o `^`, i by wysłać komunikat przez `super`, nie `self`!

Tworzenie obiektów c.d.

- Nie implementuj **new** jeśli nie wnosi to nic nowego.
- Jeśli obiekt ma być parametryzowany czymś innym niż tylko rozmiar, lepiej wybrać nazwę inną niż `new:`

  Przykład z klasy **Point**:

```
x: xInteger y: yInteger
  "Answer an instance of me
   with coordinates xInteger and yInteger."

  ^self new setX: xInteger setY: yInteger
```

  Egzemplarz **Point** mozna stworzyć np. tak:

```
Point x: 0 y: 0
```

Porównywanie obiektów

Komunikat	Opis
`== anObject`	czy odbiorca i argument są tym samym obiektem? (true/false)
`~~ anObject`	czy odbiorca i argument są róznymi obiektami?
`= anObject`	czy odbiorca jest równy argumentowi? (znaczenie zależne od klasy odbiorcy).
`~= anObject`	czy odbiorca i argument są nierówne?
`hash`	daje SmallInteger powiązany z wartością obiektu (równe obiekty dają ten sam hash).

Testowanie obiektów

Komunikat	Opis
`isNil`	czy odbiorcą jest nil? (true/false)
`notNil`	czy odbiorcą nie jest nil?
`ifNil: aBlock`	gdy odbiorcą jest nil, oblicza `aBlock` i daje wartośc bloku; wpp. daje odbiorcę.
`ifNotNil: aBlock`	dualne do powyższego

Kopiowanie obiektów

Komunikat	Opis
`copy`	Daje egzemplarz analogiczny w swoim stanie i zachowaniu do odbiorcy. Podklasy przeważnie implementują tę metodę, nie naruszając `shallowCopy`.
`shallowCopy`	"płytka" kopia (zachowuje wartości atrybutów)
`deepCopy`	"głęboka" kopia (rekurencyjnie kopiuje atrubuty)

Uwaga: nie należy nadużywać `deepCopy`

Wyświetlanie i zapisywanie obiektów

Komunikat	Opis
`printString`	daje **String** opisujący odbiorcę
`printOn: aStream`	zapisuje, na strumieniu `aStream`, **String** opisujący odbiorcę.
`storeString`	daje **String** z którego można odtworzyć odbiorcę.
`storeOn: aStream`	zapisuje taki napis na `aStream`
`asString`	daje obiekt klasy **String** odpowiadający odbiorcy

```
Object new printString
'anObject'
```

Uwaga: w klasie **Object** `asString` daje wynik `printString`, ale w podklasach może być inaczej, np.

```
'Hello' printString        '''Hello'''
'Hello' asString           'Hello'
```

Introspekcja

Komunikat	Opis
`class`	daje klasę odbiorcy (obiekt).
`isKindOf: aClass`	czy odbiorcą jest egzemplarz `aClass` lub jej podklasy?
`isMemberOf: aClass`	czy odbiorcą jest egzemplarz `aClass`? (`rcvr class == aClass`)
`respondsTo: msg`	czy odbiorca moze znaleźć metodę dla `msg`, w swojej klasie lub jej nadklasach? (`self class canUnderstand: msg`)
`canUnderstand: msg`	czy odbiorca (klasa) ma metodę dla `msg`? (może być w nadklasie).

Różne

Komunikat	Opis
`yourself`	odpowiada self.
`doesNotUnderstand: msg`	obsługa sytuacji gdy odbiorca nie rozumie komunikatu `msg`.
`error: aString`	zgłasza błąd.
`halt`	zatrzymuje wykonanie.

1. Komunikat `yourself` jest przeważnie używany w kaskadzie, gdy poprzedni komunikat dał coś innego niż odbiorcę, n.p.

   ```
   #(1 2 3 5) at: 4 put: 4
   ```

   daje 4, podczas gdy

   ```
   #(1 2 3 5) at: 4 put: 4; yourself
   ```

   daje `#(1 2 3 4)`, odbiorcę
2. `self halt` jest standardową metodą wejścia do debuggera. Stamtąd wykonanie może być wznowione.

Magnitude

Abstrakcyjna klasa wielkości, z liniowym porządkiem danym przez

`<` `<=` `>` `>=`

Podklasy:

Character

Date, Time, TimeStamp

Number — klasa liczb z operacjami arytmetycznymi

- `+ - * /`
- `// aNumber` — dzielenie całkowite
- `\\ aNumber` — modulo
- `abs, negated`
- `floor, ceiling, asInteger, asFloat,...`
- `sin, cos,...`

  Float

  Fraction — liczby wymierne (znormalizowane ułamki)

  Integer

Integer

Klasa liczb całkowitych. Podklasy:

- `SmallInteger` — jedno słowo maszynowe
- `LargeInteger` — liczby całkowite dowolnej precyzji

metody

- `even, negative`
- `factorial`
- `gcd: anInteger` — najwiekszy wspólny dzielnik
- `lcm: anInteger` — najmniejsza wszpólna wielokrotność
- `digitLength`
- `digitAt: anInteger`
- `bitAnd: n, bitOr: n, bitXor: n`
- `bitAt: n, bitAt: n put: v`

Kolekcje

Klasa	Opis
Collection	abstrakcyjna nadklasa kolekcji
Bag	multizbiór
Set	zbiór (porównanie przez =)
Dictionary	słownik — zbiór asocjacji (Association)
SequenceableCollection	Uporządkowana kolekcja, indeksowana
OrderedCollection	Porządek wg kolejności wstawiania
SortedCollection	Porządek użytkownika (“sortBlock”)
Interval	ciąg arytmetyczny
ArrayedCollection	Uporządkowana kolekcja ustalonego rozmiaru, indeksowana liczbami.

Kolekcje

Klasa	Opis
ArrayedCollection	Uporządkowana kolekcja ustalonego rozmiaru, indeksowana liczbami.
Array	tablica dowolnych obiektów
String	tablica obiektów Character
Symbol	tablica obiektów Character (z gwarancją unikalności)

Tworzenie kolekcji

- `new`
- `new: size`
- `with: element`
- `with: element1 ... with: element5`
- `withAll: kolekcja`
- `a to: b` — tworzy ciąg (`Interval`)
- `a to: b by: c`

```
Array new: 8
#(nil nil nil nil nil nil nil nil)
Set with: 7 with: 13
a Set(7 13)
Set withAll: #(1 2 3 2 1)
a Set(1 2 3)
(1 to: 3) asArray
#(1 2 3)
```

Dodawanie i usuwanie elementów

- `add: el`
- `addAll: kolekcja`
- `at: n put: x` (tylko dla indeksowanych)
- `remove: element`
- `remove: x ifAbsent: blok`

```
b:= Bag new. b addAll: #(1 2 3 2 1); yourself
a Bag(1 1 2 2 3)
s:= Set new. s addAll: #(1 2 3 2 1);yourself
a Set(1 2 3)
a:= Array new: 3. a at: 2 put: 2. a
#(nil 2 nil)
#(1 2 3 2 1) asBag remove: 1; yourself
a Bag(1 2 2 3)
```

Elementy kolekcji

- Kolekcje mogą przechowywać elementy (prawie) dowolnych klas
- Elementy kolekcji nie muszą byc jednej klasy, ale najlepiej gdy mają wspólny protokół (zwykle nadklasę)
- Tylko tablice mogą przechowywać `nil`.
- `Array new: n` zawiera n razy `nil`
- `String new: n` zawiera n razy znak o kodzie 0
- Dla kolekcji indeksowanych `at: n` daje element o indeksie n
- Podklasy **SequenceableCollection** indeksowane liczbami
- **Dictionary** indeksowane kluczami (za pośrednictwem metody `hash` klucza).

OrderedCollection

- `addFirst: elem, addLast: elem`
- `addAllFirst: kolekcja, addAllLast: kolekcja`
- `add: e1 after: e2`
- `add: e afterIndex: i`
- ostatnie dwa z before zamiast after
- `removeFirst, removeLast`

Testowanie

- `size`
- `isEmpty`
- `includes: element` — czy zawiera
- `allSatisfy: blok1Arg`
- `anySatisfy: blok1Arg`

Filtrowanie

- `select: blok1Arg` — wybierz dające true
- `reject: blok1Arg` — wybierz dające false
- `detect: blok1Arg` — znajdź pierwszy dobry
- `detect: blok1Arg ifNone: blok`
- `anyOne` — daj jakiś element

Iterowanie

- `do: blok1Arg`
- `reverseDo: blok1Arg` (**SequenceableCollection**)
- `collect: blok1Arg` — “map”
- `inject: start into: blok2Arg` — “foldl”

```
#(1 2 3)
  inject: 0
  into: [:a :b | a+b]
6
```

Uwaga: nie modyfikować kolekcji w trakcie iteracji

Interval

Kolekcje nie muszą naprawdę przechowywać obiektów.
Przykładem jest klasa `Interval`

```
i := Interval from: 1 to: 10 by: 2.
(1 to: 10 by: 2)
```

elementy kolekcji nie są tu przechowywane, a tylko generowane na żądanie.
Próba dodania elementu do kolekcji klasy **Interval** skutkuje błędem (**shouldNotImplement**)

```
Interval>>add: newObject
"Adding to an Interval is not allowed."

self shouldNotImplement
```

#collect: i #species

```
i := Interval from: 1 to: 10 by: 2.
(i collect: [:x| x*x])
#(1 9 25 49 81)
```

Jakiej klasy obiekt powinien być tworzony przez `#collect:` ? Nie jest to **Interval**
Odpowiedź uzyskamy za pomocą komunikatu `#species`

```
i species
Array
```

```
SequenceableCollection>>collect: aBlock
    | newCollection |
    newCollection := self species new: self size.
    1 to: self size do:
        [:index |
        newCollection at: index
                    put: (aBlock value: (self at: index))].
    ^ newCollection
```

SortedCollection

Domyślnie elementy sortowane za pomocą <=

```
SortedCollection new add: 3; add: 2; add: 4;yourself
a SortedCollection(2 3 4)
```

Kolekcja obiektów posortowana po dacie:

```
dc := SortedCollection sortBlock: [:o1 :o2| o1 date <= o2 date]
```

Zakładamy, że wszystkie wstawiane obiekty odpowiadają na komunikat `date` podając datę.
Nie jest istotne jakiej klasy są daty podane przez obiekty; ważne tylko, że dają się porównać.