

# Ako napísať program v Smalltalku

## Obsah

Obsah.....	1
Ako vyzerá program v Smalltalku?.....	2
Classes.....	8
Hierarchia tried, dedičnosť.....	12
Úlohy.....	15
Program Smalltalk.....	17
Identita objektov, referencie.....	19
Základné princípy objektovej orientácie.....	19
Vývojové prostredie.....	20
Workspace.....	20
Inspector.....	20
Transcript.....	20
Class Browser.....	21
Debugger.....	22
Základné triedy.....	23
Object.....	23
UndefinedObject.....	25
Bloky.....	25
True, False.....	26
Čísla.....	27
Collections.....	29
Verzie Smalltalku.....	35
Kód.....	36
Slovník.....	37

copyright: 1999, Whitestein Technologies Slovakia sro.

Žiadna časť tohoto dokumentu nesmie byť reprodukováaná v žiadnej forme ani žiadnym prostriedkom bez prísomného povolenia autora.

# Ako napísať program v Smalltalku

## Ako vyzerá program v Smalltalku?

Neexistuje main().

Treba naprogramovať niekoľko objektov spolu s príslušnými metódami (funkciami). Aplikácia je množina vzájomne pôsobiacich objektov. ríp. treba urobiť nejaké okienka. rogram sa spúšťa z menu, alebo vykonaním nejakých príkazov zavolaných z iných aplikácií.

Napr. program knižnica pozostáva z:

- objekty: Library, Reader, Book a Borrowing
- metódy ako napr.: zaeviovanie/odevidovanie nového čitateľa, novej knihy, vypožičanie knihy, vrátenie knihy. Tieto sú implementované vo vytvorených objektoch, t.j. nie sú samostatne stojace.
- okienka: vo vývojovom prostredí sa vytvoria okná pre knižnicu, v ktorých je možné prezerat' knižnicu a robiť nejaké činnosti s knižnicou
- zabezpečí sa spustenie knižnice, napr. stlačením nejakého tlačítka.

# Ako napísať program v Smalltalku

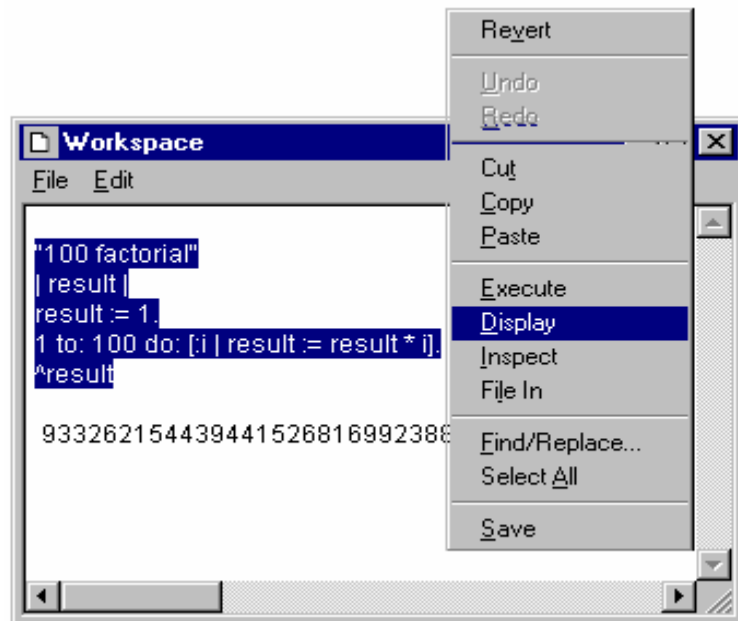
## Malý kus kódu

Malé kúsky kódu je možné spúšťať aj bez toho, že by existoval nejaký objekt.

Napr. výpočet faktoriálu:

```
"100 factorial"  
| result |  
result := 1.  
1 to: 100 do: [:i | result := result * i].  
^result
```

Prostredie Smalltalku je  
zväčša tzv. inkrementálny  
interpretér.  
Nerobia sa exe súbory.



# Ako napísať program v Smalltalku

## Základná syntax

34			SmallInteger
1.56e-3			Float
93326215443944152681699			LargeInteger
'a string'	'a string'		String
#next			Symbol
\$c	'c'		Character
true			True
false			False
nil			UndefinedObject
#(-25 'a string' \$c)			Array
3 + 4	3 + 4	7	SmallInteger
5 factorial	factorial(5)	120	SmallInteger
16.79 rounded	round(16.79)	16	SmallInteger
3 negated	-3	-3	SmallInteger
'abcdef' size	size('abcdef')	5	SmallInteger
'abc' ~= 'def'	'abc' <> 'def'	true	True
28 gcd: 12	gcd(28, 12)	4	SmallInteger
#(-25 'a string' \$c) at: 2	x[2]	'a string'	String
5 between: 3 and: 12	between(5, 3, 12)	true	True

- unary message
- binary message
- keyword message
- literal

2 factorial negated	(2 factorial) negated
3 + 4 * 6 + 3	((3 + 4) * 6) + 3
15 gcd: 32 // 3	15 gcd: (32 // 3)
2 factorial + 4	(2 factorial) + 4
5 between: 1 and: 3 squared + 4	5 between: 1 and: ((3 squared) + 4)
4 factorial gcd: 4 * 6	(4 factorial) gcd: (4 * 6)



# Ako napísať program v Smalltalku

## Control Structures

2<3

```
ifTrue: ['dva je mensie ako tri']  
ifFalse: ['dva nie je mensie ako tri']
```

'dva je mensie ako tri'

```
'abc' = 'abc' ifTrue: ['rovnake retazce']
```

'rovnake retazce'

```
"30 factorial"  
| result mul |  
result := 1.  
mul := 0.  
[ mul := mul + 1.  
  mul <= 30 ] whileTrue: [ result := result * mul].  
^result
```

26525285981219105863630848000000

```
"30 factorial"  
| result mul |  
result := 1.  
mul := 0.  
[ mul := mul + 1.  
  mul > 30 ] whileFalse: [ result := result * mul].  
^result
```

26525285981219105863630848000000

```
"30 factorial"  
| result |  
result := 1.  
1 to: 30 do: [:i | result := result * i].  
^result
```

26525285981219105863630848000000

```
"30 factorial"  
| result |  
result := 1.  
(1 to: 30) do: [:i | result := result * i].  
^result
```

26525285981219105863630848000000

```
"30 factorial"  
| result mul |  
result := 1.  
mul := 1.  
30 timesRepeat: [  
  result := result * mul.  
  mul := mul + 1 ].  
^result
```

26525285981219105863630848000000

# Ako napísať program v Smalltalku

```
"Loking up primes"
```

```
| all n |
```

```
n := 100.
```

```
all := Array new: n.
```

```
1 to: n do: [:i | all at: i put: i ].
```

```
2 to: n sqrt do: [:i |
```

```
  (all at: i) notNil ifTrue: [
```

```
    2 * i to: n by: i do: [:j | all at: j put: nil] ] ].
```

```
all select: [:i | i notNil ]
```

```
(1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97)
```

## Úlohy

**Cvičenie 1.1.:** Napíšte kód na výpočet Fibonacciho čísel.

# Ako napísať program v Smalltalku

## Classes

### Stav

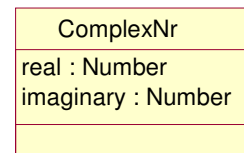
```
Magnitude subclass: #ComplexNr  
  instanceVariableNames: 'real imaginary '  
  classVariableNames: ''  
  poolDictionaries: ''
```

```
c1 := ComplexNr new
```

```
struct {  
  real: float;  
  imaginary: float  
} complexnr;
```

```
struct complexnr c1, c2;
```

```
c1.real
```



stav objektu je skrytý
---------------------------

Trieda je podobná štruktúram v c: definuje typ, ktorý ma pomenované premenné.

Na rozdiel od c, definuje trieda aj správanie, t.j. má implementované funkcie, ktoré sa dajú aplikovať iba pre objekty tohoto typu.

Objekt daného typu sa nazýva **inštancia** tejto triedy.

Premenné `real` a `imaginary` sa nazývajú **inštančné premenné**.

nie je určený typ premenných <code>real</code> a <code>imaginary</code> ; sú to pointre na ľubovoľný objekt
--

## Triedy a inštancie

Dva typy objektov: triedy a netriedy.

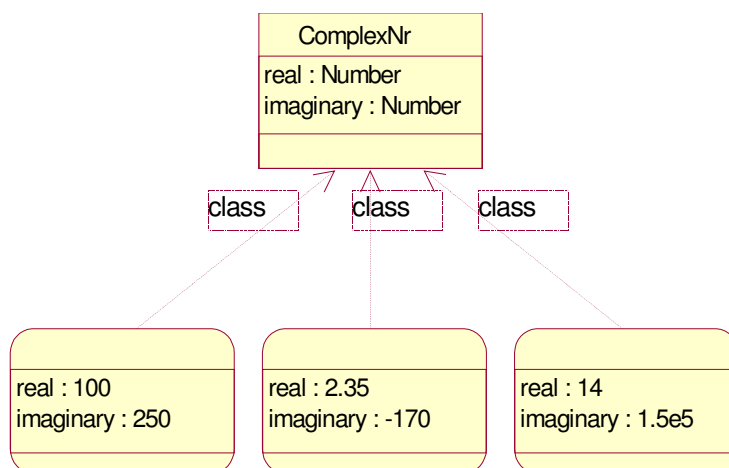
Triedy môžu mať inštancie.

Inštancie tried vznikajú napr.:

```
ComplexNr new
```

Inštancie majú svoj vlastný stav.

Inštancie sa správajú tak, ako im to definuje trieda, t.j. vedia definovať tie funkcie, ktoré sú definované v triede.



# Ako napísať program v Smalltalku

## Správanie

Správanie objektu vytvárajú jeho metódy.

Metódy programujeme vo vývojovom prostredí.

Metódy môžu byť triedne a inštančné.

Podstatné pre objekty je ich správanie.

*ComplexNr class publicMethods*

**real: num1 imaginary: num2**

```
^(self new)
  real: num1;
  imaginary: num2
```

*ComplexNr publicMethods*

**imaginary**

```
^imaginary
```

**imaginary: aNumber**

```
imaginary := aNumber
```

**real**

```
^real
```

**real: aNumber**

```
real := aNumber
```

Príklad:

"Definujeme globalnu premennu"

Smalltalk at: #C1 put: (ComplexNr real: 10 imaginary: 55)

C1 real

10

C1 imaginary

55

C1 real: -20

a ComplexNr

C1 real

-20



ComplexNr
real : Number imaginary : Number
\$real:imaginary() real() real:() imaginary() imaginary:()

Pseudo-premenná self

# Ako napísať program v Smalltalku

## Ďalšie metódy

*ComplexNr publicMethods*

### abs

"Answer absolute value for receiver."  
^(real \* real + (imaginary \* imaginary) ) sqrt

### + aComplexNr2

"Adds two complex numbers."  
^ComplexNr  
real: (real + aComplexNr2 real)  
imaginary: (imaginary + aComplexNr2 imaginary)

### inverse

"Answer inverse value. Answer copy of receiver."  
^self copy inverseAtPlace

### printOn: aStream

"Prints receiver in form: 2 + 3i ."  
aStream nextPut: \$(.  
real printOn: aStream.  
aStream nextPutAll: ' + '  
imaginary printOn: aStream.  
aStream nextPutAll: 'i )'

*ComplexNr privateMethods*

### inverseAtPlace

"Answer inverse value. Do it in receiver."  
imaginary := imaginary negated

*Number publicMethods*

### i: aNumber

"Create a complex number."  
^ComplexNr real: self imaginary: aNumber

Metódy môžu byť  
public a private.  
Významné len pre  
programátora.

C1 abs

???

C1

(-20 + 55i )

10 i: 55

(10 + 55i )

(1.56 i: 2) + (14 i: 3.55e-1)

(15.56 + 2.355i )

(1.56 i: 2) + (14 i: 3.55e-1) inverse

(15.56 + 1.645i )

# Ako napísať program v Smalltalku

## Úlohy

**Cvičenie 1.2.:** Napíšte metódu pre násobenie komplexných čísel.

## Správy vs. metódy

*ComplexNr publicMethods*

**+ aComplexNr2**

"Adds two complex numbers."

^ComplexNr

real: (real + aComplexNr2 real)

imaginary: (imaginary + aComplexNr2 imaginary)

Správa vždy vracia nejaký výsledok. Často je to self.

Správa – hovorí objektu, čo má urobiť.

Metóda – definuje ako sa správa vykoná.

Toto  
celé je  
metóda

Správy sú:

- real:imaginary:
- real
- imaginary

## Zapuzdrenie

### Encapsulation

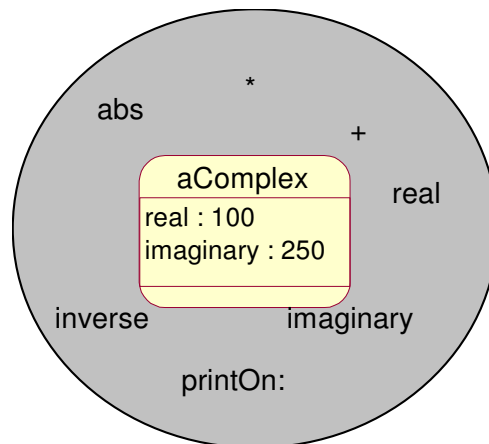
Stav objektu je neprístupný.

S objektom viem spolupracovať len prostredníctvom správ, ktorým rozumie.

Správanie objektov je najdôležitejšie. Nie je dôležitý typ objektu, ale či vie vykonať to, čo potrebujem.

Skryté by mali byť aj jeho vzťahy k iným objektom.

Všetko, čo od objektu požadujem, by mal vedieť vykonať on sám.



# Ako napísať program v Smalltalku

## Hierarchia tried, dedičnosť

Triedy tvoria hierarchiu. Hierarchia tried sa vytvára definovaním nadtriedy pre triedu.

V Smalltalku nie je viacnásobná dedičnosť.

Trieda dedí od svojej nadtriedy:

- stav, t.j. všetky inštančné premenné
- správanie, t.j. inštancia vie vykonať všetko, čo je definované v jej triede a nadtriedach.

Dedičnosť je podstatná črta objektovej orientácie. Systémy bez dedičnosti nie sú objektové.

Dedičnosť značne zvyšuje znovupoužiteľnosť kódu v Smalltalku.

## Príklad Magnitude

**Magnitude** definuje tieto metódy:

<, <=, >, >=, between:and:, min:, max:

Podtriedy musia reimplementovať metódy:

=, <

*Magnitude publicMethods*

**< aMagnitude**

"Answer a Boolean indicating true if the receiver is less than aMagnitude; answer false otherwise."  
self subclassResponsibility

**>= aMagnitude**

"Answer a Boolean indicating true if the receiver is greater than or equal to aMagnitude; answer false otherwise."  
^(self < aMagnitude) not

**Number** definuje napr. tieto metódy:

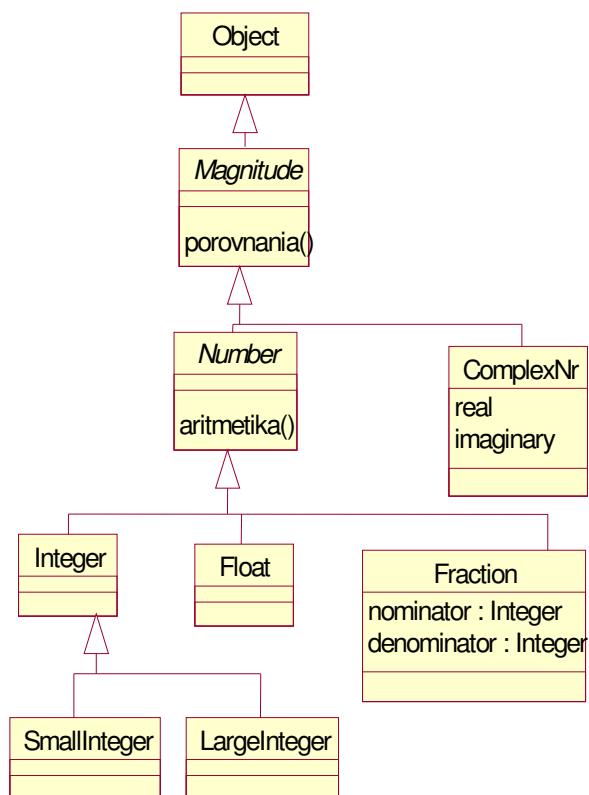
+, -, \*, /, //, \, cos, sin, arcCos, arcSin, ceiling, asInteger, ln, log, negative, positive, sqrt, timesRepeat:, to:by:, ...

Podtriedy musia samozrejme niektoré z nich reimplementovať.

*Number publicMethods*

**+ aNumber**

"Answer a type of Number that is the sum of the receiver and the argument, aNumber."  
self subclassResponsibility



# Ako napísať program v Smalltalku

## arcSin

"Answer an instance of Float which is the arc-sine of the receiver, an angle measured in radians."  
^self asFloat arcSin

Magnitude a Number sú **abstraktné triedy**.

## abs

"Answer a type of Number that is the absolute value of the receiver."  
(self < 0 )  
ifTrue: [^0 - self].  
^self

## SmallInteger redefinuje niektoré metódy:

*SmallInteger publicMethods*

### + aNumber

"Answer a type of Number that is the sum of the receiver and the argument, aNumber.  
Fail if aNumber is not a type of Number."  
<primitive: VMprSmallIntegerPlus>

### < aNumber

"Answer a Boolean indicating true if the receiver is less than the argument, aNumber; otherwise answer false.  
Fail if aNumber is not a type of Number."  
<primitive: VMprSmallIntegerLessThan>

Niektoré metódy sú implementované ako **primitívy**, t.j. nie sú napísané v Smalltalku, ale ich implementácia je súčasťou virtuálneho stroja.

## Vyhľadávanie metód

Čo sa stane vo virtuálnom stroji, keď vykonám 3+4:

- Objektu 3, ktorý je typu SmallInteger je poslaná správa + s argumentom 4.
- Opýta sa príjemcu na jeho triedu. Je to SmallInteger.
- V SmallInteger hľadá metódu pre správu +.
- Vyhľadajú metódu vykoná, v tomto prípade je to primitíva.
- Vrátí výsledok vykonanej metódy.

# Ako napísať program v Smalltalku

Čo sa stane vo virtuálnom stroji, keď vykonám `3 >= 4`:

- Opýta sa príjemcu na jeho triedu. Je to `SmallInteger`.
- V `SmallInteger` hľadá metódu pre správu `>=`. Nenájde ju, pretože trieda `SmallInteger` správu `>=` neimplementuje.
- Opýta sa na nadtriedu pre `SmallInteger`. Je to `Integer`.
- V `Integer` hľadá metódu pre správu `>=`. Nenájde ju.
- Opýta sa na nadtriedu pre `Integer`. Je to `Number`.
- V `Number` hľadá metódu pre správu `>=`. Nenájde ju.
- Opýta sa na nadtriedu pre `Number`. Je to `Magnitude`.
- V `Magnitude` hľadá metódu pre správu `>=`. Vykoná nájdenú metódu.
- Pri vykonaní metódy `>=` sa vykoná `3 < 4` a vráti sa negácia výsledku. Posiela sa metóda `<` objektu `self` čo je `smallInteger 3`. `SmallInteger` má implementovanú metódu `<`, preto ju vykoná.

## Pseudo-premenná `super`

`ComplexNr new`

(nil + nil)

Máme problém: Pre `ComplexNr` chceme vytvoriť triednu metódu `new` tak, aby vytvorená inštancia mala nastavené hodnoty `real` a `imaginary` na 0.

*ComplexNr publicMethods*

**new**

"Create instance, which instance variables are initialized."

^(super new)

real: 0;

imaginary: 0

Pseudo-premenná `super` definuje, že poslaná metóda sa má začať vyhľadávať od nadtriedy. Takto je možné redefinovať rovnako-pomenované metódy v podtriedach.

`ComplexNr new`

(0 + 0i)

# Ako napísať program v Smalltalku

## Úlohy

**Cvičenie 1.3.:** 3+4

Čo je message selector, message, receiver, argument, result?

**Cvičenie 1.4.:** array at: 1 put: 'hello'

Čo je message selector, message, receiver, argument, result?

**Cvičenie 1.5.:** Čo vráti nasledujúci kód:

```
| array |  
array := #(#xyz #(1 2 3) 'hello' -12.34)  
^array at: 2
```

**Cvičenie 1.6.:** Čo vráti nasledujúci kód:

```
3 + 4 * 5 + 2 + 1
```

**Cvičenie 1.7.:** Ktoré zátvorky sú nadbytočné:

```
(( 3 + 4 ) + ( 2 * 2 ) + ( 2 * 3 ))
```

**Cvičenie 1.8.:** Čo vráti:

```
5 + 4.8 truncated
```

**Cvičenie 1.9.:** Čo vráti:

```
5.3 @ 5.4 extent: 6.0 truncated @ 7
```

**Cvičenie 1.10.:** Vedeli by ste vysvetliť, prečo v Smalltalku nie je implementovaná priorita aritmetických operácií?

**Cvičenie 1.11.:** Prepíšte tento kód do Smalltalku:

```
value := 2;  
power := 1;  
WHILE value <= bound DO  
  BEGIN  
    value := value * 2;  
    power := power + 1;  
  END
```

**Cvičenie 1.12.:** Implementujte metódu isPalindromic do String. Text je palindromický, ak je rovnaký odpredu aj odzadu.

**Cvičenie 1.13.:** Naprogramujte metódu = pre ComplexNr.

**Cvičenie 1.14.:** Naprogramujte metódu + pre ComplexNr tak, aby vedelo pripočítať každé číslo.

# Ako napísať program v Smalltalku

**Cvičenie 1.15.:** ComplexNr sme urobili podtriedou Magnitude. Je to v poriadku? Ktorá trieda by mala byť nadtriedou ComplexNr?

**Cvičenie 1.16.:** Čo je self pre inštačnú metódu triedy ComplexNr a triednu metódu triedy ComplexNr?

# Program Smalltalk

## Program Smalltalk

### Vytváranie a rušenie objektov

Objekty vytvárame:

- poslaním vhodnej správy triede:

```
ComplexNr new
```

```
ComplexNr real: 10 imaginary: 25
```

- inými správami:

```
10 i: 25
```

- literálne:

```
123.4
```

```
##one 2 'three' #(1 1 1 1)
```

V konečnom dôsledku je každý objekt vytvorený poslaním správ `new` alebo `new:` triede.

Objekt neviem zničiť. Nepotrebné objekty odstraňuje smetiár (garbage collector).

### Globálne premenné

Objekty môžu byť globálne prístupné cez meno.

Napr.:

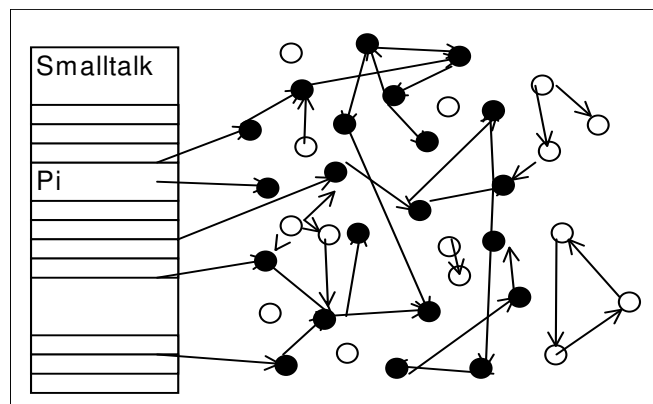
```
Smalltalk at: #Pi put: 3.14159
```

definuje globálnu premennú `Pi`. Túto môžem použiť kdekoľvek v programe, napr.:

```
ComplexNr real: Pi imaginary: Pi * Pi
```

(3.14159 + 9.8695877281i)

Meno globálnej premennej začína veľkým písmenom.



# Program Smalltalk

## Garbage collector

Smetiar zisťuje, ktoré objekty sú referencované z globálnych premenných. Tie objekty, ktoré nie sú referencované z globálnych premenných sú zničené, t.j. pamäť, ktorú zaberali, sa využije na nové objekty.

Smetiar beží na pozadí. Niekedy však môže prerušiť aj práve bežiaci program, ak je nedostatok pamäti pre nové objekty.

## Program

Smalltalk sa skladá z dvoch súborov:

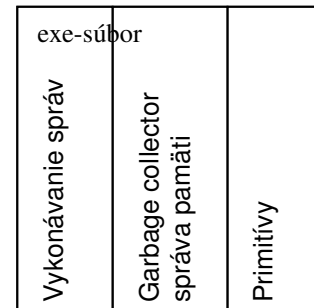
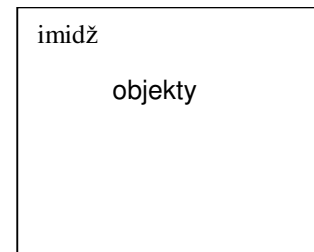
- vykonateľný súbor
- imidž

Vykonateľný súbor obsahuje:

- virtuálny stroj, ktorý vie vykonávať správy
- garbage collector a správa pamäti
- primitívne metódy

Imidž obsahuje:

- všetky objekty, t.j. aj triedy.



# Program Smalltalk

## Identita objektov, referencie

Údaje v Smalltalku je možné zdieľať.

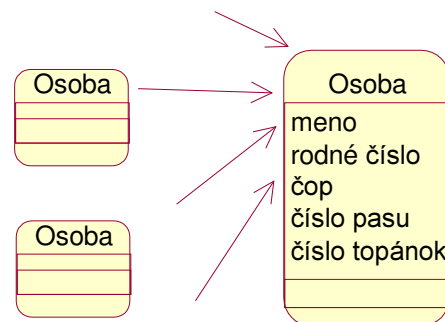
Údaj nie je súčasťou iného údaju. Údaj je referencovaný iným údajom. Zároveň môže byť referencovaný viacerými údajmi.

Referencie sú vlastne pointre v c, t.j. narábanie s adresou údaju.

Objekty majú svoju identitu. Nech sa vnútorný stav objektu akokoľvek mení, ide stále o jeden identický údaj.

Napr. objekt typu *Osoba* môže mať zmenené meno alebo identifikačné údaje, stále ide o tú istú osobu.

Či sú objekty identické, t.j. či referencie ukazujú na ten istý objekt zistujeme `==` resp. `~~`.

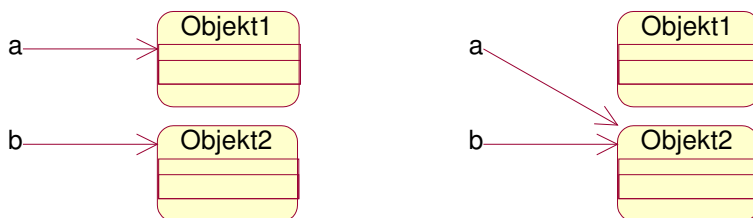


## Priradovanie

`a := b`

Tento príkaz má inakší význam v Smalltalku oproti Pascalu alebo c.

Nekopíruje hodnotu `b` do `a`, ale spôsobí, že `a` referencuje ten istý objekt ako `b`.



## Základné princípy objektovej orientácie

1. Objekty majú definovaný svoj stav a správanie prostredníctvom svojich tried.
2. Triedy môžu tvoriť hierarchiu.
3. Podtriedy dedia od svojich nadtried stav a správanie.

## Ďalší princíp: Smalltalk je netypový

Niekde sa nedefinuje a teda ani nekontroluje typ argumentov alebo príjemcov správ.

Typovosť je hlavne definovaná správaním: objekt musí vedieť vykonať správu, ktorú mu pošlem.

To, ktorá metóda sa vykoná sa rozhodne vždy až v runtime.

Táto vlastnosť značne uľahčuje programovanie aplikácií.

# Vývojové prostredie

## Vývojové prostredie

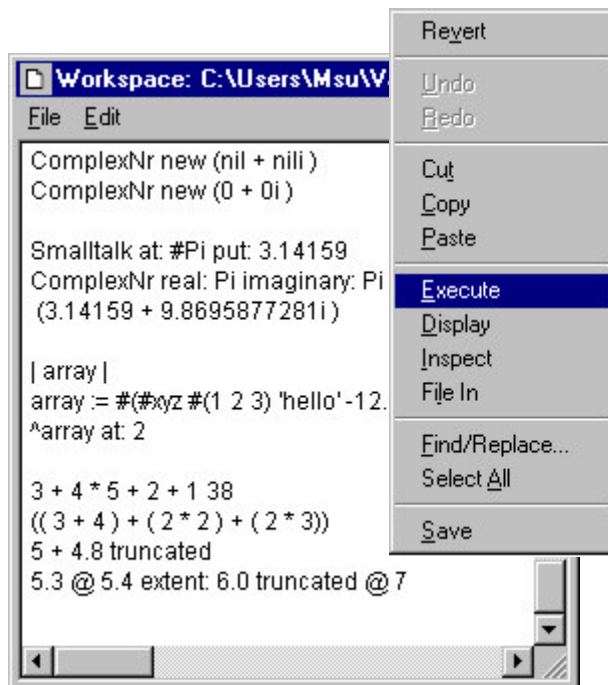
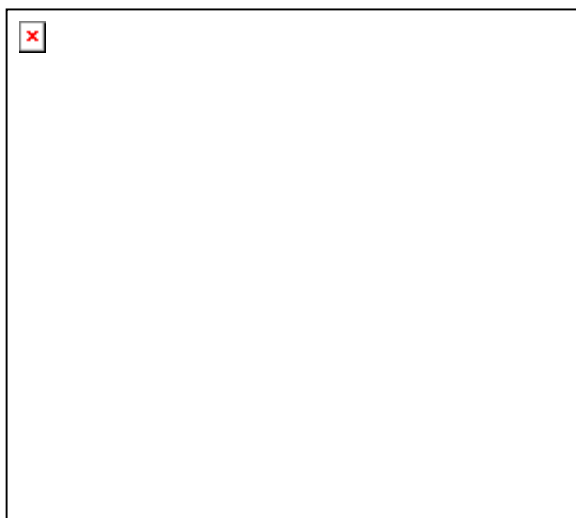
### Workspace

Slúži na vykonávanie príkazov.

Execute .. vykoná označený kód

Display .. vykoná označený kód a zobrazí výsledok

Inspect .. vykoná označený kód a otvorí inšpektor pre výsledok.



### Inspector

Umožňuje prezerať inštančné premenné objektu.

### Transcript

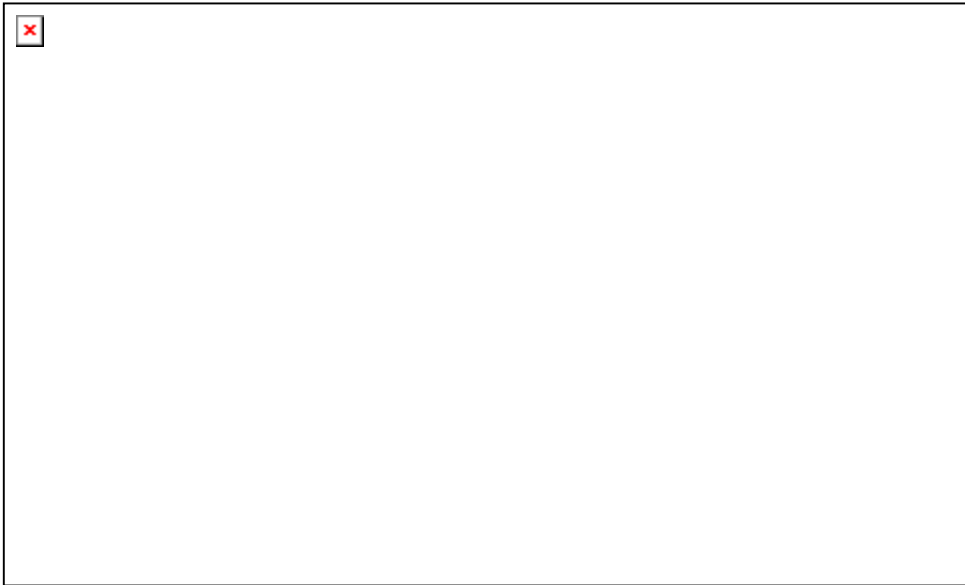
Slúži na vypisovanie správ.

Transcript cr; show: 'Takto sa pise do transcriptu'; cr



# Vývojové prostredie

## Class Browser



Umožňuje:

- prezerat' triedy a ich metódy
- definovat' nové triedy
- programovat' nové metódy

Nová trieda sa naprogramuje:

```
Magnitude subclass: #ComplexNr  
instanceVariableNames: "  
classVariableNames: "  
poolDictionaries: "
```

Nová metóda:

- vybrat' triedu
- vybrat' alebo vytvorit' protokol
- napísať metódu
- save/accept

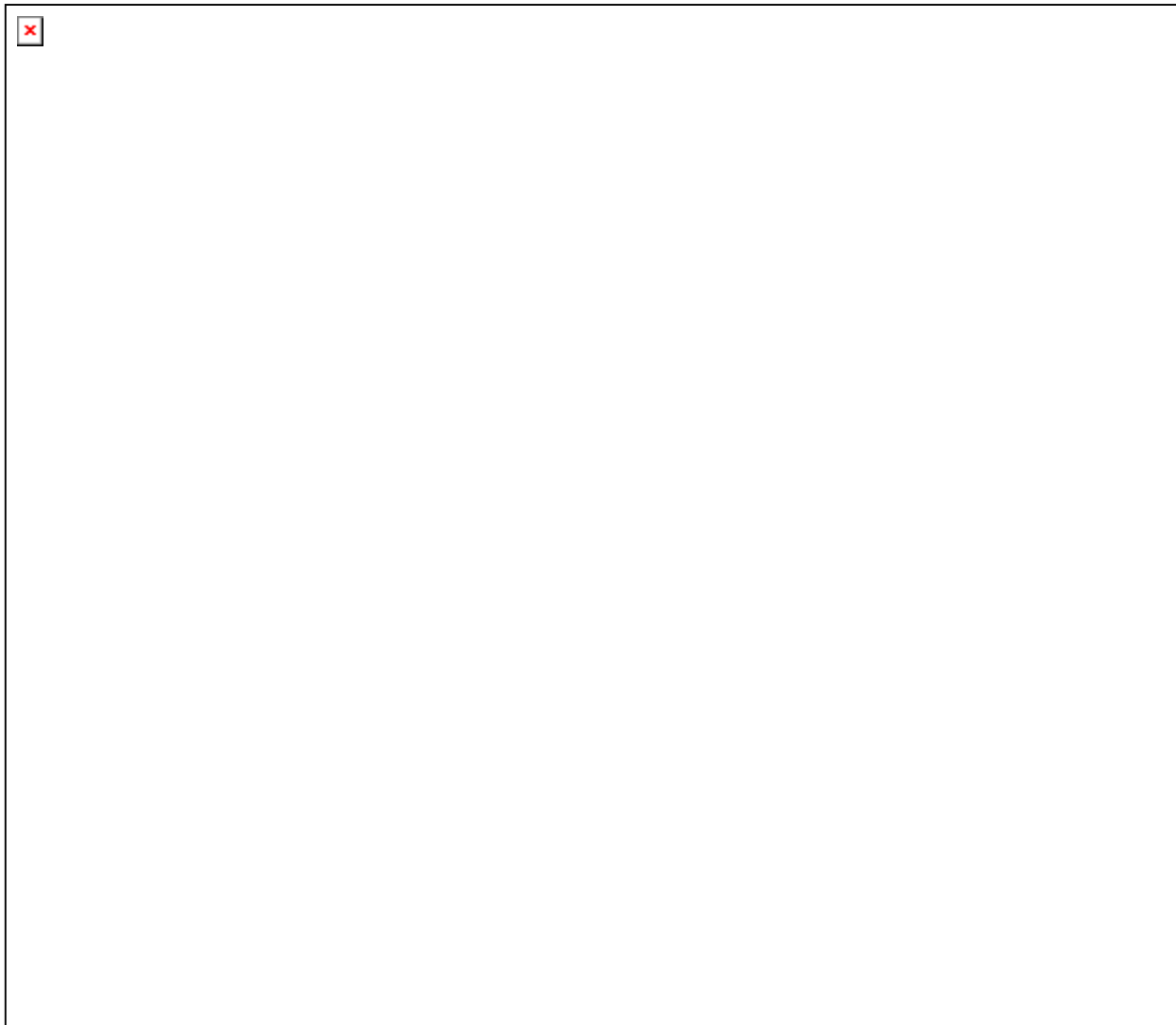
### File in / file out

Slúži na exportovanie kódu do súboru v textovom formáte, resp. na importovaní kódu do imidžu.

Exportovať sa môže jednotlivá metóda, trieda alebo viacej tried.

# Vývojové prostredie

## Debugger



Slúži na odladovanie chýb.

Je možné:

- prezerat' stack správ
- inšpektovat' príjemcov a argumenty správ
- krokovat' program
- zmenit' metódu

Odkrokovat' kód môžeme:

- vložení breakpointu
- poslaním správy halt ľubovlnému objektu

# Základné triedy

## Základné triedy

### Object

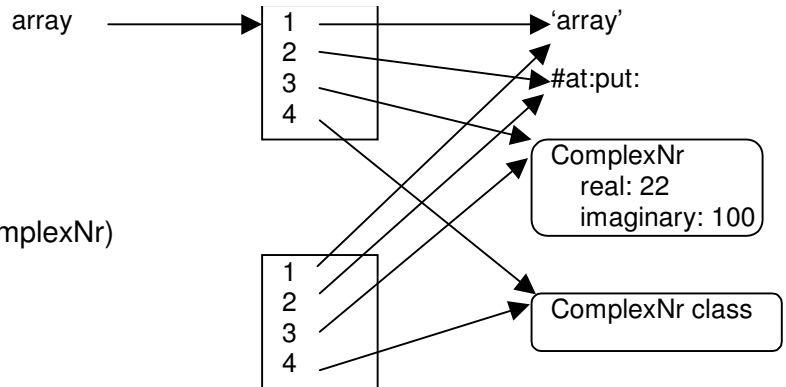
Definuje základné správanie pre všetky objekty. Nemá žiadny stav.

C1 class	ComplexNr
C1 isKindOf: Magnitude	true
C1 isMemberOf: Magnitude	false
C1 respondsTo: #min:	true
C1 isNil	false
C1 notNil	true
C1 isInteger	false
C1 yourself	(-20 + 55i )
C1 halt	otvorí sa debugger
C1 error: 'Nastala nejaka chyba'	vznikne chyba
C1 shouldNotImplement	vznikne chyba
C1 subclassResponsibility	vznikne chyba
C1 inspect	otvorí sa inšpektor
(20 i: 50) = (20 i: 50)	false
C1 == C1	true
C1 copy	(-20 + 55i )
C1 shallowCopy	(-20 + 55i )
C1 printString	'(-20 + 55i )'
C1 storeString	'((ComplexNr basicNew) instVarAt: 1 put: -20; instVarAt: 2 put: 55; yourself)'
C1 perform: #abs	58.5234995535981
C1 perform: #+ with: (3 i: 10)	(-17 + 65i )
C1 become: (100 i: 320). C1	(100 + 320i )

# Základné triedy

## Kopírovanie

```
| array |  
array := Array with: 'string'  
  with: #at:put:  
  with: (22 i: 100)  
  with: ComplexNr.  
^array copy ('string' #at:put: (22 + 100i ) ComplexNr)
```



array copy

array shallowCopy

array deepCopy

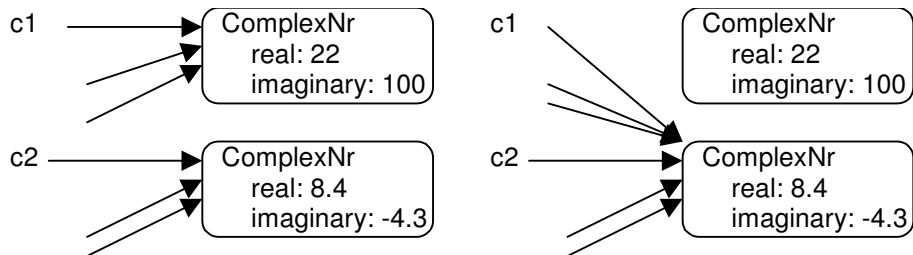
(array at: 1) == (array shallowCopy at: 1)

(array at: 1) == (array deepCopy at: 1)

## Become

```
| c1 c2 array |  
c1 := (22 i: 100).  
c2 := (8.4 i: -4.3).  
array := Array with: c1 with: c1 with: c2 with: c2.  
c1 become: c2.  
^array
```

((8.4 + -4.3i) (8.4 + -4.3i) (8.4 + -4.3i) (8.4 + -4.3i))



# Základné triedy

## UndefinedObject

Má jedinú inštanciu: nil.

Redefinuje: isNil, notNil, printOn:, storeOn:.

*Object publicMethods*

### isNil

"Answer a Boolean which is true if the receiver class inherits from UndefinedObject, and false otherwise."

"Answer false, since Object does not inherit from UndefinedObject"

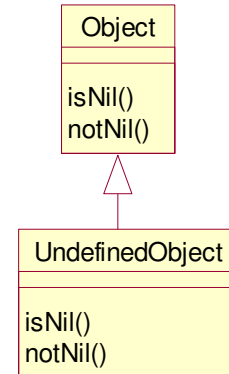
^false

*UndefinedObject publicMethods*

### isNil

"Answer true indicating that the receiver is the object nil."

^true



## Bloky

"30 factorial"

| result mul |

result := 1.

mul := 0.

[ mul := mul + 1.

mul <= 30 ] whileTrue: [ result := result \* mul].

^result

26525285981219105863630848000000

"30 factorial"

| result mul b1 b2 |

result := 1.

mul := 0.

b1 := [ mul := mul + 1.

mul <= 30 ].

b2 := [ result := result \* mul].

b1 whileTrue: b2.

^result

blok je nepomenovaná funkcia

blok je objekt typu

BlockContextTemplate

26525285981219105863630848000000

| b1 |

b1 := [ 2 + 3 ].

^b1 value

| factorialBlock |

factorialBlock := [:n |

n=1

ifTrue: [1]

ifFalse: [ n \* (factorialBlock value: n-1) ] ].

factorialBlock value: 30

26525285981219105863630848000000

**Cvičenie 1.17.:** Napíšte kód na výpočet n-tého člena postupnosti  $f(n)=f(n-1)+1/n$ .

# Základné triedy

## True, False

Majú jediné inštancie true a false.

(1<2)	true
(1<2) not	false
(1<2) & (5<4)	false
(1<2)   (5<4)	true
(1<2) eqv: (5<4)	false
(1<2) xor: (5<4)	true
(1<2) or: [5<4]	true
(1<2) or: [Transcript show: '5<4'. 5<4]	true
(1<2) and: [Transcript show: '5<4'. 5<4]	false
(1<2) ifTrue: ['plati']	'plati'
(1<2) ifFalse: ['neplati']	nil
(1<2) ifTrue: ['plati'] ifFalse: ['neplati']	'plati'

### *False publicMethods*

**ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock**

"Since the receiver is false, answer the value of  
falseAlternativeBlock."  
^falseAlternativeBlock value

### *True publicMethods*

**ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock**

"Since the receiver is true, answer the value of  
trueAlternativeBlock."  
^trueAlternativeBlock value

**Cvičenie 1.18.:** Ako je implementovaná metóda and: ?

# Základné triedy

## Čísla

Čísla sú objekty. Majú svoje triedy, v ktorých majú definované metódy.

Trieda **Magnitude** uľahčuje programovanie všetkých objektov, ktoré majú definované usporiadanie. Podtriedy musia redefinovať metódy  $<$  a  $=$ .

*Magnitude publicMethods*

**>= aMagnitude**

$\wedge$ (self < aMagnitude) not

*Number publicMethods*

**+ aNumber**

self subclassResponsibility

**abs**

(self < 0 )  
ifTrue: [ $\wedge$ 0 - self].  
 $\wedge$ self

**sin**

$\wedge$ self asFloat sin

*SmallInteger publicMethods*

**+ aNumber**

<primitive: VMprSmallIntegerPlus>  
self primitiveErrorCode = PrimErrInvalidClass  
ifTrue: [ $\wedge$ aNumber + self].  
 $\wedge$ self primitiveFailed

*Fraction publicMethods*

**+ aNumber**

$\wedge$ ((numerator \* aNumber denominator) +  
(denominator \* aNumber numerator)) /  
(denominator \* aNumber denominator)

*Float publicMethods*

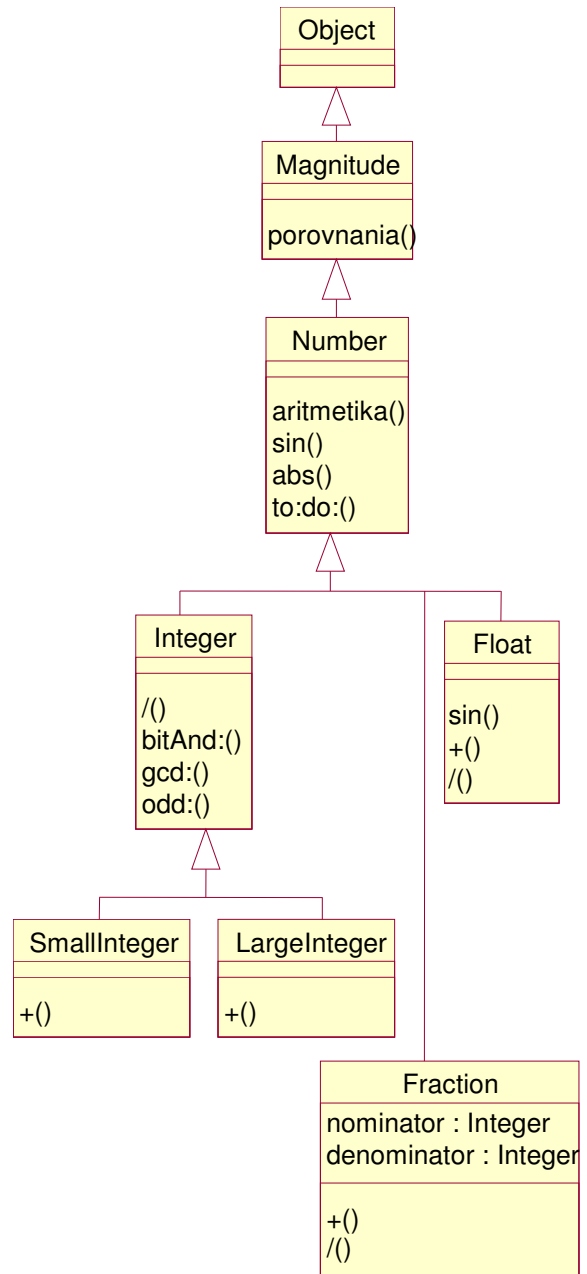
**+ aNumber**

<primitive: VMprFloatPlus>  
self primitiveErrorCode = PrimErrInvalidClass  
ifTrue: [ $\wedge$ aNumber + self].  
 $\wedge$ self primitiveFailed

**sin**

<primitive: VMprFloatSin>  
 $\wedge$ self primitiveFailed

Celé čísla sú neobmedzené.



# Základné triedy

1/2 + (1/3)	(5/6)
1/4 + (1/4)	(1/2)
1/3 + (2/3)	1
3/7 + 1	(10/7)
(5/6) class	Fraction
(12/6) class	SmallInteger
1.0e-10 class	Float
2r110101	53
16rFFFF	65535
8r0.17e2	23.4375
3.0/5	0.6
8/17.0 4.	70588235294118e-1
125 quo: 8	15
-125 quo: 8	-15
125 // 8	15
-125 // 8	-16
-125 rem: 8	-5
-125 \ 8	3
125.78 truncated	125
125.78 rounded	126
3.14159 sin	2.65358979335273e-6
1 arcSin	1.5707963267949
1 exp	2.71828182845904
100 log: 10	2.0
20 gcd: 30	10
56 lcm: 48	336
Float pi	3.14159265358979
50 bitAnd: 24	16
50 bitInvert + 1	-50

**Cvičenie 1.19.:** Čo treba urobiť, aby fungovalo  $(2 i: 3) + 1$  ?

*ComplexNr publicMethods*

**+ aComplexNr2**

"Adds two complex numbers."

^ComplexNr

real: (real + aComplexNr2 real)

imaginary: (imaginary + aComplexNr2 imaginary)

**Cvičenie 1.20.:** Čo sa stane, ak vykonáme  $1 + (2 i: 3)$  ?

# Základné triedy

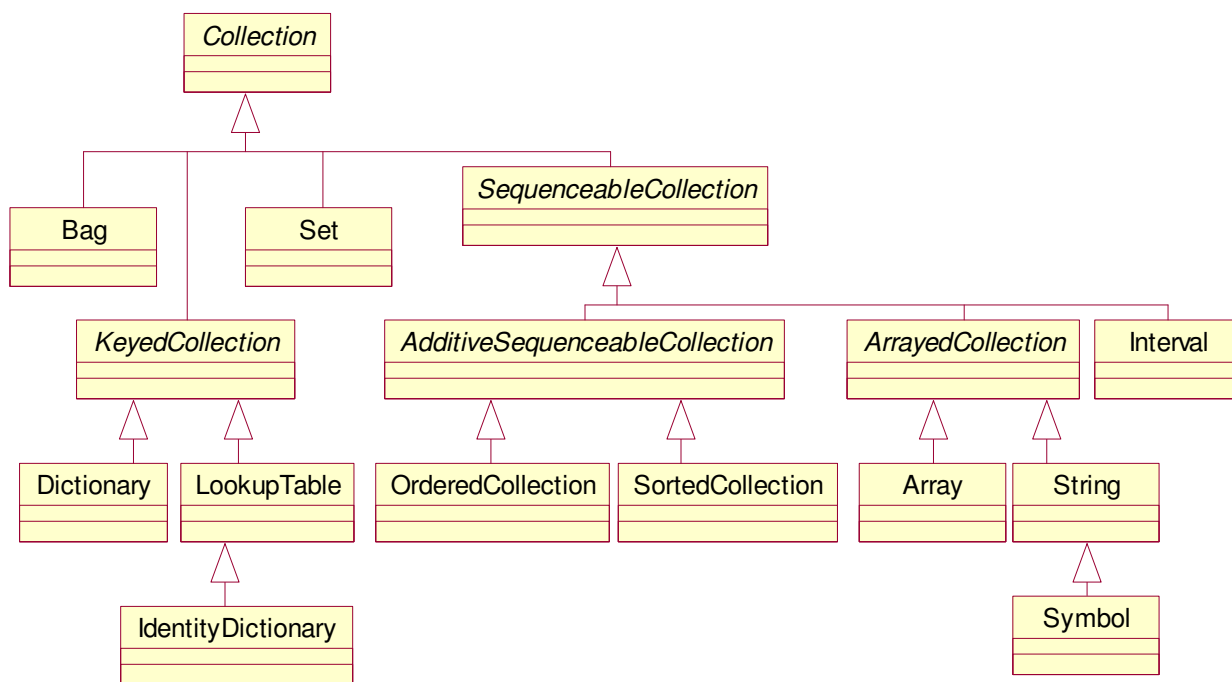
## Collections

Kolekcie sú jedným s najzákladnejších typov v Smalltalku.

Slúžia ako kontajner pre iné objekty.

Rôzne typy poskytujú bohatú funkčnosť:

- množina
- viac-násobná množina (bag)
- pole
- slovník
- usporiadaná kolekcia
- zotriedená kolekcia
- reťazec
- symbol



# Základné triedy

## Set

Umožňuje pridávať prvky.

Nepridá viacej rovnakých prvkov.

Smalltalk at: #SET1 put: (Set new: 10).

SET1 size	0
SET1 add: (2 i: 3); add: (5 i: -10); add: (12 i: 17); yourself	Set((5 + -10i ) (2 + 3i ) (12 + 17i ) )
SET1 add: (2 i: 3); yourself	Set((5 + -10i ) (2 + 3i ) (12 + 17i ) )
SET1 size	3
SET1 includes: (2 i: 3)	true
SET1 occurrencesOf: (2 i: 3)	1

Pre ComplexNr sme implementovali metódu =.

## Bag

Umožňuje pridávať prvky.

Pridá viacej rovnakých prvkov.

Smalltalk at: #BAG1 put: (Bag new: 10). Bag()

BAG1 size	0
BAG1 add: (2 i: 3); add: (5 i: -10); add: (12 i: 17); yourself	Bag((2 + 3i ) (12 + 17i ) (5 + -10i ) )
BAG1 add: (2 i: 3); yourself	Bag((2 + 3i ) (2 + 3i ) (12 + 17i ) (5 + -10i ) )
BAG1 size	4
BAG1 includes: (2 i: 3)	true
BAG1 occurrencesOf: (2 i: 3)	2

# Základné triedy

## Dictionary

Umožňuje pridávať prvky a pristupovať k nim cez kľúče.

Kľúč môže byť ľubovoľný objekt. Iba jeden rovnaký kľúč môže byť v dictionary.

Smalltalk at: #DICT1 put: (Dictionary new: 10) Dictionary()

DICT1 size	0
DICT1 at: #first put: (2@3 extent: 5@5); at: #second put: (10 i: 5.5); at: Date today put: 'dnes bol dobry den'; yourself	Dictionary((10 + 5.5i) 'dnes bol dobry den' 2 @ 3 corner: 7 @ 8)
DICT1 at: #second	(10 + 5.5i)
DICT1 at: #second put: (20 i: -50); yourself	Dictionary((20 + -50i) 'dnes bol dobry den' 2 @ 3 corner: 7 @ 8)
DICT1 at: #second	(20 + -50i)
DICT1 size	3
DICT1 includes: (20 i: -50)	true
DICT1 includes: (10 i: 5.5)	false
DICT1 occurrencesOf: (20 i: -50)	1

## OrderedCollection

Umožňuje pridávať prvky. Prvky sú usporiadané, metóda add: pridáva vždy na koniec kolekcie.

Môže byť viacej rovnakých prvkov. Je možné pristupovať podľa číselného kľúča.

Smalltalk at: #ORD1 put: (OrderedCollection new: 10). OrderedCollection()

ORD1 size	0
ORD1 add: (2 i: 3); add: (5 i: -10); add: (12 i: 17); yourself	OrderedCollection((2 + 3i) (5 + -10i) (12 + 17i))
ORD1 add: (2 i: 3); yourself	OrderedCollection((2 + 3i) (5 + -10i) (12 + 17i) (2 + 3i))
ORD1 size	4
ORD1 includes: (2 i: 3)	true
ORD1 occurrencesOf: (2 i: 3)	2
ORD1 addFirst: (1@1); yourself	OrderedCollection(1 @ 1 (2 + 3i) (5 + -10i) (12 + 17i) (2 + 3i))
ORD1 first	1 @ 1
ORD1 last (2 + 3i)	(2 + 3i)
ORD1 removeFirst; yourself	OrderedCollection((2 + 3i) (5 + -10i) (12 + 17i) (2 + 3i))
ORD1 at: 2	(2 + 3i)

# Základné triedy

## SortedCollection

Umožňuje pridávať prvky. Prvky sú zotriedené podľa pravidla daného programátorom.

Môže byť viacej rovnakých prvkov. Je možné pristupovať podľa číselného kľúča.

Smalltalk at: #SOR1

```
put: (SortedCollection  
      sortBlock: [:x :y | x real < y real]). SortedCollection()
```

SOR1 add: (2 i: 3); add: (5 i: -10); add: (-12 i: 17); yourself	SortedCollection((-12 + 17i) (2 + 3i) (5 + -10i) )
SOR1 add: (2 i: 3); yourself	SortedCollection((-12 + 17i) (2 + 3i) (2 + 3i) (5 + -10i) )
SOR1 includes: (2 i: 3)	true
SOR1 occurrencesOf: (2 i: 3)	2
SOR1 first	(-12 + 17i)
SOR1 last	(5 + -10i)
SOR1 removeLast; yourself	SortedCollection((-12 + 17i) (2 + 3i) (2 + 3i) )
SOR1 at: 1	(-12 + 17i)

## Array

Neumožňuje pridávať prvky, t.j. nie je možné meniť veľkosť poľa.

Prístup je podľa číselného kľúča.

Smalltalk at: #ARRAY1 put: (Array new: 3) (nil nil nil)

ARRAY1 at: 1 put: (2 i: 3); at: 2 put: (10@20); yourself	((2 + 3i) 10 @ 20 nil)
ARRAY1 at: 3 put: 1234; at: 1 put: (5 i: 5); yourself	((5 + 5i) 10 @ 20 1234)
ARRAY1 at: 2	10 @ 20
ARRAY1 last	1234

# Základné triedy

## Porovnanie kolekcii

	môže zväčšovať veľkosť	prístup cez kľúč (at:)	typ kľúča	pridávanie podľa kľúča (at:put:)	pridávanie (add:)	rovnaký objekt môže byť viac krát	usporiadané	zotriedené
Set	áno				áno			
Bag	áno				áno	áno		
Dictionary	áno	áno	object	áno		áno		
OrderedCollection	áno	áno	číslo		áno	áno	áno	
SortedCollection	áno	áno	číslo		áno	áno	áno	áno
Array		áno	číslo	áno		áno	áno	

## Sekvenčný prístup k prvkom

Metódy, ktoré umožňujú sekvenčný prístup k prvkom kolekcie. Štandardný protokol pre všetky kolekcie.

SET1

Set((2 + 3i) (12 + 17i) (5 + -10i) )

SET1 do: [:e | Transcript show: e printString]

(2 + 3i)(12 + 17i)(5 + -10i)

| sum |

sum := 0 i: 0.

SET1 do: [:e | sum := sum + e].

sum

(19 + 10i)

SET1 select: [:e | e abs > 10]

Set((5 + -10i) (12 + 17i) )

SET1 reject: [:e | e imaginary < 0]

Set((2 + 3i) (12 + 17i) )

SET1 collect: [:e | e abs rounded]

Set(11 4 21)

SET1 detect: [:e | e abs > 10] ifNone: [nil]

(12 + 17i)

SET1 inject: (0 i: 0) into: [:sum :e | sum + e]

(19 + 10i)

SET1 inject: (OrderedCollection with: (1 i: 0)) into: [:col :e | col add: (col last \* e); yourself]

OrderedCollection((1 + 0i) (2 + 3i) (-27 + 70i) (565 + 620i) )

# Základné triedy

## Variable subclass

ArrayedCollection **variableSubclass**: #Array  
instanceVariableNames: ”  
classVariableNames: ”  
poolDictionaries: ”

Array new: 5  
má 5 indexovaných premenných.

Objekt môže mať:

- pomenované inštančné premenné
- vopred nedefinovaný počet indexovaných premenných

## Identity vs. equality

Set môže mať najviac jeden rovnaký prvok.

IdentitySet môže mať najviac jeden identický prvok.

Podobne Dictionary a IdentityDictionary.

Mohol by existovať aj IdentityBag.

## Hash

Sety a Dictionary majú efektívne ukladanie prvkov tak, aby bolo možné rýchlo nájsť daný prvok v kolekcii. T.j. aby includes: a remove: nemuselo prejsť sekvenčne cez celú kolekciu.

Pre každý prvok je definovaný hash, čo je číslo z konečného intervalu. Toto číslo určuje na akej pozícii v kolekcii sa prvok nachádza.

# Ako napísať program v Smalltalku

## Verzie Smalltalku

Enfin

GNU Smalltalk

Zadarmo. Prístupné zdrojové texty. Pre experimentovanie.

HP Distributed Smalltalk

Rozširuje VisualWorks o prostredie pre distribuované aplikácie.

IBM Smalltalk

Smalltalk extrahovaný z VisualAge.

Objectworks Smalltalk

Veľmi pekná implementácia, veľmi veľa tried.

VisualWorks

ParcPlace. Rozšírenie ObjectWorks o vizuálne vytváranie UI.

Parts Workbench

Smalltalk/V

Smalltalk/X

SmalltalkAgents

VisualAge

IBM. Má viacpoužívateľské vývojové prostredie ENVY. Vizuálne vytváranie aplikácie s vizuálnym vytváraním vzťahov medzi objektami UI.

# Ako napísať program v Smalltalku

## Kód

Tu je kód použitý v školení.

Kód pre ComplexNr

# Vývojové prostredia

## **Slovník**

Trieda – class

Nadtrieda – superclass

Podtrieda – subclass

Príjemca – receiver

Metóda – method

Správa – message

Vyhľadávanie metód – method lookup

Zapuzdrenie – encapsulation

Kaskádna správa - cascaded message

Enumeration - vypočítavanie