

Pharo

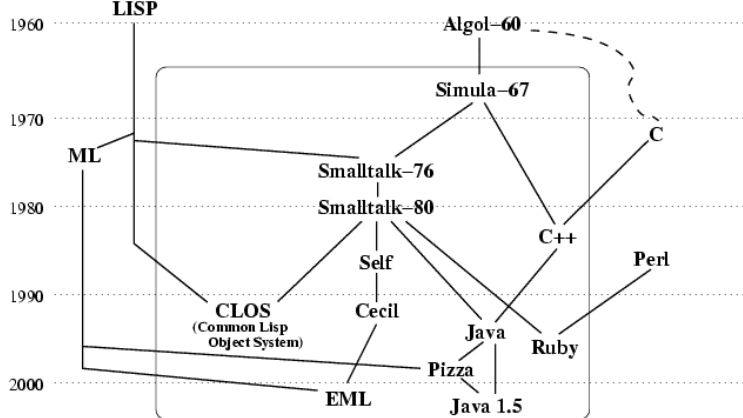
Bazirano na Pharo MOOC

Igor Dejanović

June 7, 2021

- ▶ Objektno-orijentisani dinamički reflektivni jezik.
- ▶ Xerox PARC - Alan Kay, Dan Ingalls i drugi - tokom 70-ih.
- ▶ Uticao na razvoj Actor model obrasca.
- ▶ Nastao pod uticajem Simule (prvi OO jezik, Norwegian Computing Center u Oslu - 60-te).
- ▶ Jedan od najuticajnijih jezika.
- ▶ Napredni koncepti: sve je objekat, razmena poruka, "živ" sistem, virtualna mašina.
- ▶ Konstrukcionistački pristup programiranju.
- ▶ [https://en.wikipedia.org/wiki/PARC_\(company\)](https://en.wikipedia.org/wiki/PARC_(company))
- ▶ <https://en.wikipedia.org/wiki/Smalltalk>
- ▶ Razvijen u par dana 1971 godine (*Smalltalk-71*) zbog opklade (Alan Kay).
- ▶ Kasnija verzija *Smalltalk-72* je korišćena u istraživanjima.
- ▶ *Smalltalk-76* - nasleđivanje klasa, razvojno okruženje
- ▶ Najpoznatija verzija Smalltalk-80 - meta-klase. Prva verzija dostupna van PARC-a (Apple, HP, DEC, UC Berkeley)

- ▶ Standardizovan - ANSI 1998
- ▶ Komercijalne
 - ▶ *Smalltalk-80* (Xerox PARC)
 - ▶ *VisualWorks* (ParcPlace Systems, prodato 1999 firmi Cincom)
 - ▶ *IBM VisualAge* - napušteno u korist Java. Današnji Eclipse je započeo kao VisualAge Smalltalk okruženje. Jedno vreme je i Java podrška bila implementirana u Smalltalk-u.
- ▶ FLOSS:
 - ▶ Squeak (moderna verzija Smalltalk-80) - Apple -> Disney -> HP Labs -> SAP Labs -> Y Combinator
 - ▶ Pharo - fork Squeak-a (2008) sa ciljem upotrebe u istraživanju i komercijalnim projektima (Pharo consortium, Pharo association)
 - ▶ Amber Smalltalk - Smalltalk u JavaScript-u
- ▶ <https://en.wikipedia.org/wiki/VisualAge>
- ▶ <https://en.wikipedia.org/wiki/Squeak>



<http://courses.cs.washington.edu/courses/cse341/04wi/lectures/16-smalltalk-intro.html>

- ▶ Pravi OO jezik ("sve je objekat") + IDE!
- ▶ Inspirisan Smalltalk-om
- ▶ Aktivna zajednica
- ▶ "Živ" sistem.
- ▶ Jednostavan i moćan objektni model

- ▶ Radi na Mac OSX, Linux, iOS, Android, Windows, Pi.
- ▶ 100% MIT
- ▶ Pregled nekih osobina Pharo IDE i jezika
- ▶ Preporučena upotreba Pharo Launcher alata
 - ▶ Instalacija i upravljanje slikama i virtualnim mašinama
- ▶ Alternativno, može se koristiti Zeroconf skripta:

```
# 64bit version
```

```
mkdir pharo
```

```
cd pharo
```

```
curl -L https://get.pharo.org/64/ | bash
```

```
# or if curl is not available:
```

```
wget -O- https://get.pharo.org/64 | bash
```

```
# 32bit version
```

```
curl -L https://get.pharo.org | bash
```

```
# or if curl is not available:
```

```
wget -O- https://get.pharo.org | bash
```

pharo

Pharo8.0-32bit-a153e04

Pharo.changes

Pharo.image

pharo-ui

pharo-vm

- ▶ `pharo-vm` - Pharo virtuelna mašina (*OS-specific*)
- ▶ `Pharo.image` - Perzistirano stanje/objekti
- ▶ `Pharo...sources` - Izvorni kod izdanja
- ▶ `Pharo.changes` - Promene u izvornom kodu od početka upotrebe
- ▶ `Pharo.image` i `Pharo.changes` su fajlovi gde dolazi do promena
- ▶ `Pharo.changes` se menja kada menjamo kod
- ▶ `Pharo.image` se menja kada eksplicitno tražimo perzistenciju stanja

Staje na jedan slajd:

exampleWithNumber: x

"A method that illustrates every part of Smalltalk metho

<menu>

| y |

true & false not & (nil isNil) ifFalse: [self halt].

y := self size + super size.

#\$a #a "a" 1 1.0)

do: [:each |

Transcript show: (each class name);

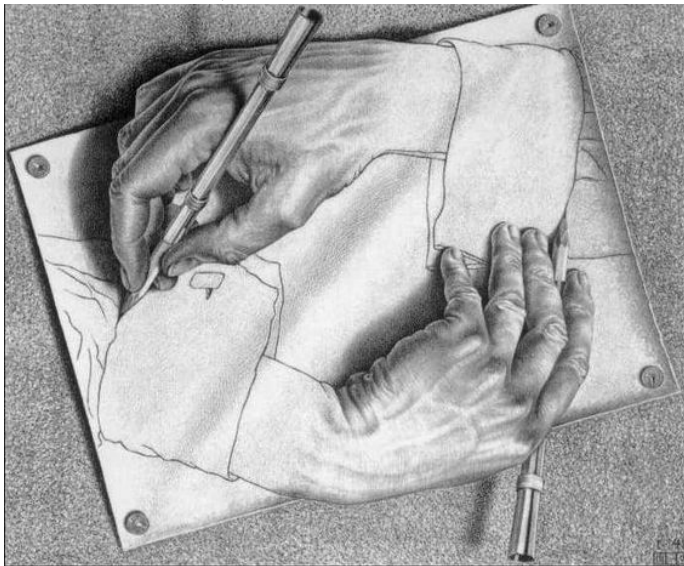
show: (each printString);

show: ' '].

^x < y

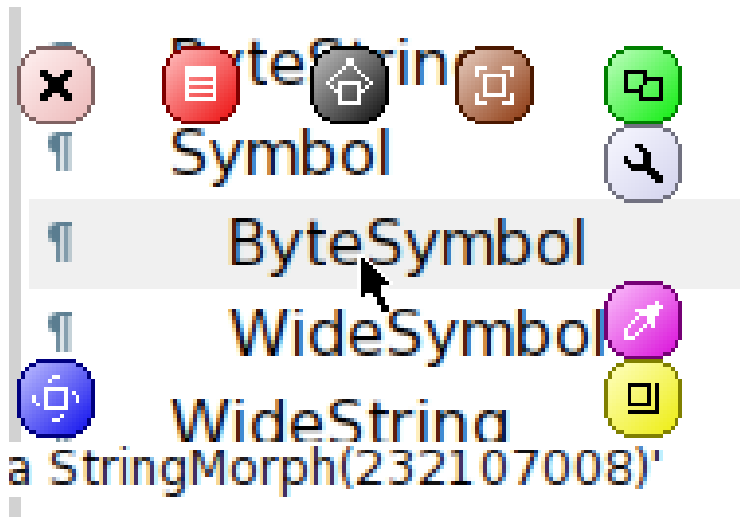
- ▶ Dinamički tipiziran
- ▶ Sve je objekat tj. instanca klase
- ▶ Sve metode su javne i virtualne
- ▶ Svi atributi su zaštićeni
- ▶ Jednostruko nasleđivanje (*Single inheritance*)
- ▶ Sve je napisano u Pharo!

- ▶ Jednostavna sintaksa/model za pristup svemu.



- ▶ Pharo nije “crna kutija”.
- ▶ Sve što vidite su objekti sa kojima možete stupiti u interakciju i menjati ih “naživo”.

- ▶ Program koji se razvija je nerazdvojni deo razvojnog okruženja.
- ▶ Npr. Alt+Shift+Click -> Halo hendleri!



- ▶ *Objekti*: mouse pointer, booleans, arrays, numbers, strings, windows, files, sound, url, socket, font, text, streams...

- ▶ *Poruke (messages)*: `size`, `+`, `at:put:`, `do:`, `collect:`, `ifTrue:ifFalse:...`
- ▶ Poruke predstavljaju **nameru** (šta treba uraditi). Metode opisuju kako treba nešto uraditi.
- ▶ Objekat koji prima poruku zovemo **prijemnikom** (*receiver*).
- ▶ Blokovi su vrsta anonimnih metoda.

4 timesRepeat:

```
[ Transcript show: 'Hello World!']
```

- ▶ Blokovi se navode unutar `[]`.
- ▶ Sve je objekat tj. instanca klase.
 - ▶ Klase i poruke su takođe objekti.
- ▶ Svo procesiranje se obavlja **razmenom poruka** (*message passing*) između objekata.
- ▶ Koristimo izraz **slanje poruke** jer:
 - ▶ metode se određuju dinamički
 - ▶ **kasno povezivanje** (*late binding*), samo virtuelni pozivi
- ▶ Postoji **samo jedan** mehanizam za pretragu metoda za sve objekte.

- ▶ Atributi instanci (*instance variables*) su zaštićeni (*protected*):
 - ▶ Privatni za objekat
 - ▶ Dostupni podklasama
- ▶ Metode su javne (*public*) i virtualne.
- ▶ Jednostruko nasleđivanje klasa.

Date today

Date today + 3 days

2 + 3

(point1 x * point2 y) - (point1 y * point2 x)

Obavlja se slanjem poruke drugom objektu

10@20

Nova instanca klase Point se kreira:

- ▶ slanjem poruke @
- ▶ objektu 10 (SmallInteger)

- ▶ sa argumentom 20 (SmallInteger)

```
'Pharo', 'is cool!'
```

```
=> 'Pharo is cool!'
```

Novi string se kreira spajanjem dva stringa tako što:

- ▶ se šalje poruka ,
- ▶ stringu 'Pharo'
- ▶ sa parametrom 'is cool!'

Slanjem poruke new ili new: klasi

```
Monster new
```

```
=> aMonster
```

U prethodnom primeru Monster je ime klase a new je poruka koja se šalje ovoj klasi. Rezultat je nova instanca klase Monster.
Kreiranje niza dužine 6.

```
Array new: 6
```

```
=> #(nil nil nil nil nil nil)
```

Slanjem poruke klasi izvšava se metoda klase (*class method*).

Tamagoshi withHunger: 10

- ▶ Bez konstruktora
- ▶ Bez statičkih metoda
- ▶ Bez deklaracije tipova
- ▶ Bez interfejsa
- ▶ Bez *package/private/protected* modifikatora
- ▶ Bez parametrizovanih tipova
- ▶ Bez boxing-a
- ▶ Ali i dalje vrlo moćan jezik!

```
'Hello World' asMorph openInWindow
```

Šaljemo poruku asMorph stringu Hello World i dobijamo grafički element (*Morph*). Dobijenom grafičkom elementu šaljemo poruku openInWindow da bi ga prikazali u prozoru.

```
(ZnEasy getPng: 'http://pharo.org/web/files/pharo.png') asM
```

- ▶ ZnEasy je ime klase. Klase su globalno dostupne i nazivi počinju sa velikim slovom.
- ▶ getPng: je poruka koju šaljemo klasi ZnEasy. Ova poruka ima argument. U ovom slučaju to je string `'http://pharo.org/web/files/pharo.png'`
 - ▶ Poruke koje imaju argumente se pišu sa : na kraju naziva i mogu biti višesložne. Ovakve poruke nazivamo keyword message.
- ▶ Poruka asMorph šalje se objektu koji vraća poruka getPng:. Ovo je obična unarna poruka bez argumenata.
- ▶ Poruka openInWindow se šalje objektu koji vraća poruka asMorph.
- ▶ Ove dve unarne poruke se primenjuju s leva na desno.

Vrsta	Primer
Komentar	"Ovo je komentar"
Karakter	\$c \$# \$@
String	'Ovo je string'
Simbol (jedinstveni string)	#prvi #+
Literal niz	#{23 56 89}
Integer	45, 2r10100
Real	1.5, 6.03e-34, 4, 2.4e7

Vrsta	Primer
Boolean	true, false (instanca True i False)
Undefined	nil (instanca UndefinedObject)
Point	10@120

- ▶ Deklaracija privremene varijable: | var |
- ▶ Dodela vrednosti varijabli: var := aValue
- ▶ Separator iskaza: obj1 message1. obj2 message2.
- ▶ Povratak vrednosti iz metode: ^ aValue

- ▶ Blokovi (leksička zatvorenja ili anonimne metode)

```
[ :x | x + 2 ] value: 5
```

```
=> 7
```

1. Unarne poruke:

- ▶ Sintaksa: receiver selector

- ▶ Primeri:

```
9 squared
```

```
Date today
```

2. Binarne poruke:

- ▶ Sintaksa: receiver selector argument

- ▶ Primeri:

```
2 + 3
```

```
3 @ 4
```

3. *Keyword* poruke:

- ▶ Sintaksa: receiver key1: arg1 key2: arg2

- ▶ Primeri:

```
2 between: 10 and: 20
```

```
5 to: 10 do: [ :i | Transcript show: i ]
```


receiver selector

Primer:

10000 factorial

Šaljemo poruku factorial objektu 10000.

receiver selector argument

Primer:

1 + 3

Šaljemo poruku + objektu 1 sa parametrom 3.

receiver keyword1: arg1 keyword2: arg2

Ekvivalentno u Javi ili C-like jezicima:

receiver.keyword1keyword2(arg1, arg2)

U Javi

postman.send(mail, recipient);

```
postman . send ( mail , recipient );
```

```
postman send mail recipient
```

```
postman send mail to recipient
```

U Pharo/Smalltalk-u

```
postman send: mail to: recipient
```

ZnClient new

```
url: 'https://en.wikipedia.org/w/index.php';
```

```
queryAt:'title' put:'Pharo';
```

```
queryAt:'action' put:'edit';
```

```
get
```

- ▶ new je unarna poruka koja se šalje klasi ZnClient
- ▶ queryAt:put: je *keyword* poruka sa dva argumenta
- ▶ get je unarna poruka
- ▶ ; je specijalan operator koji zovemo kaskada (*cascade*) - šaljemo poruku istom objektu primaocu.

- ▶ Uslovi
- ▶ Petlje
- ▶ Iteracije
- ▶ Konkurencija
- ▶ ...

factorial

"Answer the factorial of the receiver."

```
self = 0 ifTrue: [^ 1].
```

```
self > 0 ifTrue: [^ self * (self - 1) factorial].
```

```
self error: 'Not valid for negative integers'
```

- ▶ ifTrue: je poruka koja se šalje Boolean objektu koji vraća poruka = poslata objektu self sa parametrom 0.
- ▶ Postoje i ifTrue:ifFalse:, ifFalse:ifTrue: i ifFalse:
- ▶ Implementirane su u klasama True i False i možete ih pročitati. Ne postoji ništa specijalno u vezi ovih poruka!

Šta se dešava kada imamo sukcesivne poruke istog tipa?

```
1000 factorial class name  
> 'LargePositiveInteger'
```

ekvivalentno je sa:

```
((1000 factorial) class) name)
```

```
(Msg) > Unary > Binary > Keywords
```

Ova pravila redukuju potrebu za navođenjem zagrada.

```
2 + 3 squared
```

```
> 2 + 9
```

```
> 11
```

- ▶ Prvo unarna squared

- ▶ Zatim binarna +

```
2 raisedTo: 3 + 2
```

```
> 2 raisedTo: 5
```

```
> 32
```

- ▶ Prvo binarna +
- ▶ Zatim *keyword* raisedTo:

```
Color gray - Color white = Color black
```

```
> aGray - aWhite = aBlack  
> aBlack = aBlack  
> true
```

- ▶ Prvo unarne
- ▶ Zatim binarne s leva na desno: - pa onda =

```
1 class maxVal + 1  
> 1073741824
```

- ▶ unarna class, unarna maxVal, binarna +

```
1 class  
> SmallInteger
```

```
1 class maxVal  
> 1073741823
```

```
1 class maxVal + 1  
> 1073741824
```

```
(1 class maxVal + 1) class  
> LargePositiveInteger
```

```
0@0 extent: 100@100 bottomRight  
> Message not understood  
> 100 does not understand bottomRight
```

Moramo koristiti zgrade:

```
(0@0 extent: 100@100) bottomRight  
> (aPoint extent: anotherPoint) bottomRight  
> aRectangle bottomRight  
> 100@100
```

Samo poruke:



- ▶ je poruka (nije operacija), ne postoji specijalno definisani prioritet
- ▶ možemo je redefinisati za različite domene
- ▶ Jednostavnost
- ▶ Ograničenje: nemamo definisan matematički prioritet operacija

$$3 + 2 * 10$$

$$> 5 * 10$$

$$> 50$$

Moramo pisati sa zagradama:

$$3 + (2 * 10)$$

$$> 3 + 20$$

$$> 23$$

$$1/3 + 2/3$$

$$> 7/3 / 3$$

$$> 7/9$$

Moramo pisati:

$(1/3) + (2/3)$

>1

. je separator:

expression1.

expression2.

expression3

Primer:

Transcript cr.

Transcript show: 1.

Transcript show: 2

- ▶ . je separator a ne terminacija.
- ▶ Nema potrebe da se stavlja na kraju niza izraza.
- ▶ Ne stavlja se posle deklaracije privremenih promenljivih.

```
| macNode pcNode |
```

```
macNode := Workstation withName: #mac.
```

```
macNode sendPacket: 'Hello World'
```



```
|c|  
c := OrderedCollection new.  
c add: 1.  
c add: 2
```

Ekvivalentno je sa:

```
OrderedCollection new  
  add: 1;  
  add: 2
```

- ▶ add: 2 se šalje istom prijemniku poruke add: 1 a to je objekat vraćen porukom new.

```
fct(x) = x*x + 3
```

```
fct := [ :x | x * x + 3 ]
```

```
fct(2)
```

```
fct value: 2
```

- ▶ Anonimne metode

```
[ :each | Transcript show: each abs printString; cr ]
```

- ▶ Leksička “zatvorenja” (*closures*)
- ▶ Takođe su objekti:
 - ▶ Mogu se proslediti kao argumenti poruka
 - ▶ Mogu se dodeliti varijablama
 - ▶ Mogu biti povratne vrednosti metoda

```
#(1 2 -4 -86) do: [ :each | Transcript show: each abs printString; cr ]  
> 1  
> 2  
> 4  
> 86
```

- ▶ Pišu se unutar []
- ▶ Mogu imati parametre - navode se kao simboli pre | (:each)
- ▶ U ovom primeru blok se evaluira za svaki element niza. :each dobija redom vrednosti niza.
- ▶ value: poruka se koristi za evaluaciju bloka.

(1/0)

-> Greška

Ali nema greške pri definiciji bloka:

- ▶ Definicija bloka ne izvršava kod
- ▶ Definicija bloka “zamrzava” izračunavanje definisano telom bloka.

[1/0]

> [1/0]

[1/0] .

1 + 2

> 3

Obavlja se eksplicitno slanjem poruke value.

[2 + 6] value

> 8

[1/0] value

> Error

Blokovi mogu imati argumente (kao i metode):

```
[ :x | x + 2 ]
```

- ▶ `:x` predstavlja argument bloka
- ▶ `x + 2` je telo bloka

```
[ :x | x + 2 ] value: 5  
> 7
```

- ▶ Poruka `value:` izvršava blok sa parametrom 5.
 - ▶ `x` dobija vrednost 5 za vreme izvršavanja bloka.

Evaluacija bloka vraća vrednost poslednjeg izraza u bloku:

```
[ :x |  
  x + 33.  
  x + 2 ] value: 5  
> 7
```

- ▶ Blok se može sačuvati kao vrednost varijable
- ▶ Blok se može evaluirati više puta

```
| add2 |  
add2 := [ :x | x + 2 ].  
add2 value: 5.  
>7
```

```
add2 value: 33  
> 35
```

Primer:

```
[ :x :y | x + y ]
```

:x :y su argumenti bloka.

Kako se izvršava blok sa dva argumenta?

```
[ :x :y | x + y ] ??? 5 7  
> 12
```

```
[ :x :y | x + y ] value: 5 value: 7  
> 12
```

- ▶ value:value: je poruka sa dva argumenta koja se šalje bloku sa parametrima 5 i 7

Blokovi mogu definisati lokalne privremene varijable (kao i metode):

```
Collection>>affect: anObject when: aBoolean  
  self do: [ :index | | args |  
            args := ....  
            aBoolean  
            ifTrue: [ anObject do: args ]  
            ifFalse: [ anObject doDifferently: args ] ].
```

- ▶ | args | definiše privremenu varijablu args
- ▶ args postoji samo za vreme izvršavanja bloka

Kada se ^ izvrši unutar bloka dolazi do povratka iz metode u kojoj je blok definisan:

```
Integer>>factorial  
  "Answer the factorial of the receiver."  
  
  self = 0 ifTrue: [ ^ 1 ].  
  self > 0 ifTrue: [ ^ self * (self 1) factorial ].  
  self error: 'Not valid for negative integers'
```

```
0 factorial
```

```
>1
```

```
42 factorial
```

```
>1405006117752879898543142606244511569936384000000000
```

- ▶ Koristi blokove sa najviše 2 ili 3 parametra
- ▶ Definirati klasu umesto bloka za više parametara

```
1 to: 4 do: [ :i | Transcript << i ]
```

```
> 1
```

```
> 2
```

```
> 3
```

```
> 4
```

- ▶ to:do: je poruka poslata broju (instanci Integer klase)
- ▶ Mnoge druge vrste petlji: timesRepeat:, to:by:do:, whileTrue:, whileFalse:...

```
4 timesRepeat: [self doSomething ]
```

```
0 to: 100 by: 3 do: [ :i | ... ]
```

- ▶ Možete lako napraviti novu vrstu petlje koja se neće razlikovati od systemske.

```
[ ... ] whileTrue: [ ... ]
```

Izvršava argument dok god je vrednost prijmnika true

```
Color >> atLeastAsLuminentAs: aFloat  
  | revisedColor |  
  revisedColor := self.  
  [ revisedColor luminance < aFloat ]  
    whileTrue: [ revisedColor := revisedColor slightlyLighter  
      ^ revisedColor
```

Izvršava blok prijmnik sve dok je vrednost true:

```
[ ... ] whileTrue
```

Analogno, postoje i whileFalse i whileFalse:

Implementirane kao poruke.

Pitamo kolekciju da uradi iteraciju svojih elemenata:


```
#(1 2 -4 -86) do: [ :each | Transcript show: each abs printS  
> 1  
> 2  
> 4  
> 86
```

- ▶ do: - iteracija
- ▶ collect: - iteracija i mapiranje elemenata
- ▶ select: - selekcija elemenata na osnovu predikata
- ▶ reject: - eliminacija elemenata na osnovu predikata
- ▶ detect: - vraća prvi koji zadovoljava uslov
- ▶ detect:ifNone: - vraća prvi koji zadovoljava uslov ili podrazumevanu vrednost ukoliko takvog nema u kolekciji
- ▶ includes: - test da li element pripada kolekciji
- ▶ ... mnogi drugi

```
#(2 3 7) collect: [ :each | each raisedTo: 2 ]  
> #(4 9 49)
```

```
#(2 9 7) detect: [ :i | (i \\ 3) = 0 ]  
> 9
```

```
3 timesRepeat: [ Transcript show: 'Hello' ; cr ].
```

```
Date today + 12 days.
```

```
Point linesOfCode.
```

```
Smalltalk allClasses size.
```

```
Smalltalk allClasses inject: 0 into: [ :sum :each | sum + ea
```

```
VGTCatDemo runDemo.
```

```
SystemNavigation new browseAllSelect:  
    [:m| m primitive isZero and: [m pragmas notEmpty]].
```

▶ true je jedinstvena instanca klase True

▶ false je jedinstvena instanca klase False

- ▶ Klase True i False nasleđuju klasu Boolean

U Pharo Bulovi izrazi nisu ništa specijalno:

- ▶ `& | not`
- ▶ `or: and: - lazy`
- ▶ `xor:`
- ▶ `ifTrue:ifFalse:`
- ▶ `ifFalse:ifTrue:`
- ▶ `...`

```
false & (1 error: 'crazy')  
> an error
```

Argument (1 error: 'crazy') se evaluira jer ova operacija ne koristi "lenju evaluaciju" (*lazy*).

```
false and: [ 1 error: 'crazy' ]  
> false "no error!"
```

Argument [1 error: 'crazy'] se ne evaluira jer nije neophodno za određivanje vrednosti izraza - koristi se "lenja evaluacija".

U Pharo uslovi su poruke koje se šalju Bulovim vrednostima i blokovima.

```
Weather isRaining
```

```
  ifTrue: [ self takeMyUmbrella ]  
  ifFalse: [ self takeMySunglasses ]
```

- ▶ Konceptualno ifTrue:ifFalse je poruka koja se šalje objektu koji ima Bulovu vrednost (ili je true ili je false).
- ▶ Optimizovano od strane kompajlera.

ifTrue: [] i ifTrue: [] ifFalse: [] su različite poruke.

```
LogicalFont>>forceItalicOrOblique
```

```
  self slantValue = 0  
  ifTrue: [ slantValue := 1 ]
```

Analogno, ifFalse: [] i ifFalse: [] ifTrue: [] su različite poruke.

Implementirano za kolekcije.

```
myProtocol
```

```
    isEmpty: [ 'As yet unclassified' ]  
> 'As yet unclassified' ili myProtocol
```

Implementacija:

```
Collection>>isEmpty: aBlock  
^ self isEmpty  
ifTrue: [ aBlock value ]  
ifFalse: [ self ]
```

```
self listItems
```

```
    ifNotEmpty: [ :aList | aList at: index ]  
> element liste na indeksu "index" ili sama lista ukoliko je
```

Implementacija:

```
Collection>>ifNotEmpty: aBlock  
    ^self isEmpty  
        ifTrue: [self]  
        ifFalse: [aBlock cull: self]
```

Kategorije

Klase

Protokoli

Selektori

The screenshot shows a software development environment with the following components:

- Scopes Variables:** A tree view on the left showing the project structure. Red arrows point to "BasicObjects" (Kategorije) and "Point" (Klase).
- History Navigator:** A list of methods in the center. A red arrow points to "accessing" (Protokoli).
- Variables:** A list on the right showing variables "x" and "y". A red arrow points to "x" (Selektori).
- Code Editor:** The main area at the bottom showing the code for the "x" property: `"Answer the x coordinate."` and `^ x`.

At the bottom of the window, there is a status bar with the text "1/4 [1]" and a checkbox labeled "Format as you read" followed by "W" and "+L".

Point

Scoped Variables

History Navigator

object

- Last Modified Me
- Configurations
- Work
- Graphics-Display
- Kernel
 - Kernel
 - BasicObjects
 - Chronology
 - Classes
 - Copying
 - Exceptions

Character

CombinedChar

Margin

Point

Rectangle

Hier. Class ? Com.

-- all --

accessing

arithmetic

comparing

converting

copying

extent functions

geometry

interpolating

point functions

polar coordinates

*

+

-

/

//

<

<=

=

>

>=

\

Object subclass: #Point

instanceVariableNames: 'x y'

classVariableNames: ''

package: 'Kernel-BasicObjects'

1/4 [1]

Format as you read W +L

Excessive number of methods ? x Helpful? ? ?

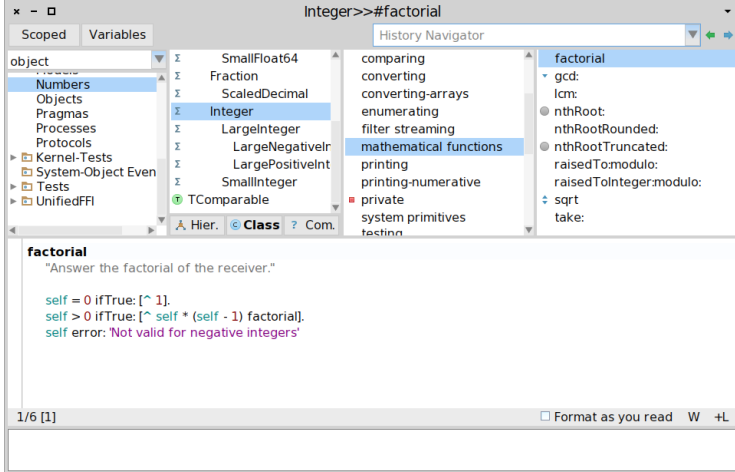
Slanje poruke nadklasi

```
Object subclass: #Point
  instanceVariableNames: 'x y'
  classVariableNames: ''
```

```
package: 'Kernel-BasicObjects'
```

- ▶ Metode su javne (*public*)
- ▶ Metode su virtualne (tj. pronalaze se u vreme izvršavanja)
- ▶ Podrazumevano vraćaju `self`

```
messageSelectorAndArgumentNames  
  "comment stating purpose of message"  
  
  | temporary variable names |  
  statements
```

```

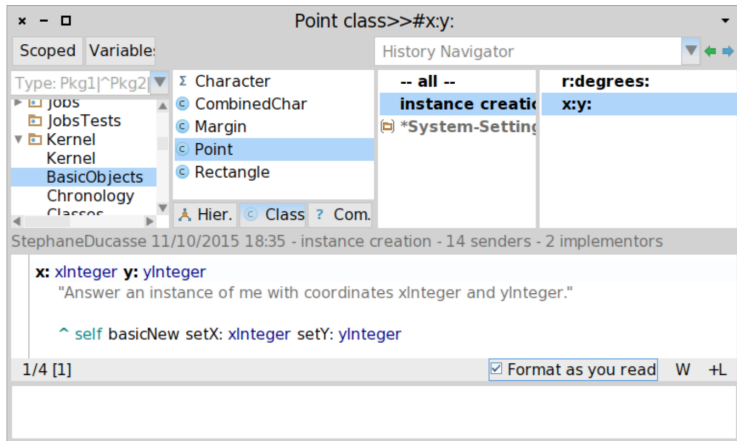
Game >> initializePlayers
  self players
  at: 'tileAction'
  put: ( MITileAction director: self )
  
```

je ekvivalentno sa:

```

Game >> initializePlayers
  self players
  at: 'tileAction'
  put: ( MTileAction director: self ).
  ^ self "< optional"

```



- Dugme Class služi za pregled i definiciju metoda klase.

- ▶ Metode na nivou klase. Odgovor na poruke koje se šalju klasi.

```
Point class >> x: xInteger y: yInteger  
  "Answer an instance of me with coordinates xInteger and yInteger"  
  
  ^ self basicNew setX: xInteger setY: yInteger
```

Dodajemo 2 u skup:

```
Set new add: 2  
>2
```

Rezultat izraza je 2 a ne skup!

```
Set>>add: newObject  
  "Include newObject as one of the receiver's elements, but  
  only if not already present. Answer newObject."  
  [...]  
  ^ newObject
```

- ▶ Metod add: vraća argument a ne objekat

```
Set new add: 2  
>2
```

```
|s|  
s := Set new.  
s add: 2.  
s
```

```
Object >> yourself  
^ self
```

```
Set new  
  add: 2;  
  yourself  
> aSet
```

- ▶ Poruke add: i yourself se šalju skupu
- ▶ kaskada ; vraća objekat koji vraća poruka yourself - u našem slučaju skup.

```
Counter class >> withValue: anInteger  
  self new  
  value: anInteger;  
  yourself
```

- ▶ Counter withValue: 10 vraća Counter klasu umesto njenu instancu.

```
Counter class >> withValue: anInteger  
  self new  
  value: anInteger;  
  yourself
```

je ekvivalentno sa:

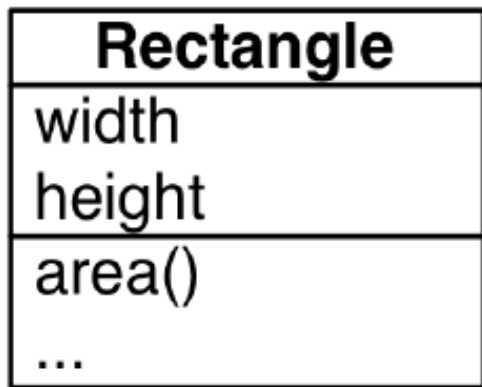
```
Counter class >> withValue: anInteger  
  self new  
  value: anInteger;  
  yourself.  
  ^self
```

Gde je self prijemnik poruke withValue: tj. klasa Counter. ▶

```
Counter class >> withValue: anInteger  
  ^self new  
  value: anInteger;  
  yourself
```

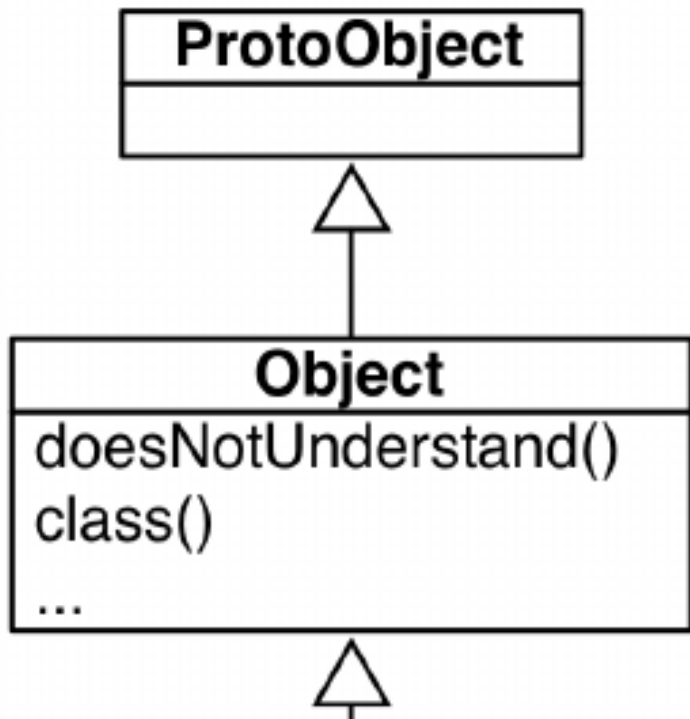
Podklasa:

- ▶ Može da doda stanje i ponašanje
- ▶ Može da koristi stanje i ponašanje nadklase
- ▶ Može da izvrši specijalizaciju i redefiniciju ponašanja nadklase



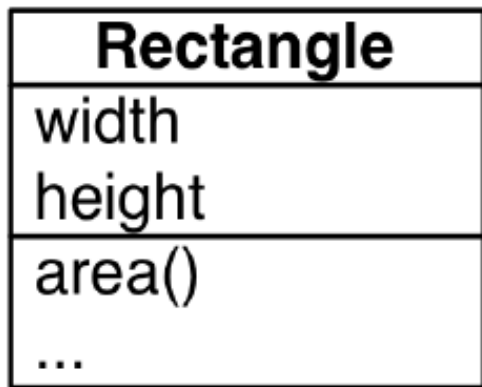
- ▶ Možemo smatrati da je klasa `Object` korenska klasa svake klase.

- ▶ Postoji i klasa `ProtoObject` ali je njena upotreba specijalna pa je nećemo razmatrati.



Nasleđivanje je:

- ▶ Statičko za stanje (u vreme definisanja klase).
- ▶ Dinamičko za ponašanje (u vreme izvršavanja).
- ▶ Dešava se za vreme definicije klase.
- ▶ Izračunava se na osnovu:
 - ▶ Varijabli posmatrane klase.
 - ▶ Varijabli svih nadklasa.

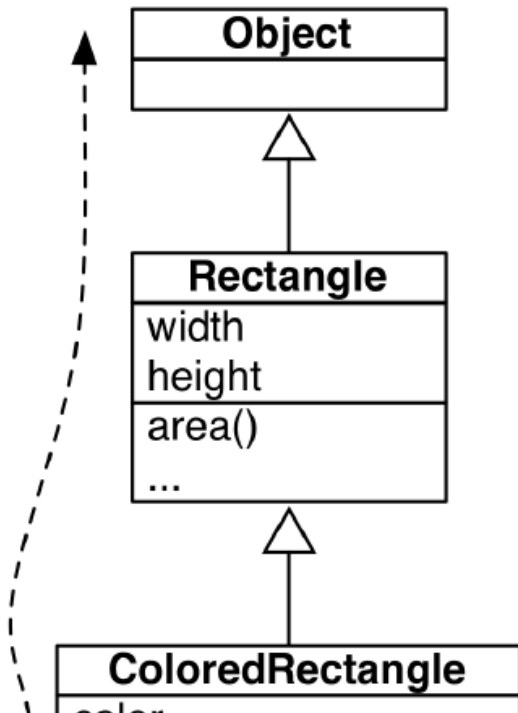


- ▶ Dešava se u vreme izvršavanja

- ▶ Metoda se traži:

- ▶ Počevši od klase objekta prijemnika


- ▶ Zatim u svim nadklasama uz lanac nasleđivanja.

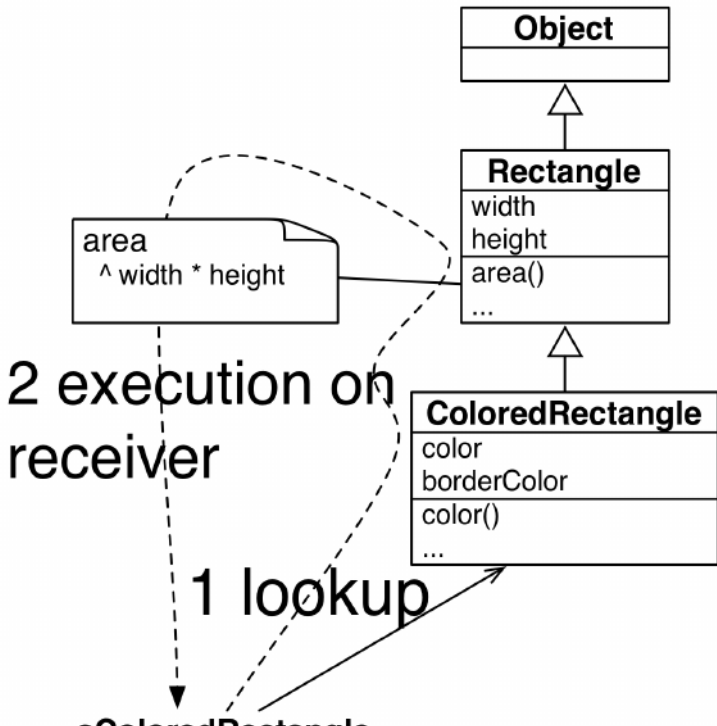


Obrada poruke se obavlja u dva koraka:

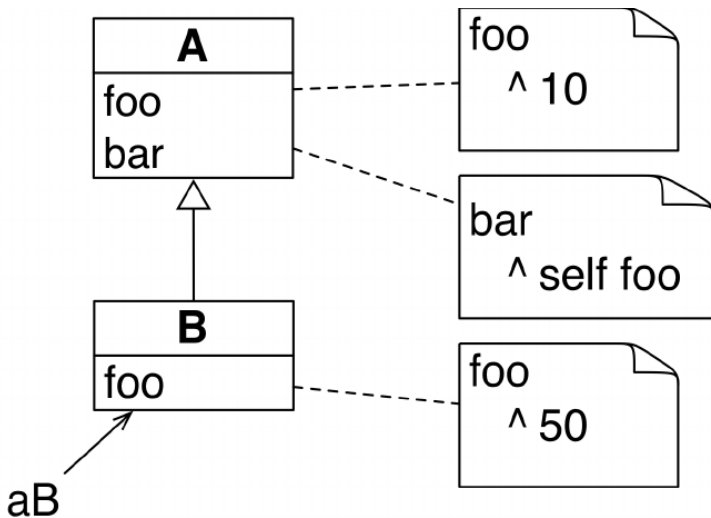
1. Pretraga odgovarajuće metode.

2. Izvršavanje metode na objektu prijemu.

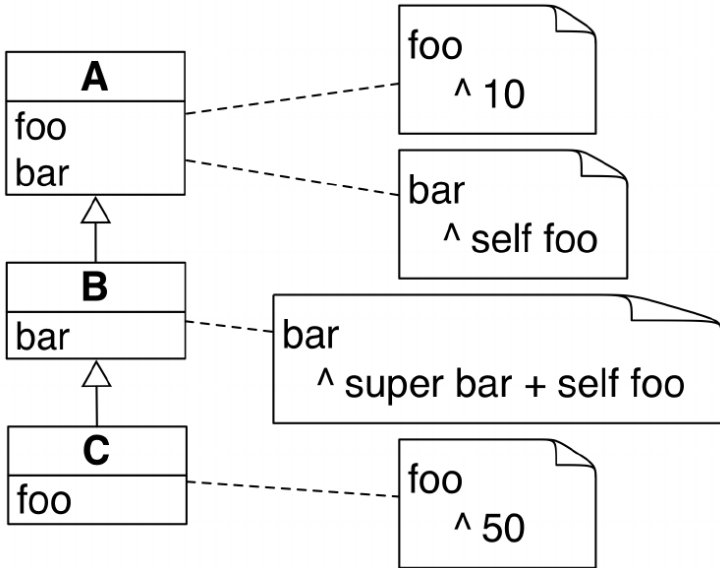
A set of small, faint navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide navigation functions.



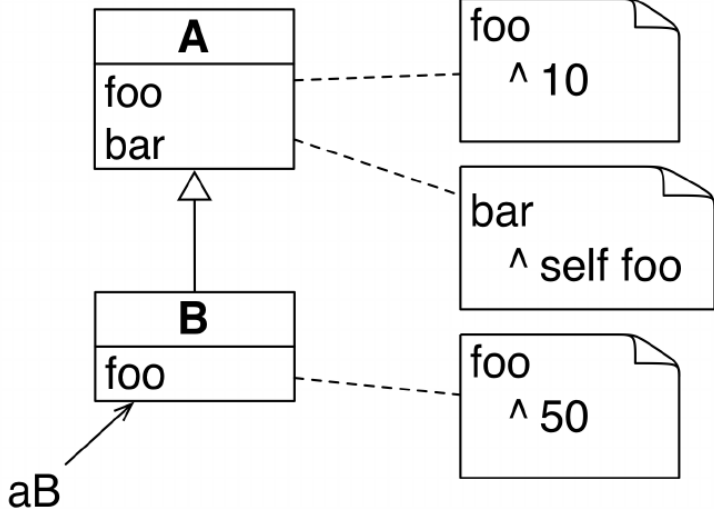
- ▶ `self` ključna reč se koristi u implementaciji metoda i **uvek** predstavlja objekat prijemnik.



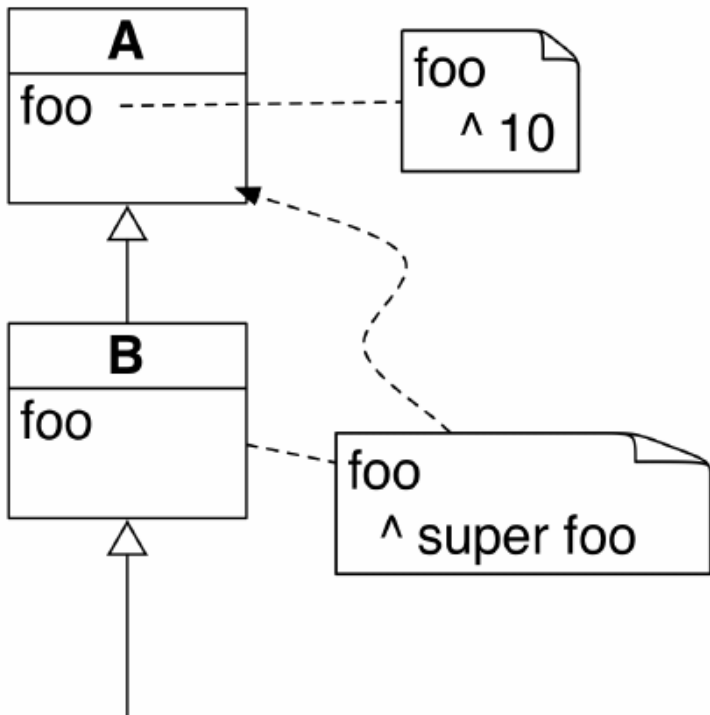
- ▶ Šta je rezultat izraza `A new foo` a šta izraza `B new foo`?
- ▶ Šta je rezultat izraza `A new bar` a šta izraza `B new bar`?



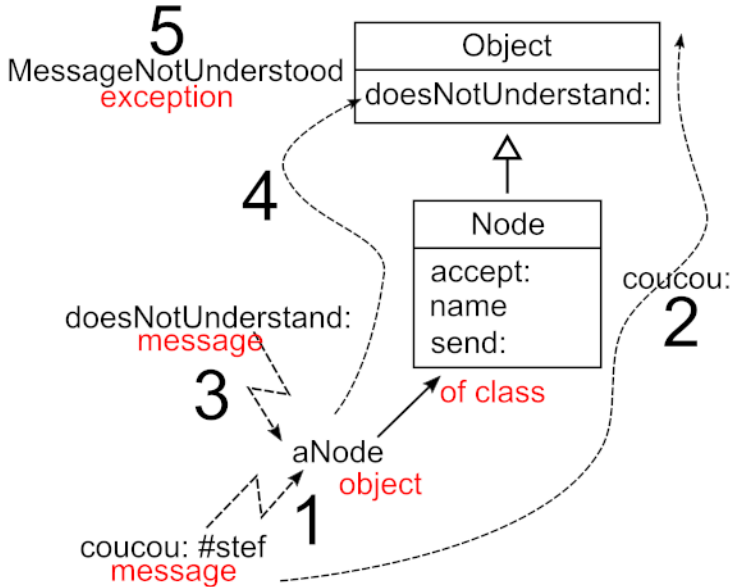
- ▶ `super` predstavlja objekat prijemnik ali pretraga poruka započinje u nadklasi klase u kojoj se `super` nalazi.
- ▶ Šta su rezultati izraza `A new bar`, `B new bar` i `C new bar`?



U metodi `A»bar` kod `^self foo` ne znamo do vremena izvršavanja na koji `foo` se poziv odnosi. To zavisi od klase konkretnog objekta prijemnika.



- ▶ U vreme kompajliranja znamo da metoda `B»foo` referencira `A»foo` putem `super`.
- ▶ Uvek počinjemo pretragu u nadklasi klase koja sadrži metodu koja koristi `super`.
- ▶ Ukoliko metoda nije pronađena standardnim mehanizmom pretrage, prijemniku se šalje poruka `doesNotUnderstand`
- ▶ Podrazumevana implementacija u `Object` klasi signalizira izuzetak `MessageNotUndertood`.



U Pharo Boolean tip ima odličan dizajn:

► `&`, `|`, `not` - *eager*

- ▶ `or:`, `and:` - *lazy*
- ▶ `ifTrue:`, `ifTrue:ifFalse`, ...

U svetu gde imate samo dve vrednosti: `true` i `false` i razmenu poruka

- ▶ kako implementirati `not`?
- ▶ kako implementirati `or`?

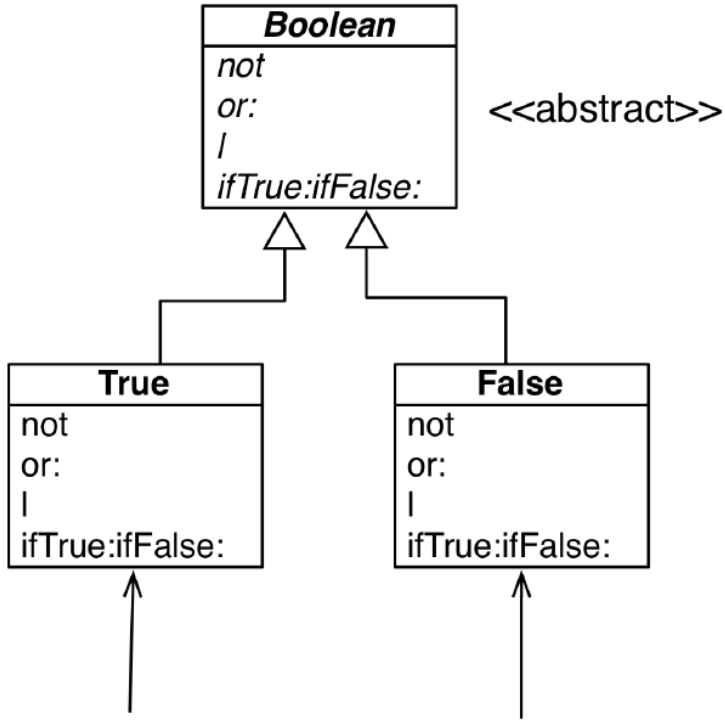
```
false not  
> true
```

```
true not  
> false
```

Rešenje ne koristi uslove.

Uslovi bi svakako morali da budu bazirani na Bulovom tipu.

- ▶ `Boolean` (apstraktna), `True` i `False`
- ▶ `true` je singleton instanca klase `True`
- ▶ `false` je singleton instanca klase `False`



U OOP, izbor iskazujemo:

- ▶ Definisanjem klasa sa kompatibilnim metodama
- ▶ Slanjem poruke instanci takve klase

Primer:

`x open`

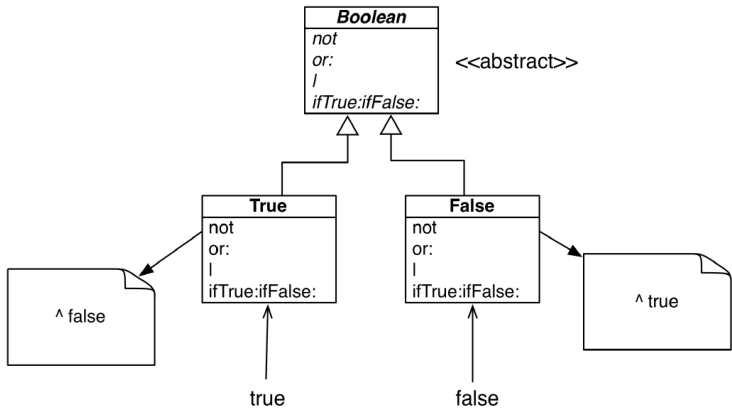
- ▶ `x` može biti fajl, prozor, alat...
- ▶ Metod se selektuje u zavisnosti od klase objekta `x`
- ▶ U Python-u poznato kao *Duck Typing*

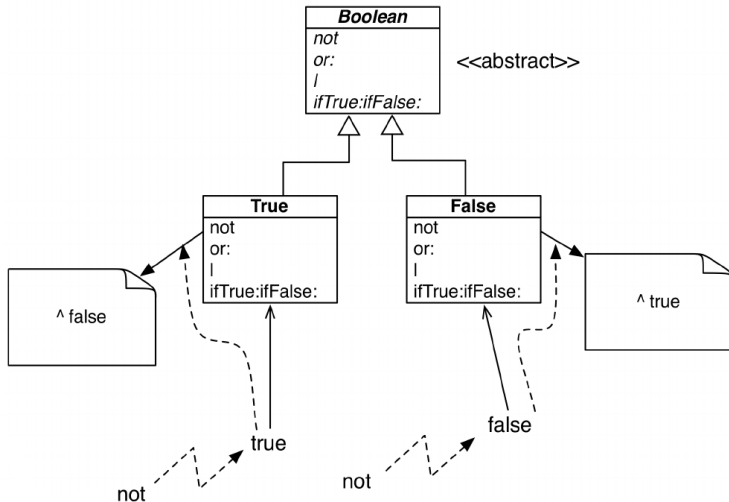
`False >> not`

```
"Negation  answer true since the receiver is false."  
^ true
```

`True >> not`

```
"Negation  answer false since the receiver is true."  
^ false
```



- ▶ Boolean je abstraktna klasa
- ▶ Podklase su True i False koje implementiraju:
 - ▶ logičke operacije `&` i `not`
 - ▶ kontrolne strukture `and:`, `or:`, `ifTrue:`, `ifFalse:`, `ifTrue:ifFalse:`, `ifFalse:ifTrue:`

```
Boolean>>not
```

```
"Abstract method. Negation: Answer true if the receiver is  
false, answer false if the receiver is true."
```

```
self subclassResponsibility
```

```
true | true > true
```

```
true | false > true
```

```
true | anything > true
```

```
false | true > true
```

```
false | false > false
```

```
false | anything > anything
```

```
Boolean >> | aBoolean
```

```
"Abstract method. Evaluating Or: Evaluate the argument.  
Answer true if either the receiver or the argument is true"
```

```
self subclassResponsibility
```

```
false | true > true  
false | false > false  
false | anything > anything
```

```
False >> | aBoolean  
  "Evaluating Or  answer with the argument, aBoolean."  
  ^aBoolean
```

```
true | true > true  
true | false > true  
true | anything > true
```

```
True >> | aBoolean  
  "Evaluating Or  answer true since the receiver is true."  
  ^true
```

A pošto je prijemnik true možemo uraditi sledeće:

```
True >> | aBoolean  
  "Evaluating Or  answer true since the receiver is true."  
  ^self
```

- ▶ Pharo MOOC
- ▶ Video lekcije
- ▶ Pharo knjige
- ▶ A community-driven collection of awesome Pharo libraries, tools, frameworks and software
- ▶ Client-side smalltalk - Amber