OOP iz drugog ugla

Objektno - orijentisano programiranje

Deo diskusije na temu: Zašto OOP?

CimBac
Član broj: 37332
http://www.elitesecurity.org/t19637-0#160751

Zasto se preslo na objektno programiranje? 02.12.2004. u 20:41

Pitanje je prilicno jednostavno, e sad da li ce odgovori biti... Nama su na faksu pricali da je osnovni razlog prelaska bio razvoj grafickih okruzenja (menija, buttona, i sl.), a meni jos uvek nije jasno koja je prednost objektnog programiranja. Da li je moguce da se npr. Total Comander ceo napise u cistom C-u (teorijski). Na primer, ja sam napisao programce koje ima butone koji se isto ponasaju kao ovi u windowsu, a koristio sam obicne strukture (struct), a ne klase. Naravno pricam o C/C++.

Napomena: Pročitati celu diskusiju.

Deo diskusije na temu: Zašto OOP?

masetrt
Puno ime: Marko Djurovic
Član broj: 3129

Re: Zasto se preslo na objektno programiranje? 03.12.2004. u 09:12

Pri radu sa vecim projektima mnogo je lakse i brze pratiti i organizovati OO kod nego klasican proceduralni. Nije stvar u tome sta moze da se postigne jednim ili drugim jer sa oba mozes postici isto, pitanje je samo koliko brzo lako i koliko ces se lako vratiti u svoj (ili tudji kod) posle recimo godinu dana pauze. Tu su naravno i mnoga druga jos vaznija olaksanja koja u principu mozes da procitas u clancima koje je napisao Bjorn Storstrap (ala sam ga ispelovao) tvorac c++.

Deo diskusije na temu: Zašto OOP?

Rapaic Rajko
Član broj: 4105
Re: Zasto se preslo na objektno programiranje? 03.12.2004. u 13:56

Recimo da imas dva ili tri velika programa napisana klasicnim struktuiranim jezikom (recimo TurboPascal 5). Svaki od tih programa je napisan za konkretnu namenu i ima prilican nivo slozenosti. Jednog dana ti dodje sef i kaze da treba to sve da sastavis u jedan program. No problem, kazes ti i pocnes...a onda se uhvatis za glavu: globalne varijable istog naziva, procedure, cak i unit-i - nocna mora. Umesto prostog copy-paste-a, ima da se ubijes od prepravljanja naziva, doterivanja scope-a itd. itd.
Sad zamisli isto to, ali s tom razlikom da su u pitanju OOP aplikacije, to jest u svakoj aplikaciji si pravio glavne klase sa metodama, field-ovima, propertijima, subklasama itd. - kako se to vec danas radi. Fuzionisanje tri aplikacije se svodi na ubacivanje klasa/koda iz sve tri aplikacije u jednu novu, i to (ako si radio klase kako treba) ide vrlo, vrlo lako.
Navedeni primer je iz mog licnog iskustva; desio mi se prvi (gore navedeni) slucaj, pre ravno 10 godina; zaginuo sam od posla. Tada jos nisam koristio OOP.

...

Deo diskusije na temu: Zašto OOP?

...

P.S. To je upravo ono sto danas radis u, recimo, Delphi-ju kad stavis komponentu na formu: ubacujes klasu/kod koja nosi sa sobom odredjenu funkcionalnost. Ta klasa je razvijena za 'pitaj boga koga', ali je tako robusno napisana, da se moze koristiti gde god to zatreba (pa i na tvojoj formi). Znaci, vec smo objasnili dve bitne osobine OOP-a: reusebilnost i robusnost. Kraj (zasad).

SAGA O OOP

Sta je to tip podataka, a sta struktura?

Tip podataka je odredjen (konacnim) **skupom vrednosti** i nekim **paketom operacija i relacija** nad elementima tog skupa.
Pr. ({-32768, ..., -1, 0, 1, ..., 32767}, +, -, * div, mod, =, <)

Struktura podataka je konglomerat podataka. Ona se formira od drugih (jednostavnijih) struktura i od tipova. Jedine operacije koje se nad strukturama mogu obavljati su **operacije selekcije**.
Pr. var **a**: **array** [1 .. 100] of real;

struktura **metod struktuiranja**

Metod struktuiranja ima sopstvene operacije selekcije.
Pr. **array** indeksiranje (a[i]), a **record** projektovanje (odabir polja sloga, a.ime) - PASCAL

Pascal:
eksplicitni metodi struktuiranja (array, record, record-case)
implicitni (preko pokazivaca).

Sta je to tip podataka, a sta struktura?

Razlika između tipa i strukture je u tome sto je nad elementima tipa moguće vršiti nekakve operacije i smestati ih u nekakve relacije, dok je nad strukturom moguće obavljati samo selekciju komponente.

Ako uradimo sledeće

```
type STR = packed array [1 .. 100] of char;

procedure Concat(s1, s2 : STR; var res : STR); ...
procedure Substr(s : STR; from, to : integer; var res : STR); ...
function  Compare(s1, s2 : STR) : integer; ...
```

STR ima skup vrednosti i operacije na elementima tog skupa (Concat, Substr, Compare) dakle STR je sada tip.

Zasto je važno praviti nove tipove?

Za uspešan rad u velikim timovima i na velikim projektima bitno pisati apstraktan kood.

NPR. Za projekat u kome je potrebno raditi nesto sa matricama: napraviti tip MATRIX (struktura MATRIX snabdevena operacijama za rad sa matricama), pa kada dodje do rešavanja konkretnog problema, samo pozivati gotove (i testirane) procedure.

```
C := Inv(A) * Transpose(B) + Det(Y) * Adj(X)
```

```
Inv(A, A1);
Transpose(B, B1);
MatMul(A1, B1, P);
Adj(X, X1);
ScalMul(Det(Y), Q);
MatAdd(P, Q, C);
```

```
for i := 1 to n do
        for j := 1 to n do
            (* rutina koja invertuje matricu A u A1 *);
    for i := 1 to n do
        for j := 1 to n do
            B1[i,j] := B[j, i];
    for i := 1 to n do
        for j := 1 to n do
            (* rutina koja mnozi A1 i B1 *);
    (* itd *)
```

SAGA O OOP

Dakle, dizajnirati odgovarajuce strukture podataka, napisati procedure za manipulaciju tim strukturama podataka.

Tako dolazimo do jednog viseg nivoa apstrakcije u programima. U trenutku kada je potrebno primeniti neke operacije na nekim strukturama, ne interesuje nas *kako* procedure rade, vec *sta* rade.

Upotreba tipa

```
var a, b: MATRIX;
begin
   NewMatrix(a); NewMatrix(b);
   ReadMatrix(a);
   Transpose(a, b);
   WriteMatrix(b);
   DisposeMatrix(a); DisposeMatrix(b)
end;
```

Implementacija

```
type MATRIX = array [1 .. X, 1 .. Y] of real;
```

```
type MATRIX = ^MatrixEntry;
     MatrixEntry =  record
                      i, j: integer;
                    entry: real;
                    right, down: MATRIX
                    end;
```

NIJE BITNA IMPLEMENTACIJA, VEĆ MANIFESTACIJE (TJ. OSOBINE) OPERACIJA DATOG TIPA PODATAKA.

SAGA O OOP

Osnovna ideja OOP-a - Abstract data type - ADT

Struktura podataka → Tip podatka data abstraction
ADT → Apstakcija + Skrivanje podataka information hiding

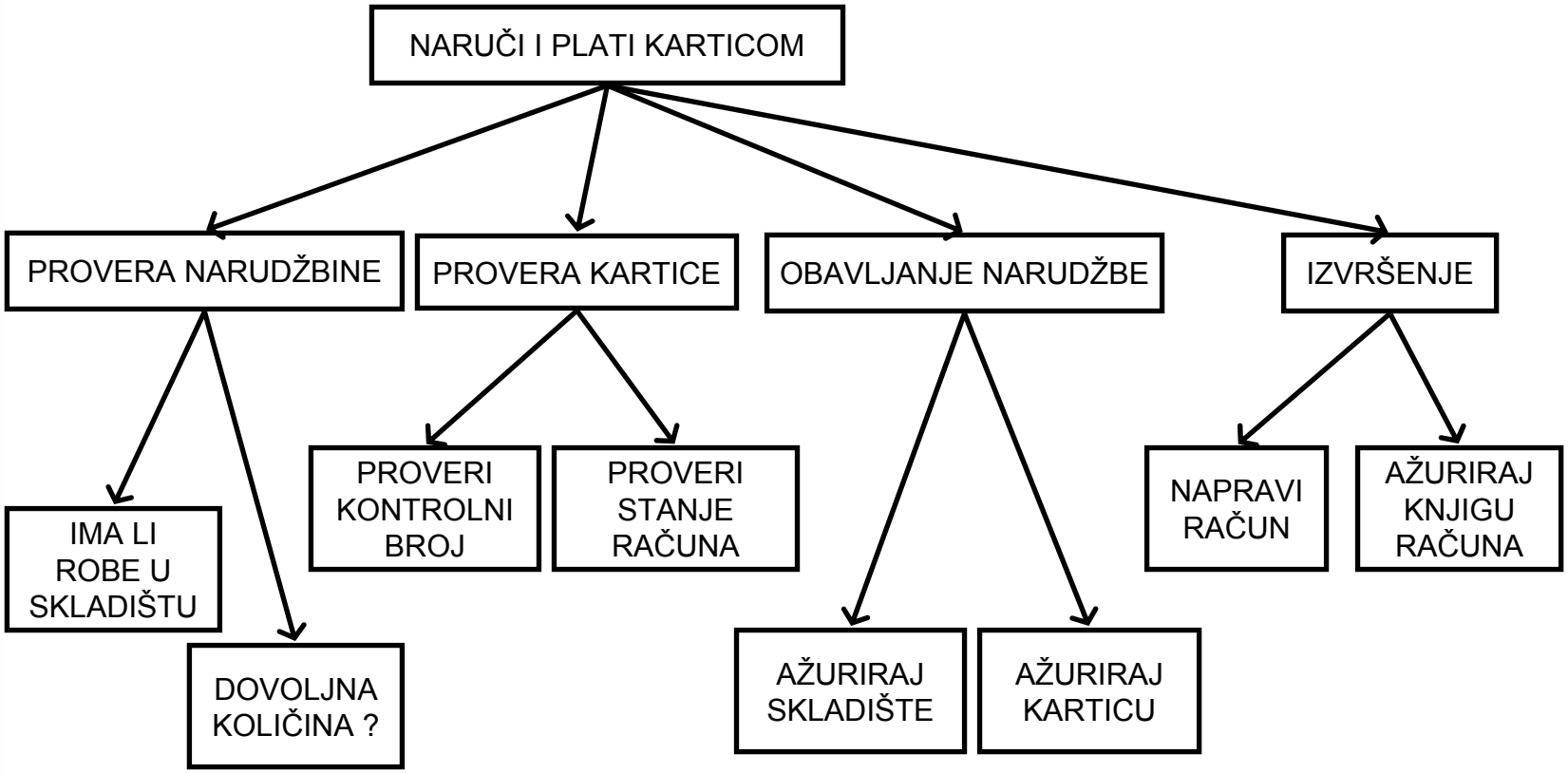
Apstraktni tip podataka je tip podataka čiju implementaciju ne znamo (primer: ne znamo da li su matrice predstavljene sa array ili preko pokazivaca), ali znamo kako se ponašaju operacije nad vrednostima tog tipa, tako da pišemo program koristeci samo osobine operacija.
Dakle, apstrahovana je jedna dimenzija problema: implementacija tipa (apstrahovati znaci izbaciti iz posmatranja ono sto u tom trenutku nije bitno)

Za pojam ADT neraskidivo vezan pojam sakrivanja informacije (information hiding). Programeru se da ime tipa i paket procedura. Implementacija tipa se SAKRIJE da je on ne vidi.

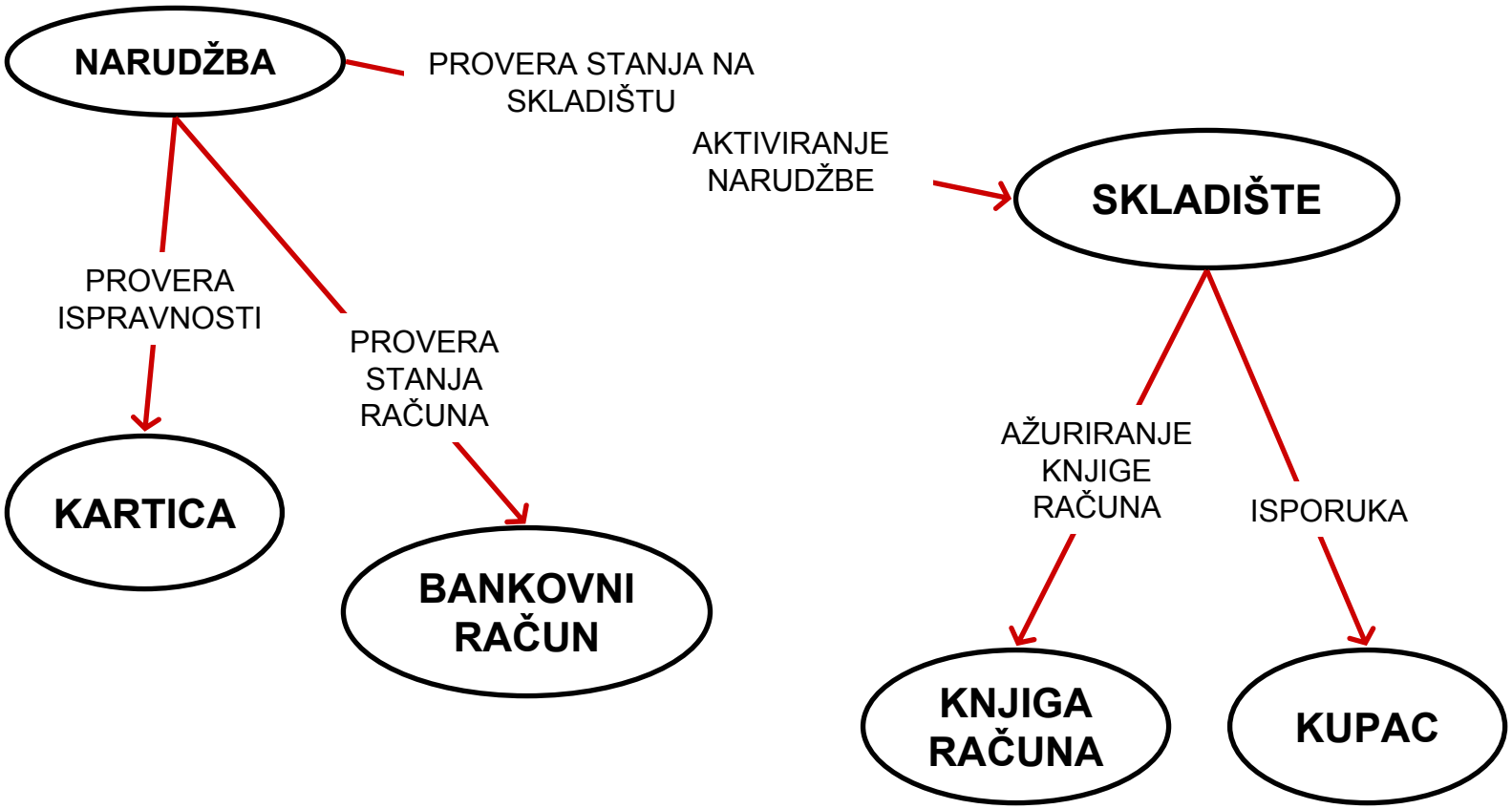
OO vs strukturno modelovanje

- **Strukturno modelovanje**
  - programiranje vodjeno postupkom, aktivnostima;
  - modelovanje se vrši analizom postupka kojim se dolazi do traženih rezultata;
  - algoritam koji se sastoji iz koraka koji se izvršavaju na određen način; algoritmom se opisuje kako i kada se nešto radi;
  - modelovanje podataka, uočavanje objekata u realnom sistemu su u drugom planu.
- **Objektno-orijentisano modelovanje**
  - realizacija softvera kao modela realnog sveta;
  - u postupku modelovanja se identifikuju objekti (iz realnog sistema) koji nešto rade, a zatim se apstrakcijom izdvajaju skupine objekata istih svojstava (pri čemu se modelovanjem ne uzimaju u ubzir nebitne karakteristike);
  - objekat ima svoje stanje i ponašanje (definišu se operacije koje se mogu nad njim izvršiti spolja);
  - objekti međusobno komuniciraju izvršavanjem operacija.

“Modeliranje postupka”
Primer: kupovina karticom



Objektni model
Primer: kupovina karticom



Zašto OOP?

code reuse

- lakše uvezivanje postojećih rešenja
- lakša nadogradnja
- lakše lociranje grešaka

- vaše greške su samo vaše
- lepa posledica - vaše dvorište će biti vidljivo samo u meri u kojoj vi to budete dopustili, sa komšijom vam je zajednička samo ograda

jeftinije i jednostavnije

modelovanje je odvojeno od implementacije

Prvi korak – modelovanje realnog sistema

- Uočavanje objekata u sistemu i to:
  - njihovih osobina za “bitnih“ za obavljanje posla

    Npr. boja kose u opštem slučaju nije bitna za kupovinu, osim ako niste u Turskoj i ženskog pola
  - ponašanja,tj. aktivnosti koje obavlja
- Uočavanje interakcije među različitim objektima
- Uočavanje objekata istog tipa, tj. objekata koji imaju iste karakteristike kojima se opisuju i ista ponašanja, kao i objekata koji imaju neke zajedničke karakteristike i ponašanja (o analizi odnosa objekata sa "nekim" zajedničkim karakteristikama malo kasnije - nasleđivanje)

Objekti

podaci = stanje objekta

opisuju stanje

+

procedure = metodi

} objekat

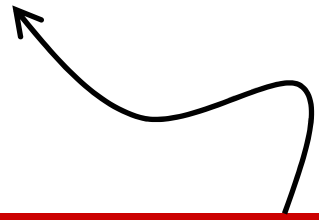
- pripadaju objektu
- menjaju stanje objekta
- njima se opisuje tzv. ponašanje objekta

Objekti

- slanje **poruke** objektu = pozivanje metoda nekog objekta

```
var m: MATRIX;
...
m.init(9,10);
```

- podaci unutar objekta moraju biti zaštićeni, može im se (eventualno) pristupiti (čitanje, promena stanja i sl.) slanjem poruke.



skrivanje podataka se naziva **učaurivanjem** (**encaptulation**) - **information hiding**

Klase

- Klasom se opisuju skupine objekata sa istom strukturom i ponašanjem.
- Objekti koji pripadaju jednoj klasi se nazivaju **instancama** te klase.

```
Class A {
        instance variables // eventualno i class variables
        instance methods // eventualno i class methods
}
```

Primer sa predavanja

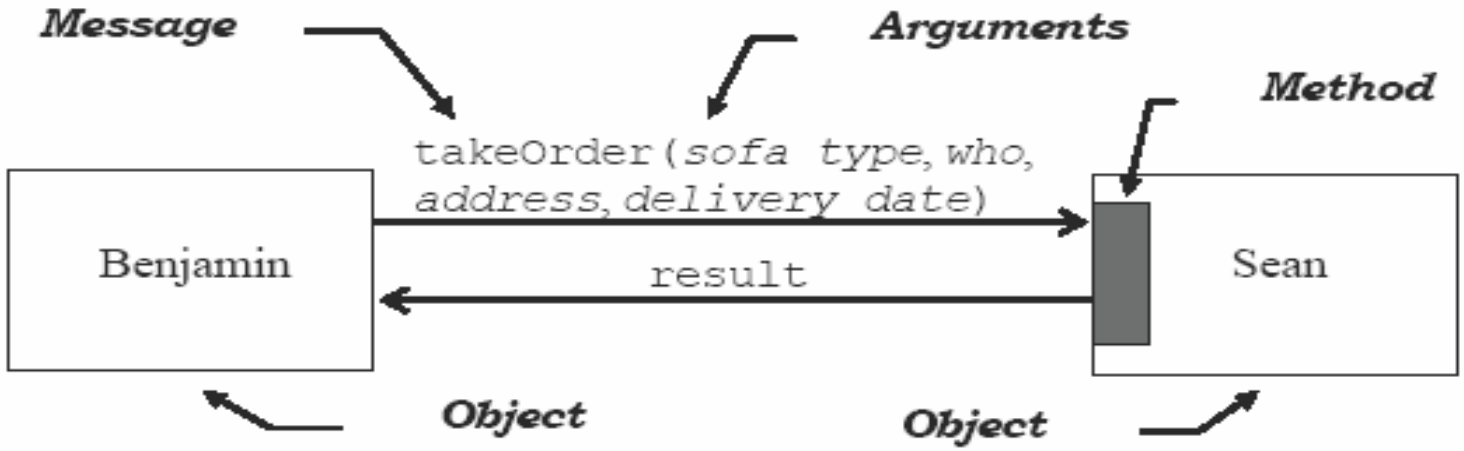
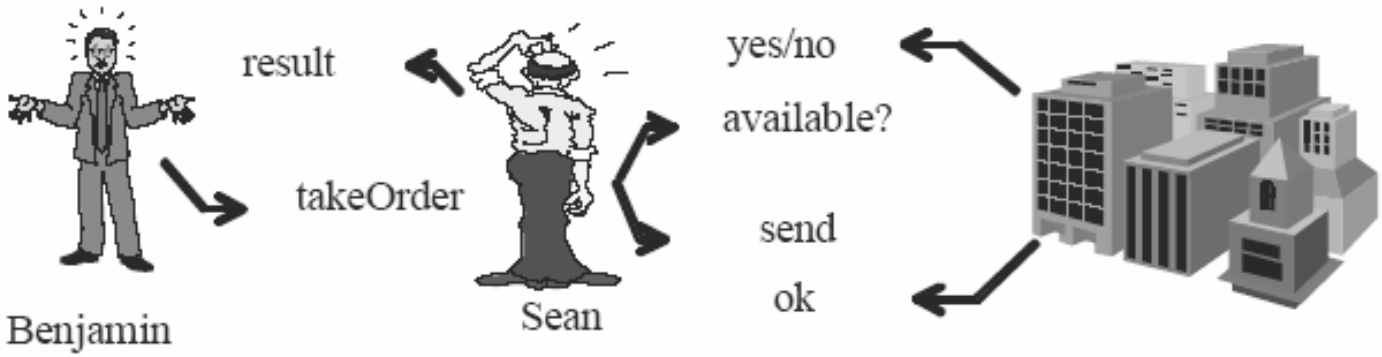


Figure 1-2: Object interactions in object-oriented programming terms.



Primer sa predavanja

```
Benjamin as an Object
Attributes:
   name = Benjamin
   address = 1, Robinson Road
   budget = 2000
Methods:
   purchase() {send a purchase
          request to a salesperson}
   getBudget() {return budget}
```

```
Sean as an Object
Attributes:
   name = Sean
Methods:
   takeOrder() {
          check with warehouse on stock
                availability
          check with warehouse on
                delivery schedule
          if ok
          then {instruct warehouse to
             deliver stock(address, date)
                return ok}
                else return not ok
}
```

Primer sa predavanja

Class Customer

Attributes:

name

address

budget

Methods:

purchase() {send a purchase request to a salesperson}

getBudget() {return budget}

Class SalesPerson

Attributes:

name

Methods:

takeOrder() {

check with warehouse on stock availability

check with warehouse on delivery schedule

if ok then

{instruct warehouse to deliver stock(address,date)

return ok}

else return not ok

}

Primer sa predavanja

```
Benjamin as an Object
Attributes:
name = Benjamin
address = 1, Robinson Road
budget = 2000
Methods:
purchase() {
   Sean.takeOrder("Benjamin", "sofa",
   "1, Robinson Road","12 November")
}
getBudget() {return budget}
```

```
Sean as an Object
Attributes:
   name = Sean
Methods:
   takeOrder() {
         check with warehouse on stock
                availability
         check with warehouse on
                delivery schedule
         if ok
         then {instruct warehouse to
            deliver stock(address, date)
               return ok}
               else return not ok
   }
```

Primer sa predavanja

message = object + method + arguments