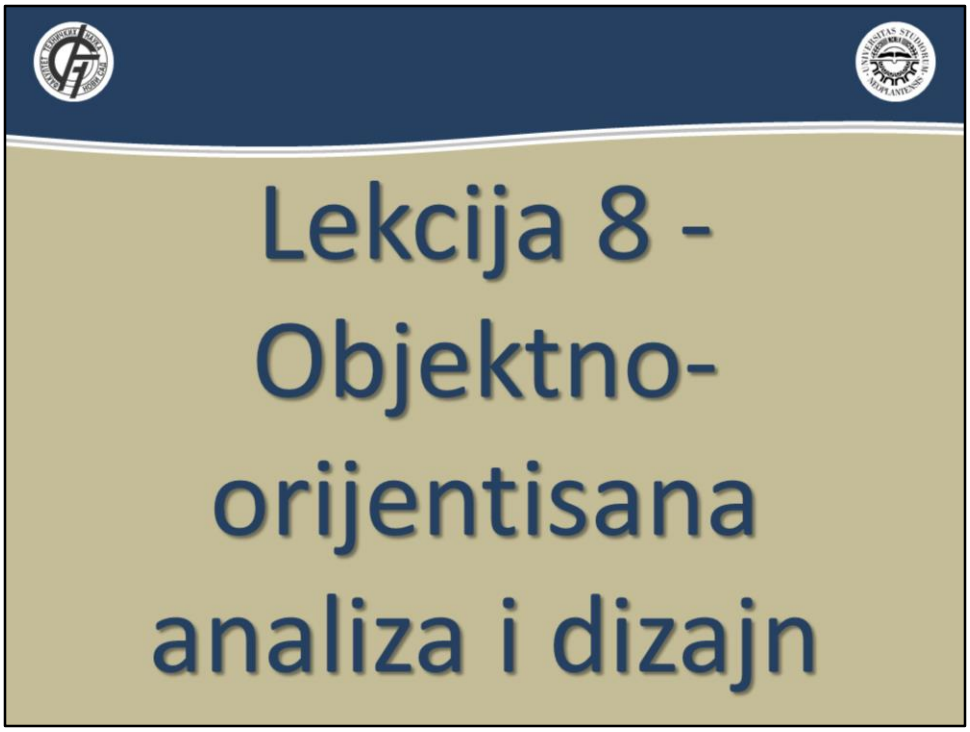

Lekcija 8 -
Objektno-
orijentisana
analiza i dizajn

Šta je objekat?

- Wirth je 70-ih godina publikovao knjigu sa naslovom *Algorithms + Data Structures = Programs*. Objekti se mogu smatrati proširenjem ove ideje u formi *Algorithms + Data Structures + Encapsulation/Information Hiding = Objects* (videti knjigu od Reenskaug-a *Working With Objects*).
- Klasa (*class*) je šablon za kreiranje sličnih objekata (koji su svi istog tipa).
- Svaki objekat ima identifikator (ime); implicitno to je adresa objekta.
- Objekti se mogu smatrati modulima i služe kao gradivna jedinica u izgradnji objektno-orijentisanog modela sistema.
- Objekti se povezuju između sebe i šalju poruke jedan drugom, pri čemu kod svake pojedinačne interakcije možemo prepoznati klijent-server apstrakciju.
- Klase se takođe povezuju, ali u tzv. hijerarhiju klasa sa jednom posebno važnom vrstom veze koja se naziva *nasleđivanje*. To se koristi kako za ponovno korišćenje kôda tako i za modelovanje apstrakcija.

Elektroenergetski softverski inženjering – Razvoj EE softvera - 2016

2

Veoma treba biti oprezan sa nasleđivanjem. Uvek treba proveriti da li postoji “is a” relacija između roditeljske klase i potomaka. Da li postoji ovakva relacija između klasa Vector i Stack u Javinoj biblioteci Colletions? Koje su posledice?

Nasleđivanje narušava enkapsulaciju i može otežati razumevanje celokupnog sistema (pogotovo ako je hijerarhija klasa “duboka”).

Na kraju, nije uvek moguće modelovati ciljni domen objektima (ne može se naći odgovarajuće mapiranje realnih objekata na sistemske).

Odnos između OOA, OOD i OOP

- **Objektno-orijentisana analiza** (OOA) predstavlja proces gde se pokušava objektima opisati problemski domen. U ovom trenutku objekti se ne moraju povinovati pravilima ciljnog programskog jezika.
- **Objektno-orijentisani dizajn** (OOD) je proces gde se objekti identifikovani tokom analize prevode u objektno-orijentisani sistemski model. Ovde se formalizuju relacije između objekata i precizno definišu detalji atributa i operacija.
- **Objektno-orijentisano programiranje** (OOP) je implementacija OO sistemskog modela korišćenjem ciljnog programskom jezika. Naravno, podrazumeva se da je ciljni jezik objektno-orijentisan (u suprotnom je realizacija sistema veoma mukotrpna).

Moguće je da će se tokom OOP-a još doraditi sistemski model, ako je ono kreirano kao generički model primenljiv u što širem opsegu programskih jezika. Na primer, ako je model napravljen da koristi višestruko nasleđivanje, a ciljni jezik to ne podržava (recimo Java), onda se tokom OOP-a mora sistemki model prilagoditi.

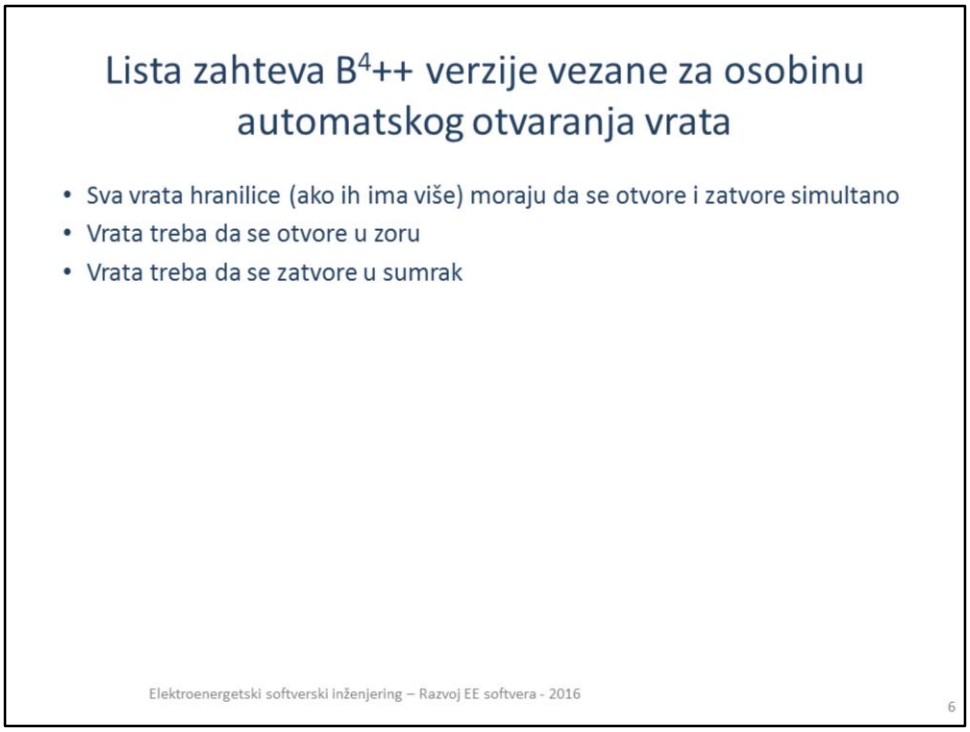
Uprošćen proces OOADP-a

1. Generiši inicijalni skup osobina (*feature*) na osnovu opisa problema (*problem statement*).
2. Kreiraj listu osobina, koje će postati osobine programa. Na osnovu nje se pravi lista zahteva.
3. Kreiraj slučajeve korišćenja (ili korisničke priče) za identifikovane osobine. Tokom ove faze će se identifikovati još osobina i zahteva.
4. Podeli osobine po podsistemima, modulima ili sličnim jedinicama da bi se grupisale funkcionalnosti.
5. Mapiraj prethodno pomenute osobine i module na domenske objekte (apstrakcije).
6. Kreiraj sistemski OO model.
7. Implementiraj i testiraj rešenje.
8. Po potrebi ponovi korake 3-8 u narednoj iteraciji.
9. Izvrši finalni test i instaliraj program kod klijenta.

Primer procesa OOAD na osnovu problema B4Feeder

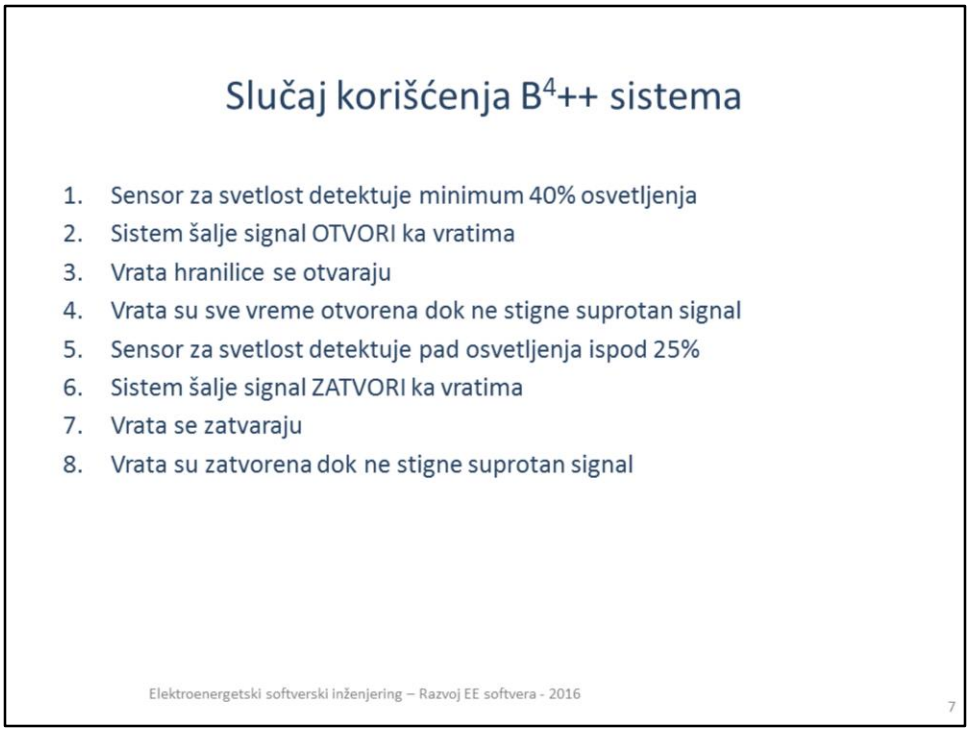
- Ovde je citat iz knjige dat u originalu u vezi opisa problema:

"Burt, the proud owner of Birds by Burt, has created the ultimate in bird feeders. Burt's Bird Buffet and Bath (B4), is an integrated bird feeder and bird bath. It comes in 12 different colors (including camo) and 1, 3, and 5 lb capacities. It will hold up to one gallon of water in the attached bird bath, it has a built-in hanger so you can hang it from a tree branch or from a pole, and the B4 is just flying off the shelves. Alice and Bob are desperate for a B4, but they'd like a few changes. Alice is a techno-nerd and a fanatic songbird watcher. She knows that her favorite songbirds only feed during the day, so she wants a custom B4 that allows the feeding doors to open automatically at sunrise and close automatically at sunset. Burt, ever the accommodating owner, has agreed and the hardware division of Birds by Burt is hard at work designing the B4++ for Alice. Your job is to write the software to make the hardware work."



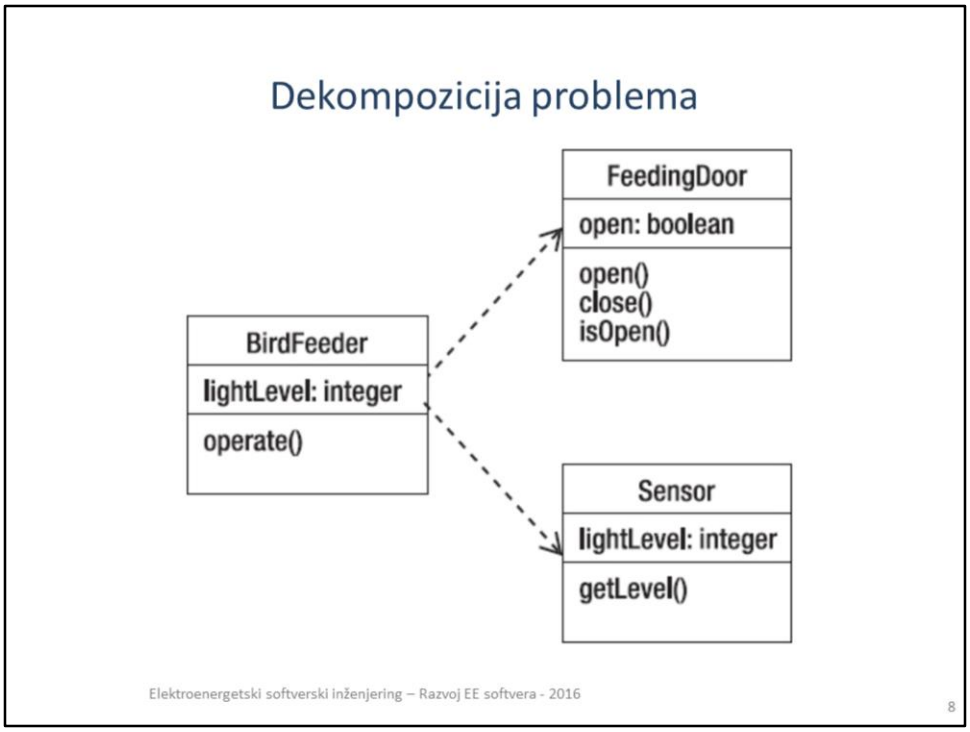
Ovde celu stvar pojednostavljuje da nema nema interakcije sa korisnikom (bilo sa vlasnikom hranilice ili samom pticom).

U knjizi je ova lista označena kao lista osobina (feature list), međutim to nije korektno. Ovde je osobina (feature) sistema zapravo automatsko otvaranje vrata. A sama lista predstavlja zahteve, koji su vezani za ovu osobinu. Svaki zahtev se može posebno implementirati i testirati. Svi zajedno ostvaruju jedan značajan inkrement u funkcinisanju B^4++ sistema.



Ovo je tzv. glavni scenario slučaja korišćenja vezane za osobinu sistema za automatsko otvaranje vrata. Ovde vidimo da je aktor zapravo eksterni uređaj, tj. on pokreće ceo proces. Ovo je primer kada korisnik nije direktno uključen u proces, ali je zainteresovan za njegov ishod (ovakav subjekt se zove *stakeholder*). Svaki slučaj korišćenja ima jasan cilj. Pored toga postoje i preduslovi, invarijante i garancije (uslovi koje važe nakon ostvarenja *use case*-a).

Najinteresantniji aspekti se ogledavaju u raznima alternativnim tokovima, pogotovo kod rukovanja greškama. Koje alternativne tokove možete otkriti? Da li oni dovode do proširivanja liste zahteva? Ako da, koji su to zahtevi?

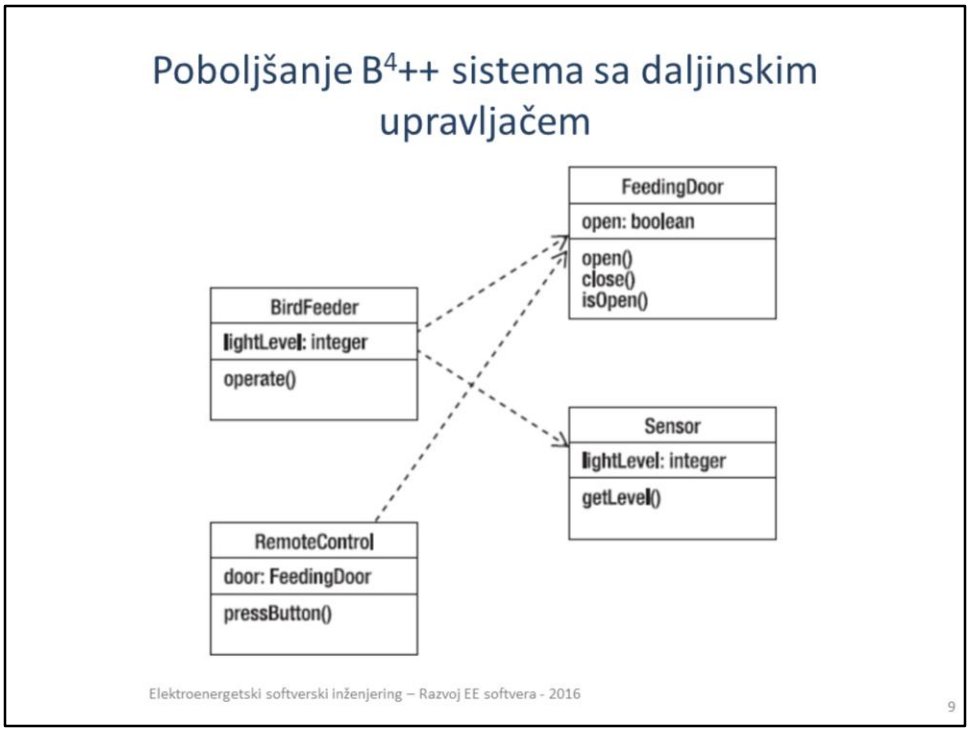


Ovde vidimo da je klasa BirdFeeder asocirana sa klasama FeedingDoor i Sensor. BirdFeeder kešira poslednje iščitan podataka iz sensora umesto da se uvek oslanja na vrednost koja je poznata samo senzoru. Da li ovo ima smisla? Ako da, koji je smisao toga?

Da li na ovom dijagrami klasa ima elemenata, koji su suvišni (kaže se opterećuju dijagram nepotrebnim detaljima)? Ako da, koji su to elementi?

NAPOMENA:

Implementacija metode operate u knjizi je krajnje trivijalna i ne odgovara realnosti. “Produkciona” verzija bi trebala da realizuje BirdFeeder kao aktivni objekat, tj. objekat sa sopstvenom niti za izvršenje glavne logike.



Na ovom dijagramu iz knjige, a i tekst to naglašava, dodavanje daljinskog upravljača se čini ne zahteva nikakve izmene na ostale tri postojeće klase. Da li je ovo zaista tako? Koji je fundamentalni propust napravljen u dizajnu (što se kasnije reflektovalo i o kôdu)? Sa koliko vrata daljinski upravljač može da upravlja, a šta je zahtevano od strane sistema sa više vrata?