Objektno orjentirano programiranje

Predavanje 3

Razmišljanje u okvirima objekata

Uvod

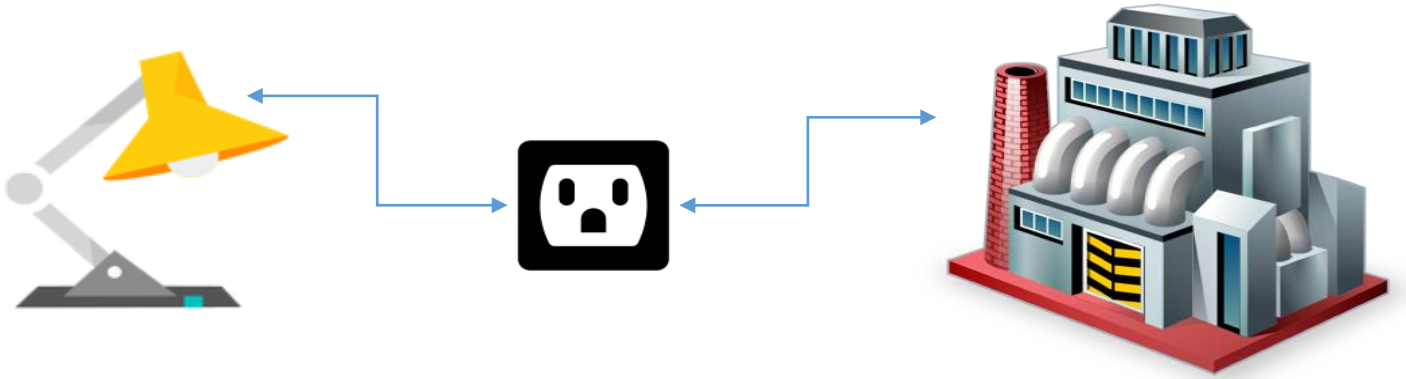
- Kao i u stvarnom svijetu, u programiranju ne postoji samo jedan pristup rješavanju problema
- Model rješenja ne mora biti savršen u prvoj iteraciji
- Najbolje istražiti više različitih pristupa
- U startu procesa nije još bitan odabir programskog jezika
- Početak je rješavanje problema/zadatka domene (eng. business problem)
- Odabir tehnologije u startu samo ako je fundamentalno vezana za sam problem

OO programiranje

- Osim objekata u OO pristupu koristimo i različite programske strukture (petlje, uvjetne izraze, itd.)
- Za početak programiranja u OO pristupu potrebno je prvo naučiti OO koncepte i OO način razmišljanja
- Tri važne stvari kod OO načina razmišljanja:
  - Poznavanje razlike između sučelja i implementacije
  - Apstraktno razmišljanje
  - Pružanje korisniku samo minimalno potrebno sučelje

Poznavanje razlike između sučelja (interfejsa) i implementacije

- Kod dizajna klase potrebno je razlučiti što korisnik treba, a što ne treba znati
- Mehanizam skrivanja podataka sadržan u procesu učahurivanja



- Sučelje kod automobila je poznato, implementacija je nepoznata za većinu ljudi

Icons by PixelKit ,WPZOOM and Iconshock

Što korisnici vide?

- Korisnici klasa (programeri koji koriste klasu) vide sučelje klase
- Programeri često koriste više različitih klasa paralelno
- Sve te klase napisali su različiti ljudi
- Programer mora znati kako ispravno koristiti neku klasu
- Ispravno napisane klase se trebaju sastojati od dva dijela: sučelje i implementacija

Sučelje (interfejs)

- Usluge koje se pružaju korisniku zajedno čine sučelje
- U najboljem slučaju samo one usluge koje su korisniku potrebne idu u sučelje
- Pravo pitanje je što korisniku zapravo treba
- Različiti ljudi će vjerojatno dati donekle različite odgovore na ovo pitanje
- Potrebno je poznavati korisnike klase koje dizajniramo i na koji način će one njima biti korisne

Implementacija

- Detalji implementacije su skriveni od korisnika
- Jedna bitna stvar koju treba imati na umu: promjene u implementaciji ne bi trebale utjecati na kod koji je korisnik pisao, niti zahtijevati da korisnik mora mijenjati svoj kod
- Interfejs sadrži sintaksu kako pozvati metodu i povratnu vrijednost – ako se ovo ne mijenja i dobije se ista povratna vrijednost onda korisnik ne mora ništa mijenjati
- Primjer korištenja mobitela – interfejs je jednostavan – utipkamo broj i pozovemo. U slučaju da se implementacija mijenja (npr. operater instalira novu opremu) to ne mijenja način na koji mi koristimo mobitel

Primjer interfejsa i implementacije

- Potrebno je napraviti klasu kora predstavlja čitač iz baze podataka
- Potrebno je poznavati koje funkcionalnosti treba naš korisnik (korisnik ove klase)
- Obično se zahtjevi navedu u dokumentu vezanom za projekt nakon analize i razgovora sa potencijalnim korisnicima

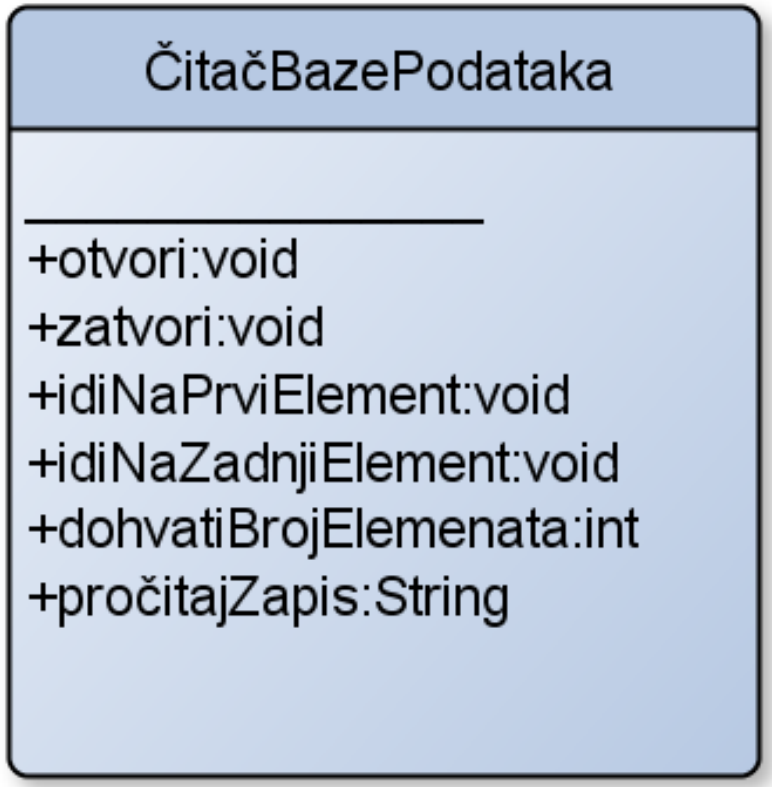
Lista zahtjeva

- Otvaranje konekcije prema bazi
- Zatvaranje konekcije prema bazi
- Postavi kursor na poziciju prvog zapisa
- Postavi kursor na poziciju zadnjeg zapisa
- Dohvati ukupan broj zapisa
- Da li ima još zapisa (da li je kursor na zadnjem zapisu)
- Postavi kursor na određeni zapis korištenjem ključa
- Dohvati određeni zapis korištenjem ključa
- Dohvati sljedeći zapis u odnosu na kursor

Primjer

- Klasa koju dizaniramo je namjenjena programerima kojima je potrebana upotreba baze podataka
- Prema tome, ovo sučelje je zapravo **API – Application-programming interface** koje će korisnik koristiti
- Ovakav interfejs je zapravo *wrapper* funkionalnosti baze
- Razlog zbog kojeg ga radimo je mogućnost prilagodbe (customization) različitih funkcionalnosti (npr. dodatna priprema podataka prije zapisa ili čitanja iz baze)
- Dodatan razlog je mogućnost promjene sustava/baze bez potrebe za mijenjanjem korisničkog koda

Primjer



- U slučaju na naknadno uvidimo da nam trebaju još neke funkcionalnosti možemo ih dodati – dizajn je iterativan proces

Interfejs

- Za svaki od zahtjeva koji smo napisali također smo napisali i metodu koja pruža tu funkcionalnost. Potrebno je pstaviti nekoliko pitanja:
- Da bi korisnik (programer) efektivno koristio klasu da li je potrebno da zna još nešto o njoj?
- Da li korisnik treba znati interni kod implementiran za otvaranje baze podataka?
- Da li korisnik treba znati interni kod implementiran za postavljanje kursora na neki određeni zapis?
- Da li korisnik treba znati interni kod implementiran za provjeru da li postoji još zapisa?

Interfejs

- Odgovor na ova pitanja je **ne**
- Nije potrebno poznavati ove informacije
- Ono što je bitno da korisnik dobije točne povratne vrijednosti i da su operacije ispravno napravljene
- Aplikacijski programer će zapravo najvjerojatnije biti još jedan apstraktni nivo iznad implementacije
- Aplikacija će koristiti ove klase za otvoriti bazu, a klase će pozvati API od baze podataka
- Da bi se prožio minimalni interfejs najbolje je krenuti od poptuno praznog interfejsa i polako dodatavati funkcionalnosti koje bi korisik želio
- Nije dobro unaprijed pretpostavljati da bi korisnik nešto želio neku specifičnu funkcionalnost

- Stvaranje wrappera može izgledati kao suvišan posao, ali ima svojih prednosti
- Npr. razmotrimo potrebu mapiranja objekata u relacijske baze podataka
- Na tržištu postoje OO baze podataka, ali i veliki broj sustava koji koristi relacijske baze
- Ako radimo programe za sustav koji koristi relacijski model može se koristiti *middleware* koji će mapirati objekte iz aplikacijskog koda u relacijski model
- Čak i ako iz početka stvorimo potpuni OO sustav često nećemo moći izbjeći korištelje ralacijskih baza – budući da većina sustava ovisi o informacijama iz nekog drug ili više različitih sustava

Object persistence

- Koncept koji se odnosi na spremanje stanja objekta na način koji omogućava njegovo kasnije korištenje
- Objekt koji ne koristi „persistnace” po prirodi nestaje kada izađe iz dosega
- Npr. stanje objekta se može spremiti u bazu podataka ili datoteku

Primjer implementacije

```
public void open(String ime){
/*Kod potreban za specifične zadatke*/
/*Pozovi Oracle API za otvaranje baze*/
/*Dodatni kod potreban za specifične zadatke*/
}
```

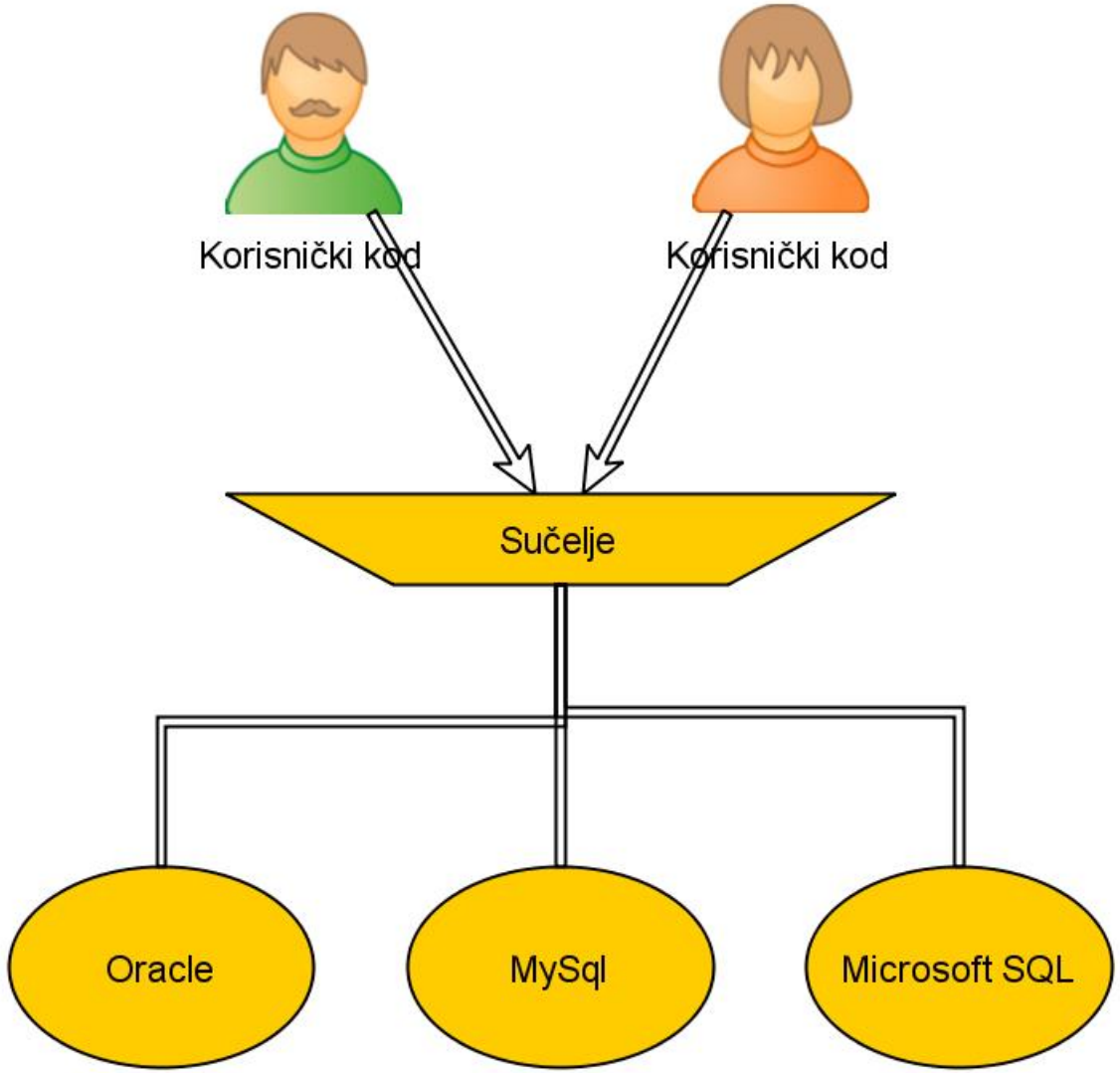
- Kada promatramo iz perspektive programera vidimo da je za metodu open potreban string Ime koji predstavlja datoteku baze podataka. To je sve što korisnik treba znati.
- U slučaju da jednog dana mijenjamo implementaciju, npr. prijelaz iz Oracle baze podataka na SQLAnywhere bazu

Primjer implementacije

```
public void open(String ime){
/*Kod potreban za specifične zadatke*/
/*Pozovi SQLAnywhere API za otvaranje baze*/
/*Dodatni kod potreban za specifične zadatke*/
}
```

- Nakon kodiranja potrebnih stvari od strane implementatora, pogledajmo što naš krajnji korisnik treba promijeniti
- Ništa – to je upravo prednost razdvajanja inetrfejsa i implementacije

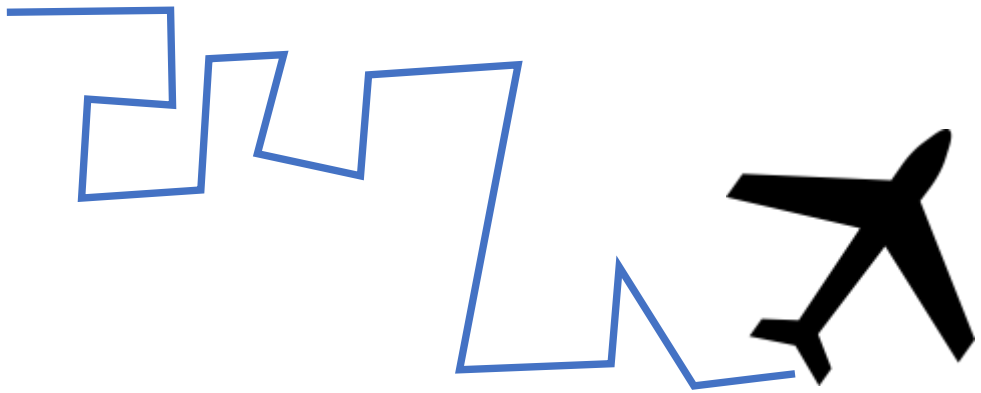
Primjer implementacije



Apstraktno razmišljanje prilikom dizajna interfejsa

- Jedna od najvećih prednosti OO programiranja je mogućnost ponovnog korištenja koda (eng. code reuse)
- Konkretni interfejsi su vrlo specifični, a apstraktni interfejsi su generalniji
- Primjer razlike između apstraktnog i konkretnog interfejsa

Odvedi me
na aerodrom

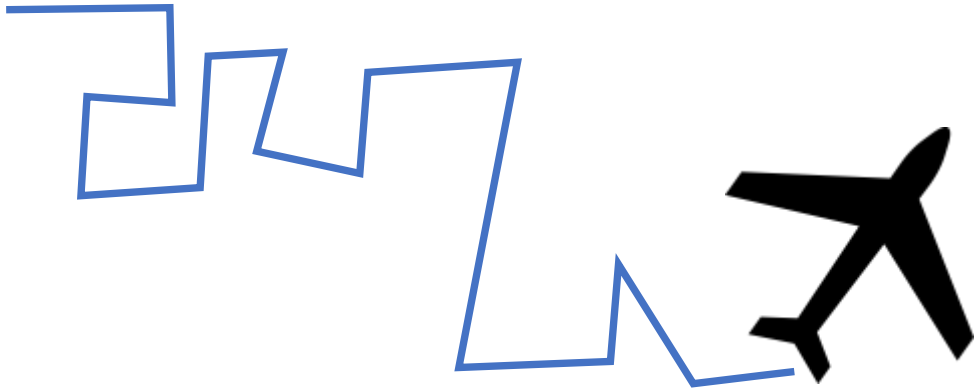


Icons by Alex Talalueva, Icons Land

Apstraktno razmišljanje prilikom dizajna interfejsa

- Postavlja se pitanje koji od ova dva scenarija ima bolju mogućnost ponovnog korištenja u različitim situacijama

Vozi ravno,
skreni desno,
ravno, lijevo …



Icons by Alex Talalueva, Icons Land

Pružanje samo minimalnog interfejsa korisniku

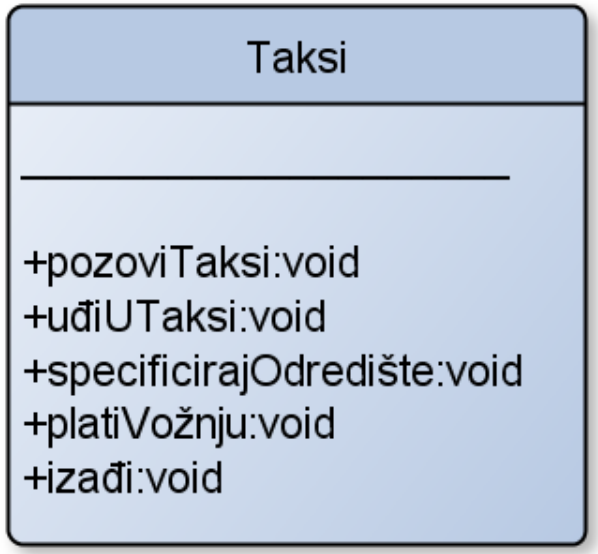
- Pružite korisniku isključivo samo ono što mu je potrebno. Ako pokaže da je potrebno proširiti interfejs to se radi iterativno.
- **Bitno je klase dizajnirati iz perspektive korisnika, a ne iz perspektive sustava**. Dizajneri klasa klase dizajniraju na način da se uklapa u specifični tehnološki model. Potrebno je razmišljati na koji način će ta klasa biti korisna korisniku.

Potrebe korisnika

- Potrebno je postići kompromis između jednostavnosti i udovoljavanju zahtjevima korisnika i mogućnosti implementacije
- Ponekad je neke zahtjeve tehnički nemoguće izvesti i treba se pronaći zajedničko rješenje

Definiranje interfejsa

- Nakon što su se sakupili svi potrebni zahtjevi kreće se u definiranje interfejsa
- **Prilikom definiranja interfejsa razmišlja se o načinu kako se objekt koristi, a ne o načinu njegove implementacije**
- U slučaju da se otkriju novi zahtjevi oni se dodaju narednim iteracijama
- Analogija interfejsa sa primjerom taksija:
  - pozovi taksi
  - uđi
  - specificiraj odredište
  - plati vožnju …



Identifikacija implementacije

- Tek nakon što su interfejsi odabrani kreće se sa identifikacijom implementacije
- Tehnički gledano – sve što nije dio interfejsa se smatra implantacijom
- Sve privatne metode se smatraju implementacijom
- Korisnik bi trebao vidjeti samo način pozivanja/komunikacije s interfejsom, ali ne i sam kod koji te metode sadrže
- Sve što se smatra implementacijom bi dizajner klase trebao moći promijeniti (u slučaju da je to potrebno) bez da se to odrazi na krajnjeg korisnika