Objektno orjentisano programiranje – C++

Objektno orjentisano projektovanje (dizajn)

Teme

- Zašto Objektno orjentisano programiranje
- Osnovne osobine OOP jezika
- Objektno orjentisani dizajn - koraci

Zašto C++ (odnosno OOP)?

- Objektno programiranje podržava apstrakciju podataka i bolju ponovnu iskoristivost koda.
- C++ podržava višestruki pristup programiranju: objekte možemo koristiti samo za neke segmente programa ili ih čak uopšte ne moramo koristiti.

Osnovne osobine OOP jezika

- Apstrakcija podataka – formiranje objekata koji obuhvataju opšte karakteristike, bez detalja.
- Skrivanje podataka – prikrivanje unutrašnje strukture objekta od ostatka programa.
- Ponovna iskoristivost – svojstvo koje omogućava da se jednom napravljeni delovi programa ponovo iskoriste u nekom drugom projektu uz minimalne izmene.
- Nasleđivanje – proces kojim se jednom objektu omogućuje da preuzme svojstva drugog objekta.
- Polimorfizam – izvedene klase imaju slobodu da same implementiraju nasleđene funkcije (ili samo neke od njih).

Objektno orjentisani dizajn - koraci

- Korak 1 – Pronalaženje odgovarajuće apstrakcije
- Korak 2 – Definicija apstrakcije
- Korak 3 – Definicija odnosa i veza između klasa
- Korak 4 – Definicija implementacijski zavisnih apstrakcija
- Korak 5 – Definicija interfejsa
- Korak 6 - Implementacija

Korak 1 – Pronalaženje odgovarajuće apstrakcije

- Pronalaženje odgovarajućeg modela za opisivanje naših podataka.
- Pre nego što se počnu pisati klase potrebno je ustanoviti s kojim klasama će se uopšte raditi.
- Dobra je praksa zadatak detaljno predstaviti na papiru i izdeliti ga na segmente.
- Pažljivom analizom zadatka odrediti dobre kandidate za objekte.

Korak 2 – Definicija apstrakcije

- Za svaki objekt precizno definisati šta radi i sa kojim objektima sarađuje.
- Tehnika CRC kartica (*class, responsibility, collaboration* – klasa, odgovornost, saradnja) – tehnika američkih stručnjaka Beka i Kaningema.

Tehnika CRC kartica

- Za svaku klasu koja se pojavljuje u projektu formira se jedna kartonska kartica tačno određene veličine.
- Na kartice se upišu naziv klase i njene glavne osobine.
- Ljudi koji rade na projektu se okupe na sastanku i svaki dobije jednu ili više kartica.
- Oni uzimaju ulogu pojedinih klasa i pokušavaju odigrati što više scenarija interakcije između objekata.
- Zaključci i primedbe se unose na kartice kako bi se na kraju mogla izvući suština svake klase.

Korak 3 – Definicija odnosa i veza između klasa

- Ovaj korak se po pravilu posmatra zajedno sa prethodnim korakom.
- Prilikom određivanja odnosa i veza među klasama mora se biti veoma pažljiv jer se lako prave greške.
- Retko koja klasa postoji sama za sebe.
- Većina klasa na neki način dolazi u interakciju (odnos) sa drugim klasama.

Osnovni tipovi odnosa između klasa:

1. Odnos ***biti*** – apstrakcija opisana klasom A je istovremeno i apstrakcija opisana klasom B. A je podskup od B.
2. Odnos ***posedovati*** – apstrakcija opisana klasom A sadržava (poseduje) neku drugu apstrakciju opisanu klasom B.
3. Odnos ***koristiti*** – apstrakcija opisana klasom A koristi ali ne poseduje apstrakciju opisanu klasom B.

Razlika između odnosa *biti* i *posedovati* – primer Monitor

- Odnos ***biti*** – klase Ekran i LCDEkran
- LCDEkran je klasa koja je nasledila klasu Ekran

- Odnos ***posedovati*** – klase Ekran i Paleta (definiše listu boja koje se trenutno mogu prikazati na ekranu)
- Ekran nije podtip klase Paleta (a ni obrnuto), on jednostavno sadrži paletu
- Klasa Paleta ne nasleđuje klasu Ekran

Odnos *koristiti*

- Jedan objekat u svom radu samo koristi neki drugi objekat, oni nisu ni na koji drugi način povezani.
- Ta dva objekta mogu komunicirati putem njihovog javnog interfejsa ili preko privatnog interfejsa ako je jedan deklarisan kao prijatelj drugog.
- Ovakav odnos se u C++ najčešće implementira pomoću pokazivača i referenci između objekata.

Vrste odnosa prema smjeru komunikacije i broju učesnika

- Prema smeru komunikacije:
  - jednosmerni
  - dvosmerni
- Prema broju učesnika:
  - jedan na jedan
  - jedan na više
  - više na jedan
  - više na jedan

Korak 4 – Definicija implementacijski zavisnih apstrakcija

- Definišu se veze i odnosi između objekata koje omogućavaju harmoničan rad svih komponenti.
- Nekad je potrebno uvesti poseban podsistem (objekt) koji će pratiti međusobne interakcije između postojećih objekata npr. kada je koji od njih aktivan a kada nije, kada se uključuje u rad a kada isključuje itd.

Korak 5 – Definicija interfejsa

- Precizno definisanje interfejsa pojedinih klasa i načina na koji će klase međusobno komunicirati.
- Skrivanje podataka – svaki objekt pruža okolini upravo onoliko informacija koliko je potrebno da bi okolina mogla uspešno komunicirati s njim.
- Podaci članovi su obično sakriveni a interfejs se osigurava pomoću funkcija članica.
- Vrlo je važno ispravno odrediti da li je neki funkcijski član klase virtuelan ili nije. **Kako to znati?**

Korak 5 – Definicija interfejsa (2)

- Virtuelan funkcijski član specificira akciju koja je vezana uz tip podataka na kojima se obavlja.
- Ako se akcija za različite tipove obavlja na isti način onda nije vezana za tip te je nije potrebno učiniti virtuelnom.
- Preporuka je da se virtuelni funkcijski članovi koriste samo kada je neophodno.

Korak 6 - Implementacija

- Implementacija podrazumeva dodavanje podataka članova, pisanje koda funkcija članica, ugradnja željenih algoritama i slično.
- Složenost implementacije direktno zavisi od složenosti samog modela koji želimo opisati pomoću objekata.
- Prilikom pisanja implementacije važno je poštovati princip skrivanja informacija – korisnik klase mora znati što manje o organizaciji klase, važan je javni interfejs.

Korak 6 – Implementacija (2)

- Princip skrivanja informacija podrazumeva skrivanje podataka članova – oni su najosetljiviji deo svakog objekta jer predstavljaju unutrašnje stanje objekta.
- Ako bi podatak član imao javni pristup tada bi nesmotreno napisan program mogao promeniti vrednost nekog podatka člana tako da objekat više ne bi imao smisla.
- To bi moglo ugroziti funkcionisanje objekta a preko njega i funkcionisanje čitavog sistema.

Zaključak

- Osobine OOP jezika: apstrakcija podataka, skrivanje podataka, ponovna iskoristivost, nasleđivanje, polimorfizam.
- Koraci u objektno orjentisanom projektovanju:
  - Korak 1 – Pronalaženje odgovarajuće apstrakcije
  - Korak 2 – Definicija apstrakcije
  - Korak 3 – Definicija odnosa i veza između klasa
  - Korak 4 – Definicija implementacijski zavisnih apstrakcija
  - Korak 5 – Definicija interfejsa
  - Korak 6 - Implementacija

Kraj prezentacije

HVALA NA PAŽNJI!