10.12.2021.

Objektno-orijentisano programiranje 2

Muva

Motivacija i osnovna ideja:

Razmotrimo primer objekata u video igri sa slike 1.

Slika 1: Grafički interfejs video igre

Na slici vidimo veliki broj građevina istog tipa. Za svaki tip građevine potrebno je imati određenu teksturu, matricu, koja nam omogućuje da iscrtamo građevinu, kao i poziciju date građevine na mapi.

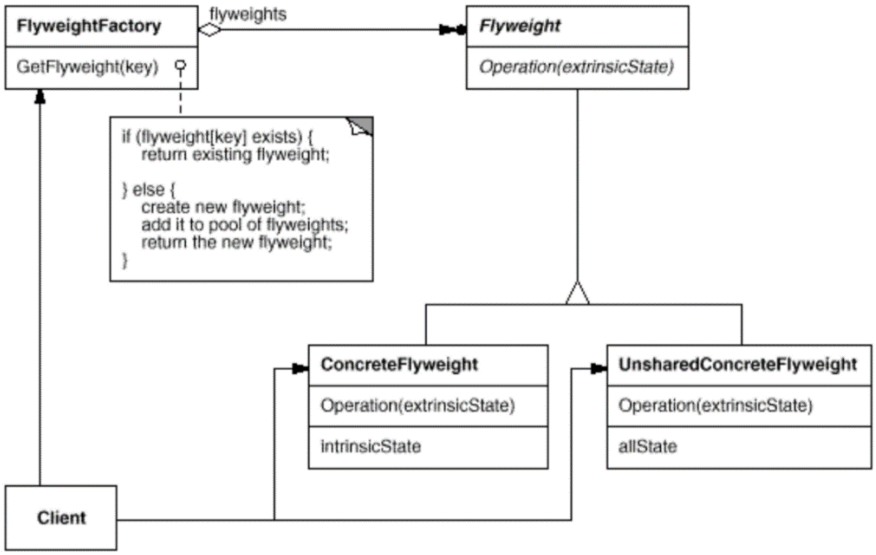
Svaki objekat tipa određene građevine može imati kao attribute RGB matrice za iscrtavanje i poziciju na mapi (kao x, y koordinate). Međutim, primetimo da svaka građevina datog tipa izgleda ***isto***. Svaki objekat određenog tipa građevine imao bi iste RGB matrice, a razlikovale bi se samo koordinate. Iako data matrica možda ne zahteva veliku količinu memorije, za veliki broj građevina (a slično važi i za vojnike) memorijski utrošak za kopije istih matrica postao bi značajan.

Rešenje za dati problem nudi projektni uzorak Muva. U slučaju da imamo objekte od kojih mnogi imaju atribute sa identičnim vrednostima, a potrebno je instancirati veliki broj takvih objekata, identične atribute (koje u terminologiji ovog projektnog uzorka nazivamo **internim stanjem**) treba objediniti u klasu čiji će objekti biti deljeni, a ne višestruko umnožavani. Objekti te klase možda neće moći "samostalno" da obavljaju svoju funkciju u programu jer nisu svi atributi takvi

da je njihovo deljenje moguće, već će njihove relevantne metode zahtevati kao argumente određene vrednosti koje predstavljaju **eksterno stanje** objekta. Na primer, umesto da klasa GrađevinaTipA ima kao atribute RGB matrice, x, y poziciju i metodu iscrtaj(), kao zajednički atribut, tj. Interno stanje, zadržaćemo RGB matricu, a poziciju x i y slaćemo metodi iscrtaj(x, y) kao eksterno stanje tako da iscrtavanje bude moguće. Da bismo u programu delili objekat, kreiranje objekta prepuštamo fabrici koja za dati *ključ* (mogu biti vrednosti internog stanja ili prosto neki id) proverava da li takav objekat već postoji i ukoliko postoji vraća pokazivač na njega. Ukoliko ne postoji kreira ga i zatim vrati pokazivač. Klasa objekata koje delimo naziva se Muva (terminologija je preuzeta iz kategorija u boksu i treba da nas asocira na to da su u pitanju laki objekti). U opštem slučaju, možemo u istoj hijerarhiji imati i objekte koje ne treba deliti (ali naglasak je na deljenim objektima).

Napomena: da bi deljenje bilo moguće (tj. bezbedno) ne smemo dozvoliti da se interno stanje objekta nakon kreiranja menja (jer ukoliko ga promenimo na jednom mestu, promena bi se odrazila na sve delove koda koji dele isti objekat).

UML dijagram projektnog uzorka Muva dat je na slici 2.



Slika 2: Projektni uzorak Muva

Zadaci:

1. Implementirati opšti oblik projektnog uzorka Muva u skladu sa UML dijagramom na slici 1.
2. Primeniti projektni uzorak Muva na situaciju opisanu u uvodu ovog fajla.
3. Napisati hijerarhiju klasa koje predstavljaju različito drveće i životinje u nekoj simulaciji. I životinje i drveće su grafički elementi i mogu se iscrtavati ukoliko je poznata koordinata objekta i rotacija (ugao od 0 do 360 stepeni). Drvo može biti tipa hrast ili smrča, a u

zavisnosti od tipa učitava se .obj fajl sa 3D modelom. Životinja može biti medved ili vuk (kreiranjem se takođe učitava 3D model). Šuma je grafički element pravougaonog oblika dužina x širina i sadrži veći broj stabala, po jedno na svakoj jedinici površine i nasumično rotirano. Šuma osim dimenzija ima i koordinatu gornjeg levog ugla. Na svakih 5 kvadratnih jedinica površine šume ide po jedna životinja. Šuma može biti listopadna ili četinarska. Ukoliko je listopadna, drveće je tipa hrast, a životinja medved, a u četinarskim šumama imamo smrče i vukove. Iscrtavanje šume podrazumeva iscrtavanje drveća i životinja u šumi.
Drveće i životinje treba organizovati kao muve. Fabrika muva treba biti realizovana kao apstraktna fabrika, a konkretne fabrike treba implementirati kao unikate.

Iterator

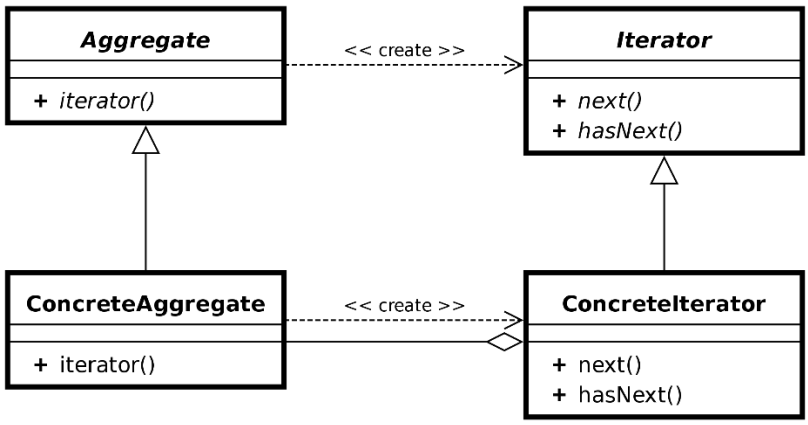
Motivacija i osnovna ideja:

Zadatak iteratora je da pruži funkcionalnosti za pristup elementima neke kolekcije podataka (na engleskom se često koristi termin aggregate, što bi moglo da se prevede kao skup). Možda je u pitanju takva struktura podataka kod koje iteriranje kroz elemente ne može da se svede na standardno for(int i=0; i < kolekcija.size(); i++) f(kolekcija[i]); Na primer, možemo raditi sa objektom koji je kompozit, sa stablom, grafom i sl.

Sa druge strane, čak i da korisnik „ručno ume“ da iterira kroz konkretnu strukturu podataka, šta ako želimo da imamo više struktura podataka kroz koje se iterira na različit način, a koje su izvedene iz zajedničke klase. Mi dobijemo samo pokazivač na osnovnu klasu – kako da iteriramo kad ne znamo čak ni koja izvedena klasa se krije iza datog pokazivača?

Korišćenje projektnog uzorka iterator rešava ovaj problem. Objekat koji predstavlja našu strukturu podataka treba samo biti u stanju da kreira iterator, a na iteratoru je da interno vodi računa o tome da li je stigao do kraja iteriranja i treba biti u mogućnosti da se pomeri na sledeći element. U slučaju da imamo više različitih klasa kolekcija izvedenih iz zajedničke klase, potrebno je implementirati iteratore za svaku od klasa, a svaka konkretna kolekcija treba da vrati iterator za svoj tip.

Učesnici projektnog uzorka Iterator i njihova komunikacija dati su UML dijagramom prikazanim na slici 3.



Slika 3: Projektni uzorak Iterator

Zadaci:

4. Implementirati klase Niz i JednostrukoPovezanaLista izvedene iz klase Kolekcija, kao i iteratore za date klase.
5. Implementirati klasu Niz i iterator za Niz tako da je moguće iterirati kroz instance klase foreach petljom, tj. da bude moguće izvršiti liniju:

   ```
   for(auto elem : niz) std::cout << elem << std::endl;
   ```

   Promenljiva niz je tipa Niz, a elem je istog tipa kao i elementi niza.