19.11.2021.

Objektno-orijentisano programiranje 2

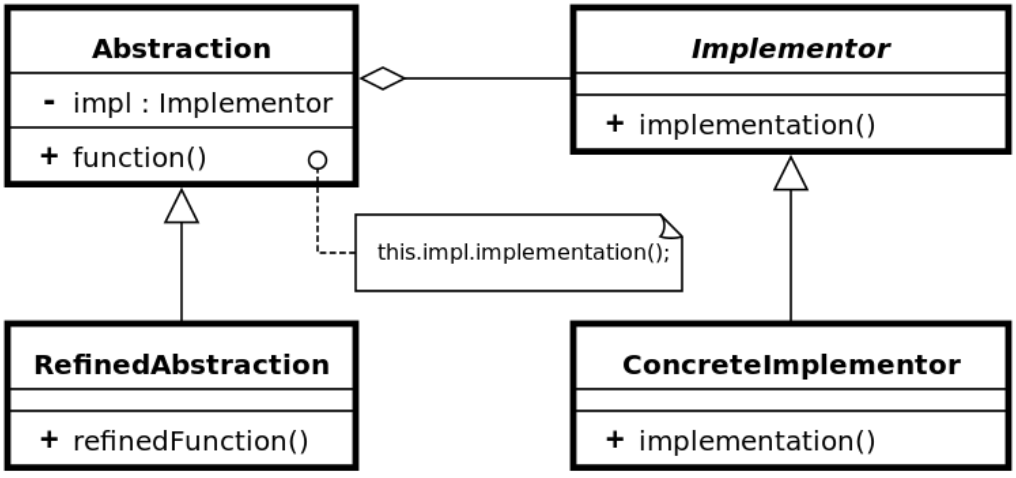
Most

Motivacija i osnovna ideja:

Pretpostavimo da želimo da implementiramo klasu za prikazivanje titla nekog videa i da kreiramo komercijalni softver sa osnovnim prikazom titla i premium verzijom koja daje veću kontrolu. Sam titl može biti dat kao .srt, .sub ili .txt fajl. Kako organizovati klase sa ovim zahtevima? Mogli bismo imati apstraktnu klasu Title i iz nje izvedene klase SrtTitle, SubTitle, TxtTitle, a iz njih dalje izvedene klase PremiumSrtTitle, PremiumSubTitle, PremiumTxtTitle (pod pretpostavkom da SrtTitle, SubTitle i TxtTitle nisu apstraktne već omogućavaju prikazivanje u osnovnoj verziji, a ako ih proglasimo za apstraktne onda su nam potrebne još tri klase: BaseSrtTile itd..). Zamislimo da osim osnovne i premium verzije imamo još međunivoa privilegovanog prikaza ili još neki format titla – stvari se komplikuju još više!

Projektni uzorak most (engl. Bridge) rešava opisani problem. Uzorak treba koristiti kada je **moguće razdvojiti *apstrakciju* od *implementacije***. To kako je titl implementiran (kao .srt, .sub ili .txt fajl) ne ograničava nas u pogledu toga kako ćemo nakon učitavanja taj titl prikazivati. Možemo razdvojiti *implementaciju* (format titla) od *apstrakcije* (prikazivanje i funkcionalnosti nad titlom) na način da u apstrakcija agregira implementaciju. U našem konkretnom primeru, klasa Title imala bi pokazivač na TitleFormat i klase izvedene iz TitleFormat bile bi zadužene za učitavanje titla. Nezavisno od hijerarhije implementacija (ili *implementatora*) imali bismo klasu PremiumTitle koja nasleđuje klasu Title.

UML dijagram projektnog uzorka Most dat je na slici 1.



Slika 1 Projektni uzorak Most

Zadaci:

1. Implementirati opšti oblik projektnog uzorka Most u skladu sa UML dijagramom na slici 1. Dodati još jednog konkretnog implementatora.
2. Primeniti projektni uzorak Most na situaciju opisanu u uvodu ovog fajla. Neka Title ima metodu parseFile(filename) koja prosleđuje poziv metodi istog imena klase TitleFormat.
3. Napisati klase Stack i LimitedCapacityStack i omogućiti da stek može biti implementiran pomoću niza ili pomoću vektora (preuzeto iz zadataka Dejana Kolundžije). Implementator steka bi trebalo da ima metode push(), pop(), empty(), size(), peek()[1]. Stack bi trebalo da u konstruktoru postavlja implementatora ili (više u duhu C++-a) da Stack bude šablon klase čiji je jedan od parametara šablona implementator[2].

Kompozit/kompozicija/sastav

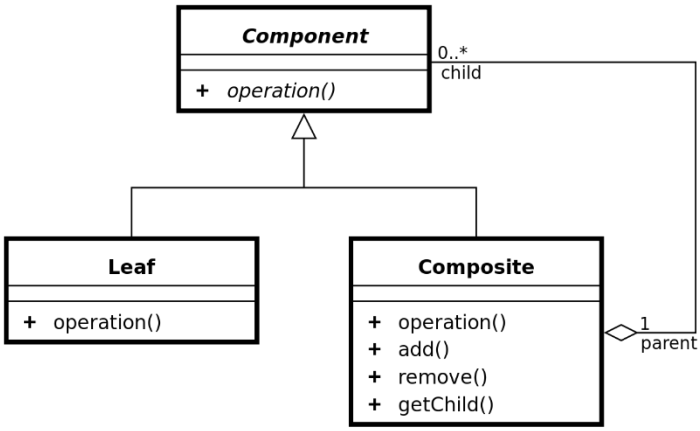
Motivacija i osnovna ideja:

Pretpostavimo da imamo klasu koja je sačinjena iz celina/delova koji su takvi da nad delovima možemo vršiti neke operacije koje možemo i nad celinom. Štaviše, možda se neki deo sastoji iz manjih delova koji opet imaju neka svojstva celine. Uzmimo za primer klase koje bi predstavljale html tagove. Neki tagovi (div recimo) se mogu sastojati od više drugih tagova, dok postoje i tagovi takvi da se ne mogu dalje deliti na tagove. U ovakvim situacijama klase možemo organizovati u skladu sa projektnim uzorkom kompozicija.

U osnovnom slučaju, potrebno je imati apstraktnu klasu Komponenta (html tag u našem primeru) i iz nje izvedene klase Kompozit (div, html i slični tagovi) koja se sastoji iz više komponenti (tagova) i List koja se ne sastoji od komponenti (tagovi koji ne mogu imati tagove kao delove). Klasa Komponenta treba da deklariše jednu ili više metoda koje je moguće izvršavati i nad delovima i nad celinom. Poziv metode nad objektom Kompozit klase obično treba da osim neke svoje logike izvrši i pozive iste metode nad delovima iz koga je sastavljen (ili da objedini rezultate, ako se vrši neka računica i sl.). U našem primeru, štampanje div taga uključuje štampu direktnog sadržaja div taga kao i štampanje svih komponenti iz kojih je div tag sačinjen. Naravno, klasa Kompozit treba da omogući i dodavanje/uklanjanje komponenti.

Učesnici projektnog uzorka kompozit i njihova komunikacija dati su UML dijagramom prikazanim na slici 2.

[1] Zapažanje: ukoliko VectorStackImplementator nasleđuje StackImplementator, a interno koristi std::vector i metode push_back(), back(), pop_back(), size(), empty(), VectorStackImplementator je ujedno i adapter za std::vector.
[2] U standardnoj biblioteci klasa stack implementirana je upravo kao šablonska klasa sa dva parametra: T i opcioni Sequence gde je sekvenca neka struktura poput std::vector ili std::list.



Slika 2 Projektni uzorak Kompozit

Zadaci:

4. Implementirati opšti oblik projektnog uzorka Kompozit na osnovu UML dijagrama sa slike 2.
5. Implementirati kreiranje json objekta i eksportovanje u fajl. Json format je veoma čest kod komuniciranja sa web servisima. Na sledećem linku dati su primeri json fajlova: https://json.org/example.html
   Json fajl se sastoji od parova ključ-vrednost, pri čemu vrednost može biti: string, null, broj, bool, lista (niz) i json objekat.
   Napisati apstraktnu klasu JsonElement sa čisto virtuelnom funkcijom exportToFile(std::ofstream&). Izvesti klase JsonObject koja ima parove key (string) – value (JsonElement*), JsonList koja ima listu (ili vektor) json elemenata i listove klase za ostale tipove. JsonList i JsonObject imaju ulogu kompozita.[3] Eksportovan fajl mora da bude formatiran sa svim potrebnim uvlačenjem pasusa.

[3] U ovom primeru vidimo da ne mora postojati samo jedna klasa kompozit i samo jedna klasa list.