14.01.2022.

Objektno-orijentisano programiranje 2

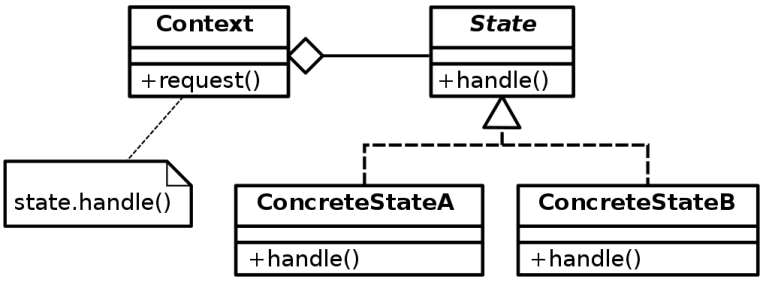
Stanje

Motivacija i osnovna ideja:

Relativno je česta situacija da neki objekat treba da ima drugačije ponašanje u zavisnosti od vrednosti nekih svojih atributa. Za "crno-bele" dileme (neki atribut je postavljen na određeni način ili nije) to znači da će se ponašanje određenih metoda svesti na oblik if (uslov) { izvrši jednu logiku } else { izvrši drugu logiku }. Sa dodavanjem atributa ili povećanjem broja relevantnih slučaja, broj grana, tj. različitih ponašanja određenih metoda se uvećava i kod veoma lako postaje kompleksan i podložan greškama. Slučajeve koje razmatramo možemo smatrati *stanjima* objekta. Pa bismo onda rekli kada je objekat u stanju s_1 određena metoda treba da se ponaša na jedan način, u stanju s_2 na drugi način, itd. Naravno, nakon što se metoda izvrši nad objektom koji je u nekom stanju, on će možda usled izvršenja te metode preći u neko drugo stanje. Umesto da metodu (ili veći broj njih, što je češći slučaj) opterećujemo svim mogućim slučajevima, tj. stanjima i unutar samih metoda regulišemo prelaze iz jednog stanja u drugo, možemo da za svaki slučaj definišemo posebnu klasu *stanja*. Svako konkretno stanje *zna* kako treba da se izvrše određene metode, kao i da li nakon njihovog izvršenja objekat (koji ćemo zvati *kontekstom*) treba da pređe u neko naredno stanje. Odgovarajuća matematička apstrakcija kojom bismo opisali prelaze objekta iz jednog stanja u drugo je *konačni automat.*

Projektni uzorak *Stanje* rešava problem dat u uvodu na opisani način. *Kontekst* ima *Stanje*, a pozivi relevantnih metoda nad klasom *Kontekst* delegiraju se objektu stanja. Konkretna stanja izvršavaju metode na odgovarajući način i mogu da, ukoliko je potrebno, promene stanje *Konteksta*. Da bi ovo bilo moguće, relevantne metode iz klase *Stanje* moraju da kao argument dobiju referencu ili pokazivač na *Kontekst* (na priloženom UML dijagramu metode su prikazane kao da nemaju argumente, što je pogrešno. Alternativa je da stanje ima pokazivač na Kontekst, što takođe nije prikazano na UML-u).

UML dijagram projektnog uzorka *Stanje* dat je na slici 1.



Slika 1: Projektni uzorak Stanje

14.01.2022.

Zadaci:

1. Osoba može da radi ili se zabavlja i može biti dobro ili loše raspoložena (jedna ista osoba može u jednom trenutku biti dobro, a u nekom drugom loše raspoložena). Način rada i zabave se razlikuje u zavisnosti od raspoloženja, a kad god radi ili se zabavlja postoji verovatnoća da će osoba promeniti raspoloženje.
   Ako je dobro raspoložena i radi, postoji verovatnoća od 50% da će postati loše raspoložena, a ukoliko je loše raspoložena 25% verovatnoće da postane dobro raspoložena. Ukoliko je osoba dobro raspoložena i zabavlja se postoji verovatnoća od 10% da će postati loše raspoložena, a ukoliko je loše raspoložena i zabavlja se, verovatnoća od 70% da će joj se raspoloženje popraviti.
2. Implementirati konačni automat Milijevog tipa koji raspoznaje sekvencu bitova 1101 i tom prilikom vraća 1, a u suprotnom 0. Sekvence mogu da se preklapaju. Stanja automata implementirati kao unikate ili muve.

   Zadatak za samostalnu vežbu:
   Simulirati sistem za rad aparata za kupovinu kafe. Aparat ima dugmad za odabir napitka, dugmad za kontrolu količine šećera, ulaz za papirni novac, dugme za kusur, dugme za sipanje odabranog napitka i sensor koji detektuje da li je napitak preuzet ili ne.
   U početku, aparat je u stanju pripravnosti i tada je moguće podešavati količinu šećera, odabrati napitak, ubaciti novac… Pritisak na dugme za kusur vratiće novac, a pritiskom na dugme za odabir napitka aparat prelazi u stanje “napitak odabran”. Pozivom metode za sipanje napitka proverava se da li je količina novca dovoljna. Ako jeste, aparat prelazi u stanje pripreme napitka kada ne dozvoljava nijednu operaciju u narednih 10 sekundi, osim vraćanja kusura. Nakon što prođe bar 10 sekundi, pozivom na bilo koju metodu aparat prelazi u stanje “gotov napitak” uz poruku da je napitak gotov i izvršava datu metodu u tom stanju. Uzimanjem napitka aparat prelazi u stanje pripravnosti.
   Za merenje vremena koristiti (potrebno je uključiti <chrono> i <thread> biblioteke):

```
auto start = std::chrono::system_clock::now();
auto end = std::chrono::system_clock::now();
auto elapsed = std::chrono::duration_cast<std::chrono::seconds>(end - start)
.count();
```

   Za čekanje x sekundi koristiti (u glavnoj funkciji možemo proveravati na svake dve sekunde da li je napitak gotov recimo):

```
std::this_thread::sleep_for(std::chrono::seconds(x));
```

Razmisliti o kompleksnosti klase AparatZaKafu kada se ne bi koristio projektni uzorak *Stanje*.

14.01.2022.

Strategija

Motivacija i osnovna ideja:

Pretpostavimo da imamo klasu čiji objekti treba da izvršavaju neku metodu čije ponašanje želimo da menjamo u vreme izvršavanja. Ovde se ne radi o tome da u zavisnosti od nekog stanja objekta treba implementirati drugačije ponašanje (tome služi projektni uzorak *Stanje*), već prosto o pružanju fleksibilnosti izbora načina implementacije i menjanja implementacije neke metode u vreme izvršavanja. Projektni uzorak *Strategija* rešava ovaj problem. Naime, slično kao kod projektnog uzorka *Stanje*, implementaciju metoda (najčešće jedne metode) prepuštamo posebnim objektima koje nazivamo *Strategije.* Objekat ima pokazivač ili referencu na *Strategiju*, a pozivom određene metode samo se prosleđuje poziv datom objektu *Strategije*.

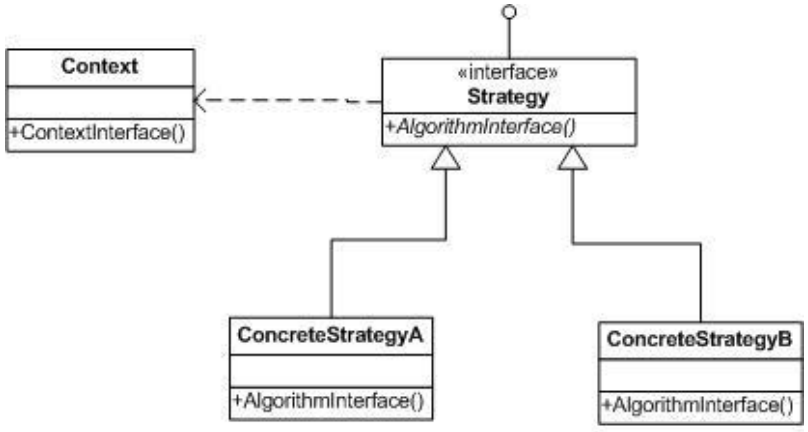
U slučaju da je motivacija za nasleđivanje neke klase samo izmena jedne metode, a klasa uz osim te jedne metode pruža i mnoge druge funkcionalnosti, često je bolji pristup korišćenje *Strategije*. Takav slučaj imali smo u primeru banke/posrednika gde su konkretne banke na drugačiji način obračunavale proviziju. To bi značilo da nećemo izvoditi nove banke, već će osnovna klasa banka imati pokazivač (ili referencu) na strategiju obračuna provizije.

Različite strategije mogu davati iste rezultate, ali ne moraju (primer sa bankama koje na različit način računaju proviziju i dolaze do različitog rezultata). Ako daju iste rezultate, onda je motivacija za različite strategije efikasnost implementacije za različite ulaze (na primer, ako množimo dve “male” matrice možemo koristiti jedan algoritam, a za “velike” neki drugi algoritam koji koristi izračunavanje na GPU-u, na primer).

Klasu koja koristi *Strategiju* nazivamo *Kontekstom* (kao u slučaju projektnog uzorka *Stanje*).

Koje su razlike između projektnih uzoraka *Most*, *Stanje* i *Strategija*?

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



Slika 2: Projektni uzorak Strategija

Zadaci:

3. Implementirati opšti oblik projektnog uzorka Strategija u skladu sa UML dijagramom sa slike 2.
4. Napisati klasu Niz kojom se implementira niz celih brojeva. Napisati metodu sortiraj kojom se dati niz sortira neopadajuće i omogućiti zadavanje algoritma sortiranja. Implementirati selection sort i counting sort algoritme. Kreirati dva niza od po 100.000 elemenata iz skupa {0, 1, 2, … , 9}, testirati algortime za sortiranje i uporediti vreme izvršavanja.
5. Napisati klasu Osoba (atributi ime, prezime, godina rođenja) i omogućiti različite načine eksportovanja osoba u fajl (kao csv, json, xml).