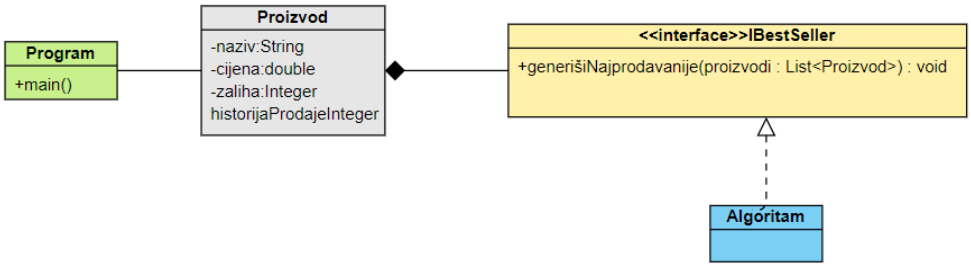
Ayana cvjećara

Paterni ponašanja

*patterni koji su boldirani će sigurno biti primijenjeni

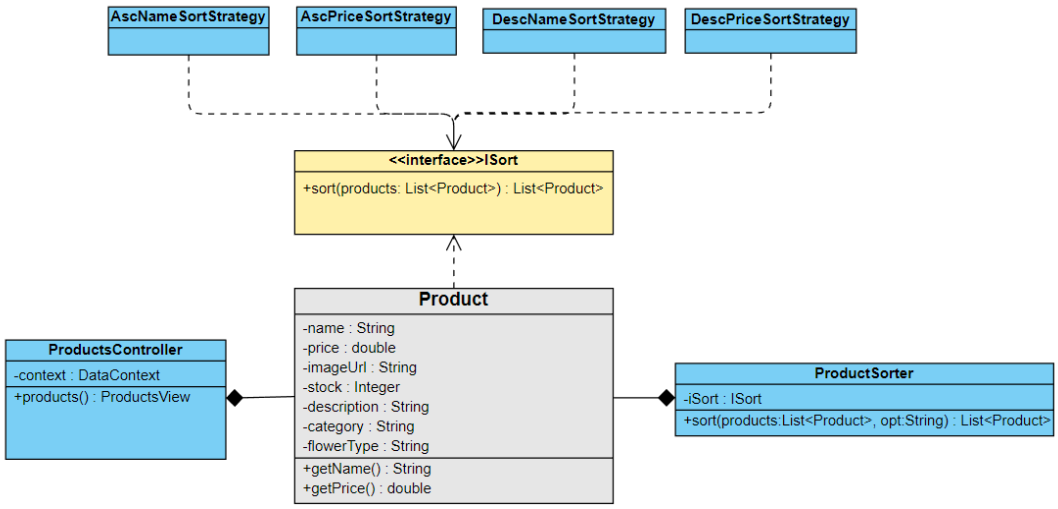
1. Strategy pattern

Strategy pattern izdvaja algoritam iz matične klase I uključuje ga u posebne klase. Pogodan je za primjenu u slučaju kada postoji više načina (algoritama) koji mogu biti primijenjeni na neki problem. U našem sistemu postoji funkcionalnost *best sellers* koja će najprodavanije proizvode prikazati na *home page* kao I pri odabiru neke od kategorija ili podkategorija proizvoda. Sistem ih generiše I ažurira na osnovu broja prodanih primjeraka u posljednjih mjesec dana jer smo mi odlučile da će to biti parametri po kojima algoritam funkcioniše.



Ovaj pattern je u našem sistemu prisutan također u sortiranju proizvoda pri pretraživanju. Korisnik može vršiti pretraživanje po nazivu I cijeni . Prikaz rezultata pretrage automatski je sortiran rastuće, ali korisnik ima I mogućnost odabira druge vrste sortiranja prema želji I potrebi.

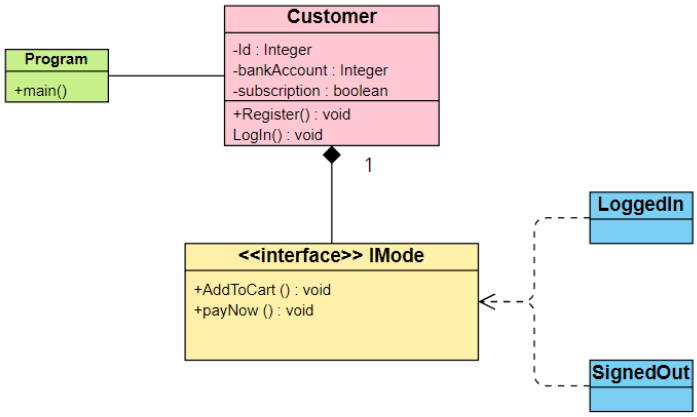
Nakon implementacije, dijagram izgleda ovako:



2. *State pattern*

Navedeni pattern mijenja način ponašanja objekata na osnovu trenutnog stanja. Stanje se ne mijenja po želji korisnika, nego automatski prema definiciji sistema. State pattern je dinamička verzija Strategy pattern-a.

U našem sistemu, potencijalni kupac koji ima napravljen korisnički račun, ali nije izvršio prijavu (dakle, uradio je *register*, ali nije *log in*) ima isti tretman I mogućnosti kao I korisnik kojeg nazivamo gost (dakle, onaj ko nije izvršio register I ne posjeduje korisnički nalog). Dok se ne prijavi, korisnik ne može izvršiti kupovinu, nego ima samo mogućnost pregleda proizvoda iz ponude I njihovih detalja kao I gost.



State pattern ostao je isti nakon implementacije.

3. *Template method pattern*

Template pattern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se koristi kada se za neki algoritam uvijek trebaju izvršiti isti koraci uz moguće izmjene za pojedinačne korake.

Ovaj pattern nećemo implementirati, ali moguća primjena bi bila u Log in funkcionalnosti. Koraci odnosno potrebne informacije za prijavu su uvijek iste, kao I izgled stranice koji se otvori nakon prijave. Međutim, prilikom prve prijave nakon kreiranja računa, mogle bismo korisniku poslati email dobrodošlice ili mu poruku prikazati na ekranu.

4. *Observer pattern*

Uloga Observer pattern-a je da uspostavi relaciju između objekata tako da kada jedan objekat promijeni stanje, drugi zainteresovani objekti se obavještavaju. U našem sistemu, mogle bismo iskoristiti ovaj pattern pri slanju notifikacije *"Nije potreban poseban povod"* koja će biti poslana korisniku ukoliko nije izvršio kupovinu određeni vremenski period. Također, slična primjena bi bila ukoliko bismo slali obavijest

korisnicima o sniženju tj.popustu na određene proizvode ili popustu na cjelokupnu ponudu za specijalne prilike, odnosno kodu za popust.

5. Iterator pattern

Ovaj pattern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija struktuirana. Nismo implementirale ovaj pattern, ali potencijalna ideja za implementaciju može biti da uvedemo opciju pregleda best seller proizvoda prema prilici ili prema cijeni. Za to bi nam bile potrebne dvije klase, OcassionIterator I PriceIterator koje će imati metodu nextProduct():Product. Klase bi nasljeđivale interface IKreatorIterator sa metodama za kreiranje iteratora.

6. Chain of Responsibility pattern

Chain of responsibility pattern namjenjen je da bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načine procesiraju podatke. Svoju primjenu ovaj pattern može pronaći u našoj funkcionalnosti izvršenja narudžbe. Cjelokupan proces naručivanja započinje pregledom proizvoda I detalja I odabirom proizvoda. Zatim, proizvod se ubacuje u korpu, nudi se mogućnost kreiranja personal card. Nakon toga, unosi se kod za popust ukoliko postoji te se eventualno ažurira cijena I bira način plaćanja I uslovi preuzimanja narudžbe. Konačno, korisnik dobiva potvrdu o uspješnoj narudžbi na email. Ova kompleksna aktivnost podijeljena je na više podaktivnosti I u njoj učestvuje više aktera kao I klasa I kontrolera.

7. Mediator pattern

Mediator pattern služi kao međuobjekt da bi se izbjeglo direktno povezivanje velikog broja objekata. Mediator je zadužen za komunikaciju između objekata, odnosno za prosljeđivanje poruke od jednog objekta do drugog. U našem sistemu, kako je već naglašeno ranije, nema kompleksnih veza ni mnogo povezanih klasa tako da ovaj pattern trenutno nije pogodan za korištenje.