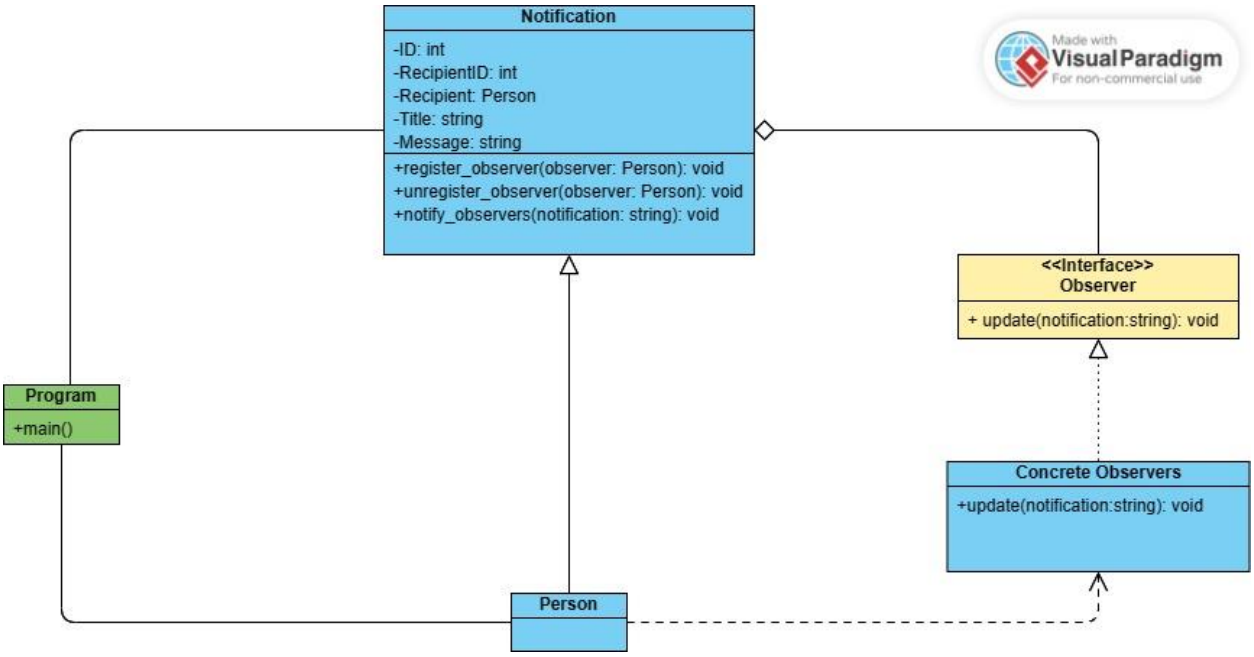
Paterni ponašanja

Observer pattern

Observer patternom postižemo labavu vezu između klase Notification i posmatrača. Klasa Notification ne mora imati eksplicitno znanje o specifičnom ponašanju ili detaljima implementacije svojih posmatrača. Jednostavno šalje obavještenja, a posmatrači njima postupaju u skladu sa svojim zahtjevima.

Ovaj obrazac pruža fleksibilnost, jer možemo lako dodati nove tipove posmatrača ili modifikovati ponašanje postojećih posmatrača bez modifikacije klase Notification. Promoviše proširivost, mogućnost održavanja i razdvajanje problema razdvajanjem pošiljaoca i primaoca obavijesti.

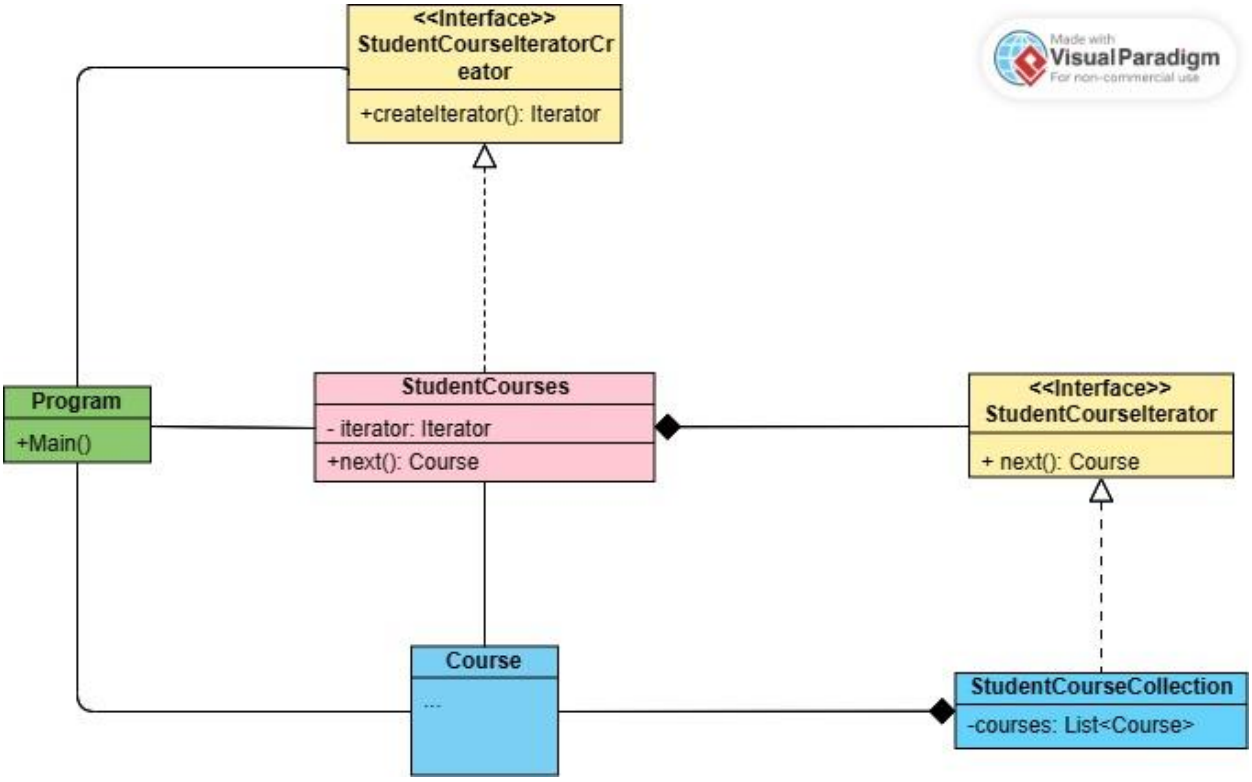


Iterator pattern

Prednosti korištenja Iterator patterna:

- Pruža standardni način za ponavljanje različitih tipova kolekcija, promovišući ponovnu upotrebu koda i smanjujući povezivanje između klasa klijenta i kolekcije.
- On enkapsulira logiku iteracije, omogućavajući kolekciji da promijeni svoju internu strukturu bez utjecaja na klijentski kod.
- Pojednostavljuje klijentski kod pružajući konzistentan i unificiran interfejs za ponavljanje različitih kolekcija.
- Podržava višestruke istovremene iteracije preko iste kolekcije bez međusobnog ometanja.

Sve u svemu, obrazac Iterator pomaže u odvajanju problema prelaska i prikupljanja, pružajući čist i fleksibilan način iteracije preko kolekcija objekata.



Chain of Responsibility

Chain of Responsibility pattern može se primijeniti kada imamo niz objekata koji mogu rukovati zahtjevom ili izvršiti određenu radnju i želimo dinamički odrediti koji objekt treba da obrađuje zahtjev na osnovu njegovog tipa ili drugih kriterija.

Ovaj pattern se može primijeniti nad Notifikacijama, gdje kako notifikacije moraju biti poslane različitim primaocima, možemo napraviti lanac rukovatelja notifikacija, gdje je svaki zadužen za specifični tip primaoca. Svaki rukovatelj može odlučiti da li može obraditi notifikaciju ili da je proslijedi sljedećem rukovatelju u lancu.

Mediator pattern

Mediator pattern se može primijeniti kada imamo skup objekata koji međusobno djeluju i želimo da odvojimo njihovu komunikaciju uvođenjem objekta posrednika. Objekat posrednik olakšava komunikaciju i koordinaciju između objekata, smanjujući njihove direktne zavisnosti i pojednostavljujući njihove interakcije.

Ovaj pattern može biti primijenjen nad zakazivanjem ispita. Npr. ExamScheduler može služiti kao mediator, odgovoran za koordiniranje komunikacije između *Exam*, *ExamType*, *ExamRegistration* i *StudentExam* objekata. Može podržavati registrovanje studenata za ispite, dodjeljivanje tipa ispita i ažuriranje rasporeda ispita.

Strategy pattern

Strategy pattern se može primijeniti kada imamo skup algoritama ili ponašanja koji se mogu dinamički birati ili mijenjati u vrijeme izvođenja. Omogućava nam da inkapsuliramo svaki algoritam ili ponašanje u posebnu klasu, čineći ih zamjenjivim i lako proširivim.

U našem slučaju, može se koristiti kada imamo različite mehanizme notificiranja (npr. SMS,email,push-notifikacije...). Možemo primijeniti pattern definiranjem *DeliveryStrategy* klasa, gdje svaka implementira specifičan mehanizam dostave.

State pattern

State pattern se može primijeniti kada imamo objekt koji mijenja svoje ponašanje na osnovu svog unutrašnjeg stanja, a mi želimo da inkapsuliramo logiku koja se odnosi na svako stanje u zasebne klase. Omogućava objektu da dinamički mijenja svoje ponašanje u vrijeme izvođenja promjenom svog unutrašnjeg stanja.

Može se primijeniti kod Request Approval Workflow, gdje možemo imati skup *RequestState* klasa, gdje svaka predstavlja specifično stanje. *Request* klasa može imati referencu na trenutni *RequestState* objekat, koji enkapsulira logiku i radnje povezane sa tim stanjem.

Template method pattern

Template method pattern se može primijeniti kada imamo algoritam ili proces koji slijedi zajedničku strukturu, ali dozvoljava da određene korake implementiraju podklase. Definira kostur algoritma u osnovnoj klasi, s nekim implementiranim koracima i ostalim koracima koji su ostali za podklase.

Ovo može biti primijenjeno na predavanje zadaća, gdje ako želimo definirani česte procese za predavanje zadaća ali da dozvoljava fleksibilnost u načinu predaje. Napravimo baznu klasu *HomeworkHandler* sa template metodom *subitHomework(). HomeworkHandler* može definisati generalne korake predaje zadaće (npr. Validacija predaje, čuvanje predaje, notifikacija studentu).