Strukturalni paterni i dijagrami interakcija – Cinestvar

Na idućoj stranici nalazi se grafički prikaz primjene odabranih strukturalnih patterna, a prije toga ćemo pojasniti ulogu iskorištenih, te potencijalnu primjenu neiskorištenih patterna ovog tipa.

1. **Adapter**

Ovaj patern smo iskoristili u njegovoj Object varijanti, za razliku od Class varijante koja je prikazana u predavanju. Klasa sala predstavlja adaptee klasu, te ima metodu koja vraća raspored sale u obliku bool matrice gdje false (0) predstavlja razmak (prolaz), a true (1) predstavlja sjedište. Interfejs zahtijeva novi standard – raspored sa naznačenim zauzetim sjedištima. Klasa Termin igra ulogu Adapter klase, te implementira traženu funkcionalnost, prvo dobavljajući raspored sale, a zatim prepravljajući isti na osnovu svog spiska zauzetih sjedišta.

1. **Façade**

Ovaj patern nismo odlučili implementirati. Da je implementacija bila apsolutno neophodna, mogli smo spojiti prozor za rezervaciju filma i prozor za detaljne informacije o filmu u jedan pogled, bez gubitka metoda koje se inače koriste. U tom slučaju, imali bismo funkcije koje daju detalje o filmu (klasa Film), i paralelno bismo mogli upravljati našom rezervacijom (klasa Rezervacija, nastavlja se na Termin), što su na dijagramu dvije distancirane klase i imaju vlastite podsisteme.

1. **Decorator**

Dekorater bi se mogao iskoristiti na salama, ukoliko bismo prešli sa broja kolona i redova na nešto apstraktnije kao oblik sale i slično. Onda bi se metoda dajRaspored() na dijagramu mogla delegirati na više dekoratera, koji bi crtali različite krnje matrice bool-ova za različite sale (regularna sala, tribina, amfiteatar, itd). Međutim, uvjerenja smo da bi takva implementacija stvorila više problema nego rješenja, pogotovo jer je sve manji broj kina u pravom životu sa varijabilnim salama, tako da ne može naći prostora u našoj aplikaciji.

1. **Bridge**

Još jedan neiskorišteni patern. On bi svoju namjenu mogao pronaći ukoliko bismo naprimjer htjeli dodati novu vrstu rezervacije, tipa da se može rezervisati čitava sala za neku proizvoljnu projekciju, mimo aktuelne filmske ponude. U tom slučaju bi bilo potrebno da nam funkcionišu i postojeće opcije, ali i nova, proširena varijanta.

1. **Composite**

Patern je iskorišten za dobavljanje informacije o broju preostalih raspoloživih sjedišta. Ovaj podatak će biti korišten pri provjeri da li je preostalo dovoljno mjesta u terminu koji je korisnik odabrao. U slučaju da nije, provjerit će se da li je u ostalim terminima za taj film za taj dan ostao dovoljan broj raspoloživih sjedišta.

1. **Proxy**

Pattern bi mogao iskorišten za validaciju prijavljenosti korisnika u sistem, prije omogućavanja funkcionalnosti rezervacije filma.

1. **Flyweight**

Ovaj strukturalni pattern se koristi da se rasterete individualne jedinke klase zajedničkom memorijom. Ovaj patern bi se mogao iskoristiti na klasu Film ukoliko želimo dodati neke vizuelne elemente ili dodatne podatke (recimo svaki žanr ima svoj .png logo, ili cijene sličnih filmova su jednake, i sl). Onda bismo već sad mogli predostrožno napraviti flyweight opciju gdje imamo Film, u kojem je atribut tip iz klase TipFilma, koja bi bila taj dijeljeni memorijski repozitorij za logo svakog žanra i za cijene ili neke kompleksnije stvari koje nije isplativo čuvati u memoriji svake pojedninačne instance Film.

Radi lakšeg snalaženja, ispod je prikazan segment dijagrama u kojem se nalaze dodani strukturalni patterni.

|  |
| --- |
|  |
| *Object adapter pattern (lijevo) i Composite pattern (desno)* |