Paterni ponašanja

- **Strategy patern -** Posmatrajući naš sistem, sa trenutno zamišljenim sistemom, nismo uočili mjesto da odmah implementujemo ovaj patern. Međutim uočili smo odličnu priliku za proširenjem našeg sistema, čime bismo ujedno i implemetnovali ovaj patern. Ideja je da dodamo par strategija za raspoređivanje studenta u sobe kada on bude tek upisan. Tako bi klasa StudentskiDom imala atribut npr. Raspored, koji bi unutar sebe imao atribut tipa Strategy, koji bi predstavljao interfejs. Sada bi različite strategije implementirale ovaj interfejs na svoj način, a željena strategija bi se postavljala prilikom upisa studenta. Na ovaj način bismo nastojali omogućiti da studenti budu zadovoljni prvobitnim rasporedom i da smanjimo broj zahtjeva za premještanje.
- **State patern -** Što se tiče ovog paterna, nismo odlučili njega implementirati, ali pri eventualnom proširenju našeg sistema, ukazala bi nam se prilika za to. Mogli bismo po potrebi, postaviti neke nivoe popunjenosti paviljona. Npr.paviljon bi se mogao naći u nekom od stanja tipa prazan, srednjePopunjen i sl. Prilikom dodavanja novog studenta provjerila bi se trenutna popunjenost i na osnovu nekog praga, bi eventualno prešao u drugo stanje. Ovo bi nam koristilo prilikom smještanja studenta u dom, gdje bi se na osnovu nekog nivoaPrioriteta paviljona (a koji bi se postavljao u zavisnosti u kojem je stanju paviljon) odredilo gdje bi studenta najbolje bilo smjestiti. Trenutno stanje bismo čuvali kao atribut klase Paviljon, a također bio bi napravljen I neki interfejs Stanje, koji bi implementirala željena stanja. Dodali bismo metodu tipa upisiStudenta, koju bi implementovale sve klase stanja.
- **Template method patern -** Priliku za implementaciju ovog paterna smo uočili prilikom bilo kakvog prikaza i sortiranja Studenta. Ideja je da napravimo interfejs koji bi unutar sebe imao metode tipa filter i sort. Nakon toga da imamo razlicite tipove klasa koje bi predstavljale kriterije za filtriranje i sortiranje, odnosno ove klase bi implementovale ovaj interfejs na svojstven način. Također imali bismo i jednu klasu Algoritam koja bi kao parametre svojih metoda, imala navedene tipove klasa kriterija, a unutar njih bi se pozivale određene metode svojstvene za proslijeđenu klasu kriterij. Implementacijom ovog paterna olakšavamo rad uprave i činimo spisak studenata preglednijim.
- **Observer patern -** Ovaj patern nismo odlučili implementovati, ali smatramo da nam ne bi bio prevelik problem da implementujemo po potrebi. Moguća prilika za implementaciju ovog paterna bi bila u slučaju da hoćemo da uvedemo neki sistem obavijesti za studente. Imali bismo klasu koja bi čuvala listu onih koji žele da primaju određeni tip obavijesti. Nakon toga ako bi StudentskiDom htio poslati npr. obavijest da neće raditi par dana, onda bi se obavijest poslala svim Studentima koji su prijavljeni za taj vid obavijesti. Odnosno, kao atribut klase StudentskiDom imali bismo klasu Obavijesti koja bi unutar sebe imala listu pretplaćenih studenata. Sada bismo imali metodu neku unutar klase StudentskiDom koja bi pozivala metodu iz klase Obavijesti tipa notify(). Također imali bismo i interfejs IObavijesti, koju bi iplementovala klasa Student i koji bi imao neku metodu update. Sada bismo, pozivom metode notify, interno pozivali i metodu update za sve one studente koji su pretplaćeni.

- **Iterator patern -** Odlučili smo da nećemo implementovati ovaj patern, međutim postoji mogućnost njegove implementacije po potrebi. Prilika za upotrebu ovog paterna, bi se javila ako bismo npr. htjeli neku informaciju uzeti od onih studenata koji se nalaze u specifičnom paviljonu. Na ovaj način bi nam naš iterator pomogao, jer ne bismo pristupali svim studentima studentskog doma, već samo onima koji se nalaze u navedenom paviljonu. Imali bismo standardni interfejs za iteratore koji bi imao metode getNext i hasNext. Također napravili bismo i klasu paviljonIterator koji bi kao parametar konstruktora imao paviljon kroz koji želimo da iteriramo,te koji bi nasljeđivao navedeni interfejs. Imali bismo i dvije pomoćne varijable, koje bi redom predstavljale brojSobe i brojTrenutnogStudentaUSobi, koji bi inicijalno bili postavljeni na 0. Sada bismo prilikom iteracije kroz listu soba, prvo provjerili da li smo došli do kraja liste naših soba, poredivši naš atribut brojSobe s veličinom liste. U slučaju da nismo došli do kraja liste, dokle god ima studenata u sobi povećavali bismo brojTrenutnogStudentaUSobi za jedan. Kada bismo došli do kraja ove liste studenata, varijablu brojTrenutnogStudentaUSobi bismo postavili na 0, a broj sobe bismo povećali za jedan.