Laborator 3 – Design Patterns

Obiective

După completarea acestui laborator veți dobândi următoarele cunoștințe:

- Implementarea design pattern-urilor
- Folosirea mecanismelor puse la dispozitie de .NET pentru a vă standartiza aplicațiile

➢ Exercițiul 1 - Observer Pattern

În acest exercițiu veți implementa un observer pattern, plecând de la următorul scenariu. Avem un magazin de ziare și clienți ce doresc să fie informați în momentul în care se primesc noi reviste. Pentru a face acest lucru , fiecare client se va abona la un singur magazin. Un magazin poate avea oricâți abonați. Clienți vor fi anunțați ori de câte ori magazinul primeste un număr nou al unei reviste și vor decide dacă doresc să o cumpere. Un client poate dori să cumpere mai multe tipuri de reviste.

Deschideti scheletul de cod din proiectul **Observer**

- Faceți astfel încât Store să implementeze interfața **IObservable<Magazine>**
- Adăgați metodele interfației **IObservable**

HINT: Pentru a adăga metodele unei interfețe faceți click pe interfață și apăsați CTRL + . urmat de tasta Enter

- Faceți astfel încât **Client să** implementeze interfața **IObserver<Magazine>**
- Adăugați metodele interfeței **IObserver**

Clasa **Store**

- Creați în interiorul clasei un tip de delegat, privat, cu aceeași signatură ca metoda OnNext a clasei Client, numit **MagazineReceived**.
- Creați un eveniment privat folosind tipul delegatului **MagazineReceived**, numit **receiving**.
- În interiorul clasei, creați o nouă clasă privată, denumită **Unsubscriber** ce implementează intefața **IDisposable.**
- În clasa **Unsubscriber** creați două câmpuri private. Primul camp va avea tipul **Store** și se va numi **store**, iar al doilea câmp va avea tipul MagazineReceived și se va numi **handler**.
- Creați un consctructor pentru clasa **Unsubscriber** având doi parametrii corespunzători celor două câmpuri și setați-le
- Adăugați clasei **Unsubcriber**, metodele intefeței **IDisposable**

- Asigurați-vă că aveți inclus **using System.Linq**
- În continuare vom implementa metoda **Dispose** a clasei **Unsubscriber.** În această metodă, vom verifica daca avem vreun delegate abonat la evenimentul **receiving.** Dacă da, vom folosi LINQ împreună cu o lambda expresie pentru a verifica dacă printre delegații înscriși la evenimet este vreunul care are acelasi target ca delegatul **handler**. Pentru aceasta veți folosi metoda **Any**, pusă la dispoziție de **LINQ** pe vectorul de delegați înscriși la evenimentul **receiving** și să vă definiți o lambda expresie ce verifică dacă un delegat x are același target ca delegatul **handler**. Dați această lambda expresie ca parametru metodei Any.
  Dacă sunt îndeplinite cele 2 condiții, vom dezabona delegatul de la eveniment.

  Încercați să scrieți această condițiile singuri. Dacă totuși nu reușiți, veți găsi la finalul laboratorului o căsuță cu titlul **Răspuns 1.** Rugați instructorul să vă explice răspunsul, dacă aveți neclarități.
- Acum vom implementa metoda **Subscribe** a clasei **Store**. Instanțiați un delegat pentru metoda **OnNext** a observer-ului primit ca parametru. Folosind o condiție similară cu cea de mai devreme, verificați dacă evenimentul **receiving** are abonați delegați și dacă printre aceștia NU se află un obiect având ca target observer-ul primit.
  Dacă cele 2 condiții sunt respectate, abonați delegatul la eveniment și retunați un nou **Unsubscriber** ce are ca **store,** obiectul current și **handler,** delegatul nou creat. Dacă nu, returnați **null**.
- Anuntați toți abonații atunci când metoda receive este apelată folosind evenimentul **receiving**

Clasa **Client**

- Adăugați urmatoarele câmpuri private: un camp de tip **IDisposable** numit **unsubscriber** și un câmp de tip **Store** numit **store.** Ambele câmpuri trebuie să fie nule. Un câmp String numit name și o proprietate de get pentru acesta
- Implementați metoda **OnNext** ce verifică dacă revista primită de magazine se află printre revistele dorite, iar dacă da și o poate cumpăra. Afișați un mesaj corespunzător.
- Decomentați metodele **Subscribe** și **Unsubscribe.**

  Testați design-ul.

Răspuns 1

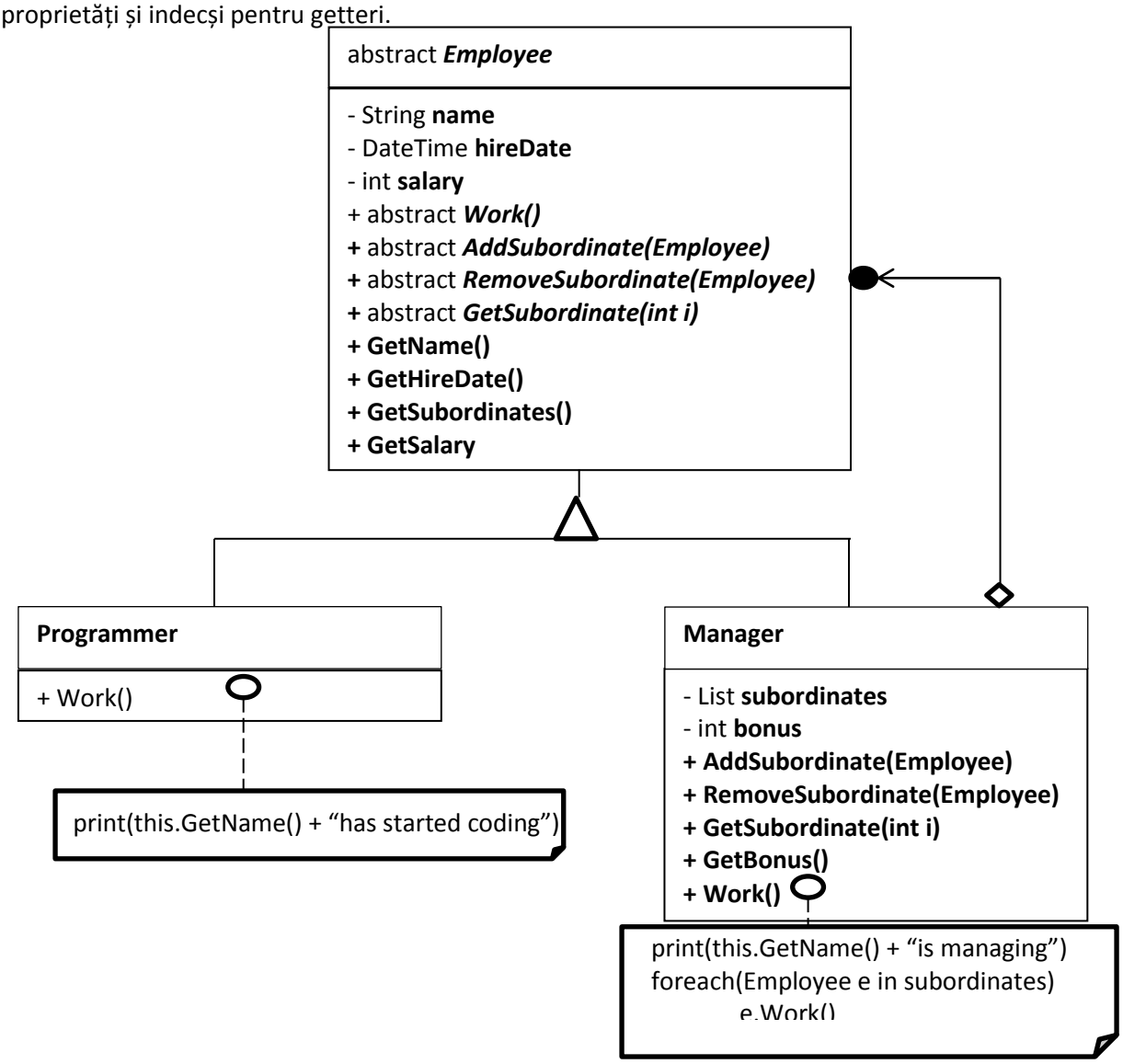
```
parent.receiving != null &&

parent.receiving.GetInvocationList().Any((Delegate x) => handler.Target == x.Target)
```

➢ Exercițiul 2 - Composite Pattern

În acest exercițiu veți implementa un composite pattern, pornind de la următorul scenariu: Avem o companie, în care agajații sunt fie programatori, fie manageri. Un programator poate avea un singur manager, iar un manager poate avea mai mulți subalterni. Subalternii unui manager pot fi la rândul lor manageri sau programatori. Fiecare angajat va lucra in felul său. Managerii îi vor pune la lucru pe subalterni, iar programatorii vor coda. Fiecare angajat va avea un nume, o dată a angajării și un salariu. Pe lângă salariul fix, un manager va avea și un bonus.

În acest exercițiu nu veți mai fi ghidați pas cu pas în implementare. În schimb, aveți la dispozitie această diagramă UML care vă arată design-ul companiei pe care trebuie s-o înțelegeți și după care vă veți ghida în implementare. Detaliile de implementare sunt la latitudinea voastră. Recomandarea este să folosiți proprietăți și indecși pentru getteri.



➢ Exercițiul 3 - Visitor Pattern

În acest exercițiu veți implementa Visitor Pattern, pentru a adăuga diferite operații pe ierarhia implementată la exercițiul 2.

- Creați o interață **IVisitor** să conțină o metodă **Visit** supraîncărcată pentru **Programmer** și **Manager**
- Adăgați o metodă abstractă **Accept(IVisitor** v**)** clasei **Employee**. Implementați metoda în clasele derivate. Corpul metodei trebuie să fie **v.Visit(this)**

În continuare vom implementa un visitor ce va calcula salariul mediu în companie.

- Creați o clasă **MediumSalary** ce implementează interfața **IVisitor.**
- Adăugați acestei clase două câmpuri. Un câmp de tip **float** numit **totalSalary**. Un câmp de tip **int** numit **nrEmployees**. Ambele inițializate cu 0.
- În metoda **Visit(Programmer)** adunați la totalSalary salariul programatorului și incrementați numărul de angajați.
- În metoda **Visit(Manager)** adunați la totalSalary salariul manager-ului și bonusul, incrementați numărul de angajați și apelați metoda Accept folosind acest visitor a tuturor subalternilor managerului.
- Adăgați un getter pentru salariul mediu ce va returna salariul total, imparțit la numărul de angajați. Aruncați o excepție dacă visitor-ul nu a fost folosit.

Creați o ierarhie, instanțiati un visitor și testați design-ul.