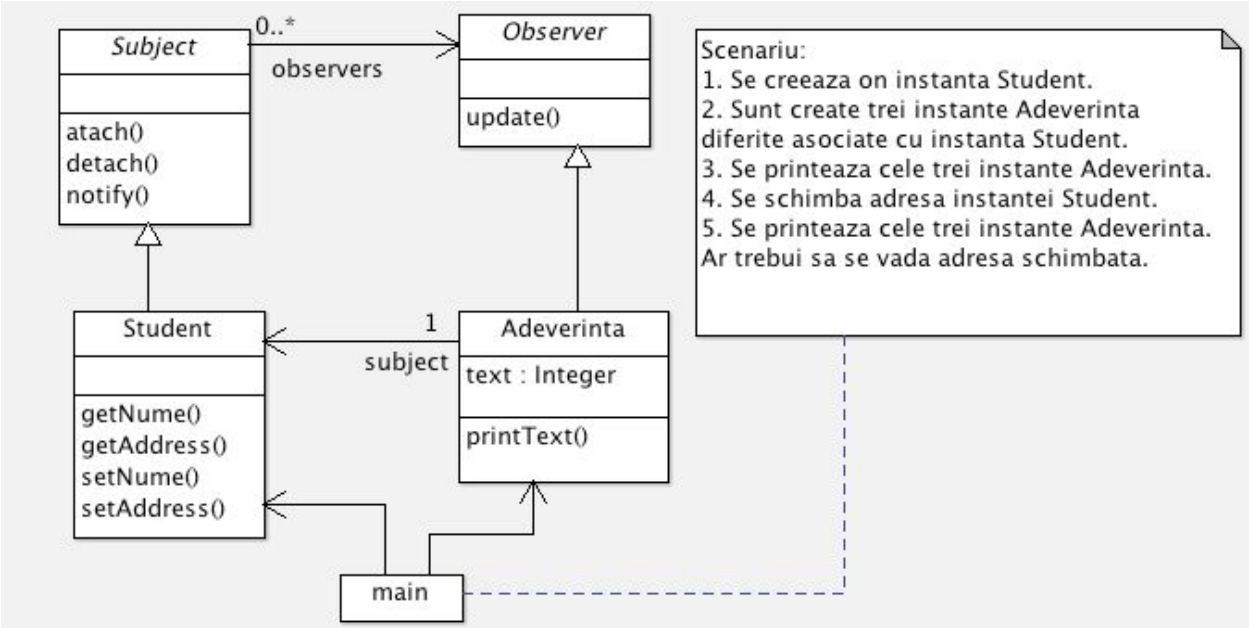
Observatii generale.

1. Subiectul pe 2 pagini:
    a. prima pagina diagrama si descrierea ei, cerintele.
    b. pagina a doua punctajul
2. Observatiile DL de la subiectul se aplica la toate.
3.

Nume, Prenume:
Grupa:

1.

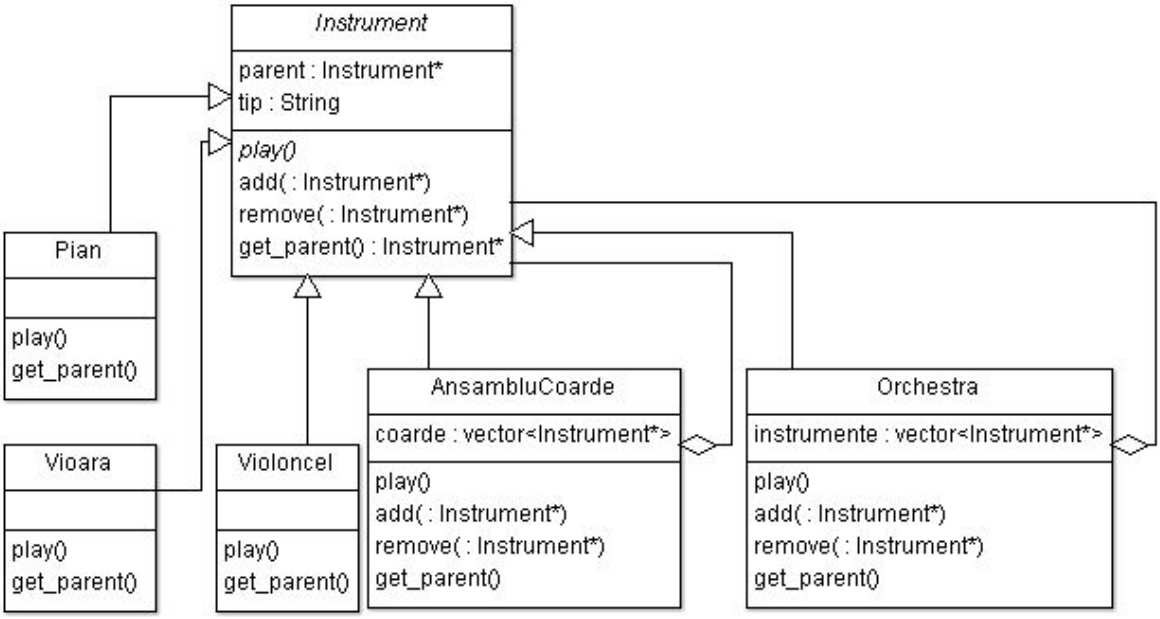


Punctaj	Operatie
4p	Codul care include signatura claselor si relatiile dintre ele compileaza fara eroare
2p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”
3p	Implementarea corecta a claselor *Subject* si *Observer*,
3p	Implementarea corecta a clasei Student
3p	Implementarea corecta a clasei Adeverinta

Nume, Prenume:

Grupa:

2.



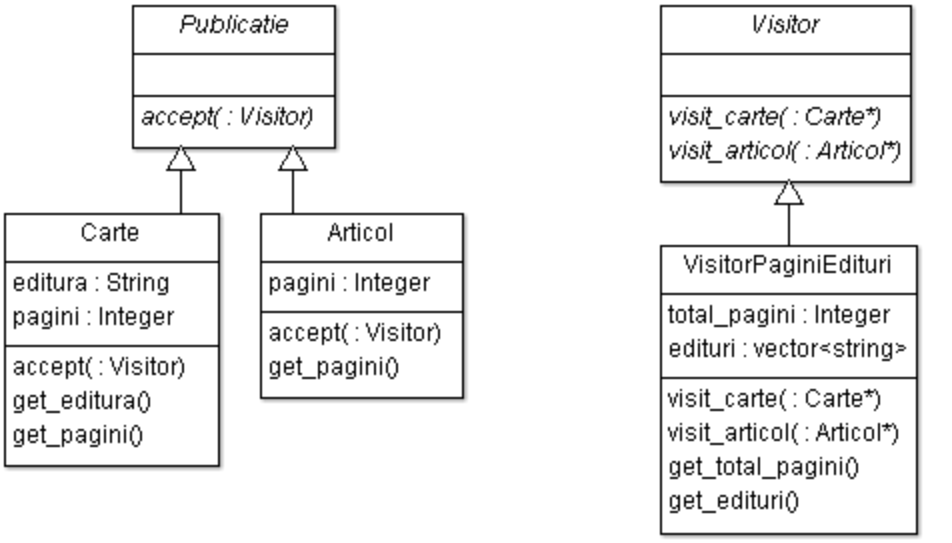
Scenariu (detalii clase si functia main):
1. Metoda play() din fiecare clasa din ierarhia Instrument va afisa la consola:
tip_parinte_instrument : tip_instrument ;
si va fi apelata in obiectele Instrument conform sablonului Composite
2. In main se creeaza o instanta Orchestra care va avea ca si componente
o instanta Pian, doua instante Vioara si o instanta Violoncel
3. Se apeleaza play() pentru instanta Orchestra
4. Cele doua instante Vioara si instanta Violoncel sunt eliminate din instanta Orchestra
si sunt adaugate intr-o noua instanta AnsambluCoarde care e adaugata in Orchestra
(metoda add din AnsambluCoarde va verifica daca instrumentul e Vioara sau Violoncel)
5. Se apeleaza din nou play() pentru instanta Orchestra

Punctaj	Operatie
4p	Codul care include signatura claselor si relatiile dintre ele compileaza fara eroare
2p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”
1 p	Descrierea corecta a metodelor din clasa Instrument
2p	Implementarea metodelor din clasele Pian, Vioara, Violoncel: - 0.5p constructori - 1p metodele play() - 0.5p metodele get_parent()
3p	Implementarea metodelor din clasa Orchestra: - 0.5p constructor si metoda get_parent() - 1p metodele add() si remove() - 1.5p metoda play()
3p	Implementarea metodelor din clasa AnsambluCoarde (punctaj similar clasa Orchestra)

Nume, Prenume:

Grupa:

3.



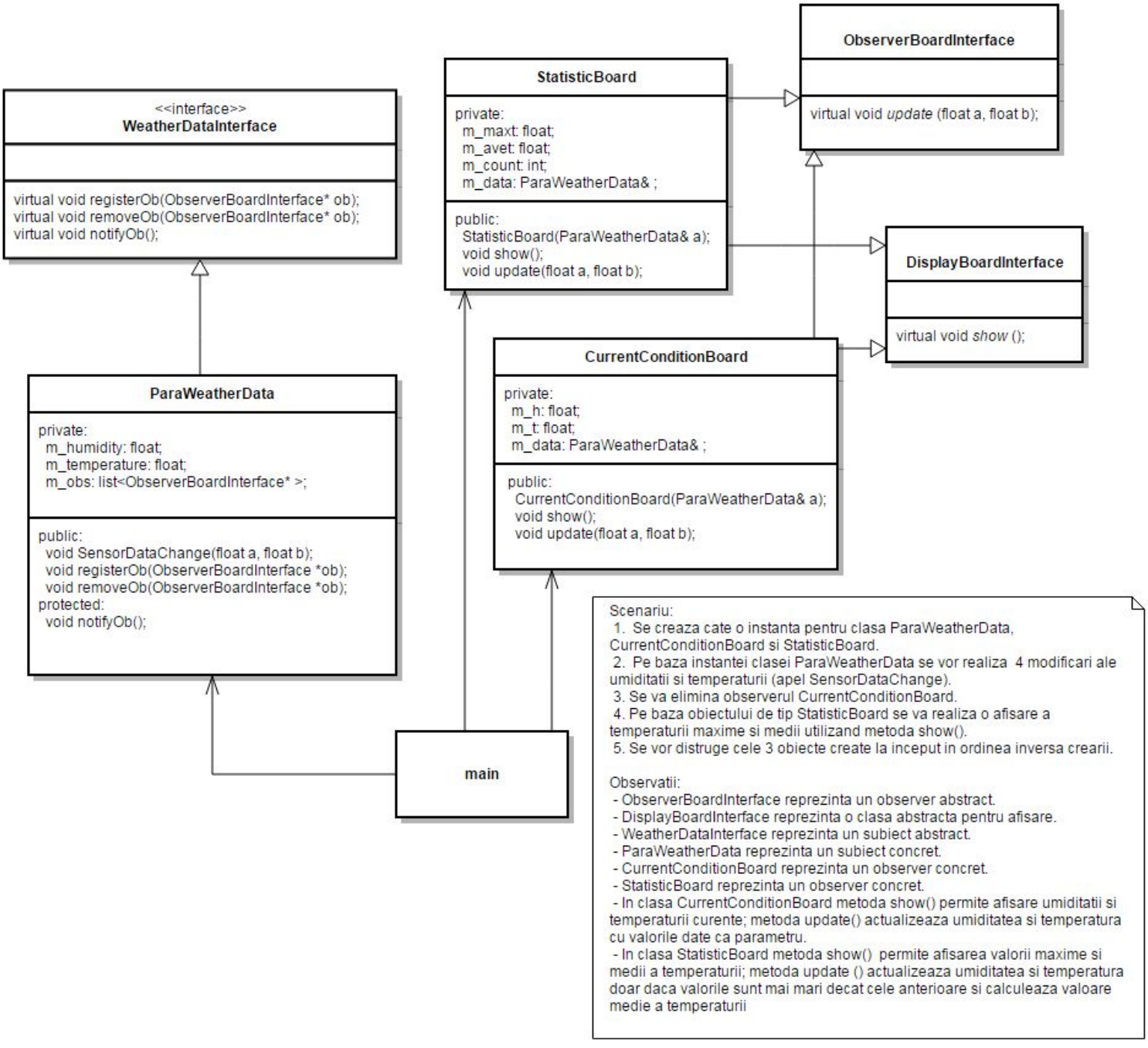
Scenariu (detalii clase si functia main):
1. Clasa VisitorPaginiEdituri are rolul de a numara totalul paginilor dintr-o serie de carti si articole si in plus in cazul cartilor de a retine si editurile; implementarile acestor functionalitati sunt oferite de metodele visit_ iar metoda accept din clasele publicatii vor apela metoda corespunzatoare
2. In main se va crea o colectie de publicatii in care se vor adauga doua instante Articol si doua instante Carte; pentru reprezentarea colectiei se va utiliza la alegere o clasa container din STL (vector, list, etc.)
3. Se va crea o instanta VisitorPaginiEdituri
4. Colectia de publicatii va fi iterata, pentru fiecare publicatie fiind apelata metoda Accept avand ca argument instanta VisitorPaginiEdituri
5. Se va afisa totalul paginilor si editurile obtinute de instanta VisitorPaginiEdituri

Punctaj	Operatie
4p	Codul care include signatura claselor si relatiile dintre ele compileaza fara eroare
2p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”
1p	Descrierea corecta a metodelor din clasa Publicatie
1.5p	Implementarea metodelor din clasa Carte: - 0.5p constructor - 0.5p metoda accept - 0.5p metodele get
1.5p	Implementarea metodelor din clasa Articol (punctaj similar clasa Carte)
1p	Descrierea corecta a metodelor din clasa Visitor
4p	Implementarea metodelor din clasa VisitorPaginiEdituri: - 1.5p metoda visit_carte - 1.5p metoda visit_articol - 0.5p constructor - 0.5p metodele get

Nume, Prenume:

Grupa:

4.

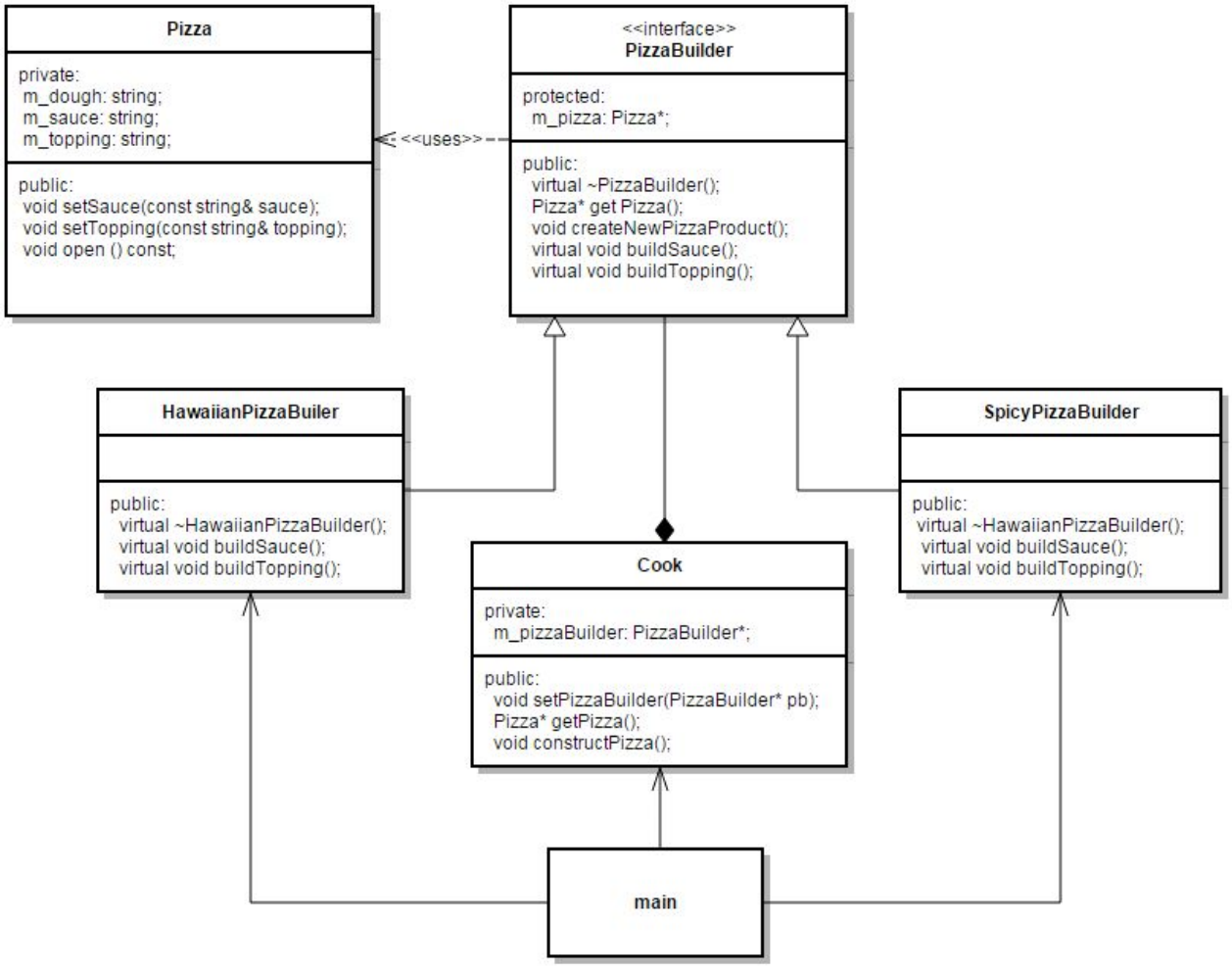


Punctaj	Operatie
4p	Codul care include signatura claselor relatiile dintre ele compileaza fara eroare.
2p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”.
3p	Implementarea clasei ParaWeatherData - 0.5p SensorDataChange() - 0.5p registerOb() - 0.5p removeOb() - 1.5p notifyOb()
1.5p	Implementarea clasei CurrentConditionBoard - 0.5p constructor - 0.5p show() - 0.5p update()
1.5p	Implementarea clasei StatisticBoard - 0.5p constructor - 0.5p show() - 0.5p update()
3p	Descrierea celor 3 clase abstracte ObserverBoardInterface, WeatherDataInterface, DisplayBoardInterface.

Nume, Prenume:

Grupa:

5.



Scenariu:
1. Se creaza o instanta a clasei Cook;
2. Se creaza cele 2 sortimente de pizza buildere (hawaiian si spicy) utilizand clasa PizzaBuilder.
3. Se apeleaza setPizzaBuilder() si constructPizza() pentru fiecare tip de pizza cu ajutorul instantei clasei Cook.
4. Se obtine cate un sortiment din fiecare tip de pizza prin apelul metodei getPizza() peste instanta clasei Cook.
5. Se apeleaza metoda open() pentru fiecare sortiment de pizza obtinut la punctul 4.
6. Se distrug cele 2 pizza buildere create la punctul 2. si cele 2 sortimente de pizza obtinute la punctul 4.

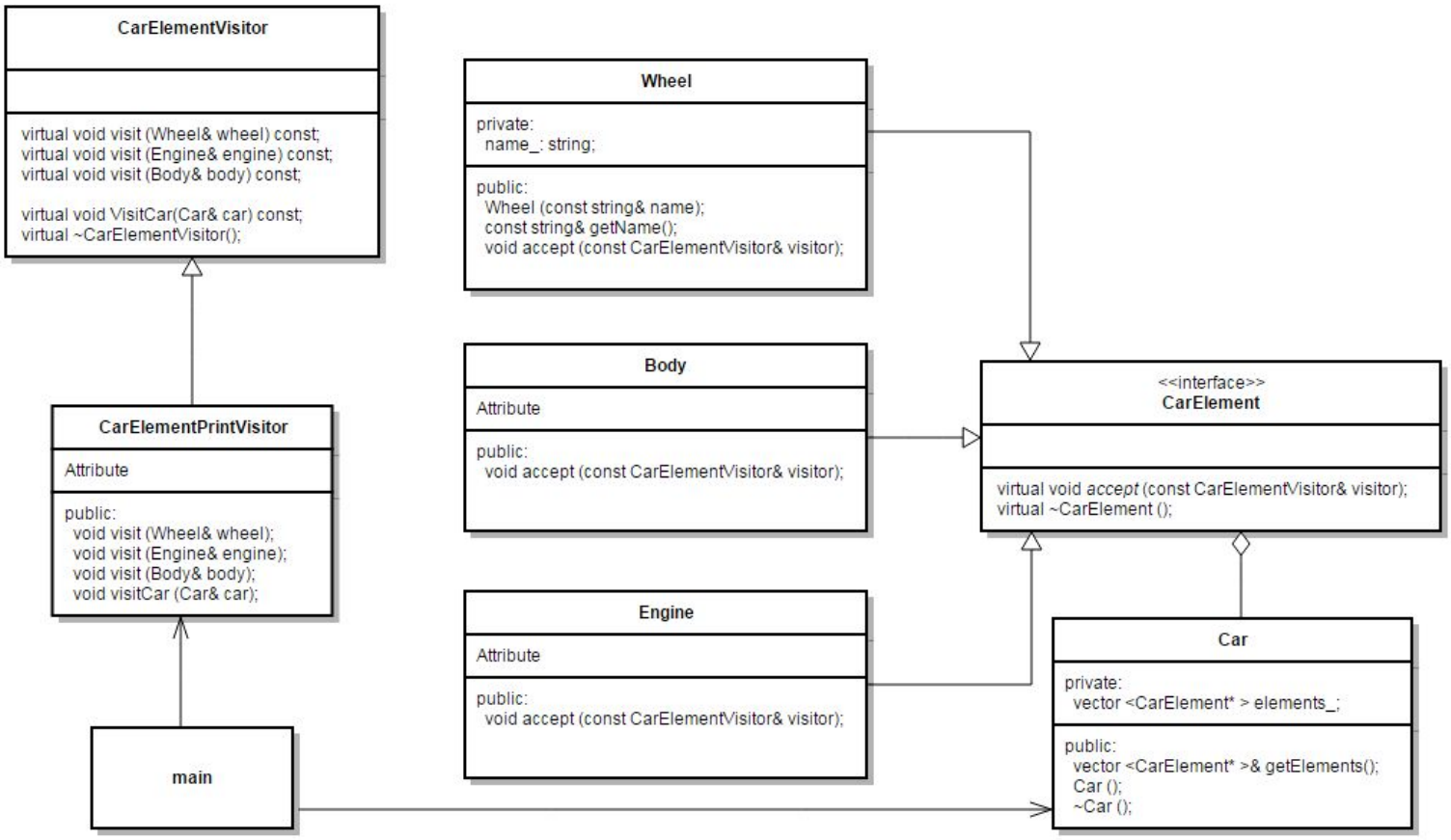
Observatii:
- PizzaBuilder este un Abstract Builder.
- Pizza reprezinta o clasa de tip Product.
- metoda open() din clasa Pizza trebuie sa afiseze la consola un mesaj particularizat fiecarui tip de pizza. De exemplu: "Pizza Hawaiian cu sosul x si toppingul y."
- Pentru fiecare tip de pizza (hawaiian/spicy) cu metoda buildSauce() si buildTopping() se va seta un anumit tip sos, respectiv topping.

Punctaj	Operatie
4p	Codul care include signatura claselor si relatiile dintre ele compileaza fara eroare.
2p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”.
2p	Implementarea clasei Pizza - 0.5p setSauce() - 0.5p setTopping() - 1p open()
2p	Implementarea clasei Cook - 0.5p setPizzaBuilder() - 0.5p getPizza() - 1p constructPizza()
2.25p	Implementarea clasei HawaiianPizzaBuilder - 0.25p destructor - 1p buildSauce() - 1p buildTopping()
2.25p	Implementarea clasei SpicyPizzaBuilder - 0.25p destructor - 1p buildSauce() - 1p buildTopping()
0.5p	Descrierea clasei abstracte PizzaBuilder.

Nume, Prenume:

Grupa:

6.



Scenariu:
1. Se creaza o instanta pentru clasa Car.
2. Se creaza un visitor pe baza clasei CarElementPrintVisitor.
3. Se apeleaza metoda visitCar() pentru visitor-ul creat la punctul 2.

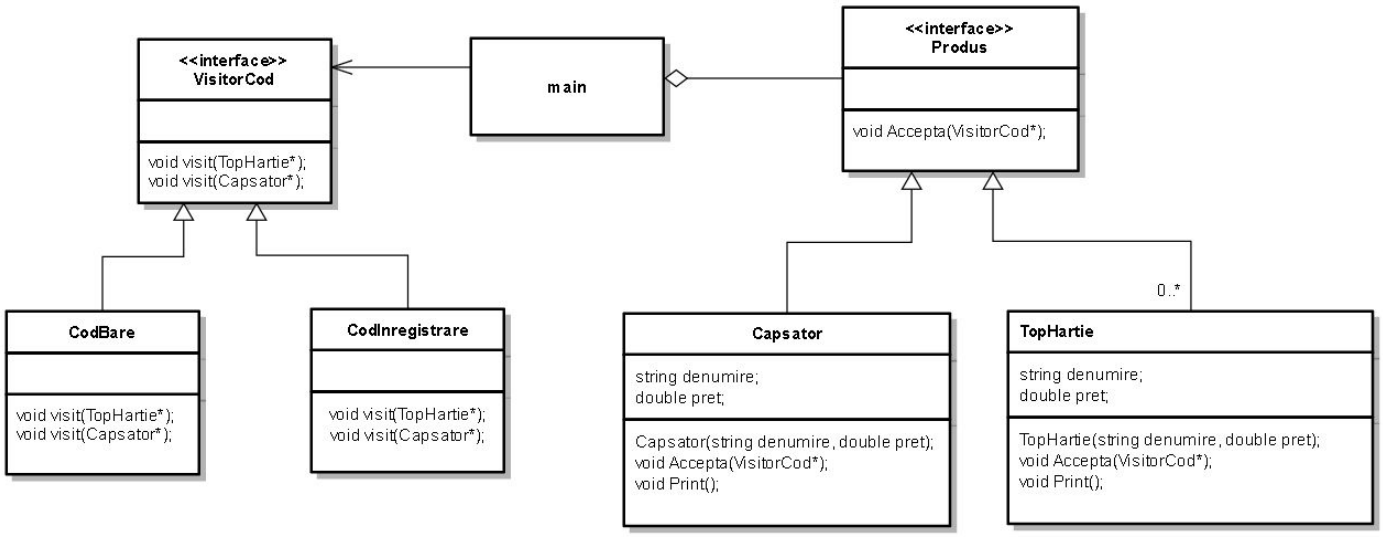
Observatii:
- clasa CarElement este o interfata pentru orice componenta a masinii.
- clasa CarElementVisitor este o clasa abstracta pentru orice tip de visitor care va fi definit.
- clasele Wheel, Engine si Body reprezinta cele 3 componente ale masinii care vor trebui vizitate.
- la nivelul clasei Car, la apelul constructorului Car() se va apela metoda push_back() pentru fiecare roata in parte (front left, front right, back left, back right) si apoi pentru body si engine. Aceste componente vor fi puse intr-un vector elements_.
- la nivelul clasei CarElementPrinVisitor fiecare metoda visit() va afisa la consola un mesaj de tipul: "Visiting car component X". Unde X este o componenta: wheel, body sau engine.
- In implementarea metodei visitcar() din clasa CarElementPrintVisitor:
  - aceasta metoda trebuie sa permita vizitarea componentelor unei masini.
  - la inceput se va afisa la consola mesajul: "Visiting car".
  - apoi se va utiliza un iterator pentru parcurgea vectorului de componente.
  - la sfarsit se va afisa la consola mesajul: "Visited car".

Punctaj	Operatie
4p	Codul care include signatura claselor si relatiile dintre ele compileaza fara eroare.
2p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”.
3p	Implementarea clasei CarElementPrintVisitor - 0.5p visit(Wheel) - 0.5p visit(Engine) - 0.5p visit(Body) - 1.5p visitCar()
2.5p	Implementarea clasei Wheel (3 x 0.5p fiecare metoda = 1.5p), clasei Body (0.5p metoda accept()), clasei Engine (0.5p metoda accept())
2.5p	Implementarea clasei Car - 0.5p getElements() - 1p constructor - 1p destructor
1p	Descrierea claselor abstracte CarElementVisitor si CarElement

Nume, Prenume:

Grupa:

7. Vizitator pentru codurile unui produs comercial



Scenariu:
1. Se creaza o instanta pointer CodBare
2. Se creaza o instanta pointer CodInregistrare
3. Se creaza un vector de pointeri Produs
4. Se creaza 4 produse Capsator si 4 produse TopHartie care se adauga in vectorul de la pasul 3 (produsele au caracteristici diferite)
5. intr-un for se apeleaza meotda Print() si imediat Accepta() de la obiectele Produs din vector (folosind iterator)

Observatii:
- clasele VisitorCod si Produs sunt clase abstracte
- clasa CodBare va implementa metodele visit() in care veti afisa un cod de bare format din 11 cifre (codul trebuie sa fie unic pentru instanta primita ca parametru)
Ex: 87301162091
- clasa CodInregistrare va implementa metodele visit() in care veti afisa un cod alfanumeric format din 6 caractere (codul trebuie sa fie unic pentru fiecare instanta primita ca parametru)
Ex: "A67RU2"
- clasa Capsator si TopHartie implementeaza metoda Print() care afiseaza caracteristicile cunoscute ale obiectelor, dar si metoda Accepta() cu rol de conectare a instantei curente la tipul specific de vizitator

.

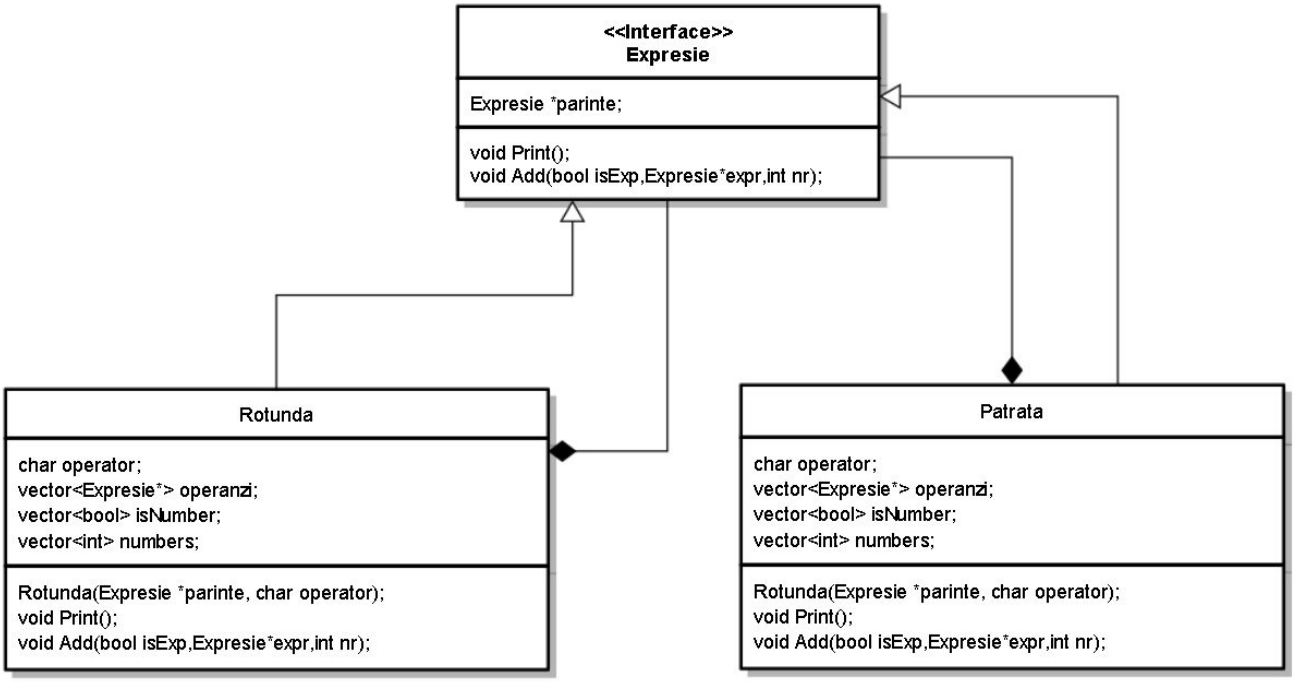
Punctaj	Operatie
4p	Codul care include signatura claselor si relatiile dintre ele compileaza fara eroare.
2p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”.
1p	Implementarea clasei CodBare - 0.5p visit(Capsator*) - 0.5p visit(TopHartie*)
1p	Implementarea clasei CodInregistrare - 0.5p visit(Capsator*) - 0.5p visit(TopHartie*)
3p	Implementarea clasei Capsator - 1p constructor - 1p Accepta() - 1p Print()
3p	Implementarea clasei TopHartie - 1p constructor - 1p Accepta() - 1p Print()
1p	Descrierea claselor abstracte VisitorCod si Produs

.

Nume, Prenume:

Grupa:

8. Compozitie expresii aritmetice



Scenariu:
1. Se creaza 2 expresii Patrata, de tip Expresie*
   - Ex1 cu parametrii (0,'+') si Ex2 cu parametrii (0,'/')
2. Se creaza 4 expresii Rotunda de tip Expresie*
   - Ex3 cu parametrii (Ex1,'*'), Ex4 cu parametrii (Ex3,'-')
   - Ex5 cu parametrii (Ex2,'-'), Ex6 cu parametrii (Ex2,'+')
3. Se adauga Ex4 la Ex3 si Ex3 la Ex1, Se adauga Ex5 si Ex6 la Ex2
5. Se adauga la Ex1 numarul 6, iar la Ex3 numarul 2
   Se adauga la Ex4 numerele 9 si 4
   Se adauga la Ex5 numerele 5 si 3
   Se adauga la Ex6 numerele 6 si 4
6. Se apeleaza consecutiv metoda Print() de la Ex1 si Ex2

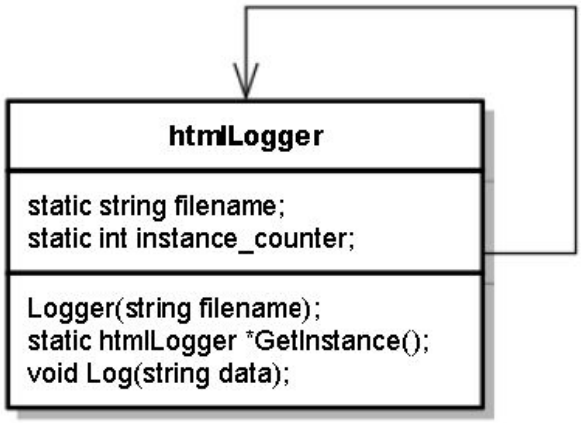
Observatii:
- Clasa abstracta Expresie defineste functiile de baza ale unei expresii
- Clasele Rotunda si Patrata vor implementa notiunile de expresie parantezata cu paranteze rotunde, (....), respectiv cu paranteze patrate [.....]
- operatorul primit in constructor va fi folosit intre toti termenii din instanta expresiei
- Expresiile care nu au parinte vor afisa in fata expresiei stringul "-->"
- cei 3 vectori definesc componentele expresiei
  - operanzi va defini gama operanzilor de tip expresie in ordinea afisarii lor de la stanga la dreapta
  - isNumber va decide daca operandul i este expresie sau numar intreg
  - numbers va contine elementele intregi din expresie
- toti cei 3 vectori au acelasi numar de elemente
- cand se adauga un operand expresie, atunci se adauga la operanzi, la numbers adaugam un -1,iar la isNumber un false
- cand se adauga un operand numar,atunci se adauga la numbers, la operanzi adaugam un NULL,iar la isNumber un true
- Print() va afisa elementele expresiei curente

Punctaj	Operatie
3p	Codul care include signatura claselor si relatiile dintre ele compileaza fara eroare.
3p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”.
4p	Implementarea clasei Rotunda - 1p constructor - 1p Add() - 2p Print()
4p	Implementarea clasei Patrata - 1p constructor - 1p Add() - 2p Print()
1p	Descrierea clasei abstracte Expresie

Nume, Prenume:

Grupa:

9. File logger (pentru X-uri)



Scenariu:
1. Se construieste o instanta [log*] la clasa singleton htmlLogger
2. Se construieste o functie pentru calculat n numere din sirul fibonacci
3. Se initializeaza fisierul html cu urmatorul sir de caractere folosind functia Log() -> scrie o noua linie in fisier
log->Log("<html><head><title>Fibonacci Logger</title></head>\n<body>\n");
3. Intr-o bucla iterativa, de 100 de ori
- se reapeleaza functia ce returneaza un pointer la instanta clasei
- se gasesc urmatoarele 10 numere din sir
- se construieste un string [fibstr] cu cele 10 noi numere generate si se scriu in fisier cu functia Log.
- stringul [fibstr] va fi decorat de subsirurile " " la stanga, respectiv " \n" la dreapta
4. la final se adauga in fisier mesajul "</body></html>"

Observatii:
- de fiecare data cand se apeleaza GetInstance() se incrementeaza variabila instance_counter
- se initiaizeaza in main, variabila instance_counter cu 0
- daca instance_counter depaseste 20, se arunca o exceptie
- exceptia va adauga la finalul fisierului (o singura data) ceea ce adauga punctul 4 si afiseaza in consola un mesaj de eroare
- orice executie ulterioara la metoda GetInstance() va returna un measj "Inaccesibil"
- metoda Log() are scopul de a scrie in fisier, in orice caz, nu va bloca fisierul pe perioada in care nu scrie (deschide, scrie, inchide)

Punctaj	Operatie
4p	Codul care include signatura clasei si structura interna
4p	Descrierea corecta a codului din functia main conform cerintelor din “Scenariu”.
1.5p	Implementarea constructorului
1.5p	Implementarea metodei GetInstance()
3p	Implementarea metodei Log()
1p	Implementarea functiei de calcul Fibonacci