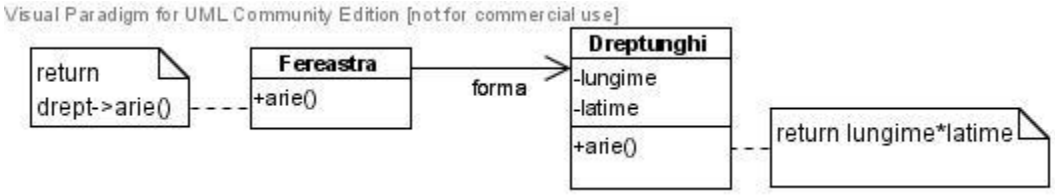
Universitatea "Al.I.Cuza" Iasi
Facultatea de Informatică

Nume:

An: I II III **Semian:** A B **Grupa:** 1 2 3 4 5 6

POO – Test 2
24.05.08

Observații:
1. *Nu este permisă consultarea bibliografiei.* ***2.*** *Toate întrebările sunt obligatorii.* ***3.*** *Dacă nu este precizat altfel, fiecare întrebare este notată cu 3 puncte. Repartiția punctelor la întrebările grilă este: 1 punct alegerea corectă a variantei, 2 puncte justificarea. Alegerea corectă se punctează numai dacă justificarea este total sau parțial corectă.* ***4.*** *Nu este permisă utilizarea de foi suplimentare.*



1) Să se explice diagrama de mai sus.

Diagrama include două clase, Fereastra şi Dreptunghi, împreună cu o relație de asociere unidirecțională între cele două clase. Feresatra cunoaste dreptunghiul asociat iar dreptunghiul nu cunoaşte la ce ferestre este asociat. Clasa Ferestra are o metodă arie(), iar clasa Dreptunghi are două atribute – lungime, lățime- şi o metodă arie(). Există şi două note care explică cum se calculează aria pentru ficare clasă.

2) Să se descrie implementarea diagramei din figura de mai sus în C++.

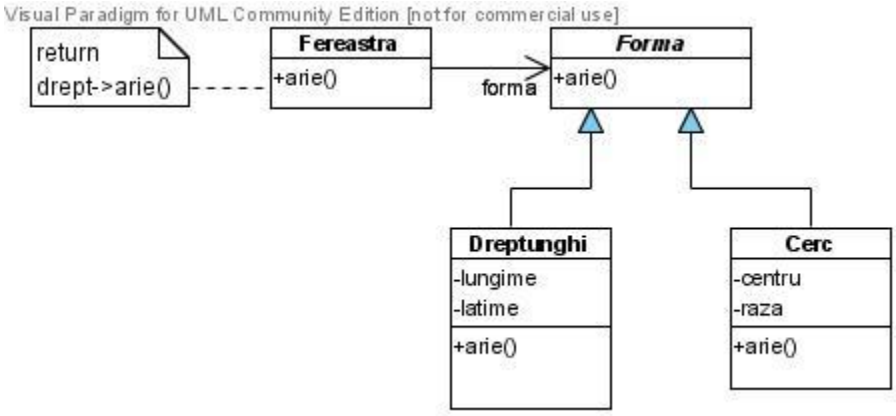
```
class Dreptunghi {
public:
  long arie() { return lungime*latime; }
 private:
  long lungime, latime;
};
```

```
class Fereastra {
public:
  long arie() { return drept->arie(); }
private:
  Dreptunghi *drept;
};
```

3) Cum se modfică diagrama dacă forma unei ferestre poate fi circulară sau dreptunghiulară. Explicați.

Se adaugă o nouă clasă abstract Forma cu metoda virtuala pura arie(), clasa Dreptunghi derivează din (de fapt implementează) clasa Forma, se mai defineşte o clasă Cerc care, la fel, derivează din Forma. Relația de asociere este acum între Fereastra şi Forma.

4) Să se deseneze noua diagramă.



5)

```
class A
{
public:
  virtual void f() {
    // request p
    ...
  }
};

class B : public A
{
public:
  void f() {
    // request p or q
  }
};
```

Ce principiu POO nu este satisfăcut de ierarhia alăturată? Explicați.
Răspuns.
Fiind vorba de o ierarhie in care metoda f() este suprascrisa, ne punem problema daca principiul substituirii are loc, adica dacaB est subclasa alui A.

Precondiția metodei A::f() este p

Precondiția metodei B::f() este p or q

Deoarece "p implică (p or q)" rezultă ca precondiția B::f() este mai slabă decât cea a lui A::f() (prima parte a contractului este îndeplinită).
Nu putem spune nimic despre respectarea principiului substituirii deoarece nu se spune nimic despre postcondiții.

6)

```
class A
{
public:
  void f() {
    // requires p
    ...
  }
};

class B : public A
{
  // invariant not p
public:
  ...
};
```

Ce este greşit în proiectarea ierarhiei alăturate? Cum poate fi corectat?
Răspuns.

Dacă b este un obiect al calsei B, atunci nu se poate apela b.A::f() deoarece invariantul clasei B şi precondiția metodei A::f() sunt contradictorii.

7) (STL)

```
class Punct { ... }

typededef list<Punct> LiniePolig1;

typededef set<Punct> LiniePolig2;

typededef vector<Punct> LiniePolig3;
```

Care dintre tipurile `LiniePolig1`, `LiniePolig2` şi `LiniePolig3` este mai potrivit pentru reprezentarea liniilor poligonale. Menționați avantajele şi dezavantajele.
Răspuns.
`LiniePolig1`
A: poate memora ordinea vârfurilor în sensul parcurgerii lor (trigonometric, arce ceasornic), poate fi circulară
D: nu există acces direct la vârfuri, pot exista vârfuri în duplicat
`LiniePolig2`
A: nu pot exista vârfuri în duplicat
D: nu există acces direct la vârfuri, Nu poate memora ordinea vârfurilor în sensul parcurgerii lor
`LiniePolig3`
A: poate memora ordinea vârfurilor în sensul parcurgerii lor (trigonometric, arce ceasornic), există acces direct la vârfuri
D: pot exista vârfuri în duplicat, nu poate fi circulară

8) (STL)

```
#include <map>
#include <string>
#include <algorithm>
#include <iostream>

using namespace std;

int main() {
  multimap<char, char> mp;
  multimap<char, char>::iterator i;
  char c;
  for (c = 'a'; c < 'f'; ++c) {
    mp.insert(make_pair(c, toupper(c)));
  }
  i = mp.find('c');
  ++i;
  if (i != mp.end())
    mp.erase(i);
  for(i=mp.begin(); i!= mp.end(); ++i)
    cout << i->first  << ", "
         << i->second << "; ";
  return 0;
}
```

Ce va afişa programul alăturat?

a) `a, A; b, B; d, D; e, E;`
b) `a, A; b, B; c, C; e, E;`
c) `a, A; b, B; c, C; d, D;`
d) nimic deoarece va da eroare (exceptie) la execuție.

Justificare.

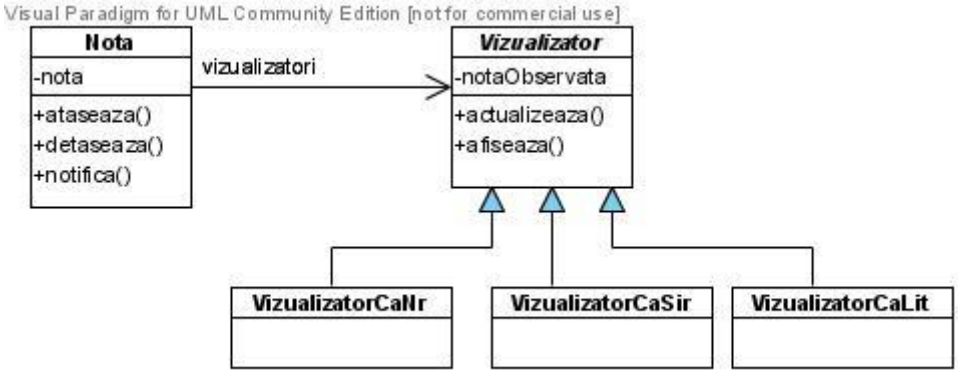
Varianta b). mp este un tablou asociativ cu chei multiple cu componente de forma (caracter, caracter).
Primul for creează tabloul asociativ ((‘a’, ‘A’), (‘b’, ‘B’), (‘c’, ‘C’), (‘d’, ‘D’), (‘e’, ‘E’)).
Dupa apelul lui find(), i va referi componenta ce memorează (‘c’, ‘C’); prin incrementare, va referi (‘d’, ‘D’).
Sterge aceasta componenta, astfel că ultimul for va afisa

```
a, A; b, B; c, C; e, E;
```

9) O notă poate fi vizualizată ca număr întreg în intervalul 1..10, ca şir de caractere (“unu”, “doi”,...), sau ca o literă (‘A’ pentru 10, ‘B’ pentru 9,...,’F’ pentru 5, ‘N’ pentru 1..4). Să se proiecteze diagrama claselor Nota, VizualizatorCaNr, VizualizatorCaSir, VizualizatorCaLit utilizând şablonul (patternul) de proiectare Observator (Observer).

10) Să se descrie în C++ clasele Nota şi VizualizatorCaNr.

Răspuns 9,10.




```
class Nota {
public:
  Nota(int pNota = 0);
  ~Nota();
  void ataseaza(Vizualizator* v) { vizualizatori.push_back(v); }
  void detaseaza(Vizualizator* v) { vizualizatori.erase(v); }
  void notifica();
  int getNota() { return nota; }
private:
  int nota;
  std::set<Vizualizator*> vizualizatori;
};
void Nota::notifica() {
  set<Vizualizator*>::Iterator i;
  for (i = vizualizatori.begin(); i != vizualizatori.end(); ++i) {
    i->actualizeaza(this);
  }
}
```

```
class VizualizatorcaNr {
public:
  VizualizatorcaNr(Nota* pNotaObs) : notaObservata(pNotaObs) {notaObservata.ataseaza(this); }
  ~VizualizatorcaNr() { }
  void actualizeaza(Nota* pNotaObs) { if (notaObservata == pNotaObs) afiseaza(); }
  void afiseaza() { cout << notaObservata.getNota; }
private:
  int notaObservata;
};
```