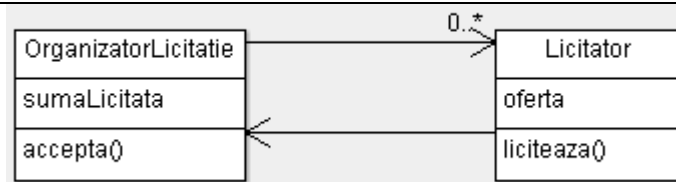


POO – Test scris

28.06.2010

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.



Figură 1

1) Să se explice termenul „polimorfism” în contextul programării orientate-obiect. Se va preciza modul de implementare în C++.

Răspuns.

1 p = stie ce inseamna polimorfism (aceeasi constructie sintactica cu mai multe intelesuri)

1 p = polimorfismul obtinut prin supraincarcare (operatori, metode)

1p = polimorfismul obtinut prin mostenire, suprascrierea metodelor si legarea tarzie

2) Să se descrie relațiile dintre clase din Figura 1. Să se precizeze cum sunt acestea implementate în C++.

Răspuns.

0.5 p = observa ca sunt relatii de asociere unidirectionale

0.5p = interpretarea multiplicatilor

cate 1p descrierea corecta in C++ a fiecarei relatii de asociere

3) Să se completeze clasele din Figura 1 cu atribute și metode pentru ca diagrama să devină instanță a șablonului Observer.

Răspuns.

1p = observa cine este subiectul si cine este observatorul

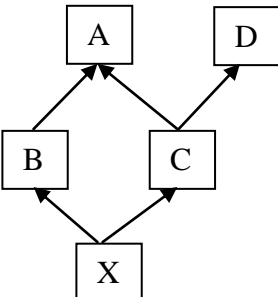
2p = metodele: attach, dettach, notify (la subiect), update (la observator) impreuna cu descriere pe scurt ce face fiecare (responsabilitatea) (0.5 fiecare metoda)

4) Să se descrie în C++ noile metode adăugate la clasa OrganizatorLicitatie la punctul 3.

Răspuns.

0.75 p fiecare metoda

<p>5)</p> <pre> class Test{ public: int x; Test():y(0), z(0){} Test(int a, int b, int c) :x(a),y(b),z(c){} int GetVal() const {return z;} protected: int y; int SetVal(int Val) {y = Val;} private: int z; int PutVal(int Val){z = Val;} }; </pre>	<p>Care din afirmațiile ce urmează, referitoare la codul alăturat, este falsă:</p> <p>a) Clasa Test are 2 constructori</p> <p>b) Programul principal poate accesa membrul x din Test</p> <p>c) Programul principal poate accesa membrul y din Test</p> <p>d) Clasele derivate din Test pot accesa membrii x si y</p> <p>e) Clasele derivate din Test nu pot accesa membrul z</p> <p>Justificare</p> <p>Explicatii:</p> <p>public: 1p</p> <p>protected: 0.5p</p> <p>private: 0.5p</p> <p>(sau de ce a,b,d,e,sunt adevarate)</p>
--	---

<p>6) Construiți diagrama pentru ierarhia claselor din programul de mai jos și precizați ce va afișa după execuție?</p> <pre> #include <iostream> using namespace std; class A{ public: A(){cout << "A ";} virtual ~A(){cout << "~A ";} }; class D{ public: D(){cout << "D ";} ~D(){cout << "~D ";} }; class B:virtual public A{ public: B(){cout << "B ";} ~B(){cout << "~B ";} }; class C:virtual public A, public D{ public: C(){cout << "C ";} ~C(){cout << "~C ";} }; class X: public B, public C { public: X(){cout << "X ";} ~X(){cout << "~X ";} }; void main(){ A* pA; pA = new X; delete pA; }; </pre>	<p>Răspuns.</p> <p>a) A B C D X ~X ~D ~C ~B ~A b) A B D C X ~X ~C ~D ~B ~A c) A B D C X ~D ~A d) A B D C X ~X ~D ~A</p>  <pre> graph BT X --> B X --> C B --> A C --> A C --> D </pre> <p>Diagrama : 0.5p Explicatia virtual public :0.5p Efectul pa = new X : 0.5p Efectul delete pA : 0.5p</p>
<p>7) Explicați codul de mai jos și precizați rezultatul execuției</p> <pre> #include<iostream> using namespace std; template <class T> class A { public: virtual void scrie() { cout<<" A "<<x;++x; } protected: T x; }; class B: public A<int> { public: B(){ x = 10; scrie();} void scrie() { cout<<" B "<<x;++x; } }; void main() { B *b = new B; b->A<int>::scrie(); ((A<int>*)b)->scrie(); } </pre>	<p>A clasă template: 0.5p scrie() metodă polimorfă: 0.5p B derivata unei specializări: 0.5p B *b = new B; produce B 10: 0.5p b->A<int>::scrie(); produce A 11: 0.5p ((A<int>*)b)->scrie(); produce B12: 0.5p</p> <p>Rezultat B 10 A 11 B 12</p>
<p>8) Explicați codul de mai jos și precizați rezultatul execuției</p> <pre> #include <iostream> #include <list> using namespace std; int main (){ </pre>	<p>Răspuns:</p> <p>lista1 – listă de int 0.5p rezultat for (1,2,3,4,5) 0.5p primul insert (10 după 1): 0.5p al doilea insert(2 de 20): 0.5p</p>

<pre> list<int> lista1; for (int i=1; i<=5; i++) lista1.push_back(i); list<int>::iterator it = lista1.begin(); ++it; lista1.insert(it,10); lista1.insert(it,2,20); list<int>::reverse_iterator rit; for (rit=lista1.rbegin() ; rit != lista1.rend(); ++rit) cout << " " << *rit; cout << endl; return 0; } </pre>	rit, rbegin, rend 0.5p rezultat(5,4,3,2,20,20,10,1) 0.5p
--	---

- 9) (6 puncte) Un arbore FooTree este definit recursiv astfel: a) un pătrat (să zicem de latură *lat*) este un arbore FooTree; b) o listă de arbori FooTree este un arbore FooTree. Aria unui arbore FooTree *t* este definită astfel: dacă *t* este pătrat, atunci este aria pătratului; dacă *t* este o secvență de arbori, atunci aria lui *t* este suma ariilor arborilor din secvență.
- Să se descrie în C++ clasele care descriu complet o structura FooTree.
 - Să se descrie în C++ o funcție care are la intrare un arbore FooTree *t* și calculează aria lui *t* (se presupune că implementarea de la a) nu include o metodă care întoarce aria).
 - Să se descrie o secvență de cod care construiește arborele $((p_2, p_3), p_4, p_5)$, unde p_i este patratul de latură i .

Răspuns .

1 punct diagrama corecta de clase (asta depinde de solutia aleasa)

1 punct pentru solutia corecta prin care face deosebirea dintre arbore elementar (patrat) si arbore compus

1.5 puncte descrierea in C++ a claselor

1.5 puncte = functia care intoarce aria (punctul b)

1p = secventa care construiește arborele de la c)

1p bonus pentru cine face legatura cu sablonul Composite (totusi asta ar presupune ca aria sa fie metoda).