POO – C++ - Laborator 12

Cuprins

1. Proiectul meniu consolă

În cadrul acestui laborator este prezentat un program care realizează un meniu arborescent la consolă, oarecum similar cu meniul principal al aplicaţiilor windows. Partea centrală a programului este o bibliotecă de clase pe baza căreia poate fi creat un meniu propriu. Biblioteca demonstrează utilizarea noţiunilor de POO învăţate până în prezent. La sfârşit, sunt propuse câteva exerciţii, care presupun realizarea unor meniuri proprii.

În continuare vom crea un proiect pe baza codului din arhiva atasata.

Să executăm programul obţinut şi să studiem funcţionarea lui.

2. Biblioteca de clase “meniu.h”

Header-ul “*meniu.h*” declară o ierarhie de 3 clase. Clasa de bază este `ElementMeniu`, care reprezintă orice element de meniu – atât meniul principal, un submeniu sau o operaţie finală. Toate elementele meniu concrete sunt clase derivate direct sau indirect din `ElementMeniu`.

Din `ElementMeniu` sunt derivate direct clasele `Meniu` şi `Operatie`. `Meniu` reprezintă meniul principal sau un submeniu. Iar `Operatie` – o operaţie finală.

3. Clasa ElementMeniu

```
...
```

```
class ElementMeniu {
private:
	char *nume;
	ElementMeniu *parinte;

protected:
	ElementMeniu(char *nume);
	virtual ~ElementMeniu();
	void afisareIncompletaTitlu();
	void afisareTitlu();

public:
	char *getNume();
	virtual void executa() = 0;
	friend class Meniu;
};
…
```

Clasa conţine membrii comuni pentru toate elementele.

Câmpurile clasei:

- `nume` – numele elementului, afişat atunci când se ajunge la acel element.
- `parinte` – elementul părinte. `null` dacă este meniul principal.

Constructorul permite crearea unui `ElementMeniu` pe baza numelui.

Câteva metode:

- `afisareIncompletaTitlu()` – afişeaza titlul tuturor părinţilor, de la rădăcina până la elementul curent, separate prin săgeată "`->`":

```
void ElementMeniu::afisareIncompletaTitlu() {
	if (parinte != NULL) {
		parinte->afisareIncompletaTitlu();
		cout << " -> " << nume;
	} else {
		cout << nume;
	}
}
```

- `afisareTitlu()` – afişează titlul aşa cum apare pe prima linie de ecran, când ne aflăm în elementul curent. Implementarea este bazată pe `afisareIncompletaTitlu()` :

```
void ElementMeniu::afisareTitlu() {
	afisareIncompletaTitlu();
	cout << ":" << endl << endl;
}
```

Aceste 2 funcţii au modul acces `protected` pentru că sunt apelate din funcţia `executa()` a claselor derivate. Apelarea lor din exteriorul ierarhiei nu este necesară.

- `executa()` – este o metodă virtuală pură. În clasele derivate conţine logica ce este executată atunci când utilizatorul selectează meniul curent.

4. Clasa Operatie

```
class Operatie : public ElementMeniu {
protected:
	Operatie(char *nume);

	/*operatia specifica acestui element
	virtual void execOperatie() = 0;  */

public:
	/*intreaga logica a elementului - afisare
titlu + operatie  */
	void executa();
};
```

Constructorul clasei preia argumentul nume şi îl transmite constructorului clasei de bază:

```
Operatie::Operatie(char *nume)
: ElementMeniu(nume) {}
```

Funcţia `executa()` realizează partea comună tuturor operaţiilor – şterge ecranul, afişează poziţia curentă în meniu, care apare pe prima line a ecranului, şi apelează `execOperatie()` pentru a-i da posibilitatea clasei derivate să-şi realizeze logica:

```
void Operatie::executa() {
	clrscr();
	this->afisareTitlu();
	this->execOperatie();
}
```

- `execOperatie()` – funcţia virtuală pură va conţine în clasele derivate logica specifică operaţiei.

Rostul de a avea 2 funcţii – `executa()` şi `execOperatie()` în loc să avem doar `executa()` a fost de a **reutiliza codul**. Logica comună tuturor operaţiilor este plasată în funcţia `executa()`, iar partea diferită – în funcţia `execOperatie()` a claselor derivate.

5. Clasa Meniu

```
class Meniu : public ElementMeniu {
private:
	static const int nrMaxElemente = 9;
	int nrElemente;
	ElementMeniu **elemente;

	void afisare();
	int citireComanda();

public:
	Meniu(char *nume);

	//va dealoca fii sai, eventual recursiv.
	~Meniu();
```

```
	void adaugaElement(ElementMeniu *element);
	void executa();
};
```

Reamintim, clasa `Meniu` reprezintă un meniu principal sau un submeniu. Clasa conţine o listă de referinţe către alte elemente de tip `ElementMeniu`.

Câmpuri:

- `nrMaxElemente` – câmp constant, numărul maxim de elemente a meniului. Întrucât spre elemente se navigheaza cu tastele 1-9, numărul lor maxim este 9.
- `nrElemente` – numărul de elemente
- `elemente` – vector de pointeri către elementele meniului. Deoarece tipul pointerilor este `ElementMeniu`, elementele pot fi de orice tip derivat din `ElementMeniu`, atât submeniuri cât şi operaţii.

Consturctor:

```
Meniu::Meniu(char *nume) : ElementMeniu(nume) {
	this->elemente = new ElementMeniu*[nrMaxElemente];
	this->nrElemente = 0;
}
```

Parametrul `nume` este transmis constructorului clasei de bază. În corpul constructorului se iniţializează câmpurile clasei.

Destructor:

```
Meniu::~Meniu() {
	for (int i=0; i<nrElemente; i++) {
		delete elemente[i];
	}
	delete[] elemente;
}
```

Până în prezent în clasele pe care le-am studiat, în destructor s-au dealocat doar acele zone de memorie care au fost alocate în cadrul clasei, de obicei în constructor. Însă în clasa `Meniu` ne este mai simplu să dealocăm şi elementele meniu referite, chiar dacă acestea au fost alocate în altă parte. Aceasta pentru că elementele meniu nu au nici o utilitate în afara meniului principal, şi trebuie dealocate odată cu el.

Destructorul clasei `Meniu` îşi va dealoca elementele, eventual recursiv, şi memoria dinamică alocată.

Unele funcţii:

- `citireComanda()` – execută o buclă infinită în care se citeşte o tastă. Dacă tasta este validă se iese din funcţie şi se returnează codul corespunzător acţiunii selectate, în caz contrar bucla se repetă.

```
/*returneaza indicele elementului activat, sau -1 pentru
iesire*/
int Meniu::citireComanda() {
	while (1) {
		char ch;
		cout << "Introduceti comanda:";
		ch = _getch();
```

```
            cout << endl;
            if (ch > '0' &&
             (ch - '0') <= this->nrElemente) {
                  //element meniu
                  int comanda = ch - '1';
              /*pt '1' va fi elementul 0*/

                  return comanda;
            } else if (ch == '0' || ch == 0x1B) {
                  /*0 sau ESC - iesire din meniu
                  return -1;*/
            } else {
                  //tasta invalida
                  cout << "Tasta invalida: " << ch
            << " Tastele valide sunt '0' - '"
           << nrElemente <<"'" << endl << endl;
            }
      }
}
```

- `adaugaElement(ElementMeniu *element)` – adaugă un element. În element, părintele este setat să fie obiectul curent – `this`.

```
void Meniu::adaugaElement(ElementMeniu *element) {
      nrElemente++;
      elemente[nrElemente - 1] = element;
      element->parinte = this;
}
```

Pentru a putea executa instrucţiunea:

```
element->parinte = this;
```

a fost nevoie de a declara clasa `Meniu` prietenă a clasei `ElementMeniu`. Vedeţi ultima linie a declaraţiei clasei `ElementMeniu` în *meniu.h*. A fost necesar pentru a putea accesa câmpul `element->parinte` din clasa `Meniu`.

- `executa()` – conţine o buclă infinită în care se afişează meniul, se citeşte o comandă şi se execută elementul selectat. Din funcţie se iese atunci când comanda citită este -1 – ieşirea din meniu.

```
void Meniu::executa() {
      for (;;) {
            int comanda;

            clrscr();
            afisareTitlu();
            afisare();
            comanda = citireComanda();
            if (comanda >=0 &&
                        comanda < nrElemente) {
                  elemente[comanda]->executa();
            } else {
                  //probabil iesirea - -1
                  return;
            }
      }
```

```
}
```

6. Clasele operaţii

O operaţie din meniu este o clasă derivată din `Operatie`. Pentru a le separa de clasele din biblioteca, operaţiile au fost definite în fişiere separate – *operatiiSimple.h* şi *operatiiSimple.cpp*. De exemplu, clasa `OperatieAdunare`:

operatiiSimple.h

```
…
class OperatieAdunare : public Operatie {
public:
	OperatieAdunare(char *nume);
	void execOperatie();
};
…
```

operatiiSimple.cpp

```
…
OperatieAdunare::OperatieAdunare(char *nume)
: Operatie(nume) {}

void OperatieAdunare::execOperatie() {
	int a, b;
	cout << "Introduceti 2 numere:";
	cin >> a >> b;
	cout << "suma = " << a + b << endl;
	pauza();
}
…
```

Tot ce trebuie să facem într-o operaţie simplă este să definim constructorul, şi să implementăm funcţia virtuală `execOperatie()`. La sfârşitul lui `execOperatie()` s-a apelat `pauza()` pentru a da posibilitatea utilizatorului să vadă rezultatul, înainte să se întoarcă în meniu.

7. Funcţia main()

În `main()` este construit arborele meniului, apoi este executat meniul principal.

meniuMain.cpp

```
#include<iostream>
#include "globale.h"
#include "meniu.h"
#include "operatiiSimple.h"
using namespace std;

int main() {
```

```
	Meniu *meniu =
		new Meniu("Meniu Principal");

	Meniu *submeniuCalculator =
		new Meniu("Calculator");
	meniu->adaugaElement(submeniuCalculator);
	submeniuCalculator->adaugaElement(
	    new OperatieAdunare("'+'"));
	submeniuCalculator->adaugaElement(
	    new OperatieScadere("'-'"));

	meniu->adaugaElement(
	    new Meniu("Meniu vid"));
	meniu->adaugaElement(
	    new ElementDespre("Despre program"));

	meniu->executa();

	/* intreg arborele de elemente va fi dealocat
recursiv*/
	delete meniu;

	cout << endl << endl
	    << "Sfarsit." << endl;
	pauza();
	return 0;
}
```

Pe prima linie este instanţiat meniul principal:

```
Meniu *meniu = new Meniu("Meniu Principal");
```

În continuare la meniu sunt adăugate elementele:

```
Meniu *submeniuCalculator = new Meniu("Calculator");
meniu->adaugaElement(submeniuCalculator);
submeniuCalculator->adaugaElement(
          new OperatieAdunare("'+'"));
submeniuCalculator->adaugaElement(
          new OperatieScadere("'-'"));

meniu->adaugaElement(new Meniu("Meniu vid"));
meniu->adaugaElement(
          new ElementDespre("Despre program"));
```

Ca să adăugăm un element trebuie să-l instanţiem şi să-l adăugăm la părintele său, apelând funcţia `adaugaElement()` din părinte:

```
submeniuCalculator->adaugaElement(new
OperatieAdunare("'+'"));
```

Pornirea meniului principal se face apelând:

```
meniu->executa();
```

Atât timp cât programul se va afla în meniu, ne vom afla în această funcţie.

La sfârşit, meniul principal este dealocat, iar destructorul său dealocă recursiv tot arborele de elemente:

```
delete meniu;
```

8. Clasa Lista

Mai jos este prezentată clasa `Lista`, care reprezintă o listă de numere. Clasa urmează să fie utilizată într-un meniu. Suportă operaţia de adăugare a unui număr, prin intermediul operatorului `+=`, ştergere a unui număr după index, şi afişarea listei. Fiecare număr are un index, începand de la 0, la fel ca un vector. Codul listei este deja adăugat la proiectul meniuri.

lista.h

```
#ifndef _lista_
#define _lista_

/*reprezinta o lista de numere. Fiecare numar are un
index, incepand de la 0, la fel ca intr-un vector.*/
class Lista {
private:
	int *elem;
	int n, dim;
public:
	Lista(int dim);
	~Lista();
	void operator+=(int num);
	void sterge(int index);

	//afiseaza numerele si indecsii lor
	void afisare();
};

#endif
```

lista.cpp

```
#include<iostream>
#include"lista.h"
using namespace std;

Lista::Lista(int dim) {
	this->dim = dim;
	elem = new int[dim];
}

Lista::~Lista() {
	delete[] elem;
}

void Lista::operator+=(int num) {
	if (n == dim) {
		cout <<"Lista plina" <<endl;
	} else {
		elem[n] = num;
```

```
            n++;
      }
}

void Lista::sterge(int index) {
      for(int i=index; i<n-1; i++) {
            elem[i] = elem[i+1];
      }
      n--;
}

void Lista::afisare() {
      for(int i=0; i<n; i++) {
            cout << i << ". " << elem[i] << endl;
      }
}
```

9. Meniul listei

Ne propunem sa creăm un meniu care să gestioneze o listă de numere reprezentată de `Lista`. Avem nevoie de minim 2 operaţii – una care să adauge un numar la listă, şi alta care să afişeze lista. De data aceasta clasele operaţii sunt mai complexe. Ele trebuie să aibă acces la obiectul listă. Iar lista trebuie să fie aceeaşi pentru ambele operaţii. Singurul mod elegant în care putem realiza acest lucru este să avem în clasele operaţii un câmp de tip pointer la `Lista`. Şi ambele operaţii să refere prin intermediul pointerului aceeaşi listă.

Lista nu poate fi instanţiată în nici una din clasele operaţii. Vom instanţia lista cu un nivel mai sus – în funcţia `main()`. Şi o vom transmite operaţiilor prin intermediul unui parametru la constructor.

Mai jos este prezentat codul celor 2 operaţii:

lista.h

```
…
class OpAdaugaInLista : public Operatie {
private:
      Lista *lista;
public:
      OpAdaugaInLista(char *nume, Lista *lista);
      void execOperatie();
};

class OpAfisareLista : public Operatie {
private:
      Lista *lista;
public:
      OpAfisareLista(char *nume, Lista *lista);
      void execOperatie();
};
…
```

lista.cpp

```
…
OpAdaugaInLista::OpAdaugaInLista(char *nume, Lista
```

```
*lista)
: Operatie(nume) {
	this->lista = lista;
}

void OpAdaugaInLista::execOperatie() {
	int num;
	cout << "num=";
	cin >> num;
	(*lista)+=num;
	pauza();
}

OpAfisareLista::OpAfisareLista(char *nume, Lista *lista):
Operatie(nume) {
	this->lista = lista;
}

void OpAfisareLista::execOperatie() {
	lista->afisare();
	pauza();
}
```

Adăugaţi cele 2 clase în program. Completaţi funcţia `main()` cu codul necesar pentru a crea un submeniu al listei cu cele 2 operaţii. Observaţi că ambele clase operaţii conţin un câmp pointer la listă. Obiectul `Lista` trebuie creat şi dealocat în `main()`.

Rulaţi programul rezultat.

10. Exerciţii

1. Implementaţi o operaţie pentru ştergerea unui element din listă după index.

2. Adăugaţi la listă operatorul de indexare – `[]`. Operatorul va servi pentru accesarea unui element după index. Implementaţi în meniu operaţia de afişare a unui element după index.

3. Implementaţi un meniu pentru 2 liste. Redefiniţi în clasa listă operatorul `+=`, astfel încât să accepte un operand dreapta de tip `Lista&`. Instrucţiunea `list1+=list2` va adăuga toate elementele din `list2` în `list1`. Implementaţi o operaţie meniu care să testeze acest operator.

   - `afisare()` – afişare mulţime.

4. Implementaţi un meniu care să testeze clasa `Multime` de la exerciţiul 1 din lab. 3.