RO: Accesul la membrii unei clase
EN: The access to a class members

Obiective:

- Înţelegerea teoretică şi practică a specificatorilor de vizibilitate, a modalităților de acces la mebri privați sau protejați ai unei clase, a accesului prin intermediul obiectelor (instanțelor), pointerilor sau adreselor.

Objectives:

- Theoretical and practical understanding of the access specifiers, of private and protected members' access, of using the objects (instances), pointers and addresses for accessing the class's members.

Rezumat:

Utilizarea așa numitor *specificatori de vizibilitate*: *public, private, protected* servește la controlul accesului la variabilele şi metodele membre ale unei clase.

În cazul în care nu este menţionat nici un specificator de acces, compilatorul consideră implicit toate variabilele şi metodele din clasă ca fiind *private.*

Semnificaţia acestor specificatori în C++ este:

public: variabilele şi metodele care urmează sunt vizibile (accesibile) din orice zonă a programului;
private: variabilele şi metodele care urmează pot fi accesate doar de către membrii clasei;
protected: variabilele şi metodele care urmează pot fi accesate în cadrul clasei curente sau în clasele derivate din ea.

De obicei se preferă ca variabilele dintr-o anumită clasă să fie *private*, pentru un mai bun control al acestora (ele pot fi modificate doar prin intermediul unor metode din clasă) şi deci pentru o protecţie a lor.

Metodele şi variabilele publice ne dau modul de interfaţare al clasei și împreună cu cele private şi protected ne dau modul de implementare a clasei.

Metodele publice care au ca scop modificarea unor variabile *private* sau *protected* poartă numele de *metode mutator/setter*.

Metodele publice care au ca scop returnarea unor variabile *private* sau *protected* poartă numele de *metode accesor/getter*.

Compilatoarele moderne genereaza la cerere metode *setter* si *getter* pentru fiecare din atributele *private* ale unei clase in mod individual, avand ca si nume *set/getNume_atribut*, prima litera din numele atributului fiind *uppercase*.

Exemple:

1.

```
/*ilustrarea accesului la atributele private si publice ale unei clase,
folosind metode mutator si accesor pt. membrii privati
*/
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

class Test1 {
        int x;
public:
        char sir[20];
        Test1() {
                x = 0;
                strcpy(sir, "Text implicit.");
                }
```

```
	//metoda care modifica variabila sir din clasa- sir public!!! – e accesibil fara setter
	void setSir(char * new_sir) {
		strcpy(sir, new_sir);
	}
	//metoda care modifica variabila x din clasa - private
	void setX(int a) { x = a;
	}
	//metoda care returneaza valoarea variabilei private x
	int getX() {return x;
	}
	char* getSir() {// e accesibil fara getter
		return sir;
	}
};

int main() {
	Test1 ob1; //instantiere
	int a;
	char s[20];
	cout << "\nIntroduceti valoarea variabilei de tip \"int\" din clasa: ";
	cin >> a;
	ob1.setX(a);
	cout << "\nIntroduceti valoarea variabilei de tip \"char array\" din clasa: ";
	cin >> s;
	//apelul metodei publice
	ob1.setSir(s);
	//afisarea valorilor atributelor clasei
	//accesarea membrului privat prin intermediul metodei accesor
	cout << "\nValoarea de tip \"int\" stocata in clasa si returnata de metoda accesor este: " <<
ob1.getX() << endl;
	//accesarea directa a membrului public
	cout << "\nValoarea de tip \"char array\" stocata in clasa si accesata direct este: " <<
ob1.sir << endl;
	//accesarea cu getter a membrului public
	cout << "\nValoarea de tip \"char array\" stocata in clasa si accesata cu getter este: " <<
ob1.getSir() << endl;
}
```

Cerinta:

Considerati atributul sir de tip private. Modificati programul pentru o functionalitatea corecta.

==========

2.

```
/*ilustrarea modului implicit de alocare a specificatorilor de vizibilitate*/

#include <iostream>
using namespace std;

class Test2{
	int x;

	Test2( ){
		cout<<"\nApel constructor explicit vid.";
	}
};

int main( ){
	//Test2 ob1; //instantiere imposibila, constructorul este privat!
	int a;
	cout<<"\nIntroduceti valoarea variabilei de tip \"int\" din clasa: ";
```

```
        cin>>a;
        //incercare (imposibila) de initializare a variabilei din clasa
        //ob1.x = a;
}

================================
```

3.

```
/*ilustrarea alternarii specificatorilor de vizibilitate; implementarea metodelor in cadrul clasei
sau in exteriorul acesteia; apelul membrilor privati de catre membrii publici vizibili din exterior;
imposibilitatea accesarii directe din exterior a membrilor privati din clasa
*/
#include <iostream>
using namespace std;

const int Max1=10;
const int Max2=10;

class Matrix{
        //atribute
        int matrix[Max1][Max2], dim1, dim2;
        //declararea metodei de afisare a unui element
        int returnElement(int row, int column);
public:
//constructor explicit cu parametri – recomandat a folosi o metoda diferita a citi valorile
        Matrix(int dim1, int dim2){
                //variabile locale
                int i, j;
                this->dim1=dim1;
                this->dim2=dim2;
                cout<<"\nIntroduceti elementele matricii: ";
                for(i=0; i<dim1; i++){
                        for(j=0; j<dim2; j++){
                                cout<<"\nmatrix["<<i<<"]["<<j<<"] = ";
                                cin>>matrix[i][j];
                        }
                }
        }

        //metoda de afisare a matricii din clasa; implementare in cadrul clasei
        void displayMatrix(){
                //variabile locale
                int i, j;
                cout<<"\nElementele matricii: ";
                for(i=0; i<dim1; i++){
                        cout<<"\n";
                        for(j=0; j<dim2; j++){
                        //apelul metodei private care returneaza valoarea unui element din matrice
                                cout<<returnElement(i, j)<<" ";
                        }
                }
                cout<<endl;
        }
        //declararea metodei de afisare a elementelor unei coloane
        void displayColumn(int col);
};

//implementarea externa a metodelor (publice sau private) declarate in clasa

void Matrix::displayColumn(int col){
        if(col<0||col >=dim2){
```

```
		cout<<"\nColoana cu numarul "<<col<<" nu exista in matricea din clasa!\n";
	}
	else{
		cout<<"\nElementele coloanei "<<col<<": ";
		for(int i=0; i<dim1; i++){
			cout<<returnElement(i, col)<<" ";
		}
	}
}

int Matrix::returnElement(int row, int column){
	return matrix[row][column];
}

int main( ){
	int dim1,dim2;
		cout<<"\nIntroduceti dimensiunile 1 si 2 ale matricii: (<=10):\n";
		cin>>dim1;
		cin>>dim2;
	Matrix m1(dim1,dim2); //instantiere cu citire valori
	m1.displayMatrix();
	int c;
	cout<<"\nIntroduceti un numar de coloana ale carei elemente vor fi afisate: (<dim2) ";
	cin>>c;
	m1.displayColumn(c);
	//incercare (imposibila) de a accesa direct un membru privat al clasei
	//m1.returnElement(0, 0);
	}
/*
Cerinta:
- introduceti o metoda publica readMatrix( ) pentru a citi elementele matricii, astfel incat constructorul sa aloce spatiul folosind dimensiunile reale ale matricii iar valoriile matricii sa fie citite cu aceasta metoda in main( ) folosind obiectul Matrix instantiat. E recomandat ca matricea sa fie declarata ca atribut printr-un pointer si constructorul sa aloce spatiu pentru ea folosind parametrii preluati in main( ) pentru dimensiunile reale ale matricii
*/
================================================
```

4.

```
/*obiecte, adrese de obiecte, pointeri si referinte la obiecte;
modalitatile specifice de accesare a membrilor clasei
*/

#include <iostream>
using namespace std;

class Test4{
	//toti membri clasei sunt publici
public:
	float f;

	Test4( ){
		cout<<"\nApel constructor explicit vid.";
		//initializarea variabilei din clasa cu o valoare implicita
		f = 0;
	}

	float ret_f(){//f nu e private, nu e necesara metoda
		return f;
	}
};
```

```
int main(  ){
//---------------------------------------------
        Test4 ob1; //instantiere (creare de obiect)
        float a;

        //afisarea atributului f din clasa, prin acces direct
        cout<<"\nValoarea atributului de tip \"float\" din clasa: "<<ob1.f;

        cout<<"\nIntroduceti o valoare de tip \"float\": ";
        cin>>a;

        //modificarea variabilei f din clasa in mod direct
        ob1.f = a;

        //afisarea atributului f din clasa, prin acces direct
        cout<<"\nValoarea variabilei de tip \"float\" din clasa: "<<ob1.f;

//---------------------------------------------
        //declararea unui pointer la un obiect din clasa
        Test4 *Pob;
        //definirea pointerului
        Pob = new Test4;

        cout<<"\ nIntroduceti o valoare de tip \"float\": ";
        cin>>a;

        //modificarea atributului din clasa folosind un pointer
        Pob->f = a;

        //afisarea atributului f din clasa, prin intermediul metodei de tip get
        cout<<"\nValoarea atributului f  de tip \"float\" din clasa: "<<Pob->ret_f();

//---------------------------------------------
        //declararea si initializarea unei referinte la un obiect din clasa
        Test4 &Rob1 = ob1;

        //afisarea atributului din clasa, prin intermediul metodei get folosind referinta
        cout<<"\nValoarea atributului f de tip \"float\" din clasa: "<<Rob1.ret_f();
}
```

Cerinta:
Considerati atributul f de tip private, definiti metode setter si getter individuale cu nume adecvat. Verificati functionalitatea lor.

==========

5.

```
//modelare numere complexe
#include <iostream>
using namespace std;

class Complex {
        // membri private
        int re;
        int im;
        double modul;//nu e adecvat
        double faza;//nu e adecvat
public:
```

```
	// membri publici
	void setRe(int r) {
		re = r;
		}
	void setIm(int i) {
		im = i;
		}

	void det_modul_faza( ) {//recomandat tratare independenta atribute
		modul = sqrt((double)re*re + (double)im*im);
		faza = atan2((double)re, (double)im);//radiani
	}
	int getRe(void) {
		return re;
	}

	int getIm(void) {
		return im;
	}

	double getModul(void) {
		return modul;
	}

	double getFaza(void) {
		return faza;
	}
};

int main() {
	Complex c1;
	c1.setRe(12);
	c1.setIm(5);
	c1.det_modul_faza( );

	cout << "Date despre obiect: " << endl;
	cout << "\tRe = " << c1.getRe() << ", Im=" << c1.getIm() << endl;
	cout << "\tModul: " << c1.getModul() << endl;
cout << "\tFaza radiani: " << c1.getFaza( ) << "\tFaza grade:" << c1.getFaza( ) * 180 / 3.14 <<
endl;
	}
```

6.

```
//gestiune CNP pana in anul 2000
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
#include <time.h>

class Person {
	// membri private
	char nume[16];
	char prenume[24];
	char cnp[14];
	// structura CNP: S AN LU ZI 6cifre; total 13 cifre
public:
	// membri publici
	void setNume(char *n);
	void setPrenume(char *p);
	int setCnp(char *c);//ret. valori functie tip CNP
```

```
	char* getNume(void) {
		return nume;
	}

	char* getPrenume(void) {
		return prenume;
	}

	char* getCnp(void) {
		return cnp;
	}

	char get_gender(void);
	int get_an_nast(void);
	int get_luna_nast(void);
	int get_zi_nast(void);
	int get_varsta(void);
};

void Person::setNume(char *n) {
	if (n != 0)
		strncpy(nume, n, 15);
	else
		strcpy(nume, "Necunoscut");
}

void Person::setPrenume(char *p) {
	if (p != 0)
		strncpy(prenume, p, 23);
	else
		strcpy(prenume, "Necunoscut");
}

int Person::setCnp(char *c) {
	char buf[3];
	int n;
	if (c != 0) {
		// validare CNP: numai pentru cifra sex, cifrele pentru AN, LU, ZI
		cnp[13] = '\0';
		if (strlen(c) != 13)
			return 1;
		if (c[0] != '1' && c[0] != '2')
			return 2;
		strncpy(buf, c + 1, 2);
		buf[2] = '\0';
		n = atoi(buf);
		if (n > 99)
			return 3;//inconsistent

		strncpy(buf, c + 3, 2);
		buf[2] = '\0';
		n = atoi(buf);
		if (n == 0 || n > 12)
			return 4;
		strncpy(buf, c + 5, 2);
		buf[2] = '\0';
		n = atoi(buf);
		if (n == 0 || n > 31)
			return 5;
```

```
                strcpy(cnp, c);
                return 0;
        }
}

char Person::get_gender(void) {
        if (cnp[0] == '1') return 'M';
        if (cnp[0] == '2') return 'F';
        return 'X';
}

int Person::get_an_nast(void) {
        char buf[3];
        strncpy(buf, cnp + 1, 2);
        buf[2] = '\0';
        return(1900 + atoi(buf));
}

int Person::get_luna_nast(void) {
        char buf[3];
        strncpy(buf, cnp + 3, 2);
        buf[2] = '\0';
        return(atoi(buf));
}

int Person::get_zi_nast(void) {
        return((cnp[5] - '0') * 10 + (cnp[6] - '0'));
}

int Person::get_varsta(void) {
        struct tm *newTime;
        time_t szClock;
        time(&szClock);
        newTime = localtime(&szClock);
        int an_c = 1900 + newTime->tm_year;
        int an_n = get_an_nast();
        int n = an_c - an_n;
        int lu_c = newTime->tm_mon + 1;
        int lu_n = get_luna_nast();
        if (lu_c < lu_n)    n--;
        else {
                if (lu_c == lu_n) {
                        int zi_c = newTime->tm_mday;
                        int zi_n = get_zi_nast();
                        if (zi_c < zi_n)
                                n--;
                                }
                }
        return n;
        }

int main( ) {
        Person p1;
        char aux_string[30];
        cout << "\nEnter Name: ";
        cin >> aux_string;//Popescu
        p1.setNume(aux_string);
        cout << "\nEnter SurName: ";
        cin >> aux_string;//Bitanescu
        p1.setPrenume(aux_string);
```

```
	cout << "\nEnter CNP: ";
	cin >> aux_string;//1890403120671
	p1.setCnp(aux_string);

	cout << "Date despre obiect: " << endl;
	cout << "\tNume: " << p1.getNume() << ", Prenume: " << p1.getPrenume() <<
		", CNP: " << p1.getCnp() << endl;
	cout << "\tSex: " << p1.get_gender() << endl;
	cout << "\tData nasterii: " <<
	p1.get_an_nast() << "/" << p1.get_luna_nast() << "/" << p1.get_zi_nast() << endl;
	cout << "\tVarsta: " << p1.get_varsta() << endl;
	}
```

```
//varianta functii de biblioteca VC++
//Gestiune CNP cu validare CNP
#include <iostream>
using namespace std;
#include <time.h>
const int dim_sir = 24;

class Person {
	// membri private
	char nume[dim_sir];
	char prenume[dim_sir];
	char cnp[14];
	// structura CNP: S AN LU ZI 6cifre; total 13 cifre
public:
	// membri publici
	void setNume(char *n);
	void setPrenume(char *p);
	int setValidCnp(char *c);

	char* getNume(void) {
		return nume;
	}

	char* getPrenume(void) {
		return prenume;
	}

	char* getCnp(void) {
		return cnp;
	}

	char get_gender(void);
	int get_an_nast(void);
	int get_luna_nast(void);
	int get_zi_nast(void);
	int get_varsta(void);
};

void Person::setNume(char *n) {
	if (n != 0)
		strncpy_s(nume, n, dim_sir-1);
	else
		strcpy_s(nume, "Necunoscut");
}

void Person::setPrenume(char *p) {
```

```
	if (p != 0)
		strncpy_s(prenume, p, dim_sir-1);
	else
		strcpy_s(prenume, "Necunoscut");
}

int Person::setValidCnp(char *c) {
	char buf[3];
	int n;
	if (c != 0) {
		// validare CNP: numai pentru cifra gen, cifrele pentru AN, LU, ZI
		if (strlen(c) != 13)//lungime cnp
			return 1;
		if (c[0] != '1' && c[0] != '2')//cod cnp
			return 2;
		strncpy_s(buf, c + 1, 2);//an
		buf[2] = '\0';
		n = atoi(buf);
		if (n > 99)
			return 3;//inconsistent
		strncpy_s(buf, c + 3, 2);//luna
		buf[2] = '\0';
		n = atoi(buf);
		if (n == 0 || n > 12)
			return 4;
		strncpy_s(buf, c + 5, 2);//zi
		buf[2] = '\0';
		n = atoi(buf);
		if (n == 0 || n > 31)
			return 5;
		strcpy_s(cnp, c);//copiere sir c valid in cnp
		return 0;
	}
	else return -1;
}

char Person::get_gender(void) {
	if (cnp[0] == '1') return 'M';
	if (cnp[0] == '2') return 'F';
	return 'X';
}

int Person::get_an_nast(void) {
	char buf[3];
	strncpy_s(buf, cnp + 1, 2);
	buf[2] = '\0';
	return(1900 + atoi(buf));
}

int Person::get_luna_nast(void) {
	char buf[3];
	strncpy_s(buf, cnp + 3, 2);
	buf[2] = '\0';
	return(atoi(buf));
}

int Person::get_zi_nast(void) {
	return((cnp[5] - '0') * 10 + (cnp[6] - '0'));
}
```

```
int Person::get_varsta(void) {
	struct tm newTime;
	time_t szClock;

	time(&szClock);
	localtime_s(&newTime, &szClock);

	int an_c = 1900 + newTime.tm_year;
	int an_n = get_an_nast();
	int n = an_c - an_n;
	int lu_c = newTime.tm_mon + 1;
	int lu_n = get_luna_nast();
	if (lu_c < lu_n)
		n--;
	else {
		if (lu_c == lu_n) {
			int zi_c = newTime.tm_mday;
			int zi_n = get_zi_nast();
			if (zi_c < zi_n)	n--;
			}
	}
	return n;
}

int main( ) {
	Person p1;
	char aux_string[dim_sir];
	cout << "\nEnter Name: ";
	cin >> aux_string;//Popescu
	p1.setNume(aux_string);
	cout << "\nEnter SurName: ";
	//cin >> aux_string;//Bitanescu
	cin.ignore();
	gets_s(aux_string, dim_sir);//Preia Prenume si cu spatiu
	p1.setPrenume(aux_string);
	cout << "\nEnter CNP: ";
	cin >> aux_string;//1890403120671
	int t_cnp=p1.setValidCnp(aux_string);
	switch (t_cnp) {
	case 0: cout << "\nCNP valid\n";
		cout << "\nDate despre obiect: " << endl;
		cout << "\tNume: " << p1.getNume() << ", Prenume: " << p1.getPrenume() << ",
CNP: " << p1.getCnp() << endl;
		cout << "\tGen: " << p1.get_gender() << endl;
		cout << "\tData nasterii: " <<p1.get_an_nast() << "/" << p1.get_luna_nast() <<
"/" << p1.get_zi_nast() << endl;
		cout << "\tVarsta: " << p1.get_varsta() << endl; break;
	case 1: cout << "\nLungime sir CNP invalid"; break;
	case 2: cout << "\nGen invalid"; break;
	case 3: cout << "\nAn invalid"; break;
	case 4: cout << "\nLuna invalida"; break;
	case 5: cout << "\nZi invalida"; break;
	default: cout << "\nProbleme CNP"; break;
	}

	return 0;
}
```

Probleme propuse:

1. Să se scrie o aplicaţie C++ care implementează o clasă numită *PilotF1*. Clasa definește variabilele private *nume* (șir de caractere), *echipa* (șir de caractere), *varsta* (int), *record* (int), *nr_pole_position* (int). Ca membri publici, clasa conține metode accesor/getter și mutator/setter distincte pentru fiecare din atributele clasei.
   In funcția *main( )*, să se creeze 3 instanțe distincte ale clasei *PilotF1* și să se folosească metodele mutator/setter pentru a inițializa datele din fiecare obiect cu informația corespunzătoare citită de la tastatură. Folosind metodele accesor/getter, să se afișeze toate datele pilotului cu cel mai bun record.
2. Să se modifice exemplul 2 astfel încât codul să poată fi lansat în execuție considerand atributul clasei *private*.
3. Pornind de la exemplul care tratează lucrul cu matrice in varianta transformata cu alocare dinamica, completați codul scris cu metodele specifice pentru:
   - afișarea elementelor de pe diagonala secundara a matricei, dacă matricea este pătratică; în caz contrar se afișează un mesaj corespunzător;
   - afișarea elementelor de sub diagonala principala;
   - afișarea unei matrice de dimensiunea celei inițiale ale cărei elemente pot avea valori de 0 (dacă elementul corespunzător este mai mare decât o valoare citita) sau 1 (în caz contrar);
4. Să se scrie o clasă care are ca variabilă privată un câmp de tip dată, definit într-o structură externă clasei (*zi – int, luna – int, an - int*). Clasa conține metode mutator/setter și accesor/getter (publice) pentru informația privată. In clasă se mai află doua metode publice care:
   - testează validitatea datei stocate;
   - scrie într-un fișier toate datele din anul curent care preced (cronologic) data stocată în clasă;

   In functia *main( )*, după instanțierea clasei și citirea de la tastatură a componentelor unei date, să se apeleze metodele membre și apoi să se verifice rezultatele obținute.
5. Folosiți accesul prin pointeri și prin referință in rezolvarea problemelor 1...4.
6. Să se scrie o aplicaţie C++ care implementează o clasă numită *Triunghi*. Clasa cuprinde atributele private pentru laturile *a, b, c,* un constructor cu parametrii, metode setter si getter adecvate. Calculați aria și perimetrul prin metode specifice clasei. Scrieți o metodă care să indice dacă triunghiul este dreptunghic sau nu. Definiti o metoda private cu parametrii in clasa care permite verificarea condiției ca laturile să formeze un triunghi. Ea va fi folosita si de metodele setter.
7. Să se scrie clasa *Seif*, cu atributele private *cifru* și *suma.* Descrieți metodele private *getSuma()* și *setSuma( )* și metodele publice *puneInSeif( )* și *scoateDinSeif( )* cu care să accesați *suma* de bani care se află în seif. Metoda *puneInSeif( )* poate apela *getSuma()* și *setSuma( ),* metoda *scoateDinSeif( )* poate apela *getSuma( )* și *setSuma( ).* Instanțiați obiecte din clasa *Seif*, iar metodele *puneInSeif( )* și *scoateDinSeif( )* vor putea accesa suma doar dacă parametrul de tip cifru utilizat corespunde obiectului instanțiat. În caz de diferență de cifru, se va da un mesaj.
8. Dezvoltați aplicația prezentată în exemplul 6 prin:
   - utilizarea valorilor returnate de metoda *setValidCnp( )* pentru a valida suplimentar (luna si ziua) CNP-ul in *main( )*
   - permiterea intoducerii de coduri CNP care încep cu alte cifre decât 1 și 2, cu analizarea semnificației noilor valori (5 – masculin, 6 - feminin).

Homework:

1. Write a C++ application that implements a class called *F1Pilot*. The class defines the private variables *name* (array of characters), *team* (array of characters), *age* (int), *best_time* (int) and pole_position_nr. As public members, the class contains mutator/setter and accessor/getter methods for each of the class's attributes.
   In function *main( )*, create 3 different instances of the *F1Pilot* class and use the mutator methods for initializing each object's data with the corresponding information read from the keyboard. Using the accessor methods, display all the data related to the pilot that has the best time.
2. Modify the 2-nd example so that the code could be launched into execution. The attribute of the class will be considered *private*.

3. Starting with the code that exemplifies the matrix operationsas modified version with dinamic memory allocation, add the specific methods for:
   - displaying the elements from the main diagonal of the matrix, in case the matrix is square; if not, display a significant message;
   - displaying the elements which are below the secondary diagonal;
   - displaying a matrix that has identical dimensions with the original matrix and its elements can have as possible values 0's (if the corresponding element is greater than a certain threshold value) or 1's (otherwise);
4. Write a C++ class that has as private variable a date field. The date is defined as a structure declared outside the class and it contains the fields *day – int, month – int, year – int*. The class contains public accessor/getter and mutator/setter methods that are capable of using the private information. The class also contains two public methods that:
   - test the validity of the stored date;
   - write into a file all the dates from the current year that precede chronologically the class stored date;

   In the *main( )* function, after instantiating the class and after reading from the keyboard all the components of a date, call the member methods and then verify the obtained results.
5. Use the pointers and the references for resolving the problems 1...4.
6. Write a C++ application that defines a class called *Triangle*. The class contains as private variables the triangle's sides a, b and c, a constructor with parameters and adequate setter and getter methods. The class will contain methods that will calculate the shape's area and perimeter. Write a distinct method that will print a specific message if the triangle is right. Develop a private method with parameters that will determine whether the values of a, b and c form a triangle. The method will be called in setter methods.
7. Write a class named *Safe* that has as private attributes the *cipher* and the amount of *money*. Implement the private methods *getMoney( )* and *setMoney( )*. The public methods *putInSafe( )* and *getFromSafe( )* will call the previous methods only if the *cipher* sent as parameter matches the value stored inside the class. Display a message if the cipher is not correct.
8. Develop the application from example 6 with new facilities that allow:
   - using the returned values of the method *setValidCnp( )* to suplementary validate the CNP (month and day) in *main( )*
   - introducing CNP codes that start with numbers different than 1 or 2 and analyze the significance of the new numbers (5 – male, 6 - female).