----------------------------------------------CLASA DE BAZA(abstracta)------------------------

class baza

{

protected:

public:

baza(){}

baza(const baza & ob){}

virtual ~baza(){}

deriv& operator=(const deriv & ob)

{

return \*this;

}

virtual void citesc(istream &os)=0;

virtual void afisez(ostream &os)=0;

friend ostream& operator <<(ostream &os, baza &ob);

friend istream& operator >>(istream &os, baza &ob);

};

ostream& operator << (ostream& os, baza& ob)

{

ob.afisez(os);

return os;

}

istream& operator >>(istream& os, baza& ob)

{

ob.citesc(os);

return os;

}

----------------------------------------------------CLASA DERIVATA-----------------------------------------

class deriv : public baza

{

private:

public:

deriv(){}

deriv(const deriv & ob){}

~deriv(){}

//gettere CONSTANTE

deriv& operator=(const deriv & ob)

{

return \*this;

}

void citesc(istream &os);//am acces la membrii privati ii citesc direct

void afisez(ostream &os)

{

baza::afisez();

}

};

---------------------CLASA TEMPLATE---------------------------------------------------

template< typename T>

class temp

{

-----//---------

//atentie la functiile friend

template<typename U>

friend ostream & operator <<(ostream &os, temp<U> &ob);

template<typename U>

friend istream & operator >>(istream &os, temp<U> &ob);

};

//atentie la declararea in AFARA clasei TEMPLATE:

template<typename T>

+prefix vizibilitate: temp<T>::

->atentie la static template=>apartine de tip

->atentie la MOSTENIREA CU DATE STATICE pentru fiecare derivata o data statica noua

------------------------------------------------------CLASA SPECIALIZATA----------------------

template<>

class special<tip>

{

------------//--------

// !numele constructorilor si tipurile obiectelor se fac cu special<tip>

//citire cu friend nu-i pun template la functie ci doar la clasa

}

template<>//la static e posibil sa nu vrea :)

--------------------------------MENU SINGLETON---------------------------

class singleton

{

static singleton \*instanta;

vector<tren\*>v;

singleton()

{

cout<<"Operatiuni: \n";

cout<<"1-> \n";

cout<<"2-> \n";

cout<<"3-> \n";

cout<<"4-> \n";

}

public:

static singleton \* getInstanta()

{

if(instanta==NULL)

instanta=new singleton;

return instanta;

}

void op1()

{

cout<<"Ati ales optiunea1 \n";

}

void op2()

{

cout<<"Ati ales optiunea2 \n";

}

void op3()

{

cout<<"Ati ales optiunea3 \n";

}

void op4()

{

cout<<"Ati ales optiunea4 \n";

}

};

singleton \* singleton :: instanta;

int main()

{

singleton \*s;

s=singleton::getInstanta();

while (true)

{

try {

int optiune;

cout << "Dati numarul optiunii: ";

cin >> optiune;

cin.get();

if (optiune <= 0 || optiune > 4)

throw

"Nu ai ales corect!! mai alege o data!! \n";

if(optiune==1)

{

s->op1();

}

else

if(optiune==2)

{

s->op2();

}

else

if(optiune==3)

{

s->op3();

}

else

if(optiune==4)

{

s->op4();

}

}

catch (const char \*s)

{

cout << s<<endl;

}

cout<<"doriti sa continuati? (da/nu): ";

string rasp;

//cin.get();

getline(cin,rasp);

if(rasp=="nu")

break;

}

return 0;

}

--------------------------------DIAMANT------------------------------------------------------------

-> daca NU pun virtual la mostenirea de la burta zice ca

nu stie ce baza sa ia la UPCASTING -> aici e o problema

->daca nu faceam upcasting nu era nicio problema

A \*a=new D;//upcasting

D \*p=dynamic\_cast<D\*>(a);//accesare functie nervituala din clasa derivata

p->print();//doar in clasa derivata derivate

------------------------------STRING+CHAR--------------------------------------

->citire : getline(cin, string);

dupa citirea unui intreg/double/caracter, cin.get()/ os.get();

->! CITIREA CARACTERELOR:

Char x[100];

for(i=0;i<4;i++)

cin>>x[i];

cin.get();

string s;

getline(cin,s);

->se poate itera prin el

->se poate folosi string.size();

------------------------------------------------TYPEINFO----------------------------------

#include<typeinfo>

cec a;

cout<<typeid(a).name();

if(typeid(a).name()==typeid(class cec).name())

cout<<"da";

else

cout<<"nu";

--------------------------CITIRE N OBIECTE--------------------------------------------------------

int n,i;

cout<<"dati n= ";

cin>>n;

for(i=0;i<n;i++)

{

tip \* ob=new tip;

cin>> \*ob;

v.push\_back(ob);

}

-----------------------STL--------------------------------------------------------------------------------------

ATENTIE!

->cu STL nu pot lucra fara cc si = !!!

->se poate lucra si nedinamic cu STL

->SI PE STL se poate volosi metoda .size();

vector<A>v;

A a,b;

v.push\_back(a);

v.push\_back(b);

for(vector<A>:: iterator it=v.begin(); it!=v.end(); ++it)

cout<<\*it;

-----------------------STL:VECTOR:----------

#include<vector>

->dinamic : tip \*p=new tip;

->declarare: vector <tip\_baza\*> v;

->citire:cin>>\*p;

->adaugare: v.push\_back(p);

->iterare si afisare:

for (vector<plata\*>::iterator it = v.begin(); it != v.end(); it++)

cout << \*\*it;

----------------------STL: MAP---------------------------------

#include<map>

->initializare :map<tip cheie, tip val> mymap;

->inserare: mymap[cheie]=valoare;

->afisare:

for (//typename->template// map<int, U>::iterator it=gestiune<U>::mymap.begin(); it!=gestiune<U>::mymap.end(); ++it)

os << it->second<< endl;//it->first;

->cand vreau pentru o singura cheie mai multe valori

int \* v=new int[10];

map<int, int> mymap;

v[0]=2;

v[1]=3;

//mymap.insert ( pair<int, int>(1,2) );

//mymap.insert ( pair<int, int>(1,3) );

mymap[1]=v;

for ( map<int,int>::iterator it=mymap.begin(); it!=mymap.end(); ++it) {

cout << it->first << " => "<<it->second;

int \* x=new int [10];

x=it->second;

int n,i;

n=2;

for(i=0;i<n;i++)

cout<<x[i]<<" ";

----------------------------------------DYNAMIC\_CAST------------------------------------------

->lucrez numai cu POINTERI!!!

->CEL PUTIN o functie VIRTUALA in clasa de baza

->daca vreau sa verific tipul unui obiect

if(tip \*p=dynamic\_cast<tip\*>(ob))

for (vector<format\*>::iterator it = v.begin(); it != v.end(); it++)

if(articol \*p=dynamic\_cast<articol\*>(\*it))

cout<<p->getNumePub();

->nu stiu ce e dar e buna(convertire iterator)

articol \*x;

x=dynamic\_cast<articol\*>(\*it);

cout<<\*x;

-------------------EXEMPLU MENU---------------------------

void menu()

{

cout<<"Optiuni: "<<endl;

cout<<"1->plata numerar"<<endl;

cout<<"2->plata cec"<<endl;

cout<<"3->plata card de credit"<<endl;

//dupa afisez pentru fiecare gestiunea

}

-------------------CITIRE SI MEMORARE N OBIECTE-------

void citire (int &n, vector<tip\*>& v, //ATENTIE SI STL ARE NEVOIE DE ADRESA)

{

int i=0;

string op;

cout<<"Dati n= ";

cin>>n;

menu();

while(i<n)

{

int optiune;

cout << "Dati numarul optiunii: ";

cin >> optiune;

cin.get();

try {

if (optiune <= 0 || optiune >= 4)

throw "Nu ai ales corect!! mai alege o data \n ";

i++;

if(optiune==1)

{

tip \*p=new tip;

cin>>\*p;

v.push\_back(p);

}

...............

}//de la try

catch(const char \*s){

cout<<s<<endl;

}

}

}

----------------------------AFISARE CELE N OBIECTE -----------

void afis(vector <tip\_baza\*> v,//restul STL)

{

cout<<"-----------------------------"<<endl;

cout << "cele n plati efectuate sunt:" << endl;

for (vector<plata\*>::iterator it = v.begin(); it != v.end(); it++)

cout << \*\*it;

//si ce mai e de afisat

}

-----------------------------------------------------MENU INT MAIN------------------

while(true)

{

int optiune;

cout << "Dati numarul optiunii: ";

cin >> optiune;

cin.get();

try {

int optiune;

cout << "Dati numarul optiunii: ";

cin >> optiune;

cin.get();

if (optiune <= 0 || optiune > 4)

throw "Nu ai ales corect!! mai alege o data ";

if(optiune==1)

{

cout<<"Ai ales optiunea 1 \n";

}

if(optiune==2)

{

cout<<"Ai ales optiunea 2 \n";

}

if(optiune==3)

{

cout<<"Ai ales optiunea 3 \n";

}

if(optiune==4)

{

cout<<"Ai ales optiunea 4 \n";

}

}

catch(const char \*s)

{

cout<<s<<endl;

}

string rasp;

cout<<"doriti sa continuati?(da/nu):";

getline(cin, rasp);

if(rasp=="nu")

break;

}

-------------------MODEL DE DIAMANT------------------------------------------

class A

{

const int x=2;

public:

A()

{

cout<<"constructor A called \n";

}

virtual ~A()

{

cout<<"destructor A called \n";

}

virtual int get\_x()

{

cout<<"A::get\_x\n";

return x;

}

};

class B: virtual public A

{

const int y=3;

public:

B()

{

cout<<"constructor B called \n";

}

~B()

{

cout<<"destructor B called \n";

}

int get\_x()

{

cout<<"B::get\_x\n";

return y;

}

};

class C: virtual public A

{

const int z=4;

public:

C()

{

cout<<"constructor C called \n";

}

~C()

{

cout<<"destructor C called \n";

}

int get\_x()

{

cout<<"C::get\_x\n";

return z;

}

};

class D: public B, public C

{

const int d=5;

public:

D()

{

cout<<"constructor D called \n";

}

~D()

{

cout<<"destructor D called \n";

}

int get\_x()

{

cout<<"D::get\_x\n";

return d;

}

void print()

{

cout<<"nimic\n";

}

};

int main()

{

A \*a=new D;//upcasting

D \*p=dynamic\_cast<D\*>(a);

p->print();

}

---------------------------------------------