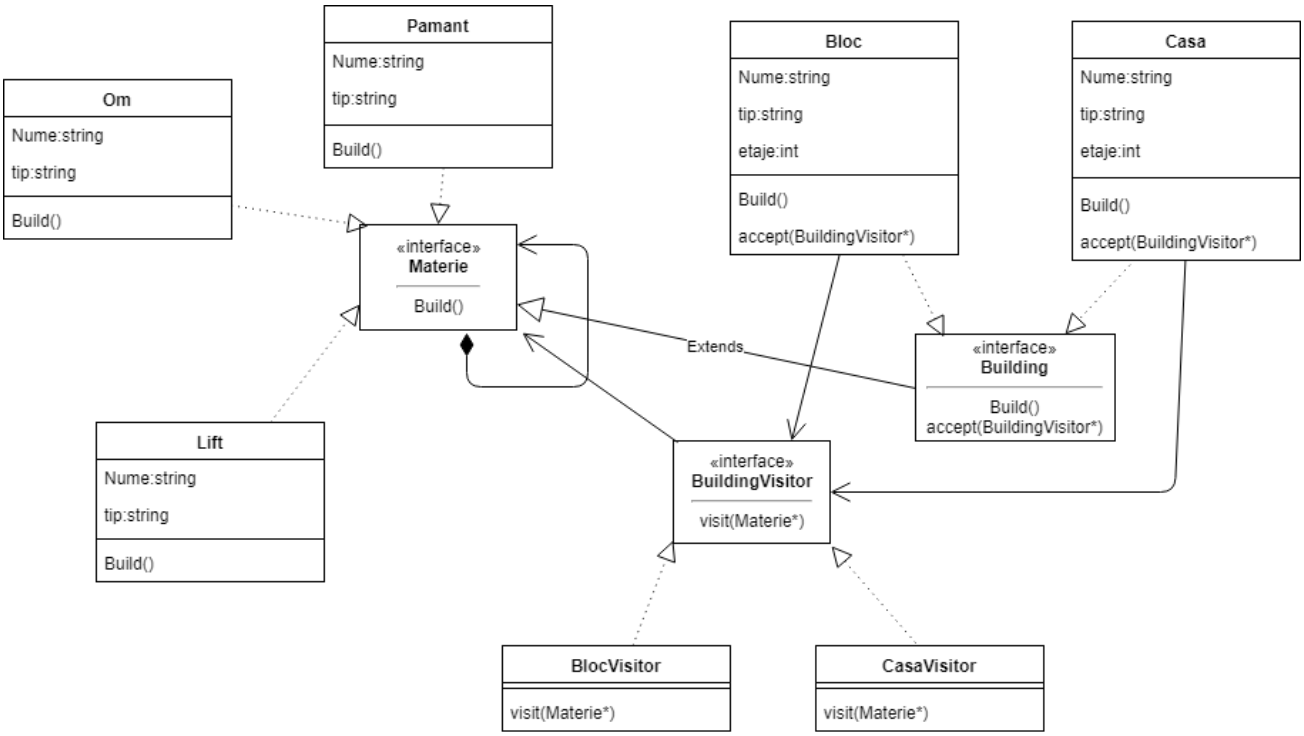
**Laborator POO – ultimu**

1. O problema echivalenta test 2
2. Trucuri POO (pentru examen)

**Problema:**




Sa se implementeze clasele aferente diagramei UML de mai sus, astfel incat codul din main() sa functioneze correct. La final trebuie sa aveti aproximativ urmatorul output:

```
PAMANT: Romania : tara
BLOC: Nr. 41, L5 : Ciment
Blocul este o cladire!
LIFT: Lift : metal
CASA: Nr. 4 : Chirpici
Casa este o cladire!
PERSOANA: Persoana1 : Femeie
PERSOANA: Persoana2 : Barbat
PERSOANA: Persoana3 : Femeie
PERSOANA: Persoana4 : Barbat
```

```
int main()
{
	Materie *teritoriu = new Pamant("Romania", "tara");
	Building *bloc = new Bloc("Nr. 41, L5", "Ciment", 10);
	Building *casa = new Casa("Nr. 4", "Chirpici", 1);
	bloc->Add(new Lift("Lift", "metal"));
	teritoriu->Add(bloc);
	teritoriu->Add(casa);
	srand(time(NULL));
	[casa]() mutable {
		for (auto i : { '1','2','3','4' })
			casa->Add(new Om(string("Persoana") + i, []() -> string {
				const char *items[3] = { "Femeie", "Barbat" };
				return items[rand() % 2];
			}()));
	}();
	bloc->accept(new BlocVisitor);
	casa->accept(new CasaVisitor);
	teritoriu->Build(0);
	return 0;
}
```

4p – Aranjarea codului pe fisiere header / cpp corespunzatoare

4p – Programul compileaz asi ruleaza correct si afiseaza ce trebuie

3p – Clasele Materie, Building, BuildingVisitor

10p – Clasele Lift, Om, Pamant, Bloc, Casa

2p – Clasele BlocVisitor si CasaVisitor

2p –orice for din clasele Materie va fi sub forma

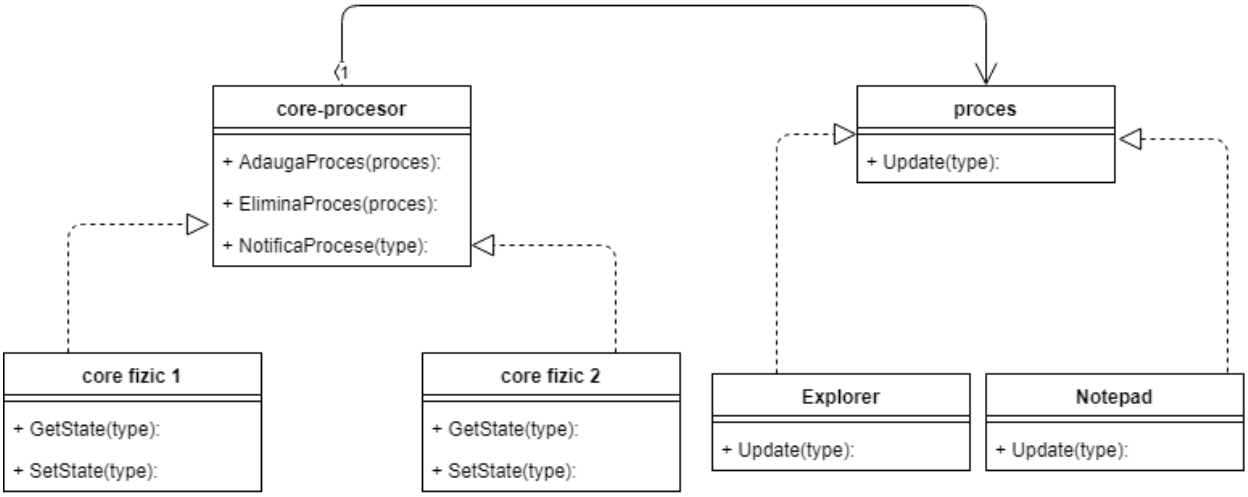
for (auto : this)

**2** Trucuri POO (pentru examen)

- Proiectare UML

  Intr-un spatiu agricol sunt lucratori care au grija de gradina si livada. Lucratorii pot fi oameni ingijitori sau sefi de echipa.  In gradina cresc castraveti si rosii, iar in livada exista meri si ciresi. Printre soiurile de meri si ciresti, exista cate o varianta altoita.

- Identificati modelul UML folosit



- Ce afiseaza programele?

```
#include <iostream>
using namespace std;
class A {
public:
	virtual int Operatie(int a, int b) { return a + b; }
};
class B {
public:
	virtual int Operatie(int a, int b) { return a - b; }
};
class C : public A{
public:
	int Produs(int a, int b) { return a * b; }
};
int main()
{
	A a;C c;B b;
	memcpy(&c, [b]() {const int x[4] = { ((int*)&b)[0], ((int*)&b)[0], ((int*)&b)[0],
((int*)&b)[0] };  return x; }(), 4 * sizeof(B));
	memcpy(&c, &b, sizeof(B));
	cout << a.Operatie(10,5) << b.Operatie(10,5);
	cout << ((A*)&b)->Operatie(10, 5) << ((B*)&a)->Operatie(10, 5);
	cout << c.Operatie(4, 2) << c.Produs(4, 2) << ((A*)&b)->Operatie(4,2);
	return 0;
}
```

```
#include <iostream>
using namespace std;

class TrickyIterator {
	short f[5];
public:
	TrickyIterator() {
		for (auto it : { 0, 1, 2, 3, 4 })
			f[it] = it;
	}
	short *begin() { return (short*)(((char*)&f[0])+4); }
	short *end() { return (short*)(((char*)&f[0])+5); } //dar cu +8
};

int main()
{
	TrickyIterator t;
	for (auto i : t)
		cout << i;
	return 0;
}
```

```cpp
#include <iostream>
using namespace std;

class TrickyIterator {
	short f[5];
public:
	TrickyIterator() {
		for (auto it : { 0, 1, 2, 3, 4 })
			f[it] = it;
	}
	short *begin() { return (short*)(((char*)&f[0])+4); }
	short *end() { return (short*)(((char*)&f[0])+5); } //dar cu +8
};

int main()
{
	TrickyIterator t;
	for (auto i : t)
		cout << i;
	return 0;
}
```

```cpp
#include <iostream>
using namespace std;

class A { const int x; public: A() :x(0) {} };
class B :public A { int y; public: B() :y(1) {} };
class C :public A, public B { int z, t; public: C() :z(3), t(4) {} };

int main()
{
	C c;
	B b;
	A a;
	cout << sizeof(a) << "," << sizeof(B) << "," << sizeof(c);
	return 0;
}
```

**Liskov:**

**Funcțiile care utilizează pointări sau referințe la clase de bază trebuie să poată folosi instanțe ale claselor derivate fără să își dea seama de acest lucru.**

**Principiul inversarii depndintelor:**

**Modulele de pe nivelurile ierarhice superioare nu trebuie să depindă de modulele de pe nivelurile ierarhice inferioare. Toate ar trebui să depindă doar de module abstracte**