Universitatea Tehnică „Gheorghe Asachi" din Iași
Facultatea de Automatică și Calculatoare

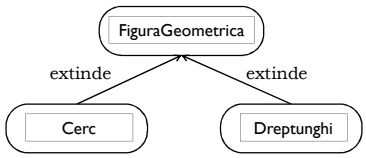
PROGRAMARE II

Curs 11

Polimorfism.

Polimorfismul

- Capacitatea unui obiect de tip A de a exista și de a fi utilizat precum un obiect de tipul B
- Exemplu: utilizarea unui obiect Cerc în locul unui obiect FigurăGeometrică pentru un obiect de tipul FigurăGeometrică



Clasa FigurăGeometrică

```
class FiguraGeometrica
{
public:
	virtual double perimetru ()
	{
		return 0;
	};
	virtual double arie ()
	{
		return 0;
	};
};
```

Clasa Cerc

```
class Cerc : public FiguraGeometrica
{
      int m_ix, m_iy, m_ir;
public:
      Cerc (int x, int y, int r):
        m_ix(x), m_iy(y), m_ir(r){};
      double arie();
      double perimetru();
};

double Cerc::arie ()
{
      return 3.1415 * m_ir * m_ir;
}

double Cerc::perimetru ()
{
      return 2 * 3.1415 * m_ir;
}
```

Clasa Dreptunghi

```
class Dreptunghi : public FiguraGeometrica
{
       int m_ix1, m_iy1, m_ix2, m_iy2;
public:
       Dreptunghi (int x1, int y1, int x2, int y2):
         m_ix1(x1), m_iy1(y1), m_ix2(x2), m_iy2(y2){};
       double perimetru();
};

double Dreptunghi::perimetru ()
{
       return 2 * ((m_ix2 - m_ix1) + (m_iy2 - m_iy1));
}
```

- Clasa Dreptunghi nu are o funcție pentru calculul ariei

Funcția main

```
int main () {

       Cerc c1 (0,0,5);
       Dreptunghi d1 (0,0,2,1);

       FiguraGeometrica * fig1 = &c1;
       FiguraGeometrica * fig2 = &d1;

       FiguraGeometrica fig3;

       cout << "Aria figurii 1 = " << fig1->arie() << endl;
       cout << "Aria figurii 2 = " << fig2->arie() << endl;

       return 0;
}
```

```
Aria figurii 1 = 78.5375
Aria figurii 2 = 0
```

- Probleme:
  - fig3 nu își justifică instanțierea pentru ca nu avem la ce sa-l folosim mai departe în program
    - Cum previn declararea unor obiecte de un anumit tip?
  - In clasa dreptunghi nu există funcția arie
    - Cum pot obliga clasele derivate să redefinească unele funcții din clasa de bază?

Funcții virtuale pure

- Funcții virtuale membre care trebuie definite obligatoriu in clasele derivate
- Sunt utilizate numai pentru a specifica faptul că trebuie să fie rescrise în clasele derivate
- Se mai numesc și funcții abstracte
- O funcție virtuală membră devine virtuală pură prin plasarea " = 0 " la finalul declarației sale:

```
virtual void f(int) = 0;
```

Clase abstracte

- O clasă ce are o funcție virtuală pură se numește clasă abstractă și nu poate instanția nici un obiect
- O funcție virtuală pură poate avea o definiție și în clasa abstractă
- O clasă abstractă - o interfață pentru obiectele accesate prin intermediul pointerilor și referințelor
- O clasă abstractă trebuie să aibă un destructor virtual
- Deoarece o clasă abstractă nu instanțiază obiecte, în mod uzual nu are constructori.

Funcții virtuale pure

```
class FiguraGeometrica
{
public:
	virtual double perimetru () = 0
	{
		return 0;
	};
	virtual double arie () = 0
	{
		return 0;
	};
};
```

- Funcțiile `perimetru` și `arie` devin funcții virtuale pure (funcții abstracte)

Funcții virtuale pure

```
Dreptunghi d1 (0,0,2,1);

Error  1	error C2259: 'Dreptunghi' : cannot instantiate
abstract class


FiguraGeometrica fig3; //se comentează această linie

Error  2	error C2259: 'FiguraGeometrica' : cannot
instantiate abstract class
```

- De ce avem eroare și la instanțierea obiectului `d1`?
  - Orice clasă derivată dintr-o clasă abstractă ce nu redefinește funcțiile virtuale pure din clasa de bază devine și ea abstractă!
  - O funcție virtuală pură ce nu a fost definită în clasa derivată rămâne o funcție virtuală pură.

Funcții virtuale pure

- Pentru a evita transformarea clasei Dreptunghi într-o clasă abstractă, trebuie să redefinim toate funcțiile virtuale pure din clasa de bază

```
class Dreptunghi : public FiguraGeometrica
{
	int m_ix1, m_iy1, m_ix2, m_iy2;
public:
	Dreptunghi (int x1, int y1, int x2, int y2):
	  m_ix1(x1), m_iy1(y1), m_ix2(x2), m_iy2(y2){};
	double arie();
	double perimetru();
};

double Dreptunghi::arie ()
{
	return (m_ix2 - m_ix1) * (m_iy2 - m_iy1);
}
```

Funcții virtuale pure

- La rularea programului, se va afișa pe ecran:

```
Aria figurii 1 = 78.5375
Aria figurii 2 = 2
```

- Pentru apelarea definiției unei funcții abstracte pure, se folosește un apel de tipul:

```
fig2->FiguraGeometrica::arie();
```

- Un astfel de apel nu își găsește utilitatea în cazuri concrete, de aceea se recomanda ca funcțiile virtuale pure să nu aibă definiție în clasa de bază:

```
virtual double perimetru () = 0;
virtual double arie () = 0;
```

Exemplu

- Să se scrie un program care să permită introducerea a `n` figuri geometrice de tip cerc sau dreptunghi, afișează figurile introduse și calculează aria totală a lor

Clasa PersoanaAC

```
class PersoanaAC
{
protected:
	string m_sCnp;
	string m_sNume;
	string m_sAdresa;
public:
	PersoanaAC();
	PersoanaAC(string cnp, string nume, string adresa);
	virtual ~PersoanaAC();
	virtual void afisareProfil();
	virtual void schimbareAdresa(string adresaNoua);
};
```

Clasa StudentAC

```
class StudentAC : public PersoanaAC
{
	int m_ianStudiu;
	int m_inotaP2;
public:
	StudentAC();
	StudentAC(string cnp, string nume, string adresa,
int anStudiu, int notaP2);
	~StudentAC();
	void afisareProfil();
	void inscriereAnStudiu(int anStudiuNou);
};
```

Type casting

```
PersoanaAC *ps5 = new StudentAC ("9999567890122","Pavel",
                    "Iasi",1,10);
ps5->inscriereAnStudiu(4);
ps5->afisareProfil();

error C2039: 'inscriereAnStudiu' : is not a member of
'PersoanaAC'
```

- Care sunt variantele de rezolvare a acestei probleme?
  - Funcții virtuale în clasa de bază
  - Cast la un anumit tip de date (type casting)

Var. I – funcții virtuale

- Declararea unei funcții virtuale în clasa de bază

```
class PersoanaAC
{
...
public:
...
	virtual void inscriereAnStudiu(int){};
};
```

- Se va încarca clasa de bază inutil dacă se vor adăuga toate funcțiile din toate clasele derivate

Type casting

- Conversia unei expresii dintr-un anumit tip în alt tip se numește este cunoscută sub numele de Type casting
- Conversia dintr-o clasă de bază către o clasă derivată este în mod uzual denumită *downcast* datorită modului în care este reprezentată grafic ierarhia de clase.
- Conversia dintr-o clasă derivată către o clasă de bază este în mod uzual denumită *upcast*.
- Similar, conversia de la o clasă derivată către o altă clasă derivată se numește *crosscast*.

Type casting

- Conversia se realizează cu ajutorul operatorilor:
  - *dynamic_cast <>*
  - *static_cast <>*
  - *reinterpret_cast <>*
  - *const_cast <>*

Var. II – type casting

```
PersoanaAC *ps5 = new StudentAC ("9999567890122","Pavel",
                  "Iasi",1,10);

dynamic_cast <StudentAC*> (ps5)->inscriereAnStudiu(5);

static_cast <StudentAC*> (ps5)->inscriereAnStudiu(5);

ps5->afisareProfil();
```

- Conversia unei expresii dintr-un anumit tip în alt tip se numește este cunoscută sub numele de Type casting

Operatorul *dynamic_cast*

- Operatorul *dynamic_cast* are nevoie de doi operanzi:
  - un tip de date încadrat între două paranteze unghiulare < *T** >
  - un pointer încadrat de două paranteze *(p)*

```
dynamic_cast<T*>(p)
```

- Dacă *p* este de tipul *T** sau a unui tip *D** derivat din *T* atunci rezultatul este exact ca și cum s-ar atribui *p* unui *T**.
- *dynamic_cast<T*>(p)* analizează obiectului pointat de *p*. Dacă acel obiect este de clasa *T* ori are o clasă unică *T*, dynamic_cast returnează un pointer de clasă *T**, altfel *nullptr*.

Operatorul *dynamic_cast*

- Dacă p este *nullptr* atunci *dynamic_cast<T*>(p)* returnează *nullptr*.
- Operatorul *dynamic_cast* necesită un pointer sau o referință către un tip polimorfic pentru downcast ori crosscast.

```
class B1
{
public:
    virtual void f(void){};
}

class DB1:public B1
{
public:
    void f(void){};
}

void g(B1* pb1)
{
    DB1* pdb1 = dynamic_cast<DB1*>(pb1); //OK
}
```

Type casting

- dynamic_cast <>
  - Verifică la rulare daca pointerul obținut este de tipul obiectului la care se face cast
  - Nu transformă în void *
- static_cast <>
  - Nu efectuează nici o verificare la rulare asupra obiectului, deci nu se recomandă a fi folosit
  - Necesită verificări suplimentare din partea programatorului pentru a fi sigur ca s-a efectuat cu succes conversia