Clasele

- Entități asemănătoare structurilor ce încapsulează caracteristici (sub formă de atribute) și comportamente (sub formă de metode) prin abstractizarea entităților din lumea reală
- Sunt baza Programării Orientate Obiect (sau Orientată pe Obiecte)
- Permit crearea de tipuri de date noi

Definire

```
class nume_clasa
{
    //atribute si metode
};
```

Atribute

```
class TelefonMobil
{
    string producator;
    string model;
    float nivelBaterie;
    //...
};
```

Metode

```
class TelefonMobil
{
    float nivelBaterie;
    //...
    void incarca(float nivelIncarcare)
    {
        nivelBaterie += nivelIncarcare;
    }
};
```

Tipuri speciale de metode

- Constructorii
- Destructorii
- Funcțiile de acces

Mai multe detalii despre fiecare în cursurile viitoare.

Obiectele

- Instanțele unei clase (variabilele de tipul unei clase) se numesc obiecte, de aici și denumirea paradigmei - Programare Orientată Obiect
- O clasă (cu anumite excepții gândite special în acest sens) poate avea oricâte instanțe
- Diferența dintre o clasă și un obiect e asemănătoare cu diferența dintre o formă de prăjituri și prăjiturile realizate cu acea formă

Crearea de obiecte

```
int main()
{
    TelefonMobil telefonPersonal;
    TelefonMobil telefonDeServiciu;
}
```

Folosirea atributelor și metodelor

```
int main()
{
    TelefonMobil telefonPersonal;
    telefonPersonal.nivelBaterie = 20;
    telefonPersonal.incarca(50);
    cout << telefonPersonal.nivelBaterie; //va afisa 70
}
```

Domenii de vizibilitate

- privat (**private**) - membrii (atributele și metodele) definiți în această zonă pot fi accesați doar de clasa în care sunt definiți - **domeniu implicit**
- protejat (**protected**) - membrii definiți în această zonă de vizibilitate pot fi accesați doar de clasa în care sunt definiți și de clasele derivate din ea (ce o moștenesc - mai multe detalii despre derivare în cursurile viitoare)
- public (**public**) - membrii definiți în această zonă de vizibilitate pot fi accesați de orice entitate (clasa respectivă, clasele derivate, alte clase, funcția main, etc.)

Utilizare domenii de vizibilitate

```
class TelefonMobil
{
public:
	float nivelBaterie;
	void incarca(float nivelIncarcare)
	{
		nivelBaterie += nivelIncarcare;
	}
private:
	string model;
protected:
	string producator;
};
```

Domenii de vizibilitate

- Membrii ce sunt definiți în zona publică spunem, de obicei, că formează **interfața clasei** (modalitatea prin care clasa comunică cu exteriorul)
- A nu se confunda interfața clasei cu noțiunea de interfață (interface - clasă abstractă ce conține doar metode virtuale pure, despre care vom discuta în unul din cursurile viitoare)

Precizare

În C++ singura diferență practică dintre clase (class) și structuri (struct) este dată de faptul că în structuri vizibilitatea implicită este cea publică!

Cu toate acestea între cele două tipuri de entități există o diferență la nivel de concept: structurile se utilizează pentru construcții simple, pe când clasele pentru cele cu o complexitate ridicată.

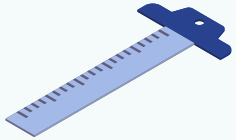
A nu se confunda structurile și clasele din C++ cu cele din alte limbaje (ex: C#)!

Curs 04

CLASE (CONT.)

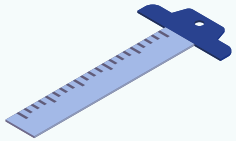
Constructori, destructori, metode de acces

Constructorii



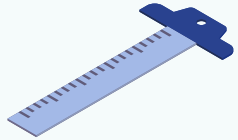
- Metode speciale ce au numele clasei și nu au tip returnat
- Utilizate la crearea obiectelor (inițializarea atributelor)
- Putem crea oricâți constructori, cât timp ei diferă prin numărul sau tipul parametrilor
- Dacă nu există niciun constructor într-o anumită clasă, atunci automat se crează un constructor implicit (fără parametri)

Constructorii



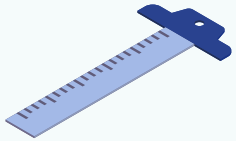
- În cazul în care un constructor cu parametri este prezent, constructorul implicit nu mai este creat automat și nu se pot crea obiecte folosind acest constructor decât dacă acest constructor este scris explicit
- Cu toate că pot exista oricâți constructori în cadrul unei clase, pentru un obiect se va folosi un constructor și numai unul (în funcție de apelul utilizat la crearea obiectului)

Tipuri de constructori



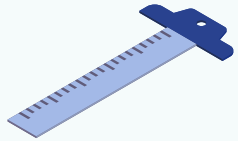
- constructor implicit (fără parametri)
- constructor(i) cu parametri
- constructor de copiere

Constructorul implicit



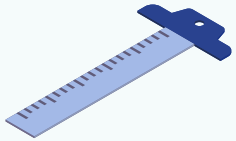
```
class TelefonMobil
{
public:
//...
      TelefonMobil()
      {
            producator = "";
            model = "";
            nivelBaterie = 0;
      }
//...
};
```

Utilizare



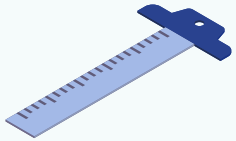
```
int main()
{
    TelefonMobil telefonPersonal;
    TelefonMobil* pTelefonDeServiciu = new TelefonMobil();
    //...
    delete pTelefonDeServiciu;
}
```

Constructorul cu parametri



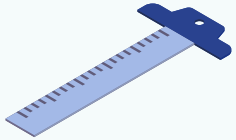
```
class TelefonMobil
{
public:
//...
      TelefonMobil(string _producator, string _model)
      {
            producator = _producator;
            model = _model;
            nivelBaterie = 0;
      }
//...
};
```

Utilizare



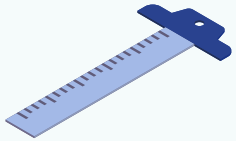
```
int main()
{
    TelefonMobil telefonPersonal("Samsung", "S20");
    TelefonMobil* pTelefonDeServiciu =
            new TelefonMobil("Apple", "Iphone 12");
    //...
    delete pTelefonDeServiciu;
}
```

Comasarea celor doi constructori



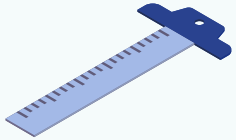
```
class TelefonMobil
{
public:
//...
      TelefonMobil(string _producator = "", string _model = "")
      {
            producator = _producator;
            model = _model;
            nivelBaterie = 0;
      }
//...
};
```

Apelul explicit al unui constructor din alt constructor



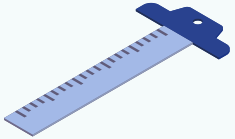
```
class TelefonMobil
{
public:
//...
      TelefonMobil(string _producator, string _model) : TelefonMobil()
      {
            producator = _producator;
            model = _model;
            //nivelBaterie = 0;
      }
//...
};
```

Utilizarea inițializării la definire



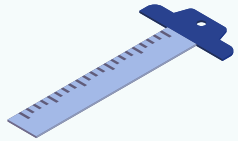
```
class TelefonMobil
{
public:
//...
	TelefonMobil(string _producator, string _model) : producator(_producator), model(_model), nivelBaterie(0)
	{
	}
//...
};
```

Caz special: constructorul cu un parametru



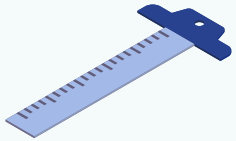
```
class TelefonMobil
{
public:
//...
	TelefonMobil(int _nivelBaterie)
	{
		nivelBaterie = _nivelBaterie;
	}
//...
};

int main()
{
	TelefonMobil telefonPersonal = 20;
}
```

Constructorul de copiere



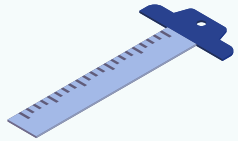
- Constructor utilizat explicit pentru a crea copii ale obiectelor
- Compilatorul apelează totuși implicit constructorul de copiere în două situații:
  - transmiterea unui obiect ca parametru într-o funcție prin valoare
  - returnarea unui obiect prin valoare dintr-o funcție
- Dacă nu este creat explicit, atunci se crează automat un constructor de copiere în clasă ce realizează copiere byte cu byte a obiectului curent
- Acest lucru poate crea probleme atunci când clasa are membri pointeri

Constructorul de copiere



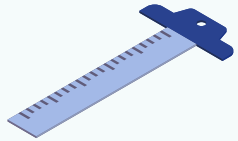
```
class TelefonMobil
{
public:
//...
      TelefonMobil(const TelefonMobil& telefon)
      {
            producator = telefon.producator;
            model = telefon.model;
            nivelBaterie = telefon.nivelBaterie;
      }
//...
};
```

Utilizare



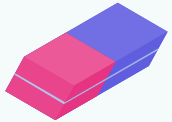
```
int main()
{
    TelefonMobil telefonPersonal("Samsung", "S20");
    //in acest fel
    TelefonMobil telefonDeServiciu(telefonPersonal);
    //sau in acest fel
    TelefonMobil telefonPersonal2 = telefonPersonal;
}
```

Utilizare



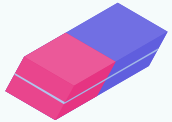
```
TelefonMobil incrementeazaNivelBaterie(TelefonMobil telefon)
{
	telefon.nivelBaterie++;
	return telefon;
}

int main()
{
	TelefonMobil telefonPersonal("Samsung", "S20");
	telefonPersonal.nivelBaterie = 50;
	TelefonMobil telefon = incrementeazaNivelBaterie(telefonPersonal);
	//constructorul de copiere se va apela automat de doua ori pentru linia de cod de mai sus
}
```

Destructorul



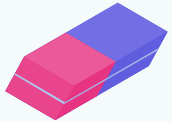
- Metodă specială ce are același nume ca și clasă, precedat de caracterul ~
- Utilizat la distrugerea obiectelor (dezalocarea memoriei alocate în constructori, eliberarea resurselor, etc.)
- Există un singur destructor ce se apelează pentru toate obiectele, indiferent de constructorul cu care au fost create
- Dacă nu există destructor într-o anumită clasă, atunci automat se crează unul gol
- Se va defini un destructor explicit doar atunci când e nevoie de acest lucru

Destructor



```
class TelefonMobil
{
public:
//...
      ~TelefonMobil()
      {
            //dezalocare memorie sau eliberare resurse
      }
//...
};
```

Apel destructor



```
int main()
{
      TelefonMobil telefonPersonal("Samsung", "S20");
      TelefonMobil* pTelefonDeServiciu =
                  new TelefonMobil("Apple", "Iphone 12");
      //pentru obiectele alocate dinamic la apelul delete
      delete pTelefonDeServiciu;
}
//pentru celelalte obiecte atunci cand ies din uz
//in cazul acestui exemplu la iesirea din programul principal
```