Laborator #1
Clase & Obiecte

Concepte de bază

Concepte de bază

Clasă = tip de date definit de utilizator care conține date și funcții membre.

Date membre – **starea** clasei

Funcții membre – **comportamentul** clasei

Sintaxă

```
class <nume_clasa> {
<specificator_de_acces>:
        <date_membre>
        <functii_membre>
};
```

Obiect = instanță a unei clase.

Fiecare obiect primește o copie a tuturor datelor membre.

Sintaxă

```
<nume_clasa> <nume_obiect>;
```

Exemplu

```
#include <iostream>
#include <string>

class Masina{
public:
    std::string culoare, marca;
    int an, km, pret;

    void init(std::string culoare, std::string marca,
              int an, int km, int pret){
        this->culoare = culoare;
        this->marca = marca;
        this->an = an;
        this->km = km;
        this->pret = pret;
    }

    void display_info(){
        std::cout << culoare << std::endl;
        std::cout << marca << std::endl;
        std::cout << an << std::endl;
        std::cout << km << std::endl;
        std::cout << pret << std::endl;
    }
};
```

`this` = pointer implicit către obiectul a cărui funcție se execută.

`this` este argument transmis implicit către toate funcțiile membre non-statice.

Exemplu

```
#include <iostream>
#include <string>

class Masina{
public:
    std::string culoare, marca;
    int an, km, pret;

    void init(std::string culoare, std::string marca,
              int an, int km, int pret){
        this->culoare = culoare;
        this->marca = marca;
        this->an = an;
        this->km = km;
        this->pret = pret;
    }

    void display_info(){
        std::cout << culoare << std::endl;
        std::cout << marca << std::endl;
        std::cout << an << std::endl;
        std::cout << km << std::endl;
        std::cout << pret << std::endl;
    }
};
```

```
int main () {
    std::string v_culoare = "ALB";
    std::string v_marca = "Dacia";
    int v_an = 2021;
    int v_km = 32000;
    int v_pret = 25000;

    Masina m;

    m.init(v_culoare, v_marca, v_an, v_km, v_pret);
    m.an = 2020;
    m.display_info();

    return 0;
}
```

```
ALB
Dacia
2020
32000
25000

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

Specificatori de acces

Specificatori de acces

Specificatori de acces = cuvinte cheie care modifică drepturile de a accesa membrii (datele sau funcțiile) unei clase:

1. `public:` membrii clasei pot fi accesați de oriunde este vizibilă clasa.
2. `protected:` - membrii clasei pot fi accesați de membrii aceleiași clase (+ 'prieteni') și de către membrii claselor derivate. (@moștenire)
3. `private:` - membrii clasei pot fi accesați doar de membrii aceleiași clase (+ 'prieteni')

Default: `private`

Efectul unui specificator de acces durează până la întâlnirea unui nou specificator de acces sau până la finalul clasei (oricare eveniment apare primul).

Specificatori de acces

```
#include <iostream>
#include <string>

class Masina{
private:
    std::string culoare, marca;
    int an, km, pret;

public:
    void init(std::string culoare, std::string marca,
              int an, int km, int pret){
        this->culoare = culoare;
        this->marca = marca;
        this->an = an;
        this->km = km;
        this->pret = pret;
    }

    void display_info(){
        std::cout << culoare << std::endl;
        std::cout << marca << std::endl;
        std::cout << an << std::endl;
        std::cout << km << std::endl;
        std::cout << pret << std::endl;
    }
};
```

```
int main () {
    std::string v_culoare = "ALB";
    std::string v_marca = "Dacia";
    int v_an = 2021;
    int v_km = 32000;
    int v_pret = 25000;

    Masina m;

    m.init(v_culoare, v_marca, v_an, v_km, v_pret);
    m.an = 2020;
    m.display_info();

    return 0;
}
```

```
C:\Users...      In function 'int main()':
C:\Users... 43   error: 'int Masina::an' is private within this context
C:\Users... 8    note: declared private here
                 === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 ...
```

Soluție

Data hiding = procesul de a “ascunde” (restricționa accesul la) datele membre față de restul lumii.

- Pe cât posibil, facem toate datele membre de tip `private`.
- Pentru a le accesa/modifica, creăm funcții membre (getters/setters) de tip `public` care interacționează cu acestea.

http://web.archive.org/web/20080906224409/http:/www.itmweb.com/essay550.htm
https://stackoverflow.com/questions/24626/abstraction-vs-information-hiding-vs-encapsulation/8694874#8694874

Exemplu

```
class Rectangle{
private:
    float width;
    float height;

public:
    void set_width(float width){
        this -> width = width;
    }

    float get_width(){
        return this->width;
    }

    void set_height(float height){
        this->height = height;
    }

    float get_height(){
        return this->height;
    }
};
```

```
#include<iostream>

class Rectangle {...};

int main(){
    Rectangle r;

    r.set_width(5);
    r.set_height(10);

    std::cout << "width=" << r.get_width() << '\n';
    std::cout << "height=" << r.get_height() << '\n';

    return 0;
}
```

Nu mai există funcție de inițializare?
Fiecare dată membră trebuie inițializată independent?

Constructori

Constructori (ctor)

Constructor = funcție membră specială a unei clase, cu același nume cu clasa, folosită pentru a inițializa datele membre ale unui obiect.

- Este executat automat atunci când se creează un obiect al clasei respective.
- Nu se poate apela explicit dintr-o instanță a clasei.
- Nu are tip de date returnat (nici măcar `void`).

Constructor implicit (default ctor)

```
#include <iostream>

class Rectangle{
private:
    float width;
    float height;

public:
    // Constructor default fara parametri:
    Rectangle(){
        this->width = 0;
        this->height = 0;
    }

};

int main(){
    Rectangle r; // apel constructor default

    std::cout << "width:" << r.get_width() << "\n"; // afiseaza 0
    std::cout << "height:" << r.get_height() << "\n"; // afiseaza 0

    return 0;
}
```

Constructor parametrizat (parameterised ctor)

```
#include <iostream>

class Rectangle{
private:
    float width;
    float height;

public:
    // Constructor parametrizat cu doi parametri:
    Rectangle(float width, float height){
        this->width = width;
        this->height = height;
    }

};

int main(){
    Rectangle r1; // eroare: constructorul default nu mai exista
    Rectangle r2(7, 10); // width = 7, height = 10

    return 0;
}
```

Existența unui constructor parametrizat suprimă constructorul default

Constructor parametrizat (parameterised ctor)

```
#include <iostream>

class Rectangle{
private:
    float width;
    float height;

public:
    // Constructor default cu toți parametrii având valori default:
    Rectangle(float width=0, float height=0){
        this->width = width;
        this->height = height;
    }

};

int main(){
    Rectangle r1; // width = 0, height = 0
    Rectangle r2(5); // width = 5, height = 0
    Rectangle r3(7, 10); // width = 7, height = 10

    return 0;
}
```

Funcție cu valori implicite ⇔ la momentul apelului, dacă nu este transmisă o valoare pentru argumentele respective, atunci acestea sunt inițializate cu valorile implicite din lista de argumente.

În timpul apelului funcției, argumentele sunt copiate de la stânga la dreapta.

Funcții supraîncărcate

Funcții supraîncărcate = două (sau mai multe) funcții care au același nume, dar diferă prin lista de parametri (fie prin tipul lor, fie prin numărul lor).

Constructori supraîncărcați (overloaded ctor)

```
class Rectangle{
private:
    float width;
    float height;

public:
    Rectangle(){
        std::cout << "Constructor default.\n";
        this->height = 0;
        this->width = 0;
    }
    Rectangle(float x){
        std::cout << "Constructor cu param float.\n";
        this->width = x;
        this->height = x;
    }
    Rectangle(int x){
        std::cout << "Constructor cu param int.\n";
        this->width = sqrt(x);
        this->height = sqrt(x);
    }
    Rectangle(float width, float height){
        std::cout << "Constructor cu 2 parametri.\n";
        this->width = width;
        this->height = height;
    }
};
```

```
#include <iostream>

class Rectangle{…}

int main () {
    Rectangle r1; // width = 0, height = 0
    Rectangle r2(5.23f); // width = 5.23, height = 5.23
    Rectangle r3(25); // width = 5, height = 5
    Rectangle r4(3, 10); // width = 3, height = 10
    return 0;
}
```

```
Constructor default.
Constructor cu param float.
Constructor cu param int.
Constructor cu 2 parametri.

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```

Sfârșit laborator #1