

Reprezentarea UML a claselor.

Mihai Gabroveanu

De ce UML?

- n este un standard deschis
- n surprinde întreg ciclul de viață al dezvoltării software-ului
- n acoperă multe tipuri de aplicații

Ce este UML?

- n **UML** acronim pentru *Unified Modeling Language*
- n Este un limbaj vizual de modelare utilizat pentru specificarea, construirea și documentarea sistemelor de aplicații orientate obiect și nu numai.

Diagrame UML

- n O **diagramă** este o prezentare grafică ale unui set de elemente, cel mai adesea exprimate ca un graf de noduri (elementele) și arce (relațiile).
- n Tipuri de diagrame UML:
 - Diagrame ale Cazurilor de Utilizare (Use Case)
 - Diagrame de clase
 - Diagrame de obiecte
 - Diagrame de secvență
 - Diagrame de stare
 - Diagrame de colaborare
 - Diagrame de activitate

Diagramele de Clase

- n** *Diagramele de clase* sunt folosite în modelarea orientată obiect pentru a descrie structura statică a sistemului, modului în care este el structurat.
- n** Oferă o notație grafică pentru reprezentarea:
 - „ **claselor** ” - entități ce au caracteristici comune
 - „ **relațiilor** ” - relațiile dintre două sau mai multe clase

Reprezentarea UML a claselor

5

Specificarea Atributelor

- n** Sintaxa specificării atributelor din compartimentul atributelor este următoarea:

vizibilitate idAtribut : tip = valoare_implicita

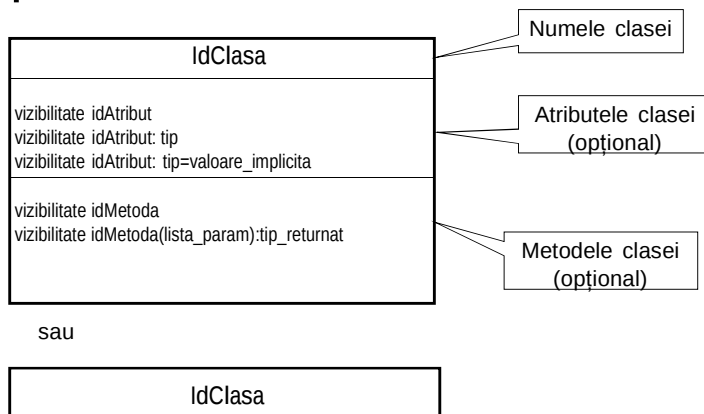
unde:

- „ *vizibilitate* ” reprezintă protecția atributului și poate să aibă una din următoarele valori
 - + = public,
 - = privat și
 - # = protected
- „ *idAtribut* ” este identificatorul atributului
- „ *tip* ” este tipul acestuia
- „ *valoare_implicita* ” reprezintă valoarea inițială a atributului (opțională)

Reprezentarea UML a claselor

7

Reprezentarea unei clase



Reprezentarea UML a claselor

6

Specificarea Metodelor

- n** Sintaxa specificării metodelor sau operațiilor din compartimentul metodelor este următoarea:

vizibilitate idMetoda(idP1:tip1, ..., idPn:tipn) : tip_returnat

unde:

- „ *vizibilitate* ” reprezintă protecția metodei. Poate să aibă una din următoarele valori
 - + = public,
 - = private și
 - # = protected
- „ *idMetoda* ” este identificatorul metodei
- „ *idP1, ..., idPn* ” sunt parametri metodei
- „ *tip1, ..., tipn* ” sunt tipurile parametrilor
- „ *tip_returnat* ” reprezintă tipul valorii returnate de metodă

Reprezentarea UML a claselor

8

Relațiile

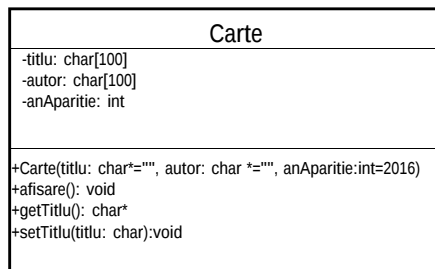
- n Toate relațiile din cadrul diagramei de clase sunt reprezentate grafic printr-o succesiune de segmente care leagă o clasă de alta.
- n Tipuri de *relații*:
 - .. asocieri
 - .. generalizări
 - .. dependențe
 - .. relații de rafinare.
- n **Asocierea** este o conexiune între clase, ce reprezintă o legătură semantică între obiectele claselor implicate în relație.

Reprezentarea UML a claselor

9

Reutilizarea Codului

Diagrame de Clase. Exemplu



```
class Carte{  
    private:  
        char titlu[100];  
        char autor[100];  
        int anAparitie;  
    public:  
        Carte(char *titlu="", char *autor="", int an=2016);  
        void afisare();  
        char* getTitlu();  
        void setTitlu(char *titlu);  
};
```

Reprezentarea UML a claselor

10

Reutilizarea Codului

- n În programarea orientată obiect *reutilizarea codului* se poate realiza prin:
 - .. **Compunere** (Agregare) – includerea de obiecte în cadrul altor obiecte
 - .. **Moștenire** – posibilitatea de a defini o clasă extinzând o altă clasă deja existentă
 - .. **Clase și funcții template**

Reprezentarea UML a claselor

12

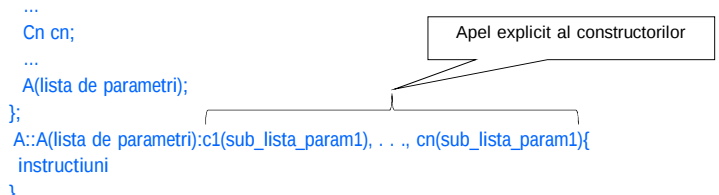
Agregarea

- n **Agregare** = mecanismul prin care definim o clasă nouă ce include una sau mai multe date membre care au ca tip clase deja definite.
- n Ne permite construirea de clase complexe pornind de la clase mai simple

Agregare.Constructori (II)

- n Apelul constructorilor claselor C1, ..., Cn se poate face:
 - *explicit la definirea constructorului clasei A*

```
class A {  
    C1 c1;  
    ...  
    Cn cn;  
    ...  
    A(lista de parametri);  
};  
A::A(lista de parametri):c1(sub_lista_param1), ... , cn(sub_lista_param1){  
    instructiuni  
}
```


 - *automat de către compilator* – dacă există obiecte cK pentru care nu se apelează constructorul cu parametri la definirea constructorului clasei A, atunci se apelează automat constructorii impliciti ai acelor clase (deci trebuie să avem un astfel de constructor)

Agregare. Constructori (I)

- n Dacă o clasă A conține ca date membru obiecte ale altor clase (C1, ..., Cn) atunci constructorul clasei A va avea suficienți parametri pentru a inițializa datele membru ale claselor C1,..., Cn.
- n La crearea unui obiect al clasei A se vor apela mai întâi constructorii claselor C1, ... ,Cn pentru a se inițializa obiectele incluse în obiectul clasei A și apoi se vor executa instrucțiunile constructorului clasei A.

Tipuri de agregare

- n Există două variante de agregare diferențiate de relația dintre agregant și subobiecte agregate:
 - **Compoziția** - subobiectele agregate aparțin exclusiv agregatului din care fac parte, iar durata de viață coincide cu cea a agregatului
 - **Agregarea propriu-zisă** - subobiectele au o existență independentă de agregatele ce le partajază. Mai mult, un obiect poate fi partajat de mai multe agregate.

Agregare. Destructori

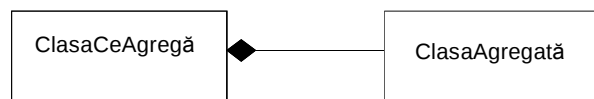
- La distrugerea unui obiect al clasei A se va executa mai destructorul clasei A apoi se vor apela automat destructorii claselor C1, ... ,Cn în ordinea inversă apelurilor constructorilor.

Exemplu

- Considerăm următorul enunț: *Un **parc auto** este format dintr-o mulțime de mașini. Pentru fiecare **mașină** din parc se cunosc următoarele detalii: numărul de înmatriculare, marca, seria **motorului** și tipul de carburant utilizat de acesta.*
- Să se modeleze utilizând diagrame UML.

Agregarea. Reprezentare UML

Compozitia



Agregarea propriu-zisă

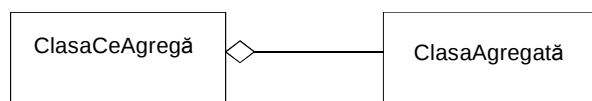
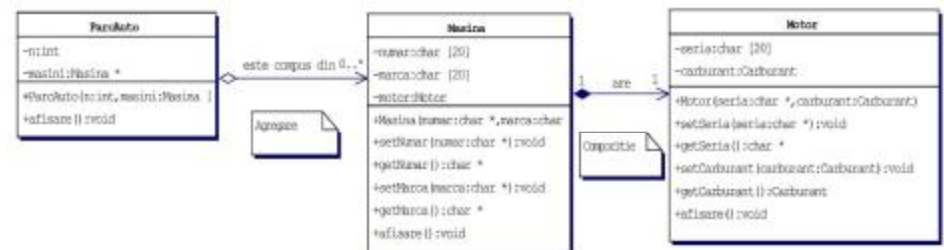


Diagrama UML



Clasa Motor

```
enum Carburant { benzina=0, motorina};

class Motor {
private:
    char seria[20];
    Carburant carburant;
public:
    Motor(char *seria, Carburant carburant);
    void setSeria(char *seria);
    char* getSeria();
    void setCarburant(Carburant carburant);
    Carburant getCarburant();
    void afisare();
};

Motor::Motor(char *seria, Carburant carburant){
    setSeria(seria);
    setCarburant(carburant);
}

void Motor::setSeria(char *seria){
    strcpy(this->seria, seria);
}

char* Motor::getSeria(){
    return seria;
}

Carburant Motor::getCarburant(){
    return carburant;
}

void Motor::setCarburant(Carburant carburant){
    this->carburant = carburant;
}

void Motor::afisare(){
    cout<<"Serie Motor:"<<seria<<endl;
    cout<<"Tip Combustibil:" <<
    ((carburant==motorina)?"motorina":"benzina");
    cout<<endl;
}
```

Reprezentarea UML a claselor

21

Clasa ParcAuto

```
class ParcAuto{
    int n;
    Masina *masini;
public:
    ParcAuto(int n, Masina masini[]);
    void afisare();
};

ParcAuto::ParcAuto(int n, Masina masini[]){
    this->n = n;
    this->masini = new Masina[n];
    for(int i=0;i<n;i++)
        this->masini[i] = masini[i];
}

void ParcAuto::afisare(){
    for(int i=0;i<n;i++)
        masini[i].afisare();
}

int main(){
    Masina m[] = {Masina("DJ-01-UCV", "Audi",
    "1234", motorina), Masina("DJ-02-UCV", "Logan",
    "2121", benzina)};
    ParcAuto parc(2,m);
    parc.afisare();
    getch();
}
```

Reprezentarea UML a claselor

23

Clasa Masina

```
class Masina{
private:
    char numar[20];
    char marca[20];
    Motor motor;
public:
    Masina(char *numar="", char *marca="", char
    *seria="", Carburant carburant=benzina);
    void setNumar(char *numar);
    char* getNumar();
    void setMarca(char *marca);
    char* getMarca();
    void afisare();
};

Masina::Masina(char *numar, char *marca, char
*seria, Carburant carburant):motor(seria,
carburant ){
    setNumar(numar);
    setMarca(marca);
}

void Masina::setNumar(char *numar){
    strcpy(this->numar, numar);
}

char* Masina::getNumar(){
    return numar;
}

void Masina::setMarca(char *marca){
    strcpy(this->marca,marca);
}

char* Masina::getMarca(){
    return marca;
}

void Masina::afisare(){
    cout<<"Numar:"<<numar<<endl;
    cout<<"Marca:"<<marca<<endl;
    motor.afisare();
}
```

Reprezentarea UML a claselor

22

Temă

1. Implementați clasa DataCalendaristică (zi, lună, an) și Persoana(nume, data nașterii). Să se implementeze o metodă care calculează vârsta persoanei la o anumită dată.
2. Realizați diagrama de clase UML pentru a reprezenta următoarele clase:
 - n Jucător de fotbal
 - n Antenor
 - n Echipă
3. Să se implementeze în C++ clasele din exercițiul anterior

Reprezentarea UML a claselor

24