Laborator #3
Moștenire

Concepte de bază

Moștenire = procesul prin care o clasă (de bază) este extinsă într-o nouă clasă (derivată) prin preluarea datelor și funcțiilor membre.

- Clasa derivată “moștenește” caracteristicile clasei de bază.
- Clasa derivată poate adăuga informații suplimentare la cele moștenite.
- Definește o relație de tipul “este o/un” (is a) => ierarhizare.
- Se previne rescrierea codului atunci când se extinde funcționalitatea unei clase.
- Clasa de bază trebuie definită complet înainte de clasa derivată.

Concepte de bază

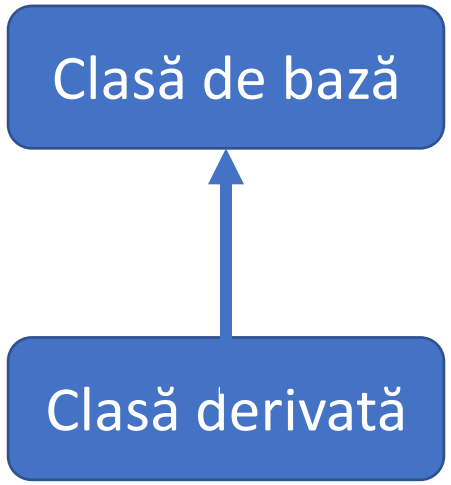
O clasă derivată moștenește de la clasa de bază toți membrii, mai puțin:

- Constructorii și destructorii;
- Operatorul de asignare (`operator=`);
- “Prietenii” săi;
- Membrii privați.

Concepte de bază

Sintaxă:

```
class <nume_derivata>:
                [virtual] [specificator_acces] <nume_baza_1>,
                [virtual] [specificator_acces] <nume_baza_2>,...{
        ...
};
```



Concepte de bază

Accesul oferit de specificatorii de acces din clasă

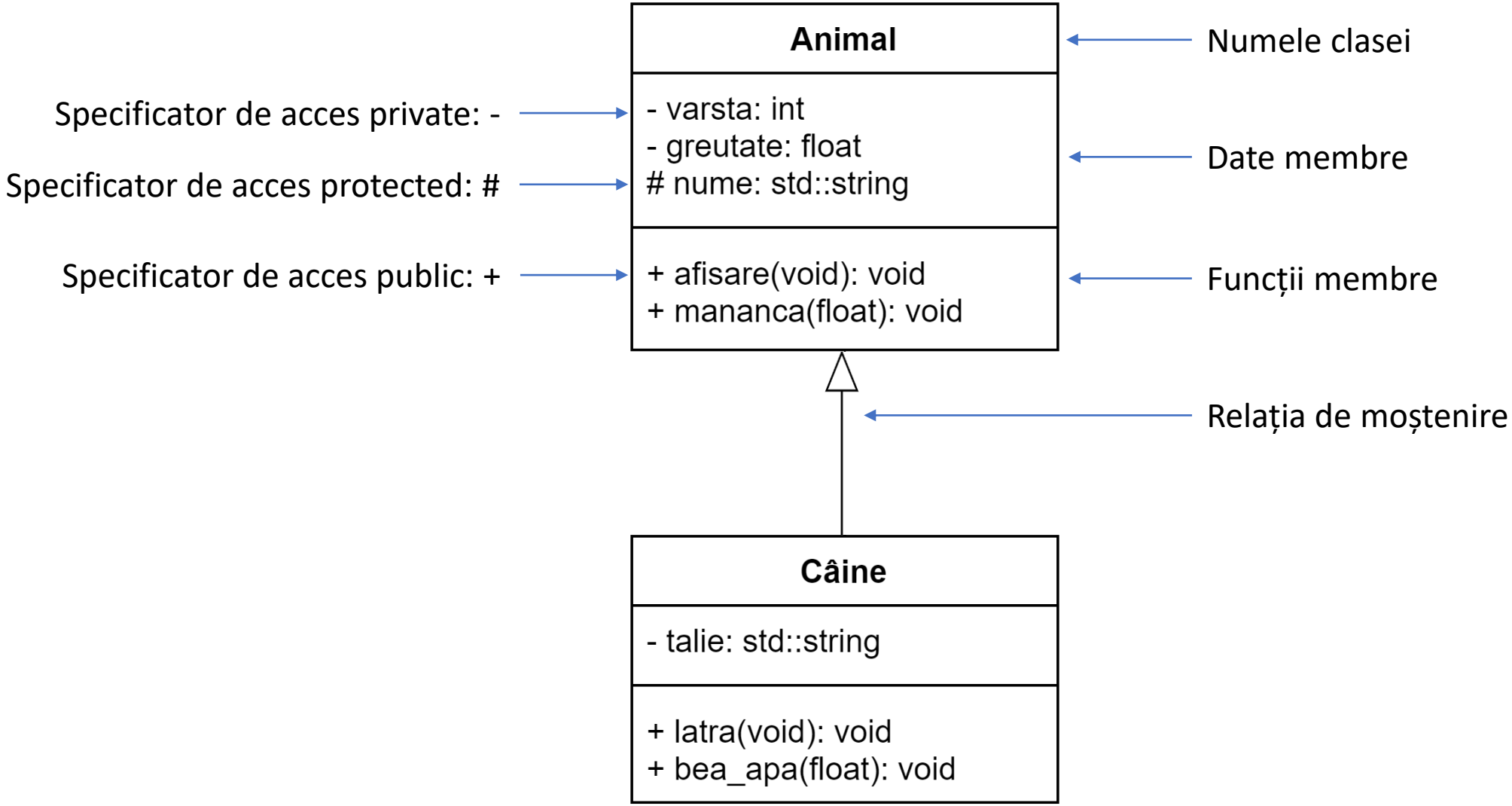
	Specificator acces		
	`public`	`protected`	`private`
Membrii aceleiași clase	Da	Da	Da
Membrii clasei derivate	Da	Da	Nu
Non-membri (exterior)	Da	Nu	Nu

Specificatorii de acces rezultați în clasa derivată

Specificator acces în clasa de bază	Specificator acces moștenire		
	`public`	`protected`	**`private`** ←
`public`	`public`	`protected`	`private`
`protected`	`protected`	`protected`	`private`
`private`	inaccesibil	inaccesibil	inaccesibil

Dacă nu se declară explicit un specificator (default)

UML (Unified Modeling Language)

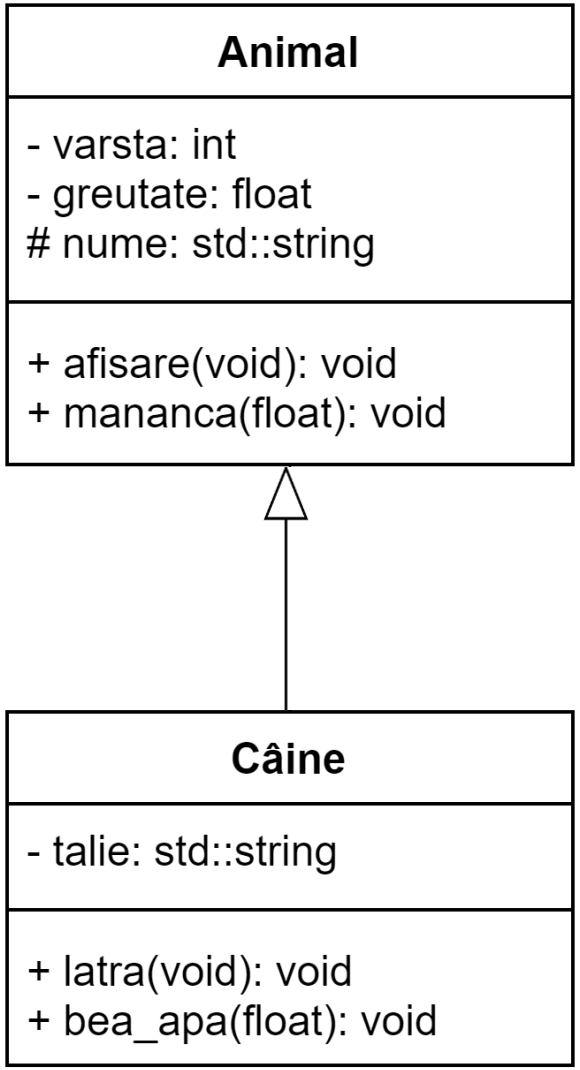


Exemplu

```
#include <iostream>

class Animal{
private:
    int varsta;
    float greutate;
protected:
    std::string nume;
public:
    void afisare();
    void mananca(float);
};

class Caine: public Animal{
private:
    std::string talie;
public:
    void bea_apa(float);
    void latra();
};
```



Ordinea apelării constructorilor/destructorilor

Ordinea apelării constructorilor

O clasă derivată preia de la clasa de bază toți membrii (cu excepțiile menționate) => pentru crearea unei instanțe din derivată trebuie să fie instanțiată, mai întâi, clasa de bază prin apelarea constructorului său.

Într-un lanț de moșteniri succesive, ordinea apelului constructorilor este de la clasa de bază spre clasa cea mai derivată.

Ordinea apelării destructorilor

O clasă derivată conține toți membrii bazei sale, la care mai adaugă o serie de membri particulari, necunoscuți clasei de bază. Fiecare destructor se va ocupa de a șterge datele member ale clasei din care face parte.

Într-un lanț de moșteniri succesive, ordinea apelului destructorilor este de la clasa cea mai derivată spre clasa de bază.

Ordinea apelării constructorilor/destructorilor

```
#include <iostream>

class Baza{
public:
    Baza(){
        std::cout << "Constructor Baza" << std::endl;
    }
    ~Baza(){
        std::cout << "Destructor Baza" << std::endl;
    }
};

class Derivata: public Baza{
public:
    Derivata(){
        std::cout << "Constructor Derivata" << std::endl;
    }
    ~Derivata(){
        std::cout << "Destructor Derivata" << std::endl;
    }
};

int main(){
    Derivata d;
    return 0;
}
```

```
Constructor Baza
Constructor Derivata
Destructor Derivata
Destructor Baza

Process returned 0 (0x0)   execution time : 0.285 s
Press any key to continue.
```

Ascunderea membrilor

Ascunderea membrilor

Definirea în clasa derivată a unui membru (dată/funcție) cu același nume cu un membru din clasa de bază duce la ascunderea membrului din clasa de bază.

Accesarea membrilor ascunși din clasa de bază se poate face prin calificarea completă a numelui membrului în momentul utilizării.

Aducerea membrului ascuns din clasa de bază în același domeniu de definiție (scope) cu clasa derivată se face folosind sintaxa `using`.

Ascunderea datelor membre

```
#include <iostream>

class Animal{
protected:
    float greutate;
public:
    void afisare(){
        std::cout << "Animal::afisare()" << std::endl;
        std::cout << greutate << std::endl;
    }
};
class Caine : public Animal{
public:
    float greutate;
};

int main(){
    Caine c;
    c.greutate = 5;
    c.afisare();

    return 0;
}
```

`float greutate` ascunde data membră `Animal::greutate`

Inițializare `Caine::greutate` cu valoarea 5

Afișare `Animal::greutate` – valoare nedefinită

`Caine::greutate` și `Animal::greutate` sunt două variabile diferite, cu același nume, însă domenii de definiție diferite

Ascunderea datelor membre

```
#include <iostream>

class Animal{
protected:
    float greutate;
public:
    void afisare(){
        std::cout << "Animal::afisare()" << std::endl;
        std::cout << greutate << std::endl;
    }
};
class Caine : public Animal{
public:
    float greutate;
};

int main(){
    Caine c;
    c.Animal::greutate = 5;
    c.afisare();

    return 0;
}
```

`float greutate` **ascunde data membră** `Animal::greutate`

Inițializare `Animal::greutate` **cu valoarea 5**

Afișare `Animal::greutate` **– valoarea 5**

Ascunderea datelor membre

```
#include <iostream>

class Animal{
protected:
    float greutate;
public:
    void afisare(){
        std::cout << "Animal::afisare()" << std::endl;
        std::cout << greutate << std::endl;
    }
};
class Caine : public Animal{
public:
    using Animal::greutate;
};

int main(){
    Caine c;
    c.greutate = 5;
    c.afisare();

    return 0;
}
```

Aducerea datei membre `Animal::greutate` în domeniul de definiție al clasei `Caine` și schimbarea specificatorului de acces din `protected` în `public`

Inițializare `Animal::greutate` cu valoarea 5

Afișare `Animal::greutate` – valoarea 5

Ascunderea funcțiilor membre

```cpp
#include <iostream>

class Animal{
private:
    int varsta;
protected:
    float greutate;
    std::string nume;
public:
    void afisare(){
        std::cout << "Animal::afisare()" << std::endl;
        std::cout << varsta << std::endl;
        std::cout << nume << std::endl;
        std::cout << greutate << std::endl;
    }
};

class Caine : public Animal{
private:
    std::string talie;
public:
    void afisare(){
        std::cout << "Caine::afisare()" << std::endl;
        std::cout << talie << std::endl << std::endl;
    }
};
```

```cpp
int main(){
    Caine c;
    c.afisare(); // apeleaza Caine::afisare()
    c.Animal::afisare(); // apeleaza Animal::afisare()

    return 0;
}
```

`Caine::afisare()` ascunde funcția `Animal::afisare()`

Ascunderea funcțiilor membre

```cpp
#include <iostream>

class Animal{
private:
    int varsta;
protected:
    float greutate;
    std::string nume;
public:
    void afisare(){
        std::cout << "Animal::afisare()" << std::endl;
        std::cout << varsta << std::endl;
        std::cout << nume << std::endl;
        std::cout << greutate << std::endl;
    }
};

class Caine : public Animal{
private:
    std::string talie;
public:
    void afisare(int a){
        std::cout << "Caine::afisare()" << std::endl;
        std::cout << talie << std::endl << std::endl;
    }
};
```

```cpp
int main(){
    Caine c;
    c.afisare(); // eroare: no matching function for
call to 'Caine::afisare()'
    c.afisare(1); // apeleaza Caine::afisare(int)
    return 0;
}
```

`Caine::afisare(int)` ascunde funcția `Animal::afisare()`

Ascunderea funcțiilor membre

```
#include <iostream>

class Animal{
private:
    int varsta;
protected:
    float greutate;
    std::string nume;
public:
    void afisare(){
        std::cout << "Animal::afisare()" << std::endl;
        std::cout << varsta << std::endl;
        std::cout << nume << std::endl;
        std::cout << greutate << std::endl;
    }
};
class Caine : public Animal{
private:
    std::string talie;
public:
    using Animal::afisare;
    void afisare(int a){
        std::cout << "Caine::afisare()" << std::endl;
        std::cout << talie << std::endl << std::endl;
    }
};
```

```
int main(){
    Caine c;
    c.afisare(); // apeleaza Animal::afisare()
    c.afisare(1); // apeleaza Caine::afisare(int)

    return 0;
}
```

Se aduce în domeniul de definiție funcția `Animal::afisare()`

Sfârșit laborator #3