


# Laborator #4

## Virtualitate

# Suprascrierea funcțiilor

 Dacă o **funcție (ne-virtuală)** din clasa de bază este redefinită în clasa derivată și apelul ei se va face prin intermediul unui pointer/referință la un (sub)obiect, atunci se va apela funcția din clasa corespondentă tipului **pointer-ului/referinței**.

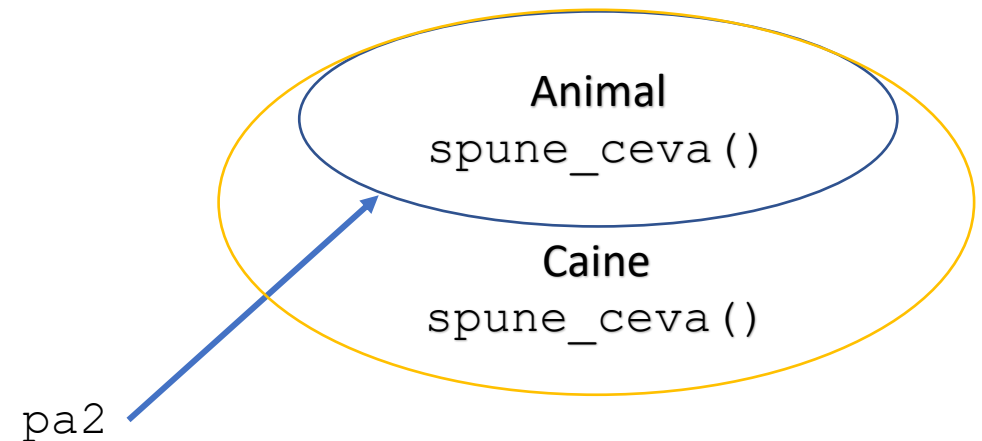
La momentul compilării se cunoaște asocierea dintre obiecte și funcțiile apelate => se realizează legarea statică (early/static binding).

# Suprascrierea funcțiilor

```
#include <iostream>
class Animal {
public:
    void spune_ceva() {std::cout << "Animal vorbitor" << std::endl;}
};
class Caine : public Animal {
public:
    void spune_ceva() {std::cout << "Caine vorbitor" << std::endl;}
};
int main() {
    Animal* pa1 = new Animal();
    Caine* pc = new Caine();
    → Animal* pa2 = new Caine();

    pa1->spune_ceva();
    pc->spune_ceva();
    → pa2->spune_ceva();

    return 0;
}
```



# Suprascrierea funcțiilor

```
#include <iostream>
class Animal {
public:
    void spune_ceva() {std::cout << "Animal vorbitor" << std::endl;}

};
class Caine : public Animal {
public:
    void spune_ceva() {std::cout << "Caine vorbitor" << std::endl;}

};
int main() {
    Animal* pa1 = new Animal();
    Caine* pc = new Caine();
    → Animal* pa2 = new Caine();


    pa1->spune_ceva();
    pc->spune_ceva();
    → pa2->spune_ceva();

    return 0;
}
03/12/21
```

```
Animal vorbitor
Caine vorbitor
Animal vorbitor

Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.
```

# Suprascrierea funcțiilor

 Dacă o **funcție virtuală** din clasa de bază este redefinită în clasa derivată și apelul ei se va face prin intermediul unui pointer/referință la un (sub)obiect, atunci se va apela funcția din clasa corespondentă tipului **(sub)obiectului** către care indică pointerul/referința.

La momentul compilării nu se cunoaște asocierea dintre obiecte și funcțiile apelate. Aceasta devine cunoscută în momentul rulării programului => se realizează legarea dinamică (late/dynamic binding).

# Suprascrierea funcțiilor

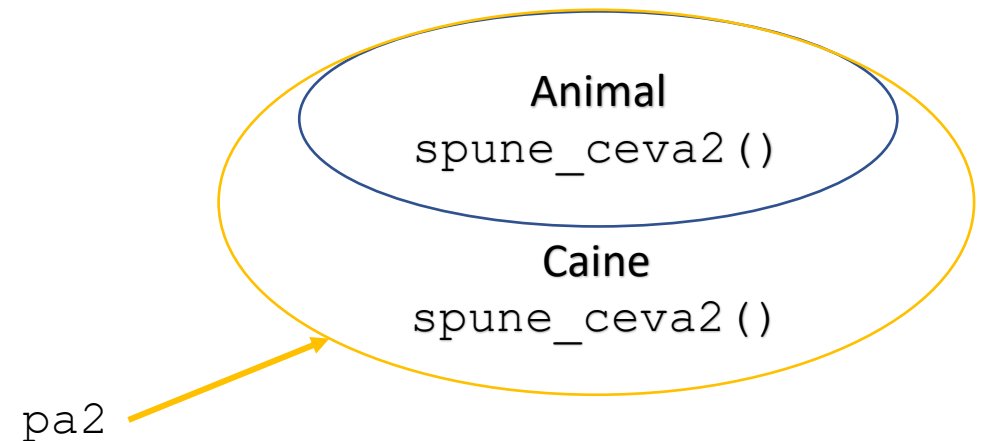
```
#include <iostream>
class Animal {
public:
    void spune_ceva() {std::cout << "Animal vorbitor" << std::endl;}
    → virtual void spune_ceva2() {std::cout << "[virtual] Animal vorbitor" << std::endl;}
};
class Caine : public Animal {
public:
    void spune_ceva() {std::cout << "Caine vorbitor" << std::endl;}
    → void spune_ceva2() {std::cout << "[virtual] Caine vorbitor" << std::endl;}
};
int main() {
    Animal* pa1 = new Animal();
    Caine* pc = new Caine();
    Animal* pa2 = new Caine();

    pa1->spune_ceva();
    pc->spune_ceva();
    pa2->spune_ceva();

    → { pa1->spune_ceva2();
        pc->spune_ceva2();
        pa2->spune_ceva2();
    }

    return 0;
}
```

03/12/21



# Suprascrierea funcțiilor

```
#include <iostream>
class Animal {
public:
    void spune_ceva() {std::cout << "Animal vorbitor" << std::endl;}
    → virtual void spune_ceva2() {std::cout << "[virtual] Animal vorbitor" << std::endl;}
};
class Caine : public Animal {
public:
    void spune_ceva() {std::cout << "Caine vorbitor" << std::endl;}
    → void spune_ceva2() {std::cout << "[virtual] Caine vorbitor" << std::endl;}
};
int main() {
    Animal* pa1 = new Animal();
    Caine* pc = new Caine();
    Animal* pa2 = new Caine();

    pa1->spune_ceva();
    pc->spune_ceva();
    pa2->spune_ceva();

    → { pa1->spune_ceva2();
        pc->spune_ceva2();
        pa2->spune_ceva2();
    }

    return 0;
}
```

03/12/21

```
Animal vorbitor
Caine vorbitor
Animal vorbitor
[virtual] Animal vorbitor
[virtual] Caine vorbitor
[virtual] Caine vorbitor

Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

# Suprascrierea funcțiilor

```
#include <iostream>
class Animal {
public:
    virtual void talk(){std::cout << "Animal vorbitor" << std::endl;}
};
class Caine : public Animal {
public:
    void talk(){std::cout << "Caine vorbitor" << std::endl;}
};
class Caine_maidanez : public Caine {
public:
    void talk(){std::cout << "Caine maidanez vorbitor" << std::endl;}
};

void fun(Animal *ptr) {ptr->talk();}

int main() {
    Animal* pa1 = new Animal();
    Animal* pa2 = new Caine();
    Animal* pa3 = new Caine_maidanez();

    fun(pa1);
    fun(pa2);
    fun(pa3);

    return 0;
}
```

Cuvântul cheie `virtual` este  
moștenit implicit în clasele derivate

```
Animal vorbitor
Caine vorbitor
Caine maidanez vorbitor

Process returned 0 (0x0)   execution time : 0.008 s
Press any key to continue.
```



# Suprascrierea funcțiilor – destructori virtuali

```
#include <iostream>
class Animal {
public:
    ~Animal() { std::cout << "Destructor Animal" << std::endl; }
};
class Caine : public Animal {
public:
    ~Caine() { std::cout << "Destructor Caine" << std::endl; }
};
class Caine_maidanez : public Caine {
public:
    ~Caine_maidanez() { std::cout << "Destructor Caine_maidanez" << std::endl; }
};

int main() {
    Animal* pa = new Animal;
    Animal* pc = new Caine;
    Animal* pcm = new Caine_maidanez;

    delete pa;
    delete pc;
    delete pcm;

    return 0;
}
```

Lipsa destructorilor virtuali poate cauza comportament nedefinit!

```
Destructor Animal
Destructor Animal
Destructor Animal

Process returned 0 (0x0)   execution time : 0.009 s
Press any key to continue.
```

# Suprascrierea funcțiilor – destructori virtuali

```
#include <iostream>
class Animal {
public:
    virtual ~Animal(){ std::cout << "Destructor Animal" << std::endl;}
};
class Caine : public Animal {
public:
    ~Caine(){ std::cout << "Destructor Caine" << std::endl;}
};
class Caine_maidanez : public Caine {
public:
    ~Caine_maidanez(){ std::cout << "Destructor Caine_maidanez" << std::endl;}
};

int main(){
    Animal* pa = new Animal;
    Animal* pc = new Caine;
    Animal* pcm = new Caine_maidanez;

    delete pa;
    delete pc;
    delete pcm;

    return 0;
}
```

```
Destructor Animal
Destructor Caine
Destructor Animal
Destructor Caine_maidanez
Destructor Caine
Destructor Animal

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

# Suprascrierea funcțiilor – funcții pur virtuale

def

Funcțiile *pur virtuale* sunt funcții virtuale care **trebuie** suprascrise în clasele derivate.

- O funcție virtuală devine **pur virtuală** dacă i se atribuie valoarea 0 după declarație.
- O funcție pur virtuală nu poate fi definită în continuare declarației (în interiorul clasei). Definiția ei (dacă există) trebuie să aibă loc în afara clasei.
- O clasă care conține cel puțin o funcție pur virtuală devine **abstractă**. Clasele abstracte nu pot fi instanțiate! Se mai numesc și **interfețe**.
- Dacă o clasă moștenește o clasă abstractă și nu suprascrie toate funcțiile pur virtuale, atunci devine și ea abstractă.

# Suprascrierea funcțiilor – funcții pur virtuale

```
#include <iostream>

class Baza { // clasa abstracta
public:
    virtual void foo() = 0; // functie pur virtuala
};

class Derivata: public Baza {}; // nu suprascrie functia pur virtuala din Baza => clasa abstracta

class Derivata_din_nou: public Derivata {
public:
    void foo() {std::cout << "Derivata_din_nou::foo()" << std::endl;}
};

int main(){
    Baza *d = new Derivata;
    d->foo();
    return 0;
}
```

```
18 error: invalid new-expression of abstract class type 'Derivata'
10 note: because the following virtual functions are pure within 'Derivata':
8 note: 'virtual void Baza::foo()'
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

# Suprascrierea funcțiilor – funcții pur virtuale

```
#include <iostream>

class Baza { // clasa abstracta
public:
    virtual void foo() = 0; // functie pur virtuala
};

class Derivata: public Baza {}; // nu suprascrie functia pur virtuala din Baza => clasa abstracta

class Derivata_din_nou: public Derivata {
public:
    void foo() {std::cout << "Derivata_din_nou::foo()" << std::endl;}
};

int main(){
    Baza *d = new Derivata_din_nou;
    d->foo();
    return 0;
}
```

```
Derivata_din_nou::foo()

Process returned 0 (0x0)   execution time : 0.027 s
Press any key to continue.
```

# Sfârșit laborator #4