POO & C++

Programmation Orientée Objet
&
Langage C++

UVSQ université PARIS-SACLAY
ISTY
Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

Olivier Snoeck
olivier@snoeck.fr

Rappel

On a vu que la POO est un paradigme de programmation avec :

- L’utilisation de classes
- L’utilisation d’attributs
- L’utilisation de constructeurs
- L’utilisation de destructeurs

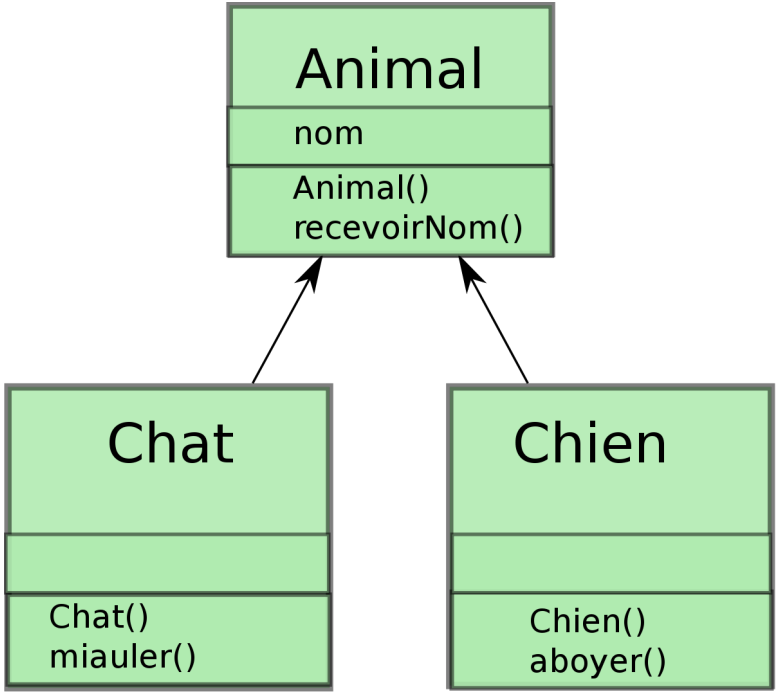
```cpp
#include <iostream>

class Apero
{
public:
    // Constructeur par défaut
    Apero()
    { std::cout << "L'heure de l'apero a sonne !" <<   std::endl; }
    // Destructeur
    ~Apero()
    { std::cout << "A table !" <<  std::endl; }
public:
    // Méthode publique
    void bis()
     {std::cout << "Encore un ? "<< std::endl; }
};

/*void Apero::bis()
{ std::cout << "Encore un ? "<< std::endl; }
*/
int main()
{
Apero bic;
std::cout  << " Super ! " <<  std::endl;
bic.bis();
std::cout << " Non merci " << std::endl;
return 0;
}
```

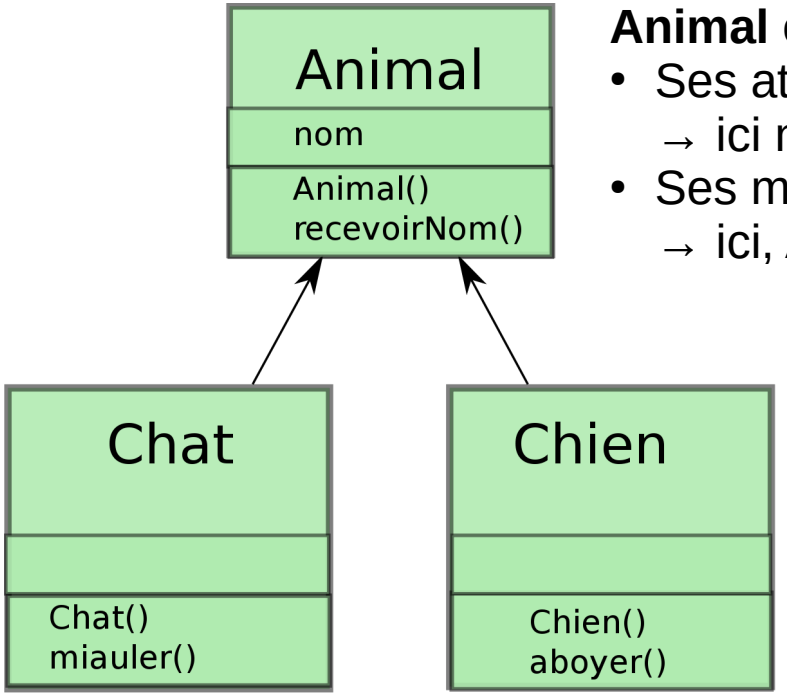
L’héritage

La notion d’héritage est la possibilité de réutiliser une classe



L’héritage

La notion d’héritage est la possibilité de réutiliser une classe

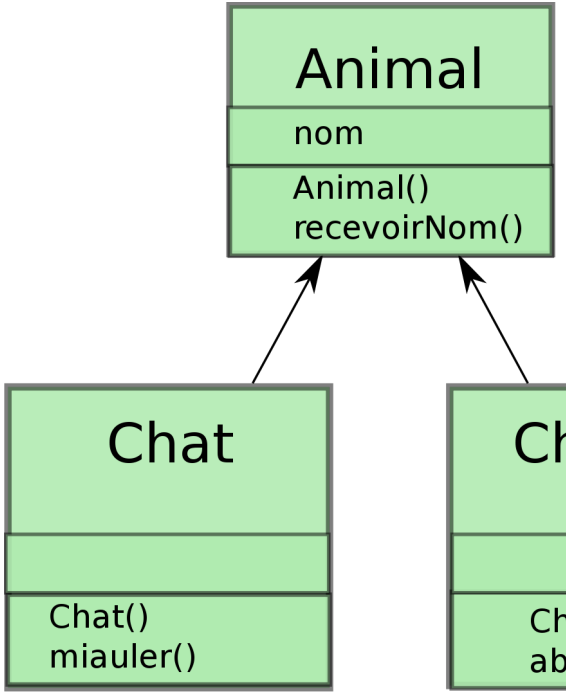


Animal est la super-classe avec :

- Ses attributs :
  → ici nom
- Ses méthodes :
  → ici, Animal() et recevoirNom()

L’héritage

La notion d’héritage est la possibilité de réutiliser une classe



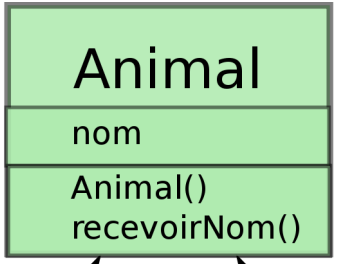
Animal est la <u>super-classe</u> avec :
- Ses attributs :
  → ici nom
- Ses méthodes :
  → ici, Animal() et recevoirNom()

Chat est une <u>sous-classe</u> avec :
- Ses attributs :
  → 0
- Ses méthodes :
  → ici, Chat() et miauler()

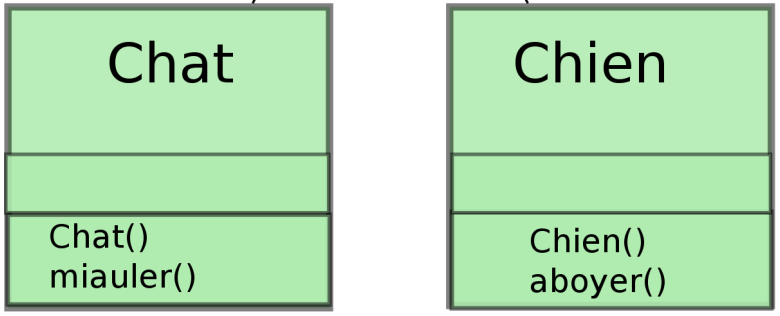
Chien est une <u>sous-classe</u> avec :
- Ses attributs :
  → 0
- Ses méthodes :
  → ici, Chien() et aboyer()

L’héritage

La notion d’héritage est la possibilité de réutiliser une classe



Animal est la super-classe avec :
- Ses attributs :
  → ici nom
- Ses méthodes :
  → ici, Animal() et recevoirNom()



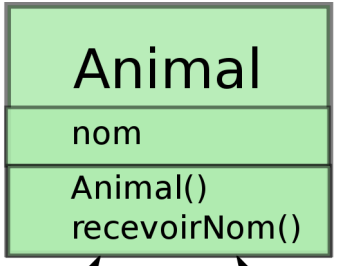
Chat est une sous-classe avec :
- Ses attributs :
  → 0
- Ses méthodes :
  → ici, Chat() et miauler()

Chien est une sous-classe avec :
- Ses attributs :
  → 0
- Ses méthodes :
  → ici, Chien() et aboyer()

L’héritage c’est, la classe **Chat** hérite de la classe **Animal** (attributs et méthodes)

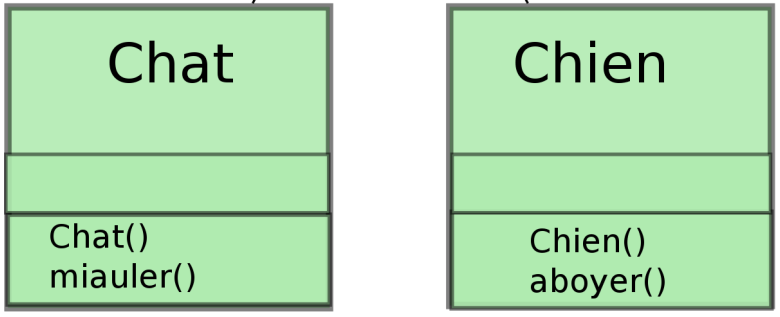
Vocabulaire :

- A est une **super-classe** de B et B est une **sous-classe** de A ;
- A est une **généralisation** de B et B est une **spécialisation** de A ;
- A est la **classe mère** de B et B est une **classe fille** de A.



Animal est la <u>super-classe</u> avec :

- Ses attributs :
  → ici nom
- Ses méthodes :
  → ici, Animal() et recevoirNom()



Chat est une <u>sous-classe</u> avec :

- Ses attributs :
  → 0
- Ses méthodes :
  → ici, Chat() et miauler()

Chien est une <u>sous-classe</u> avec :

- Ses attributs :
  → 0
- Ses méthodes :
  → ici, Chien() et aboyer()

L'héritage c'est, la classe **Chat** <u>hérite</u> de la classe **Animal** (attributs et méthodes)

Programmation

Pour faire un héritage en C++, il faut faire suivre le nom de la **classe fille** par la **classe mère** dans la déclaration avec les restrictions d'accès aux données, chaque élément étant séparé des autres par une virgule.
La syntaxe (donnée pour class, identique pour struct) est la suivante :

```
class Classe_mere1
{
    /* Contenu de la classe mère 1. */
};

class Classe_fille : public|protected|private Classe_mere1
{
    /* Définition de la classe fille. */
};
```

Droits d’accès sur les membres hérités

		Mots clé utilisé pour l’héritage		
Accès aux données		**public**	**protected**	**private**
Mots clé utilisé :	public	*public*	*protected*	*private*
Pour les champs	protected	*protected*	*protected*	*private*
Pour les méthodes	private	*Pas d’accès*	*Pas d’accès*	*Pas d’accès*

- Les données publiques d'une classe mère deviennent soit publiques, soit protégées, soit privées selon que la classe fille hérite en public, protégé ou en privé.
- Les données privées de la classe mère sont toujours inaccessibles, et les données protégées deviennent soit protégées, soit privées.

```
#include <iostream>
using namespace std;

// Classe de base
class Animal {
protected:
    string nom;

public:
    Animal(string n) : nom(n) {}

    void manger() {
        cout << nom << " mange." << endl;
    }

    void dormir() {
        cout << nom << " dort." << endl;
    }
};
```

https://wandbox.org/permlink/Dbl5Kc1cjN3zSa89

```cpp
#include <iostream>
using namespace std;

// Classe de base
class Animal {
protected:
    string nom;

public:
    Animal(string n) : nom(n) {}

    void manger() {
        cout << nom << " mange." << endl;
    }

    void dormir() {
        cout << nom << " dort." << endl;
    }
};

// Classe dérivée
class Chien : public Animal {
public:
    Chien(string n) : Animal(n) {}

    void aboyer() {
        cout << nom << " aboie." << endl;
    }
};

// Classe dérivée
class Chat : public Animal {
public:
    Chat(string n) : Animal(n) {}

    void miauler() {
        cout << nom << " miaule." << endl;
    }
};
```

https://wandbox.org/permlink/Dbl5Kc1cjN3zSa89

```cpp
#include <iostream>
using namespace std;

// Classe de base
class Animal {
protected:
    string nom;

public:
    Animal(string n) : nom(n) {}

    void manger() {
        cout << nom << " mange." << endl;
    }

    void dormir() {
        cout << nom << " dort." << endl;
    }
};

// Classe dérivée
class Chien : public Animal {
public:
    Chien(string n) : Animal(n) {}

    void aboyer() {
        cout << nom << " aboie." << endl;
    }
};

// Classe dérivée
class Chat : public Animal {
public:
    Chat(string n) : Animal(n) {}

    void miauler() {
        cout << nom << " miaule." << endl;
    }
};

int main() {
    Chien chien("Rex
    Chat chat("Mimi"

    chien.manger();
    chien.aboyer();

    chat.dormir();
    chat.miauler();

    return 0;
}
```

https://wandbox.org/permlink/Dbl5Kc1cjN3zSa89