Initiation aux concepts objets en C++

Du C au C++ - Découverte du paradigme objet

Florent DIEDLER

Sommaire

Sommaire (détail)

Objectifs

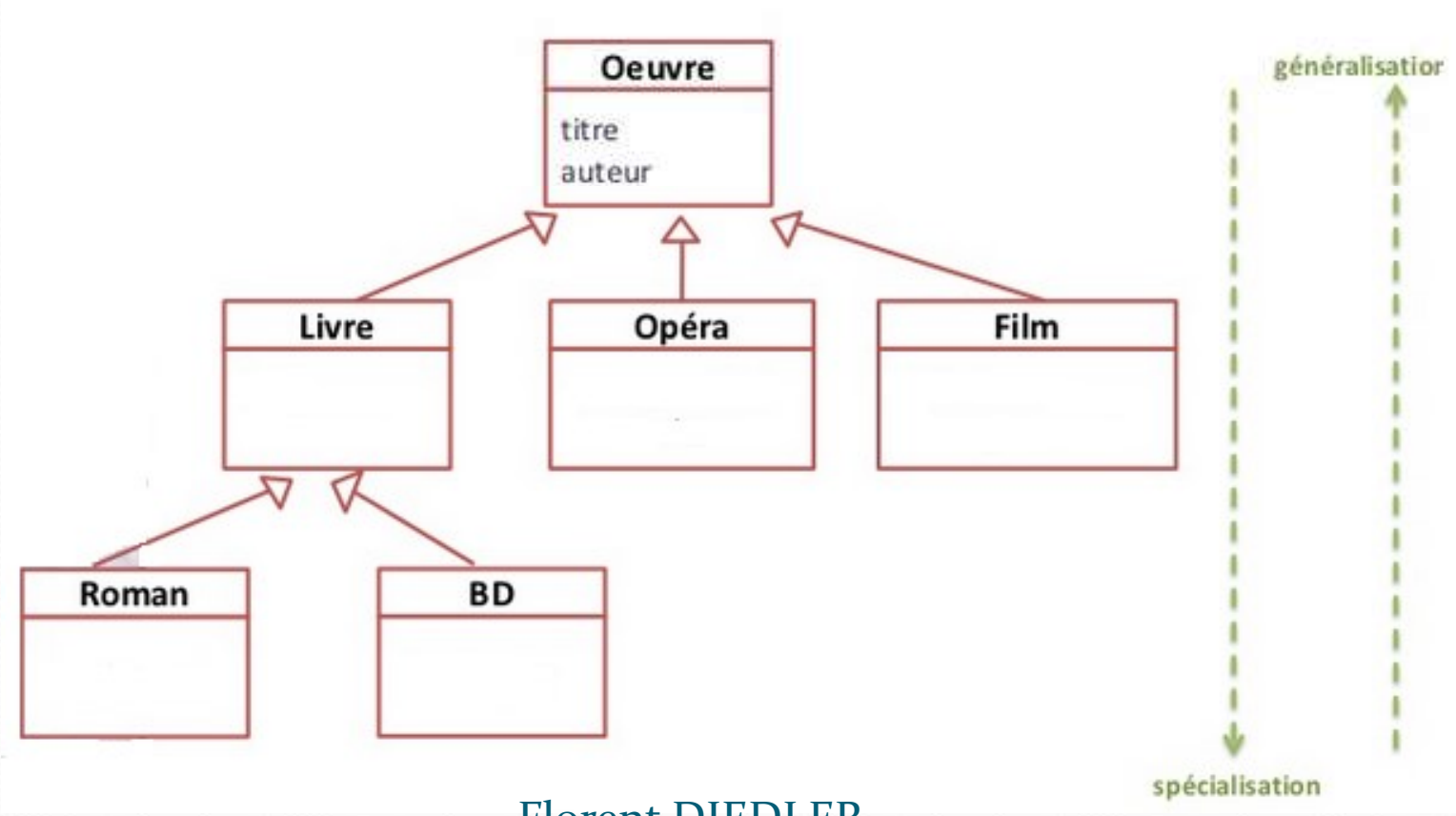
- Comprendre la notion d'héritage et de polymorphisme
- Comprendre l'intérêt des classes abstraites
- Implémenter ces notions en C++

Concepts et vocabulaire (1/3)

- Héritage = Relation de parenté entre deux classes
- Classe mère = classe de base (ou super-classe)
- Classe dérivée (fille) = classe spécialisée

Concepts et vocabulaire (2/3)

- Concept fondamental avec exemple :



Concepts et vocabulaire (3/3)

- **Classe dérivée (fille) hérite des attributs et méthodes de la classe mère**

- Avantages :
  - Factorisation du code
  - Réutilisabilité
  - Modularité
  - Création d'interface…

Héritage simple - Exemple (1/3)

- Classe « Person »
  - Nom, age
- Clase « Student »
  - Nom, age, promotion, notes
- Classe « Employee »
  - Nom age, salaire, compétences

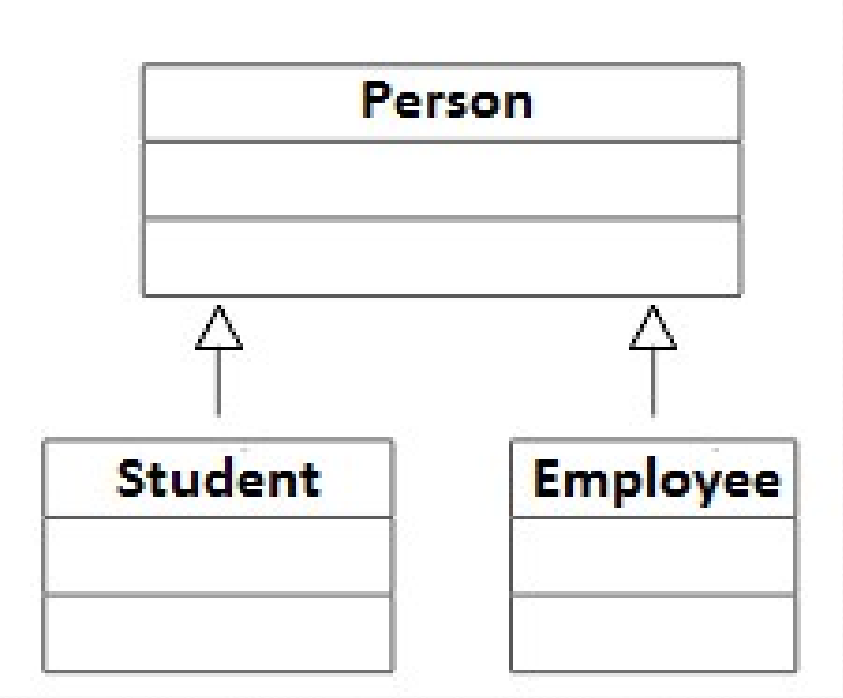
Héritage simple - Exemple (2/3)

- Plusieurs classes (min. 2 classes)
- Attributs « nom » et « age » en commun !
- Relation « est un »

RELATION D'HERITAGE

Héritage simple - Exemple (3/3)

- Considérons le diagramme de classe :



Héritage simple – C++ (1/9)

- Déclaration de la classe mère « Person » :

```
class Person
{
    private: // CANNOT be modified in derived classes
        int m_age;

    protected: // can be modified in derived classes
        std::string m_name;
        int m_birthdayYear;

    public:
        Person(std::string _name, int _birthdayYear);
        ~Person();

        void display() const;

        int getBirthdayYear() const;
        std::string getName() const;
};
```

Héritage simple – C++ (2/9)

- Implémentation de la classe mère « Person » :

```
Person::Person(std::string _name, int _birthdayYear)
    : m_name(_name), m_birthdayYear(_birthdayYear)
{
    m_age = 2015 - m_birthdayYear;
}

Person::~Person()
{
}

void Person::display() const
{
    std::cout << "Name = " << m_name << std::endl;
    std::cout << "Birthday year = " << m_birthdayYear << std::endl;
    std::cout << "Age = " << m_age << std::endl;
}

int Person::getBirthdayYear() const { return m_birthdayYear; }
std::string Person::getName() const{ return m_name; }
```

Héritage simple – C++ (3/9)

- Déclaration de la classe fille « Student »:

```
class Student : public Person
{
    private:
        int m_promotion;
        std::vector<float> m_grades;

    public:
        Student(std::string _name, int _birthdayYear, int _promotion);
        ~Student();

        void display() const;
        bool addGrade(float _grade);

        int getPromotion() const;
};
```

héritage publique

Héritage simple – C++ (4/9)

- Implémentation de la classe fille « Student »:

```
Student::Student(std::string _name, int _birthdayYear, int _prom
    : Person(_name, _birthdayYear), m_promotion(_promotion)
{
}

Student::~Student()
{
}

void Student::display() const
{
    // Call display() method from Person class
    // Display global informations about this student
    Person::display();

    // Display specific informations about this student
    std::cout << "Promotion = " << m_promotion << std::endl;
    std::cout << "************* GRADES *********" << std::endl;
    if (m_grades.size() == 0){
        std::cout << "No grade..." << std::endl;
    }
    else {
        for (const auto grade : m_grades) {
            std::cout << "Grade = " << grade << std::endl;
        }
    }
    std::cout << "******************************" << std::endl;
}
```

Héritage simple – C++ (5/9)

- Implémentation de la classe fille « Student » (suite)

```
bool Student::addGrade(float _grade)
{
    if (_grade >= 0 && _grade <= 20) {
        m_grades.push_back(_grade);
        return true; // ok
    }

    return false; // error...
}

int Student::getPromotion() const { return m_promotion; }
```

Héritage simple – C++ (6/9)

- Déclaration de la classe fille « Employee »:

```
class Employee : public Person
{
    private:
        int m_salary;
        std::vector<std::string> m_skills;

    public:
        Employee(std::string _name, int _birthdayYear, int _salary);
        ~Employee();

        void display() const;
        void addSkill(std::string _skill);

        int getSalary() const;
};
```

héritage publique

Héritage simple – C++ (7/9)

- Implémentation de la classe fille « Employee »:

```
Employee::Employee(std::string _name, int _birthdayYear, int _salary)
    : Person(_name, _birthdayYear), m_salary(_salary)
{
}

Employee::~Employee()
{
}

void Employee::display() const
{
    // Call display() method from Person class
    // Display global informations about this student
    Person::display();

    // Display specific informations about this employee
    std::cout << "Salary = " << m_salary << " euros" << std::endl;
    std::cout << "************* SKILLS *********" << std::endl;
    if (m_skills.size() == 0) {
        std::cout << "No skills..." << std::endl;
    }
    else {
        for (const auto skill : m_skills) {
            std::cout << "Skill : " << skill << std::endl;
        }
    }
    std::cout << "******************************" << std::endl;
}
```

Héritage simple – C++ (8/9)

- Implémentation de la classe fille « Employee » (suite)

```
void Employee::addSkill(std::string _skill)
{
    m_skills.push_back(_skill);
}

int Employee::getSalary() const
{
    return m_salary;
}
```

Héritage simple – C++ (9/9)

- Utilisation des classes :

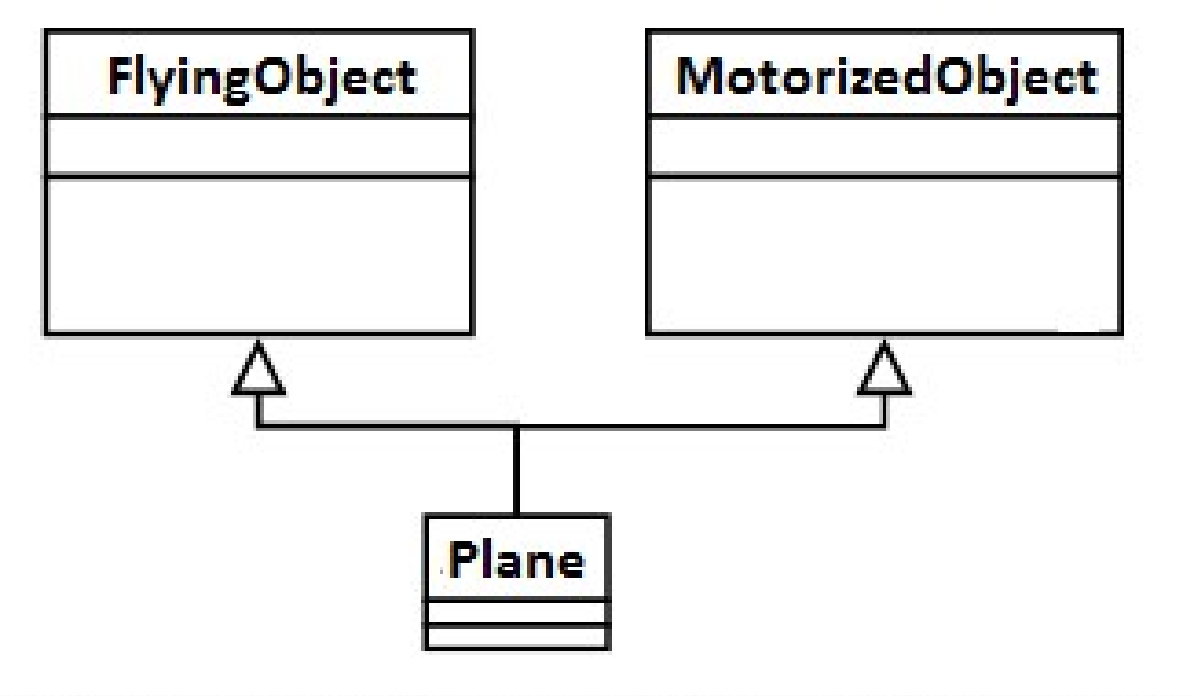
```
Student s("DIEDLER", 1988, 2013);
if (!s.addGrade(5.5))
    std::cout << "Error adding grade..." << std::endl;
if (!s.addGrade(15.0))
    std::cout << "Error adding grade..." << std::endl;
if (!s.addGrade(-12.0))
    std::cout << "Error adding grade..." << std::endl;
s.display();
std::cout << std::endl;

Employee e("DUPONT", 1968, 2450);
e.addSkill("C++");
e.addSkill("Management");
e.display();
std::cout << std::endl;

Person p = e; // ok because inheritance
p.display();
```

Héritage multiple (1/4)

- Classe fille héritant de plusieurs classes mères
- Hérite toujours des attributs et méthodes
- Notion existant dans quelques langages de POO

Héritage multiple (2/4)

- Considérons l'exemple suivant :



Héritage multiple (3/4)

- Concrètement :

```
class FlyingObject()
{
    private:
        int m_nbWings;
    public:
        FlyingObject();
        ~FlyingObject();
};
```

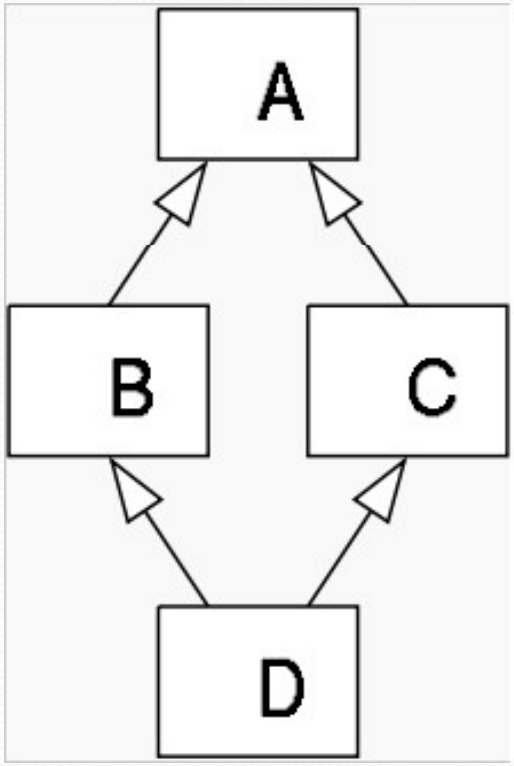
```
class MotorizedObject()
{
    private:
        int m_nbWheels;
    public:
        MotorizedObject();
        ~MotorizedObject();
};
```

```
class Plane
    : public FlyingObject, public MotorizedObject
{
    public:
        Plane();
        ~Plane();
};
```

Héritage multiple !

Héritage multiple (4/4)

- Problèmes majeurs :
  - Héritage en diamant
  - Ambiguïté sur le nom des méthodes
  - Constructeurs…
- Solution : Interdire l'héritage multiple
  - Exemple : Java



Polymorphisme

- Notion difficile à comprendre au début
- Examinons un exemple pour mieux comprendre

Polymorphisme – Problème (1/6)

- Soit la classe de base « Animal »

```
class Animal
{
    private:
        std::string m_name = "Default";
        int m_color = 0;
        int m_height = 140;

    public:
        Animal();
        ~Animal();

        void manger() const;
        void display() const;
};
```

Polymorphisme – Problème (2/6)

- Et son implémentation :

```
Animal::Animal()  { /* Init members here */ }
Animal::~Animal() { /* free memory here */ }

void Animal::manger() const
{
    std::cout << "Animal eats..." << std::endl;
}

void Animal::display() const
{
    std::cout << "I am an animal ! " << std::endl;
}
```

Polymorphisme – Problème (3/6)

- Et les classes dérivées :

```
class Lion : public Animal
{
    public:
        Lion();
        ~Lion();

        void display() const;
};

class Bird : public Animal
{
    public:
        Bird();
        ~Bird();

        void display() const;
};
```

Polymorphisme – Problème (4/6)

- Et les classes dérivées :

```
Lion::Lion() : Animal() { /* Init specific members here */}
Lion::~Lion() { /* Free specific memory here */}
void Lion::display() const
{
    std::cout << "I am a lion !" << std::endl;
}

Bird::Bird() : Animal() { /* Init specific members here */}
Bird::~Bird() { /* Free specific memory here */}
void Bird::display() const
{
    std::cout << "I am a bird !" << std::endl;
}
```

Polymorphisme – Problème (5/6)

- Test 1 :

```
Animal a;
Bird b;

/*******************************************
* First test : Display info about animals  *
*******************************************/
a.display();
b.display();
/*******************************************
* it works, we have correct output :)      *
*******************************************/
```

- Sortie standard (sur console) :

I am an animal !
I am a bird !

Polymorphisme – Problème (6/6)

- Test 2 :

```
// Function that display informations about an animal
void display_animal(Animal  a) { a.display(); }

void usePolymorph()
{
    Animal a;
    Bird b;

    /*****************************************
    * Second test : Display info about animals *
    *****************************************/
    display_animal(a);
    display_animal(b);
}
```

- Sortie standard (sur console) :

I am an animal !

I am an animal ! ← *PAS CORRECT*

Polymorphisme – Analyse

- La nature de l'objet « b » de type « Bird » est perdue…
- Serais-ce un bug ?
  - Non – Héritage ==> *b* « est un » animal…
- Peut-on régler le problème ?
  - Oui – Polymorphisme !

Polymorphisme – Corrections (1/2)

- Méthode « display » de la classe mère devient virtuelle
- Fonction « display_animal » prend un pointeur ou une référence en paramètre
- Le destructeur devient virtuel !!

Polymorphisme – Corrections (2/2)

```
// Function that display informations about an animal
void display_animal(Animal& a) { a.display(); }
```

Référence (ou pointeur) !!

```
void usePolymorph()
{
    Animal a;
    Bird b;

    /*******************************************
    * Second test : Display info about animals *
    *******************************************/
    display_animal(a);
    display_animal(b);
}
```

```
class Animal
{
    private:
        std::string m_name = "Default";
        int m_color = 0;
        int m_height = 140;

    public:
        Animal();
        virtual ~Animal();
```

Destructeur virtuel !

```
        void eat() const;
        virtual void display() const;
};
```

Fonction display() virtuelle

Classe abstraite (1/3)

- Définition : Classe incomplète non instanciable !

- Propriété :
  - Possède au minimum une méthode virtuelle pure
  - Doit être héritée

- Exemple :

```
class AbstractAnimal
{
    public:
        AbstractAnimal();
        ~AbstractAnimal();

        // Abstract method
        virtual void type() = 0;
};
```

Classe abstraite (2/3)

- Doit être « dérivée » pour pouvoir être utilisée

```
class NewBird : public AbstractAnimal
{
    public:
        NewBird();
        ~NewBird();

        // Redefinition of "type" (abstract function)
        void type()
        {
            std::cout << "TYPE = Bird" << std::endl;
        }
};
```

- Classes dérivées doivent redéfinir les méthodes virtuelles pures

Classe abstraite (3/3)

- Utilisation d'une classe abstraite :

```
void useAbstractClass()
{
    // Not permitted because AbstractAnimal is abstract !!
    //AbstractAnimal a;
    //a.type();

    // But this lines are allowed
    AbstractAnimal* a;
    a = new NewBird();
    a->type();
    delete a;
}
```

Pour résumer (1/2)

- Héritage simple :
  - Classe fille dérive d'une classe mère et hérite de tous ses attributs (s'ils sont « protected ») et méthodes
  - Permet de factoriser le code
- Héritage multiple :
  - Difficile à mettre en place correctement
  - Difficile à maintenir

Pour résumer (2/2)

- Polymorphisme :
  - Modifier le comportement d'un objet en fonction de son type (objets polymorphes)
  - Permet encore de factoriser le code

- Classe abstraite :
  - Définir les grandes lignes du comportement d'une classe
  - Permet encore de factoriser le code ☺

Pour aller plus loin…

- Introduction aux DP (Design Pattern)
  - http://ericreboisson.developpez.com/livres/developpement/java/design/patterns/
- Amitié en C++
  - http://cpp.developpez.com/faq/cpp/?page=Les-amis-friend
- La REFERENCE en C++ :
  - http://www.cplusplus.com/reference/

Merci de votre attention !