

Modéliser avec le langage UML

Diagramme de Classes

Laure Gérard

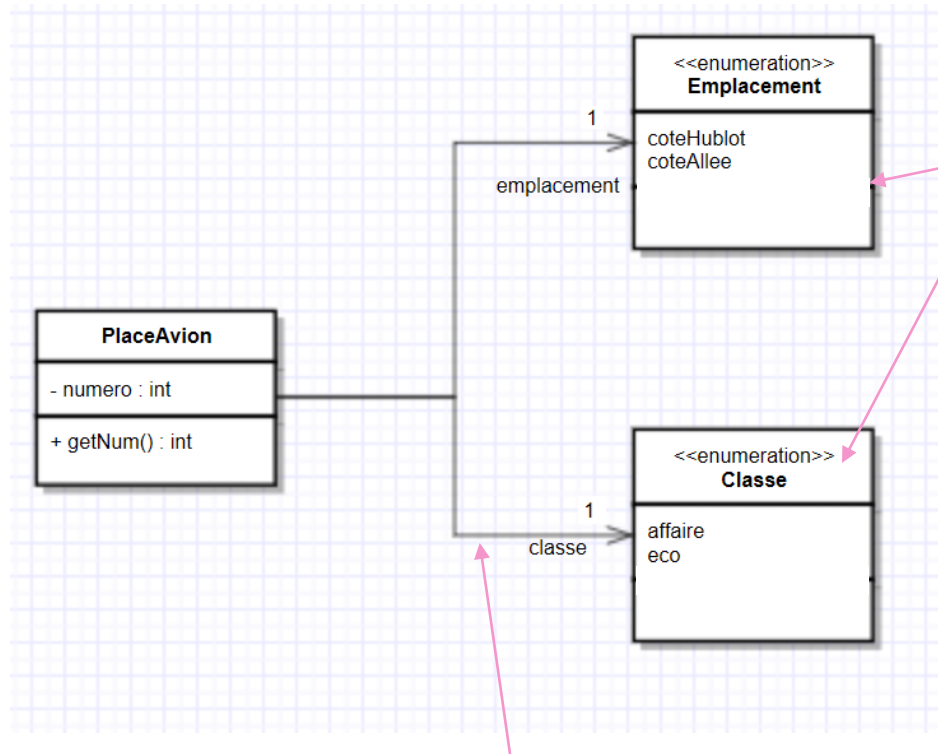
laure.gerard@grenoble-epsi.fr

Enumération

Comment modéliser un attribut de classes dont les valeurs possibles sont limitées?

- ▶ Pour modéliser la localisation d'une place assise dans un avion deux informations doivent être prises en compte
 - ▶ Emplacement : coté hublot ou coté allée
 - ▶ Classe : Affaire et Eco
- ▶ On va définir deux énumérations qui seront considérées comme deux types de données pour les attributs emplacement et confort.

Comment modéliser un attribut de classes dont les valeurs possibles sont limitées?



Représentée comme une classe, une énumération est reconnaissable par le stéréotype «enumeration». Elle ne contient que des attributs dont le nom correspond à chacune des valeurs possible.

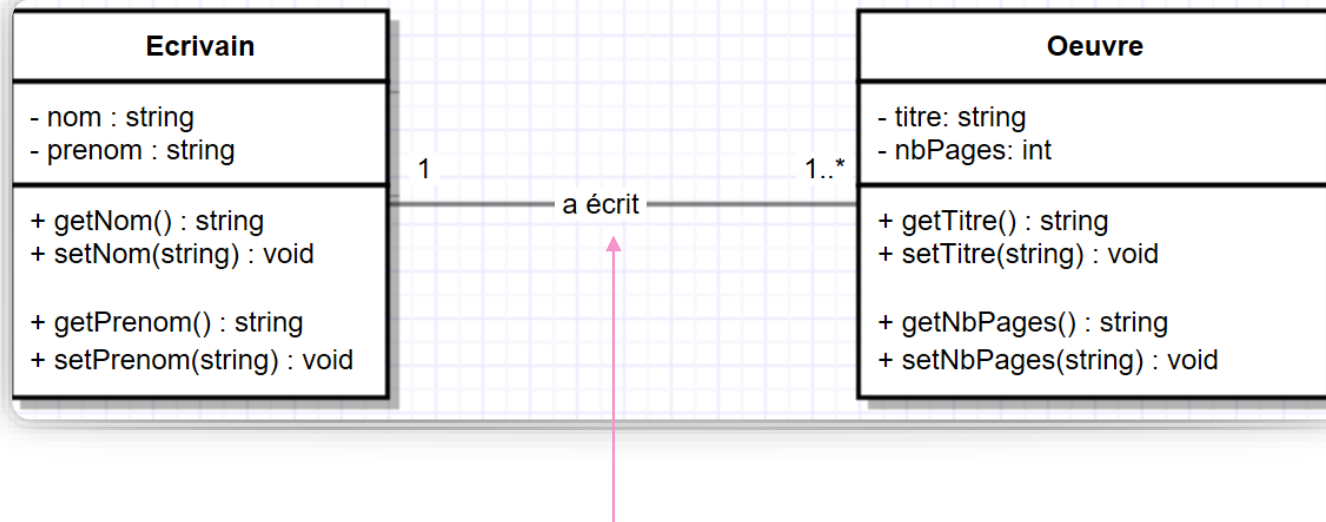
Reliée à la classe via une association orientée vers l'énumération qui peut être compléter par une multiplicité et un rôle.

Exercice 3 : Calendrier

- ▶ Une date est caractérisée par une
 - ▶ Journée : lundi, mardi, mercredi, jeudi, vendredi, samedi ou dimanche
 - ▶ Numéro (entier ≤ 31)
 - ▶ Mois : janvier, février, mars, avril, mai, juin, juillet, août, septembre, octobre, novembre, décembre.
 - ▶ Année (entier)
- ▶ Modéliser le concept de date.

Comment développer à partir
d'un diagramme UML?

L'association



Une association est une relation symétrique.

⇒ Un attribut de type `List<Œuvre>` est défini dans la classe **Ecrivain** représentant l'ensemble de ces œuvres.

⇒ Un attribut de type **Ecrivain** est défini dans la classe **Œuvre** représentant l'auteur de cette œuvre.

```

class Oeuvre
{
public:
    Oeuvre();
    ~Oeuvre();

    string getTitre();
    string getNbPages();
    Ecrivain getAuteur();

    void setTitre(string titre);
    void setNbPages(int nbPages);
    void setAuteur(Ecrivain auteur);

private :
    string m_titre;
    int m_nbPages;
    Ecrivain m_auteur;
};

```

Une œuvre a un auteur.

L'association

```

class Ecrivain
{
public:
    Ecrivain(string nom, string prenom);
    ~Ecrivain();

    string getNom();
    string getPrenom();
    vector<Oeuvre> getOeuvres();

    void setNom(string nom);
    void setPrenom(string prenom);

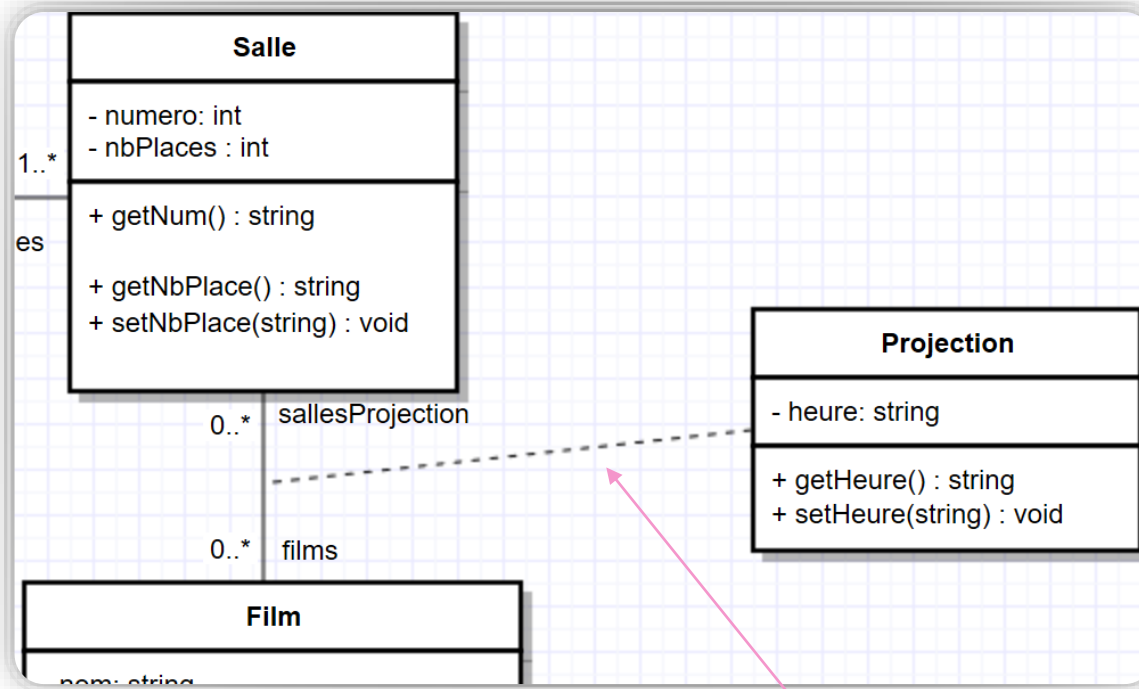
    void addOeuvre(Oeuvre oeuvre);

private :
    string m_nom;
    string m_prenom;
    vector<Oeuvre> m_listeOeuvres;
};

```

Un auteur a une liste d'œuvres.

La classe d'association



Une classe d'association apporte des informations supplémentaires sur la relation entre une Salle et un Film.

La classe d'association


```
class Projection
{
public:
    Projection(Film film, Salle salle, string date);
    ~Projection();

    Film getFilm();
    Salle getSalle();
    string getHeure();

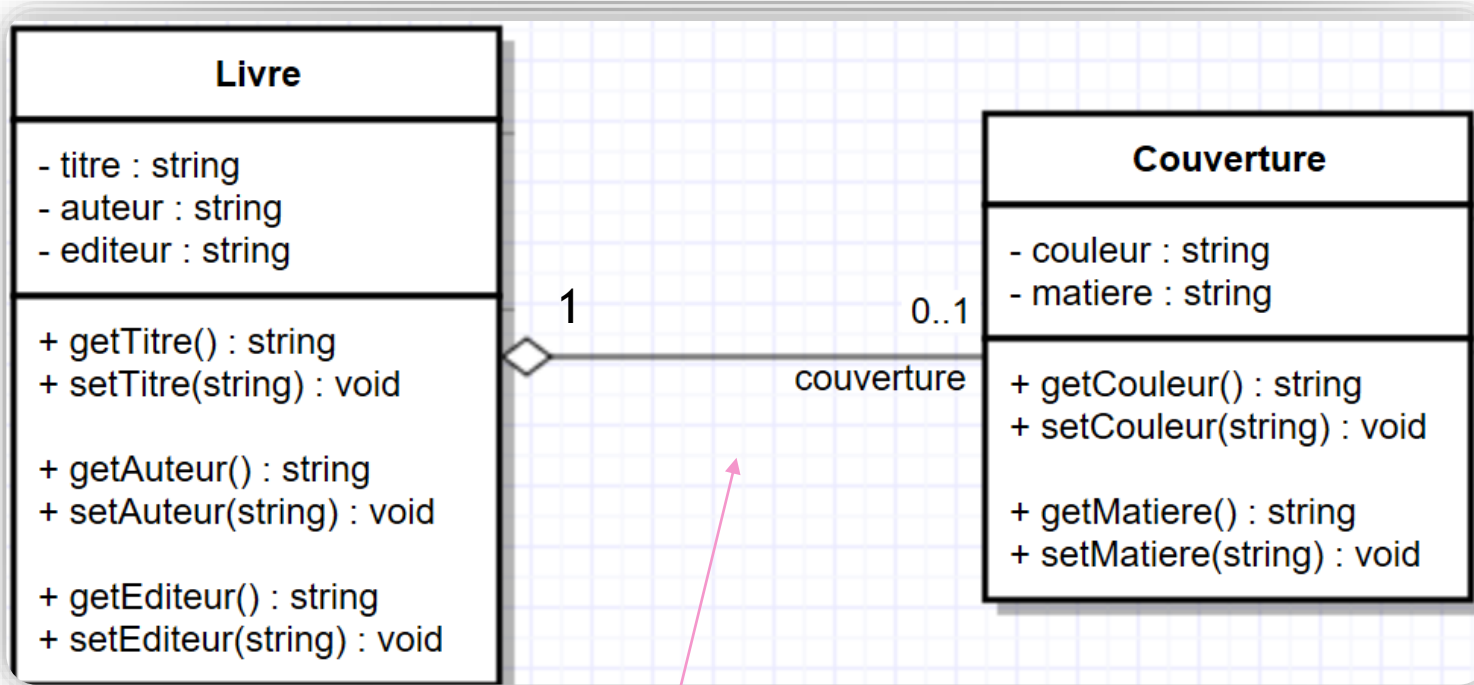
    void setHeure(string heure);

private :
    Film m_film;
    Salle m_salle;
    string m_heure;
};
```

Pour représenter cette relation, on ajoute un attribut de type Salle et un attribut de type Film.



L'agrégation



Attribut couverture à ajouter dans la classe
Livre

```
private Couverture couverture;
```

L'agrégation

Ajout des fonctionnalités permettant de manipuler cet attribut :

```
// Constructeur
public Livre(Couverture couverture)
{
    this.auteur = " ";
    this.editeur = " ";
    this.titre = " ";
    this.couverture = couverture;
}

// Méthodes de classe
public Couverture GetCouverture () { return this.couverture; }
public void SetCouverture(Couverture c) { this.couverture = c; }
```

Les cycles de vie n'étant pas liés, l'objet Couverture doit être instancié avant d'être associé à l'objet Livre.

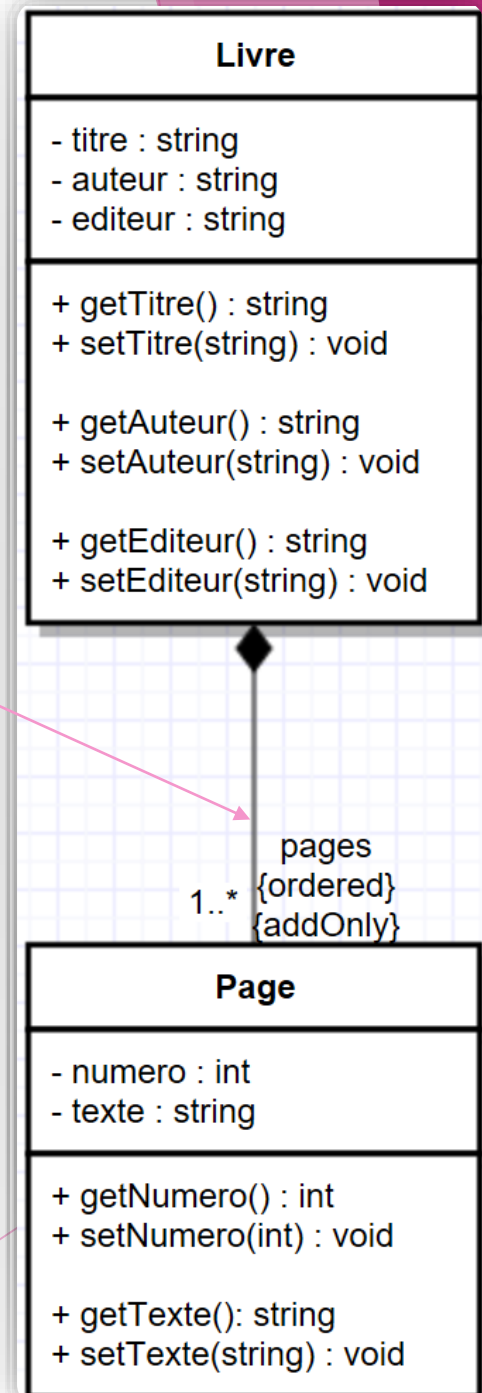
La composition

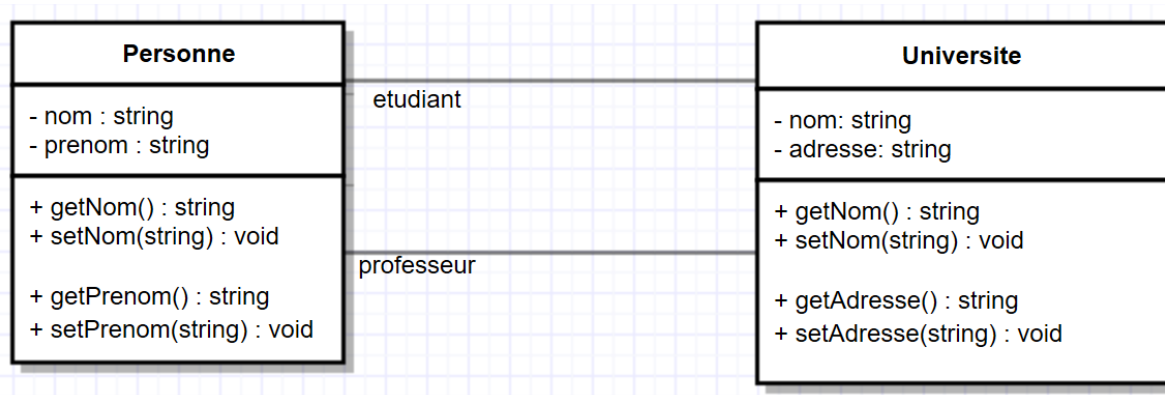
Attribut pages de type tableau de Page à ajouter dans la classe Livre.

Dans le constructeur de Livre :
Initialisation de la liste de pages
contenant un objet Page.

Ajout des fonctionnalités permettant de
manipuler cet attribut :

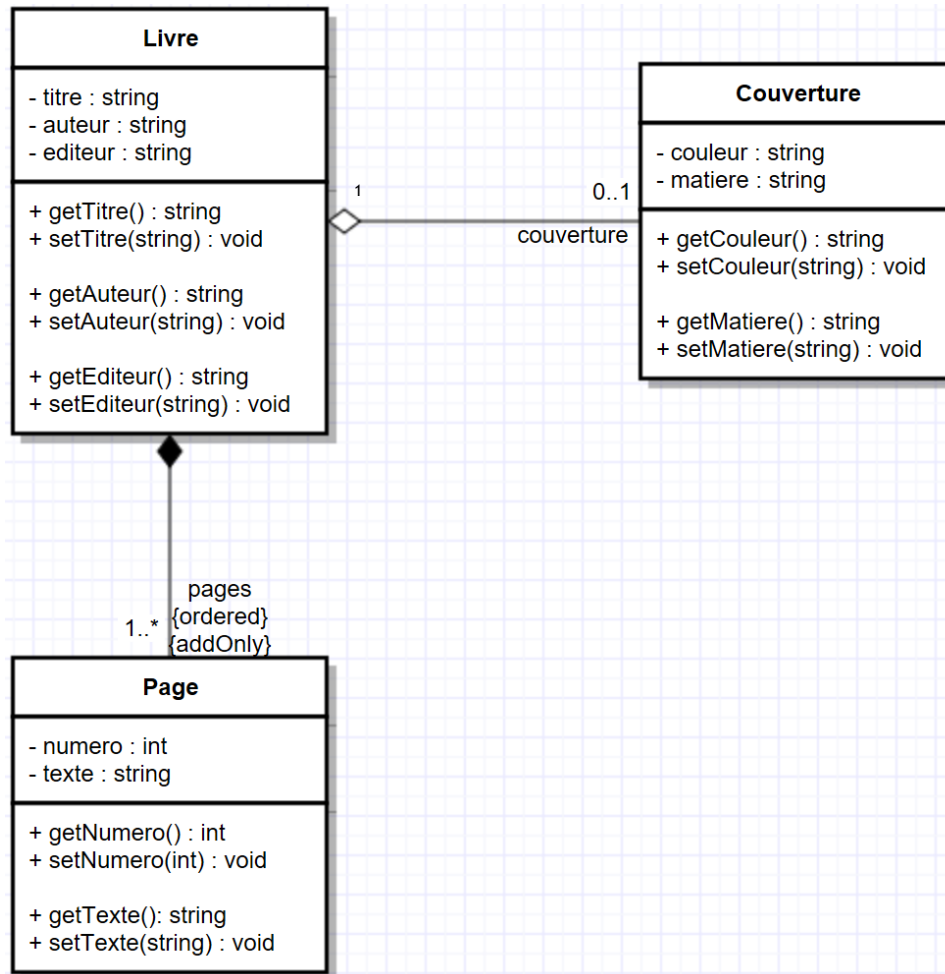
```
vector<Page> getPages();  
void addPage(string texte);
```





Exercice

Développer le modèle ci-dessus.



Exercice

Développer le modèle ci-dessus.

Reverse Engineering

Reverse-engineering ou rétro-ingénierie

- ▶ Etude et analyse d'un système pour en déduire son fonctionnement interne.
- ▶ Objectifs
 - ▶ Comprendre en détail le fonctionnement d'un logiciel.
 - ▶ Améliorer la sécurité et la qualité d'un logiciel.
- ▶ Deux étapes
 - ▶ Identifier les différentes classes, leurs attributs et leurs méthodes.
 - ▶ Identifier et modéliser les relations.

Exercice 1

- ▶ Réaliser le diagramme de classes correspondant aux différentes classes :
 - ▶ Coup
 - ▶ Joueur
 - ▶ Jeu

```
#ifndef COUP_H
#define COUP_H
class Coup
{
public:
    Coup(int choix, char symbole);
    ~Coup();

    int getChoix();
    char getSymbole();

private :
    // numero de la case
    int m_choix;
    // symbole à afficher
    char m_symbole;
};
#endif
```

classe Coup

```

- #ifndef JOUEUR_H
- #define JOUEUR_H

- #include <iostream>
- #include <string>
- #include <vector>
- using namespace std;

- #include "Coup.h"
- class Coup;

- class Joueur
- {
- public:
-     Joueur(int numero);
-     ~Joueur();

-     string getNom();
-     void setNom(string nom);

-     int getNumero();

-     Coup* jouer();
-     void afficherCoups();

- private :
-     string m_nom;

-     int m_numero;
-     char m_symbole;

-     vector<Coup*> m_coups;
- };

- #endif

```

Constructeur dans le fichier .cpp

```

- Joueur::~Joueur()
- {
-     // libération en mémoire de chaque coup du joueur
-     for (int i = 0; i < m_coups.size(); i++) {
-         m_coups[i]->~Coup();
-     }
- }

```

Classe Joueur

```

-#ifndef JEU_H
-#define JEU_H
-
-#include "Coup.h"
-#include "Joueur.h"
-
-#include <iostream>
-#include <vector>
-using namespace std;
-
-class Jeu
-
-{|
-public:
-    Jeu();
-    ~Jeu();
-
-    void lancerJeu();
-    void afficherPlateau();
-    void modifierPlateau(Coup* coup);
-    Joueur* getJoueur1();
-    Joueur* getJoueur2();
-    int validerPlateau();
-
-private :
-    Joueur* m_joueur1;
-    Joueur* m_joueur2;
-    // Valeurs des cases à afficher
-    vector<char> m_valeurs;
-};
-#endif

```

Destructeur dans le .cpp

```

-::~Jeu()
-
-    system("cls");// efface la console
-    m_joueur1->~Joueur();
-    m_joueur2->~Joueur();
-

```

Classe Jeu

Exercice 2

- ▶ Réaliser le diagramme de classes correspondant aux différentes classes :
 - ▶ Point
 - ▶ Segment
 - ▶ Triangle
 - ▶ Cercle
 - ▶ Rectangle
 - ▶ Losange

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

    float getX();
    void setX(float x);

    float getY();
    void setY(float y);

private:
    float m_x;
    float m_y;
};
```

Classe Point

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

    Point* getSommet1();
    void setSommet1(Point* sommet1);

    Point* getSommet2();
    void setSommet2(Point* sommet2);

    float getLongueur();
    void setLongueur(float longueur);

private :
    Point* m_sommet1;
    Point* m_sommet2;

    float m_longueur;
};
```

Classe Segment


```

class Triangle
{
public:
    Triangle();
    ~Triangle();

    void dessiner();

    vector<Point*> getSommets();
    void setSommets(vector<Point*> sommets);

    vector<Segment*> getCotes();
    void setCotes(vector<Segment*> segments);

    int getMaxSommets();
    int getMaxCotes();

private:
    vector<Point*> m_sommets;
    vector<Segment*> m_cotes;

    int m_nbMaxSommets;
    int m_nbMaxCotes;

};

```

Destructeur dans le .cpp

```

Triangle::~Triangle()
{
    // destruction des cotés du Triangle
    for (unsigned int i = 0; i < m_nbMaxCotes; ++i)
    {
        m_cotes[i]->~Segment();
    }
    // destruction des sommets du Triangle
    for (unsigned int i = 0; i < m_nbMaxSommets; ++i)
    {
        m_sommets[i]->~Point();
    }
}

```

Classe Triangle

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

    void dessiner();

    vector<Point*> getSommets();
    void setSommets(vector<Point*> sommets);

    vector<Segment*> getCotes();
    void setCotes(vector<Segment*> segments);

    int getMaxSommets();
    int getMaxCotes();

private :
    vector<Point*> m_sommets;
    vector<Segment*> m_cotes;

    int m_nbMaxSommets;
    int m_nbMaxCotes;

};
```

Classe Rectangle

```

class Losange
{
public:
    Losange();
    ~Losange();

    void dessiner();

    vector<Point*> getSommets();
    void setSommets(vector<Point*> sommets);

    vector<Segment*> getCotes();
    void setCotes(vector<Segment*> segments);

    int getMaxSommets();
    int getMaxCotes();

private:
    vector<Point*> m_sommets;
    vector<Segment*> m_cotes;

    int m_nbMaxSommets;
    int m_nbMaxCotes;

};

```

Classe Losange

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

    Point* getCentre();
    void setCentre(Point* centre);

    int getRayon();
    void setRayon(int rayon);

    void dessiner();

private :
    Point* m_centre;
    int m_rayon;
};
```

Classe Cercle

Exercice 3 : Nouvelle modélisation

- Proposer une nouvelle modélisation pour l'exercice 2.

