Programmation orientée objet (POO)

Programmation orientée objet

- En programmation il est commun de créer et de manipuler des **briques** logicielles appelées « *objets* ».

- Un objet représente une idée, un **concept** (un animal, un vecteur, une chaine de caractère, …)

- Il possède une **structure interne** et un **comportement**. De plus, il sait interagir avec d'autres objets.

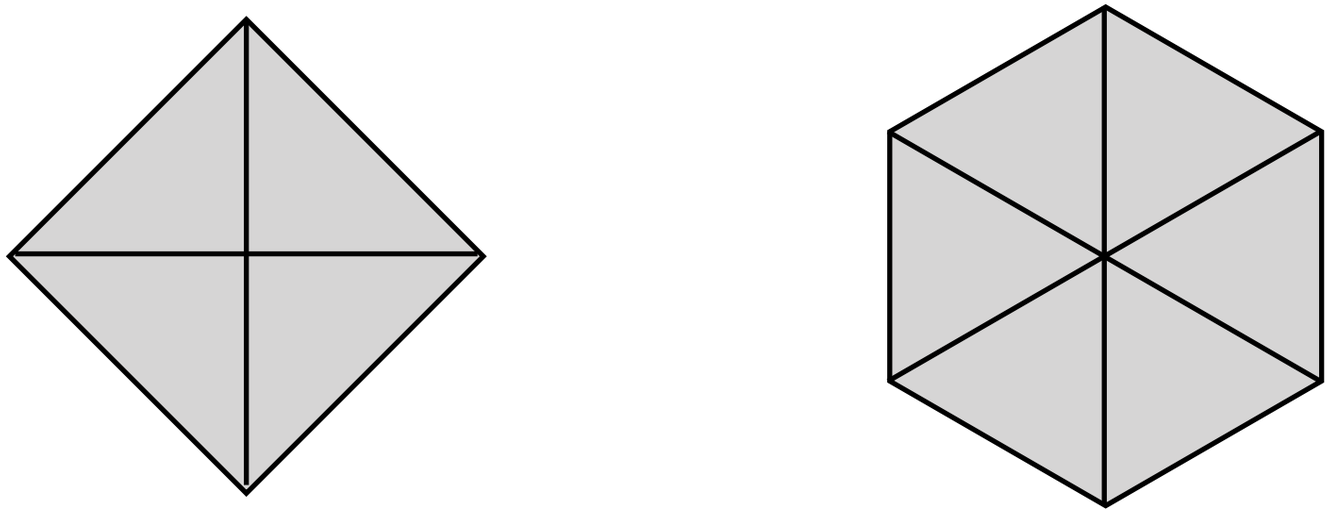
- La structure et le comportement d'un objet sont décrits en dehors du code principal dans une « *classe* ».

- On dira qu'un objet est une « *instance* » (un réalisation) d'une certaine classe.

Exemple

- Imaginez que vous voulez écrire un code calculant l'aire de la surface d'un polygone à n cotés.

- Tout polygone peut être découpé un plusieurs triangles adjacents partageant tous en même sommet.



- Le surface totale peut donc être calculée en sommant la surface de ces n triangles adjacents.

L'objet triangle

- Dans l'esprit de la POO, nous allons dès lors définir la classe « *triangle* ».

- Pour définir un triangle il nous suffira de connaitre ses trois sommets notés ici A, B et C.

- Sur base des sommets, nous pourrons alors calculer les longueurs a, b et c des cotés.

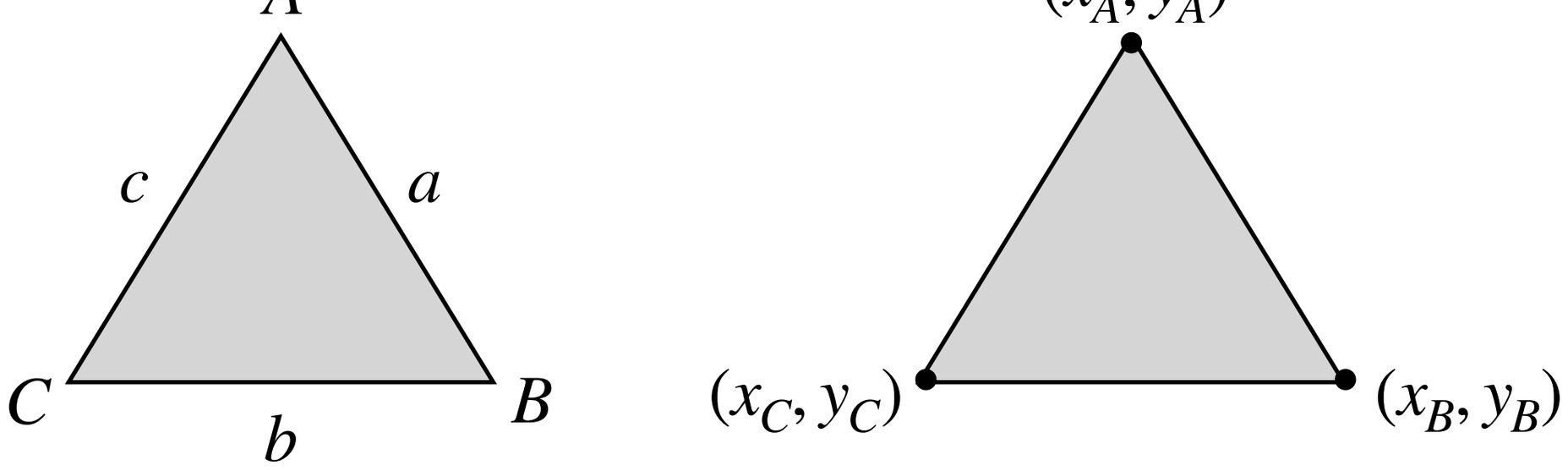
- La formule de Héron nous donne alors l'expression pour l'aire de la surface.

$$A = \sqrt{p(p-a)(p-b)(p-c)}, \quad p = \frac{a+b+c}{2}$$

L’objet point

- Les sommets d’un triangle sont des points dans le plan. Nous allons donc définir une classe « *point* ».

- Un point sera entièrement définit par ses coordonnées dans le plan, soit deux valeurs réelles x et y.



- Voyons maintenant comment incorporer des classes dans un code cpp.

Les classes en cpp

- Une classe cpp est généralement composée de deux fichiers compagnons.

- Un fichier **.** h, appelé « *header* », utilisé pour **déclarer** la structure interne et le comportement.

- Un fichier **.** cpp, utilisé pour **détailler** cette structure et interne et le comportement.

- Les éléments faisant référence à la structure interne sont appelés « *attributs* ».

- Les éléments faisant référence au comportement sont appelés « *méthodes* ».

Attributs et méthodes

- Les attributs d'une classe sont généralement **privés**, i.e. ils ne sont pas accessibles en dehors de la classe.

- Pour accéder aux valeurs des attributs, il faut passer par une méthode (appelée « *accesseur* » ou « *getter* »).

- Par défaut, il existe des méthodes appelées **constructeur** et **destructeur**.

- Le constructeur est appelé automatiquement quand un objet est **créé**.

- Le destructeur est appelé automatiquement quand un objet est **détruit**.

Exemple

- La classe « *point* » est composée des fichiers `point.h` et `point.cpp`
- Les attributs d'un point sont les `doubles` `m_x` et `m_y` codant ses coordonnées.
- Les méthodes d'un point comprennent le *constructeur* `point`, le *destructeur* `~point`,
- la méthode *d'initialisation* `init()` utilisée pour assigner des valeurs à `m_x` et `m_y`,
- et les méthodes de type *accesseurs* `x()` et `y()` rendant les valeurs de `m_x` et `m_y`.

Exemple

- Voici le contenu du fichier `point.h`

```
class point
{

// attributs de la classe point
private:
    double m_x; // coordonnée x du point
    double m_y; // coordonnée y du point

// méthodes de la classe point
public:
    point();                          // constructeur
    ~point();                         // destructeur
    void init(double,double);         // méthode d'initialisation
    double x();                       // méthode accesseur de m_x
    double y();                       // méthode accesseur de m_y
};
```

Exemple

- Voici le debut du fichier `point.cpp`

```cpp
#include "point.h"
#include <math.h>

point::point()
{
  //constructeur
}

point::~point()
{
  //destructeur
}
```

- **Remarque**: Pour l'instant, ne vous tracassez pas trop pour ces méthodes.

Exemple

- Voici la suite du fichier `point.cpp`

```
void point::init(double i_x,double i_y)
{
    m_x = i_x;
    m_y = i_y;
}

double point::x()
{
   return m_x;
}

double point::y()
{
   return m_y;
}
```

Appeler une méthode

- Pour appeler une méthode il faut utiliser l'opérateur « . »

```
#include "point.h"

int main(int argc, char **argv)
{
   double rayon = 1.;
   double alpha = 0.;

   // trois objets de type point
   point a,b,c;

   // initialisation de ces points
   a.init(0.,0.);
   b.init(rayon*cos(alpha),rayon*sin(alpha));
   c.init(rayon*cos(alpha+dalpha),rayon*sin(alpha+dalpha));
}
```

Forward declaration

- Comme un triangle est fait de 3 points, il faut que la classe triangle **sache** ce qu'est un point.

- Pour ce faire, il faut **déclarer** une classe « *point* » au début du header `triangle.h`

- De plus, il faut inclure le header de la classe « *point* » dans le fichier `triangle.cpp`

- Dans le `triangle.h` on manipule donc des *points*, alors que cette classe ne sera que définie plus tard.

- Ceci est appelée un **déclaration avancée** ou forward declaration en anglais.

Exemple

- Voici le contenu du fichier triangle.h

```
class point;

class triangle
{

// attributs de la classe triangle
private:
    point m_sommet[3];    // tableau de points

// méthodes de la classe triangle
public:
    triangle();                          // constructeur
    ~triangle();                         // destructeur
    void init(point,point,point);  // initialisateur
    point sommet(int);                   // accesseur
    double perimetre();   // méthode rendant le périmètre
    double aire();        // méthode rendant l'aire
};
```

Fichier principal et compilation

- Une fois que les différentes classes ont été définies, il faut les inclure dans le fichier `main.cpp`

- Comme son nom l'indique ce fichier sera le document **principal**, *main* en anglais.

- Afin d'obtenir un executable, il faudra **compiler** les différents fichier.

- Pour cela, importez le *main* et les *classes* dans un **projet** dans votre éditeur

- Les fichiers complets concernant l'exemple des polygones sont disponibles dur eCampus.

Exercice

- Simulez les rebonds d’une balle lâchée d’une hauteur initiale en utilisant une classe « *sphere* ».