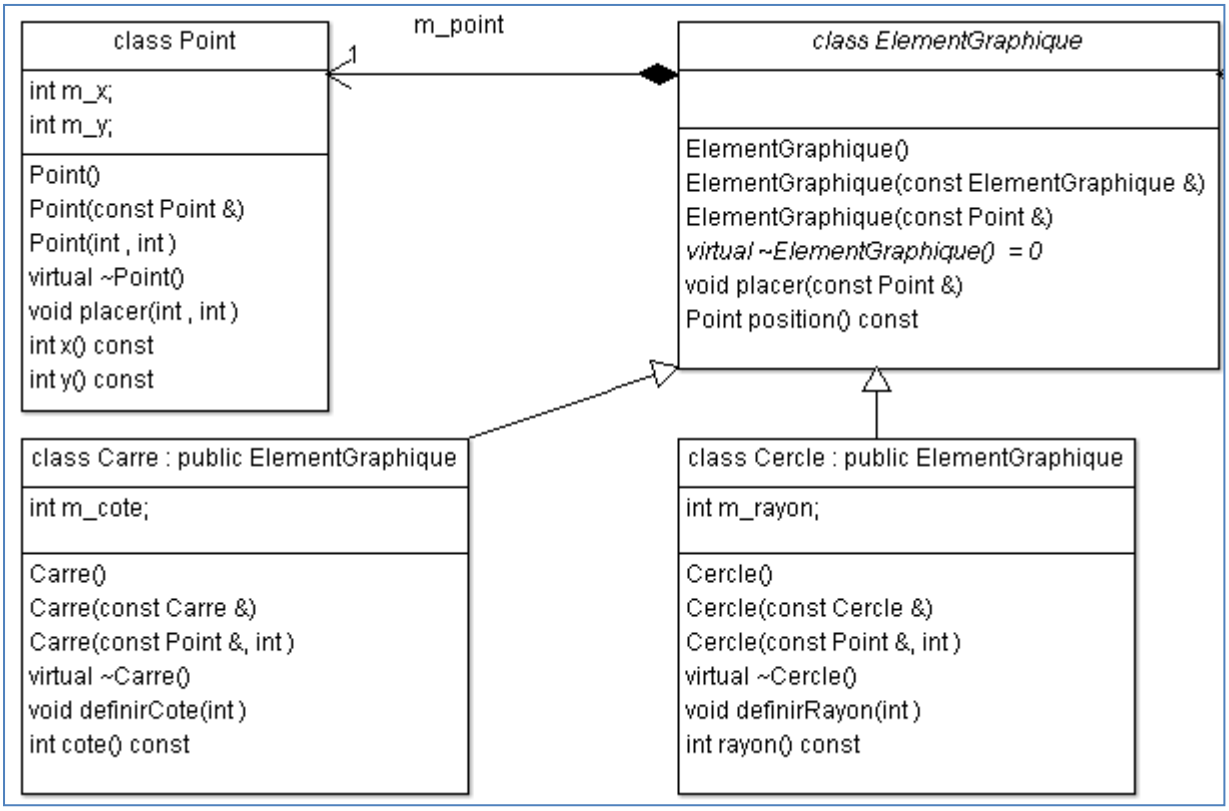
TD2

Exercice 1

Créer un nouveau projet.

Implémenter en C++ le diagramme de classes suivant :



Pour chaque classe, écrire les déclarations dans un fichier « .h » et les définitions dans un fichier « .cpp ».

Remarque : mettre « m_ » en préfixe des noms de variables membres d'une classe est une technique de meilleure lisibilité du code.

Le point de base est le coin supérieur gauche pour les carrés et le centre pour les cercles.

La valeur par défaut des abscisses et des ordonnées des points est 0.

La valeur par défaut des côtés des carrés et des rayons des cercles est 0.

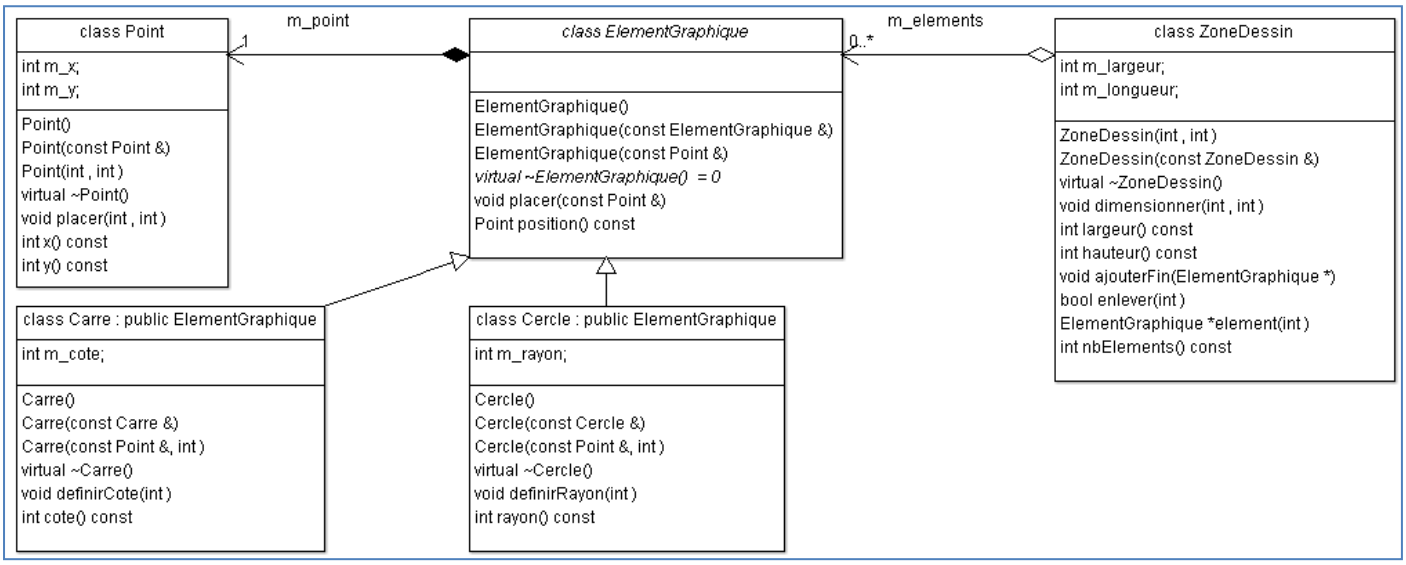
Pour chacune de ces classe, surcharger l'opérateur "<<" par fonction amie afin de pouvoir afficher des instances de vos classes via « std::cout » (inclus dans « iostream ») pour afficher dans la console les caractéristiques de l'objet à afficher, par exemple pour un cercle de centre (1,2) et de rayon 3 : « Cercle : rayon = 3 ElementGraphique : x = 1 y = 2 ».

Ecrire un programme principal qui crée un carré par défaut et un cercle de centre (5, 7) et de rayon 6 puis qui les affiche.

Exercice 2

Ajoutez à présent une nouvelle classe « ZoneDessin », représentant une zone dans laquelle on peut mettre des éléments graphiques. Cette classe stockera les adresses des éléments graphiques simplement dans un vecteur STL (std::vector<ElementGraphique*>).

Notre diagramme de classes ressemble donc à présent à celui ci-dessous (ElementGraphique) :



Pour ZoneDessin aussi, surcharger l'opérateur "<<" par fonction amie afin de pouvoir afficher des instances de cette classe via « std::cout » pour afficher dans la console les différents éléments, par exemple « Zone Dessin largeur : 100 hauteur : 100 nbElements : 1 Cercle : rayon = 3 ElementGraphique : x = 1 y = 2 »

Ecrire un programme qui crée une zone de dessin, y ajoute quelques carrés et cercles, puis affiche la zone de dessin.

Exercice 3

L'objectif de cet exercice est d'enrichir notre bibliothèque en ajoutant une méthode de recherche d'un élément graphique dans la classe ZoneDessin.

- Dotez toutes vos classes de la surcharge de l'opérateur d'égalité 'operator==’. Voici les définitions de ces surcharges afin de ne pas passer trop de temps sur ça :
    - “ElementGraphique.cpp”:

```
bool ElementGraphique::operator==(const ElementGraphique& opD) const
{
    return
            (this->position().x() == opD.position().x())
        &&  (this->position().y() == opD.position().y());
}
```

    - “Carre.cpp”:

```
bool Carre::operator==(const Carre & opD) const
{
    return (this->cote() == opD.cote()) && (this->ElementGraphique::operator==(opD));
}

bool Carre::operator==(const ElementGraphique& opD) const
{
    if (typeid(opD) == typeid(*this))
    {
      //return (*this == (Carre&)opD);
        return ((*this) == static_cast<const Carre&>(opD));
    }

    return false;
}
```

    - “Cercle.cpp”:

```
bool Cercle::operator==(const Cercle& opD) const
{
```

```
    return (this->rayon() == opD.rayon()) && (this->ElementGraphique::operator ==
(opD));
}

bool Cercle::operator==(const ElementGraphique& opD) const
{
    if (typeid(opD) == typeid(*this))
    {
      //return (*this == (Cercle&)opD);
        return ((*this) == static_cast<const Cercle&>(opD));
    }
    else
    {
        return false;
    }
}
```

- Ajoutez dans la classe ZoneDessin la méthode :

  int indice (const ElementGraphiques &) const

  - Cette méthode renvoie l'indice de la première case du tableau contenant un élément graphique identique à celui passé en paramètre, -1 s'il n'y en a pas.
  - Testez votre méthode dans un programme principal. Observez que la comparaison ne se fait pas qu'au niveau de l'élément graphique : deux carrés placés au même endroit ne sont pas "identiques" s'ils ont une dimension différente. Qu'est-ce qui permet un tel comportement ? Observer également qu'un cercle ne doit jamais être considéré comme identique à un carré…

Exercice 4

Observez que dans la méthode "`indice()`", vous êtes obligés de faire vous-même le parcours du vecteur. En effet, vous ne pouvez pas utiliser la fonction **`std::find()`** utilisable avec le conteneur « vector » car la zone de dessin contient des pointeurs, la fonction « find » comparerait donc les adresses des éléments et non les éléments eux-mêmes.

Afin de pallier à ce problème, ajouter au fichier « ZoneDessin.h » la classe « PEG » pour « Pointeur vers Elément Graphique » qui va simplement encapsuler le type « ElementGraphique* » :

```
class PEG
{
    protected:
    ElementGraphique * m_base;
    const ElementGraphique * m_constBase;

    public:
    ElementGraphique * base() const {return this->m_base;}
    bool operator==(const PEG &opD) const {return (* (this->m_constBase)) == (* (opD.m_constBase));}
    PEG(ElementGraphique * pEG) : m_base(pEG), m_constBase(pEG) {}
    PEG(const ElementGraphique * pEG) : m_base(NULL), m_constBase(pEG) {}
    PEG(const PEG & ori) : m_base(ori.m_base), m_constBase(ori.m_constBase) {}
};
```

Modifiez la classe « ZoneDessin » pour que sa donnée membre soit de type **`vector <PEG>`** et modifiez les définitions des méthodes en conséquence (ne modifiez pas les signatures des méthodes). Modifiez en particulier le corps la méthode "`indice()`" afin que celle-ci fasse appel à la fonction **`std::find()`**.