Lien GitHub: *https://github.com/ozzy-born/Heritage*

PARTIE 1

- Quelle méthode surface() est appelée ?

La méthode surface() utilisé est celle de la classe dérivée associé

- Pourquoi ?

Car l'objet appartient à cette classe et la méthode aussi.

```
void partie1()
{
    CTriangle triangle("Triangle_1", 10, 5);    // Creation d un triangle de base 5 et de hauteur 10
    CRectangle rectangle("Rectangle_1", 10, 5); // Creation d un rectangle de longueur 10 et de largeur 5
    CCarre carre(5);                            // Creation d un carre de cote 5
    COctogoneRegulier octogone("Octogone_1", 5);// Creation d un octogone regulier de cote 5

    triangle.afficher();    // Affichage du triangle
    triangle.surface();     // Affichage de la surface du triangle
    rectangle.afficher();   // Affichage du rectangle
    rectangle.surface();    // Affichage de la surface du rectangle
    carre.afficher();       // Affichage du carre
    carre.surface();        // Affichage de la surface du carre
    octogone.afficher();    // Affichage de l octogone regulier
    octogone.surface();     // Affichage de la surface de l octogone regulier
}
```

```
Microsoft Visual Studio Debug Console
Test des classes formes
Triangle de base : 5 et de hauteur : 10
et de Surface : 25
Rectangle de 10 x 5
et de Surface : 50
Carre de cote : 5
et de Surface : 25
Octogone regulier de cote : 5
et de Surface : 120.711
```

PARTIE 2

- Le résultat est-il correct ?

Le résultat n'est pas celui attendu.

- Expliquez ce qui se passe.

Nous utilisons la méthode de la classe CForme Car le pointeur point vers la Classe CForme.

```
void partie2_3() {
    CForme* mForme = new CRectangle("Rectangle_1", 10, 5);                    // Creation d un rectangle de longueur 10 et de largeur 5 via un pointeur de la classe de base
    std::cout << "Surface du rectangle via pointeur sur forme : " << mForme->surface() << std::endl;    // Affichage de la surface du rectangle via le pointeur de la classe de base
}
```

```
Microsoft Visual Studio Debug Console
Test des classes formes
Surface du rectangle via pointeur sur forme : 0
```

PARTIE 3

- Le comportement change-t-il ?

Oui, Le résultat est maintenant cohérent avec la forme donnée

- Pourquoi ?

vu que les méthodes sont virtuelles, le compilateur se charge de choisir la méthode de l'objet créé et non du type de référence.

```
void partie2_3() {
    CForme* mForme = new CRectangle("Rectangle_1", 10, 5);                    // Creation d un rectangle de longueur 10 et de largeur 5 via un pointeur de la classe de base
    std::cout << "Surface du rectangle via pointeur sur forme : " << mForme->surface() << std::endl;    // Affichage de la surface du rectangle via le pointeur de la classe de base
}
```

```
Microsoft Visual Studio Debug Console
Test des classes formes
Surface du rectangle via pointeur sur forme : 50
```

PARTIE 4

- Expliquez pourquoi on ne peut plus instancier la classe CForme

La classe CForme a une méthode abstraite ce qui en fait une classe abstraite et ce qui nous force à instancier des objet à partir des classes qui découllent de de CForme.

```
void partie4() {
    CForme forme(*forme_abstraite*);    //Compilation error : on ne peut pas instancier une classe abstraite
}
```

```
Ln: 35, Ch: 2, Col: 5   MIXED
Error List
Entire Solution   3 Errors   1 Warning   7 Messages   Build + IntelliSense   Search Error List
Code   Description                                                  Project          File                    Line
E0322  object of abstract class type "CForme" is not allowed:       Heritage_formes  Test_Heritage_formes.cpp  35
```

PARTIE 5

```
void partie5() {
    std::vector<CForme*> formes;                          //Creation d'un vecteur de type CForme*
    formes.push_back(new CRectangle("Rectangle_1", 10, 5));     // Creation d un rectangle de longueur 10 et de largeur 5 dans le vecteur
    formes.push_back(new CCarre(5));                            // Creation d un carre de cote 5 dans le vecteur
    formes.push_back(new CTriangle("Triangle_1", 10, 5));       // Creation d un triangle de base 5 et de hauteur 10 dans le vecteur
    formes.push_back(new COctogoneRegulier("Octogone_1", 5));   // Creation d un octogone regulier de cote 5 dans le vecteur

    for (int i = 0; i < formes.size(); i++) {
        formes[i]->afficher();  // Affichage de la forme à l'emplacement i du vecteur

    }

    for (int i = 0; i < formes.size(); i++) {
        delete formes[i];       //Liberation de la memoire
    }
}
```

```
Microsoft Visual Studio Debug Console
Rectangle de 10 x 5
et de Surface : 50
Carre de cote : 5
et de Surface : 25
Triangle de base : 5 et de hauteur : 10
et de Surface : 25
Octogone regulier de cote : 5
et de Surface : 120.711
```

PARTIE 6

<u>- Pourquoi Implémenter surface() uniquement dans CRectangle fonctionne ?</u>

L'implémentation fonctionne parce que la classe CCarre a accès aux méthodes de CRectangle et en hérite. Cependant dans cet exemple un problème survient, certainement car la Classe CRectangle demande un hauteur et une largeur pour ses paramètres mais que la classe CCarre n'a qu'un paramètre cote.

```
void partie6() {
    CCarre caree(5);

    caree.afficher();       // Affichage du carre
    caree.surface();        // Appel de la methode surface() de la classe CCarre
}
```

```
Microsoft Visual Studio Debug Console
Carre de cote : 5
et de Surface : 6.87195e+08
```

PARTIE 7

```
void partie7() {
    std::vector<CForme*> formes;                          //Creation d'un vecteur de type CForme*
    formes.push_back(new CRectangle("Rectangle_1", 10, 5));     // Creation d un rectangle de longueur 10 et de largeur 5 dans le vecteur
    formes.push_back(new CCarre(5));                            // Creation d un carre de cote 5 dans le vecteur
    formes.push_back(new CTriangle("Triangle_1", 10, 5));       // Creation d un triangle de base 5 et de hauteur 10 dans le vecteur
    formes.push_back(new COctogoneRegulier("Octogone_1", 5));   // Creation d un octogone regulier de cote 5 dans le vecteur
    formes.push_back(new CCercle(5));                           // Creation d'un cercle de rayon 5

    for (int i = 0; i < formes.size(); i++) {
        formes[i]->afficher();  // Affichage de la forme à l'emplacement i du vecteur

    }

    for (int i = 0; i < formes.size(); i++) {
        delete formes[i];       //Liberation de la memoire
    }
}
```

```
Microsoft Visual Studio Debug Console
Test des classes formes
Rectangle de 10 x 5
et de Surface : 50
Carre de cote : 5
et de Surface : -1.84943e+09
Triangle de base : 5 et de hauteur : 10
et de Surface : 25
Octogone regulier de cote : 5
et de Surface : 120.711
Cercle de rayon 5
et de Surface : 78.5397
```

PARTIE 8

Héritage :
L'héritage donne les méthodes et attributs publiques de la classe mère à une classe enfant.

Surchage :
La surcharge permet de donner des paramètres à une méthode.

Classe Abstraite :
L'abstraction des classes c'est le fait de rendre complètement virtuel des méthodes, les classes abstraites peuvent donner leurs méthodes mais ne peuvent pas être instanciées.

Polymorphisme :
Le polymorphisme c'est le fait d'avoir une classe pouvant prendre plusieurs formes, c'est à dire d'avoir plusieurs déclinaison possible selon les classes enfants de la classe mère.