	<b>La programmation orientée objet (POO) en C++</b>		<b>BTS SN-IR</b>
	Nom, Prénom : Menasria Jugurtha	Cours	

*Ce TP d'acquisition en POO vise uniquement à acquérir les bases, à appréhender un concept, des notions et des modèles qui sont fondamentaux, ce TP est fortement guidé, respectez chaque étape et répondez aux questions.*

## Premiers programmes en POO

Les premiers programmes que vous allez écrire sont uniquement là pour vous permettre d'appropriier les outils de base en POO.

N'oubliez de créer un répertoire où vous rangerez les programmes de ce TP.

### Application 1 :

Soit le programme ci-dessous, le saisir et observer le résultat en console :

```
#include <iostream>
using namespace std;
class Carre
{
public:
    int perimetre (int cote);
    int surface (int cote);
private :
    int cote;
};

int Carre::perimetre (int cote)
{
    return 4*cote ;
}
int Carre ::surface (int cote)
{
    return (cote*cote);
}
int main()
{
    int c;
    Carre carre1;
    cout << "*****Calcul propriétés d'un' carré*****" << endl;
    cout << "Entrez la valeur du coté en m:" ;
    cin >> c ;
    cout << "Périmètre du carré : ";
    cout << carre1.perimetre(c)<<" m" <<endl;
    cout << "Surface du carré : ";
    cout << carre1.surface(c)<<" m2" <<endl;
    return 0;
}
```

1. Quel est le nom de la classe de ce programme ? Quel est le nom de l'objet créé à partir de cette classe ?

**Réponse:** La classe de ce programme est "Carre". Les noms des objets créés par cette classe sont Carrel.

2. Quels sont les éléments de syntaxe qui permettent de repérer une classe dans un programme ?

**Réponse:** Le terme Class est affiché avant le nom de la classe.

- ### 3. Que signifient les mots public, privé ?

**Réponse:** Les mots public et privé servent à classer les différentes méthodes et attributs de la classe dans deux catégories:  
soit elles sont publiques et elles sont accessibles n'importe quand,  
soit elles sont privées et elles ne sont pas modifiables hormis avec des méthodes de sa propre classe.

4. Quel est le nom de l'objet créé dans ce programme ?

**Réponse:** Le nom de l'objet créé est Carrel.

- ## 5. Quelles sont les méthodes de ce programme ?

**Réponse:** Les deux méthodes sont Périmètre et Surface.

6. Quels sont les éléments de syntaxe utilisés pour la définition d'une méthode ?

**Réponse:** `int "Nom de la class": "Nom de la méthode"(){}`

7. Quels sont les éléments de syntaxe utilisés lors de l'appel d'une méthode ?

**Réponse:** "Nom de l'objet"."Nom de la méthode()";

### Application 2 :

Modifiez le programme précédent de façon à intégrer la méthode saisie : Cette méthode permettra à l'utilisateur de rentrer la valeur du côté du carré.

**Réponse:**

```

main.cpp x Carre.hpp x Carre.cpp x
1  #include "Carre.hpp"
2  #include <iostream>
3
4  using namespace std;
5
6  int Carre::perimetre(int cote)
7  {
8      return Carre::cote*4;
9  }
10
11 int Carre::surface (int cote)
12 {
13     return Carre::cote*Carre::cote;
14 }
15
16 void Carre::saisie()
17 {
18     int x;
19     cin >> x;
20     Carre:: cote=x;
21 }
22

```

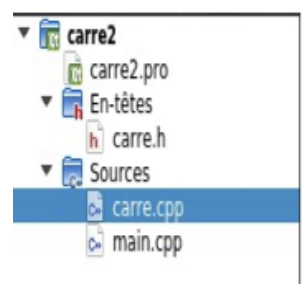
```

main.cpp x Carre.hpp x Carre.cpp x
1  #include <iostream>
2  #include "Carre.hpp"
3
4  using namespace std;
5
6  int main()
7  {
8      int cl;
9      Carre carrel;
10     cout << "*****Calcul proprietes d'un carre*****" << endl;
11     cout << "Entrez la valeur du cote en m:";
12     carrel.saisie();
13     cout << "Perimetre du carre : ";
14     cout << carrel.perimetre(cl)<<" m" <<endl;
15     cout << "Surface du carre : ";
16     cout << carrel.surface(cl)<<" m2" <<endl;
17
18     return 0;
19 }
20

```

## Déclaration d'une classe et des méthodes d'une classe en C++

La déclaration d'une classe se fait dans un fichier d'en tête (header), si nous reprenons l'exemple précédent : la classe Carre devient un fichier d'en tête **Carre.h**  
 La définition des méthodes de la classe se fait dans un fichier source ( .cpp), si nous reprenons l'exemple précédent la définition des méthodes devient un fichier source **Carre.cpp**



La première ligne du code dans le fichier Carre.cpp sera #include "Carre.h"  
 Il faudra aussi intégrer cette ligne dans la fonction main.

### Application 3 :

A partir du programme précédent, organisez celui-ci de façon à obtenir, un fichier d'en tête **Carre.h**, un fichier source **Carre.cpp**, testez le résultat en console.

#### Réponse:

The image shows two screenshots of a C++ IDE. The top screenshot displays the **Carre.hpp** file with the following code:

```

1  #ifndef CARRE_H
2  #define CARRE_H
3
4  class Carre
5  {
6      public:
7          void saisie();
8          int perimetre(int);
9          int surface(int);
10     private :
11         int cote;
12 };
13
14 #endif // CARRE_H
15

```

The bottom screenshot displays the **Carre.cpp** file with the following code:

```

1  #include "Carre.hpp"
2  #include <iostream>
3
4  using namespace std;
5
6  int Carre::perimetre(int cote)
7  {
8      return Carre::cote*4;
9  }
10
11  int Carre::surface (int cote)
12  {
13      return Carre::cote*Carre::cote;
14  }
15
16  void Carre::saisie()
17  {
18      int x;
19      cin >> x;
20      Carre:: cote=x;
21  }
22

```

## Encapsulation et Interface

Tout ce qu'il n'est pas nécessaire de connaître à l'extérieur d'un objet devrait être dans le corps de l'objet et identifié par le mot clé **private** : Attribut d'instance **privée** = inaccessible depuis l'extérieur de la classe. C'est également valable pour les méthodes.

Erreur de compilation si référence à un(e) attribut/méthode d'instance privée :

Si aucun droit d'accès n'est précisé, c'est private par défaut.

Dans la plupart des cas :

• **Privé :**

- Tous les attributs
- La plupart des méthodes

• **Publique :**

- Quelques méthodes bien choisies (interface)  
« **Accesseurs** » et « **Manipulateurs** »

Si on a besoin d'utiliser des attributs privés depuis l'extérieur de la classe

- Si le programmeur le juge utile, il inclut les méthodes publiques comme :

```
class Carre
{
public:
    int perimetre (int cote);
    int surface (int cote);
private:
    int cote;
};
```

**Manipulateurs (« méthodes set ») :**

- **Modification**
- **Affectation de l'argument** à une variable d'instance précise
- 

```
void setCote(int c) { cote = c; }
int getCote() { return c; }
```

**Accesseurs (« méthodes get ») :**

- **Consultation**
- **Retour** de la valeur d'une variable d'instance précise

**La variable « this »**

- « this » est un pointeur sur l'instance courante.

```
void setCote(int c) {
    this->cote = c; }
```

## Application 4 :

Réaliser un programme permettant de demander la longueur et la largeur d'un rectangle et d'afficher sa surface. (Vous utiliserez les méthodes get et set).

La définition de la classe, une définition d'une des méthodes et le début du programme principal vous est fourni ci-dessous.

**Conseil : Travaillez pas à pas**

**Réponse:**

```

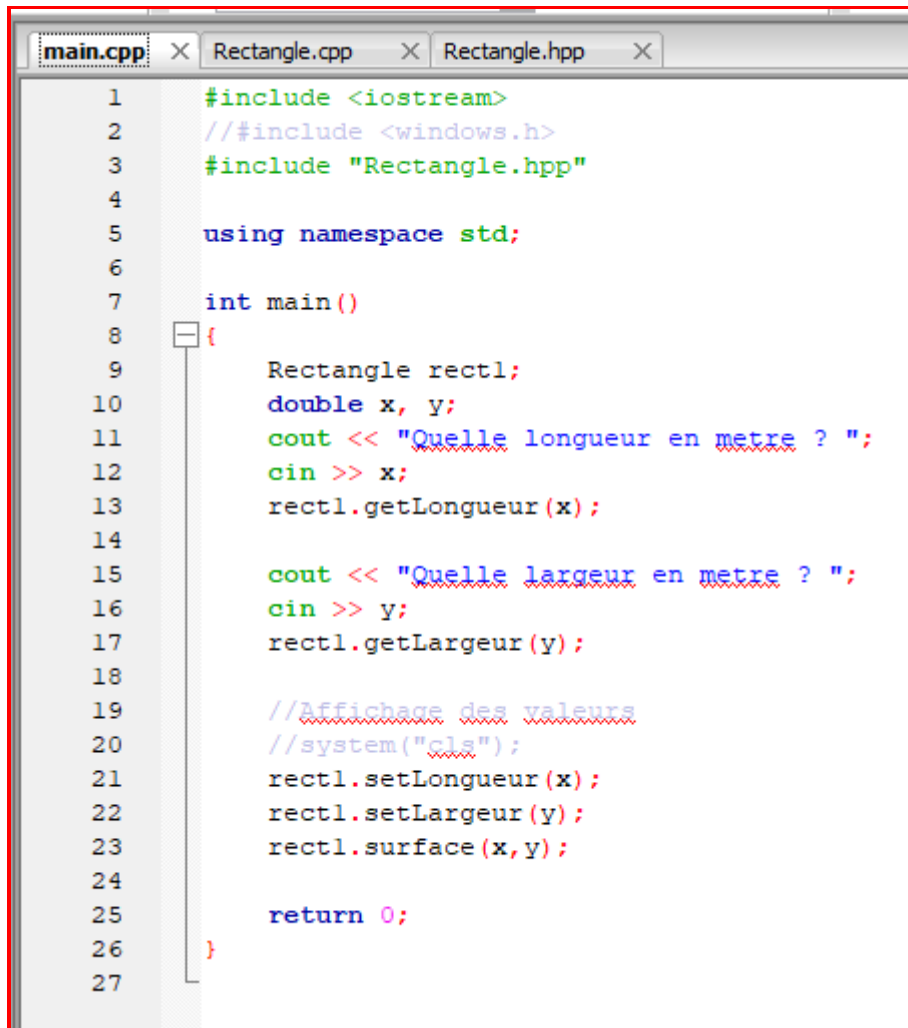
main.cpp x Rectangle.cpp x Rectangle.hpp x
1  #ifndef RECTANGLE_H
2  #define RECTANGLE_H
3
4
5  class Rectangle
6  {
7      public:
8          Rectangle();
9          virtual ~Rectangle();
10         double surface(double ,double);
11         double getLongueur(int);
12         double getLargeur(int);
13         void setLongueur(double);
14         void setLargeur(double);
15
16         protected:
17
18         private:
19             double longueur;
20             double largeur;
21     };
22
23 #endif // RECTANGLE_H
24

```

```

main.cpp x Rectangle.cpp x Rectangle.hpp x
15
16
17 double Rectangle::getLargeur(int largeur)
18 {
19     return largeur;
20 }
21
22 double Rectangle::getLongueur(int longueur)
23 {
24     return longueur;
25 }
26
27 void Rectangle::setLongueur(double longueur)
28 {
29     system("cls");
30     cout << "La longueur selectionne en metre: " << longueur << ".\n";
31 }
32
33 void Rectangle::setLargeur(double largeur)
34 {
35     cout << "La largeur selectionne en metre: " << largeur << ".\n";
36 }
37
38 double Rectangle::surface(double longueur,double largeur)
39 {
40     if (longueur==largeur)
41     {
42         cout << "\nCeci n'est pas un rectangle mais un carre car les deux variables sont identiques\n"<<longueur<< " = "<<largeur;
43     }
44     else
45     {
46         cout << "\nLa surface du rectangle vaut "<<longueur*largeur<< " metre carre.\n";
47     }
48
49     return longueur*largeur;
50 }
51

```



```

1  #include <iostream>
2  // #include <windows.h>
3  #include "Rectangle.hpp"
4
5  using namespace std;
6
7  int main()
8  {
9      Rectangle rect1;
10     double x, y;
11     cout << "Quelle longueur en metre ? ";
12     cin >> x;
13     rect1.getLongueur(x);
14
15     cout << "Quelle largeur en metre ? ";
16     cin >> y;
17     rect1.getLargeur(y);
18
19     //Affichage des valeurs
20     //system("cls");
21     rect1.setLongueur(x);
22     rect1.setLargeur(y);
23     rect1.surface(x,y);
24
25     return 0;
26 }
27

```

### Définition de la classe rectangle :

```

class Rectangle {
public:
double surface(double ,double );
double getLongueur(int);
double getLargeur(int);
void setLongueur(double );
void setLargeur(double );

private:
// déclaration des attributs
double longueur;
double largeur;
};

```

### La définition de la méthode permettant de consulter la valeur du rectangle s'écrit :

```

double Rectangle::getLargeur(int largeur)
{
    return largeur;
}

```

### Le début du programme principal peut s'écrire de la façon suivante si l'objet créé est rect1 :

```

int main() {
Rectangle rect1;
/*Rectangle rect2;*/
double x, y;
cout << "Quelle hauteur? ";
cin >> x;
}

```

```
rect1.setHauteur(x);
.....
```

## La couche objet : les constructeurs et destructeurs

Le **constructeur** et le **destructeur** sont deux méthodes particulières qui sont appelées respectivement à la création et à la destruction d'un objet. Toute classe a un constructeur et un destructeur par défaut, fournis par le compilateur.

- un constructeur porte le même nom que la classe dans laquelle il est défini
- un constructeur n'a pas de type de retour (même pas *void*)
- un constructeur peut avoir des arguments
- la définition d'un constructeur n'est pas obligatoire lorsqu'il n'est pas nécessaire

### Application 5 :

Soit le programme ci-dessous :

```
#include <iostream>
using namespace std;
class Point
{
private:
    double x;
public:
    Point(double param_x);
    void affiche ();
    double getX () const;
    void setX(double nouveau_x);
};
Point::Point (double param_x)
{
    x=param_x; }
double Point :: getX () const
{
    return x;
}
void Point :: setX (double nouveau_x)
{
    this->x=nouveau_x;
}
int main()
{
    Point p1(7);
    Point p2 (2);
    p1.setX(5);
    cout << "L'abscisse de P1 est " << p1.getX() << endl;
    cout << "L'abscisse de P2 est " << p2.getX() << endl;
    return 0;
}
```

1. Le saisir et observer le résultat obtenu en console ?

**Réponse:** Le programme récupère les valeurs de P1(x) et P2(x) et les affiche à la fin du programme.

```
E:\UM_BTS_SNIR\C++\TP SNIR2\TP1\Point\bin\Debug\Point.exe
L'abscisse de P1 est 5
L'abscisse de P2 est 2

Process returned 0 (0x0)   execution time : 0.023 s
Press any key to continue.
```

2. Ce programme utilise un constructeur, le repérez et indiquez son rôle

**Réponse:** `Point::Point (double param_x)` est le constructeur qui donne à "x" la valeur de "param\_x". Le rôle du constructeur est de créer l'objet et de l'initialiser afin de l'utiliser au cours du programme.

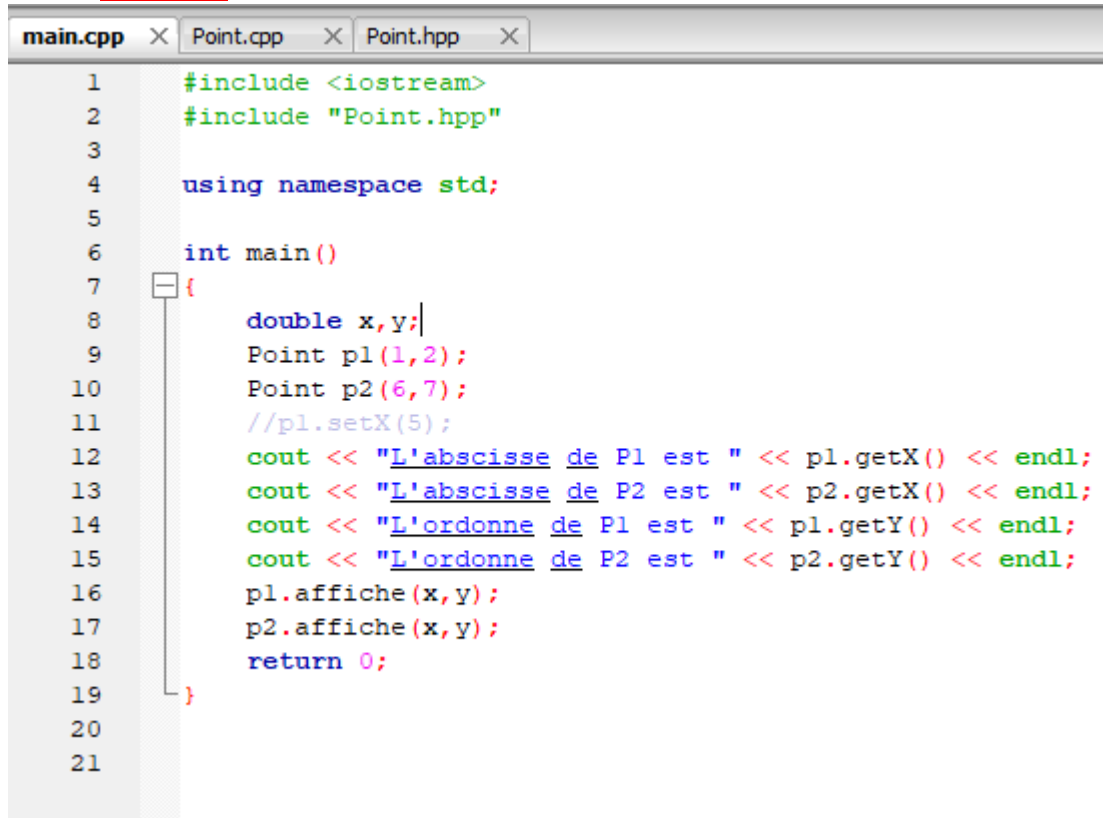


3. Ce programme utilise des accesseurs et des manipulateurs, les repérer dans le programme et indiquer leur rôle.

**Réponse:** GetX GetY sont les accesseurs, tandis que SetX et SetY sont les manipulateurs. Leur rôle est d'assigner leur valeur pour les accesseur et les afficher à l'utilisateur pour les manipulateurs.

4. Un point dans un plan est défini par deux valeurs, son abscisse et son ordonnée, le programme ci-dessous ne prend en compte qu'une seule valeur : l'abscisse, le modifier de façon à tenir compte de son ordonnée.

**Réponse:**



```
1  #include <iostream>
2  #include "Point.hpp"
3
4  using namespace std;
5
6  int main()
7  {
8      double x,y;
9      Point p1(1,2);
10     Point p2(6,7);
11     //p1.setX(5);
12     cout << "L'abscisse de P1 est " << p1.getX() << endl;
13     cout << "L'abscisse de P2 est " << p2.getX() << endl;
14     cout << "L'ordonne de P1 est " << p1.getY() << endl;
15     cout << "L'ordonne de P2 est " << p2.getY() << endl;
16     p1.affiche(x,y);
17     p2.affiche(x,y);
18     return 0;
19 }
20
21
```

```

main.cpp x Point.cpp x Point.hpp x
1  #include "Point.hpp"
2
3  Point::Point (double param_x, double param_y) //Fonction qui va assigner la position X et Y du point
4  {
5      x=param_x;
6      y=param_y;
7  }
8
9  double Point::getX() const                    //Fonction qui va retourner la position X du point
10 {
11     return x;
12 }
13
14 void Point::setX (double nouveau_x)           //Fonction qui va assigner une nouvelle position au point à l'abscisse
15 {
16     this->x=nouveau_x;
17 }
18
19 double Point::getY() const                    //Fonction qui va retourner la position Y du point
20 {
21     return y;
22 }
23
24 void Point::setY(double nouveau_y)           //Fonction qui va assigner une nouvelle position au point à l'ordonnée
25 {
26     this->y=nouveau_y;
27 }
28
29 void Point::affiche(double x,double y)        //Fonction qui va afficher la position X et Y de l'objet Point.
30 {
31     cout << "\nLe point se trouve sur la position X=<<this->x<< " Y=<<this->y;
32 }
33

```

```

main.cpp x Point.cpp x Point.hpp x
1  #ifndef POINT_H
2  #define POINT_H
3  #include <iostream>
4  #include <windows.h>
5  using namespace std;
6
7
8  class Point
9  {
10 private:
11     double x; // Attribut type double pour les abscisses
12     double y; // Attribut type double pour les ordonnées
13 public:
14     Point(double param_x, double param_y); //Constructeur de la classe
15     void affiche(double x, double y);       //affiche la position du Point
16
17     double getX() const;                   //Une constante qui va afficher la position x du point
18     void setX(double nouveau_x);           //Une fonction qui va retourner rien qui va mettre à jour la position x du point
19
20     double getY() const;                   //Une constante qui va afficher la position y du point
21     void setY(double nouveau_Y);           //Une fonction qui ne retourne rien qui va mettre à jour la position y du point
22 };
23
24 #endif // POINT_H
25

```

5. Sauvegardez ce programme **dans votre compte rendu avec un maximum de commentaires**, vous devez au final avoir trois fichiers :

- Un fichier d'en tête
- Un fichier .cpp contenant les méthodes de ce programme
- Un fichier main.cpp

**Réponse:** C'est fait, regardez screen au-dessus !