I. Description

Elles permettent de regrouper, au sein d'une même entité, des attributs, méthodes, constructeurs et destructeurs. C'est une structure trés améliorée : une classe permet non seulement d'accéder aux données de l'exterieur mais aussi d'encapsuler" des données pour qu'elles ne soient pas accessibles de l'exterieur. Elles se situent dans les fichiers .hpp.

II. Syntaxe

Voici la syntaxe de création de la classe Vecteur :

```
class Vecteur {    //la classe Vecteur ne contient rien };
```

III. Les attributs :

Il en existe deux sortes : **private** et **public**

- **private :** les données définies ici ne sont accessibles que de l'interieur de la classe elle-même.
- **public :** les données définies ici sont accessibles de l'interieur et de l'exterieur de la classe.

IV. Les méthodes

Les méthodes sont des "fonctions" définies à l'interieur de la classe, pouvant agir sur les attributs de *private* et *public*. Les méthodes sont accessibles à l'exterieur de la classe si elles sont définies en *public*.

exemple, on écrit dans le .hpp :

```
class Vecteur { private: int _dimension; // attribut de la dimension public: int prendre_dimension(); // prototype d'une méthode };
```

la méthode est décrite dans le .cpp :

```
int Vecteur::prendre_dimension() { return _dimension; }
```

V. Constructeurs

Ce sont les méthodes qui créent les objet du type d'une classe. Elles sont définies dans *public*.
Le constructeur est appelé lors de la création et porte le nom de la classe, par exemple pour la classe Vecteur

```
int main() { Vecteur v; //création de l'objet Vecteur de nom v --> appel du constructeur return 0; }
```

Il est décrit dans la classe de la façon suivante :

```
class Vecteur { private: public: //constructeur : Vecteur(); //prototype du constructeur ne retournant rien (même pas void) //portant le nom de la classe. };
```

Il existe différents constructeurs possibles, parmi ceux-là en voila 3 :

```
class Vecteur { private: public: //constructeur : Vecteur(); //prototype du constructeur ne retournant rien (même pas void) //portant le nom de la classe. };
```

VI. Destructeur

Le destructeur est appelé automatiquement lorsque le programme n'a plus besoin de l'objet ou doit le détruire, exemple : lors de la sortie d'une fonction dans laquelle on a créé l'objet. Il n'y a qu'un destructeur par classe. La syntaxe est la suivante :

```
class Vecteur {
public:
//constructeurs:
...
//destructeur:
~Vecteur(); //prototype du destructeur portant le nom de la classe
};
```

VII. Utilisation

- Ecriture du corps d'une fonction :

Après avoir écrit le prototype d'une fonction dans une classe se trouvant dans un .hpp, il faut décrire ce que fait la fonction dans le .cpp. Pour accéder à la classe de l'exterieur de celle-ci, afin d'y écrire le corps de la fonction, on utilise la syntaxe : " nom_classe::nom_fonction ". Exemple :

Le .hpp donne :

```
class Vecteur {
private:
int _dimension;
public:
int prendre_dimension();
};
```

Le .cpp donne :

```
int Vecteur::prendre_dimension() {
//Le fait d'avoir utilisé "Vecteur::" permet de faire comme si on se trouvait
// dans la classe : on peut avoir accès aux données private puisqu'on est dans
// la classe, utiliser les méthodes décrite...
return _dimension; //on retourne la dimension du Vecteur.
}
```

- Pour utiliser les méthodes d'une classe à l'exterieur de la classe :

Supposons que nous soyons dans le "main", c'est à dire à l'exterieur de la classe. Pour appeler la méthode "methode1(int)" de la classe "ma_classe" j'écrit :

```
int main() {
...
ma_classe.methode1(4); //exemple de l'appel d'une methode d'une classe.
... }
```