# STS SNIR PROGRAMMATION Lycée Charles de Foucauld - Nancy

Fiche C++ n° 5

**Principales notions abordées :**

➢ **classes abstraites**

Quand vous travaillez sur un projet, il est intéressant de commencer par une vue d'ensemble. Ceci vous amène à créer une classe de base (ou classe générique). Cette classe comporte des données et des fonctions membres dont certaines ne peuvent pas être détaillées à ce stade du développement.

* ***Les méthodes indéfinies sont appelées virtuelles pures. Une telle méthode se déclare en ajoutant =0 à la fin de sa déclaration***
* ***Une classe qui renferme au moins une méthode virtuelle pure est appelée classe abstraite.***
* ***On ne peut pas créer d'instance (ou d'objet) d'une classe abstraite***
* ***Une classe abstraite ne peut pas être utilisée comme argument ou type de retour d'une fonction***
* ***Une classe dérivée qui ne redéfinit pas une méthode virtuelle pure est elle aussi abstraite***
* ***Par contre, les pointeurs et les références sur une classe abstraite sont parfaitement légitimes et justifiés.***

**Exemple :**

|  |  |
| --- | --- |
| class A  {  private : ...... public : virtual void f1() = 0; virtual void f2() = 0;  }; | A est abstraite car elle dispose de deux méthodes virtuelles pures. |

|  |  |
| --- | --- |
| class B : public A  {  private : ... public : void f1()  { //contenu de la fonction f1}  }; | B est abstraite car f1() a été redéfinie mais pas f2(). |

|  |  |
| --- | --- |
| class C : public A  {  private : ... public :  void f1()  { // contenu de la fonction f1} void f2()  {// contenu de la fonction f2 }  }; | La classe C est concrête car f1() et f2() sont redéfinies. |
|  |  |

|  |  |
| --- | --- |
| class D : public B  {  private:... public:  void f2()  {// contenu de la fonction f2 }  }; | D est concrète car B avait redéfini f1() et donc il ne restait plus que f2() à redéfinir. |

Ref : Fiche C++ numéro 5\_light.docx maj le 19/12/2019 13:16:00 par Vincent ROBERT PAGE 1/2

# STS SNIR PROGRAMMATION Lycée Charles de Foucauld - Nancy

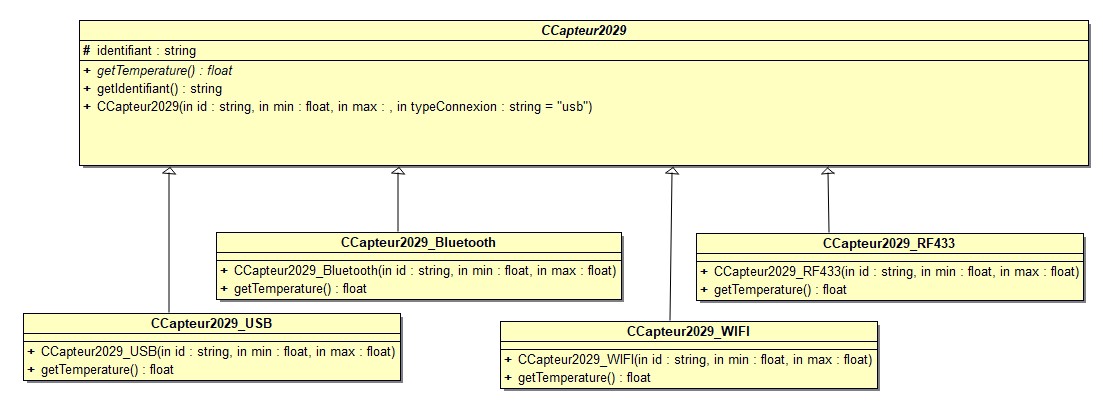
**Exemple d'application des classes abstraites** :

Nous sommes en 2029. Vous êtes chef de projet en Informatique. Votre société développe actuellement un capteur de température de dernière génération et vous demande de créer une application capable de récupérer les informations de ce capteur. Le directeur technique vous indique que ce capteur peut être interrogé de plusieurs manières :

* en filaire, par une connexion USB qui émule un port série virtuel
* en Bluetooth
* en WIFI
* en liaison radio RF 433 MHz

Vous devez donc construire une application qui sera capable de dialoguer de plusieurs façons différentes avec ce capteur. Vous allez déléguer à l'un de vos techniciens la réalisation d'un programme capable de dialoguer en filaire avec le capteur, à un autre un programme capable de dialoguer en Bluetooth, à un troisième un programme dialoguant en Wifi et au dernier un programme dialoguant en liaison RF 433 MHz. L'ensemble de ces programmes seront dans le futur rassemblés en une seule application.

Dans le but d'homogénéiser l'application de chacun de vos techniciens, vous avez la brillante idée1 de créer une classe abstraite qui donnera le canevas des méthodes à créer.



Classe abstraite

(

en

*italique*

)

car la

méthode

*getTemperature()*

est différente

selon l'interface choisi.

**On ne peut pas créer d'o**

**bjets**

**(**

**instancier) d**

**e cette c**

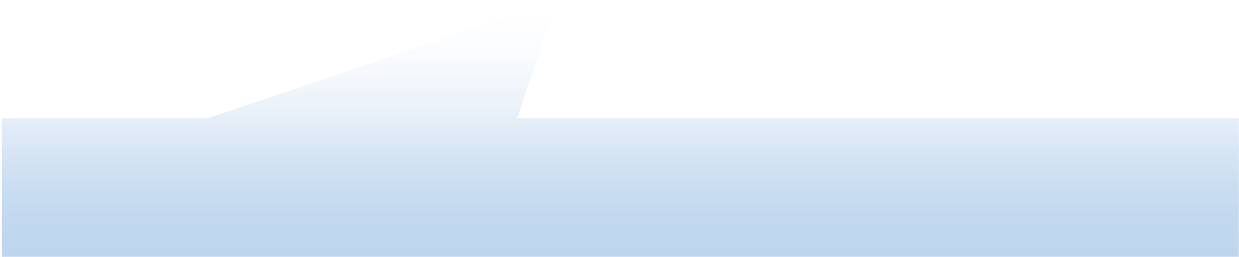
**lasse.**



La méthode

*getTemperature( )*

est virtuelle pure



Toutes ces classes sont concrètes. Elles redéfinissent la méthode

**getTemperature(**

**)**

en

respectant scrupuleusement son prototype.

1 Vous repenserez alors à votre vieux prof d'Informatique qui vous enseignait le C++ alors que vous étiez en BTS 😊

Ref : Fiche C++ numéro 5\_light.docx maj le 19/12/2019 13:16:00 par Vincent ROBERT PAGE 2/2

main()

{

CCapteur2029 \*capteur ;

// On crée un capteur USB

Capteur = new CCapteur2029\_USB(« 123789 », -57, 213) ;

Cout << capteur->getTemperature() ;

}