Notions avancées du C++ (2)

Exemple d'introduction (système InfoBus)

Pour introduire la notion de classe abstraite, nous allons partir d'un exemple. Un calculateur doté de son application informatique doit envoyer via une liaison série une trame contenant des informations à un autre calculateur (boitier BC1004).

Par conséquence on doit concevoir et programmer cette application permettant de paramétrer et gérer le port com du calculateur.

Classe CCommunication :

Cette classe représente le fait que pour communiquer nous allons devoir paramétrer l'interconnexion entre le calculateur équipé du port RS232 et le boitier BC1004.

La classe communication se veut être une classe générale en effet pour communiquer entre deux équipement on peut le faire par une liaison Série mais on peut tout aussi bien le faire par une liaison Ethernet ou CAN etc.

Quel que soit le support utilisé on devra envoyer un message et recevoir un message. C'est ce que nous allons signifier à notre classe Communication

- Comme pour la fonction main de la classe application nous indiquons à la classe **CCommunication** qu'elle possède la méthode ***EnvoyerMessage*** et ***RecevoirMessage***

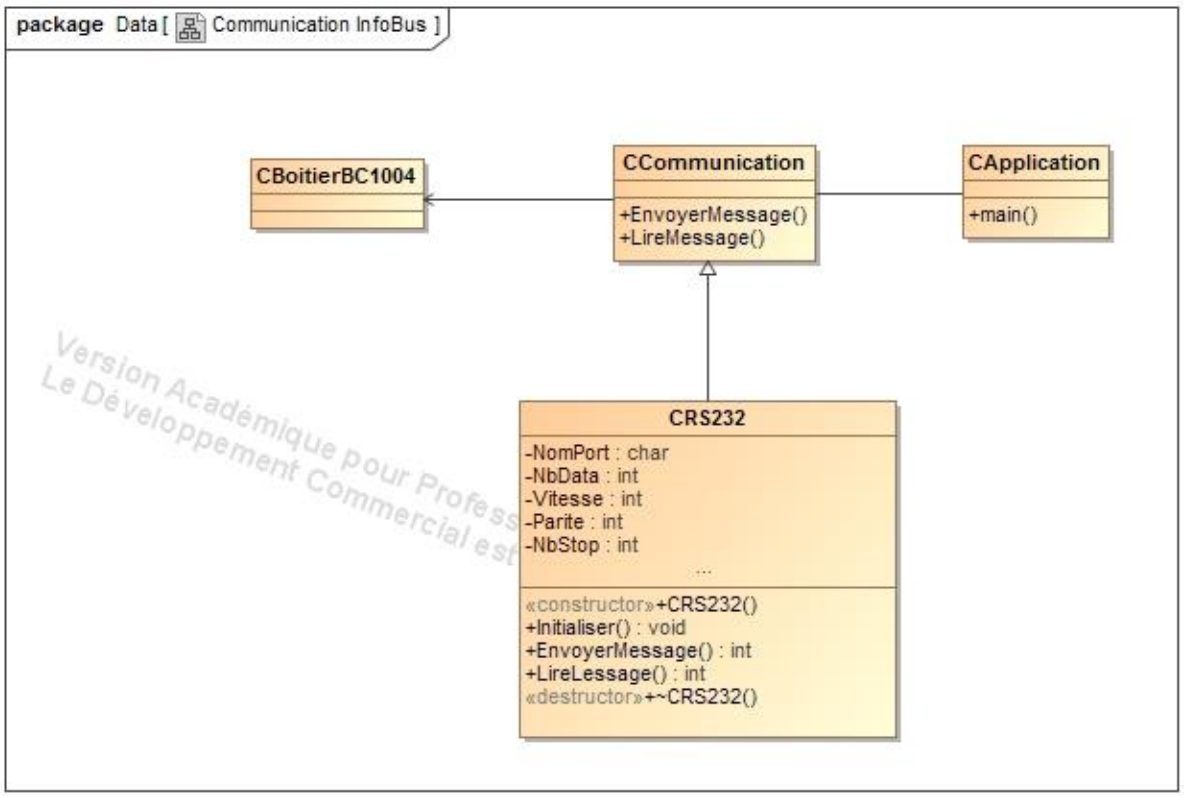
Spécialisation de la classe CCommunication :

Notre communication est réalisée par une liaison particulière qui est une liaison série. Nous allons donc modifier notre diagramme afin de mettre en évidence cette spécialisation.

Cette fois nous allons définir complètement notre classe puisque c'est elle qui servira de support à la création des objets de ce type. Comme vous l'avez déjà vu dans l'étude de la mise en œuvre d'une

liaison série ce type de support doit être paramétré. On doit notamment connaître la taille des données pour chaque trame, le nombre de bits de stop etc.

Le diagramme de classe devient donc le suivant :



En observant notre diagramme on constate que dans la classe CCommunication nous avons deux méthodes EnvoyerMessage et LireMessage et l'on retrouve ces deux méthodes dans la classe CRS232

Ceci est normal mais la **représentation n'est pas correcte**. En effet de façon générique on peut dire que quel que soit le support de communication utilisé il faudra envoyer le message et lire le message

La manière de lire le message sera spécialisée suivant le support utilisé. On ne lit pas de la même façon un message transporté par une liaison série et une liaison Ethernet.

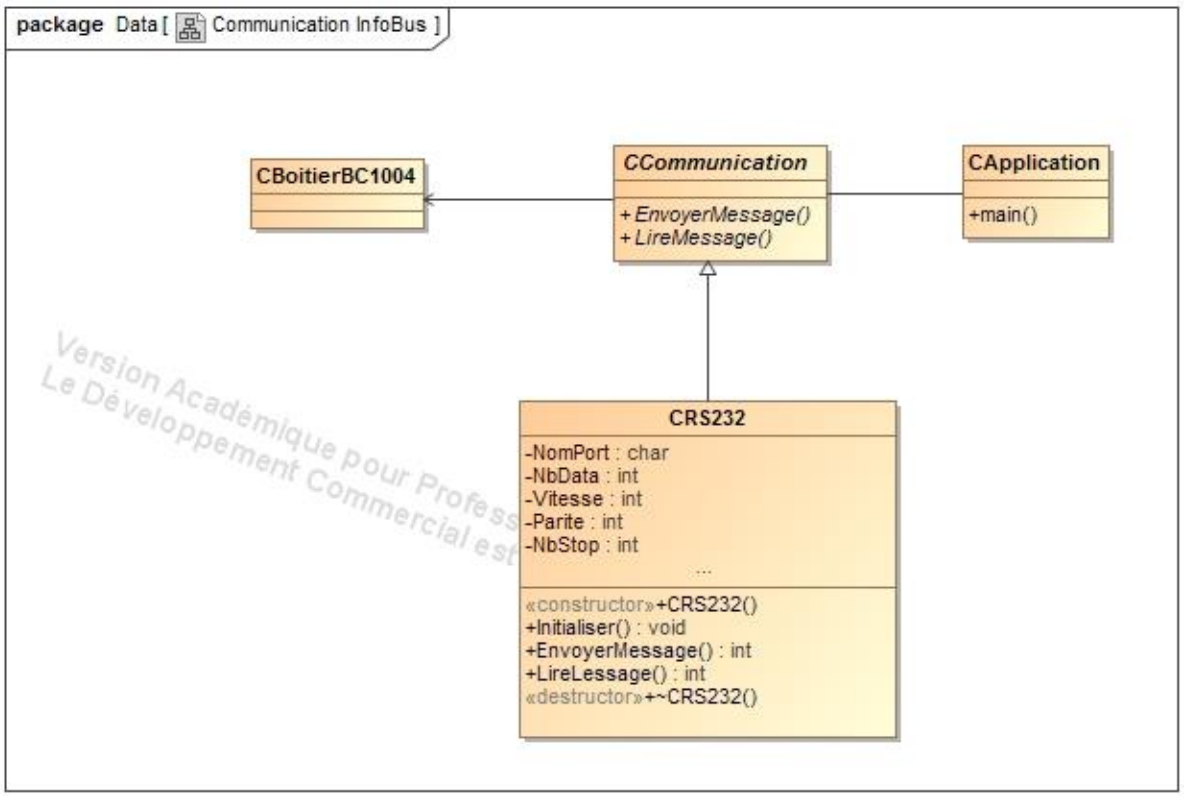
Cette remarque nous oblige donc à **redéfinir la méthode** dans chaque **classe héritée** de la classe CCommunication. Attention, **ne confondez pas redéfinition et surcharge** (dans le cadre de la surcharge la **signature** (prototype) de la méthode est différente).

On peut dire alors que les deux **méthodes** de la classe CCommunication sont **virtuelles** ceci nous oblige à le préciser dans la classe principale.

Mais comme il faut **spécialiser** ces deux méthodes pour **chaque support** il faut donc les **re-déclarer dans chaque classe héritée** ce qui nous oblige à dire que les deux **méthodes de la classe mère sont virtuelles pures** c'est à dire qu'au niveau du **code chaque méthode virtuelle pure sera égale à 0**

Cette déclaration particulière à pour conséquence que **la classe mère devient abstraite** et par conséquence **elle ne peut produire un objet** on dit qu'elle **n'est pas instanciable.**

Le diagramme de classe devient :



On reconnait une **classe abstraite** dans un diagramme de classe si **son nom** est représenté **en italique**. Les **méthodes virtuelles** pures sont également **écrites en italique**.

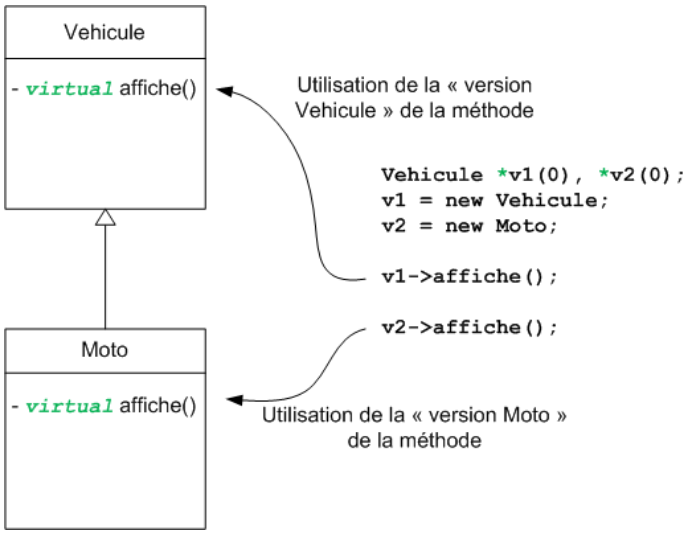
Rappel : principe du polymorphisme :

- Capacité du système à choisir dynamiquement la méthode qui correspond au type de l'objet en cours de manipulation

ou

- choisir en fonction des besoins quelle méthode appeler et ce au cours même de l'exécution.

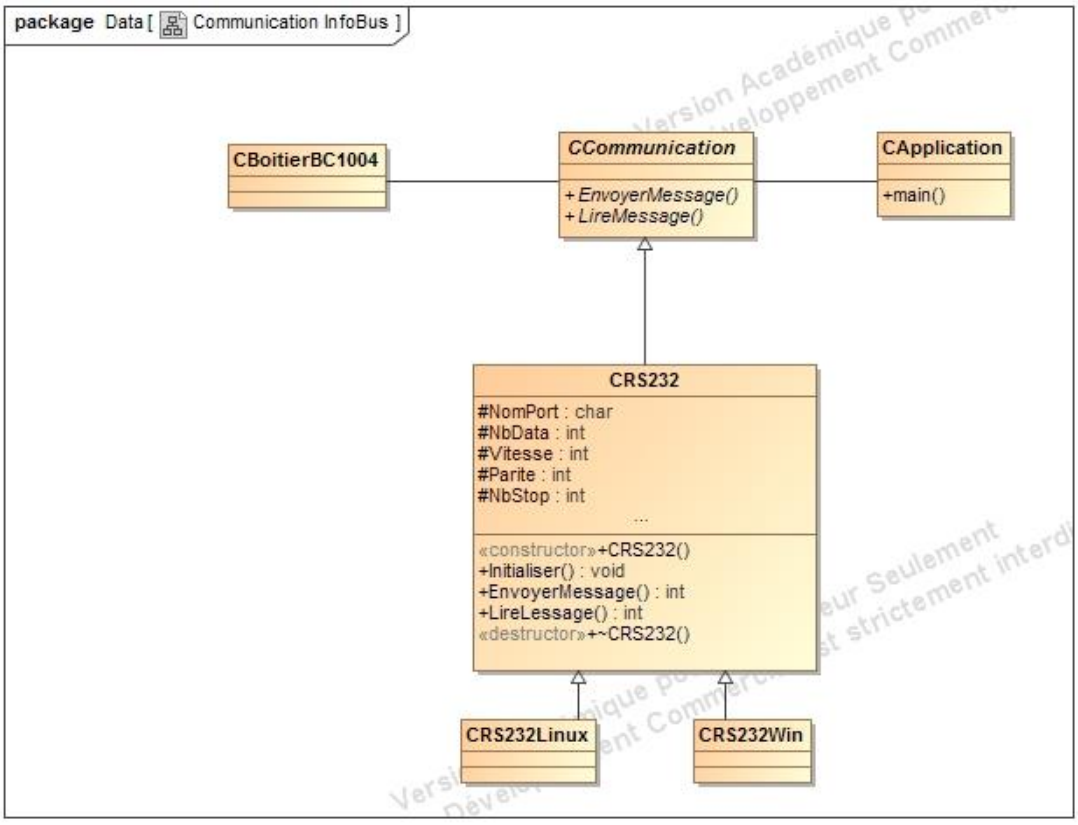
Exemple :



Le **polymorphisme** est implémenté en C++ avec les **fonctions virtual et l'héritage**.

Maintenant que nous avons déclaré les différents attributs et méthodes nécessaires à la programmation d'une liaison série il nous reste à adapter notre étude au système d'exploitation utilisé.

En effet la programmation d'une liaison série sous windows diffère de celle sous Linux. L'objectif n'est pas de compléter la classe CRS232 mais de spécialiser une classe Crs232Win qui reprendra alors toutes les caractéristiques de la classe mère à laquelle on ajoutera la spécificité de Windows.



- CCommunication.h

```
class CCommunication
{

	public :
		// le constructeur peut être nécessaire si la classe contient des
		//attributs a initialiser
		CCommunication();
		virtual  ~CCommunication();// un destructeur peut être virtuel


		virtual void EnvoyerMessage()= 0;  // méthode virtuelle pure
		virtual void LireMessage()= 0;    // méthode virtuelle pure

}
```

- CRS232.h

- CRS232Win.h