TP Chaîne d'acquisition – 5
Programmation Orientée Objet PHP

Objectif :
Dans le TP précédent, un nano-ordinateur sous Raspberry OS (Linux) doit recevoir des informations issues d'un système externe (Arduino) via sa liaison série, traiter ses informations pour les afficher en HTML ou les enregistrer dans une base de données.
Le code créé ne respect pas complètement la philosophie OBJET. Ce TP va nous aider à nous améliorer en Programmation Orientée Objet (POO)

On vous fournit :

- Une platine Arduino avec un logiciel embarqué qui génère des trames GPS, ou qui est capable de recevoir des commandes/
- Un nano-ordinateur Raspberry et sa carte SD préchargée.
- Un câble de liaison USB

Pré-requis : Le TP Chaine d'acquisition 1 et 4

Sommaire

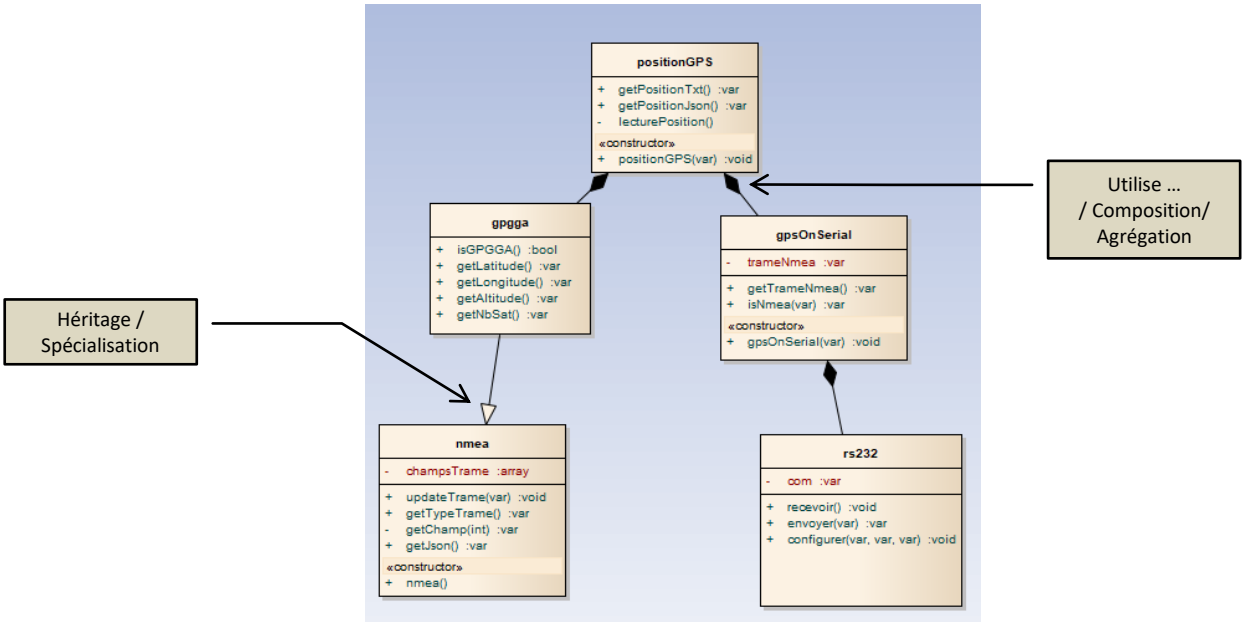
1 ANALYSE (A ETUDIER)

Nous avons créé un code utilisant une ou plusieurs **fonctions**, au fur et à mesure des besoins.

Une réflexion davantage orientée « objet » pourrait nous amener à ce genre de décision :

1. Un GPS pourrait être un objet logiciel qui utilise un objet liaison série.
   Cet objet fournirait les trames NMEA présentes sur la liaison série.
2. Une trame NMEA pourrait être un objet logiciel.
   On lui confie une chaine de caractères et il l'analyse.
   Si c'est trame NMEA, il pourrait nous fournir le type de trame ($GPGGA, $GPAAM, …)
3. Une trame $GPGGA qui est une trame particulière NMEA pourrait être une spécialisation de la classe NMEA.

Dans ce cas on aurait le diagramme de classes suivant :



Classe ou **Objet** ? Une « **classe** » est un modèle. L' « **objet** » est une « matérialisation » de la classe. On utilise l'expression « instance de classe ».
Pour obtenir un « objet logiciel » il faut obligatoirement créer d'abord la classe correspondante.

Comparaison : Le plan d'une maison correspond à une « classe ».
La maison construite selon ce plan correspond à l' « objet »
La maison est réelle, le plan est juste l'idée de la maison
Le même plan peut servir à construire plusieurs maisons identiques.
Une fois construites, les maisons ont leur propre vie indépendante.

La classe **rs232** est chargée de lire effectivement le port série, jusqu'à la réception d'un caractère particulier.
La classe **gpsOnSerial** représente un GPS qui reçoit ses trames (repérées par des délimiteurs) par une liaison série.
La classe **nmea** représente une trame nmea quelconque. Une trame commence par $ et est séparée par des virgules. Le premier champ indique le type de trame.
La classe **gpgga** est une spécialisation de la classe cl_nmea. Elle permet d'obtenir les informations précises de la trame nmea (latitude, longitude, …)
La classe **positionGPS** est la classe principale de l'application. Grâce aux classes gpgga et gpsOnSerial. Le programme principal sera donc ramené à sa plus simple expression grâce à cette bibliothèque logicielle que nous aurons créée.

Ce diagramme de classe n'est pas la seule façon de représenter notre besoin. Chaque programmeur aura certainement sa propre analyse …

Le diagramme ci-dessus a été créé par le logiciel **Entreprise Architect**, et ensuite les fichiers php ont été générés automatiquement. Les classes sont prêtes, il suffit de compléter le code des méthodes.

Nous allons répartir le code PHP trouvé au TP 1 dans les différentes classes.
L'ensemble va générer plus de code que dans la version non-objet.
Mais l'avantage de la solution est sa modularité et ses possibilités d'évolution.

TRAVAIL :
Décompressez les fichiers php fournis, transférez ces fichiers sur le Raspberry et ouvrez les en mode ssh/sftp via BitVise et avec NotePad++

2 DEVELOPPER ET TESTER UNE CLASSE

2.1 La classe « rs232 »

Cette classe était fournie en C++ mais ici, on va la développer nous-mêmes.

TRAVAIL :
Ouvrez le fichier rs232.php
Repérez le constructeur de classe, et les 3 méthodes à compléter.
Le constructeur restera vide.

2.1.1 Commençons par la méthode « configurer »

```
public function configurer($port, $vitesse, $options)
      {
            $this->com = fopen($port,"r+");
            $cmd = "stty $vitesse $options < " . $port;
            system($cmd);
      }
```

Explications :
C'est une méthode de classe mais l'écriture se fait comme celle d'une fonction classique. On précise juste si la fonction est « publique » : utilisable par tous
« private » : utilisable à l'intérieur de l'objet seulement
ou « protected » : utilisable à l'intérieur de l'objet et dans les objets hérités.

On remarque également la variable : `private $com;`
déclarée dans la zone des « attributs » ou « propriété » de la classe. C'est une variable globale à la classe.
Elle correspond au lien « utilise » dans le diagramme de classes.
Pour l'utiliser, on doit précéder son nom de `$this->`

2.1.2 La méthode « recevoir »

```
public function recevoir($carFin)
{
	$car = "";
	$ligne = "";
	while ($car != $carFin) {

		$car = fread ($this->com, 1);
		$ligne .= $car;
	}
	return $ligne;
}
```

Nous retrouvons strictement le code développé dans le TP précédent.
Le « return » retourne une ligne terminée par le caractère fourni en argument.
Tapez ce code au bon endroit.

2.1.3 La méthode « envoyer »

Bon … on ne peut pas plus simple :

```
public function envoyer($chaine)
{
	fwrite($chaine);
}
```

2.1.4 Tester notre classe rs232

Avant d'utiliser notre classe dans une application complexe, il faut prouver qu'elle fonctionne en créant un petit module de test.
En voici un :

```
<?php
	require_once ('rs232.php'); // Equivalent du #include

	$port = "/dev/ttyUSB0";
	$rs232 = new rs232();  // Création de l'objet

	$rs232->configurer($port, 4800, "-echo -opost crtscts -hupcl");

	while (true)
	{
		$lu = $rs232->recevoir("\n");
		echo $lu;
	}
?>
```

TRAVAIL : Créez le fichier de test.
Connectez le simulateur de trame et mettez le bon « tty » dans le code ci-dessus. Testez.

2.2 Explications sur la classe « gpsOnSerial » (LIRE)

Le code est fourni. Cette classe permet d'obtenir une trame NMEA quelconque sur la liaison série. Une trame NMEA commence par $. On réutilise donc notre classe rs232 développée précédemment.

2.2.1 Le constructeur de classe :

On crée un objet rs232 à partie de l'information fournie en argument (le nom du port) et on effectue les réglages adaptés à notre gps-Arduino :

```
function __construct($port)
      {
        $this->trameNmea = '$VIDE';
        $this->m_rs232 = new rs232();
        $this->m_rs232->configurer($port, 4800, "-echo -opost crtscts -hupcl");
      }
```

2.2.2 Reconnaitre une trame NMEA

C'est une trame qui commence par le signe $. On effectue une lecture du port série jusqu'à obtenir une chaine de caractères commençant par « $ » :

```
      public function getTrameNmea()
            {
                  $ligne = $this->m_rs232->recevoir("\n");

                  while (  ! $this->isNmea($ligne) )
                        $ligne = $this->m_rs232->recevoir("\n");

                  $this->trameNmea = $ligne;
                  return $this->trameNmea;
            }
```

C'est la méthode « isNmea » qui détermine si la chaine de caractère commence par « $ » :

```
      public function isNmea($ligne)
            {
                  if ( $ligne[0] == "$")
                        return true;
                  else
                        return false;
            }
```

2.3 La classe « nmea »

Le code est fourni. L'élément central de cette classe est la méthode « updateTrame » qui transforme une chaine de caractère ($trame) en tableau. Chaque élément du tableau sera une partie de la chaine de caractères, en se basant sur le séparateur « virgule ». C'est la fonction « explode » du PHP qui réalise cela :

```
            public function updateTrame($trame) {
                  $this->zones = explode(",", $trame);
            }
```

2.4 La classe « gpgga » : hérite de « nmea »

Nous allons créer une spécialisation (héritage) de la classe « nmea » : la classe « gpgga » (fichier gpgga.php) Cette classe permettra d'accéder aux éléments d'une trame gpgga.

Notez le lien d'héritage (voir le diagramme de classes) qui se traduit par le code suivant :

```
class gpgga extends nmea {
```

C'est le mot « extends » qui exprime l'héritage.

Pour tester que la trame NMEA reçu est une trame GPGGA, le code de la méthode « isGPGGA » est le suivant :

```
public function isGPGPGA() {
      if ( $this->getTypeTrame() == '$GPGGA')
            return true;

      return false;
}
```

Explications : On appelle la méthode `getTypeTrame` qui appartient à la classe « nmea ». Comme notre classe en hérite, elle est autorisée à la faire comme s'il s'agissait d'une de ses propres méthodes.

TRAVAIL :

1. Saisir le code au bon endroit
2. Créer le code des autres méthodes de la classe.
3. Ecrire le code du programme de test de cette classe. Comme on ne teste QUE cette classe, on crée une trame de test indépendante de la liaison série :

```
$trame = "$GPGGA,142001.289,4352.8801,N,00545.4841,E,1,04,3.2,200.2,M,,,,01"
```

3 TEST FINAL

Le code des autres classes est fourni
Ecrire le programme principal qui affichera la position gps en utilisant la classe « positionGPS »
Vous constaterez qu'avec une bonne bibliothèque, le programme principal devint très léger…