But du TP :
Se familiariser avec la programmation en **PHP/POO** des classes/objets.
Après une phase de familiarisation avec les notions apprises en cours (Pratique 1), nous implémenterons une structure de données simple (une pile) dans la Pratique 2, avant de nous en servir dans les pratiques suivantes.

Adresse email de retour du rapport de TP : `eric.cabret@gmail.com`

Pratique 1 :
Tester les concepts PHP/POO de classes et d'objets décrits dans le cours.

1.0 : Reprendre le cours principal "**PHPPOO_C4_PHP_POO.pdf**"

1.1 : Créer une ou plusieurs pages Web permettant de tester les concepts décrits dans le cours :

- Page 0 : Déterminer la version de PHP installée sur votre installation avec **phpinfo()**.
- Page 21/22 : Tester le fonctionnement de la majorité des "**méthodes magiques**" de **PHP**.
- Page 24 : Notion de classe/constructeur/instance d'objet : tester la classe **Employe**
- Page 27 : Tester la notion de **constantes** en **PHP/POO**
- Page 30 : **Clonage** d'objets
- Page 31 : Démontrer la différence entre **==** et **===**
- Page 38 : Rendre privé l'attribut **$salaire** d'**Employe** et tenter d'écrire -12000 directement dedans. Que se passe t'il ?
- Page 39/40/41 : Notion de **variable** (tester le fonctionnement) et de **méthode** de **classe**
- Page 42 à 45 : Composition d'objets :
  Créer une classe Voiture composée de Moteur, Portiere, Roue et Pneu

- Page 16 : Initiation à **UML**. Installer le logiciel **ArgoUML** (http://argouml.tigris.org/) et l'utiliser pour représenter la classe **Personne** décrite en page 16 du cours : ce graphique correspond à un « mini » **diagramme UML de classes**.
  Générer le code PHP correspondant grâce à **ArgoUML** puis tester ce code.

Merci de faire une copie d'écran pour chacun des exemples avec un titre qui précise ce qui est testé.

Pratique 2 :
Utilisation du PHP POO pour définir une pile logicielle

Une pile est un ensemble d'objets gérés de telle manière que le dernier élément ajouté est le premier que l'on peut extraire. On effectuera des tests de piles avec un élément de pile de type une **valeur entière**.

2.1 : Définir la classe **`Pile`**.
Le constructeur de cette classe construira une pile vide.
Les éléments seront gardés dans un tableau de dimension variable (100 par défaut si la dimension est non précisée).
Un champ privé stockera à la fois le nombre d'éléments actuellement dans la pile et l'indice de son sommet.

2.2 : Définir et tester une méthode **`estVide()`** permettant de tester si la pile est vide.

2.3 : Définir et tester la méthode **`empile()`** qui ajoute un élément au sommet de la pile.

2.4 : Définir et tester la méthode **`depile()`** qui retourne le contenu du sommet et le retire de la pile.

2.5 : Définir et tester la méthode **`sommet()`** qui retourne le contenu du sommet de la pile sans le retirer.

2.6 : Définir et tester la méthode **`affiche()`** qui affiche le contenu d'une pile, sans la modifier.
Les éléments au sommet de la pile sont affichés en premier.

Pratique 3 :
Extensions de la pile logicielle précédente

Le but de cette pratique est de créer et tester 2 méthodes qui vont faire des traitements sur plusieurs piles :

- `union($A, $B)`
- `tri($A)`

Ces méthodes seront à ajouter dans la classe `Pile` précédente en tant que **méthodes de classe**.

3.1 : Définir et tester la **méthode de classe** `union($A, $B)` qui admet 2 piles `$A` et `$B` en paramètre et qui retourne une pile qui est la fusion de ces 2 piles.

Astuce : il sera utile de rajouter une méthode supplémentaire dans la classe `Pile` précédente qui se nommera `getTailleCourante()`.

3.1bis : Dans l'exercice **3.1** précédent, on constate que les piles `$A` et `$B` sont vidées.

Corriger la méthode `union($A, $B)` précédente afin qu'elle préserve les contenus des piles `$A` et `$B` d'origine.

3.2 : De manière similaire à l'exercice **3.1**, écrire une **méthode de classe** `tri($P)` qui créée une pile vide `$res`, remplit cette pile avec les éléments de `$P` d'une manière ordonnée avec le minimum en haut et retourne `$res`.
Tester votre méthode de tri avec une pile `$A` contenant les valeurs suivantes `4 (haut de la pile), 3, 2, 5, 8, 2, 6, 9, 3`.
Indication :
L'algorithme proposé sera le suivant : utiliser une pile auxiliaire `$aux` qui est vide au début et tant que la pile `$P` n'est pas vide, il faudra considérer les deux cas suivants :

- si la pile `$res` est vide ou si l'élément au sommet de `$P` est plus petit que celui de `$res` : on retire l'élément au sommet de la pile `$P` pour l'empiler dans la pile `$res`, puis si la pile `$aux` n'est pas vide, on retire tous les éléments de la pile `$aux` pour les empiler sur la pile `$res`.
- sinon : on déplace l'élément au sommet de la pile `$res` vers la pile `$aux`.