Step by step + explication :

→ un script Employee.php qui définit la classe Employee comportant quatre proprié- tés :

- id : entier, accès privé.
- name : chaîne de caractères, accès public. • salary : réel, accès protégé.
- age : entier, accès privé.

Employee doit implémenter l'interface IEmployee qui déclare les méthodes suivantes : • Constructeur avec id, name, salary et age en paramètres.

• Accesseurs (*getters*) et mutateurs (*setters*).

• Méthode magique d'affichage des propriétés de l'objet.

→ un script employee_display.php qui crée un tableau de trois employés et les affiche à l'écran. On affichera également le salaire moyen de ces trois employés. Pensez à utiliser les fonctions array_*.

→ un script employee_raise.php qui utilise le programme précédent et augmente la salaire de chacun des employés de 5%. Ce calcul s'effectuera avec une fonction employee_raise qui attend en paramètre un employé, vérifie que le paramètre est bien un objet et est de classe Employee, ou sinon lève une exception.

→ un script employee_sort.php qui trie un tableau d'employés en ordre de salaire croissant.

→ Le script ManagerTest.php contient un ensemble de tests unitaires destinés à être exécuter avec PHPUnit pour une classe Manager. Un manager est un employé qui a sous ses ordres un ensemble d'employés (ses subordonnés).

la classe Manager (fichier Manager.php) qui hérite de Employee, im- plémente l'interface IManager.php. Les subordonnés d'un manager pourront être stockés sous forme d'un tableau contenant leurs identifiants.

→ une classe Team (fichier Team.php) qui permet de stocker des employés et des managers. Dotez Team d'une méthode d'affichage des employés et managers. Pour un manager, on affichera le nom des employés qu'il a sous ses ordres.

→ un script employee_reflex1.php qui affiche les informations suivantes concernant un employé en utilisant les fonctions de réflexion (get_object_vars, . . .) :

• nom de la classe.

• nom de la classe parente. • nom des champs et valeurs.