M1 SIS
Examen de contrôle continu n°1
Octobre 2011

Lisez le sujet en entier avant de commencer.
Vous n'avez droit qu'à la javadoc comme documentation.

Des exemples d'utilisation des classes demandées sont donnés en italique et doivent être implanté dans une classe exécutable Test.

Les résultats est à envoyé sous la forme d'une archive (zip ou tgz) à bruno@univ-tln.fr

1°/Des employés.
Écrire les classes Java nécessaires pour représenter le problème suivant : Les personnes ont un identifiant numérique et un nom. Un employé est une personne qui a un salaire horaire (variable d'un employé à l'autre). Un ingénieur est un employé qui touche en plus de son salaire une prime (variable d'un ingénieur à l'autre) et une indemnité (la même pour tous les ingénieurs). Les employés et les ingénieurs ont une méthode int calculerSalaire(int nbHeures) qui retourne leur salaire. Chaque classe n'aura qu'un seul constructeur qui prend tous les paramètres utiles et les accesseurs/modificateurs.

Employe e1 = new Employe(1,« Pierre Durand »,12) ;
Employe e2 = new Employe(2,« Pierre Dupond»,14) ;
Ingenieur i1 = new Ingenieur(3,« Marie Martin »,45,100) ;
Ingenieur i2 = new Ingenieur(4,«Laurent Durant »,50,150) ;

2°/Leur identifiants.
Modifier vos classes pour calculer automatiquement l'identifiant des personnes et être sur qu'ils sont uniques.

Employe e1 = new Employe(« Pierre Durand »,12) ;
Employe e2 = new Employe(« Pierre Dupond»,14) ;
Ingenieur i1 = new Ingenieur(« Marie Martin »,45,100) ;
Ingenieur i1 = new Ingenieur(«Laurent Durant »,50,150) ;

3°/Une mission.
On souhaite maintenant représenter une mission. Une mission est un travail qui dure un certain nombre d'heures et qui est réalisé par une liste d'employés (qui peuvent être ingénieurs ou non) et qui nécessite du matériel (voiture, pioche, ordinateur). Chaque matériel a un identifiant, un coût horaire (donnés dans le constructeur) et une méthode int getCout(int nbHeures). Les personnes et les matériels n'appartiennent pas à la même hiérarchie d'héritage (ie le seul ancêtre comment est Object).

Ecrire les classes voiture, pioche, ordinateur et Mission. Vous utiliserez les notions d'aggrégation et/ou de composition et les Collections.
Matériel[] materiel = {new Voiture(1,30), new Pioche(2,5), new Ordinateur(3,15)}
System.out.println(materiel[2.getCout()]);
Mission mission = new Mission() ;
mission.addEmploye(e1) ;mission.addEmploye(e2) ;
mission.addEmploye(i1) ;mission.addEmploye(i2) ;

mission.addMateriel(materiel[0]) ;mission.addMateriel(materiel[1]) ;
mission.addMateriel(materiel[2]) ;

4°/La facture d'une mission.
Une facture est associée à une mission et concerne un nombre d'heures (donnés dans le constructeur). Une facture s'applique à une liste d'**ItemDeFacture** (qui sont des employés ou du matériel mais qui doivent avoir une méthode **int getTarif(nbHeures)**). Une facture possède une méthode getTotal() qui retourne la somme des tarifs des items pour le nombre d'heures indiqué dans la facture.

Sans modifier la hiérarchie d'héritage précédente :
a- ajouter ce qu'il faut pour que les employés et les matériels puissent être utilisés comme des ItemDeFacture (ie soient polymorphes sans avoir d'autre ancêtre commun que Object). Le tarif d'un employé est son salaire plus une marge de 10% et celui d'un matériel est son coût plus une marge de 20% plus des frais fixes de 15€.
b- ajouter à la classe Mission une méthode **List<ItemDeFacture> getItemDeFacture()**
c- écrire la classe Facture.

Facture facture = new Facture(mission.getItemDeFacture (),50) ;
System.out.println(facture.getTotal()) ;

5°/Délégation par aggrégation.
Il doit maintenant être possible d'invoquer la méthode **int getTarifReduit(int pourcentage, int nbHeures) {return (int) getTarif(nbHeures)*pourcentage/100;}** sur tous les ItemDeFacture.

Donner une solution pour factoriser ce code en utilisant la délégation par aggrégation.

System.out.println(facture.getTarifReduit()) ;