Examen

20 janvier 2012

4 pages – Durée 3h – Responsable : E. Waller
Seul document autorisé : une feuille (recto/verso)A4 manuscrite
Si un point de l'énoncé vous semble personnellement non clair, consultez immédiatement un enseignant.

Le but des exercices (sauf le dernier) est d'écrire un programme Java complet pour le cahier des charges fourni. Ce programme est construit progressivement dans des exercices successifs autonomes. Chaque exercice dépend donc du précédent, et il est conseillé de les faire dans l'ordre (sauf le dernier qui est indépendant), mais ce n'est pas obligatoire. *Remarque* : L'énoncé a été conçu pour qu'il soit possible de programmer les exercices dans l'ordre de façon à ce que chaque nouvel exercice ne fasse que rajouter du code sans modifier (ou presque pas) du code existant. *Attention* : Chaque exercice doit être *obligatoirement* rédigé séparément : il n'est pas possible de donner un seul code contenant plusieurs exercices.

Le barème est indicatif. Pour chaque exercice il correspond à la fois à la difficulté et/ou au temps nécessaire pour faire cet exercice.
Important: Utilisez chaque fois que c'est pertinent les concepts vus dans le module. Faites le meilleur choix à chaque fois, et suivez les consignes du cours. Vérifiez soigneusement les programmes que vous écrivez afin qu'ils fonctionnent tels quels. Suivez les principes de qualité vus dans le module.
Remarques : Les ratures sur les copies ne seront pas pénalisées. Il est donc possible de ne pas utiliser de brouillon, tout faire directement sur la copie, puis si nécessaire effectuer clairement des ratures et/ou ajouts. Ne recopiez pas inutilement du code (indiquez-le plutôt par exemple en numérotant les lignes puis en indiquant les numéros).

Organisation des copies *obligatoire* : Vous devez changer de copie à chaque exercice. Un point sera attribué si cette organisation est *rigoureusement* respectée.

Cahier des charges : Personnes, employés, étudiants, voitures, impôt

Données On considère des personnes et des voitures.

Une voiture a une marque et une couleur (pour simplifier on ne considère d'autre propriété).

Il y a trois sorte de personnes.

Des personnes simples avec un nom, une année de naissance (entier), un âge (entier) qui est l'année courante moins l'année de naissance, le montant d'un impôt qui est deux fois leur âge, et une voiture.

Des employés, avec un nom, une année de naissance, un âge, un salaire (entier, qui peut varier d'un employé à l'autre), un impôt égal à deux fois leur salaire, et une voiture.

Des étudiants, avec un nom, une année de naissance, un âge, les noms des modules qu'ils suivent (pas nécessairement tous les mêmes), une voiture, et un impôt égal à dix euros plus le nombre de modules suivis, plus encore 100 euros sauf s'il n'a pas de voiture.

Il n'y a pas d'étudiant employé, ni d'employé étudiant.

On veut pouvoir accéder, à partir d'une personne, à sa voiture, mais on n'a jamais besoin d'accéder, à partir d'une voiture, à l'un de ses propriétaires.

On veut pouvoir accéder, à partir d'un étudiant, à ses noms de module, mais on n'a jamais besoin d'accéder aux étudiants à partir de ces noms.

On n'a jamais besoin d'accéder aux noms de modules sans passer par les étudiants.

Précisions et restrictions Il peut y avoir plusieurs voitures avec la même marque et la même couleur. Une personne a au plus une voiture, et peut ne pas en avoir. Une voiture peut avoir plusieurs propriétaires, ou aucun.

Pour simplifier on suppose ce qui suit. Tous les noms de personne sont différents. Le nombre maximal de personnes, de voitures, et le nombre de modules de chaque étudiant, est fixé au début de l'exécution du programme. Enfin, pour simplifier, on ne fera aucun vérification (noms, années, etc.).

Traitements considérés On veut pouvoir effectuer les traitements suivants.

1. Afficher toutes les informations de l'application : d'abord l'ensemble des personnes, puis ensuite l'ensemble des voitures ; lors de l'affichage d'une personne, on n'affichera de sa voiture que sa marque.
2. Afficher le montant total des impôts qui sera perçu par l'état.
3. Afficher, depuis le *main* et *sans fonction*, tous les noms de module existants (certains noms pourront être répétés).
4. Incrémenter de 1 l'année courante.

Exemple On considère l'exemple suivant.

Dédé né en *1987*, n'a pas de voiture.
Rita, née en *1988*, a une *Rolls-Royce crème*.
Riton, né en *1987*, employé gagnant *2500* euros, est copropriétaire avec *Rita* de la *Rolls-Royce crème*.
Lulu, née en *1986*, étudiante en *musique* et *langues*, a une *Ferrari rouge*.
Il y a une *2CV verte* n'appartenant à personne. L'année courante est 2012. Il y a au plus 888 personnes et 666 voitures.

Saisie et création des données Les données traitées par votre programme seront exactement celles de l'exemple ci-dessus. Certaines seront saisies sur la ligne de commande, et les autres figureront "en dur" dans votre programme.

Attention : En dehors de la saisie et de la création des données, qui est très simple car pour simplifier l'exemple est très simple, le reste de votre programme doit fonctionner *quelles que soient* les données considérées.

Attention : Dans chaque exercice, votre programme devra *obligatoirement* être appelé comme suit, où les paramètres sont le nom et l'année de naissance du propriétaire de la *RollsRoyce crème*, la couleur de la *Ferrari* de *Lulu*, l'année courante, et le nombre maximum de voitures.

```
java Examen Rita 1988 rouge 2012 666
```

Exercice 1 (6 points)

Attention : rédigez sur une nouvelle copie.

Rappel : Dans tous les exercices, vous devez utiliser l'appel Java complet donné dans le cahier des charges.

Dans cet exercice, on considère uniquement des personnes simples avec toutes leurs informations, sauf la voiture. Pour simplifier, on ignore les informations concernant les voitures passées sur la ligne de commande. Vous ne saisirez et créerez donc dans cet exercice que les données correspondant à cela.
Vous rangerez les personnes dans un tableau appelé *lesPersonnes*.

Donnez un programme complet gérant cela, et incluant les traitements 1 et 2.

Exercice 2 (3)

Attention : rédigez sur une nouvelle copie.

Dans cet exercice on considère, en plus des personnes simples, les voitures. On reprend l'exercice 1, qu'on adapte.

Donnez un programme complet gérant cela, et incluant les traitements 1 et 2.
(*Rappel* : Ne recopiez pas inutilement du code de l'exercice précédent.)

Exercice 3 (8)

Attention : rédigez sur une nouvelle copie.

Dans cet exercice, on considère la totalité du cahier des charges. On reprend l'exercice 2, qu'on adapte.

1. Donnez le programme complet gérant cela (incluant les traitements 1, 2 et 3). Vous pourrez au choix : soit donner la totalité du code, soit donner un croquis comme indiqué ci-dessous.
   (*Rappel* : Ne recopiez pas inutilement du code ou des informations de l'exercice précédent.)

   Contenu du croquis : Croquis global suffisament détaillé de l'application complète. Indiquez la hiérarchie des classes, les champs avec leur type, les noms des fonctions/méthodes avec le cœur de leur code en abrégé. Indiquez ce qui est statique ou dynamique.

2. Expliquez précisément et en détail le fonctionnement de la fonction effectuant le traitement 2 (2 à 3 lignes). Expliquez précisément et en détail le fonctionnement de la partie concernant les personnes de la fonction effectuant le traitement 1 et de celles qu'elle appelle par imbrication (5 à 6 lignes).

Exercice 4 (2)

Attention : rédigez sur une nouvelle copie.

On considère l'exécution du programme suivant (il compile sans erreur).
1. Donnez la configuration de la mémoire immédiatement après l'exécution de la ligne contenant le commentaire : `// ici`, et dites ce qui s'est affiché. Vous utiliserez un croquis détaillé et l'algorithme vu en cours. En particulier, on allouera obligatoirement les cases mémoire dans l'ordre croissant à partir de 101. Comme en cours, si une case contient successivement plusieurs valeurs, on les écrira de gauche à droite dans la case en les barrant d'un seul trait au fur et à mesure. De même, on ne réutilisera pas les zones de fonctions.
2. Poursuivez le déroulement sur le même croquis en utilisant une autre couleur, donnez la dernière configuration de la mémoire avant l'arrêt du programme, et dites ce qui s'est affiché.

```
// MemoireExa1_2011.java

class Machin {
    int valeur;
}
class Bidule {

    int valeur;
    Machin obj;

    static int f (Bidule b, Bidule c) {
        int n = 4;
        Memoire.d = n;
        b.obj = new Machin();
        b.valeur = Memoire.d;
        Memoire.d++;
        c.obj = b.obj;
        c.obj.valeur++;
        return b.valeur;
    }
}
class Memoire {

    static int d;

    public static void main (String[] args ) {
        int n = 1;
        Bidule[] t = new Bidule[3];
        t[n] = new Bidule();
        t[n-1] = t[n];
        t[n].obj = new Machin();
        t[n-1].obj.valeur++;
        n--;
        int m = Bidule.f(t[n], t[n+1]);
        n++;
        System.out.println(m+" "+n+" "+d);
        System.out.println(t[n-1].valeur+" "+t[n-1].obj.valeur);
        System.out.println(t[n].valeur+" "+t[n].obj.valeur); // ici
        n--;
        m = Bidule.f(t[n-1], t[n]);
        System.out.println(m+" "+n+" "+d);
        System.out.println(t[n-1].valeur+" "+t[n-1].obj.valeur);
        System.out.println(t[n].valeur+" "+t[n].obj.valeur);
    }
}
```