Héritage (TD 10)

Exercices table : 1.
Rappel : Respectez les consignes de la page *Fonctionnement* et celles des feuilles précédentes.

Attention : A partir de ce bloc et dans toute la suite du module, on écrira tout directement avec des fonctions. Mais bien sûr, il faut aussi tout savoir faire sans fonctions.

0. (a) Placez-vous dans le répertoire adéquat demandé dans la page *Environnement.*
   (b) **Familiarisation** Tapez et exécutez le programme suivant de la feuille d'exemples du cours, en respectant les consignes de la page *Fonctionnement* : *Ex2.java.*

1. Dans cet exercice on considère le cahier des charges du TD mais avec les restrictions suivantes.

   - Un client a un seul compte, simple ou négocié. Sur la ligne de commande, on ne considère donc que le premier compte de chaque client.
   - Pour les comptes négociés, les fonctions d'affichage et de calcul de l'autorisation seront appelées respectivement *afficherNegocie* et *autorisationNegociee*, ceci afin de ne pas gérer aujourd'hui la surcharge en présence d'héritage.
     Pour simplifier dans ce TD, dans l'affichage d'un compte négocié, on l'affichera comme un compte simple (y compris avec son autorisation de compte simple), mais suivi de ses spécificités de compte négocié.
   - Pour simplifier dans ce TD, lors du parcours du tableau des comptes (affichage, autorisation maximale), un compte négocié sera considéré sans ses spécificités, c'est à dire comme s'il était un compte simple.
   - Remarque : il n'y a bien sûr plus de création "en dur" des comptes négociés, tous les comptes sont saisis sur la ligne de commmande comme spécifié dans le cahier des charges.

   (a) **Programmation** A partir du corrigé du TD précédent, reprogrammez-le en utilisant le nouveau matériel chaque fois que c'est pertinent, et justifiez à chaque fois. Suivez la progression suivante, en transformant une seule classe à la fois.
      i. Peut-on appliquer la méthodologie de factorisation par héritage vue en cours ? Si oui faites-le. Dans tous les cas justifiez.
      ii. Comptes simples seuls.
      iii. Comptes négociés. Peut-on ranger tous les comptes dans le même tableau ? Si oui faites-le. Dans tous les cas justifiez.
      iv. Dans le *main*, déclarez une variable de type *CompteNegocie*, affectez-lui un *CompteNegocie*, et appelez *afficherNegocie* et *autorisationNegociee* dessus. Expliquez le déroulement en détail. Conclusion ?
      v. Peut-on appeler ces deux fonctions dans la boucle de parcours du tableau ? Justifez. Essayez. Conclusion ?
      vi. Clients.
      vii. Remarquez que l'affichage de votre programme est exactement celui du TD précédent. Pourquoi ?

   (b) **Discussion**

i. Indiquez tous les appels de méthode surchargée de votre code, et pour chacun comment Java résoud la surcharge (et ce que signifie "resoudre la surcharge pour un appel"). Y a-t-il une différence avec le TD précédent ?

ii. Indiquez, par rapport au corrigé du dernier TD, si votre code est plus satisfaisant, en explicitant les critères considérés.

iii. Indiquez tous les points de votre code dont la programmation n'est pas satisfaisante, en explicitant les critères considérés (non à cause de vous mais à cause du matériel Java vu qui est encore insuffisant).

2. Même exercice, mais un client a son ensemble de comptes, chaque compte étant simple ou négocié. On conserve les contraintes simplificatrices de l'exercice précédent.

3. *Déroulement mémoire. Travail personnel.*
On considère l'exécution du programme suivant (il compile sans erreur).

   (a) Donner la configuration de la mémoire immédiatement après l'exécution de la ligne contenant le commentaire : `// ici`. Vous utiliserez un croquis détaillé et l'algorithme vu en cours et TD. En particulier, on allouera obligatoirement les cases mémoire dans l'ordre croissant à partir de 101. Comme en cours et en TD, 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.

   (b) Donner le dernier état de la mémoire avant l'arrêt du programme.

   (c) Dire ce qu'affiche ce programme.

```
// MemoireHeritage.java

class Bidule {
    int valeur;

    int g (int n) {
        n--;
        this.valeur += n;
        return this.valeur;
    }
}

class BiduleSpecial
    extends Bidule {

    int autreValeur;
}

class Memoire {
    public static void main (String[] args ) {
        Bidule[] t;
        int n = 2;
        n--;
        t = new Bidule[n+1];
        BiduleSpecial bs = new BiduleSpecial();
        bs.autreValeur = 9;
        t[n-1] = new Bidule();
        t[n] = bs;
        n--;
        System.out.print(t[n].g(n));
        System.out.println(" "+t[n].valeur+" "+n); // ici
        t[n] = t[n+1];
        System.out.print(t[n].g(n));
        System.out.println(" "+t[n].valeur+" "+n+" "+bs.autreValeur);
        System.out.print(bs.g(n));
        System.out.println(" "+bs.valeur+" "+n+" "+bs.autreValeur);
    }
}
```

4. *Entraînement. Travail personnel.* Comme aux TD précédents, reprogrammez l'examen de l'an passé, en utilisant tout le matériel vu jusqu'à présent chaque fois que c'est pertinent, et justifiez à chaque fois. Indiquez tous les points de votre code dont la programmation n'est pas satisfaisante, en explicitant les critères considérés (non à cause de vous mais à cause du matériel Java vu qui est encore insuffisant).

   Rédigez l'examen sur papier, puis tapez-le : il doit fonctionner du premier coup. Certains concepts n'ont pas encore été vus, ce qui rendra certaines parties de code un peu lourdes, mais c'est peu.