Programmation objet TD 2
Encapsulation, Constructeurs, Méthodes

Exercice 1
-Qu'est ce qu'un message à soi même ?
-A quoi sert typiquement un constructeur ?
-Quel est le modificateur le plus restrictif pour restreindre l'accès à une méthode ?

Exercice 2
Nous désirons modifier les classes écrites lors du TD 1 (exercice 3) afin de les rendre plus robustes. Voici une partie du code des classes utilisées lors du TD1:

```
classe Vehicule
 champs modele: chaine, dateAchat: nbe, prixAchat: nombre,
  numeroImmatriculation: chaine, permis: chaine, fin Vehicule
 méthode coutLocation(): nombre si self<=age() < 365 alors
        Retourner prixAchat / 200;sinon Retourner prixAchat / 250
  fin coutLocation
classe Voiture sous classe de Vehicule,
 champs autoradio: booléen, fin Voiture
classe Camion sous classe de Vehicule,
 champs volume: nombre fin Camion
classe Autocar sous classe de Camion,
 champs places: nombre fin Autocar

...
```

1. Aucun constructeur n'a été fourni dans le code des classes. Ecrire un constructeur, en JAVA, pour chacune des classes, qui prenne suffisamment de paramètres pour initialiser complètement l'instance créée.
Combien la classe `vehicule` a de constructeurs, en JAVA ? en LOLO si les constructeurs avaient été programmés en LOLO ?
2. Dans un programme test, construire un autocar en l'instantiant : quelles sont les méthodes qui ont été utilisées ?
3. Les classes de l'application sont appelées à être développées par plusieurs programmeurs, et à évoluer.
Quelle solution proposez vous, en ce qui concerne les champs, pour assurer cette gestion ?
On considérera dorénavant être dans ce cas.
4. Un changement dans le calcul des coûts de location est demandé : Dans le cas d'un `autocar` de plus de 40 places, le coût est majoré de 50 € par rapport au calcul du coût habituel. Fournissez le code idoine.
5. Ecrire une méthode *affiche* adaptée à chaque type de véhicule.
On suppose avoir une méthode universelle (sur tout objet) `ecrire()`

Exercice 3
Nous désirons maintenant programmer la gestion des agences appartenant à l'entreprise de location de véhicules. Plus précisément, il existe deux types `d'agences` : les `bureaux` de location, ouverts aux clients et dans lesquels les véhicules peuvent être loués, et les `garages`, dans lesquels la maintenance et les réparations sont effectués. Toutes les agences ont un `nom` et un numéro de `téléphone`. Différents personnels sont affectés dans les différentes agences : un nombre d'`administratifs` est affecté dans tous les types d'agences, alors qu'un nombre de `commerciaux` est affecté uniquement aux bureaux de location, et un nombre de `mécaniciens` est affecté uniquement aux garages.

Chaque bureau de location peut envoyer des véhicules en réparation dans un ensemble de garages, choisis par la direction générale (au plus 4 garages par bureau).
En LOLO la classe ensemble existe ; on peut utiliser les méthodes ensemblistes : union(_), intersection(_), cardinalité(), ajouteElt(_), enleveElt(_), de même que des boucles "pour chaque élément e de Ensemble Faire ...e<=...". Ces opérateurs s'appliquent en prenant en paramètre un ensemble ou un élément. La classe ensemble contient deux constructeurs, un constructeur sans paramètre qui rend un ensemble vide, et un constructeur a un paramètre qui forme un ensemble constitué du singleton passé en paramètre.

1. Dessiner l'arbre des classes avec les champs, puis écrire ces classes (chaque champ sera déclaré *privé*).
2. On veut ajouter à chaque Agence, un champ commentaire que n'importe qui peut utiliser comme bon lui semble. Proposer la modification nécessaire.
3. Pour chaque classe, écrire un constructeur permettant d'initialiser complètement l'instance construite; on suppose que l'ensemble des garages associée à un bureau est toujours initialisée à l'ensemble vide.
4. Ecrire une (ou des) méthode(s) taillePersonnel() retournant le nombre de personnes affectées à n'importe quelle agence.
5. Ecrire une méthode ajoutGarage de la classe représentant les bureaux de location permettant de rajouter un garage (passé en paramètre) au bureau. Ecrire une méthode retraitGarage permettant le retrait d'un garage.
6. Ecrire une méthode ajoutBureau de la classe représentant les garages (Garage) permettant de rajouter ce garage à un bureau de location passé en paramètre. Ecrire une méthode retraitBureau permettant le retrait.
7. Ecrire une méthode nbGarages retournant le nombre effectif de garages connus par un bureau de location.
8. Ecrire une méthode nbMecaniciens calculant le nombre de mécaniciens disponibles à partir d'une agence : pour un garage, il s'agit du nombre de mécaniciens de ce garage, et pour un bureau de location, il s'agit de tous les mécaniciens affectés aux garages connus par ce bureau de location.

Exercice 4
a) Le programme suivant est-il correct ? Si oui, que donne-t-il ?
b) Pourrait-on remplacer l'accès protected sur age et masculin par un accès privé, et pourquoi ?

```
class Homme{
   protected int age;
   protected boolean masculin;
   Homme(){
      age = 0; masculin = false; }
   Homme(boolean masc){
      age = 20;
      if (masc){masculin = true;}
      else {masculin = false;}}
   public void direBonjour(){;};
   public int getAge(){return age;}
   public boolean getMasc(){return masculin;}}

class Francais extends Homme{
   Francais(){super();}
   Francais(boolean masc){super(masc);}
   public void direBonjour(){System.out.println("Bonjour");}}

class Anglais extends Homme{
   Anglais(){super(); }
   Anglais(boolean masc){ super(masc);}
```

```
    public void direBonjour(){   System.out.println("Good morning ");}}

class Hollandais extends Homme{
   Hollandais(){super();}
   Hollandais(boolean masc){super(masc);}
  public void direBonjour(){System.out.println("Goeie dag ");}}

public class Bonjour{
   public static void main(String[] argv){
      Homme personnes[] = new Homme[3];
      personnes[0] = new Anglais();
      personnes[1] = new Hollandais(false);
      personnes[2] = new Francais(true);
      for (int i = 0; i < 3; i++){
         personnes[i].direBonjour();
         System.out.println("J'ai " + personnes[i].getAge() + " ans.");
         System.out.print("Je suis ");
         if (personnes[i].getMasc()){System.out.println("un homme.");}
         else{System.out.println("une femme.");}}}}}
```

Exercice 5 Partage d'objets (Charon chap 4)

Qu'a-t-on en mémoire avec le programme suivant ?

```
class C {int n;}

class A {
    C c = new C();
    A() {System.out.println("c dans A : " + c);}}

class B {
    C c;
    B(C _c) {
         c = _c; System.out.println("c dans B : " + c);}
    void valuer(int x) {
         c.n = x;}}

class Partage {
    public static void main(String[] arg) {
         A a = new A();
         B b =  new B(a.c);
         b.valuer(2);
         System.out.println(a.c.n);}}
```

Exercice 6 (Charon chap 4)

Que donne l'exécution du programme suivant:

```
class Surcharge {
         int n;
         double x;
  Surcharge() {n = 1; x = 3.5;}
  Surcharge(int n, double x) {this.n = n; this.x = x;}
  int operation(int p) {return 10*p + n;}
  double operation(double y, int p) {return x*p + y;}
  double operation(int p, double y) {return (double)n/p + y;}}

class EssaiSurcharge {
         public static void main(String[] arg) {
                  Surcharge surcharge;
                  surcharge = new Surcharge();
                  System.out.println(surcharge.operation(2));
                  System.out.println(surcharge.operation(1.5, 4));
                  System.out.println(surcharge.operation(4, 1.5));
                  surcharge = new Surcharge(7, 2.0);
                  System.out.println(surcharge.operation(2));}}
```