Sujet de Juin 2008 – Première session
Indications de correction

Première partie

1. On distingue deux niveaux :

- l'objet `Echantillon`, qui contient au moins des **attributs** pour le nom de la plante (une chaîne de caractères) et le lieu (chaîne de caractères ou nombre entier selon un code ou nombres représentant la position dans un repère donné, etc.) et la date de récolte (nombres).
- l'objet `Collection`, qui contient (par **agrégation**) une suite d'échantillons, *a priori* sous forme d'un **tableau d'objets** de la classe `Echantillon`.

Pour chaque classe, on doit spécifier les **méthodes** habituelles, notamment **constructeurs**, accès *via* des méthodes de lecture/affichage d'un échantillon, et accès aux échantillons pour la collection.

2. **Spécialisation** de la classe `Echantillon` sous forme `EchantillonParasite`, avec ajout d'un attribut « `planteHote` ». **Surcharge** des méthodes saisie/affichage.

La définition de la classe `Collection` ne change pas.

3. L'échantillon contient le nom, le lieu et l'étage géologique. La collection reste une suite d'échantillons.

4. La classe `Echantillon` devient **abstraite**, et est **dérivée** en `EchantillonBotanique` et `EchantillonGeologique`. L'`EchantillonBotanique` est lui-même **dérivé** en `EchantillonBotaniqueParasite`.

La classe `Collection` reste une suite d'`Echantillons`.

Deuxième partie

1. Exemple de classe `Dessin` :

```
import java.util.*;
import java.awt.*;
class Dessin {

  private static void delay (int ms) {
    long time = System.currentTimeMillis();
    while (System.currentTimeMillis() - time < ms);
  }

  public static void main(String args[]) {

    Frame fenetre = new Frame ("10 carres");
    fenetre.setSize (400, 400);
    fenetre.setVisible (true);
    Graphics g = fenetre.getGraphics();
    delay (200);
    g.setColor (Color.white);
    g.fillRect (0,0,400,400);

    Carre[] Carres = new Carre[10];
    Random r = new Random();
    int x,y;
```

```
    for(int i=0;i< 10;i++) {
      x = r.nextInt(400);
      y = r.nextInt(400);
      Carres[i] = new Carre(x,y,10);
      Carres[i].dessine(g);
    }
  }
}
```

2. On ajoute aux attributs de `Carre` (par exemple après `protected int cote;`) :

```
protected Color couleur;
```

On donne à cet attribut sa valeur par défaut en ajoutant au constructeur :

```
couleur = Color.black;
```

Et on crée une méthode pour modifier la valeur de cet attribut :

```
public void setCouleur(Color cl) {
      couleur = cl;
}
```

Pour que le carré soit dessiné avec la bonne couleur, on remplace `g.setColor(Color.black);` dans la méthode `dessine(...)` par :

```
      g.setColor (couleur);
```

Dans `Dessin.java`, on ajoute une variable pour la couleur, et une pour le tirage aléatoire d'un nombre décimal, par exemple avant `int x,y` :

```
Color c;
double h;
```

Dans la boucle qui crée et place les carrés, on ajoute :

```
h = r.nextDouble();
if(h<0.25)
  c = Color.blue;
else
  c = Color.red;
```

Et on attribue cette couleur, juste après avoir créé le carré (`Carres[i] = new Carre(x,y,10);`) :

```
Carres[i].setCouleur(c);
```

NB. On peut aussi modifier le constructeur de `Carre` (ou en créer un second) pour attribuer la couleur dès la création d'une instance :

```
public Carre (int x, int y, int taille, Color cl) {
      xCentre = x;
      yCentre = y;
      cote = taille;
      couleur = cl;
}
```

Il n'y a plus alors d'appel à `setCouleur(...)`, mais la création de chaque carré se fait par :

```
Carres[i] = new Carre(x,y,10,c);
```

3. On ne sait pas, tant que tous les carrés n'ont pas été créés, lequel sera le plus petit (bleu) et lequel sera le plus grand (rouge). On ne peut donc pas les dessiner au fur et à mesure de leur création. Il faut faire deux boucles, par exemple :

```
int t[] = new int[10]; // tableau pour les tailles
int s=0,l=0;           // indices du plus petit et plus grand carré
for(int i=0;i< 10;i++) {
  t[i] = 5 + r.nextInt(21);
  if(t[i]<t[s])
    s=i;
  if(t[i]>t[l])
    l=i;
  x = r.nextInt(400);
  y = r.nextInt(400);
  Carres[i] = new Carre(x,y,t[i]);
}
```

```
for(int i=0;i< 10;i++) {
  if(i==s)
    Carres[i].setCouleur(Color.blue);
  if(i==l)
    Carres[i].setCouleur(Color.red);
  Carres[i].dessine(g);
}
```

Troisième partie

On part d'un tableau **t**, qui comporte **nc** cases. Chaque case contient un nombre entier, qui est soit pair soit impair.

Pour savoir si un nombre **N** est pair ou impair, deux possibilités :

- on effectue une division entière par 2 (c'est ce que fait Java quand on divise deux entiers entre eux). Supposons que **N/2** donne alors **Q**. Si **2*Q==N** alors **N** est pair.
- on effectue une division par 2 en nombres décimaux, et on soustrait au résultat sa partie entière (en Java, la fonction `Math.floor()` donne la partie entière d'un réel). Si **(N/2)-|N/2|==0**, alors **N** est pair.

On suppose donc qu'en utilisant une de ces deux méthodes, on peut obtenir une valeur de vérité `vrai` ou `faux` pour une proposition telle que « **x est pair** ».

Une idée pour un algorithme est d'utiliser deux indices, **premP** et **dernI**, correspondant respectivement au premier nombre pair et au dernier nombre impair.
Si **premP** < **dernI**, comme tout nombre pair doit être placé après tout nombre impair, on échange les deux. On met alors à jour les deux indices, et ainsi de suite jusqu'à ce que le premier pair soit après le dernier impair.

premP = indice du premier nombre pair

dernI = indice du dernier nombre impair

tant que (premP<dernI)
    échanger les valeurs de t[premP] et t[dernI]
    recalculer premP et dernI

Ce qui donne, en formalisant un peu plus et en faisant attention aux bornes :

premP = 0
tant que (premP<nc) et (t[premP] est impair)
    premP++

dernI = nc-1
tant que (dernI≥0) et (t[dernI] est pair)
    dernI--

tant que (premP<dernI)
    temp = t[premP]
    t[premP]=t[dernI]
    t[dernI]=temp
    tant que (premP<nc) et (t[premP] est impair)
        premP++
    tant que (dernI≥0) et (t[dernI]est pair)
        dernI--