IUT de Paris Descartes – Base de la Programmation Objet

Travaux Pratiques – Sujet n°4

Objectifs

- Définir des constructeurs
- Définir et vérifier des préconditions

Le problème des tours de Hanoï

Ce problème est un jeu de réflexion imaginé par le mathématicien français Édouard Lucas en 1883 qui consiste à déplacer des disques de diamètres différents d'une tour de "départ" à une tour d' "arrivée" en passant par une tour "intermédiaire" et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois,
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ.

Le problème serait originaire d'Inde où une légende raconte que dans un temple où se trouvent trois poteaux sur lesquels s'empilent 64 disques d'or, les prêtres de Brahmâ déplacent continuellement les disques selon les règles édictées ci-dessus. D'après cette même légende, le dernier déplacement de disque marquera la fin du monde. Cette légende existe sous plusieurs versions et dans différentes religions.

Un jeu à 64 disques requiert un minimum de $2^{64} - 1$ déplacements. En admettant que le déplacement d'un disque prenne 1 seconde (cela fait 86400 déplacements par jour), la fin du monde aurait lieu au bout de 2.14×10^{14} jours, ce qui nous fait à peu près 600 milliards d'années, soit une quarantaine de fois l'âge estimé de l'univers (15 milliards d'années selon certaines sources). Ouf !

La résolution algorithmique de ce problème est souvent employée pour illustrer la puissance d'expression de la récursivité. En effet, pour déplacer n disques d'une tour vers une autre tout en sachant que nous disposons d'une troisième, il suffit de déplacer $n - 1$ disques de la première vers la troisième, de bouger le dernier disque de la première vers la seconde, puis enfin de déplacer les $n - 1$ disques de la troisième vers la seconde.

1. Le programme qui vous est fourni met en œuvre cet algorithme. Il consiste en un programme principal et deux fonctions annexes (`Appli.java`) ainsi qu'une classe (`Tour.java`).
   (a) Créez un projet et importez-y les deux fichiers.
   (b) Lisez le programme principal et la méthode statique `déplacer`. Vous noterez que le programme principal nécessite d'être complété et que `déplacer` invoque `bouger`. Le code de cette méthode de classe est à compléter dans la question 6. Son rôle est de déplacer le disque se trouvant au sommet de la tour "`de`" vers la tour "`vers`".
2. Vous noterez que dans le programme principal, l'initialisation du tableau `tours` est facilitée si nous pouvons construire des tours pleines et des tours vides. Ajoutez un second constructeur à la classe `Tour`. Il doit remplir la tour de disques de diamètres N, $N - 1$, ..., 1. Vous réécrirez le code du premier constructeur de manière à ce qu'il invoque le nouveau.

Solution:

```
public class Tour {
  /**
   * Construit une tour portant <code>nbd</code> disques de diamètres
   * respectifs <code>nbd</code>, <code>nbd - 1</code>, ..., 1 (de la
   * base au sommet).
   *
   * @param nbd
   *            Diamètre du disque se trouvant à la base de la tour.
   */
  public Tour(int nbd) {
    assert (nbd < MAX);
    disques = new int[MAX];
    nb = 0;
    for (; nbd > 0; --nbd)
      ajouter(nbd);
  }

  /**
   * Construit une tour vide de tout disque.
   */
  public Tour() {
    this(0);
  }
}
```

3. Complétez l'initialisation du tableau `tours` dans le programme principal.

Solution:

```
public class Appli {
  ...
  public static void main(String[] args) {
    final int TAILLE = 3, N = 20;
    Tour[] tours = new Tour[TAILLE];

    tours[0] = new Tour(N);
    for (int i = 1; i < TAILLE; ++i)
      tours[i] = new Tour();
    ...
  }
}
```

4. Énumérez les méthodes devant être ajoutées à la classe `Tour` de façon à permettre la programmation de la méthode `Appli.bouger`. Donnez le prototype et la précondition de chacune. Faites en sorte que l'utilisateur de la classe dispose des moyens nécessaires à la vérification de ces préconditions avant d'invoquer les méthodes.

Solution:

```
public class Tour {
  ...
  /**
```

```
    * Ajoute un disque de diamètre <code>d</code> au sommet de la tour.
    *
    * @param d
    *            Diamètre du disque ajouté.
    */
   public void ajouter(int d) {
     assert (!estPleine() && (estVide() || sommet() > d));
     ...
   }

   /**
    * Retire le disque au sommet de la tour.
    */
   public void retirer() {
     assert (!estVide());
     ...
   }

   /**
    * Retourne le diamètre du disque se trouvant au sommet de la tour.
    */
   public int sommet() {
     assert (!estVide());
     ...
   }

   /**
    * Indique si la tour est vide.
    */
   public boolean estVide() {
     ...
   }

   /**
    * Indique si la tour est pleine.
    */
   public boolean estPleine() {
     ...
   }

   ...
}
```

5. Implémentez les nouvelles méthodes de la classes `Tour`.

Solution:

```
public class Tour {
   ...
   /**
    * Ajoute un disque de diamètre <code>d</code> au sommet de la tour.
    *
    * @param d
```

```
   *              Diamètre du disque ajouté.
   */
  public void ajouter(int d) {
    assert (!estPleine() && (estVide() || sommet() > d));
    disques[nb++] = d;
  }

  /**
   * Retire le disque au sommet de la tour.
   */
  public void retirer() {
    assert (!estVide());
    --nb;
  }

  /**
   * Retourne le diamètre du disque se trouvant au sommet de la tour.
   */
  public int sommet() {
    assert (!estVide());
    return disques[nb - 1];
  }

  /**
   * Indique si la tour est vide.
   */
  public boolean estVide() {
    return nb == 0;
  }

  /**
   * Indique si la tour est pleine.
   */
  public boolean estPleine() {
    return nb == MAX;
  }

  ...
}
```

6. Implémentez la fonction `bouger`.

Solution:

```
public class Appli {
  private static void bouger(Tour de, Tour vers) {
    assert (!de.estVide() &&
              (vers.estVide() || de.sommet() < vers.sommet()));
    assert (!de.estVide() &&
              (vers.estVide() || de.sommet() < vers.sommet()));
    int d = de.sommet();
    de.retirer();
    vers.ajouter(d);
```

```
  }
  ...
}
```