

## Programmation Orientée Objet Examen du 17 juin 2004 (2<sup>ème</sup> session)

Durée 2h

Seuls documents autorisés : documents distribués et notes personnelles de cours

### I

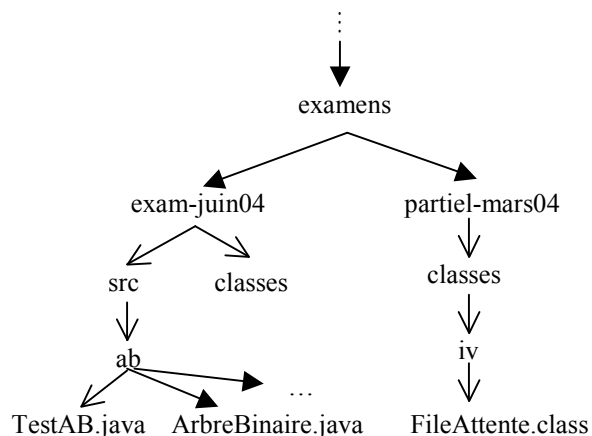
On a l'arborescence de répertoires et fichiers ci-contre :

Les classes `TestAB`, `ArbreBinaire`, etc ... sont les classes du paquetage `ab` de la question II. La classe `TestAB` est la classe principale. La classe `ArbreBinaire` utilise la classe `FileAttente` du paquetage `iv`.

1. Donner la ou les commandes *unix* compilant les fichiers `TestAB.java`, `ArbreBinaire.java` etc ... sachant que les *pseudo-codes* doivent se trouver dans le répertoire .../examens/exam-juin04/classes.

2. Donner la commande *unix* exécutant la classe principale `TestAB`.

On précisera dans quel répertoire on se trouve pour exécuter ces commandes.



### II

On rappelle ci-après les documentations des classes `ArbreBinaire` et `FileAttente`.

```
package ab;

/** Implémentation d'arbres binaires et de plusieurs représentations. */
public class ArbreBinaire {
    protected Object racine; // protected pour énumérations
    protected ArbreBinaire gauche, droite; // sous-arbres gauche et droit

    /** Crée un arbre binaire composé d'un seul noeud (cad une feuille). */
    public ArbreBinaire(Object racine) {...}

    /** Crée un arbre binaire composé de la racine spécifiée et des deux sous-arbres spécifiés */
    public ArbreBinaire(Object racine, ArbreBinaire gauche, ArbreBinaire droite) {...}

    /** Crée un arbre binaire composé de la racine spécifiée et du sous-arbre
     *  spécifié (qui sera les sous-arbres gauche et droit). */
    public ArbreBinaire(Object racine, ArbreBinaire droite) {...}

    /** Teste si cet Arbre binaire est une feuille. */
    public boolean feuille() {...}

    /** Renvoie une représentation indentée de cet arbre binaire, c'est-à-dire :
     *  * pour chaque sous-arbre il y a passage à la ligne et une indentation
     *  * proportionnelle à la profondeur de ce sous-arbre. */
    public String toString() {...}

    /** Renvoie une représentation préfixe de cet arbre binaire,
     *  * c'est-à-dire dans l'ordre :
     *  * racine puis sous-arbres gauche puis droite préfixés. */
    public String formePrefixe() {...}

    ...
}
```

```

package iv;

/** Implémente une file d'attente (premier entré, premier sorti). */
public class FileAttente {
    /** Crée une file d'attente vide. */
    public FileAttente() {...}
    /** Rentre l'objet spécifié dans cette file d'attente. */
    public void rentrer(Object o) {...}
    /** Sort le premier élément de cette file d'attente. */
    public void sortir() {...}
    /** Renvoie le premier élément de cette file d'attente. */
    public Object premier() {...}
    /** Teste si cette file d'attente est vide. */
    public boolean vide() {...}
    /** Renvoie une représentation sous forme de chaîne de cette file d'attente. */
    public String toString() {...}
}

```

On désire compléter la définition de la classe ArbreBinaire.

**1a.** Définir une méthode `formeParNiveau` qui renvoie une chaîne représentant un arbre binaire niveau par niveau.

```

/** Renvoie une représentation par niveau de cet arbre binaire, c'est-à-dire dans l'ordre :
 * racine puis noeuds de niveau 1, puis 2, etc ... */
public String formeParNiveau() {...}

```

Exemple : pour l'arbre a :

```

      +
     / \
    -   g
   / \ / \
  f  4 5.0 y
  |
  x

```

a.`formeParNiveau` renvoie la chaîne `+ - g f 4 5.0 y x`

*Aide* : On utilisera une *file d'attente* d'arbres binaires initialisée avec l'arbre binaire considéré. Tant que la file ne sera pas vide, on prendra la racine du premier élément de la file, que l'on sortira, et on ajoutera à la file ses sous-arbres non vides.

Sur l'exemple, on aura donc successivement dans la file les (sous-)arbres suivants (en notation préfixe) :

[ + - f x 4 g 5.0 y ]	donne +
[ f x 4 , g 5.0 y ]	donne -
[ g 5.0 y , f x , 4 ]	donne g
[ f x , 4 , 5.0 , y ]	donne f
[ 4 , 5.0 , y , x ]	donne 4
[ 5.0 , y , x ]	donne 5.0
[ y , x ]	donne y
[ x ]	donne x
[ ]	vide

**1b.** Définir une méthode `formeParNiveauxSepares(Object marque)` qui fait de même mais en insérant une *marque* entre chaque niveau.

```

/** Renvoie une représentation de cet arbre binaire par niveaux séparés par la marque spécifiée. */
public String formeParNiveauxSepares(Object marque) {...}

```

Exemple : pour l'arbre a précédent,

a.`formeParNiveauxSepares(" / ")` renvoie la chaîne `/ + / - g f / 4 5.0 y / x /`

a.`formeParNiveauxSepares("\n")` renvoie la chaîne `\n + \n - g f \n 4 5.0 y \n x \n`  
qui donnera à l'affichage

```

+
- g
f 4 5.0 y
x

```

*Aide* : on insérera à bon escient des occurrences de l'objet *marque* dans la file d'attente.

2. Compléter les définitions ci-dessous des classes `EnumerationParNiveau` et `EnumerationParNiveauxSepares` et les utiliser pour redéfinir les méthodes `formeParNiveau` et `formeParNiveauxSepares(Object marque)` de la classe `ArbreBinaire`.

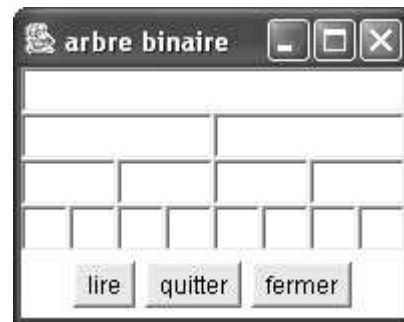
```
package ab;
import ...
/** Classe décrivant une énumération par niveau d'un arbre binaire. */
public class EnumerationParNiveau implements Enumeration {
    ArbreBinaire a;
    FileAttente f;
    public EnumerationParNiveau(ArbreBinaire a) {...}
    public boolean hasMoreElements() {...}
    public Object nextElement() {...}
}
```

```
package ab;
import ...
/** Classe décrivant une énumération d'un arbre binaire par niveaux séparés par une marque. */
public class EnumerationParNiveauxSepares extends EnumerationParNiveau {
    private Object marque;
    public EnumerationParNiveauxSepares(ArbreBinaire a, Object marque) {...}
    public Object nextElement() {...}
}
```

3.

On rappelle la définition de la classe `DessinArbre` permettant l'affichage graphique d'un arbre binaire.

On veut pouvoir *lire* un arbre binaire dans une telle fenêtre graphique.



```
package ab;
import java.awt.Frame;
import java.awt.Panel;
import java.awt.Button;
import java.awt.TextField;
import java.awt.GridLayout;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class DessinArbre extends Frame implements ActionListener {
    private ArbreBinaire a;
    private Button bQuitter = new Button("quitter");
    private Button bFermer = new Button("fermer");

    public DessinArbre(ArbreBinaire a) {
        super("arbre binaire");
        Panel p = new Panel();
        this.add("North", this.dessin(a));
        this.add("South", p);
        p.add(bQuitter); p.add(bFermer);
        bQuitter.addActionListener(this);
        bFermer.addActionListener(this);
        this.addWindowListener(new GestionnaireFermeture(this));
    }
}
```

```

/** Construit et renvoie le Panel où est dessiné l'arbre spécifié */
private Panel dessin(ArbreBinaire a) {
    Panel pArbre = new Panel();
    pArbre.setLayout(new BorderLayout());
    TextField rac = new TextField(a.racine.toString());
    pArbre.add("North",rac);
    Panel pSousArbres = new Panel();
    pArbre.add("Center",pSousArbres);
    pSousArbres.setLayout(new GridLayout(1,2));
    if (a.gauche != null) pSousArbres.add(this.dessin(a.gauche));
    if (a.droit != null) pSousArbres.add(this.dessin(a.droit));
    return pArbre;
}

/** Arrête le programme (resp. ferme la fenêtre) si on clique sur le bouton "quitter"
 * (resp. "fermer". */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == bQuitter) System.exit(0);
    else this.setVisible(false);
}
}

```

Modifier cette classe de façon à pouvoir *lire* un arbre binaire dans une telle fenêtre graphique.

Pour cela, outre

- les modifications nécessaires de la classe DessinArbre,
- la définition d'une méthode lire,
- on définira un nouveau constructeur de la classe ArbreBinaire,

et on utilisera la classe PanelArbre sous-classe de la classe Panel permettant de mémoriser, pour chaque Panel, l'arbre correspondant et ses sous-composants graphiques nécessaires.

On donne ci-après la définition complète de la classe PanelArbre

```

package ab;
import java.awt.Panel;
import java.awt.TextField;
public class PanelArbre extends Panel {
    private ArbreBinaire a;
    protected TextField tfRac; // le champ où est affiché ou lu la racine de l'arbre
    protected PanelArbre pag ,pad; // les PanelArbre des deux sous-arbres
    /** Crée un PanelArbre pour l'arbre et le champ spécifiés. */
    public PanelArbre(ArbreBinaire a, TextField tfRac) {
        this.a=a;
        this.tfRac=tfRac;
    }
}

```

la documentation du nouveau constructeur de la classe ArbreBinaire

```

/** Crée un arbre binaire vide de la hauteur spécifiée, c'est-à-dire un arbre binaire dont
    tous les nœuds sont étiquetés par la chaîne vide. */
    public ArbreBinaire(int n) {...}

```

qui servira à construire un dessin d'arbre dont tous les nœuds sont vides.

les nouveaux attributs de la classe DessinArbre

```

private PanelArbre pArbre;
private Button bLire = new Button("lire");

```

la documentation de la nouvelle méthode dessin de la classe DessinArbre

```

/** Construit et renvoie le PanelArbre où est dessiné l'arbre spécifié. */
    private PanelArbre dessin(ArbreBinaire a) {...}

```

la documentation de la nouvelle méthode actionPerformed de la classe DessinArbre

```

/** Affiche l'arbre binaire dessiné si on clique sur le bouton "lire".
 * Arrête le programme (resp. ferme la fenêtre) si on clique sur le bouton "quitter"
 * (resp. "fermer". */
    public void actionPerformed(ActionEvent e) {...}

```

la documentation de la nouvelle méthode lire de la classe DessinArbre

```

/** Renvoie l'arbre binaire dessiné dans le PanelArbre spécifié. */
    private ArbreBinaire lire(PanelArbre pArbre) {...}

```