

Programmation Orientée Objet Examen du 13 mai 2004

Durée 2h

Seuls documents autorisés : documents distribués et notes personnelles de cours

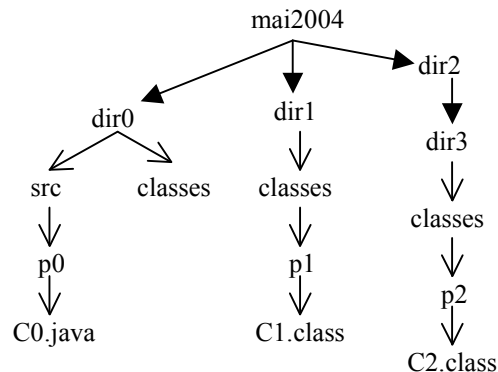
I

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

Le code source de la classes C0 est le suivant :

```
package p0;
import ...

public class C0 {
    public static void main(String[] args) {
        C1 c1=new C1(args[0]);
        C2 c2=new C2(args[1]);
        System.out.println(c1+" "+c2);
    }
}
```



1. Compléter les lignes `import ...`.
2. Donner une ligne de commandes Unix pour compiler la classes C0. Le pseudo-code devra se trouver sous le répertoire `mai2004/dir0/classes`.
3. Donner une ligne de commandes Unix pour exécuter la classe principale C0.

II

1. On donne ci-après le code d'une classe principale et le résultat de son exécution.

```
package ab;

public class TestAB {
    public static void main(String[] args) {
        ArbreBinaire a =
            new ArbreBinaire("+",
                new ArbreBinaire("-",
                    new ArbreBinaire("f",
                        new ArbreBinaire("x")
                    ),
                    new ArbreBinaire(new Integer(4))
                ),
                new ArbreBinaire("g",
                    new ArbreBinaire(new Double(5)),
                    new ArbreBinaire("y")
                )
            );
        System.out.println(a);
        System.out.println("forme préfixe : " + a.formePrefixe());
        System.out.println("forme postfixe : " + a.formePostfixe());
        System.out.println("forme infixe : " + a.formeInfixe());
        System.out.println("forme infixe parenthésée : " + a.formeInfixePar());
    }
}
```

```

+
-
  f
    x
      4
    g
      5.0
      y
forme préfixe : + - f x 4 g 5.0 y
forme postfixe : x f 4 - 5.0 y g +
forme infixe : f x - 4 + 5.0 g y
forme infixe parenthésée : ( ( ( f x ) - 4 ) + ( 5.0 g y ) )
  
```

Documenter et définir la classe `ArbreBinaire`.

2. Définir une classe `DessinArbre` et rajouter son appel dans la classe principale `TestAB` pour obtenir à l'écran la fenêtre graphique ci-dessous.



Un click sur le bouton "quitter" doit provoquer l'arrêt du programme.

Un click sur le bouton "fermer" ou sur la case de fermeture de la fenêtre doit fermer la fenêtre.

Aide : La fenêtre est une `Frame`. Le dessin de l'arbre est dans un `Panel` de deux `Panel`s : un pour la racine et un pour les sous-arbres. Le `Panel` des sous-arbres contient zéro, un ou deux `Panel`s : un pour chaque sous-arbre non vide. La méthode dessinant un arbre est ainsi récursive : pour dessiner un arbre, on appelle récursivement la méthode pour chacun de ses sous-arbres non vides.

3. a) Compléter la définition de la classe abstraite `EnumerationArbre` qui implémente l'interface `Enumeration` et de ses quatre sous-classes `EnumerationPrefixe`, `EnumerationPostfixe`, `EnumerationInfixe` et `EnumerationInfixePar` qui énumèrent les nœuds d'un arbre binaire selon les quatre formes préfixe, postfixe, infixe et infixe parenthésée.

b) Redéfinir les méthodes `formePrefixe`, `formePostfixe`, `formeInfixe` et `formeInfixePar` de la classe `ArbreBinaire` en utilisant ces énumérations.

```
package ab;
import ...;
/** Classe abstraite décrivant des énumérations d'arbres binaires. */
public abstract class EnumerationArbre implements Enumeration {
    /** l'arbre à énumérer */
    protected ArbreBinaire a;
    /** indique si la racine a été énumérée */
    protected boolean racineEnumeree = false;
    /** les énumérations des sous-arbres */
    protected EnumerationArbre eag, ead;
    /** partie commune à toutes les sous-classes du constructeur */
    public EnumerationArbre(ArbreBinaire a) {...}
    /** Teste s'il y a encore des éléments à énumérer, c'est-à-dire
     *  - si la racine n'a pas encore été énumérée
     *  - ou s'il reste des éléments à énumérer dans un des sous-arbres (non vide).
     */
    public boolean hasMoreElements() {...}
    /** Renvoie l'élément suivant de cette énumération . /
    public abstract Object nextElement();
}
```

```
package ab;
/** Classe décrivant une énumération préfixe d'un arbre binaire. */
public class EnumerationPrefixe extends EnumerationArbre {
    /** Crée une énumération préfixe de l'arbre binaire spécifié. */
    public EnumerationPrefixe(ArbreBinaire a) {...}
    /** Renvoie l'élément suivant de cette énumération, c'est-à-dire
     *  - la racine si celle-ci n'a pas encore été énumérée,
     *  - sinon l'élément suivant du sous-arbre gauche si celui-ci n'est pas vide
     *      et a encore des éléments à énumérer,
     *  - sinon l'élément suivant du sous-arbre droit si celui-ci n'est pas vide
     *      et a encore des éléments à énumérer,
     *  - sinon null.
     */
    public Object nextElement() {...}
}
```

```

package ab;
/** Classe décrivant une énumération postfixe d'un arbre binaire. */
public class EnumerationPostfixe extends EnumerationArbre {
    /** Crée une énumération postfixe de l'arbre binaire spécifié. */
    public EnumerationPostfixe(ArbreBinaire a) {...}
    /** Renvoie l'élément suivant de cette énumération, c'est-à-dire
     * - l'élément suivant du sous-arbre gauche si celui-ci n'est pas vide
     *   et a encore des éléments à énumérer,
     * - sinon l'élément suivant du sous-arbre droit si celui-ci n'est pas vide
     *   et a encore des éléments à énumérer,
     * - sinon la racine si celle-ci n'a pas encore été énumérée,
     * - sinon null.
     */
    public Object nextElement() {...}
}

```

```

package ab;
/** Classe décrivant une énumération infixes d'un arbre binaire. */
public class EnumerationInfixe extends EnumerationArbre {
    /** Crée une énumération infixes de l'arbre binaire spécifié. */
    public EnumerationInfixe(ArbreBinaire a) {...}
    /** Renvoie l'élément suivant de cette énumération, c'est-à-dire
     * - l'élément suivant du sous-arbre gauche si celui-ci n'est pas vide
     *   et a encore des éléments à énumérer,
     * - sinon la racine si celle-ci n'a pas encore été énumérée,
     * - sinon l'élément suivant du sous-arbre droit si celui-ci n'est pas vide
     *   et a encore des éléments à énumérer,
     * - sinon null.
     */
    public Object nextElement() {...}
}

```

```

package ab;
/** Classe décrivant une énumération infixes parenthésée d'un arbre binaire. */
public class EnumerationInfixePar extends EnumerationArbre {
    /** Indique si on a ouvert une parenthèse. */
    private boolean par = false;
    /** Crée une énumération infixes parenthésée de l'arbre binaire spécifié. */
    public EnumerationInfixePar(ArbreBinaire a) {...}
    /** Teste s'il reste des éléments à énumérer ou si une parenthèse a été
     * ouverte et non fermée. */
    public boolean hasMoreElements() { return super.hasMoreElements() || par; }
    /** Renvoie l'élément suivant de cette énumération, c'est-à-dire
     * - Si pas de parenthèse ouverte et s'il y a au moins un sous-arbre non vide,
     *   ouvre et renvoie une parenthèse (ouvrante),
     * - sinon renvoie l'élément suivant du sous-arbre gauche si celui-ci n'est pas
     *   vide et a encore des éléments à énumérer,
     * - sinon renvoie la racine si celle-ci n'a pas été énumérée,
     * - sinon renvoie l'élément suivant du sous-arbre droit si celui-ci n'est pas
     *   vide et a encore des éléments à énumérer,
     * - sinon, si une parenthèse a été ouverte, la ferme et la renvoie (fermante),
     * - sinon renvoie null.
     */
    public Object nextElement() {...}
}

```