Interface des Programmes d'Application

Master Informatique
Université Bordeaux 1
décembre 2005
Aucun document autorisé, durée 3 heures

Justifier clairement et succinctement vos choix, et décrivez brièvement vos algorithmes lorsque vous le jugez nécessaire.

Toute réponse difficile à comprendre sera considérée comme fausse.

Vous pouvez rajouter du code (méthode, donnée ou classe) non demandé à chaque fois que vous le jugerez nécessaire. Ecrivez entièrement ce code, en particulier dans le cas d'exceptions, de classes abstraites ou d'interfaces.

Vous pouvez toujours considérer une classe ou une méthode demandée dans une question précédente comme disponible, même si vous n'avez pas traité cette question. Il vous est également conseillé de lire le sujet entièrement avant de commencer votre travail.

Si vous souhaitez utiliser une classe (ou une méthode particulière) de l'API Java dont vous ne vous souvenez pas du nom, donnez-lui un nom explicite et précisez-le clairement sur votre copie.

Vous pouvez à chaque fois que vous le jugez compréhensible utiliser des abréviations à la place des noms complets de classe et méthode ou encore '...' pour remplacer du code non modifié.

Les instructions `import` et `package` ne seront pas précisées.

Partie 1 - Labyrinthe

On considère les interfaces `Labyrinthe` et `Salle` et les classes `Grille` et `SalleGrille`, qui implémentent ces interfaces.

```
public interface Labyrinthe {
        // itere les salles composant le labyrinthe
        public Iterator<Salle> salles();
        // renvoie la salle d'entree
        public Salle entree();
        // renvoie vrai ssi il existe un passage de s1 a s2
        public boolean passage(Salle s1, Salle s2);
}

public interface Salle {
        // indique si cette salle est une sortie
        public boolean sortie();
        // labyrinthe auquel appartient la salle
```

```
        public Labyrinthe labyrinthe();
        // salles voisines. Modifier le tableau resultat n'a aucun effet sur le
        // labyrinthe
        public Salle[] voisines();
}

public class Grille implements Labyrinthe {
        protected int hauteur;
        protected int largeur;
        protected SalleGrille[][] salles;

        public Grille(int hauteur, int largeur) {
                this.hauteur = hauteur;
                this.largeur = largeur;
                creerSalles();
        }

        protected void creerSalles() {
                salles = new SalleGrille[hauteur][largeur];
                for (int i = 0; i < hauteur; i++) {
                        for (int j = 0; j < largeur; j++) {
                                salles[i][j] = new SalleGrille(this, i, j);
                        }
                }
        }

        public Iterator<Salle> salles() {
                return new Iterator<Salle>() {
                        int i = 0;
                        int j = 0;

                        public boolean hasNext() {
                                return i < hauteur;
                        }

                        public Salle next() throws NoSuchElementException {
                                Salle s = salles[i][j++];
                                if (! hasNext()) {
                                        throw new NoSuchElementException();
                                }
                                if (j == largeur) {
                                        i++;
                                        j = 0;
                                }
                                return s;
```

```
                        }

                        public void remove() throws UnsupportedOperationException {
                                throw new UnsupportedOperationException();
                        }
                };
        }

        public int hauteur() {...}

        public int largeur() {...}

        public Salle entree() {...}

        public boolean passage(Salle s1, Salle s2) {...}
}

class SalleGrille implements Salle {
        int i;
        int j;
        Grille grille;
        int hauteur;
        int largeur;

        SalleGrille(Grille g, int i, int j) {
                this.i = i;
                this.j = j;
                this.grille = g;
                largeur = g.largeur();
                hauteur = g.hauteur();
        }

        public boolean sortie() {
                return i == hauteur - 1 && j == largeur - 1;
        }

        private boolean v(int i1, int i2) {
                return i1 == i2 - 1 || i1 == i2 + 1;
        }

        boolean voisine(SalleGrille sg) {
            return sg.grille == grille && ((i == sg.i && v(j, sg.j)) || (j == sg.j && v(i, sg
        }

        public Labyrinthe labyrinthe() {
```

```
                return grille;
        }

        public Salle[] voisines() {
                int nv = (i > 0 && i < hauteur - 1 ? 2 : 1);
                nv += (j > 0 && j < largeur - 1 ? 2 : 1);
                Salle[] v = new SalleGrille[nv];
                int iv = 0;
                if (i > 0) {
                        v[iv++] = grille.salles[i - 1][j];
                }
                if (j < largeur - 1) {
                        v[iv++] = grille.salles[i][j + 1];
                }
                if (i < hauteur - 1) {
                        v[iv++] = grille.salles[i + 1][j];
                }
                if (j > 0) {
                        v[iv++] = grille.salles[i][j - 1];
                }
                return v;
        }
}
```

On appellera par la suite *coordonnées* les valeurs des variables `i` et `j` d'une instance de `SalleGrille`.

Question 1 Compléter les méthodes `hauteur`, `largeur`, `entree` et `passage` de la classe `Grille`. La salle d'entrée sera la salle de coordonnées $i = 0$ et $j = 0$.

Question 2 Modifier la classe `SalleGrille` pour que l'instruction `System.out.print(s);` où `s` désigne une instance de `SalleGrille` de coordonnées i_0 et j_0 produise l'affichage sur la sortie standard de "(i_0, j_0)".

Quel sera alors le résultat de l'exécution des instructions suivantes :

```
Labyrinthe l = new Grille(2,3);
for (Iterator<Salle> it = l.salles(); it.hasNext();) {
        System.out.print(it.next() + " ");
}
System.out.println();
```

Question 3 Ecrire une classe `GrilleAvecMur` qui hérite de la classe `Grille` et dont le but est de permettre l'ajout de murs entre les salles d'une grille. L'ajout d'un mur horizontal (respectivement vertical) de coordonnées (i, j) empêchera le passage de la salle de coordonnées (i, j) à la salle de coordonnées $(i + 1, j)$ (respectivement $(i, j + 1)$).

La classe `GrilleAvecMur` possède deux nouvelles méthodes :
`public void ajouterMurHorizontal(int i, int j)` et
`public void ajouterMurVertical(int i, int j)`
et redéfinit la méthode
`public boolean passage(Salle s1, Salle s2)` en renvoyant vrai si et seulement si les salles `s1` et `s2` sont voisines et il n'existe pas de mur entre ces deux salles. Les murs seront mémorisés à l'aide de tableaux de type `boolean [][]`.

Partie 2 - Personnage

On considère l'interface `Personnage` qui représente un personnage dans un labyrinthe.

```
public interface Personnage {
        // positionne un personnage dans la salle d'entree d'un labyrinthe.
        public void entrer(Labyrinthe l);
        // labyrinthe ou se trouve le personnage.
        public Labyrinthe labyrinthe();
        // salle ou se trouve le personnage.
        public Salle position();
        // deplace le personnage si il existe un passage entre sa position
        // et la nouvelle salle.
        public void aller(Salle s) throws DeplacementInterditException;
        // vrai si et seulement si la position du personnage est une sortie.
        public boolean estSorti();
}
```

Question 1 Ecrire une classe `PersonnageDefaut` qui donne une implémentation de `Personnage`.

Question 2 Ecrire la classe `DeplacementInterditException`.

On considère l'interface `Deplacement` qui calcule un déplacement à chaque appel de la méthode `suivant`.

```
public interface Deplacement {
        // propose un deplacement en fonction de la salle passee en parametre.
        public Salle suivant(Salle s);
}
```

Question 3 Ecrire une classe `PersonnageAvecDeplacement` qui décore un personnage en lui ajoutant une méthode `public void deplacer()`. La méthode `deplacer()`

déplace le personnage à l'aide de la méthode `suivant` d'une instance de `Deplacement`, passée en paramètre à la construction.

Question 4 Ecrire une classe `PersonnageAvecDessin` qui décore un personnage en lui ajoutant une méthode `public void dessine(Graphics g, int x, int y, int l, int h)` dessinant un "bonhomme" de hauteur h, largeur l aux coordonnées (x,y) d'un graphics g. Le code de la méthode `dessine` n'est pas demandé (on mettra ...).

Partie 3 - Plus Court Chemin

On dispose d'une classe `AlgorithmesGraphes` implémentant une méthode `public static Iterator<Sommet> plusCourtChemin(Sommet depart, Sommet destination)` qui renvoie un itérateur dont les éléments successifs constituent un plus court chemin entre les sommets depart et destination (depart et destination compris).

`Sommet` est une interface décrite ci-dessous.

```
public interface Sommet {
        // itere les voisins du sommet.
        public Iterator<Sommet> voisins();
        // vrai si v est un voisin du sommet.
        public boolean voisin(Sommet v);
}
```

Question 1 Ecrire une classe `SalleGrilleAdaptee` qui étend la classe `SalleGrille` et qui implémente `Sommet`.

Question 2 Ecrire une classe `GrilleAvecMurAdaptee` qui étend la classe `GrilleAvecMurs` et dont les salles sont des instances de `SalleGrilleAdaptee`.

Question 3 Ajouter à la classe `GrilleAvecMurAdaptee` une méthode `public Deplacement creerDeplacementPCC()` qui crée un déplacement dont chaque appel à la méthode `public Salle suivant(Salle s)` renvoie le sommet suivant d'un plus court chemin entre l'entrée de la grille et la sortie, ou `s` si aucun chemin n'existe.

Partie 4 - Application

On dispose d'une classe `DessinGrilleAvecMursEtPersonnage` qui étend la classe `java.swing.JComponent` et qui dessine une grille avec murs et un personnage si la position de celui-ci est non nulle.

Question 1 Expliquer en quelques phrases (10 maximum) le comportement de l'application `LabyrintheApplicationPCC` décrite ci-dessous.

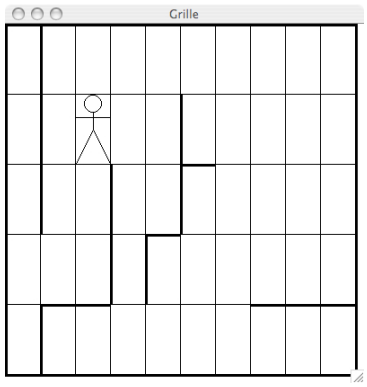

FIG. 1 – Exemple de dessin de grille avec murs et personnage

```
public class LabyrintheApplicationPCC {

    private static Grille creerFrame(int hauteur, int largeur) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("Grille");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final GrilleAvecMurAdaptee grille =
            new GrilleAvecMurAdaptee(hauteur, largeur);
        final PersonnageAvecDeplacement p =
            new PersonnageAvecDeplacement(new PersonnageDefaut(),
                                          grille.creerDeplacementPCC());
        final PersonnageAvecDessin pd = new PersonnageAvecDessin(p);
        final DessinGrilleAvecMurEtPersonnage dessin =
            new DessinGrilleAvecMurEtPersonnage(grille, pd);

        final MouseListener ml = new MouseAdapter() {
                public void mouseClicked(MouseEvent arg0) {
                    int h = dessin.hauteurSalle();
                    int l = dessin.largeurSalle();
                    int x = arg0.getX();
                    int y = arg0.getY();
                    if (Math.abs(y - y / h * h) < 3) {
                        grille.ajouterMurHorizontal(y / h - 1, x / l);
                    } else if (Math.abs(x - x / l * l) < 3) {
                        grille.ajouterMurVertical(y / h, x / l - 1);
                    }
                    dessin.repaint();
                }
```

```
            };
        dessin.addMouseListener(ml);

        dessin.addKeyListener(new KeyAdapter() {
                public void keyPressed(KeyEvent evt) {
                    if (evt.getKeyChar() == 'd') {
                        if (p.position() == null) {
                            dessin.removeMouseListener(ml);
                            p.entrer(grille);
                        } else {
                            try {
                                p.deplacer();
                            } catch (DeplacementInterditException e) {
                            }
                        }
                        dessin.repaint();
                    }
                }
            });
        dessin.setFocusable(true);
        dessin.requestFocus();
        frame.setContentPane(dessin);
        frame.pack();
        frame.setVisible(true);
        return grille;
    }

    public static void main(final String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    creerFrame(Integer.parseInt(args[0]),
                               Integer.parseInt(args[1]));
                }
            });
    }
}
```