Interface des Programmes d'Application - Corrigé

Master Informatique
Université Bordeaux 1
décembre 2005

Partie 1 - Labyrinthe

Question 1 - 2 points Compléter les méthodes `hauteur`, `largeur`, `entree` et `passage` de la classe `Grille`.

```
[4]
public int hauteur() {
    return hauteur;
}

public int largeur() {
    return largeur;
}

public Salle entree() {
    return salles[0][0];
}

public boolean passage(Salle s1, Salle s2) {
    return ((SalleGrille) s1).voisine((SalleGrille) s2);
}
```

0,5 point enlevé si le cast en SalleGrille n'est pas présent ou si utilisation d'un for

Question 2 - 2 points 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();
```

Il faut redéfinir dans la classe SalleGrille la méthode toString().

```
public String toString() {
    return "(" + i + "," + j + ")";
}
```

et le résultat affiché sera le suivant :

```
(0,0) (0,1) (0,2) (1,0) (1,1) (1,2)
```

Question 3 - 3 points 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.

```
public class GrilleAvecMur extends Grille {

    boolean[][] mursHorizontaux;

    boolean[][] mursVerticaux;

    public GrilleAvecMur(int hauteur, int largeur) {
        super(hauteur, largeur);
        mursHorizontaux = new boolean[hauteur - 1][largeur];
        mursVerticaux = new boolean[hauteur][largeur - 1];
        for (int i = 0; i < hauteur - 1; i++) {
            for (int j = 0; j < largeur; j++) {
                mursHorizontaux[i][j] = false;
            }
        }
        for (int i = 0; i < hauteur; i++) {
            for (int j = 0; j < largeur - 1; j++) {
                mursVerticaux[i][j] = false;
            }
        }
    }

    /** ajoute un mur entre la case (i,j) et la case (i+1,j) */
    public void ajouterMurHorizontal(int i, int j) {
        mursHorizontaux[i][j] = true;
    }

    /** ajoute un mur entre la case (i,j) et la case (i,j+1) */
    public void ajouterMurVertical(int i, int j) {
        mursVerticaux[i][j] = true;
    }

    public boolean passage(Salle s1, Salle s2) {
        SalleGrille sg1 = (SalleGrille) s1;
```

```
        SalleGrille sg2 = (SalleGrille) s2;

        if (sg1.i == sg2.i) {
            return (sg2.j == sg1.j + 1 && !mursVerticaux[sg1.i][sg1.j])
                || (sg1.j == sg2.j + 1 && !mursVerticaux[sg2.i][sg2.j]);
        } else if (sg1.j == sg2.j) {
            return (sg2.i == sg1.i + 1 && !mursHorizontaux[sg1.i][sg1.j])
                || (sg1.i == sg2.i + 1 && !mursHorizontaux[sg2.i][sg2.j]);
        } else {
            return false;
        }
    }
}
```

1,5 points pour la construction et les méthodes d'ajouts de murs, (en particulier l'appel à super() avec les paramètres nécessaires et l'allocation des tableaux) et 1,5 points pour la méthode passage.

Partie 2 - Personnage

Question 1 - 2 points Ecrire une classe `PersonnageDefaut` qui donne une implémentation de `Personnage`.

```
public class PersonnageDefaut implements Personnage {

    private Salle position = null;

    public void entrer(Labyrinthe l) {
        position = l.entree();
    }

    public void aller(Salle s) throws DeplacementInterditException {
        if (!labyrinthe().passage(position, s)) {
            throw new DeplacementInterditException(position, s);
        }
        position = s;
    }

    public Salle position() {
        return position;
    }

    public Labyrinthe labyrinthe() {
```

```
        return position.labyrinthe();
    }

    public boolean estSorti() {
        return position.sortie();
    }

}
```

-0,5 pt si la levée de l'exception DeplacementInterditException est absente ou mal écrite, -1 pt si vous conservez les coordonnées de la salle position au lieu de la salle elle-meme (vous créez une dépendance avec les classes Grille et SalleGrille au lieu d'utiliser uniquement les interfaces).

Question 2 - 1,5 points Ecrire la classe `DeplacementInterditException`.

```
public class DeplacementInterditException extends Exception {
    Salle origine;

    Salle destination;

    public DeplacementInterditException(Salle origine, Salle destination) {
        this.origine = origine;
        this.destination = destination;
    }

    public Salle origine() {
        return origine;
    }

    public Salle destination() {
        return destination;
    }
}
```

-1 pt si les salles ne sont pas stockées dans l'exception, -0,5 si elles sont stockées mais sans accesseurs.

Question 3 - 2 points 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 - 1point 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 ...).

Les questions 3 et 4 étaient liées, la deuxième devant vous inciter à factoriser le code de la délégation dans une classe abstraite. -1 pt si le code est purement et simplement dupliqué. -0,5 pt si le personnage décoré est créé dans la classe décoratrice car celle-ci ne peut plus décorer un personnage existant.

```
public abstract class PersonnageDecore implements Personnage {
    protected Personnage p;

    PersonnageDecore(Personnage p) {
        this.p = p;
    }

    public void entrer(Labyrinthe l) {
        p.entrer(l);
    }

    public void aller(Salle s) throws DeplacementInterditException {
        p.aller(s);
    }

    public Salle position() {
        return p.position();
    }

    public Labyrinthe labyrinthe() {
        return p.labyrinthe();
    }

    public boolean estSorti() {
        return p.estSorti();
    }
}

public class PersonnageAvecDeplacement extends PersonnageDecore {

    private Deplacement deplacement;

    public PersonnageAvecDeplacement(Personnage p, Deplacement d) {
        super(p);
        this.deplacement = d;
    }
```

```
    public void deplacer() throws DeplacementInterditException {
        aller(deplacement.suivant(position()));
    }
}

public class PersonnageAvecDessin extends PersonnageDecore {

    public PersonnageAvecDessin(Personnage p) {
        super(p);
    }

    public void dessine(Graphics g, int x, int y, int l, int h) {
        ...
    }
}
```

Partie 3 - Plus Court Chemin

Question 1 - 2 points Ecrire une classe `SalleGrilleAdaptee` qui étend la classe `SalleGrille` et qui implémente `Sommet`.

```
class SalleGrilleAdaptee extends SalleGrille implements Sommet {

    SalleGrilleAdaptee(GrilleAvecMurAdaptee g, int i, int j) {
        super(g, i, j);
    }

    public Iterator<Sommet> voisins() {

        return new Iterator<Sommet>() {
            int i = 0;
            Salle[] v = voisines();
            Sommet suivant = null;

            public boolean hasNext() {
                while (i < v.length && suivant == null) {
                    if (grille.passage(SalleGrilleAdaptee.this, v[i])) {
                        suivant = (SalleGrilleAdaptee) v[i];
                    }
                    i++;
                }
                return suivant != null;
            }
```

```
            public Sommet next() {
                if (!hasNext()) {
                    throw new NoSuchElementException();
                }
                Sommet tmp = suivant;
                suivant = null;
                return tmp;
            }

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

    }

    public boolean voisin(Sommet v) {
        return grille.passage(this, (Salle) v);
    }
}
```

Question 2 - 1 point Ecrire une classe `GrilleAvecMurAdaptee` qui étend la classe `GrilleAvecMurs` et dont les salles sont des instances de `SalleGrilleAdaptee`.

Question 3 - 1,5 point 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.

```
public class GrilleAvecMurAdaptee extends GrilleAvecMur {

    public GrilleAvecMurAdaptee(int hauteur, int largeur) {
        super(hauteur, largeur);
    }

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

```
    }

    public Deplacement creerDeplacementPCC() {
        return new Deplacement() {
                Iterator<Sommet> chemin = null;
                public Salle suivant(Salle s) {
                    if (chemin == null) {
                        chemin = AlgorithmesGraphes.plusCourtChemin(
                                    (Sommet)entree(),
                                    (Sommet)salles[hauteur - 1][largeur - 1]);
                        if (chemin == null) {
                            return s; //pas de PCC
                        }
                        chemin.next(); // pour eliminer l'entree
                    }
                    if (chemin.hasNext()) {
                        return (Salle)chemin.next();
                    }
                    return s;
                }
            };
    }
}
```

Partie 4 - Application

Question 1 - 2 points Expliquer en quelques phrases (10 maximum) le comportement de l'application `LabyrintheApplicationPCC`.

L'application crée une fenetre, de titre "Grille", contenant une grille dont les dimensions sont passées en paramètres de la ligne de commande.

Dans un premier temps, l'utilisateur peut créer des murs en cliquant à proximité (3 pixels) d'une barre de la grille.

L'utilisation du caractère 'd' provoque la première fois l'arret de l'édition des murs et l'apparition d'un personnage dans la salle d'entrée.

Les utilisations suivantes de 'd' font avancer le personnage le long d'un plus court chemin jusqu'à la sortie. Si un tel chemin n'existe pas, le personnage reste sur la case de départ.

On quitte l'application en fermant la fenetre.