

## Programmation Orientée Objet Examen du 6 juin 2003

Durée 2h

Seuls documents autorisés : documents distribués et notes personnelles de cours

### I

1. La classe `Cla` a été définie dans un paquetage `paquet`. Pour la tester une classe principale `Test` a été définie dans le même paquetage.

Les fichiers `Test.class` et `Cla.class` des pseudo-codes se trouvent dans le répertoire `/users/groupe1/durand/projetsJava/classes/paquet`.

Donner une commande `java` exécutant la classe principale de ce paquetage. On précisera dans quel répertoire on doit se trouver pour exécuter cette commande.

2. Même question en supposant maintenant que la classe `Cla` nécessite l'utilisation d'une autre classe `ClaAux` d'un paquetage `paquetAux` dont le pseudo-code se trouve dans le répertoire `/users/groupe1/durand/utilitaires/classes/paquetAux`.

### II

**Le but de cet exercice est de définir une classe abstraite `Nombre`, deux sous-classes `Reel` et `Complexe` de `Nombre` et une classe `Equation`.**

Ces classes permettent de résoudre des équations du second degré dont les coefficients sont soit des réels, soit des complexes. Pour simplifier, on ne traitera que des équations de la forme  $x^2 + bx + c$  où le coefficient principal est égal à 1 et où  $b$  et  $c$  sont des réels ou des complexes, instances de la classe `Reel` ou de la classe `Complexe`.

**Définir ces classes en respectant les indications ci-dessous.**

La classe `Reel` comprendra un attribut `val` de type `double` dont la valeur sera celle du réel représenté.

Le constructeur aura comme paramètre un `double` correspondant à la valeur du réel.

La classe `Complexe` comprendra deux attributs `reel` et `imag` de type `double` dont les valeurs seront les parties réelle et imaginaire du complexe représenté.

Le constructeur aura deux paramètres de type `double` correspondant aux parties réelle et imaginaire du complexe.

Les méthodes `somme`, `produit` et `racineCarree` seront des méthodes d'instance abstraites de la classe `Nombre` et seront définies dans ses deux sous-classes.

La méthode `carre` sera une méthode définie dans la classe `Nombre`.

Si la définition de la méthode `racineCarree` de la classe `Reel` est simple, la méthode `racineCarree` de la classe `Complexe` nécessite la résolution d'une équation du second degré à coefficients réels qui sera résolue en faisant appel à la classe `Equation`.<sup>1</sup>

La classe `Equation` comprendra deux attributs `b` et `c` instances de la classe `Nombre`. On appellera `racine1` la méthode renvoyant la plus grande racine et `racine2` l'autre racine. On tiendra compte, bien sûr, de la simplification apportée en se restreignant aux équations dont le coefficient principal est égal à 1.

---

<sup>1</sup> En effet, si  $a$  et  $b$  sont réels et si on cherche  $x$  et  $y$  tels que  $\sqrt{a+ib} = x+iy$  on a  $a+ib = (x+iy)^2 = x^2 - y^2 + 2ixy$ , d'où

$$x^2 - y^2 = a \text{ et } 2xy = b, y = b/2x, x^2 - \left(\frac{b}{2x}\right)^2 = a \text{ et enfin } x^4 - ax^2 - \frac{b^2}{4} = 0$$

L'équation  $X^2 - aX - \frac{b^2}{4} = 0$  ayant, si  $b \neq 0$ , deux racines non nulles de signes différents,  $x^2$  est égal à la racine positive de cette équation (qui est aussi la racine la plus grande) et  $x$  à la racine carrée de  $x^2$ .

(Si  $b=0$ , le calcul de  $\sqrt{a}$  est immédiat, réel ou imaginaire pur selon le signe de  $a$ ).

### III

#### Le but de cet exercice est de programmer une interface graphique pour le jeu des dominos.

On rappelle que ce jeu consiste à essayer de remplir une *grille* de  $n \times m$  cases ( $n$  lignes et  $m$  colonnes) par des *dominos*, c'est-à-dire des pièces constituées de deux cases adjacentes.

L'utilisation d'une interface graphique va non seulement être agréable au joueur qui n'aura qu'à cliquer sur des *cases-boutons* pour *poser* ou *retirer* un domino, mais va aussi simplifier la programmation, car les informations nécessaires seront attachées aux composants graphiques.

On construira une *fenêtre graphique* comportant deux cases à cocher (Checkbox et CheckboxGroup) indiquant si l'on veut poser ou retirer un domino, une grille de  $n \times m$  cases-boutons et un *champ* pour écrire éventuellement des messages. Un domino sera *construit* puis *posé* après deux clics successifs sur deux boutons adjacents. Pour qu'il soit posable, il n'est pas nécessaire de tester si les cases sont dans la grille puisque les boutons sont associés à des cases de la grille. Il suffit de tester que les cases sont *vides*.

1. Les informations attachées aux boutons seront d'une part les coordonnées de la case correspondante (non visualisées pour le joueur) et d'autre part le *numéro* d'un domino éventuellement posé sur cette case (visualisé pour le joueur).

Le *numéro* du domino sera affiché sur le *bouton* et il suffira de cliquer sur une des deux cases du domino pour le *retirer*. On n'aura pas besoin de mémoriser autrement la liste des dominos, on aura seulement besoin de connaître le *nombre de dominos déjà posés*. Après un premier clic, la case correspondante sera marquée d'une croix (un X) en attendant une deuxième case convenable (*vide et adjacente*). On utilisera judicieusement les méthodes `getLabel`, `setLabel`, `getActionCommand` et `setActionCommand` pour cela.

Les images en annexe précisent l'interface graphique demandée.

#### Compléter le listing suivant, partiel, d'un tel programme, aux endroits encadrés.

```
package dominos;
import es.Keyboard;
public class Test {
    public static void main(String[] args) {
        int n = Keyboard.readInt("dimensions de la grille ? "); int m = Keyboard.readInt();
        Grille g = new Grille(n,m);
        Jeu j = new Jeu(g); j.pack(); j.show();
    }
}
```

```
-----
package dominos;
```

```
    les import
```

```
public class Jeu extends Frame implements ActionListener {
    private Grille g;
    private int numero; // le numéro du dernier domino posé (0 si pas de domino)
    private Case premiereCase = null; // première case cochée, null sinon
```

```
    déclaration des composants graphiques
```

```
    /** crée une partie pour ce jeu sur la grille 'g' */
    public Jeu(Grille g) {
```

```
        assemblage des composants
        mise en place des écouteurs
    }
```

```
    /** Pose un domino sur la grille de ce jeu si c'est possible, avec un nouveau
     *  numéro. Sinon affiche un message.*/
    public void poser(Domino d){
```

```
        remplit les deux cases de ce domino par un nouveau numéro
    }
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        extrait les coordonnées de la case dont le bouton a été cliqué, soit cClic cette case
        selon que :
        - la case 'poser' ou 'retire' a été cochée
        - cClic est vide ou non
        - il y a déjà eu, ou non, une première case choisie (dans le cas 'poser')
        on exécute l'un des traitements adéquats suivants :
        - marquer une première case
```

- créer et poser un domino (avec message)
- retirer un domino (avec message)
- vider une première case déjà choisie et marquée

En cas d'impossibilité (choix d'une case déjà occupée, cases non adjacentes, retirer une case vide, ... )  
afficher un message d'erreur

```

}

/** retire le domino qui se trouve sur la case c */
public void retirerDomino(Case c) {

```

cherche parmi les cases voisines de  $c$  celle qui a le même numéro que  $c$  et vide les deux cases

```

}

-----
package dominos;

```

les import

```

public class Grille {
    protected int n,m; // les dimensions de cette grille
    protected Case[][] lesCases ; // les cases de cette grille
    protected Panel p ; // le panel qui contiendra les boutons de ces cases

```

```

    /** crée une grille de nxm cases (n lignes et m colonnes) */
    public Grille(int n, int m) {

```

création du tableau *lesCases* de cases vides  
assemblage des boutons de ces cases dans le panel *p*

```

    /** Teste si toutes les cases de cette grille sont remplies */
    public boolean pleine() {

```

```

    /** renvoie le vecteur des cases adjacentes à la case c */
    protected Vector voisines(Case c) {

```

```

}

-----
package dominos;

```

```

public class Domino {
    private Case[] lesDeuxCases; // les deux cases de ce domino

```

```

    /** crée un domino à poser lsur les cases c1 et c2 */
    public Domino(Case c1, Case c2) {

```

```

    /** renvoie les deux cases de ce domino sous forme d'un tableau de deux cases */
    public Case[] getCases() {

```

```

}

-----
package dominos;

```

les import

```

public class Case {
    private int x,y; // les coordonnées de cette case
    private Button bouton ; // le bouton de cette case

```

```

    /** crée une case vide de coordonnées x et y */
    public Case(int x, int y) {

```

remplir les champs  
mémoriser dans les boutons de cette case les valeurs de  $x$  et  $y$ , par exemple sous la forme de la chaîne  $x + " " + y$

```

    /** Remplit cette case par la chaîne 's' */
    public void remplir(String s) {

```

affiche  $s$  sur le bouton de cette case

```

/** Vide cette case */
public void vider() {
    supprimer l'affichage
}

/** Teste si cette case est vide */
public boolean vide() {
    affichage vide
}

/** Marque cette case */
public void marquer() {
    afficher une croix (caractère X)
}

/** Teste si cette case est marquée */
public boolean marquee() {
    teste l'affichage d'une croix
}

/** renvoie la première coordonnée de cette case */
public int getX() {
}

/** renvoie la deuxième coordonnée de cette case */
public int getY() {
}

/** renvoie le bouton de cette case */
public Button getBouton() {
}

/** renvoie l'affichage de cette case */
public String getVal() {
}

/** teste si cette case est adjacente à la case c */
public boolean adjacente(Case c) {
}
}

```

2. Pour rendre la visualisation plus agréable, on colorie les deux cases d'un même domino d'une même couleur. Définir un tableau de couleurs de taille  $N$  et colorier le domino de numéro  $i$  de la couleur  $i \bmod N$ .

**Programmer les modifications à apporter pour cela.**

3. Dans la question 2, la numérotation des dominos est encore indispensable car deux dominos adjacents dont les numéros diffèrent de  $N$  auront la même couleur. On peut remédier à cet inconvénient en choisissant pour chaque domino une couleur non utilisée par aucune des cases voisines.

**Combien de couleurs différentes sont alors nécessaires** (en plus du gris clair - par défaut - pour les cases vides, et du blanc - par exemple - pour les cases *marquées*) ?

Dans ce cas, on n'a plus besoin des numéros. L'attribut `numéro` sera alors supprimé.

**Indiquer les quelques modifications à faire pour choisir les couleurs et récupérer les informations nécessaires à partir de ces couleurs.**

## Annexes

A1. Quelques attributs et méthodes non vus en cours qui pourront être utiles :

Classe String

```
public String substring(int beginIndex)
```

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Exemples:

"unhappy".substring(2) returns "happy"

"Harbison".substring(3) returns "bison"

"emptiness".substring(9) returns "" (an empty string)

public String substring(int beginIndex,int endIndex)

Returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex.

Exemples:

"hamburger".substring(4, 8) returns "urge"

"smiles".substring(1, 5) returns "mile"

mais l'utilisation de la classe StringTokenizer sera plus élégante !

### Classe Color

```
static Color black
static Color blue
static Color cyan
static Color darkGray
static Color gray
```

```
static Color green
static Color lightGray
static Color magenta
static Color orange
static Color pink
```

```
static Color red
static Color white
static Color yellow
```

### Classe Component

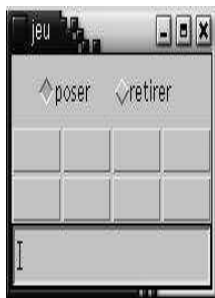
Color getBackground()

Gets the background color of this component.

void setBackground(Color c)

Sets the background color of this component.

## A2. Visualisation du jeu



fenêtre initiale



après choix d'une première case,



et d'une deuxième case,



et de deux autres dominos



on essaie un quatrième,



sans succès



alors on retire un domino



et on continue



et on cherche une autre solution

