**ELEMENTS DE CORRECTION**

DS S2 Programmation Orientée Objet

Jean-Claude MARTIN 17 Juin 2016 Durée : 2h. Documents autorisés : tout sauf livres.

Note sur toute la copie :

instructions package (0.5), indentation et qq commentaires (0.5)

1. Modèle de Pion (2 pts)

classe abstraite (1), attributs protected (0.5), type des 3 attributs (0.5)

|  |
| --- |
| **//Pion.java**  **//--------------------------------------------**  **package morpion.modeles ;**  **//--------------------------------------------**  **public abstract class Pion {**  **protected int type ; // A supprimer pour la question 3 puisque l'on crée des sous-classes PionUtilisateur, PionSysteme, ...**  **protected int ligne ;**  **protected int colonne ;**  **}** |

1. Modèle de Pion de Morpion (1.5 pts)

classe concrète (0.25), extends Pion (0.25), constantes (0.5), instructions méthodes static (0.5 (ne pas pénaliser s'ils appellent un constructeur sans paramètre)

|  |
| --- |
| **//PionMorpion.java**  **//--------------------------------------------**  **package morpion.modeles;**  **//--------------------------------------------**  **public class PionMorpion extends Pion {**  **public final static int PION\_UTILISATEUR = 1 ;**  **public final static int PION\_SYSTEME = 2;**    **//--------------------------------------------**  **public static PionMorpion creerPionUtilisateur (int ligne, int colonne) {**  **return new PionMorpion (PION\_UTILISATEUR, ligne, colonne);**  **}**    **public static PionMorpion creerPionSysteme (int ligne, int colonne) {**  **return new PionMorpion (PION\_SYSTEME, ligne, colonne);**  **}**  **}** |

3) Point de vue critique (1 pt)

types au lieu de classes (0.5 (ne pas pénaliser s'ils n'ont pas mis static)), méthodes static (0.5)

|  |
| --- |
| **Il a été demandé de créer une seule classe avec un attribut type. Il faudra donc tester la valeur de cet attribut et en fonction de sa valeur, exécutez des instructions différentes. Ce n’est pas du tout modulaire de faire ainsi des cascades de if testant le type d’un objet et c’est contraire aux principes de la Programmation Orientée Objet.** |

4) Amélioration (1.5 pts)

PionUtilisateur et PionSysteme : classe concrete (0.25), extends Pion (0.25), constructeur (0.5), supprimer l'attribut type dans la classe Pion (0.5)

|  |
| --- |
| **// PionUtilisateur**  **//----------------------------------------**  **package morpion.modeles;**  **public class PionUtilisateur extends Pion {**  **public PionUtilisateur (int ligne, int colonne) {**  **super (ligne, colonne);**  **}**  **}**  **// PionSysteme**  **//----------------------------------------**  **package morpion.modeles;**  **public class PionSysteme extends Pion {**  **PionSysteme (int ligne, int colonne) {**  **super (ligne, colonne);**  **}**  **}**  Dans la classe Pion, on supprime la déclaration de l’attribut type. |

5) Modele de Jeu de plateau (3.5 pts)

1. Est-ce que Plateau est une classe générique ?   
   oui (0.25), car il y a un type générique après le nom de la classe (0.25)

|  |
| --- |
| **Oui : il y a un type générique <T> après le nom de la classe.** |

1. Pour quelle raison le programmeur a-t-il indiqué (T[][]) dans l’instruction suivante :  
   plateau = (T[][]) new Object [n][n];

|  |
| --- |
| **Cela aurait été mieux de pouvoir utiliser le type générique sans caster mais l’instruction**  **plateau = new T [n][n];**  **génère une ERREUR DE COMPILATION : “Cannot create a generic array of T”**  **On est donc obligé de faire un cast :-(** |

1. On vous donne le code ci-dessous :

package morpion.modeles ;

import java.util.ArrayList;

public interface Jouable<T> {

Plateau<T> getPlateau();

ArrayList<T> listeDesCoups();

void ajouter (T p);

void faireJouerSysteme();

boolean userWins();

boolean systemWins();

}

Pour quelle raisons le programmeur de cette interface a-t-il indiqué “public” devant l’interface ?

seulement 0.25 s'ils disent "puissent être implémentée par n'importe quelle classe" (il faut en fait dire plus généralement \*accédée ou utilisée\*, voire par exemple l'attribut modele de la classe Controleur qui est de type Jouable : la déclaration de cet attribut utilise l'interface Jouable sans l'implémenter)

|  |
| --- |
| **Pour pouvoir accéder à cette interface depuis les packages vues et contrôleurs.** |

d) Est-ce que l’on peut écrire “new Jouable()” ?

|  |
| --- |
| **Non. On ne peut pas créer d’instance d’une interface !** |

e) On considère une classe ModeleJeuMorpion qui implémente l’interface Jouable.   
Donnez juste :

* l’entête de la classe de la classe ModeleJeuMorpion
* la déclaration des attributs plateau et listeCoups de la classe ModeleJeuMorpion.

Vous n’avez pas à donner le reste du code de cette classe.

<Pion> : 0.5, implements Jouable 0.5, attributs 0.5

|  |
| --- |
| **public class ModeleJeuMorpion implements Jouable<Pion> {**  **private Plateau<Pion> plateau ;**  **private ArrayList <Pion> listeDesCoups;** |

6) VuePlateauMorpion (5.5 pts)

1. les déclarations des attributs de la classe VuePlateauMorpion (2 pts)

constantes (0.25), grille de boutons (1) (0.5 si liste de 9 boutons), modele (0.5), taille (0.25)

|  |
| --- |
| **// ATTRIBUTS**  **private static final String SYMBOLE\_UTILISATEUR = "X";**  **private static final String SYMBOLE\_SYSTEME = "O";**    **private JButton grille [][]; // La grille de boutons**  **private ModeleJeuMorpion modele; // Le modèle dont cette classe est une vue**  **private int taille ; // La taille de la grille** |

1. le code du constructeur de VuePlateauMorpion (2 pts)

mémorisation du modèle dans un attribut (0.25), GridLayout (0.25), allocation tableau de boutons (0.25), new de chaque bouton (0.25), ajout de chaque bouton dans la vue (0.25), ajout d'un écouteur (0.75)

|  |
| --- |
| **public VuePlateauMorpion (ModeleJeuMorpion m) {**  **// Mémoriser les paramètres dans les attributs**  **this.taille = m.getTaille() ;**  **this.modele = m ;**  **// Fixer la mise en page**  **setLayout (new GridLayout (taille, taille));**    **// Allouer une grille de boutons**  **grille = new JButton [taille][taille];**    **// Créer la grille avec tous les boutons et un écouteur pour chaque bouton**  **for (int l=0; l<taille; l++)**  **for (int c=0; c<taille; c++) {**  **// Créer un bouton**  **grille [l][c] = new JButton ();**  **grille [l][c].setFont(new Font("Courier",Font.BOLD,48));**  **// Ajouter le bouton dans la grille**  **add (grille [l][c]);**    **// Créer et ajouter un écouteur sur le bouton**  **// en mémorisant la ligne et la colonne**  **grille [l][c].addActionListener(new ControleurMorpion (l, c, modele, this));**  **}**  **}** |

1. le code de la méthode miseAJour () de VuePlateauMorpion (1.5 pts)

boucle sur tous les boutons (0.5), accéder au pion correspondant dans le plateau (0.5), afficher le bon texte sur le bouton (0.5)

|  |
| --- |
| **public void miseAJour() {**  **Plateau<Pion> plateau = modele.getPlateau();**  **// Pour chaque bouton de la grille**  **for (int l=0; l<taille; l++)**  **for (int c=0; c<taille; c++) {**  **Pion p = (plateau.get(l, c));**  **JButton b = grille [l][c];**  **// Afficher sur le bouton le symbole qui correspond au pion**  **if (p !=null) {**  **if (p instanceof PionUtilisateur)**  **b.setText(SYMBOLE\_UTILISATEUR);**  **else**  **b.setText(SYMBOLE\_SYSTEME);**    **// Désactiver ce bouton**  **b.setEnabled(false);**  **}**  **}**  **}** |

7) classe ControleurMorpion (3 pts)

1. implements ActionListener (0.5)
2. modele (0.25), vue (0.25), ligne et colonne (0.25 de bonus)
3. entete (0.5), créer un pion correspondant au bouton (0.5), appel de vue.miseAJour (0.5), appel de faireJouerSysteme (0.5)

|  |
| --- |
| **//-------------------------------**  **public class ControleurMorpion implements ActionListener {**  **int l, c ; // La ligne et colonne du pion contrôlé par ce contrôleur**  **Jouable m; // Le modèle de jeu**  **VuePlateauMorpion v ; // La vue**  **// Constructeur**  **public ControleurMorpion (int l, int c, Jouable m, VuePlateauMorpion v) {**  **this.l = l ;**  **this.c = c ;**  **this.m = m ;**  **this.v = v ;**  **}**    **//-----------------------------------------------**  **public void actionPerformed(ActionEvent arg0) {**  **// Mettre à jour le modèle en ajoutant un pion utilisateur en l, c**  **m.ajouter(new PionUtilisateur(l, c));**    **// Si l'utilisateur a gagné, dire à la vue de se mettre à jour (tous les pions en vert)**  **if (m.userWins()) {**  **System.out.println("L'utilisateur a gagne");**  **v.miseAJour ();**  **}**  **else {**  **// Sinon**  **// Dire à la vue de se mettre à jour**  **v.miseAJour ();**    **// Faire jouer le système**  **m.faireJouerSysteme();**    **// Dire à la vue de se mettre à jour :**  **v.miseAJour ();**    **// Si le système a gagné, afficher un message dans la console et mettre à jour la vue**  **if (m.systemWins()) {**  **System.out.println("Le systeme a gagne");**  **v.miseAJour ();**  **}**  **}**  **}**  **}** |

8) Mémorisation d’une partie (1.5 pts)

appel de la méthode de sérialisation quand la partie est finie ou s'il y a un bouton / menu "sauvegarder" qui est cliqué (0.5), liste des instructions (1)

|  |
| --- |
| **On peut appeler la méthode ecrirePartie(), par exemple dans actionPerformed dans les cas où l’utilisateur a gagné ou dans le cas où le système a gagné.**  **On peut aussi ajouter un menu avec un bouton “Sauvegarder” et appeler cette méthode dans l’écouteur associé à ce bouton.**  **Voici la méthode à définir dans ModeleJeuMorpion et à déclarer dans Jouable :**  **public void ecrirePartie() {**  **ObjectOutputStream oos = null;**  **try {**  **FileOutputStream fichier = new FileOutputStream("partie.ser");**  **oos = new ObjectOutputStream(fichier);**  **oos.writeObject(listeDesCoups);**  **oos.flush();**  **} catch (final java.io.IOException e) {**  **e.printStackTrace();**  **}**  **}** |