Exercices de Patterns (6)

Le code suivant a été développé en implémentant expressément cinq patterns. Attention, implémenter n'est pas la même chose qu'utiliser un pattern déjà existant en Java. Quels sont ces cinq patterns ? Indiquer la correspondance entre la théorie et le code fourni. Il peut être utile de dessiner un diagramme de classe simplifié (sans attributs ni méthodes)

ConstantesJeu.java :

```
package vie;
public interface ConstantesJeu {
  int NBR_LIGNES = 40; // Nombre de lignes de cellules dans le jeu
  int NBR_COLONNES = 40; // Nombre de colonnes dans le jeu
  int PAD_EN_LARGEUR = 50; // Largeur en pixels du pad autour du jeu
  int PAD_EN_HAUTEUR = 100; // Hauteur en pixels du pad autour du jeu
  int LARGEUR_GRILLE = 400; // Largeur en pixels de la grille
  INT HAUTEUR_GRILLE = 400; // Hauteur en pixels de la grille
}
```

JeuDeLaVieGUI.java :

```
package vie;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JeuDeLaVieGUI extends JFrame implements ConstantesJeu, Espion {
  private final static Color COULEUR_VIVANT_PAR_DEFAUT = Color.RED;
  private final static Color COULEUR_MORT_PAR_DEFAUT = Color.BLACK;
  private JeuDeLaVie jeu;
  private JeuDeLaViePanel vue;
  private JPanel contentPane;
  private JButton boutonAvance;
  public JeuDeLaVieGUI(String titre, JeuDeLaVie jeu) {
    this(titre, jeu, COULEUR_VIVANT_PAR_DEFAUT,COULEUR_MORT_PAR_DEFAUT);
  }
  public JeuDeLaVieGUI(String titre, JeuDeLaVie jeu, Color couleurVie,
                       Color couleurMort){
    super(titre);
    this.jeu = jeu;
    vue = new JeuDeLaViePanel(jeu, couleurVie, couleurMort);
    boutonAvance = new JButton("Avance");
    contentPane = new JPanel();
    jeu.attacher(this);
    initialiser();
  }
  // Construit le GUI et permet au bouton "Avance" de passer à
  // génération suivante.
  private void initialiser(){
    this.setSize(vue.getWidth()+PAD_EN_LARGEUR,
                 vue.getHeight()+PAD_EN_HAUTEUR);
    contentPane.setLayout(new FlowLayout());
    contentPane.add(vue);
    contentPane.add(boutonAvance);
    this.setContentPane(contentPane);
    this.setVisible(true);
    boutonAvance.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        jeu.avancer();
      }
    });
    this.setDefaultCloseOperation(DISPOSE ON CLOSE);
  }
  // Redessine pour afficher les changements.
  public void utiliserRenseignements(){
    repaint();
  }
}
```

JeuDeLaViePanel.java :

```
package vie;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.BorderFactory;
import javax.swing.JPanel;
public class JeuDeLaViePanel extends JPanel implements ConstantesJeu {
  private JeuDeLaVie jeu;
  private int hauteurCellule;
  private int largeurCellule;
  private Color couleurVie;
  private Color couleurMort;
  public JeuDeLaViePanel(JeuDeLaVie jeu, Color couleurVie, Color couleurMort){
    this.jeu = jeu;
    this.largeurCellule = LARGEUR_GRILLE / jeu.getColonnes();
    this.hauteurCellule = HAUTEUR_GRILLE / jeu.getLignes();
    this.couleurVie = couleurVie;
    this.couleurMort = couleurMort;
    initialiser();
  }
  private void initialiser(){
    this.setSize(LARGEUR_GRILLE, HAUTEUR_GRILLE);
    this.setPreferredSize(this.getSize());
    this.setBackground(couleurMort);
    this.setBorder(BorderFactory.createLineBorder(Color.black));
    // Ajouter un listener sur la souris afin de mettre
    // la cellule où se trouve la souris on ou off.
    this.addMouseListener(new MouseAdapter(){
      public void mousePressed(MouseEvent e) {
        jeu.toggle(e.getPoint().y / hauteurCellule,e.getPoint().x / largeurCellule);
      }
    });
  }
  public void paintComponent(Graphics g){
    super.paintComponent(g);
    dessinerGrille(g);
    dessineVie(g);
  }
  // Dessine les lignes de la grille.
  private void dessinerGrille(Graphics g){
    g.setColor(couleurVie);
    for(int i = 0; i <= jeu.getLignes(); i++){
      g.drawLine(0, i*hauteurCellule,LARGEUR_GRILLE,i*hauteurCellule);
    }
    for(int i = 0; i <= jeu.getColonnes(); i++){
      g.drawLine(i*largeurCellule, 0,i*largeurCellule,HAUTEUR_GRILLE);
    }
  }
  // Dessine toutes les cellules en vie.
  private void dessineVie(Graphics g){
    g.setColor(couleurVie);
    for(int i =0; i < jeu.getLignes(); i++){
      for(int j = 0; j < jeu.getColonnes(); j++){
        if ( jeu.estVivante(i,j) ){
          g.fillRect(j*largeurCellule, i*hauteurCellule,largeurCellule, hauteurCellule);
        }
      }
    }
  }
}
```

Espion.java :

```
package vie;
public interface Espion {
  void utiliserRenseignements();
}
```

JeuDeLaVie.java :

```
package vie;
import java.util.*;
/** Cette classe implémente le jeu de la vie */
public class JeuDeLaVie {
  private int lignes;
  private int colonnes;
  private Cellule grille[][];
  private List<Espion> espions;
  private Parcourir parcourt;
  public JeuDeLaVie(int lignes, int colonnes, Parcourir parcourt) {
    this.lignes = lignes;
    this.colonnes = colonnes;
    this.parcourt = parcourt;
    grille = new Cellule[lignes][colonnes];
    espions = new ArrayList<Espion>();
    for(int i =0; i < lignes; i++){
      for(int j=0; j < colonnes; j++){
        grille[i][j] = new Cellule(i, j);
      }
    }
  }
  protected void effacerGrille(Cellule[][] g){
    for(int i =0; i < lignes; i++){
      for(int j=0; j < colonnes; j++){
        g[i][j] = new Cellule(i,j);
      }
    }
  }
  public int getLignes(){
    return lignes;
  }
  public int getColonnes(){
    return colonnes;
  }
  public boolean estVivante(int li, int co){
    return grille[li][co].estVivante();
  }
  public Cellule cellulleEn(int li, int co){
    return grille[li][co];
  }
  // Inverse le statut de la cellule de position li,co
  public void toggle(int li, int co){
    grille[li][co].toggle();
    prévenirEspions();
  }
  // Cette méthode implemente les règles du Jeu de la Vie.
  // Pour chaque cellule,
  // on trouve le nombre de voisins et on rend la cellule vivante selon
  // les règles définies dans le parcourt
  public void avancer(){
    ArrayList<Activité> activités = new ArrayList<Activité>();
    for(int i = 0; i < lignes; i++)
      for(int j = 0; j < colonnes; j++)
        grille[i][j].générer(this, activités, parcourt);
    for (Activité activité: activités)
        activité.activer();
    prévenirEspions();
  }
  // Ajoute un espion.
  public void attacher(Espion espion){
    if (espion == null) return; this.espions.add(espion);
  }
  // Supprime un espion.
  public void détacher(Espion espion){
    this.espions.remove(espion);
  }
  // Informe tous les espions de tenir compte des renseignements obtenus.
```

```
  public void prévenirEspions(){
    for (Espion espion: espions)
      espion.utiliserRenseignements();
  }
}
```

Cellule.java :

```
package vie;

import java.util.*;

public class Cellule {
  private int ligne;
  private int colonne;
  private Situation situation;

  public Cellule(int ligne, int colonne) {
    this.ligne = ligne;
    this.colonne = colonne;
    this.situation = EstMort.getInstance();
  }

  public void vit() {
    situation = situation.vit();
  }

  public void meurt() {
    situation = situation.meurt();
  }

  public boolean estVivante() {
    return situation.estVivante();
  }

  public void toggle() {
    situation = situation.toggle();
  }

  public void ajouterAuxVoisinsVivants(List<Situation> voisinsVivants) {
    situation.ajouterAuxVoisinsVivants(voisinsVivants);
  }

  // Compte le nombre de voisins vivants de cette cellule ci dans le jeu
  public int nombreDeVoisins(JeuDeLaVie jeu) {
    int x = 0;
    int y = ligne - 1;
    List<Situation> voisinsVivants = new ArrayList<Situation>();
    if (y < 0) {
      y = jeu.getLignes() - 1;
    }
    for (int liCpt = 1; liCpt <= 3; liCpt++) {
      x = colonne - 1;
      if (x < 0) {
        x = jeu.getColonnes() - 1;
      }
      for (int coCpt = 1; coCpt <= 3; coCpt++) {
        if (x != colonne || y != ligne) {
          jeu.cellulleEn(y, x).ajouterAuxVoisinsVivants(voisinsVivants);
        }
        x = (x + 1) % jeu.getColonnes();
      }
      y = (y + 1) % jeu.getLignes();
    }
    return voisinsVivants.size();
  }

  public void générer(JeuDeLaVie jeu, List<Activité> activités, Parcourir parcourt) {
    situation.générer(this, jeu, activités, parcourt);
```

```
  }
}
```

Situation.java :

```
package vie;

import java.util.List;

public abstract class Situation {
  public abstract Situation vit();

  public abstract Situation meurt();

  public abstract Situation toggle();

  public abstract boolean estVivante();

  public abstract void ajouterAuxVoisinsVivants(List<Situation> voisinsVivants);

  public abstract void générer(Cellule cellule, JeuDeLaVie jeu, List<Activité>
activités,
      Parcourir visiteur);
}
```

EstVivante.java :

```
package vie;

import java.util.List;

public class EstVivante extends Situation {
  private static EstVivante instance = null;

  private EstVivante() {}

  public static EstVivante getInstance() {
    if (instance == null)
      instance = new EstVivante();
    return instance;
  }

  public Situation vit() {
    return this;
  }

  public Situation meurt() {
    return EstMorte.getInstance();
  }

  public boolean estVivante() {
    return true;
  }

  public Situation toggle() {
    return meurt();
  }

  public void ajouterAuxVoisinsVivants(List<Situation> voisinsVivants) {
    voisinsVivants.add(this);
  }

  public void générer(Cellule cellule, JeuDeLaVie jeu, List<Activité> activités,
      Parcourir parcourt) {
    parcourt.parcourtCelluleVivante(cellule, jeu, activités);
  }
}
```

EstMorte.java :

```
package vie;

import java.util.List;

public class EstMorte extends Situation {
  private static EstMorte instance = null;

  private EstMorte() {}

  public static EstMorte getInstance() {
    if (instance == null)
      instance = new EstMorte();
    return instance;
  }

  public Situation vit() {
    return EstVivante.getInstance();
  }

  public Situation meurt() {
    return this;
  }

  public boolean estVivante() {
    return false;
  }

  public Situation toggle() {
    return vit();
  }

  public void ajouterAuxVoisinsVivants(List<Situation> voisinsVivants) { // ne pas
s'ajouter : on
                                                                        // est mort
  }

  public void générer(Cellule cellule, JeuDeLaVie jeu, List<Activité> activités,
      Parcourir parcourt) {
    parcourt.parcourtCelluleMorte(cellule, jeu, activités);
  }
}
```

Activité.java :

```
package vie;

public abstract class Activité {
  private Cellule cellule;

  public Activité(Cellule cellule) {
    this.cellule = cellule;
  }

  // Envoyer la requête sauvée (vit ou meurt) à la cellule. public abstract void
activer();
  public Cellule getCellule() {
    return cellule;
  }
}
```

Vit.java :

```
package vie;

public class Vit extends Activité {
  public Vit(Cellule cellule) {
    super(cellule);
  }

  public void activer() {
    getCellule().vit();
  }
}
```

Fichier Meurt.java :

```
package vie;

public class Meurt extends Activité {
  public Meurt(Cellule cellule) {
    super(cellule);
  }

  public void activer() {
    getCellule().meurt();
  }
}
```

Parcourir.java :

```
package vie;

import java.util.*;

public abstract class Parcourir {
  public abstract void parcourtCelluleVivante(Cellule cellule, JeuDeLaVie jeu,
      List<Activité> activités);

  public abstract void parcourtCelluleMorte(Cellule cellule, JeuDeLaVie jeu,
      List<Activité> activités);
}
```

ParcourtNormal.java :

```
package vie;

import java.util.List;

public class ParcourtNormal extends Parcourir {
  // Une cellule devient vivante si :
  // soit elle était vivante et a 2 ou 3 voisins vivants // soit elle était morte et a
exactement 3
  // voisins vivants.
  // Dans les autres cas la cellule meurt ou reste morte.
  public void parcourtCelluleVivante(Cellule cellule, JeuDeLaVie jeu, List<Activité>
activités) {
    int n = cellule.nombreDeVoisins(jeu);
    if (n != 2 && n != 3) {
      activités.add(new Meurt(cellule));
    }
  }

  public void parcourtCelluleMorte(Cellule cellule, JeuDeLaVie jeu, List<Activité>
activités) {
    int n = cellule.nombreDeVoisins(jeu);
    if (n == 3) {
      activités.add(new Vit(cellule));
    }
  }
}
```

ParcourtFort.java :

```
package vie;

import java.util.List;

public class ParcourtFort extends Parcourir {
  public void parcourtCelluleVivante(Cellule cellule, JeuDeLaVie jeu, List<Activité>
activités) {
    int n = cellule.nombreDeVoisins(jeu);
    if (n != 2 && n != 3) {
      activités.add(new Meurt(cellule));
    }
  }

  public void parcourtCelluleMorte(Cellule cellule, JeuDeLaVie jeu, List<Activité>
activités) {
    int n = cellule.nombreDeVoisins(jeu);
    if (n == 3 || n == 2) {
      activités.add(new Vit(cellule));
    }
  }
}
```

Enfin une programme principal met ceci en œuvre :

Principale.java :

```
package vie;

import java.awt.*;

public class Principale implements ConstantesJeu {
  public static void main(String[] args) {
    Parcourir parcourt = new ParcourtNormal();
    //Parcourir parcourt = new ParcourtFort();;
    JeuDeLaVie jeu = new JeuDeLaVie(NBR LIGNES,NBR COLONNES, parcourt);
    JeuDeLaVieGUI vie = new JeuDeLaVieGUI("Le Jeu de la Vie",jeu,Color.red,
Color.black);
  }
}
```

Nom du Pattern :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

Nom du Pattern :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

Exercices de Patterns (6)

<u>Nom du Pattern</u> :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

<u>Nom du Pattern</u> :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

<u>Nom du Pattern</u> :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**