UE Ingénierie Logicielle - Design Patterns - Travaux d'exercices n°2

EXERCICE 1

On considère une application interactive « View-Book-Search » qui permet de réserver des voyages (le code fourni -cf. annexe 1- fait abstraction des aspects métier de la réservation).
Le problème est d'ajouter à cette application une fonctionnalité de paiement. Cette fonctionnalité doit être accessible au moyen d'un nouveau bouton Pay qui vient s'ajouter aux boutons View, Book et Search de l'interface, tel que :

- quand on clique sur Pay, View est activé, Book et Search sont désactivés,
- quand on clique sur Book, Pay est activé,
- quand on clique sur View et Search, Pay n'est pas activé.

Pour réaliser cette évolution, quelles modifications faut-il apporter ? Quelles sont les classes touchées ?
Proposer une réorganisation de l'application qui permettrait de limiter l'impact de l'évolution à apporter.
Concevoir la solution :

- Etape 1 : refabrication de l'application
- Etape 2 : ajout du bouton "Pay"

EXERCICE 2

On considère l'application dont le code donné est donné dans l'annexe 2.

Etendre cette application sans apporter de modification à l'existant (en dehors de la classe `TestStudio`) dans le but de permettre à un objet instance de `Studio` de demander à une instance de `Capitaine` ou de `Perroquet` de « chanter ».
Justifier la réponse, indiquer le design pattern utilisé et préciser quels sont les participants et leurs rôles.

EXERCICE 3

Vous intervenez dans le cadre d'un projet de maintenance d'une application de robotique développée par un ancien étudiant du Master MIAGE qui a mis en application ses connaissances sur les patrons de conception. Cette application met en œuvre des robots localisés géographiquement, capables de connaître leur position et de se déplacer en une position donnée ; pour cela, les robots peuvent utiliser différentes méthodes pour calculer le chemin à suivre.

Malheureusement, toute la documentation sur l'application a disparu ! Vous devez donc reprendre le code (donné dans l'annexe 3.1).

1. *Donnez-le diagramme de classes et retrouvez les patrons de conception utilisés. Pour chacun identifiez les différents participants. Expliquez.*

2. *Quel est l'impact sur le code existant des évolutions ci-dessous ?*
   - *Ajout d'une classe `CC6` qui implante `CalculateurDeChemin`*
   - *Ajout d'une classe `RobotBis` qui utilise un `CalculateurDeChemin` comme `Robot`*

Vous devez maintenant intégrer à l'application la méthode de calcul de chemin qui est définie dans la classe CCR (dont le code est donné dans l'annexe 3.2) en réutilisant cette classe sans la modifier (afin de pouvoir intégrer des robots qui utilisent cette méthode).

3. *Quel design pattern devez-vous utiliser ? Expliquez. Complétez le diagramme de classe et préciser le rôle des participants.*

EXERCICE 4

Le programme Java donné en annexe 4 définit :

- un système d'objets interconnectés, instances des classes `C1`, `C2`, `C3` et `C4`,
- un client qui requiert des services sur ces objets.

L'annexe 4 montre aussi la trace de l'exécution.

1. Retrouver le diagramme de classe.

2. On cherche à simplifier pour les clients l'accès au système d'objets interconnectés et à réduire le couplage entre les classes qui définissent ce système et les clients. Pour cela, on veut reprendre le code et introduire une façade (instance de la classe `Façade`) entre le système d'objets et le client.

   Concevoir la solution et donner le code de la classe `Façade`.
   Réécrire la classe `Client` et, dans la classe `Test`, modifier le code de création du client en conséquence.
   Modifier maintenant la classe `Façade` de sorte qu'elle offre directement le service `beta()` au client.

3. Considérons le cas général d'un objet client qui réalise des appels de méthodes vers différents objets d'un système d'objets par l'intermédiaire d'une façade conformément au pattern Façade.

   Si on effectue des modifications au sein des classes qui définissent le système d'objets et son organisation, quel est (de manière générale) l'impact sur :
   - *les classes qui définissent les clients ?*
   - *les classes qui définissent les façades ?*

   Justifier.

4. En quoi le pattern Façade diffère-t-il du pattern Proxy ?

EXERCICE 5

Décidément, la documentation est mal gérée… Elle a encore disparu !

Dans une démarche d'ingénierie inverse, reconstituer le diagramme de classes à partir du code et de la trace d'exécution ci-dessous, et le documenter. Quels sont les types concrets des objets du « supertype » C ? Quelle est la particularité de D ?

```
C c;
c = new CC();
c.m(); System.out.println();
D dc1, dc2, dc1bis;
dc1 = new DC1(c);
c = dc1; c.m(); System.out.println();
dc2 = new DC2(c);
c = dc2; c.m(); System.out.println();
dc1bis = new DC1(c);
c = dc1bis; c.m(); System.out.println();
```

Trace d'exécution :

```
m() de CC
m() de CC + m() de DC1
m() de CC + m() de DC1 + m() de DC2
m() de CC + m() de DC1 + m() de DC2 + m() de DC1
```

ANNEXE 1

```
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

//Button interface
interface Command {
	void execute();
}
//A concrete button
class BtnView extends JButton implements Command {
	BtnSearch btnSearch;
	BtnBook btnBook;
	LblDisplay show;
	BtnView(ActionListener al) {
		super("View");
		addActionListener(al);
	}
	void register(BtnSearch s, BtnBook b, LblDisplay d) {
		btnSearch = s;
		btnBook = b;
		show = d;
	}
	public void execute() {
		this.setEnabled(false);
		btnSearch.setEnabled(true);
		btnBook.setEnabled(true);
		show.setText("viewing...");
	}
}
//A concrete button
class BtnSearch extends JButton implements Command {
	BtnView btnView;
	BtnBook btnBook;
	LblDisplay show;
	BtnSearch(ActionListener al) {
		super("Search");
		addActionListener(al);
	}
	void register(BtnView v, BtnBook b, LblDisplay d) {
		btnView = v;
		btnBook = b;
		show = d;
	}
	public void execute() {
		this.setEnabled(false);
		btnView.setEnabled(true);
		btnBook.setEnabled(true);
		show.setText("searching...");
	}
}
```

```
//A concrete button
class BtnBook extends JButton implements Command {
	BtnView btnView;
	BtnSearch btnSearch;
	LblDisplay show;
	BtnBook(ActionListener al) {
		super("Book");
		addActionListener(al);
	}
	void register(BtnSearch s, BtnView v, LblDisplay d) {
		btnSearch = s;
		btnView = v;
		show = d;
	}
	public void execute() {
		this.setEnabled(false);
		btnView.setEnabled(true);
		btnSearch.setEnabled(true);
		show.setText("booking...");
	}
}

class LblDisplay extends JLabel {
	LblDisplay() {
		super("Just start...");
		setFont(new Font("Arial", Font.BOLD, 24));
	}
}

public class ViewBooksearchDemo extends JFrame implements ActionListener {
	ViewBooksearchDemo() {
		JPanel p = new JPanel();
		BtnView v = new BtnView(this);
		BtnBook b = new BtnBook(this);
		BtnSearch s = new BtnSearch(this);
		LblDisplay d = new LblDisplay();
		v.register(s, b, d);
		b.register(s, v, d);
		s.register(v, b, d);
		p.add(v);
		p.add(b);
		p.add(s);
		getContentPane().add(d, "North");
		getContentPane().add(p, "South");
		setSize(400, 200);
		setVisible(true);
	}
	public void actionPerformed(ActionEvent ae) {
		Command comd = (Command) ae.getSource();
		comd.execute();
	}
	public static void main(String[] args) {
		new ViewBooksearchDemo();
	}
}
```

ANNEXE 2

```
public interface Chanteur {
	public void chanter();
}

public class RossignolChanteur implements Chanteur {
	public void chanter() {
		System.out.println("Aaaahh ! Je riiiis de me voir si belle...");
	}
}

public class Capitaine {
	public void jurer() {
		System.out.println("Mille milliards de mille sabords !!!");
	}
}

public class Perroquet {
	public void jaser() {
		System.out.println("Allô-ô-ô-ô, j'écou-ou-te ! j'écou-ou-te !");
	}
}

public class Studio {
	Chanteur leChanteur;
	public Studio(Chanteur leChanteur) {
		this.leChanteur = leChanteur;
	}
	public void enregistrerChant() {
		// ouvrir les micros et lancer l'enregistrement sonore
		System.out.println("Début d'enregistrement de : "+leChanteur);
		leChanteur.chanter();
		// terminer l'enregistrement et fermer les micros
		System.out.println("Fin d'enregistrement de : "+leChanteur);
	}
}

public class TestStudio {
	public static void main(String[] args) {
		Studio studio;
		RossignolChanteur bianca = new RossignolChanteur();
		studio = new Studio(bianca);
		studio.enregistrerChant();
	}
}
```

ANNEXE 3.1

```
// Chemin, Position, Prévision sont définis par ailleurs

public class Robot {
	private CalculateurDeChemin cc;
	public Robot(CalculateurDeChemin cc) {
		this.cc = cc;
	}
	public Position getPosition() {
		… // retourner la position courante
	}
	public void allerA(Position p) { // réalise le déplacement du robot
		… // se préparer à se déplacer
		Chemin c = cc.calculer(getPosition(),p);
		… // suivre le chemin c
	}
}

public interface CalculateurDeChemin {
	public Chemin calculer(Position départ, Position arrivée);
		// construire un chemin pour aller de « départ » à « arrivée »
}

public class CC1 implements CalculateurDeChemin{
	public Chemin calculer(Position départ, Position arrivée) {// méthode 1
		…
	}
}
public abstract class CC2 implements CalculateurDeChemin {
	public Chemin calculer(Position départ, Position arrivée) {// méthode 2
		… // faire plein de choses
		Prévision prévision = prévoirTrafic();
		… // faire plein d'autres choses
	}
	protected abstract Prévision prévoirTrafic();
}
public class CC3 extends CC2 {
	protected Prévision prévoirTrafic() {
		… // prévoir le trafic d'une certaine façon
	}
}
public class CC4 extends CC2 {
	protected Prévision prévoirTrafic() {
		… // Prévoir le trafic d'une autre façon
	}
}
public class CC5 extends CC1 {
	public Chemin calculer(Position départ, Position arrivée) {// méthode 5
		Chemin c = super.calculer(départ, arrivée);
		… // Optimiser le chemin c
	}
}
```

ANNEXE 3.2

```
public class CCR {
	protected Position pDépart; // position de départ
	protected Position pArrivée; // position à atteindre
	public Chemin calculer(){// méthode réutilisable
		… // Construire un chemin en fonction de pDépart et pArrivée
	}
}
```

ANNEXE 4

```
class C1 {
      C2 c2; C3 c3;
      void set(C2 c2, C3 c3){this.c2 = c2; this.c3 = c3;}
      void a(){System.out.println("a de "+this); c2.a(); c3.a();}
      void b(){System.out.println("b de "+this); c3.b();}
}
class C2 {
      C3 c3; C4 c4;
      void set(C3 c3, C4 c4) {this.c3 = c3; this.c4 = c4;}
      void a(){System.out.println("a de "+this); c4.a();}
      void b(){System.out.println("b de "+this); c3.b();}
}
class C3 {
      C2 c2; C4 c4;
      void set(C2 c2, C4 c4){this.c2 = c2; this.c4 = c4;}
      void a(){System.out.println("a de "+this); c2.a();}
      void b(){System.out.println("b de "+this); c4.b();}
}
class C4 {
      void a(){System.out.println("a de "+this);}
      void b(){System.out.println("b de "+this);}
}
class Client {
      C1 monPremierC1; C1 monSecondC1; C3 monC3; C4 monC4;
      Client(C1 monPremierC1, C1 monSecondC1, C3 unC3, C4 unC4) {
            this.monPremierC1 = monPremierC1;
            this.monSecondC1 = monSecondC1;
            this.monC3 = unC3;
            this.monC4 = unC4;      }
      void alpha(){monPremierC1.a(); monC4.b();}
      void beta(){monSecondC1.a(); monC3.b();}
}
public class Test {
      public static void main(String[] args){
            C1 k1 = new C1(); C1 l1 = new C1();
            C2 k2 = new C2(); C3 k3 = new C3(); C4 k4 = new C4();

            k1.set(k2, k3); l1.set(k2, k3); k2.set(k3, k4); k3.set(k2, k4);

            Client client = new Client(k1,l1,k3,k4);

            System.out.println("############"); client.alpha();
            System.out.println("############"); client.beta();
      }
}
```

```
############
a de C1@190d11
a de C2@a90653
a de C4@de6ced
a de C3@c17164
a de C2@a90653
a de C4@de6ced
b de C4@de6ced
############
a de C1@1fb8ee3
a de C2@a90653
a de C4@de6ced
a de C3@c17164
a de C2@a90653
a de C4@de6ced
b de C3@c17164
b de C4@de6ced
```