Laboratoire GEN *No I*

Laboratoire «Design patterns»

MVC - MODELE OBSERVABLE-OBSERVE

Eric Lefrançois – 25 Sept 2011

PREAMBULE

Un père Noël passe et repasse avec son chariot au dessus d'une campagne vallonnée pendant que la neige tombe à gros flocons.

Le père Noël est un objet actif dont les déplacements sont observés par le panneau graphique, chargé de le représenter.

Ce même panneau graphique observe chacun des flocons, qui constituent autant d'objets graphiques.

OBJECTIFS

Mise en oeuvre du modèle MVC, et en particulier du modèle «Observable-Observé».

1 Mise en place

Démarrer un projet «Together » basé sur les fichiers donnés par l'enseignant.

En tout et pour tout:

Noel.java *Programme à compléter*

PNoel.gif

Lune.gif

Montagne.gif

Votre première fois avec Together ?

- Créer un répertoire qui constituera votre «répertoire de travail»
- Dans ce répertoire :

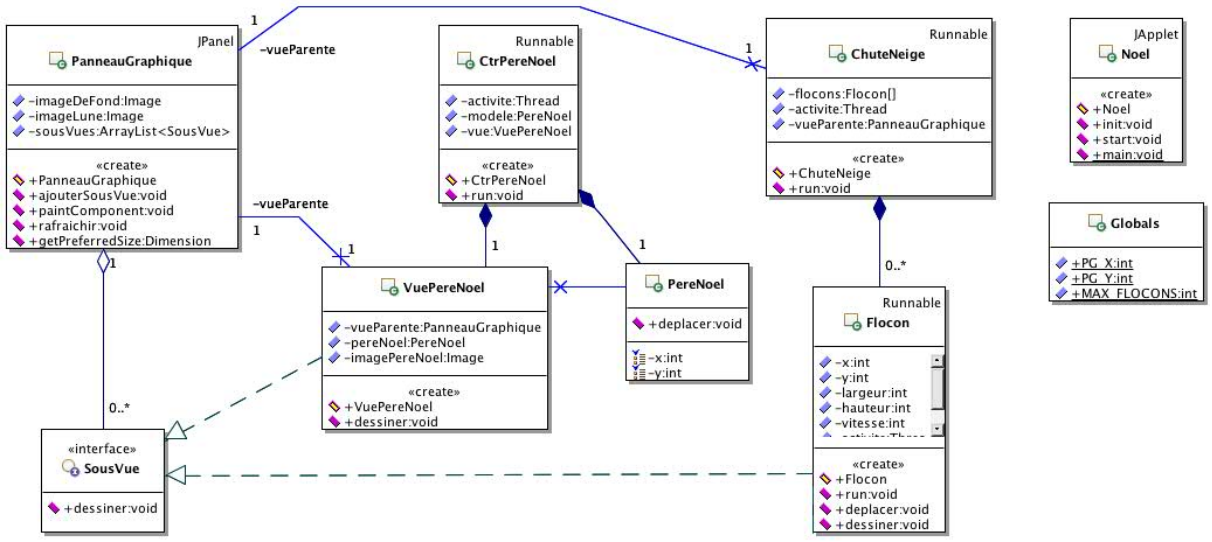
  Placer les 3 fichiers images ainsi que le source `Noel.java`
- Lancer Together..
  1. Conserver à priori le répertoire de travail proposé par défaut par Together (`/Users/admin/Documents/workspace`)
  2. Créer un nouveau projet de type « Java modeling »
  3. Donner un nom à ce projet
  4. Sélectionner « Créer un projet à partir d'une source existante », en choisissant le répertoire contenant vos fichiers.
  5. Pour le reste, conserver les options par défaut proposées par Together, en contrôlant le cas échéant la machine virtuelle utilisée par défaut.

Si tout s'est bien déroulé, on doit voir apparaître le diagramme de classes opéré par «reverse engineering»

Compiler et exécuter le programme

Pour compiler et exécuter le programme, il suffit de cliquer sur le triangle vert situé dans la barre d'outils de Together.

A l'occasion de la première exécution, Together demande à l'utilisateur d'indiquer le nom de la classe principale («Select class with main..»). Dans notre cas, il suffit alors de sélectionner la seule classe qui dispose d'une fonction «`main`»: la classe «`Noel`».



2 *Diagramme de classes du programme*

Essayez d'obtenir le diagramme UML présenté plus haut!

- Complétez le diagramme généré automatiquement par Together..
- Symboles de composition (losanges pleins), d'agrégation (losange vide),
- Cardinalités des associations
- Labels des associations ou rôles des objets

Observez l'effet de ces modifications sur le code du programme..

Tâtez le terrain en essayant d'ajouter des attributs ou des méthodes bidon à partir du diagramme de classes ou à partir du code lui-même. Observez l'effet de ces adjonctions sur l'artefact correspondant (diagramme ou code).

Vous aurez alors constaté l'impact de la modification du diagramme de classes sur le code, et notamment l'adjonction de commentaires spéciaux, utilisés par Together pour conserver les différents symboles graphiques que vous aurez rajoutés.

3 *Compréhension du programme*

Lire le code du programme et tâcher de comprendre le rôle des différents objets mis en oeuvre..

En gros:

`Noel` (classe principale)
Création des 3 principaux objets: le «Père Noël», la «Chute de neige» et le «Panneau graphique».

`PereNoel`
Représente le «Père Noel», un objet actif («Runnable»), qui passe son temps à traverser le panneau graphique. Ce dernier est implémenté au moyen de 3 classes : le contrôleur, le modèle et la vue correspondante.

`ChuteNeige`
Contient un tableau de 300 flocons. La chute de neige est un objet actif («Runnable») dont l'activité se limite à la création «en douceur» des 300 flocons, à raison d'un nouveau flocon toutes les 100 msec.
La classe `ChuteNeige` rassemble à la fois le modèle et le contrôleur de la chute de neige. Il n'y a pas de vue associée. En revanche, chaque flocon aura une vue associée.

`Flocon`
Représente un flocon, un objet actif («Runnable»), qui prend naissance en haut du panneau graphique à une position x aléatoire, puis descend jusqu'au sol pour «renaître à nouveau» en haut du panneau graphique et ainsi de suite.

Ce programme, très gourmand en ressource machine, ne comporte pas moins de 301 threads tournant simultanément !!! Ce n'est pas forcément un exemple à suivre ..

En l'état, la classe `Flocon` rassemble à la fois le modèle, le contrôleur et la vue d'un flocon.

`PanneauGraphique`
Un `JPanel` dont la procédure de dessin («`paintComponent`») est dédiée principalement à l'affichage des images de fond, du père Noël et des flocons.
La procédure **`rafraichir`**, montrée ci-dessous, peut être invoquée par l'une ou l'autre des sous-vues contenues dans le panneau graphique, à savoir la sous-vue du père Noël, et les sous-vues de chaque flocon.

```
public void rafraichir (Object ssvue) {
        // Une sous-vue demande à etre rafraichie
        this.repaint();
}
```

L'instruction « `this.repaint()` » a pour effet d'invoquer indirectement la méthode «`paintComponent`» qui a pour effet de redessiner complètement la vue principale (qui est donc effacée !) avec toutes ses sous-vues : Père Noël et flocons.

Si plusieurs «`repaint()`» s'enchaînent les uns à la suite des autres très rapidement, une exécution unique «`paintComponent`» peut rassembler toute une série de «`repaint`» en fonction de ce que décide la machine virtuelle et de la rapidité du processeur.

4 P1: Afficher les déplacements du père Noël

Mettre en place le modèle Observable-Observé entre le père Noël (sujet observé) et la vue Père Noël (l'observateur).

Pour ce faire, utiliser la classe `Observable` et l'interface `Observer` mis à disposition par l'environnement Java.

A chaque fois que l'observateur recevra un message de mise à jour du sujet observé, la méthode **`rafraichir`** de `PanneauGraphique` devra être invoquée.

5 P2 : Un flocon MVC..

Restructurez la classe **`Flocon`** en adoptant le modèle MVC, découpez cette classe en deux classes :

- La class **`Flocon`**, qui rassemblera le contrôleur et le modèle de chaque flocon (un peu à la manière de la classe `ChuteNeige`).
- La classe **`VueFlocon`**, qui rassemblera la vue associée au flocon (un peu à la manière de la classe `VuePereNoel`).

Remarque de conception!!

Une vision MVC « jusquauboutiste » aurait pu nous inciter à découper `Flocon` en trois classes à l'instar du Père Noël. Mais cela n'est pas toujours justifié dans le cas de petites classes comme c'est le cas avec `Flocon`. On peut alors rassembler Modèle et Contrôleur dans la même classe.

6 P3: Afficher le déplacement des flocons

Vous l'aurez constaté, la réalisation du point précédent a pour effet d'afficher le déplacement des flocons !! Mais c'est trop facile...

Vous allez donc désactiver provisoirement la mécanique «Observable/Observé» du point précédent en plaçant l'instruction «`notifyObservers`» dans un commentaire.

Puis, vous allez mettre en place le modèle Observable-Observé au niveau de chaque flocon.

7 P4: Mettre en place une couche de neige

La neige s'accumule et forme une couche de neige ...

Indications:

- La couche de neige (une nouvelle classe..) peut être mise en oeuvre au moyen d'un tableau dont chaque case contient la hauteur de la couche en fonction de la coordonnée x du pixel.
- Chaque flocon peut être observé par la couche de neige (en plus du panneau graphique..), qui elle-même sera observée par sa vue.

8 P5: Diagramme de classes

Le programme est terminé..

Tâchez d'opérer un «print» du diagramme de classes que vous avez obtenu avec Together.

9 Code du programme à compléter

```
/*
Fichier:  Noel.java

Exemple d'utilisation du modèle Observer/Observable
Exécuter le programme en application autonome
Ou exécuter le programme en mode Applet en ouvrant une fenêtre de 315*315

Le répertoire contenant les classes à exécuter doit contenir les 3 fichiers image:
"Lune.gif", "Montagne.gif" et "PNoel.gif"

Date :        Eric Lefrançois, Août 2011
*/


import java.util.*;   // Random, Observer, Observable
import java.awt.*;
import javax.swing.*;
import java.awt.image.ImageObserver;
```

```
// -------------------------------------------------------------------------
class Globals {
// Constantes globales:  taille des composants

      public static int PG_X = 315;             // Taille en X du panneau graphique
      public static int PG_Y = 315;             // Taille en Y du panneau graphique
      public static int MAX_FLOCONS = 300;      // Nombre max. de flocons

}

// -------------------------------------------------------------------------
interface SousVue {
      public void dessiner (Graphics g);
}

// -------------------------------------------------------------------------
class ChuteNeige implements Runnable {
// Controleur et modele rassembles

      private Flocon[] flocons = new Flocon [Globals.MAX_FLOCONS];
      private Thread activite;

      private PanneauGraphique vueParente;

      public ChuteNeige (PanneauGraphique vue) {
            this.vueParente = vue;

            activite = new Thread (this);
            activite.start();
      }

      public void run () {
            // Activite limitee à la creation des flocons
            int nbFlocons=0;    // Nombre de flocons crees
            while (nbFlocons < Globals.MAX_FLOCONS) {
                  try {Thread.sleep(100);}catch (InterruptedException e) {}
                  flocons[nbFlocons]=new Flocon(vueParente);
                                                      // Creer un nouveau flocon
                  nbFlocons++;
            }
      }
}

// -------------------------------------------------------------------------
class Flocon implements Runnable, SousVue {
// Controleur, modele et vue pour un flocon
      private int x, y;             // Coordonnees courantes du flocon
      private int largeur;          // largeur du flocon (en pixels)
      private int hauteur;          // hauteur du flocon (en pixels)
      private int vitesse;          // DeltaT entre deux deplacements en Y

      private Thread activite;

      // Un generateur de nombres aleatoires
```

```
	public static Random rdGen = new Random();

	public Flocon (PanneauGraphique vueParente) {
		x = rdGen.nextInt(Globals.PG_X);
		y = 0;
		largeur = 1+rdGen.nextInt(3);		// Generation nb aleat. entre 1 et 4
		hauteur = largeur + rdGen.nextInt(2);
		vitesse = hauteur;			// Vitesse directement fonction de la taille

		vueParente.ajouterSousVue(this);

		activite = new Thread(this);
		activite.start();
	}

	public void run () {
		while (true) {
			try {Thread.sleep(50*(7-
vitesse));}catch(InterruptedException e) {}
			deplacer();
		}
	}

	public void deplacer () {
		boolean aDroite = rdGen.nextInt(2)==1;
		x = aDroite ? x+1 : x-1;
		y += vitesse ;
		if (y+hauteur> Globals.PG_Y+10) {
			x = rdGen.nextInt(Globals.PG_X);
			y = 0;
		}
	}

	public void dessiner (Graphics g) {
		g.setColor(Color.white);
		g.fillRect(x, y, largeur, hauteur);
		g.setColor(Color.lightGray);
		g.drawRect(x, y, largeur, hauteur);
	}
}

// ----------------------------------------------------------------------
class CtrPereNoel implements Runnable {
// Controleur du pere Noel
	private Thread activite;
	private PereNoel modele;
	private VuePereNoel vue;

	public CtrPereNoel (PanneauGraphique parent) {
		modele= new PereNoel();
		vue = new VuePereNoel(parent, modele);

		activite = new Thread (this);
		activite.start();

	}

	public void run () {
```

```
            while (true) {
                  try {Thread.sleep(50);}catch (InterruptedException e) {}
                  modele.deplacer();
            }
      }
}

class PereNoel {
// Modele du pere Noel
      private int x=0;
      private int y= Globals.PG_Y/4;;

      public int getX() {return x;}
      public int getY() {return y;}

      public void deplacer() {
            x += 2;
            if (x > Globals.PG_X+100) x = -200;
      }
}

class VuePereNoel implements SousVue {
// Vue du pere Noel
      private PanneauGraphique vueParente;
      private PereNoel pereNoel;
      private Image imagePereNoel;

      public VuePereNoel (PanneauGraphique parent, PereNoel modele) {
            vueParente = parent;
            vueParente.ajouterSousVue(this);
            pereNoel=modele;
            imagePereNoel =
                  Toolkit.getDefaultToolkit().getImage ("PNoel.gif");
      }

      public void dessiner(Graphics g) {
            g.drawImage
                  (imagePereNoel, pereNoel.getX(), pereNoel.getY(),
                                    (ImageObserver)vueParente);
      }
}

//-------------------------------------------------------------------------
class PanneauGraphique extends JPanel {
// Vue passive, pour affichage uniquement

      private  Image imageDeFond, imageLune;           // Images de fond

      private ArrayList<SousVue> sousVues= new ArrayList<SousVue>();

      // Constructeur
      public PanneauGraphique () {
            setBackground(new Color (0, 0, 65));
            // Chargement des images Montagne.gif, et Lune.gif, situees dans le repertoire
            // des classes.
            imageDeFond =
             Toolkit.getDefaultToolkit().getImage ("Montagne.gif");
```

```
            imageLune = Toolkit.getDefaultToolkit().getImage ("Lune.gif");
      }

      public synchronized void ajouterSousVue(SousVue ssVue) {
      // Synchronisation necessaire-La liste des sous-vues est accedees simultanement
      // par paintComponent et au moment de la creation des flocons (ajout des sous-vues)
            sousVues.add(ssVue);
      }

      public void paintComponent (Graphics g) {
            super.paintComponent(g);
            g.drawImage (imageDeFond, 0, Globals.PG_Y-44, this);
            g.drawImage
                  (imageLune, Globals.PG_X-100, Globals.PG_Y/4, this);

            /*Note .....
            Le "this", 4eme parametre de drawImage represente "l'image observer"".
            Cet objet controle le chargement de l'image en memoire (chargee
            habituellement depuis un fichier). Il est responsable de dessiner
            cette image de maniere asynchrone au reste du programme, au fur et à
            mesure que l'image se charge.
            Ainsi, le programmeur peut donner l'ordre de charger une image ("getImage"),
            puis il peut la dessiner aussitet (drawImage), sans attendre qu'elle
            soit chargee. La procedure drawImage retourne aussitot.
            L'image observer est implemente par la classe Component (dont herite
            la classe JPanel).
            Le cas echeant, il est possible de redefinir cet objet, ce qui permettrait
            de controler le chargement de l'image, d'attendre qu'elle soit entierement
            chargee avant de l'afficher, etc...
            */

            // Affichage des sous-vues (Pere Noel et flocons)
            synchronized(this) {
                  Iterator<SousVue> i=sousVues.iterator();
                  while (i.hasNext()) {
                        SousVue sv= (SousVue)i.next();
                        sv.dessiner(g);
                  }
            }
     }

      public void rafraichir (Object ssvue) {
      // Une sous-vue demande a être rafraichie
            this.repaint();
      }

   public Dimension getPreferredSize() {
  // Retourne la taille souhaitee pour le composant (remplace le "getSize")
        return new Dimension (Globals.PG_X, Globals.PG_Y);
   }

}
//---------------------------------------------------------------------------
public class Noel extends JApplet {
// Controleur principal
// Creation des "modeles", des "vues"
```

```
// Associations diverses


      public Noel () {}

      public void init () {
        getContentPane().setLayout (new BorderLayout ());

      // Creation de la vue principale
        PanneauGraphique panneauGraphique = new PanneauGraphique ();

       // Positionnement de la vue principale au centre de la fenetre
        getContentPane().add (panneauGraphique, BorderLayout.CENTER);


            new CtrPereNoel(panneauGraphique);
            new ChuteNeige(panneauGraphique);
   }
   public void start() {}

   public static void main (String[] arg) {
  // Point d'entree du programme
      JFrame f = new JFrame ();
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setTitle("Chablon");
      Noel a = new Noel();
      f.getContentPane().add (a, BorderLayout.CENTER);
      a.init ();
      f.pack ();
      f.setResizable (false);
      f.setVisible (true);
      a.start ();
  }
}
//---------------------------------------------------------------------------
```