Master STS Mention Informatique Parcours ISI
Unité 178EN004
Génie Logiciel et Conception Avancée

TD3

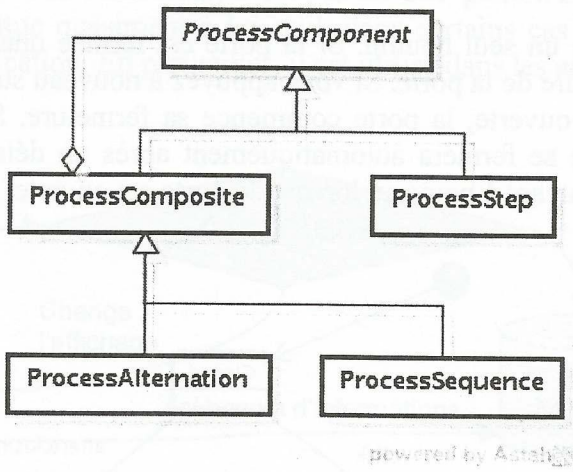
Exercice n°1 : contrôle de machines de production

Vous êtes en charge de la réalisation d'une application de contrôle des machines de production d'une usine de fabrication de feux d'artifice. Selon le type de feux d'artifice, le processus de fabrication peut comporter une ou plusieurs machines. Afin de connaître à tout moment le nombre de machine impliqué dans un processus, il faut concevoir une représentation des machines et une méthode *getMachineCount() : integer*.

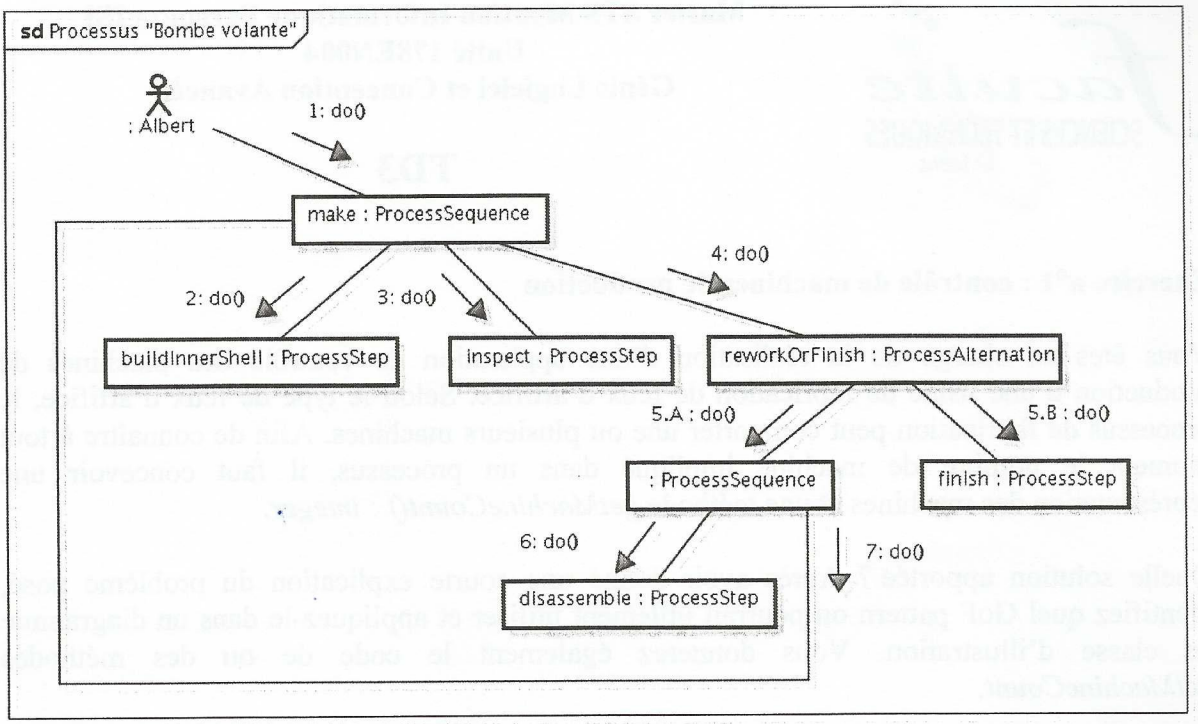
Quelle solution apportée ? Après avoir donné une courte explication du problème posé, identifiez quel GoF pattern on pourrait utilement utiliser et appliquez-le dans un diagramme de classe d'illustration. Vous donnerez également le code de ou des méthodes *getMachineCount*.

Exercice n°2 : processus de fabrication

L'usine Oozinoz fabrique des feux d'artifices. Un logiciel contrôle les processus de fabrication qui sont constitués d'étapes simples ou d'étapes composites séquentielles (constituées d'une succession d'étapes) ou d'étapes composites alternatifs (constituées elle-même d'étapes alternatives).



Par exemple le processus de fabrication d'une « bombe volante » est le suivant :

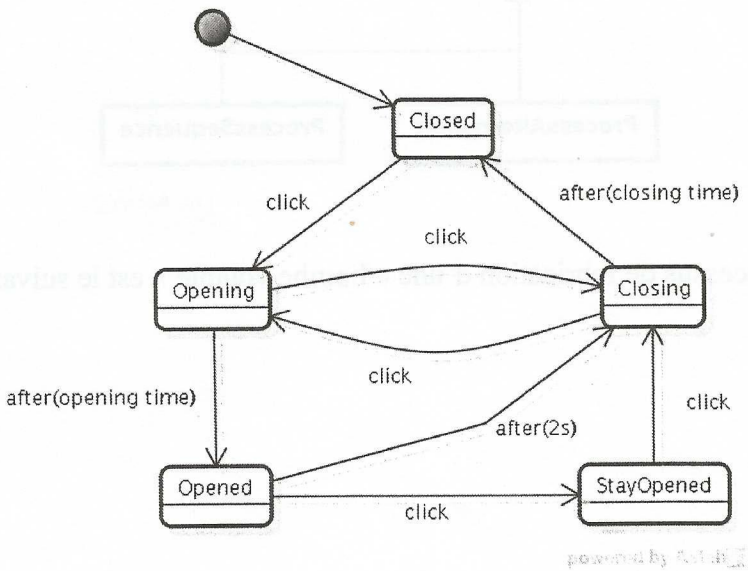


Le logiciel de contrôle doit permettre, par exemple, d'obtenir les étapes de processus de fabrication des « bombes volantes ».

Quelle solution apportée ? Identifiez quel GoF pattern on pourrait utilement utiliser et appliquez-le.

Exercice n°3 : porte carrousel

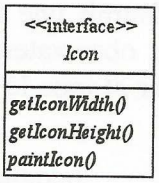
La porte fonctionne avec un seul bouton. Si la porte est fermée une pression sur le bouton lance le début de l'ouverture de la porte. Si vous appuyez à nouveau sur le bouton avant que la porte soit complètement ouverte, la porte commence sa fermeture. Si vous laissez la porte s'ouvrir entièrement, elle se fermera automatiquement après un délai de 2 secondes. Vous pouvez éviter cela en appuyant à nouveau lorsque la porte est ouverte.



On souhaite concevoir un logiciel de contrôle d'une porte carrousel. Identifiez quel GoF pattern on pourrait utilement utiliser et appliquez-le.

Exercice n°4 : les pochettes de CD

Vous voulez écrire une application qui affiche les pochettes de vos CD favoris. Les images des pochettes sont récupérés sur un service en ligne puis affichées dans votre application. Les images impléméntent l'interface Icon :



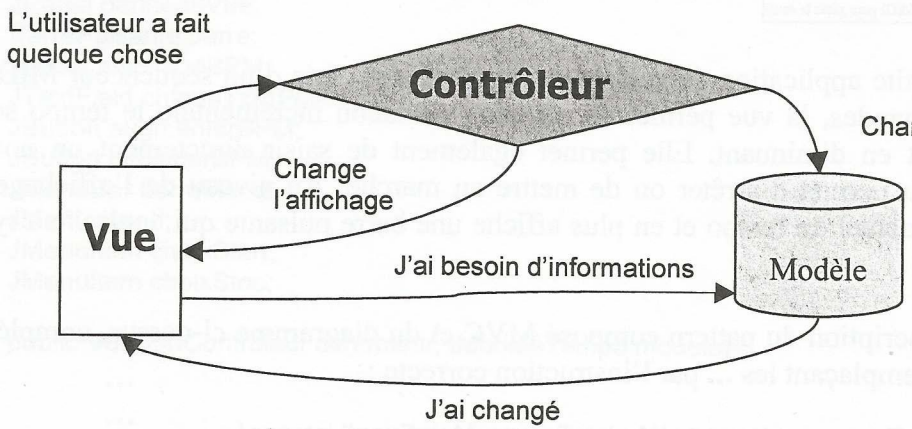
Le hic, c'est que selon la charge du réseau et le débit de votre connexion, le chargement d'une prochette de CD peut prendre un certain temps. Votre application doit donc afficher quelque chose pendant que l'image est téléchargée et ne pas bloquer le reste de l'application.

Quelle solution apportée ? Identifiez quel GoF pattern on pourrait utilement utiliser et appliquez-le.

Exercice n°5 : le pattern composée MVC

Il est évidemment possible d'utiliser pluiseurs pattern et de les faire collaborer au sien d'une même conception. On obtient alors une composition de pattern.

Le pattern MVC (Modèle Vue Contrôleur) n'est pas un GoF pattern mais il n'en est pas moins très très utilisé. Il constitue quasiment à lui seul, dans certains cas simples, le squelette de l'architecture d'une application. En particulier, il est utilisé dans les architectures n-tiers.

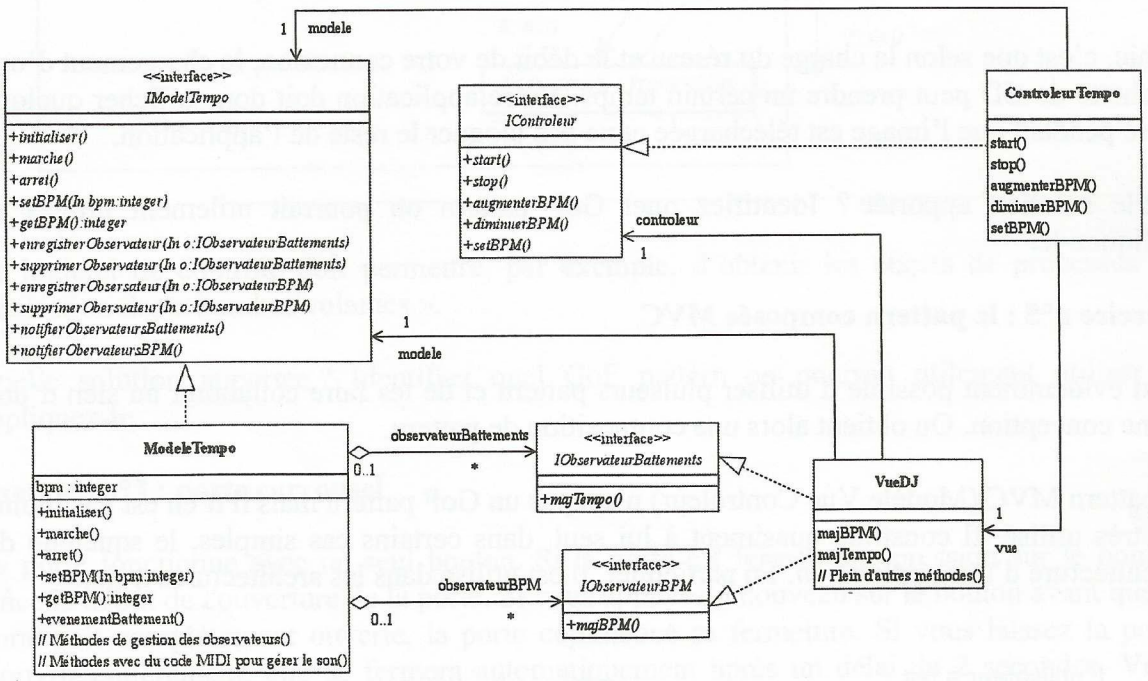


Il est constitué de plusieurs patterns qui collaborent :

- Le contrôleur : il gère les entrées des utilisateurs et détermine ce qu'elles signifient pour le modèle (c'est-à-dire si le modèle doit changer d'état). Il peut y avoir plusieurs contrôleur pour une même vue. Pour cela, le contrôleur utilise le pattern **Stratégie** car

il peut implémenter plusieurs comportements différents. Il fournit à la vue la stratégie qui correspond à l'état actuel du modèle.

- La vue : elle fournit une représentation du modèle et un moyen de demander un changement d'état au modèle. Cette représentation peut être partielle tout comme il peut y avoir plusieurs vuesz différentes pour un même modèle. En général la vue reçoit directement du modèle l'état et les données qu'elle affiche. La vue utilise le pattern **Composite** puisque qu'elle est composée d'élément graphique composite (comme une fenêtre) et d'élément graphique simple (comme un bouton).
- Le modèle : il contient les données et la logique applicative. Il n'a pas conscience de la vue ni du contrôleur même s'il fournit une interface pour accéder à son état et le manipuler et s'il peut informer les observateurs des changement d'état. Pour cela, il implémente le pattern **Observateur** et joue le rôle de sujet tandis que les vues jouent le rôle d'observateurs.



Il s'agit d'une petite application de gestion du tempo d'une piste d'un séquenceur MIDI. Au niveau des commandes, la vue permet de modifier de facon incrémentale le tempo soit en l'augmentant soit en diminuant. Elle permet également de saisir directement un nouveau tempo. Son menu permet d'arrêter ou de mettre en marche. Au niveau de l'affichage, elle affiche la valeur actuel du tempo et en plus affiche une barre pulsante qui "pulse" au rythme du tempo.

A partir de la description du pattern composé MVC et du diagramme ci-dessus, complétez le code suivant en remplaçant les ... par l'instruction correcte :

```
public class ModeleTempo implements IModeleTempo, MetaEventListener {
        Sequencer sequenceur;
        ArrayList observateursBattements = new ArrayList();
        ArrayList observateursBPM = new ArrayList();
        int bpm = 90;
        // autres variables d'instances

        public void initialiser() {
```

```
            setMidi();
            construirePisteEtDemarrer();
        }

        public void marche() {
            sequenceur.start();
            setBPM(90);
        }

        public void arret() {
            setBPM(0);
            sequenceur.stop();
        }

        public void setBPM(int bpm) {
            this.bpm = bpm;
            sequenceur.setTempoInBPM(bpm);
            ...
        }

        public int getBPM() {
            return bpm;
        }

        void evenementBattement() { // Méthode appelée à chaque nouveau battement
            ...
        }

        // Code d'enregistrement et de notification des observateurs

        // Beaucoup de code MIDI pour gérer le son
}

public class VueDJ implements ActionListener, IObservateurBattements, IObersvateurBPM {
        ...
        ...
        JFrame cadreVue;
        JPanel panneauVue;
        BarrePulsante barre;
        JLabel affichageBPM;
        JTextField champTxtBPM;
        JButton augmenterBPM;
        JButton diminuerBPM;
        JMenuBar barreMenu;
        JMenu menu;
        JMenuItem choixStart;
        JMenuItem choixStop;

        public VueDJ(IControleur controleur, IModeleTempo modele) {
            ...
            ...
            ...
            ...
        }

        public void creerVue() {
            // Création de tous les composants graphiques
        }
```

```
        public void majBPM() {
                int bpm = ...
                if (bpm = 0)
                        affichageBPM.setText("hors ligne");
                else
                        affichageBPM.setText("Nombre de BPM : " + bpm);
        }

        public void majTempo() {
                barre.setValue(100); // La barre va redescendre tranquillement
        }

        public void creerCommandes() {
                // Création de toutes les commandes
        }

        public void activerChoixStop() {
                choixStart.setEnabled(true) ;
        }

        public void desactiverChoixStop() {
                choixStop.setEnabled(false) ;
        }

        public void activerChoixStart() {
                choixStop.setEnabled(true) ;
        }

        public void desactiverChoixStart() {
                choixStart.setEnabled(false) ;
        }

        public void actionPerformed(ActionEvent evenement) {
                if (evenement.getSource() = defenirBPM)
                        ...
                if (evenement.getSource() = augmenterBPM)
                        ...
                if (evenement.getSource() = diminuerBPM)
                        ...
        }

        // Autres méthodes
}

public class ControleurTempo implements IControleur {
        ...
        VueDJ vue ;

        public ControleurTempo(IModeleTempo modele) {
                ...
                ...
                vue.creerVue();
                vue.creerCommandes();
                vue.desactiverChoixStop();
                vue.activerChoixStart();
                modele.initialiser();
        }

        public void start() {
```

```
            ...;
            vue.desactiveChoixStart() ;
            vue.activeChoixStop() ;
        }

        public void stop() {
            modele.arret() ;
            ...
            ...
        }

        public void augmenterBPM() {
            ...
        }

        public void diminuerBPM() {
            ...
        }

        public void setBPM(int bpm) {
            ...
        }
}
```