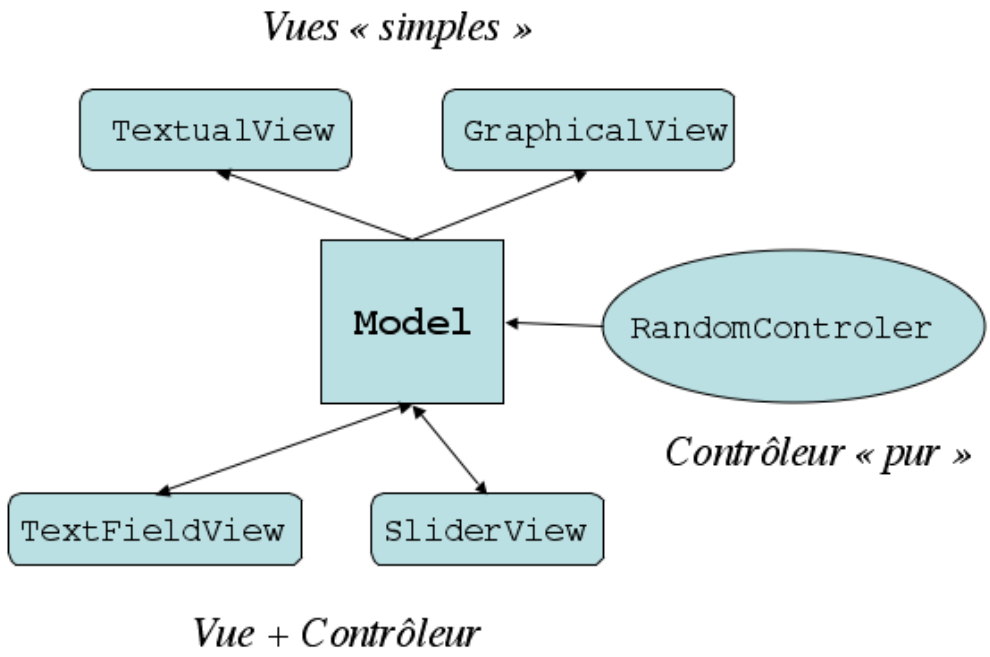
TP 7 : Le design pattern `Model View Controler`

Nous allons dans ce TP explorer le principe du *patron de conception* `MVC` : `Model View Controler`. Pour cela, nous allons prendre pour prétexte une petite application dans laquelle on manipule trois valeurs R, V et B, dont la somme doit toujours faire 100. Lorsque l'utilisateur modifie l'une des valeurs, les deux autres doivent être mise à jour en gardant leur proportion relative.

L'architecture globale de l'application sera la suivante :



Première partie : le Modèle !

Un petit exemple valant mieux qu'un long discours, imaginons qu'au départ l'état du modèle soit :
`R = 33, V = 33, B = 34`
Si l'utilisateur cahnge la valeur de R a 50, le modèle devra se retrouver dans l'état suivant :
`R = 50, V = 25, B = 25`

Concevez une classe `Model` qui étend la classe `java.util.Observable` (elle sera "observable" pour les différentes vues que nous lui associerons), et fournit les accesseurs nécessaires ainsi que les fonctions de modification de l'état du modèle (en n'oubliant de notifier les éventuels abonnés lors d'une évolution du modèle !).

Cette classe devra implémenter l'interface suivante (c'est pour pouvoir utiliser l'interface graphique un peu plus tard) :

```
package misc.mvc;
import java.util.Observer;
public interface RVBModel {
  public int getR();
  public int getV();
  public int getB();
  public int get(int idx);
  public void setR(int r);
  public void setV(int v);
  public void setB(int b);
  public void set(int idx, int value);

  public void addObserver(Observer o);
}
```

Remarque: Utilisez le même paquetage que ci-dessus (ie. `misc.mvc`), sinon cela ne fonctionnera pas. c'est-à-dire

Testez votre modèle en mode texte, par exemple de cette manière :

```
public static void main(String[] args) {
```

```
  RVBModel m = new Model(33,33,34);
  System.out.println(m);
  m.setR(50);
  System.out.println(m);
}
```

Une autre manière plus "intelligente" (tout du moins plus complète) consiste à écrire une classe vue textuelle, se contentant d'afficher sur la sortie standard l'état courant du modèle, et d'écrire une autre classe contrôleur qui va modifier périodiquement le modèle.

Ecrivez une classe `TextualView` qui implémente l'interface `java.util.Observer` et qui une fois abonnée au modèle se contente d'afficher sur la sortie standard l'état du modèle (ie. dans la méthode `update`).

Ecrivez ensuite une classe `RandomControler`, qui ne s'intéresse aucunement au modèle, si ce n'est pour modifier son état aléatoirement (cette classe n'a donc pas besoin (et ne doit pas !) d'implémenter l'interface `Observer`). Cette classe définira la méthode suivante :

```
public void go(int turn) {
  for (int i=0; i<turn; i++) {
    try {
      Thread.sleep(1500);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    int idx = ((int) (Math.random()*100)) % 3;
    model.set(idx, (int) (Math.random()*100));
  }
}
```

Remarque: Vous voyez au passage qu'une méthode `set(int index, int value)` peut être pratique en plus des `setR, setV, setB` ....
Remarque: Le bloc `try/catch` sert juste à introduire une temporisation (donnée en millisecondes).

La classe principale lançant l'exécution devient donc :

```
class Main {
  public static void main(String[] args) {
    RVBModel model = new Model(33,33,34);
    TextualView vue = new TextualView(model);
    RandomControler controler = new RandomControler(model);
    controler.go(15);
  }
}
```

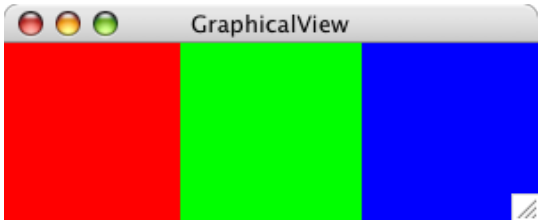
C'est la première fois que nous identifions clairement les trois entités en jeu : le modèle (`Model`), une vue (`TextualView`) et un contrôleur "pur" (`RandomControler`).

Deuxième partie : une vue "graphique"

La deuxième partie ne sera pas très complexe : vous trouverez ici une vue représentant graphiquement les trois valeurs R, V et B (Rouge, Vert et Bleu). Il vous suffit de l'associer à votre modèle pour vérifier que vous avez bien implémenté le modèle (`GraphicalView` est aussi dans le paquetage `misc.mvc`) :

```
public static void main(String[] args) throws InterruptedException {
  RVBModel model = new Model(33,33,34);
  JPanel p = new GraphicalView(model, Color.RED, Color.GREEN,
Color.BLUE);
  JFrame f = new JFrame("GraphicalView");
  f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  f.getContentPane().add(p);
  f.pack();
  f.setVisible(true);
}
```

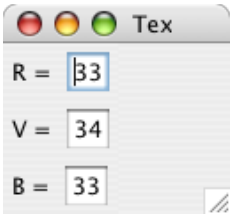
Si tout fonctionne bien, lorsque vous lancerez le programme ci-dessus vous devriez voir apparaître une fenêtre contenant trois rectangles de couleurs changeant aléatoirement de taille.



Troisième partie : une vue "textfield" et un contrôleur associé

Dans cette partie, vous allez créer la première interface de saisie graphique permettant d'interagir directement avec le modèle.

Nous utiliserons pour cette première version de simples zones de texte (ie `JTextField`).



Quatrième partie : une vue "échelle" et un controleur associé

Dans cette dernière partie, vous allez ajouter une seconde vue associant un contrôleur, mais basée cette fois sur la classe `JSlider`. Les *sliders* servent à la fois à refléter l'état du modèle, mais aussi à le modifier.

A la fin du TP, il faut que vous puissiez lancer une application créant les trois vues possibles et les associant au même modèle. Vous constaterez que les modifications effectuées via l'un des contrôleurs se répercuteront "automatiquement" sur les autres vues !

Joli, non ?