Análisis del Patrón MVC

Desarrollado por David G. Kotlirevsky

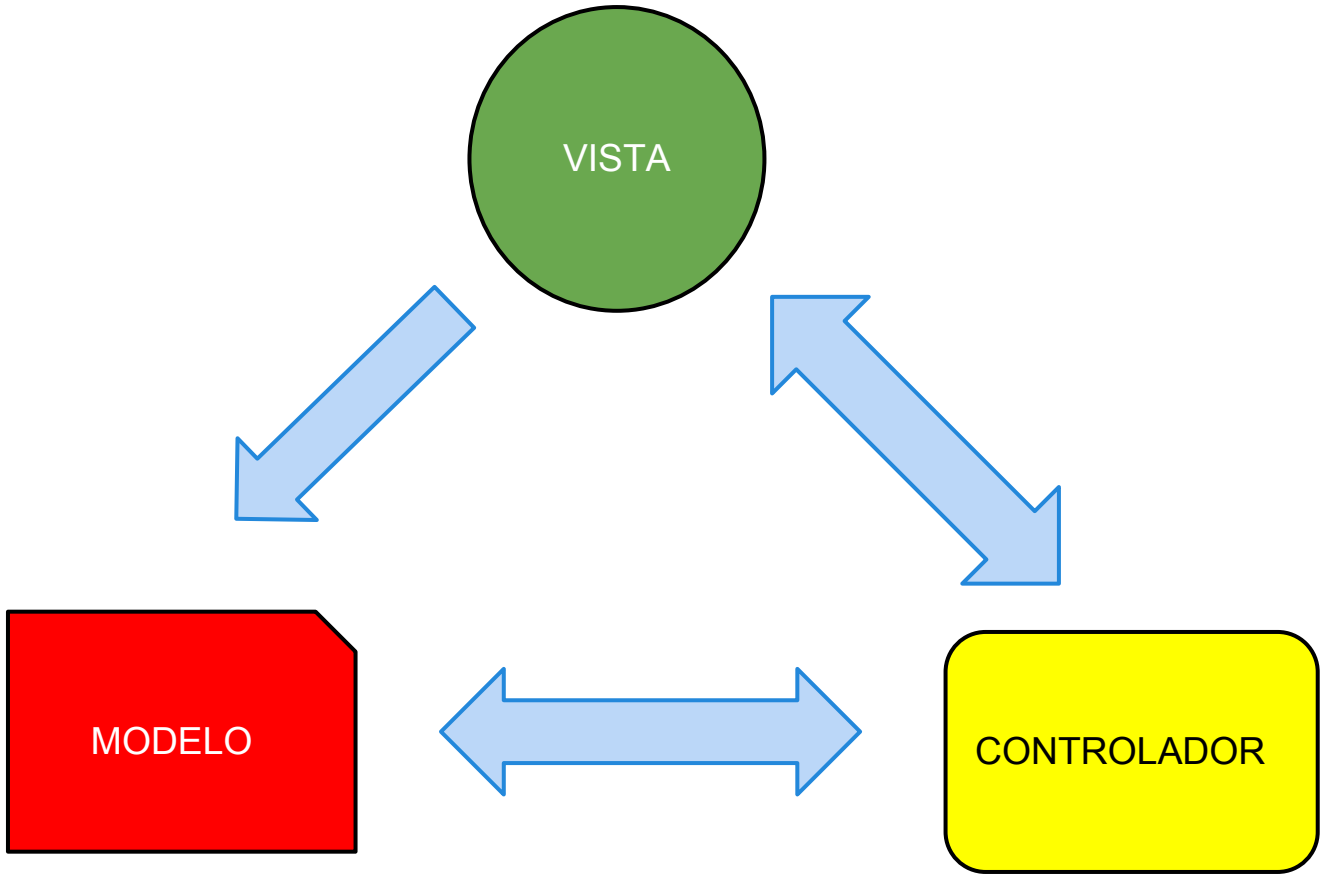
Qué es el Patron MVC

* Es una forma de resolver la escalabilidad y el desacoplamiento entre las vistas de una aplicación y el Modelo que gestiona los datos

* Su nombre es un acrónimo de:

MODELO
VISTA
CONTROLADOR

Qué es el Patrón MVC



Qué es el Patrón MVC

Conceptualmente:

* La vista interactúa libremente con el controlador y viceversa.

* El Modelo interactúa libremente con el Controlador y viceversa.

* La Vista se actualiza en funcion del los cambios de estado del Modelo. El Modelo NO conoce quien es la Vista.

Qué es el Patrón MVC

Es la Sumatoria aplicada de 3 patrones de Diseño :

MVC

OBSERVER

+

MEDIATOR

+

SINGLETON

Ejemplo

*Veremos en un ejemplo como funciona destacando cada parte y su componente

Vista

(Observadores) : -> Los Autos

Controlador

(Aplica Mediator y Singleton) : Maneja los avisos

Modelo

(El que es observado) -> un Semáforo

Download ?

El código del ejemplo es descargable con este comando (tener instalado git ;))

git clone git@github.com:davidgk/mvcPatternAnalisis_eclipseProyect.git

Vista (Son los Observadores)

```
public class Moto extends AbstractVehiculo {

    public Moto(String marca) {
        super(marca);
    }

}
```

```
public class Camion extends AbstractVehiculo {

    public Camion(String marca) {
        super(marca);
    }

}
```

```
public abstract class AbstractVehiculo {

    private String marca;

    public AbstractVehiculo(String marca) {
        this.marca = marca;
    }

    public void updateMiComportamiento(SEMAFORO_ESTADOS estado) {
        switch (estado) {
            case ROJO:
                System.out.println("******************");
                System.out.println(this.marca + " : Me Detengo");
                System.out.println("******************");
                break;
            case AMARILLO:
                System.out.println("******************");
                System.out.println(this.marca + " : Miramos con precaucion");
                System.out.println("******************");
                break;
            case VERDE:
                System.out.println("******************");
                System.out.println(this.marca + " : Avanzamos");
                System.out.println("******************");
                break;
        }
    }
}
```

```
public class Auto extends AbstractVehiculo {

    public Auto(String marca) {
        super(marca);
    }
}
```

Controlador

Aplica Mediator y Singleton.

En este caso para evitar que el modelo conozca la vista, El controlador posee la lista de vistas a actualizar así como el método responsable de tal tarea.

```
public class ControllerTrafic {

    private ArrayList<AbstractVehiculo> listaObservadores;

    // Contrstrucor con inicializador de lista que contendra los observadores
    private ControllerTrafic(){
        this.listaObservadores = new ArrayList<AbstractVehiculo>();
    }

    // getInstance() singleton
    public static ControllerTrafic getInstance() {
        return ControllerSingleton.instance;
    }

    // Singleton thread safe creator
    private static class ControllerSingleton{
        private static ControllerTrafic instance = new ControllerTrafic();
    }

    //getter
    public ArrayList<AbstractVehiculo> getListaObservadores() {
        return listaObservadores;
    }

    public void actualizarMisObservadores(Semaforo semaforo) {
        for (AbstractVehiculo vehiculo : listaObservadores) {
            vehiculo.updateMiComportamiento(semaforo.getEstado());
        }
    }

}
```

Modelo

Al modelo se le asigna el controlador por constructor.

Por medio del mismo avisará sus cambios de estado.

```
public class Semaforo {

    private ControllerTrafic controller;
    private SEMAFORO_ESTADOS estado;

    //estados validos validos para el ejemplo
    public enum SEMAFORO_ESTADOS{
        ROJO,AMARILLO,VERDE
    }

    // Constructor
    public Semaforo(ControllerTrafic controller) {
        this.controller = controller;
    }

    public SEMAFORO_ESTADOS getEstado() {
        return estado;
    }

    public void setEstado(SEMAFORO_ESTADOS newEstado) {
        this.estado = newEstado;
        controller.actualizarMisObservadores(this);
    }
}
```

Una pequeña prueba

```java
@Test
public void mvcPatternTest(){
    /**Creamos el controlador
     que regulara el flujo y permitira que:
     1.- La vista se actualize a los cambios del modelo
     2.- el modelo repercuta los cambios qe le sucedan en la vista
     3.- El modelo no queda acoplado a la vista
     */
    ControllerTrafic controller =ControllerTrafic.getInstance();

    /**Creamos los Observadores , Que son las vistas*/
    AbstractVehiculo auto = new Auto("marcaAuto");
    AbstractVehiculo camion = new Camion("marcaCamion");
    AbstractVehiculo moto = new Moto("marcaMoto");

    /** El controller adquiere registro de las vista que debera actualizar*/
    controller.getListaObservadores().add(auto);
    controller.getListaObservadores().add(camion);
    controller.getListaObservadores().add(moto);

    /**Creamos el Observable , es decir el Modelo del Patron,
    al cambiar este , repercutira los cambios en la vista por medio del aviso
    A traves del controlador
    */
    Semaforo sem = new Semaforo(controller);

    /**
     * Cambiamos Varias veces el estado del semaforo
     * y veremos como repercute esto en las vistas
     */
    System.out.println("**********************");
    System.out.println("Semaforo en ROJO");
    sem.setEstado(SEMAFORO_ESTADOS.ROJO);
    System.out.println("**********************");
    System.out.println("Semaforo en AMARILLO");
    sem.setEstado(SEMAFORO_ESTADOS.AMARILLO);
    System.out.println("**********************");
    System.out.println("Semaforo en VERDE");
    sem.setEstado(SEMAFORO_ESTADOS.VERDE);
    System.out.println("**********************");
}
```

El resultado

```
**********
Semaforo en ROJO
**********
marcaAuto : Me Detengo
**********
**********
marcaCamion : Me Detengo
**********
**********
marcaMoto : Me Detengo
**********
**********
Semaforo en AMARILLO
**********
marcaAuto : Miramos con precaucion
**********
**********
marcaCamion : Miramos con precaucion
**********
**********
marcaMoto : Miramos con precaucion
**********
**********
Semaforo en VERDE
**********
marcaAuto : Avanzamos
**********
**********
marcaCamion : Avanzamos
**********
**********
marcaMoto : Avanzamos
**********
**********
```

Gracias!

Consultas, dudas y/o sugerencias a:

info.clases.programacion@gmail.com

www.clasesprogramacion.com