Gestionar las variables globales

Las variables globales son inevitables, por ejemplo en un juego la resolución de pantalla se utilizará en muchas clases del código.

Gestionar las variables globales

Lo habitual … declararlas en la primera clase en la que es necesaria. Muy mala aproximación, ya que quedan desperdigadas por nuestras clases y son difíciles de encontrar.

Dificulta la reutilización de las clases, ya que crean dependencias entre ellas.

Gestionar las variables globales

Una solución mejor es crear una clase cuya única finalidad sea contener las variables globales de la aplicación. Las clases dependen de una sóla clase que contiene las variables y no creamos dependencias entre ellas.

Para simplificar será la solución que tomaremos.

```
public  class GlobalValues {

    public static final int CAMERA_WIDTH = 720;
    public static final int CAMERA_HEIGHT = 480;
    public static final int CELL_WIDTH = 32;
    public static final int XMAX = 14;
    public static final int YMAX = 14;
```

Gestionar las variables globales

- Con la solución anterior es necesario compilar al cambiar el valor de una constante.
- Una solución mejor es crear un fichero .properties (key,value) y un Singleton, también llamado instancia única (Design Patterns : Elements of Reusable Object-Oriented Software, Ed. Addison Wesley) para acceder a los valores.

Gestionar las variables globales

- El problema de la solución anterior es que en entornos distribuidos el concepto de instancia única es confuso, con decenas o cientos de JVMs funcionando.
- La solución en aplicaciones clusterizables es tener un servidor JNDI (Java Naming and Directory Interface) que gestiona las variables globales para todas las instancias de la aplicación.

Problemas

- Tenemos las responsabilidades divididas en nuestro modelo. Queremos dar una interfaz sencilla que presente los casos de uso dando pocos detalles del modelo.
- Queremos crear una interfaz que oculte los detalles de implementación, por ejemplo si está implementada la persistencia con JDBC/EJB/Hibernate.

Problemas

- Es posible que las responsabiliades de un caso de uso se encuentren distribuidas en varias aplicaciones ejecutándose en máquinas diferentes.

Facade

- Vamos a utilizar el patrón Facade, una simplificación de los patrones Session Facade/Bussiness delegate.
- Crearemos una interfaz (Delegate) en la que cada método representa un caso de uso. En aplicaciones con múltiples actores es normal crear un delegate por cada actor.

Facade

```
public interface CalculadoraFacadeDelegate {

    Integer sumar(Integer x, Integer y);
    Integer restar(Integer x, Integer y);
    Integer factorial(Integer x);

}
```

-

Facade

- Ahora podemos implementar una fachada que cubra los casos de uso del delegate. Podemos tener múltiples fachadas utilizando distintas tecnologías y este hecho será invisible para quien use la interfaz (delegate).
- Surge un nuevo problema, y es que una clase que implemente varios casos de uso tendrá una longitud enorme.

Acciones del modelo

- La solución es crear una acción por cada caso de uso, cada método de la fachada invocará su acción correspondiente.

Acciones del modelo

```
public interface Action {

    public Object execute();
}
```

Acciones del modelo

- Los argumentos del método de la fachada serán pasados en el constructor de la acción. El método de la acción devolverá un Object, la fachada hará un downcast a la clase correspondiente.

Acciones del modelo

```
public class SumarAction implements Action{

    Integer x;
    Integer y;

    public SumarAction(Integer x, Integer y){
        this.x=x;
        this.y=y;



    public Object execute() {
        return new Integer(x.intValue()+y.intValue());
    }
```

Invocando acciones

```
public Integer sumar(Integer x, Integer y) {
        Action accion = new SumarAction(x,y);
        return (Integer)accion.execute();
    }
```

Conclusiones

- Las responsibilidades estaban divididas estaban divididas en múltiples clases/aplicaciones/subsistemas.
- Ahora tenemos una interfaz fácil de utilizar que describe los casos de uso.
- Podemos tener diferentes implementaciones, que serán invisibles para quien use la interfaz.
- Gracias a las acciones del modelo las fachadas no se convertirán en clases de gran tamaño, además las acciones se pueden ir implementando en paralelo en un equipo de desarrollo.