Aventura textual

Nelson Páramo Valdivielso

Contenido

[Enunciado del proyecto 3](#_Toc504579476)

[Requisitos del motor 3](#_Toc504579477)

[PROYECTO 4](#_Toc504579478)

[Models 4](#_Toc504579479)

[Diagrama de clases 4](#_Toc504579480)

[Aplicattion 5](#_Toc504579481)

[Diagrama de clases 5](#_Toc504579482)

[Implementation.Prop 5](#_Toc504579483)

[Diagrama de clases 5](#_Toc504579484)

[Implementation.Interactions 6](#_Toc504579485)

[Diagrama de clases 6](#_Toc504579486)

[Implementation.Interactions 6](#_Toc504579487)

[Diagrama de clases 6](#_Toc504579488)

[PATRONES UTILIZADOS 7](#_Toc504579489)

[Singleton 7](#_Toc504579490)

[Command 7](#_Toc504579491)

[Mediator 9](#_Toc504579492)

[Patrón Factory 9](#_Toc504579493)

[PROBLEMAS Y POSIBLES IMPLEMENTACIONES DE CARA AL FUTURO 10](#_Toc504579494)

[Introducir nuevas historias. 10](#_Toc504579495)

[Problemas 10](#_Toc504579496)

# Enunciado del proyecto

El proyecto consistirá en **completar el motor de ficción interactiva** comenzado en los ejercicios en clase ([enunciado](https://ubuvirtual.ubu.es/mod/page/view.php?id=2906431) y [una posible solución](https://ubuvirtual.ubu.es/mod/page/view.php?id=2908679)), a realizar **individualmente**. El lenguaje de implementación será **Java**.

Se realizará **una única entrega** a través de un **repositorio online** (GitHub, GitLab...), el día **24 de enero** como muy tarde, con un peso del **40%** sobre la nota final de la asignatura. Si algún commit está etiquetado como versión final, se tomará ese como entrega. Si no hay ninguno etiquetado, se usará el último commit antes de su vencimiento (aunque se trate de commits intermedios de desarrollo). **La memoria debería estar también disponible en el repositorio**.

# Requisitos del motor

1. Como **evaluador** quiero **acceso al código y su historial** para **ver la implementación**
2. Como **evaluador** quiero **una memoria** para **evaluar el uso de buenas prácticas y patrones de diseño**
3. Como **evaluador** quiero **instrucciones** para **poder construir y lanzar la aplicación**
4. Como **cliente** quiero **que la historia pueda tener varias escenas**
5. Como **cliente** quiero **poder interactuar con elementos de la escena**
6. Como **cliente** quiero **poder moverme entre escenas**
7. Como **cliente** quiero **que las interacciones con un elemento puedan afectar a otros elementos**
8. Como **cliente** quiero **poder crear mis propias historias, definiéndolas en algún fichero**

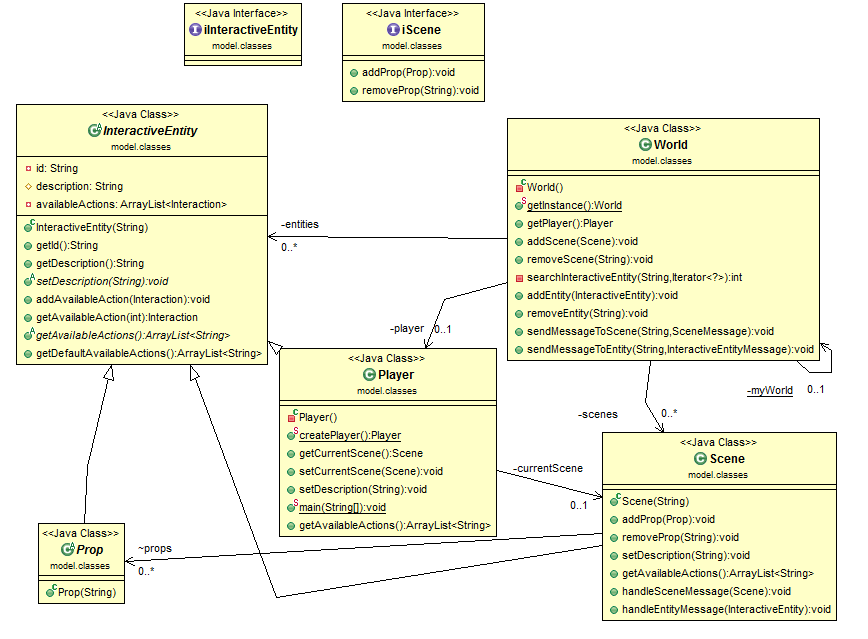
# PROYECTO

El proyecto se ha divido en diversos paquetes para facilitar su implementación

## Models

El paquete models hace referencia al modelo del sistema, donde se han implementado las clases: InteractiveEntity, Player, Prop, Scene y World.

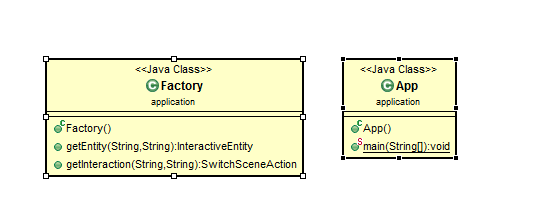
### Diagrama de clases



## Aplicattion

La cual contiene el método main de la aplicación y la clase Factory que es la encargada de la creación de objetos.

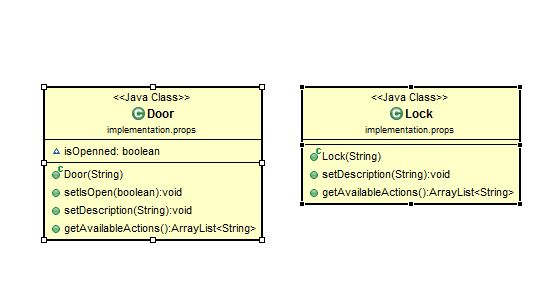
### Diagrama de clases



## Implementation.Prop

Es el encargado de crear el atrezo de las escenas.

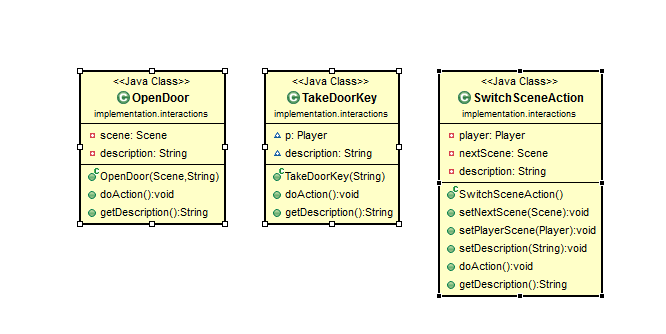
### Diagrama de clases



## Implementation.Interactions

Se encarga de crear las interaciones con los objetos definidos en cada escena.

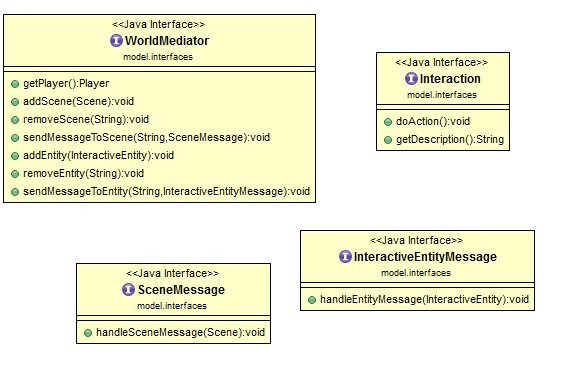
### Diagrama de clases



## Implementation.Interactions

Se encarga de crear las Interfaces definidas en el sistema. La más importante es el WorldMediator donde estarán las escenas con su correspondiente atrezo.

### Diagrama de clases



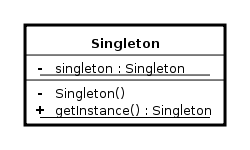
# PATRONES UTILIZADOS

## Singleton

El patrón **singleton** o instancia única es un permite la restricción de la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto

Trate de garantizar que una clase solo tenga una instancia y proporcionar un único punto de acceso global a ella.

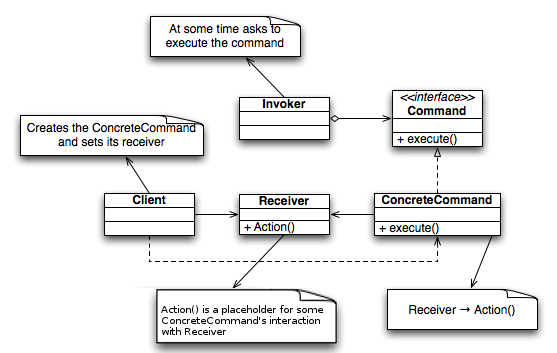
Es implementado creando en nuestra clase un método que crea una instancia del objeto, si todavía no existe alguna, para restringir la instanciación más veces se regula el alcance del constructor con un modificador de acceso protegido o privado.



**En mi caso, a la hora de realizar la práctica,** el patrón Singleton a sido utilizado a la hora de instanciar la clase World, debido a que va a ser un objeto único y que solo va a ser necesario crearlo una única vez.

## Command

El patrón **command** permite solicitar que se realice una determinada acción a un objeto sin conocer realmente el contenido de la operación, ni el verdadero receptor de la misma. De esta forma se encapsula la petición como un objeto, lo que nos ayuda a la parametrización del os métodos, permitiendo gestionar colas o registros de mensajes y deshacer operaciones. El fin de este patrón es soportar la restauración de un estado a partir de un momento concreto, y nos ofrece una interfaz común que permite invocar acciones de una manera uniforme y extender el sistema con nuevas operaciones de una manera más fácil.



En mi caso, a la hora de realizar la práctica, el patrón Command ha sido utilizado para que la implementación de las acciones no dependa de una clase en concreto, es decir, independientemente de la implementación de una interaccion la forma de llamarla es mediante el método doAction(), la cual no tiene parámetros de entrada, ya que los datos que va a utilizar el método están contenidos en la clase que implementa la acción y se establecen con una función Set.

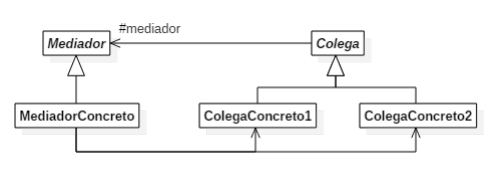
Considero que lo más importante es el diseño del modelo de datos ya que cualquier entidad interactiva puede estar contenida en otra de una forma recursiva, es decir, que puedes tener n entidades con tenidas unas dentro de otras e implementar las acciones propias de cada entidad de forma independiente sin que la implementación afecte al modelo de datos

En mi caso, existe una jerarquía entre la escena y el atrezo, y ambas son entidades interactivas por lo que una está contenida dentro de la otra, el modelo que se ha diseñado permite que una entidad interactiva sea parte de otra, que a su vez contiene a otra y así sucesivamente, lo que hace dicha entidad interactiva no estar sujeta a una implementación concreta sino a un modelo de implementación, es decir, una interfaz, así que cualquier acción que implemente dicha interfaz podrá ser realizada por la entidad interactiva independientemente de su implementación. Por lo que una hipotética acción AbrirPuerta se ejecutara de la misma forma que una acción cogerPlato, internamente harán cosas diferentes pero a la vista de una entidad de mayor nivel será lo mismo.

## Mediator

Define un objeto que encapsula cómo interactúan varios objetos, reduciendo su acoplamiento (no tienen que referenciarse entre ellos explícitamente) y permitiendo variar su interacción dinámicamente.

Se debe usar el patrón mediador cuando un conjunto de objetos deban comunicarse de una forma compleja pero bien definida, cuando la reutilización de un objeto sea complicada debido a que hace referencia y se comunica con muchos otros objetos o un comportamiento distribuido entre varias clases deba ser personalizable sin demasiadas subclases.

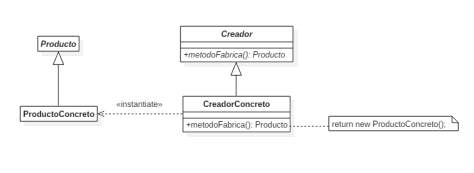


**En mi caso, en la práctica**, mediante el patrón Mediator se quería conseguir que el objeto worldMediator iba a estar con tenida en cada una de las entidades interactivas de tal forma de que cuando se ejecutase un acción sería el mediador el que realmente ejecutaría la acción a vista de todas las entidades interactivas, es decir, que si se ejecuta la acción abrir puerta es el mediador el que ejecuta dicha acción ya que contiene la referencia de todas las entidades interactivas

## Patrón Factory

Definir una interfaz para crear un objeto, pero que sean las subclases quienes decidan la clase a instanciar. Se delega la instanciación a las subclases.

El patrón Factory se utiliza cuando una clase no puede anticipar la clase de objetos que se deben crear o quiere que sean las subclases las encargadas de decidir el objeto a crear, por lo que se delega la responsabilidad a una o varias clases de apoyo.



**En mi caso, en la práctica**, se utiliza para instanciar la clase Player.

# PROBLEMAS Y POSIBLES IMPLEMENTACIONES DE CARA AL FUTURO

## Introducir nuevas historias.

Para crear nuevas historias lo único que habría que hacer es implementar atrezo y posibles acciones sobre ese atrezo, debido a que las escenas se crean de forma genérica, ya que sólo tienen una descripción y un identificador.

Por lo que se debería atribuir atrezo a una escena que tendrá ligadas acciones concretas, es decir, para querer introducir por ejemplo una espada, se debería implementar acciones como desenvainar o atacar.

Estas acciones deberían estar implementadas en el WorldMediator, y se añadirían a la avaliableActions del prop.

Si es una llave pues la acción abrir puerta se podrá realizar cuando el jugador posea la llave y la acción cogerLlave será una acción de la escena o de otro prop; por ejemplo, si la llave está en una caja la escena tendrá la acción abrirCaja y la caja tendrá la acción cogerLlave

## Problemas

* Uno de los problemas más grandes a los que me he enfrentado es a las dificultades de integrar patrones estructurales con patrones de comportamiento.
* Otro de los problemas que no he llegado a solucionar es la introducción de las historias a través de un fichero XML. Ya que exigían realizar un parseo correcto, y no estaba familiarizado con ninguna librería que me facilitase el trabajo, por lo que busque alternativas introduciendo los props en escenas creadas de forma genérica.

# Conclusiones

Al haber realizado la práctica he llegado a las siguientes conclusiones:

Todos los patrones son ideas que no están exentas de sus ventajas e inconvenientes, lo que es importante conocer bien los patrones de diseño para adecuarlo a nuestras necesidades.

El patrón de diseño elegido debe ser resistente al cambio, por lo que nos debe facilitar lo máximo los futuros posibles cambios, por lo que se debe abstraer el problema concreto en otro más genérico y resolver este último. Que es lo que los patrones consiguen precisamente.

El patrón escogido debe estar abierto a la reutilización, por lo que cuando se hace un diseño debemos pensar que nos debe ser útil para futuras ideas.

En general sirven para minimizar el riesgo de realizar un mal diseño.

El conocimiento de los patrones de diseño va a ser imprescindible a la hora de encontrar un futuro trabajo, debido a lo anteriormente mencionado, por lo que si hubiera sabido de ellos en anteriores asignaturas, los habría aplicado sin lugar a dudas.

# Bibliografía

Patrones de diseño

* <https://es.wikipedia.org/wiki/Mediator_(patr%C3%B3n_de_dise%C3%B1o)>
* <https://es.wikipedia.org/wiki/Command_(patr%C3%B3n_de_dise%C3%B1o)>
* <https://es.wikipedia.org/wiki/Factory_Method_(patr%C3%B3n_de_dise%C3%B1o)>
* <https://es.wikipedia.org/wiki/Singleton>
* Apuntes y pdfs de la asignatura.