Este es un esquema de las clases, los atributos y los métodos. Es una idea, no lo copiéis a saco, puede que los tipos de las funciones no sean esos, o cualquier otra cosa.

Si tenéis dudas revisar la especificación y la carpeta \_PATRONES.

Se irá modificando según surjan problemas o cosas nuevas. Si estáis con cualquier clase, revisad si se ha modificado algo.

**MODELO**

Clase Coordenada {

// Atributos

int x;

int y;

}

Clase Vagon {

// No creo que sean necesarios atributos. Como mucho el char que lo representaría por pantalla

}

Clase Tren {

// Atributos

vector de vagones; // Límite de 5 vagones. No vayamos a sobrarnos

boolean remolque;

boolean enMarcha;

Coordenada coordenadaActual;

Coordenada origen;

// Y de nuevo, el char que lo representa si lo consideráis necesario

//Constructor

//Por defecto parado.

public Tren(Coordenada origen, int numeroVagones) {

// Bucle simple que vaya añadiendo vagones

}

public Tren(Coordenada origen, boolean remolque) {

// Constructora del remolque.

// En marcha por defecto

}

// Métodos

public void anadeVagon() { // U.C.4.

// si excede el limite, excepción

}

public void eliminaVagon() { // U.C.5.

// lo mismo

}

public void setMarcha(boolean enMarcha) { U.C.12 y U.C.13.

this.enMarcha = enMarcha;

}

}

public enum EstadoSenal { VERDE, ROJO }

Clase Señal {

// Atributos

Coordenada posición;

EstadoSenal estado;

\*\*Char para que lo representa en pantalla

// Constructor

public Senal (Coordenada posicion){

this.posicion = posición;

this.estado = EstadoSenal.ROJO; // por defecto en Rojo

}

//Métodos

public void setEstado(){

//Si esta rojo se pone verde y viceversa

}

public EstadoSenal getEstado(){

}

public Coordenada getCoordenadaSenal (){

}

}

Clase Mundo {

esta clase sólo recibe los commands de controlador, no toma decisiones por sí misma

// Atributos

vector de trenes

vector de señales

boolean ejecucionParada

boolean[] activos // Para cuando pare la ejecución, saber cuáles estaban en marcha

Observadores de las entidades del controlador

// Métodos

public void crearTren(Coordenada origen, int numeroVagones) { U.C.1

}

public void modificarTren(Coordenada origen, int nuevoNumeroDeVagones) { U.C.2.

según el número del parámetro, inserta o elimina vagones del tren de las coordenadas

}

public void eliminarTren(Coordenada origen) { U.C.3.

//Despues de eliminar el tren, actualizar()

}

private int buscaTren(Coordenada origen) {

busca el tren en el vector

}

private void crearRemolque(Coordenada origen){

//crearTren(origen, true)

}

public void tratarColision(Coordenada colision, Tren implicado1, Tren implicado2){

setSenal() de todas los semaforos de la via a rojo

eliminarTren(implicado1)

eliminarTren(implicado2)

crearRemolque (colision)

//la especificación es redundante porque dice que despues de crearlo lo //arranquemos, pero no hace falta porque está por defecto

setSenal() de todas los semaforos de la via a verde

actualizar()

}

public void actualizar() {

// En cada "turno" actualiza la coordenadaActual de los trenes que estén en marcha

}

public void pararEjecucion() {

// Guarda en el array de activos los que estén en marcha y justo después, para toda actividad

}

public void reanudarEjecucion() {

// Se vale del array de activos para reanudar la marcha de ciertos trenes.

}

public void crearSenal (Coordenada posicion){ U.C. 6

senal(posicion);

}

public void modificarSenal (Coordenada posicion){ U.C. 7

setEstado();

}

public void eliminarSenal (Coordenada posicion){ U.C. 8

}

Prívate void buscarSenal(Coordenada posicion){

//busca la señal en el vector

}

}

**CONTROLADOR**

Class Controlador {

// Atributos

Los observers de las entidades (patrones)

Los listeners de la interfaz

// Métodos

Los callers a los listeners que posee

}

Clase Ficheros {

trabajo con ficheros. U.C. 15

}