Introducción a la práctica.

Esta práctica está fundamentada en tres ideas centrales.

La primera es implementar todas las especificaciones mencionadas en el enunciado de la práctica obligatoria del departamento o mejorarlas. Por ello se ha realizado el juego space invaders con, al menos, los requisitos mínimos, comentado el código, y generados el Javadoc y los distintos diagramas de clases y de objetos.

La segunda es hacerlo de forma que se enlace con los conceptos principales del libro, como puedan ser clases, herencia, cohesión, polimorfismo, autoboxing y un sinfín de puntos concretos que el libro en cuestión trata para desarrollar el tema de la programación orientada a objetos.

En tercer lugar, más allá de las especificaciones del libro y de la práctica, hacerlo con un diseño cómodo que explore diferentes estrategias de "cómo se pueden hacer las cosas" a la hora de enfrentarse ante un proyecto, basado principalmente en mi experiencia como analista informático, en otras fuentes de aprendizaje que he utilizado y también como repaso o investigación de la asignatura hermana a ésta y de excepcional interés, diseño de aplicaciones orientadas a objetos.

También asegurar, que por mucho programar en Java, C++, php y C#, es posible seguir terminando las frases con un punto y final, y no con un punto y coma al escribir, si se me permite la licencia de bromear con esto.

El cuerpo descriptivo de la memoria consta de los diagramas requeridos por el departamento y por apartados adicionales que se adentran un poco más en la estructura en sí de la aplicación, la cual tiene una serie de características particulares que se entienden mejor a través de pequeñas explicaciones y diagramas adicionales.

No obstante, el objetivo principal es en todo momento desarrollar los conceptos de programación orientada a objetos que se han visto durante el curso, y hacerlo mediante ejemplos concretos, tato técnicos como de estilo.

La comunicación entre los elementos del juego.

He implementado dos formas de hacer comunicarse a las clases.

La primera es la corriente basada en eventos, la cual fue en un principio la única que tenían todas las clases, mediante la utilización de objetos tipo *EventObject*.

La segunda es a través de un patrón tipo Observador, gestionado por una clase intermedia que gestiona las relaciones entre observadores y sujetos observables.

Esta segunda ha sustituido totalmente en una segunda refactorización general a la primera excepto en la comunicación directa con los controles Swing.

Existen dos interfaces genéricas que son *IObservable* e *IObservador*. Estas dos interfaces son implementadas por los objetos que quieren comunicarse entre ellos.

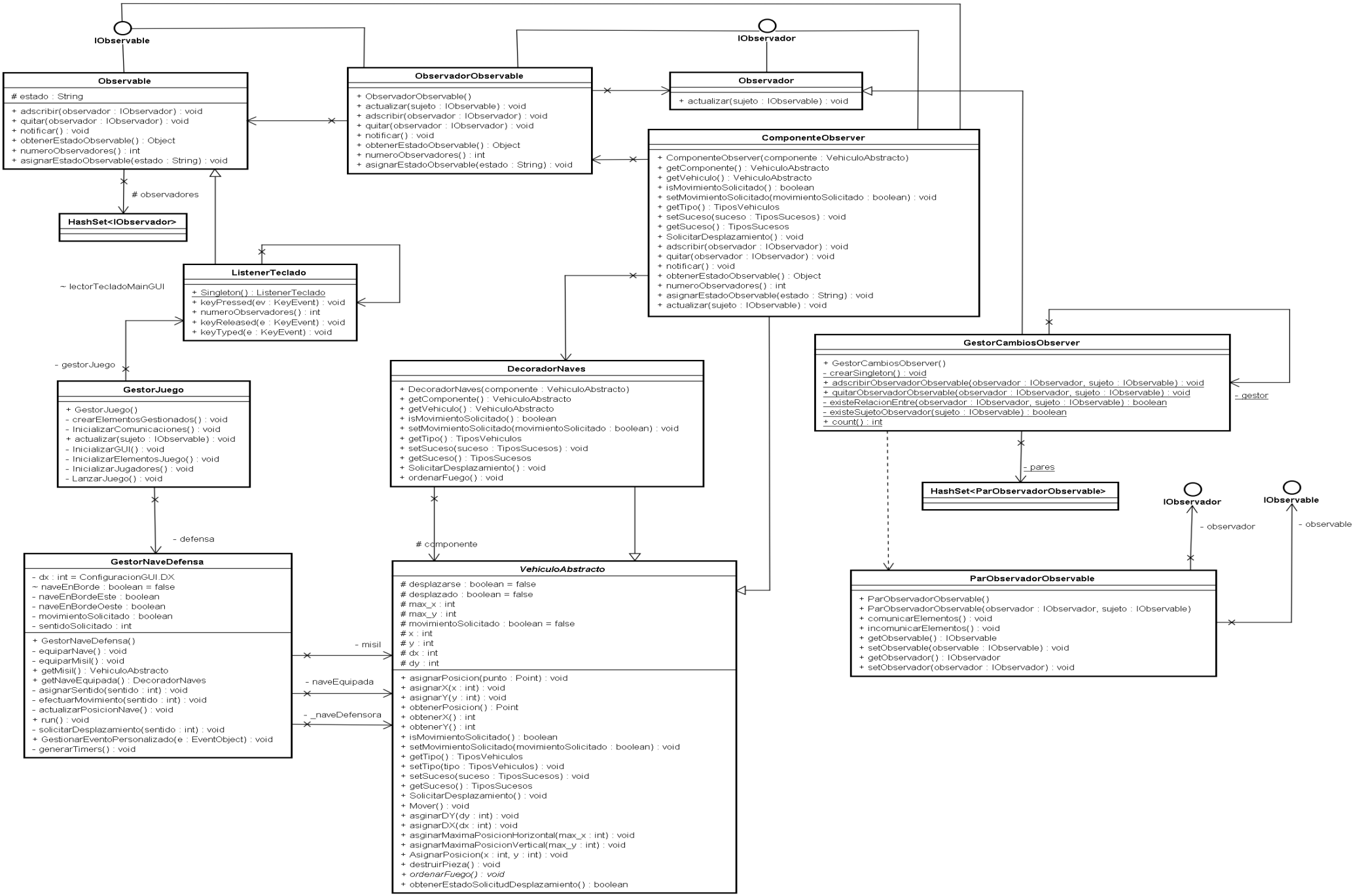
Para **disminuir el acoplamiento** de las clases que se comunican por esta vía he utilizado varias estrategias.

La **primera** ha sido crear una clase tipo *Singleton* llamada *GestorCambioObserver* que se encarga de gestionar mediante métodos estáticos todas las relaciones entre las clases que se comunican.

En **segundo** lugar, para evitar la herencia cuando no ha sido necesario y así la saturación de clases, y para mejorar la **cohesión funcional**, los elementos tipo naves y misiles no tienen como tales posibilidad de comunicarse unos con otros por sí mismos. Tal como se haría en el mundo real, lo que he optado por hacer es agregarles un “equipo de comunicación”, una **extensión**, totalmente compatible con los tipos de naves y misiles pero transparente para ellos. Estas extensiones guardan una referencia del objeto sobre el que se aplican, son intercambiables y su tipo también es compatible con el tipo de los vehículos en todos los sentidos. De hecho, incluso los misiles son extensiones en sí mismos del mismo modo que lo son los dispositivos de comunicación. La superclase *DecoradorNaves* hace posible esta funcionalidad. La subclase de *DecoradorNaves* llamada *ComponenteObserver* es la que realmente implementa *IObservador* e *IObservable* y se puede acoplar a los vehículos (naves) del juego.

En **tercer** lugar, he creado **clases concretas para heredar *Listeners*** si ello ha mejorado la calidad del código, como es el caso de la clase *ListenerTeclado*, la cual hace de **puente** entre *KeyListener* y *Observable*. Esto es porque Observable no funciona con eventos entendidos como objetos *EventObject*, sino con notificaciones basadas en cadenas de texto.

El siguiente diagrama de clases muestra de manera no pormenorizada un esquema de las estrategias adoptadas con el ***GestorJuego*** como eje central.:


IObservable
IObservador
Observable
estado : String
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
ObservadorObservable
+ ObservadorObservable()
+ actualizar(sujeto : IObservable) : void
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
Observador
+ actualizar(sujeto : IObservable) : void
ComponenteObserver
+ ComponenteObserver(componente : VehiculoAbstracto)
+ getComponente() : VehiculoAbstracto
+ getVehiculo() : VehiculoAbstracto
+ isMovimientoSolicitado() : boolean
+ setMovimientoSolicitado(movimientoSolicitado : boolean) : void
+ getTipo() : TiposVehiculos
+ setSuceso(suceso : TiposSucesos) : void
+ getSuceso() : TiposSucesos
+ SolicitarDesplazamiento() : void
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
+ actualizar(sujeto : IObservable) : void
# observadores
HashSet<IObservador>
ListenerTeclado
+ Singleton() : ListenerTeclado
+ keyPressed(ev : KeyEvent) : void
+ numeroObservadores() : int
+ keyReleased(e : KeyEvent) : void
+ keyTyped(e : KeyEvent) : void
~ lectorTecladoMainGUI
- gestorJuego
GestorJuego
+ GestorJuego()
- crearElementosGestionados() : void
- InicializarComunicaciones() : void
+ actualizar(sujeto : IObservable) : void
- InicializarGUI() : void
- InicializarElementosJuego() : void
- InicializarJugadores() : void
- LanzarJuego() : void
- defensa
GestorNaveDefensa
- dx : int = ConfiguracionGUI.DX
~ naveEnBorde : boolean = false
- naveEnBordeEste : boolean
- naveEnBordeOeste : boolean
- movimientoSolicitado : boolean
- sentidoSolicitado : int
+ GestorNaveDefensa()
- equiparNave() : void
- equiparMisil() : void
+ getMisil() : VehiculoAbstracto
+ getNaveEquipada() : DecoradorNaves
- asignarSentido(sentido : int) : void
- efectuarMovimiento(sentido : int) : void
- actualizarPosicionNave() : void
+ run() : void
- solicitarDesplazamiento(sentido : int) : void
+ GestionarEventoPersonalizado(e : EventObject) : void
- generarTimers() : void
- misil
- naveEquipada
- _naveDefensora
DecoradorNaves
+ DecoradorNaves(componente : VehiculoAbstracto)
+ getComponente() : VehiculoAbstracto
+ getVehiculo() : VehiculoAbstracto
+ isMovimientoSolicitado() : boolean
+ setMovimientoSolicitado(movimientoSolicitado : boolean) : void
+ getTipo() : TiposVehiculos
+ setSuceso(suceso : TiposSucesos) : void
+ getSuceso() : TiposSucesos
+ SolicitarDesplazamiento() : void
+ ordenarFuego() : void
# componente
VehiculoAbstracto
desplazarse : boolean = false
desplazado : boolean = false
max_x : int
max_y : int
movimientoSolicitado : boolean = false
x : int
y : int
dx : int
dy : int
+ asignarPosicion(punto : Point) : void
+ asignarX(x : int) : void
+ asignarY(y : int) : void
+ obtenerPosicion() : Point
+ obtenerX() : int
+ obtenerY() : int
+ isMovimientoSolicitado() : boolean
+ setMovimientoSolicitado(movimientoSolicitado : boolean) : void
+ getTipo() : TiposVehiculos
+ setTipo(tipo : TiposVehiculos) : void
+ setSuceso(suceso : TiposSucesos) : void
+ getSuceso() : TiposSucesos
+ SolicitarDesplazamiento() : void
+ Mover() : void
+ asignarDY(dy : int) : void
+ asignarDX(dx : int) : void
+ asginarMaximaPosicionHorizontal(max_x : int) : void
+ asignarMaximaPosicionVertical(max_y : int) : void
+ AsignarPosicion(x : int, y : int) : void
+ destruirPieza() : void
+ ordenarFuego() : void
+ obtenerEstadoSolicitudDesplazamiento() : boolean
GestorCambiosObserver
+ GestorCambiosObserver()
- crearSingleton() : void
+ adscribirObservadorObservable(observador : IObservador, sujeto : IObservable) : void
+ quitarObservadorObservable(observador : IObservador, sujeto : IObservable) : void
- existeRelacionEntre(observador : IObservador, sujeto : IObservable) : boolean
- existeSujetoObservador(sujeto : IObservable) : boolean
+ count() : int
- gestor
- pares
HashSet<ParObservadorObservable>
IObservador
IObservable
- observador
- observable
ParObservadorObservable
+ ParObservadorObservable()
+ ParObservadorObservable(observador : IObservador, sujeto : IObservable)
+ comunicarElementos() : void
+ incomunicarElementos() : void
+ getObservable() : IObservable
+ setObservable(observable : IObservable) : void
+ getObservador() : IObservador
+ setObservador(observador : IObservador) : void

El movimiento de las piezas

Las piezas del juego se pueden mover de dos formas distintas en cuanto a la fuente de la solicitud:

1. **El programa solicita de forma automática el movimiento**.
2. **El usuario solicita al programa que mueva una pieza**.

En el primer caso, un ***Timer*** se encarga de realizar solicitudes periódicas, en intervalos que pueden ser o no **aleatorios**.

En el segundo caso, se hace a través de eventos ***KeyEvent*** sobre la interfaz gráfica del juego, los cuales son recibidos por una clase especial llamada *ListenerTeclado* que hace de puente con el sistema genérico de comunicación basado en un patrón *Observer* que se ha adoptado.

En este apartado se describe el movimiento, siendo la comunicación entre clases materia de otro apartado.

Todos los movimientos son realizados por Vehículos que salen de la factoría de naves o por sus componentes móviles que están definidos en las extensiones de las naves. Todos ellos heredan de VehichuloAbstracto su tipo.

El motivo de heredar de vehículo abstracto es el mismo por el que un GPS universal no sería viable comercialmente si no dispone de un modo de “agarrarse” a un vehículo cualquiera que abarque la mayoría de los que existen en el mercado. En este sentido, el GPS “hereda” del concepto de vehículo parte de su forma, de manera que puede adherirse al cristal del mismo, aún sin ser completamente un vehículo y teniendo en cuenta que los vehículos no tienen porqué estar específicamente preparados para llevar un GPS. Igual podría pasar con una funda para el automóvil o un equipo de música. Incluso conceptualmente es el GPS el que “agarra” al coche y no al revés.

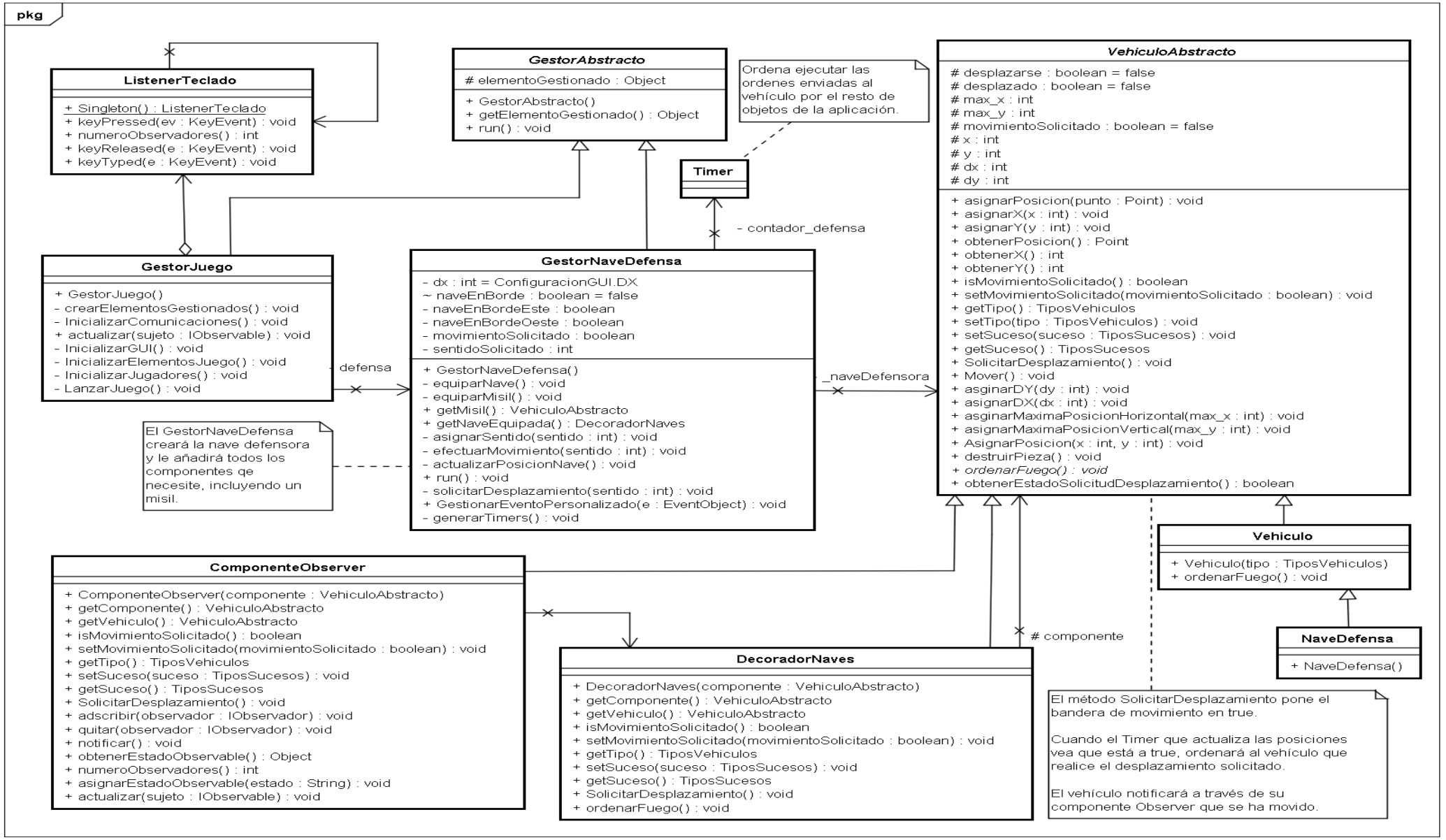
Cuando el sistema quiere mover una pieza del juego, lo realiza a través del gestor de la clase correspondiente. El objeto envía una solicitud de movimiento al vehículo concreto y éste la guarda para atenderla.

A su vez, un *Timer* dentro del mismo gestor se encarga periódicamente de comprobar si alguno de sus objetos gestionados tiene una solicitud de movimiento. Si es así, ordena al vehículo que la ejecute.

Tras ello, se notificará a las clases interesadas que el movimiento se ha realizado (por ejemplo, el tablero podría querer saberlo para actualizar las piezas).

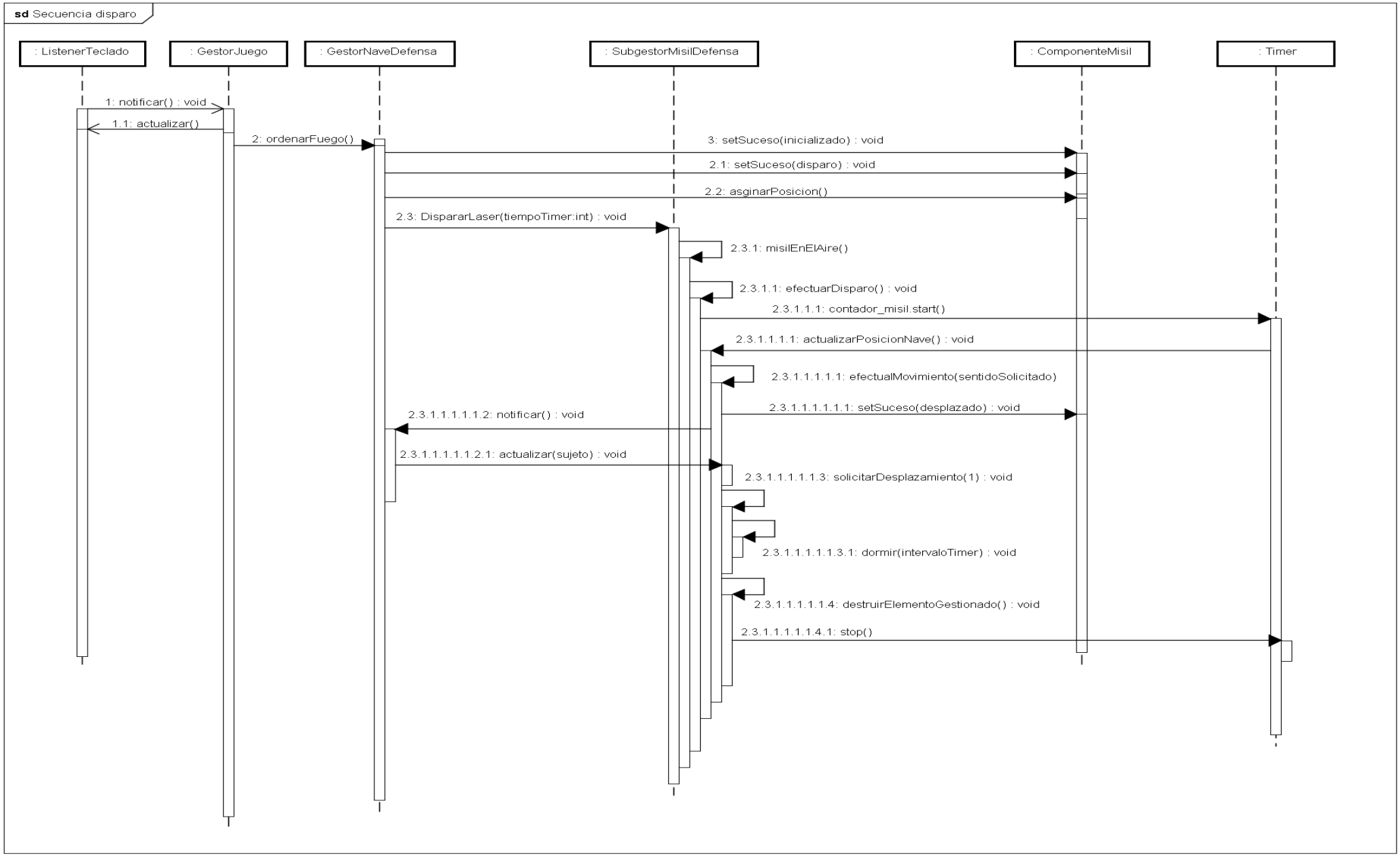
Finalizado este ciclo, el objeto volverá a estado de reposo hasta que se cree otra solicitud de movimiento.

El diagrama siguiente muestra clases relacionadas en el movimiento de la nave de defensa a modo ilustrativo tal como está implementado en este punto del desarrollo.



El movimiento de piezas está muy relacionado con el sistema de disparo. Los misiles usan un algoritmo similar al resto de móviles para desplazarse. A continación se muestra un esquema de cómo está diseñado todo el sistema de disparo de las piezas.

Secuencia de disparo



Al igual que el movimiento, la secuencia de disparo se activa tanto por el usuario en el caso de la nave de defensa como por el propio programa en el caso de las naves UFO.

El movimiento se corresponde con el diagrama de clases anterior del movimiento de los objetos.

En el diagrama de secuencia expuesto muestra de forma bastante exhaustiva los principales pasos que siguen a la solicitud de disparo desde el teclado. Para el caso de las solicitudes realizadas por la aplicación el diagrama es esencialmente el mismo, pues el grueso del mismo reside en superclases abstractas. La diferencia fundamental es que en vez de ser un evento de teclado el que pone en funcionamiento la notificación de que se ha solicitado un disparo, es un *Timer* el que efectúa peticiones periódicas de disparo en intervalos aleatorios dentro de un rango definido previamente.

Los gestores y *sub-gestores* contienen los mecanismos de control de los objetos gestionados y las reglas de movimiento, de forma que toda la lógica del movimiento queda encapsulada fuera de las clases que representan los vehículos. En la vida real las normas de tráfico igualmente se representan en códigos legales y los vehículos en automóviles.

Finalmente, los objetos que constituyen el láser o misil son extensiones de los vehículos, los cuales se acoplan a las naves y a cualquier otra extensión de forma totalmente transparente, pues heredan su forma de la clase *DecoradorNaves*, la cual a su vez hereda de *VehiculoAbstracto*.

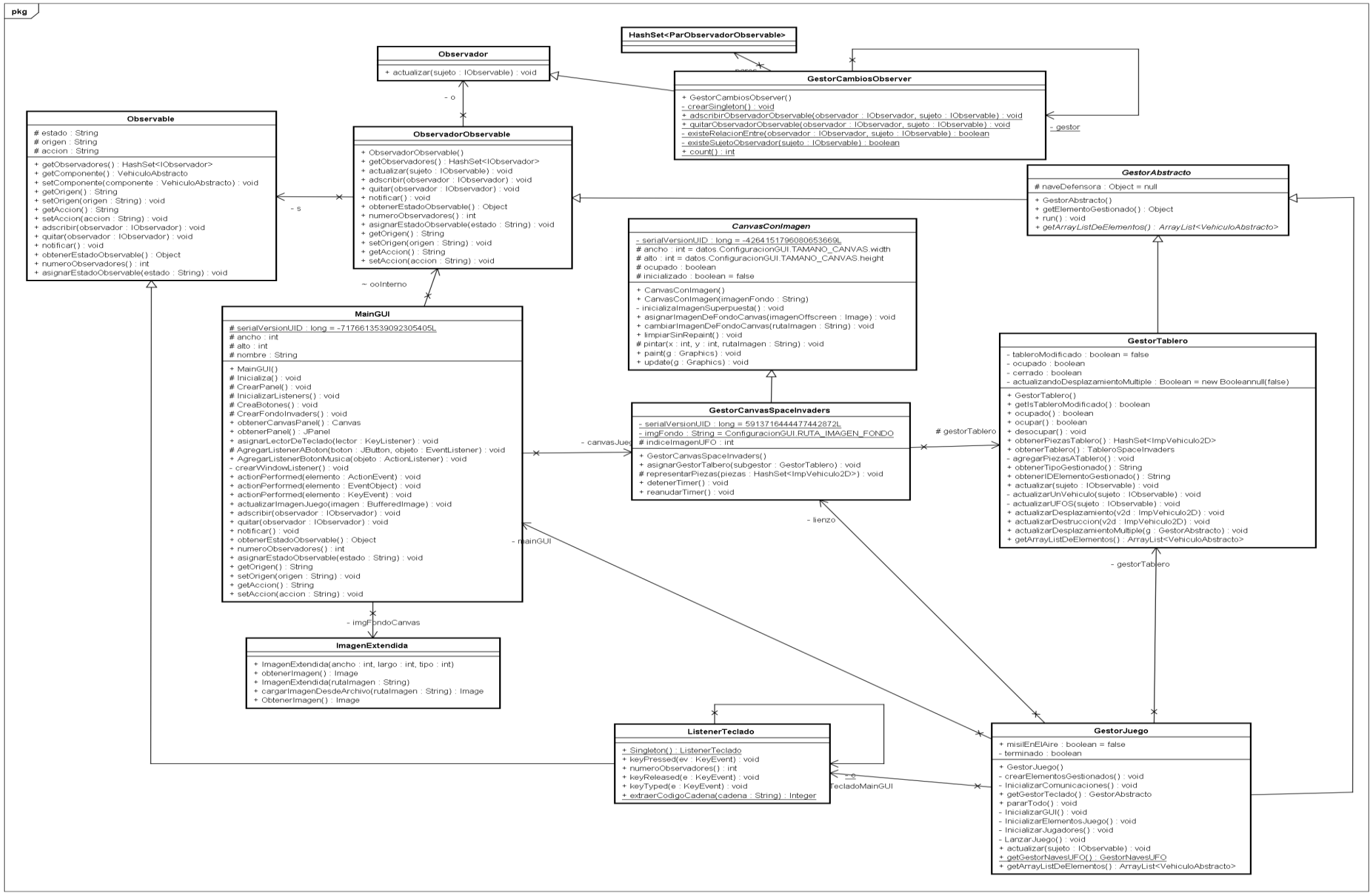
El movimiento y el disparo están organizados por áreas funcionales, de forma que se ejecutan en hilos distintos para que exista mayor fluidez. Debido a que no existe ninguna estructura que necesite un funcionamiento productor-consumidor, puesto que incluso los láseres son siempre el mismo objeto que cambia de estado, la sincronización entre hilos se puede implementar con unas pocas banderas lógicas, y se evita de esta manera hacer uso de métodos o clases tipo *synchronized* que requieren recursos adicionales, y disminuyen el rendimiento del conjunto.

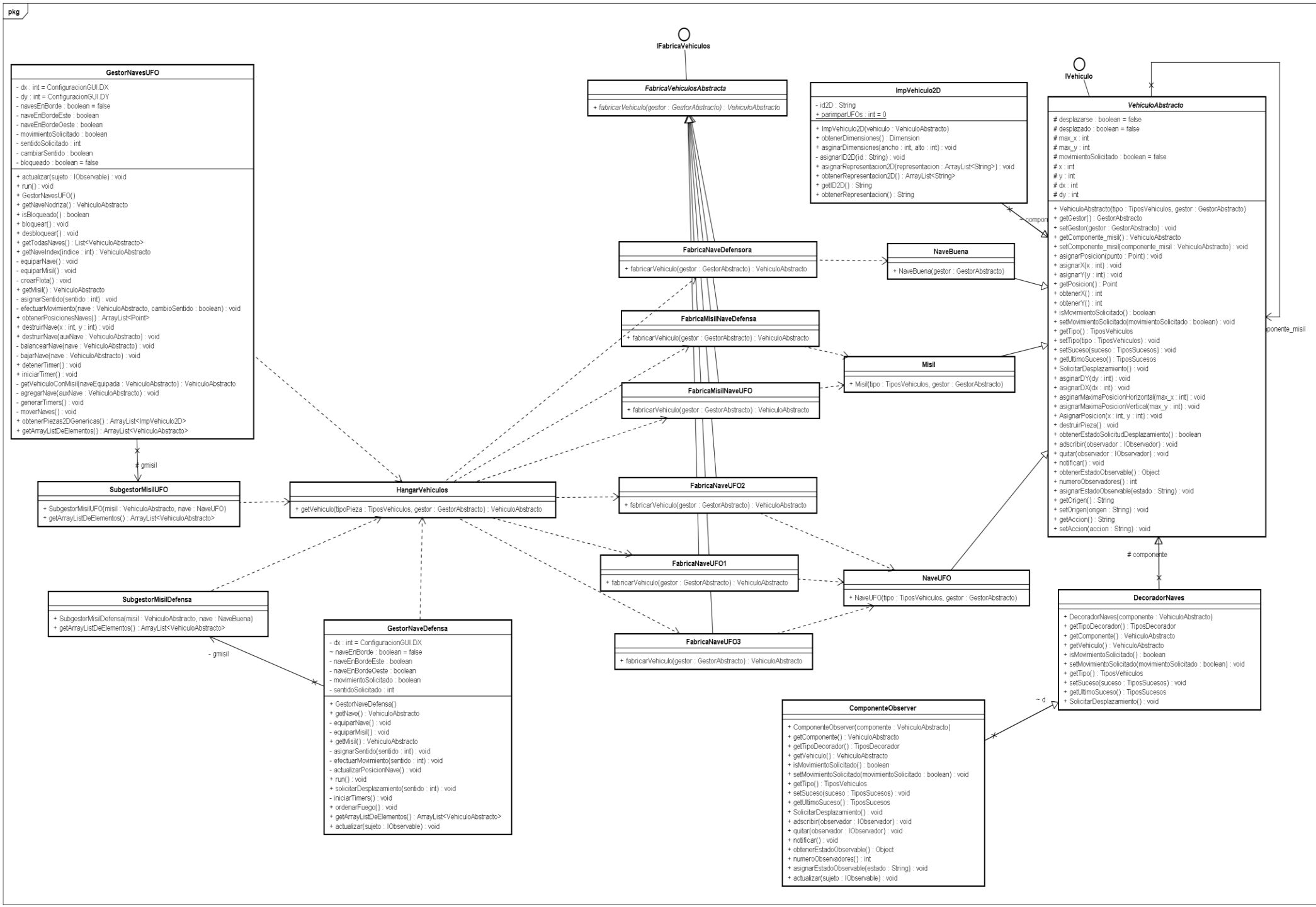
Como se aprecia en el diagrama de secuencia, existe una fuerte jerarquía de clases que intentan hacer coherente el diseño, de manera que dentro de una complejidad razonable, el programa tenga la suficiente modularidad para asegurar una correcta gestión y la mayor facilidad para la reusabilidad.

En la versión actual, los decoradores sólo se usan para implementar observadores, pues la gestión de extensiones para los misiles se ha realizado directamente mediante agregación de unas clases dentro de otras, según una jerarquía gestor-nave-misil.

Página siguiente: Diagrama interfaz gráfica.

-


pkg
HashSet<ParObservadorObservable>
Observador
+ actualizar(sujeto : IObservable) : void
GestorCambiosObserver
+ GestorCambiosObserver()
- crearSingleton() : void
+ adscribirObservadorObservable(observador : IObservador, sujeto : IObservable) : void
+ quitarObservadorObservable(observador : IObservador, sujeto : IObservable) : void
- existeRelacionEntre(observador : IObservador, sujeto : IObservable) : boolean
- existeSujetoObservador(sujeto : IObservable) : boolean
+ count() : int
- gestor
- o
Observable
estado : String
origen : String
accion : String
+ getObservadores() : HashSet<IObservador>
+ getComponente() : VehiculoAbstracto
+ setComponente(componente : VehiculoAbstracto) : void
+ getOrigen() : String
+ setOrigen(origen : String) : void
+ getAccion() : String
+ setAccion(accion : String) : void
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
- s
ObservadorObservable
+ ObservadorObservable()
+ getObservadores() : HashSet<IObservador>
+ actualizar(sujeto : IObservable) : void
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
+ getOrigen() : String
+ setOrigen(origen : String) : void
+ getAccion() : String
+ setAccion(accion : String) : void
~ ooInterno
GestorAbstracto
naveDefensora : Object = null
+ GestorAbstracto()
+ getElementoGestionado() : Object
+ run() : void
+ getArrayListDeElementos() : ArrayList<VehiculoAbstracto>
CanvasConImagen
- serialVersionUID : long = -4264151796080653669L
ancho : int = datos.ConfiguracionGUI.TAMANO_CANVAS.width
alto : int = datos.ConfiguracionGUI.TAMANO_CANVAS.height
ocupado : boolean
inicializado : boolean = false
+ CanvasConImagen()
+ CanvasConImagen(imagenFondo : String)
- inicializaImagenSuperpuesta() : void
+ asignarImagenDeFondoCanvas(imagenOffscreen : Image) : void
+ cambiarImagenDeFondoCanvas(rutaImagen : String) : void
+ limpiarSinRepaint() : void
# pintar(x : int, y : int, rutaImagen : String) : void
+ paint(g : Graphics) : void
+ update(g : Graphics) : void
MainGUI
serialVersionUID : long = -7176613539092305405L
ancho : int
alto : int
nombre : String
+ MainGUI()
# Inicializa() : void
# CrearPanel() : void
# InicializarListeners() : void
# CreaBotones() : void
# CrearFondoInvaders() : void
+ obtenerCanvasPanel() : Canvas
+ obtenerPanel() : JPanel
+ asignarLectorDeTeclado(lector : KeyListener) : void
# AgregarListenerABoton(boton : JButton, objeto : EventListener) : void
+ AgregarListenerBotonMusica(objeto : ActionListener) : void
- crearWindowListener() : void
+ actionPerformed(elemento : ActionEvent) : void
+ actionPerformed(elemento : EventObject) : void
+ actionPerformed(elemento : KeyEvent) : void
+ actualizarImagenJuego(imagen : BufferedImage) : void
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
+ getOrigen() : String
+ setOrigen(origen : String) : void
+ getAccion() : String
+ setAccion(accion : String) : void
- canvasJuego
- mainGUI
- imgFondoCanvas
GestorCanvasSpaceInvaders
- serialVersionUID : long = 5913716444477442872L
- imgFondo : String = ConfiguracionGUI.RUTA_IMAGEN_FONDO
indiceImagenUFO : int
+ GestorCanvasSpaceInvaders()
+ asignarGestorTalbero(subgestor : GestorTablero) : void
# representarPiezas(piezas : HashSet<ImpVehiculo2D>) : void
+ detenerTimer() : void
+ reanudarTimer() : void
# gestorTablero
- lienzo
GestorTablero
- tableroModificado : boolean = false
- ocupado : boolean
- cerrado : boolean
- actualizandoDesplazamientoMultiple : Boolean = new Booleannull(false)
+ GestorTablero()
+ getIsTableroModificado() : boolean
+ ocupado() : boolean
+ ocupar() : boolean
+ desocupar() : void
+ obtenerPiezasTablero() : HashSet<ImpVehiculo2D>
+ obtenerTablero() : TableroSpaceInvaders
- agregarPiezasATablero() : void
+ obtenerTipoGestionado() : String
+ obtenerIDElementoGestionado() : String
+ actualizar(sujeto : IObservable) : void
- actualizarUnVehiculo(sujeto : IObservable) : void
- actualizarUFOS(sujeto : IObservable) : void
+ actualizarDesplazamiento(v2d : ImpVehiculo2D) : void
+ actualizarDestruccion(v2d : ImpVehiculo2D) : void
+ actualizarDesplazamientoMultiple(g : GestorAbstracto) : void
+ getArrayListDeElementos() : ArrayList<VehiculoAbstracto>
- gestorTablero
ImagenExtendida
+ ImagenExtendida(ancho : int, largo : int, tipo : int)
+ obtenerImagen() : Image
+ ImagenExtendida(rutaImagen : String)
+ cargarImagenDesdeArchivo(rutaImagen : String) : Image
+ ObtenerImagen() : Image
ListenerTeclado
+ Singleton() : ListenerTeclado
+ keyPressed(ev : KeyEvent) : void
+ numeroObservadores() : int
+ keyReleased(e : KeyEvent) : void
+ keyTyped(e : KeyEvent) : void
+ extraerCodigoCadena(cadena : String) : Integer
- o
TecladoMainGUI
GestorJuego
+ misilEnElAire : boolean = false
- terminado : boolean
+ GestorJuego()
- crearElementosGestionados() : void
- InicializarComunicaciones() : void
+ getGestorTeclado() : GestorAbstracto
+ pararTodo() : void
- InicializarGUI() : void
- InicializarElementosJuego() : void
- InicializarJugadores() : void
- LanzarJuego() : void
+ actualizar(sujeto : IObservable) : void
+ getGestorNavesUFO() : GestorNavesUFO
+ getArrayListDeElementos() : ArrayList<VehiculoAbstracto>


pkg
IFabricaVehiculos
GestorNavesUFO
- dx : int = ConfiguracionGUI.DX
- dy : int = ConfiguracionGUI.DY
- navesEnBorde : boolean = false
- naveEnBordeEste : boolean
- naveEnBordeOeste : boolean
- movimientoSolicitado : boolean
- sentidoSolicitado : int
- cambiarSentido : boolean
- bloqueado : boolean = false
+ actualizar(sujeto : IObservable) : void
+ run() : void
+ GestorNavesUFO()
+ getNaveNodriza() : VehiculoAbstracto
+ isBloqueado() : boolean
+ bloquear() : void
+ desbloquear() : void
+ getTodasNaves() : List<VehiculoAbstracto>
+ getNaveIndex(indice : int) : VehiculoAbstracto
- equiparNave() : void
- equiparMisil() : void
- crearFlota() : void
+ getMisil() : VehiculoAbstracto
- asignarSentido(sentido : int) : void
- efectuarMovimiento(nave : VehiculoAbstracto, cambioSentido : boolean) : void
+ obtenerPosicionesNaves() : ArrayList<Point>
+ destruirNave(x : int, y : int) : void
+ destruirNave(auxNave : VehiculoAbstracto) : void
- balancearNave(nave : VehiculoAbstracto) : void
- bajarNave(nave : VehiculoAbstracto) : void
+ detenerTimer() : void
+ iniciarTimer() : void
- getVehiculoConMisil(naveEquipada : VehiculoAbstracto) : VehiculoAbstracto
- agregarNave(auxNave : VehiculoAbstracto) : void
- generarTimers() : void
- moverNaves() : void
+ obtenerPiezas2DGenericas() : ArrayList<ImpVehiculo2D>
+ getArrayListDeElementos() : ArrayList<VehiculoAbstracto>
# gmisil
SubgestorMisilUFO
+ SubgestorMisilUFO(misil : VehiculoAbstracto, nave : NaveUFO)
+ getArrayListDeElementos() : ArrayList<VehiculoAbstracto>
HangarVehiculos
+ getVehiculo(tipoPieza : TiposVehiculos, gestor : GestorAbstracto) : VehiculoAbstracto
SubgestorMisilDefensa
+ SubgestorMisilDefensa(misil : VehiculoAbstracto, nave : NaveBuena)
+ getArrayListDeElementos() : ArrayList<VehiculoAbstracto>
- gmisil
GestorNaveDefensa
- dx : int = ConfiguracionGUI.DX
~ naveEnBorde : boolean = false
- naveEnBordeEste : boolean
- naveEnBordeOeste : boolean
- movimientoSolicitado : boolean
- sentidoSolicitado : int
+ GestorNaveDefensa()
+ getNave() : VehiculoAbstracto
- equiparNave() : void
- equiparMisil() : void
+ getMisil() : VehiculoAbstracto
- asignarSentido(sentido : int) : void
- efectuarMovimiento(sentido : int) : void
- actualizarPosicionNave() : void
+ run() : void
+ solicitarDesplazamiento(sentido : int) : void
- iniciarTimers() : void
+ ordenarFuego() : void
+ getArrayListDeElementos() : ArrayList<VehiculoAbstracto>
+ actualizar(sujeto : IObservable) : void
FabricaVehiculosAbstracta
+ fabricarVehiculo(gestor : GestorAbstracto) : VehiculoAbstracto
FabricaNaveDefensora
+ fabricarVehiculo(gestor : GestorAbstracto) : VehiculoAbstracto
FabricaMisilNaveDefensa
+ fabricarVehiculo(gestor : GestorAbstracto) : VehiculoAbstracto
FabricaMisilNaveUFO
+ fabricarVehiculo(gestor : GestorAbstracto) : VehiculoAbstracto
FabricaNaveUFO2
+ fabricarVehiculo(gestor : GestorAbstracto) : VehiculoAbstracto
FabricaNaveUFO1
+ fabricarVehiculo(gestor : GestorAbstracto) : VehiculoAbstracto
FabricaNaveUFO3
+ fabricarVehiculo(gestor : GestorAbstracto) : VehiculoAbstracto
ImpVehiculo2D
- id2D : String
+ parimparUFOs : int = 0
+ ImpVehiculo2D(vehiculo : VehiculoAbstracto)
+ obtenerDimensiones() : Dimension
+ asginarDimensiones(ancho : int, alto : int) : void
- asignarID2D(id : String) : void
+ asignarRepresentacion2D(representacion : ArrayList<String>) : void
+ obtenerRepresentacion2D() : ArrayList<String>
+ getID2D() : String
+ obtenerRepresentacion() : String
NaveBuena
+ NaveBuena(gestor : GestorAbstracto)
Misil
+ Misil(tipo : TiposVehiculos, gestor : GestorAbstracto)
NaveUFO
+ NaveUFO(tipo : TiposVehiculos, gestor : GestorAbstracto)
IVehiculo
VehiculoAbstracto
desplazarse : boolean = false
desplazado : boolean = false
max_x : int
max_y : int
movimientoSolicitado : boolean = false
x : int
y : int
dx : int
dy : int
+ VehiculoAbstracto(tipo : TiposVehiculos, gestor : GestorAbstracto)
+ getGestor() : GestorAbstracto
+ setGestor(gestor : GestorAbstracto) : void
+ getComponente_misil() : VehiculoAbstracto
+ setComponente_misil(componente_misil : VehiculoAbstracto) : void
+ asignarPosicion(punto : Point) : void
+ asignarX(x : int) : void
+ asignarY(y : int) : void
+ getPosicion() : Point
+ obtenerX() : int
+ obtenerY() : int
+ isMovimientoSolicitado() : boolean
+ setMovimientoSolicitado(movimientoSolicitado : boolean) : void
+ getTipo() : TiposVehiculos
+ setTipo(tipo : TiposVehiculos) : void
+ setSuceso(suceso : TiposSucesos) : void
+ getUltimoSuceso() : TiposSucesos
+ SolicitarDesplazamiento() : void
+ asignarDY(dy : int) : void
+ asignarDX(dx : int) : void
+ asignarMaximaPosicionHorizontal(max_x : int) : void
+ asignarMaximaPosicionVertical(max_y : int) : void
+ AsignarPosicion(x : int, y : int) : void
+ destruirPieza() : void
+ obtenerEstadoSolicitudDesplazamiento() : boolean
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
+ getOrigen() : String
+ setOrigen(origen : String) : void
+ getAccion() : String
+ setAccion(accion : String) : void
iponente_misil
componente
DecoradorNaves
+ DecoradorNaves(componente : VehiculoAbstracto)
+ getTipoDecorador() : TiposDecorador
+ getComponente() : VehiculoAbstracto
+ getVehiculo() : VehiculoAbstracto
+ isMovimientoSolicitado() : boolean
+ setMovimientoSolicitado(movimientoSolicitado : boolean) : void
+ getTipo() : TiposVehiculos
+ setSuceso(suceso : TiposSucesos) : void
+ getUltimoSuceso() : TiposSucesos
+ SolicitarDesplazamiento() : void
~ d
ComponenteObserver
+ ComponenteObserver(componente : VehiculoAbstracto)
+ getComponente() : VehiculoAbstracto
+ getTipoDecorador() : TiposDecorador
+ getVehiculo() : VehiculoAbstracto
+ isMovimientoSolicitado() : boolean
+ setMovimientoSolicitado(movimientoSolicitado : boolean) : void
+ getTipo() : TiposVehiculos
+ setSuceso(suceso : TiposSucesos) : void
+ getUltimoSuceso() : TiposSucesos
+ SolicitarDesplazamiento() : void
+ adscribir(observador : IObservador) : void
+ quitar(observador : IObservador) : void
+ notificar() : void
+ obtenerEstadoObservable() : Object
+ numeroObservadores() : int
+ asignarEstadoObservable(estado : String) : void
+ actualizar(sujeto : IObservable) : void