

tivo

Original

«Abstract» FabricaSprite
String
ta(): Sprite dor(): Sprite onioRojo(): Sprite \marillo(): Sprite iDeFuego(): Sprite \azaba(): Sprite asma(): Sprite): Sprite): Sprite e(): Sprite (): Sprite): Sprite leras(): Sprite des(): Sprite desDestructibles(): Sprite oResbaladizo(): Sprite paPinchos(): Sprite ica(): Sprite es(): Sprite radiza(): Sprite era(): Sprite akichi(): Sprite

FabricaEntidades
- fabrica: FabricaSprite
+ getSilueta(): Sprite + getJugador(x: int, y: int): Jugador + getDemonioRojo(x: int, y: int): DemonioRojo + getTrollAmarillo(x: int, y: int): TrollAmarillo + getRanaDeFuego(x: int, y: int): RanaDeFuego + getCalabaza(x: int, y: int): Calabaza + getFantasma(x: int, y: int): Fantasma + getAzul(x: int, y: int): BotellaAzul + getRoja(x: int, y: int): BotellaRoja + getVerde(x: int, y: int): BotellaVerde + getFruta(x: int, y: int): Fruta + getVida(x: int, y: int): VidaExtra + getEscaleras(x: int, y: int): Escaleras + getParedes(x: int, y: int): Paredes + getParedesDestructibles(x: int, y: int): ParedDestructible + getSueloResbaladizo(x: int, y: int): SueloResbaladizo + getTrampaPinchos(x: int, y: int): TrampaPinchos + getPlataforma(x: int, y: int): Plataforma + getMoviles(x: int, y: int): PlataformaMovil + getQuebradiza(x: int, y: int): PlataformaQuebradiza + getMoghera(x: int, y: int): Moghera + getKamakichi(x: int,y: int): Kamakichi

ParserNivel
- fabrica: FabricaEntidades
+ generarNivel(modo:int): Nivel - cargarNivel(archivo:String, modo:int): Nivel + obtenerArchivoNivel(modo:int):String

int, alto: int): void



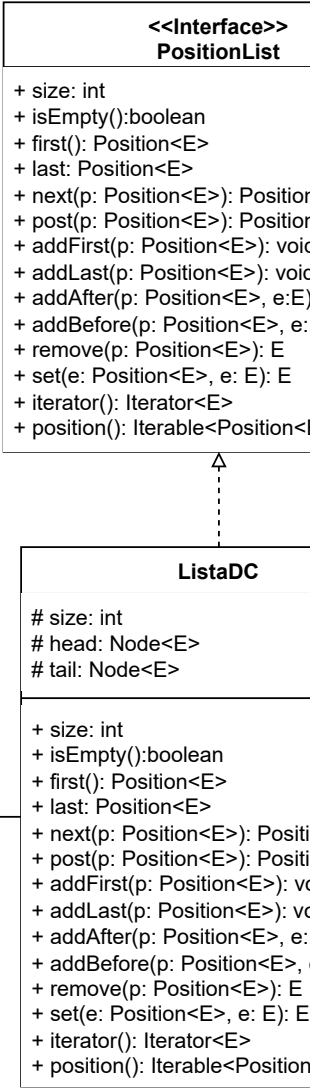
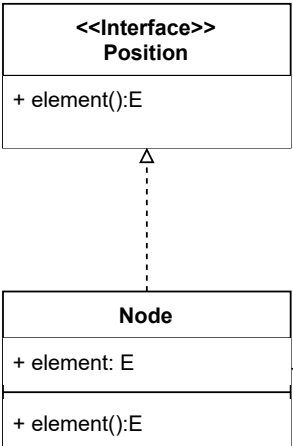
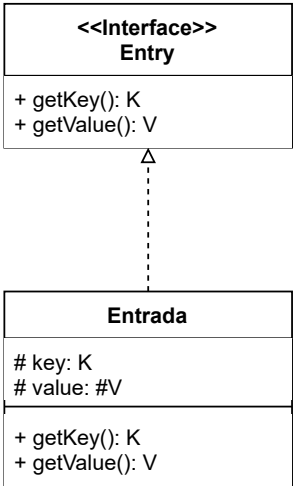
Rosa: incor
Azul: faltan
Verde: dud
Neronic: si

nsistencias entre los 2 diagramas(o revisar posible problema)
ordenar
as
stema de orden

Preguntar por los niveles y los sprites

+ transparentarBoton(JButton boton)

+ transparentarBoton(JButton boton)




```

+ registrarOyenteBotonClasico()
+ registrarOyenteBotonContraTimer()
+ registrarOyenteBotonSupervivencia()
+ transparentarBoton(boton: JButton)

```

```

+ actualizarLabelsPuntaje(j: Jugador)
+ actualizarLabelsTiempo(j: Jugador)
+ actualizarScrollHaciaJugador(j: Jugador)
+ centrarEnJugador(posXJugador: int)
+ agregarPanelJuegoConFondoYScroll()
+ agregarLabelsEditablesTiempo()
+ agregarLabelsEditablesPuntaje()
+ agregarLabelsEditablesVidas()
+ decorarLabelsTiempo()
+ decorarLabelsPuntaje()
+ decorarLabelsVidas()

```

```

+ agregarBotonDecorar
+ registrarOyenteBotonRe
+ registrarOyenteBotonSe
+ registrarOyenteBotonNi
+ transparentarBoton(bot

```

```

n<E>
n<E>
d
d
): void
E): void

E>>

```

```

<<Interface>>
Map

+ size(): int
+ isEmpty(): boolean
+ get(key:K): V
+ put(key:K, value:V)
+ remove(key: K): V
+ keys():Iterable<K>
+ values(): Iterable<V>
+ entries(): Iterable<K,V>

```

```

HashMap<K,V>

- capacidadInicial: int
# buckets[]: ListaDC<Entrada<K,V>>
- size: int

+ size(): int
+ isEmpty(): boolean
+ get(key:K): V
+ put(key:K, value:V)
+ remove(key: K): V
+ keys():Iterable<K>
+ values(): Iterable<V>
+ entries(): Iterable<K,V>

ion<E>
ion<E>
oid
oid
E): void
e:E): void
:
<E>>

```

```

Ei

# direccion : int
# observer: Obs
# estadoAereo :
# velocidad: int

+ mover(): void
+ actualizar(dire
+ registrarObser
+ establecerEstad

```

```

BolaFuego

- duracion(): Timer

//direccion de disparo depende del
enemigo

Bol

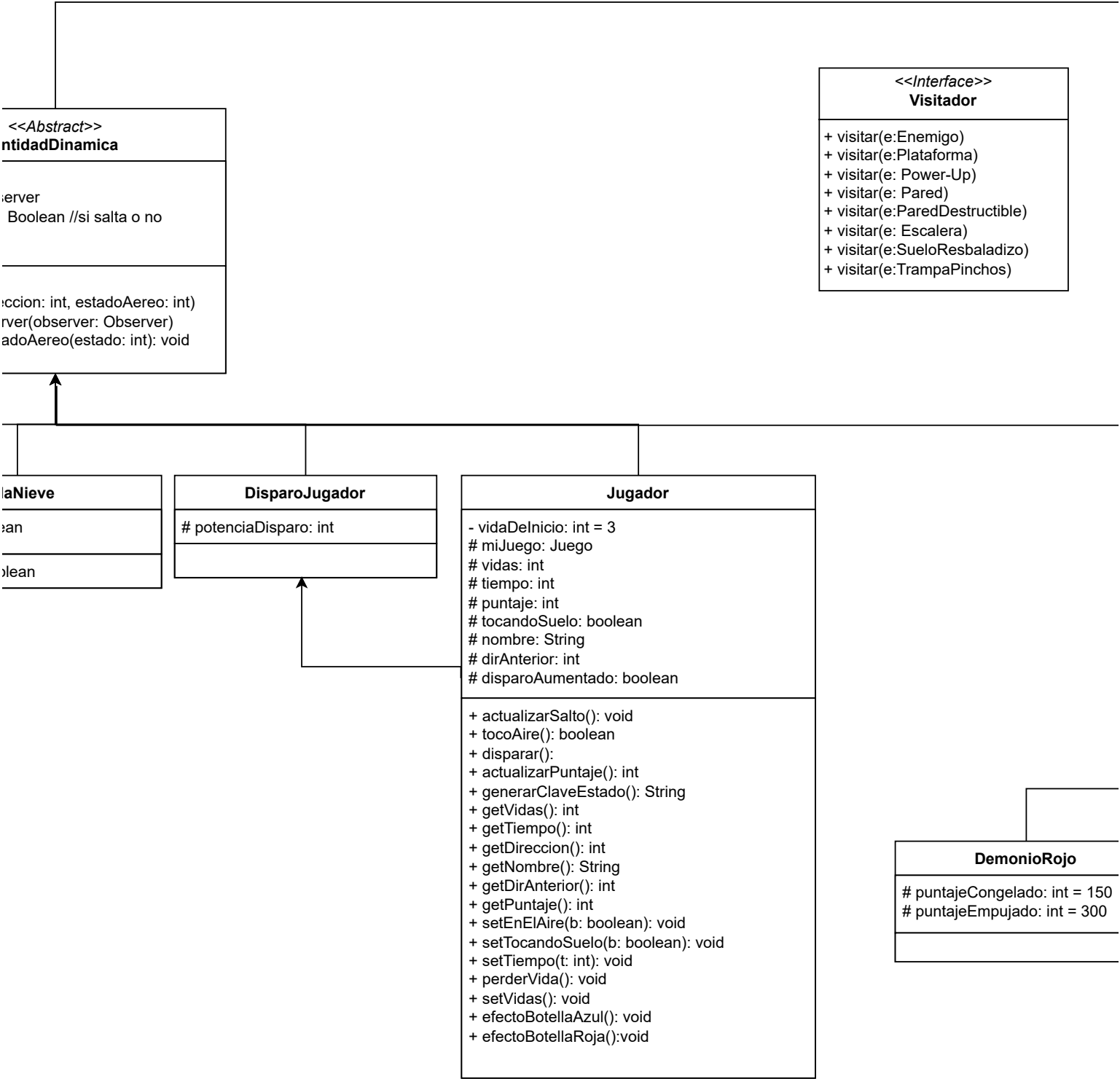
-estaActiva: boole

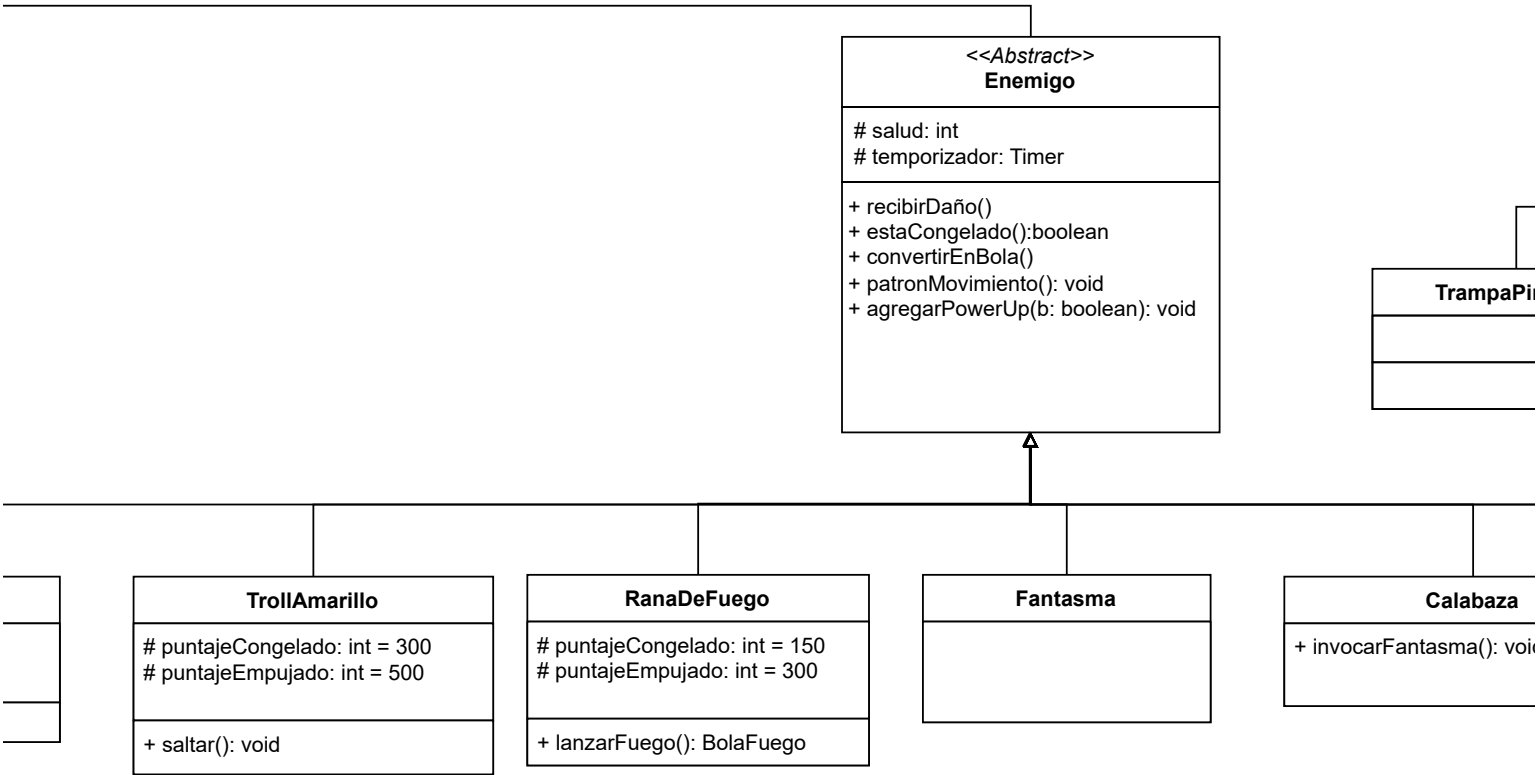
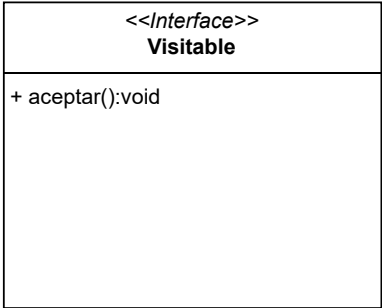
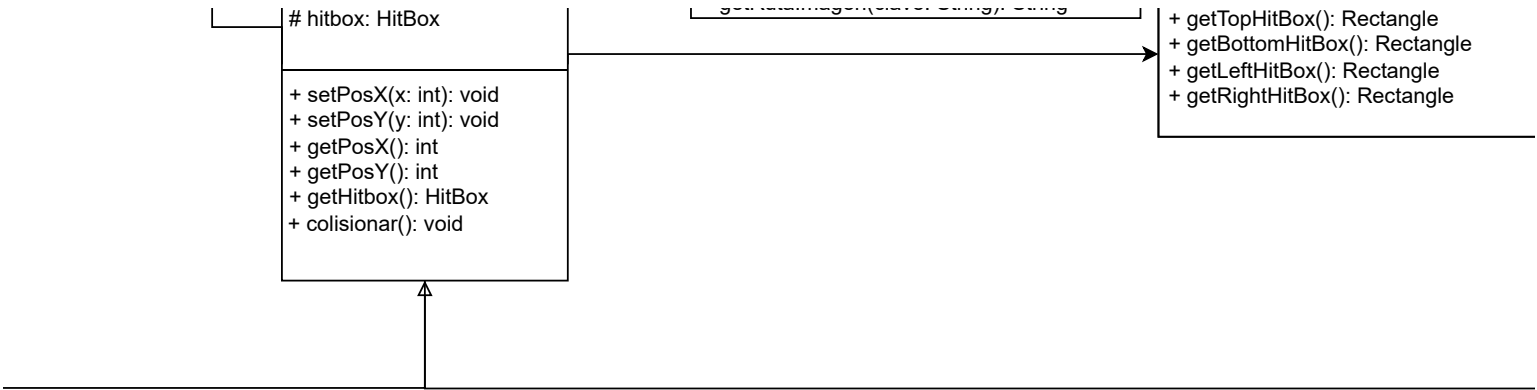
+getEstado(): boo

```

...nuevo(),
iniciar(),
salir(),
nuevaPartida(),
on: JButton)

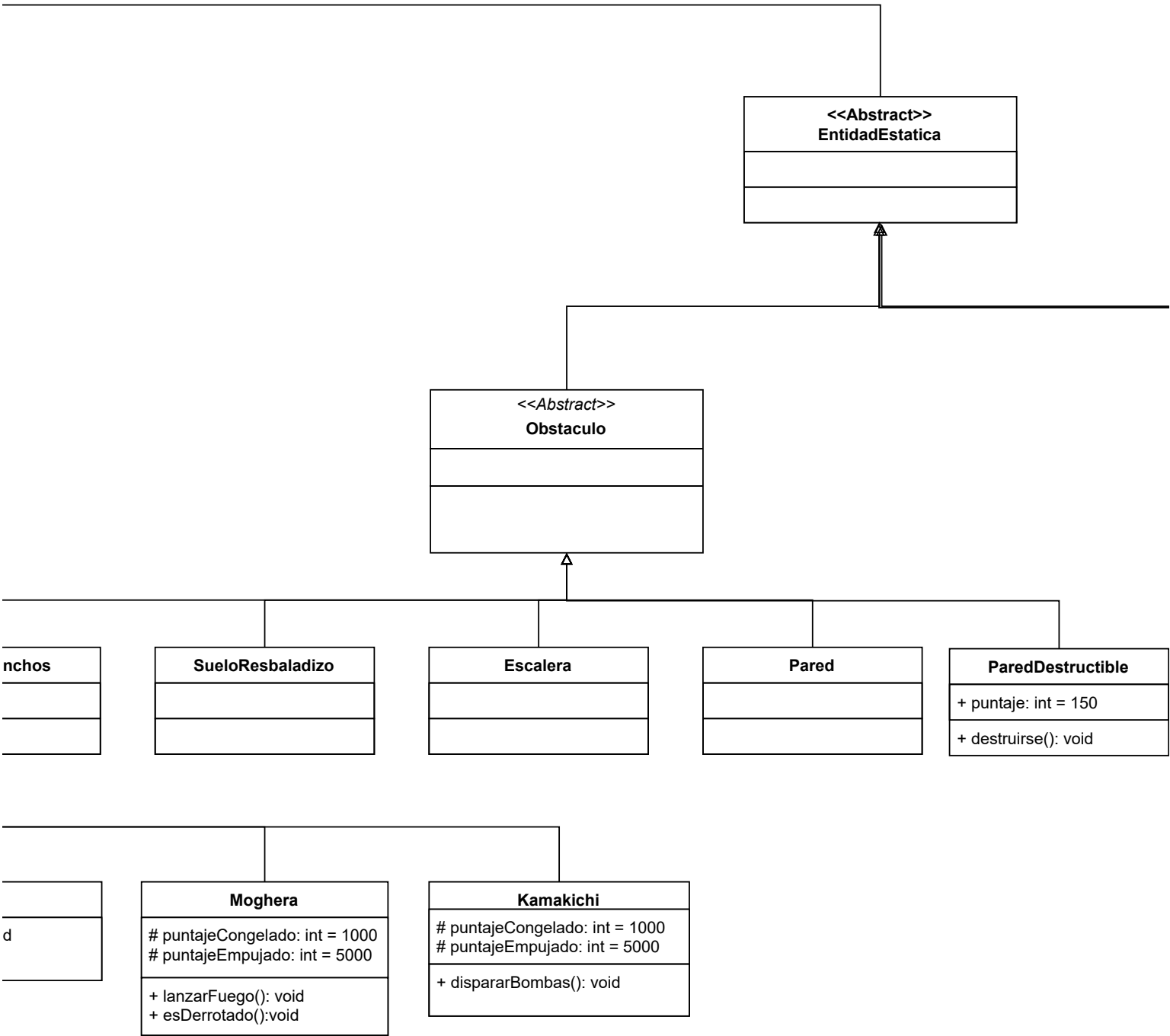
Pendiente: ver como ingresar usuario para el ranking



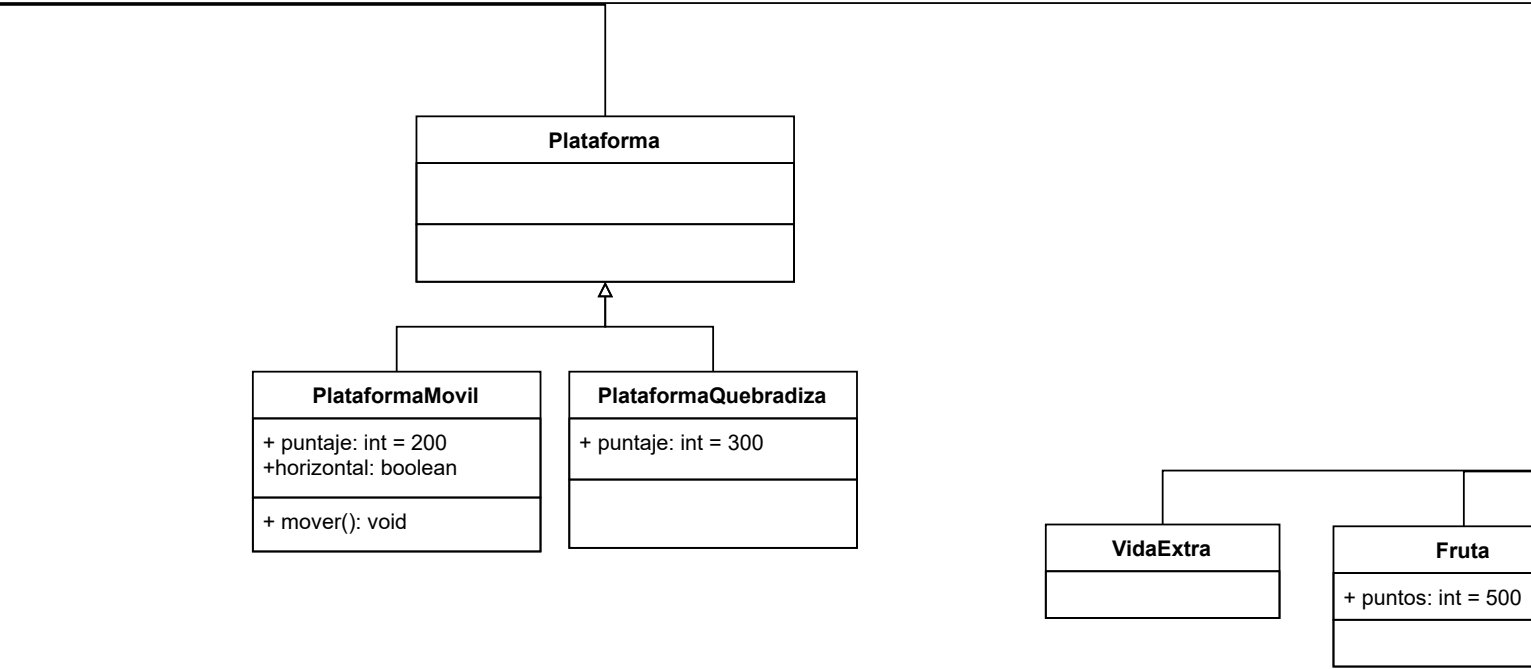


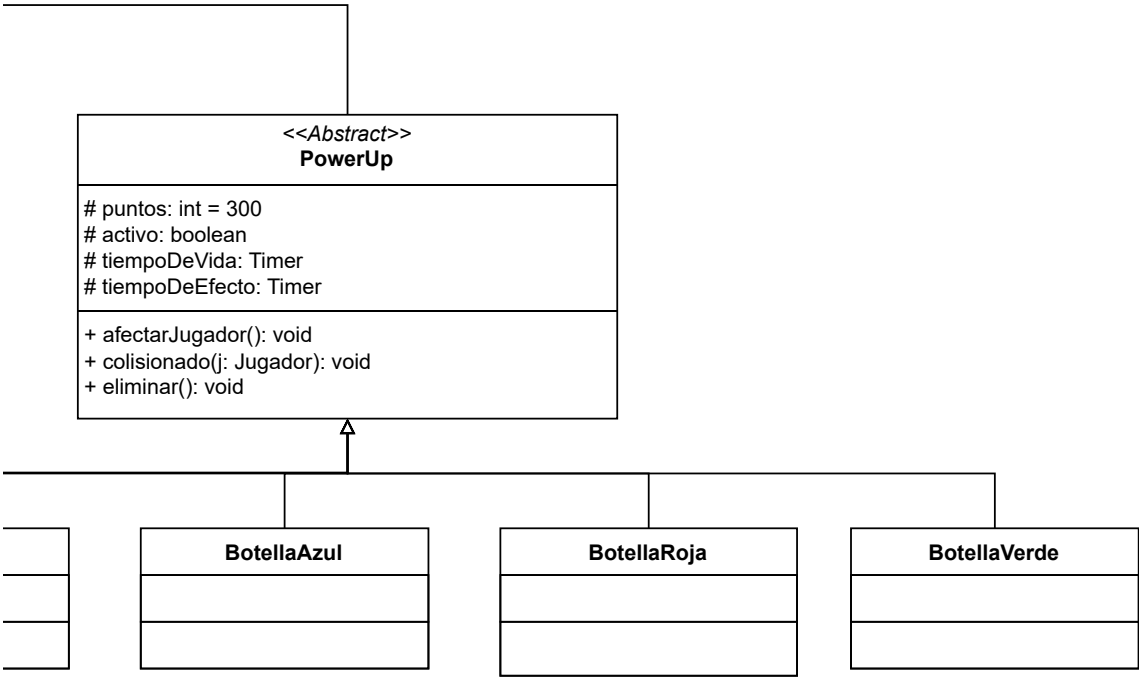


Naranja: si
Amarillo: fa
Violeta: dur
Blanco: info
Transparen



sistema de orden
 faltan cosas(aun definiendola)
 da si borrar
 o del proyecto
 antes: faltan diseñar





<<Interface>> Observer
+ actualizar(): void + actualizar(direccion: int, estadoAereo: int): void

ObserverGrafico
#entidadObservada: Entidad
+ actualizar(): void # actualizarImagen(): void # actualizarPosicionTamano(): void

ObserverEntidades	ObserverJugador
	# panelPantallaJuego: PanelPantallaJuego # jugadorObservado: Jugador
+ actualizar(direccion: int, estadoAereo: int): void	+ actualizar(): void + actualizar(direccion: int, estadoAereo: int): void

PanelPantallaPrincipal
imagenFondo: JLabel # botonIniciar: JButton # botonRanking: JButton # miControladorGrafico: ControladorGrafico
+ agregarImagenFondo() + agregarBotonIniciar() + agregarBotonRanking() + registrarOyenteBotonIniciar() + registrarOyenteBotonRanking() + transparentarBoton(JButton boton)

PanelPantallaDominio
imagenFondo: JLabel # botonOriginal: JButton # botonAlternativo: JButton # miControladorGrafico: ControladorGrafico
+ agregarImagenFondo() + agregarBotonOriginal() + agregarBotonAlternativo() + registrarOyenteBotonOriginal() + registrarOyenteBotonAlternativo() + transparentarBoton(JButton boton)

«interface» ControladoraGrafico
+ registrarControladorJuego(j Juego): void + mostrarPantallaIncial(): void + mostarPantallaJuego(): void + mostrarPantallaFinJuego(): void + registrarEntidad(e: Entidad): Observer + registrarEntidad(j: Jugador): Observer + registrarSilueta(e: Entidad): Observer + accionarInicioJuego(): void

«interface» ControladorDeVista
+ accionarInicioJuego(): void + accionarPantallaRanking(): void + accionarPantallaModoJuego(): void + cambiarModoJuego(modo: int): void

ControladorGrafica
ventana: JFrame # panelPantallaPrincipal: PanelPantallaPrincipal # panelPantallaJuego:Panel: PantallaJuego # panelPantallaModos: PanelPantallaModo # panelPantallaDominio: PanelPantallaDominio # panelPantallaFinDelJuego: PanelPantallaFinDelJuego # panelPantallaRanking: PanelPantallaRanking # miJuego: Juego
- configurarVentana():void # registrarOyenteVentana():Void - addWindowKeyListener():void + mostrarPantallaInicial():void + mostrarPantallaJuego():void + mostrarPantallaFinDeJuego():void + accionarInicioJuego():void + accionarPantallaModoJuego():void + accionarPantallaDominio():void + cambiarModoJuego():void + pasarNivel():void + reiniciarNivel():void + registrarEntidad(e: Entidad): Observer + registrarEntidad(j: Jugador): Observer + registrarSprite(e: Entidad): Observer + getJuego(): Juego

PanelPantallaModo
imagenFondo: JLabel # botonClasico: JButton # botonContraTimer: JButton # botonSupervivencia: JButton # miControladorGrafico: ControladorGrafico
+ agregarImagenFondo() + agregarBotonClasico() + agregarBotonSupervivencia() + agregarBotonContraTimer()

PanelPantallaJuego
panelJuego: JPanel # imagenFondoJuego: JLabel # labelPuntaje: JLabel # labelNivel: JLabel # labelVidas: JLabel # labelTiempo: JLabel # miControladorGrafico: ControladorGrafico
+ actualizarInfoJugador(j: Jugador) + actualizarLabelsVidas(j: Jugador)

PanelPantallaRanking
imagenFondo: JLabel # botonReinicio: JButton # botonSalir: JButton # botonNuevaPartida: JButton # miControladorGrafico: ControladorGrafico
+ agregarImagenFondo() + agregarBotonReinicio() + agregarBotonSalir() + agregarBotonNuevaPartida()

aFinDeJuego
utton ControladorGrafico
rtida()

PanelPantallaRanking
imagenFondo: JLabel # botonSalir: JButton # miControladorGrafico: ControladorGrafico
+ agregarImagenFondo() + agregarBotonSalir() + registrarOyenteBotonSalir() + transparentarBoton(boton: JButton)

Ranking
topClasica: MapHash<String, int> # topSupervivencia: MapHash<String, int> # top:ContrarTimer MapHash<String, int>
+ setNuevo(s: String, i: int): void # ordenarTop(): void + getTop(): Entrada<String, int>

Lat
+ main()

niv # pu # tir # rei # jug # so # rar # co # do
+ ini + rei + su + ac + ac + ge + ge + ge + ge + ge + ge

silu # jug # floo # lista # lista # lista # lista # lista # lista
+ elim + elim + elim + elim + elim + agre + agre + agre + agre + efec

launcher

Juego
vel: Nivel ntajeJugador: int ner: Timer iniciarJugador: boolean gador: Jugador nido: Sonido nking: Ranking ntroladorGrafico: ControladorGrafica minio: FabricaSprite
iciar(): void iniciarNivel(): void perarNivel(): void tualizarVida(): void tualizarPuntaje(puntajeASumar: int): void tVidasJugador(): int tPuntaje(): int tNivelActual(): Nivel tTiempoActual(): int tRanking(): Ranking tModoDeJuego(): int tControladorGrafico(): ControladorGrafica

Nivel
eta: Sprite ador: Jugador r: int aEnemigos: ListaDC<Entidad> aPowerUps: ListaDC<Entidad> aObstaculos: ListaDC<Entidad> aPlataformas: ListaDC<Entidad> aDisparoJugador: ListaDC<Entidad>
inarPlataforma(p: Entidad): void inarJugador(j: Entidad): void inarEnemigo(e: Entidad): void inarPowerUp(p: Entidad): void inarBolaDeFuego(b: Entidad): void inarBolaDeNieve(b: Entidad): void garJugador(j: Entidad): void garPlataforma(p: Entidad): void garPowerUp(p: Entidad): void garEnemigo(e: Entidad): void garBolaDeNieve(b: Entidad): void toBotellaVerde(): void

Sonido
- instancia:Sonido - mapeo: Map <String,Clip>
- Sonido() + getInstancia(): Sonido + reproducir(s: String): void - cargarSonido(r:String) :Clip

Timer
- tiempoActual: int - tiempoMaximo: int
+incrementar(): void +decrementar(): void +llegoAlTope(): boolean +getTiempo(): int +setTiempoActual(): void

Alternativa

rutaACarpeta:
+ getSpriteSiluet + getSpriteJugac + getSpriteDemc + getSpriteTrollA + getSpriteRana + getSpriteCalab + getSpriteFanta + getSpriteAzul(+ getSpriteRoja(+ getSpriteVerde + getSpriteFruta + getSpriteVida(+ getSpriteEscal + getSpriteParec + getSpriteParec + getSpriteSuelc + getSpriteTram + getSpriteEstati + getSpriteMovili + getSpriteQueb + getSpriteMogh + getSpriteKame

<<Abstract>> Entidad
sprite: Sprite # posX: int # posY: int

Sprite
+ mapeoDeSprites: HashMap<String,String> + getRutaImagen(clave: String): String

HitBox
- hitBox: Rectangle
+ actualizarHitbox(x: int, y: int, ancho: i + getHitBox(): Rectangle

tivo	Original

«Abstract» FabricaSprite
String
ta(): Sprite dor(): Sprite onioRojo(): Sprite \marillo(): Sprite iDeFuego(): Sprite abaza(): Sprite asma(): Sprite): Sprite): Sprite e(): Sprite (): Sprite): Sprite leras(): Sprite des(): Sprite desDestructibles(): Sprite oResbaladizo(): Sprite paPinchos(): Sprite ica(): Sprite es(): Sprite radiza(): Sprite era(): Sprite akichi(): Sprite

FabricaEntidades
- fabrica: FabricaSprite
+ getSilueta(): Sprite + getJugador(x: int, y: int): Jugador + getDemonioRojo(x: int, y: int): DemonioRojo + getTrollAmarillo(x: int, y: int): TrollAmarillo + getRanaDeFuego(x: int, y: int): RanaDeFuego + getCalabaza(x: int, y: int): Calabaza + getFantasma(x: int, y: int): Fantasma + getAzul(x: int, y: int): BotellaAzul + getRoja(x: int, y: int): BotellaRoja + getVerde(x: int, y: int): BotellaVerde + getFruta(x: int, y: int): Fruta + getVida(x: int, y: int): VidaExtra + getEscaleras(x: int, y: int): Escaleras + getParedes(x: int, y: int): Paredes + getParedesDestructibles(x: int, y: int): ParedDestructible + getSueloResbaladizo(x: int, y: int): SueloResbaladizo + getTrampaPinchos(x: int, y: int): TrampaPinchos + getPlataforma(x: int, y: int): Plataforma + getMoviles(x: int, y: int): PlataformaMovil + getQuebradiza(x: int, y: int): PlataformaQuebradiza + getMoghera(x: int, y: int): Moghera + getKamakichi(x: int,y: int): Kamakichi

ParserNivel
- fabrica: FabricaEntidades
+ generarNivel(modo:int): Nivel - cargarNivel(archivo:String, modo:int): Nivel + obtenerArchivoNivel(modo:int):String

int, alto: int): void

+ transparentarBoton(JButton boton)	+ transparentarBoton(JButton boton);
-------------------------------------	--------------------------------------

<<Interface>> Entry
+ getKey(): K + getValue(): V

<<Interface>> Position
+ element():E

Entrada
key: K # value: #V
+ getKey(): K + getValue(): V

Node
+ element: E
+ element():E

<<Interface>> PositionList
+ size: int + isEmpty():boolean + first(): Position<E> + last: Position<E> + next(p: Position<E>): Position<E> + post(p: Position<E>): Position<E> + addFirst(p: Position<E>): void + addLast(p: Position<E>): void + addAfter(p: Position<E>, e:E) + addBefore(p: Position<E>, e: E) + remove(p: Position<E>): E + set(e: Position<E>, e: E): E + iterator(): Iterator<E> + position(): Iterable<Position<E>>

ListaDC
size: int # head: Node<E> # tail: Node<E>
+ size: int + isEmpty():boolean + first(): Position<E> + last: Position<E> + next(p: Position<E>): Position<E> + post(p: Position<E>): Position<E> + addFirst(p: Position<E>): void + addLast(p: Position<E>): void + addAfter(p: Position<E>, e: E) + addBefore(p: Position<E>, e: E) + remove(p: Position<E>): E + set(e: Position<E>, e: E): E + iterator(): Iterator<E> + position(): Iterable<Position<E>>

└

+ registrarOyenteBotonClasico()
+ registrarOyenteBotonContraTimer()
+ registrarOyenteBotonSupervivencia()
+ transparentarBoton(boton: JButton)

+ actualizarLabelsPuntaje(j: Jugador)
+ actualizarLabelsTiempo(j: Jugador)
+ actualizarScrollHaciaJugador(j: Jugador)
+ centrarEnJugador(posXJugador: int)
+ agregarPanelJuegoConFondoYScroll()
+ agregarLabelsEditablesTiempo()
+ agregarLabelsEditablesPuntaje()
+ agregarLabelsEditablesVidas()
+ decorarLabelsTiempo()
+ decorarLabelsPuntaje()
+ decorarLabelsVidas()

+ agregarBotonDecorarEnemigo()
+ registrarOyenteBotonReiniciar()
+ registrarOyenteBotonSalir()
+ registrarOyenteBotonNivel()
+ transparentarBoton(boton: JButton)

n<E>
n<E>
d
d
>): void
>E): void

E>>

<<Interface>>
Map

+ size(): int
+ isEmpty(): boolean
+ get(key:K): V
+ put(key:K, value:V)
+ remove(key: K): V
+ keys():Iterable<K>
+ values(): Iterable<V>
+ entries(): Iterable<K,V>

Enemigo

direccion : int
observer: Obs
estadoAereo :
velocidad: int

+ mover(): void
+ actualizar(direccion: int)
+ registrarObserver()
+ establecerEstadoAereo(int)

ion<E>
ion<E>
oid
oid
>E): void
>E): void

:

n<E>>

HashMap<K,V>

- capacidadInicial: int
buckets[]: ListaDC<Entrada<K,V>>
- size: int

+ size(): int
+ isEmpty(): boolean
+ get(key:K): V
+ put(key:K, value:V)
+ remove(key: K): V
+ keys():Iterable<K>
+ values(): Iterable<V>
+ entries(): Iterable<K,V>

BolaFuego

- duracion(): Timer

//direccion de disparo depende del enemigo

Bola

-estaActiva: boolean

+getEstado(): boolean

... einicio() alir() uevaPartida() on: JButton)	
---	--

<<Abstract>> ntidadDinamica
erver Boolean //si salta o no
ccion: int, estadoAereo: int) rver(observer: Observer) adoAereo(estado: int): void

<<Interface>> Visitador
+ visitar(e:Enemigo) + visitar(e:Plataforma) + visitar(e: Power-Up) + visitar(e: Pared) + visitar(e:ParedDestructible) + visitar(e: Escalera) + visitar(e:SueloResbaladizo) + visitar(e:TrampaPinchos)

laNieve	DisparoJugador
an	# potenciaDisparo: int
lean	

Jugador
- vidaDelInicio: int = 3 # miJuego: Juego # vidas: int # tiempo: int # puntaje: int # tocandoSuelo: boolean # nombre: String # dirAnterior: int # disparoAumentado: boolean + actualizarSalto(): void + tocoAire(): boolean + disparar(): + actualizarPuntaje(): int + generarClaveEstado(): String + getVidas(): int + getTiempo(): int + getDireccion(): int + getNombre(): String + getDirAnterior(): int + getPuntaje(): int + setEnElAire(b: boolean): void + setTocandoSuelo(b: boolean): void + setTiempo(t: int): void + perderVida(): void + setVidas(): void + efectoBotellaAzul(): void + efectoBotellaRoja():void

DemonioRojo
puntajeCongelado: int = 150 # puntajeEmpujado: int = 300

hitbox: HitBox
+ setPosX(x: int): void + setPosY(y: int): void + getPosX(): int + getPosY(): int + getHitbox(): HitBox + colisionar(): void

+ getTopHitBox(): Rectangle + getBottomHitBox(): Rectangle + getLeftHitBox(): Rectangle + getRightHitBox(): Rectangle
--

<<Interface>> Visitable
+ aceptar():void

<<Abstract>> Enemigo
salud: int # temporizador: Timer
+ recibirDaño() + estaCongelado():boolean + convertirEnBola() + patronMovimiento(): void + agregarPowerUp(b: boolean): void

TrampaPii

TrollAmarillo
puntajeCongelado: int = 300 # puntajeEmpujado: int = 500
+ saltar(): void

RanaDeFuego
puntajeCongelado: int = 150 # puntajeEmpujado: int = 300
+ lanzarFuego(): BolaFuego

Fantasma

Calabaza
+ invocarFantasma(): void



<<Abstract>> EntidadEstatica

<<Abstract>> Obstaculo

nchos

SueloResbaladizo

Escalera

Pared

ParedDestructible
+ puntaje: int = 150
+ destruirse(): void

d

Moghera
puntajeCongelado: int = 1000 # puntajeEmpujado: int = 5000
+ lanzarFuego(): void + esDerrotado():void

Kamakichi
puntajeCongelado: int = 1000 # puntajeEmpujado: int = 5000
+ dispararBombas(): void

Plataforma

PlataformaMovil
+ puntaje: int = 200 +horizontal: boolean
+ mover(): void

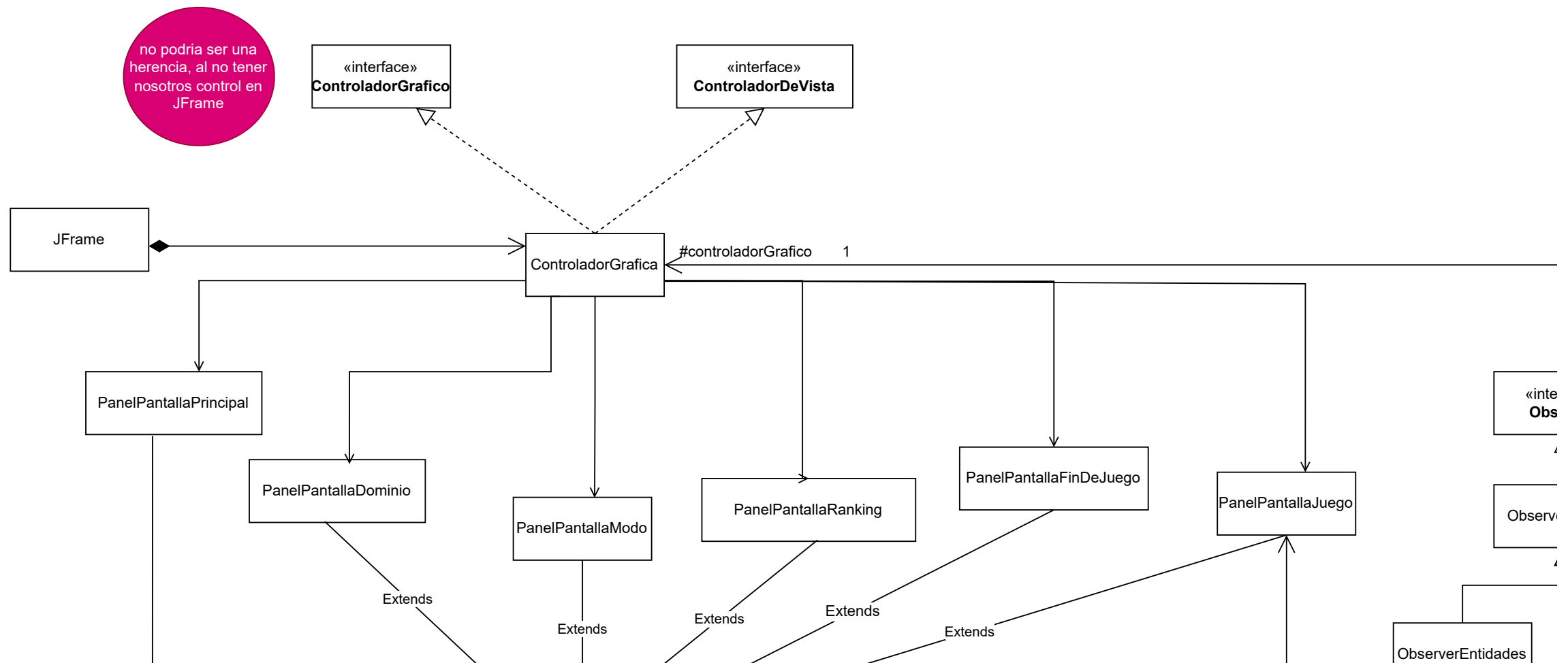
PlataformaQuebradiza
+ puntaje: int = 300

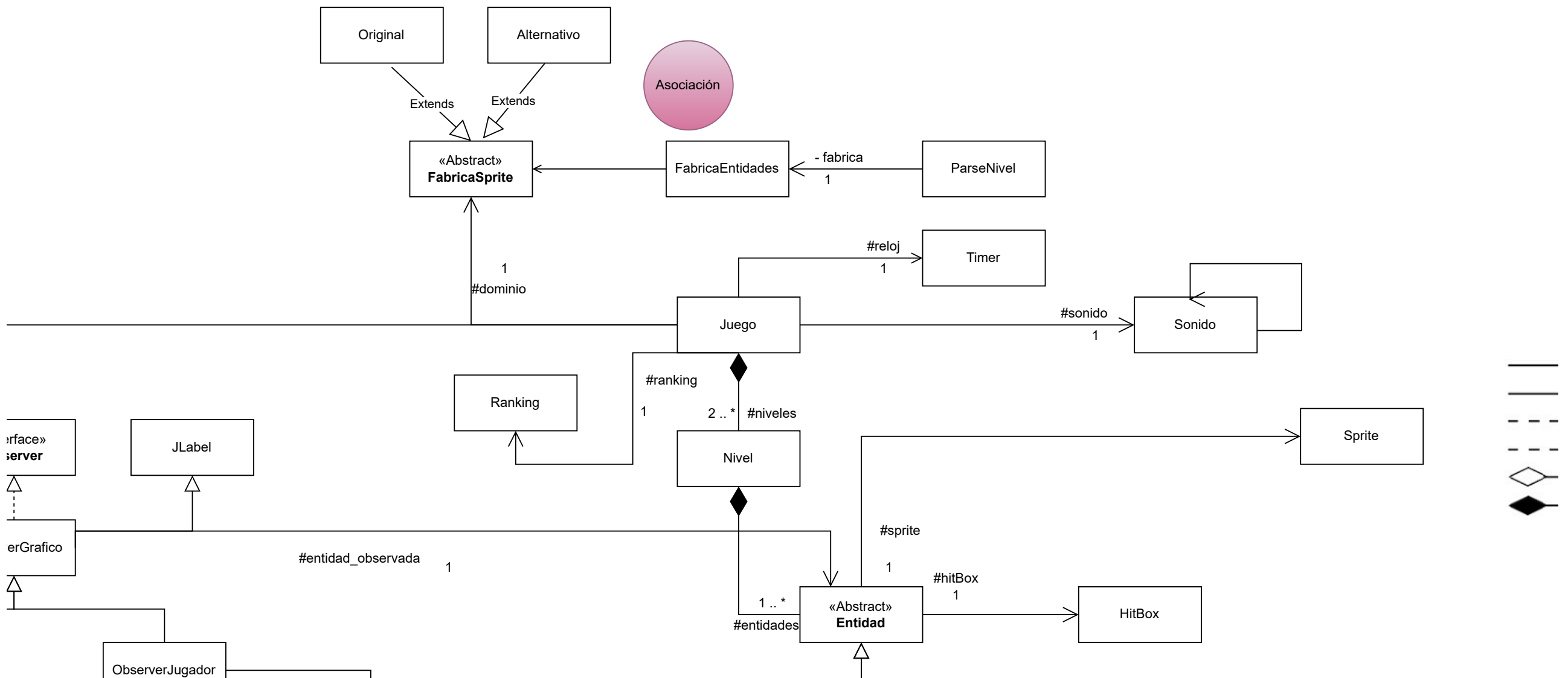
VidaExtra

Fruta
+ puntos: int = 500

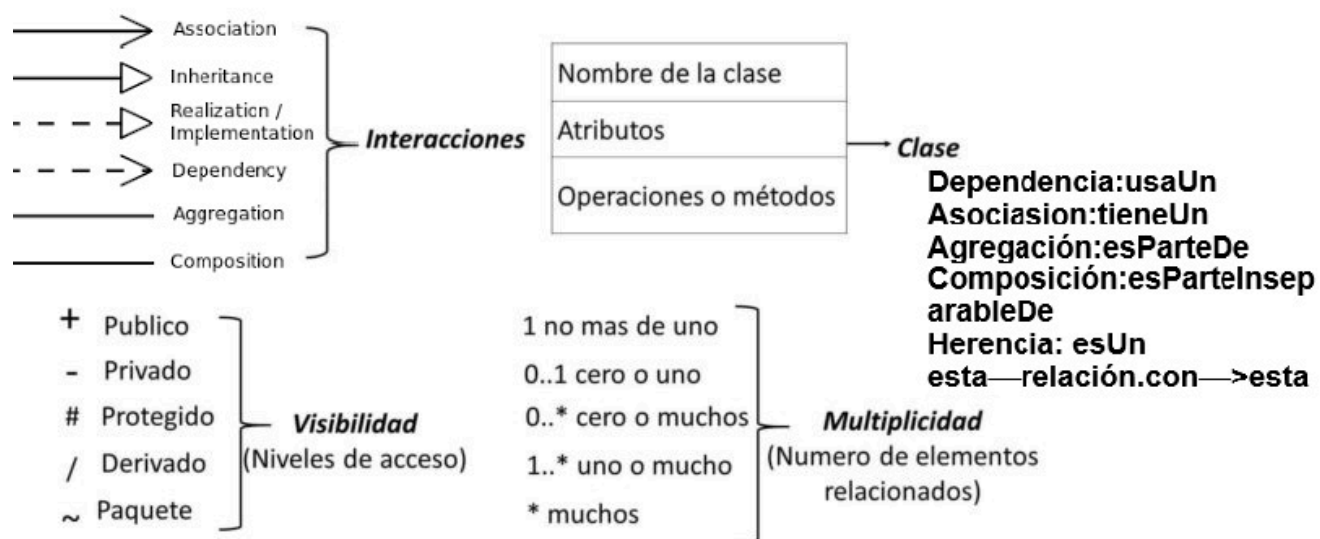
<<Abstract>> PowerUp
puntos: int = 300 # activo: boolean # tiempoDeVida: Timer # tiempoDeEfecto: Timer
+ afectarJugador(): void + colisionado(j: Jugador): void + eliminar(): void

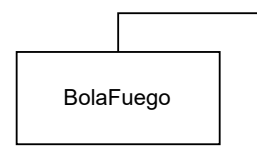
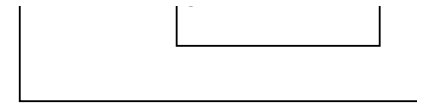
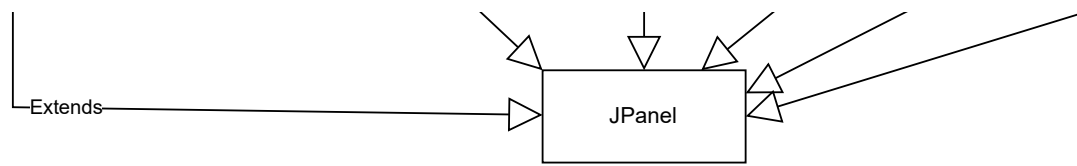
	BotellaAzul	BotellaRoja	BotellaVerde

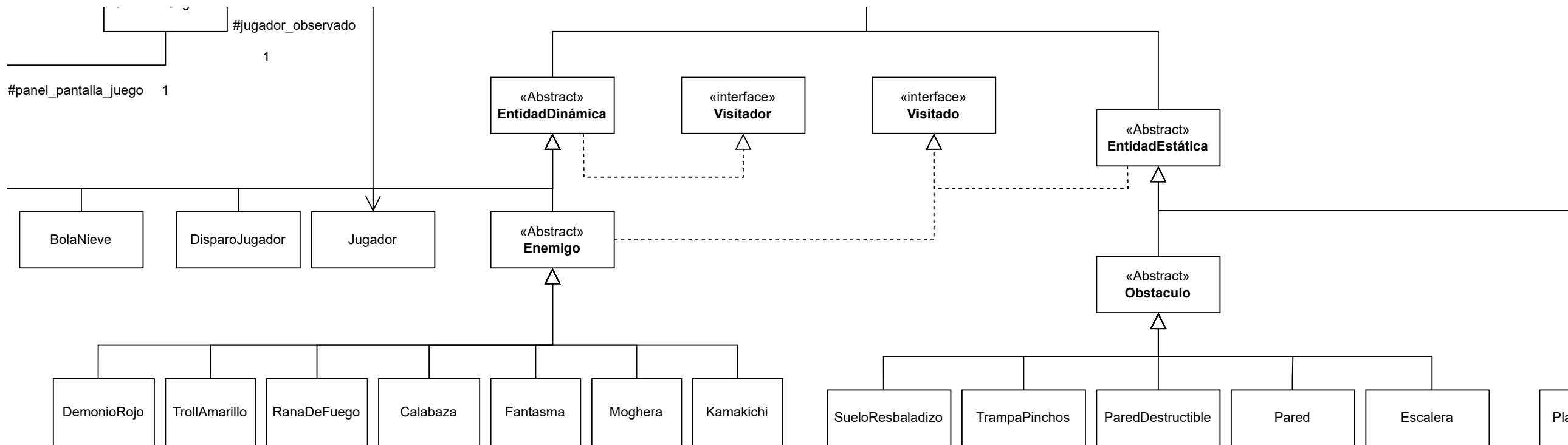


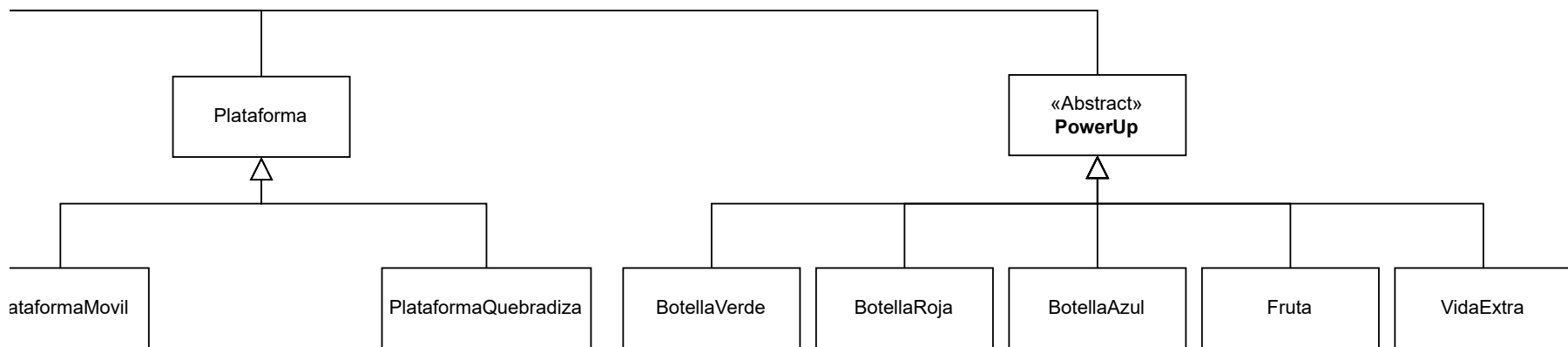


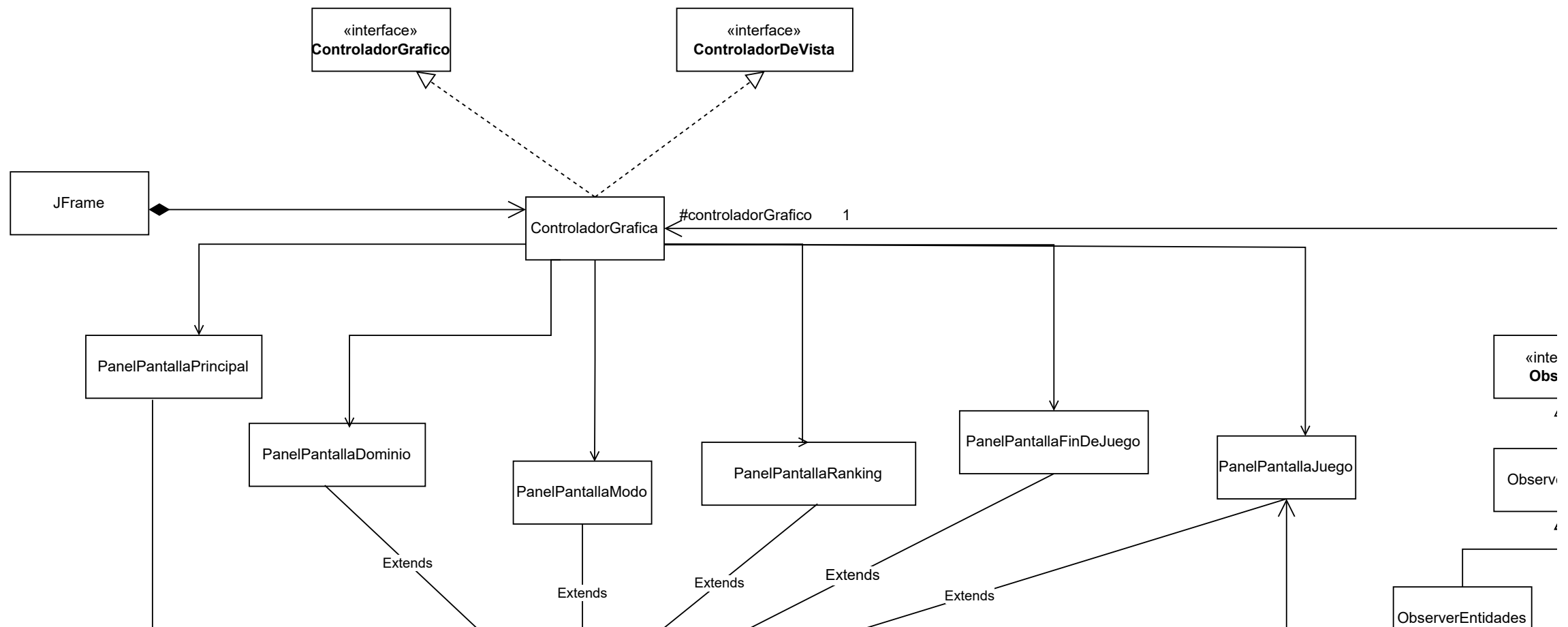
Elementos y símbolos en los diagramas de clases UML

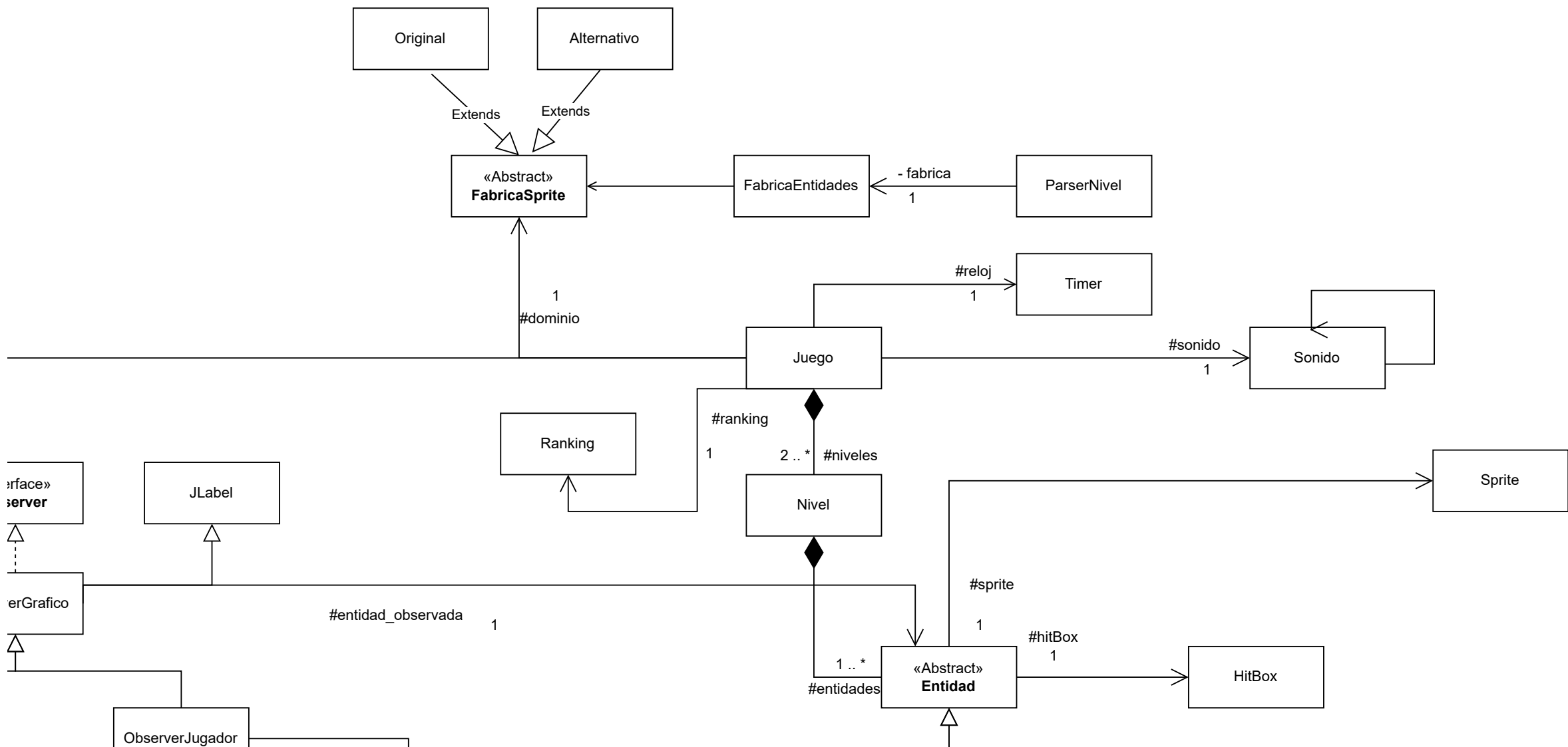


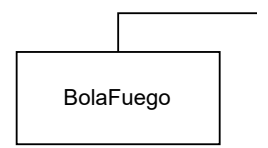
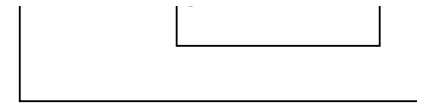
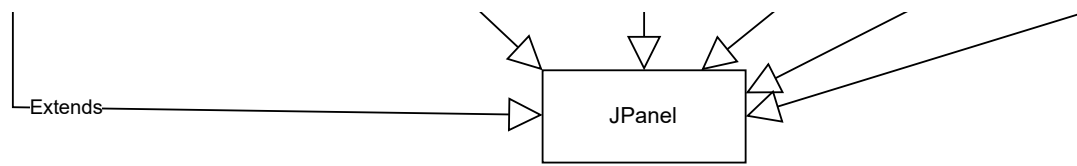


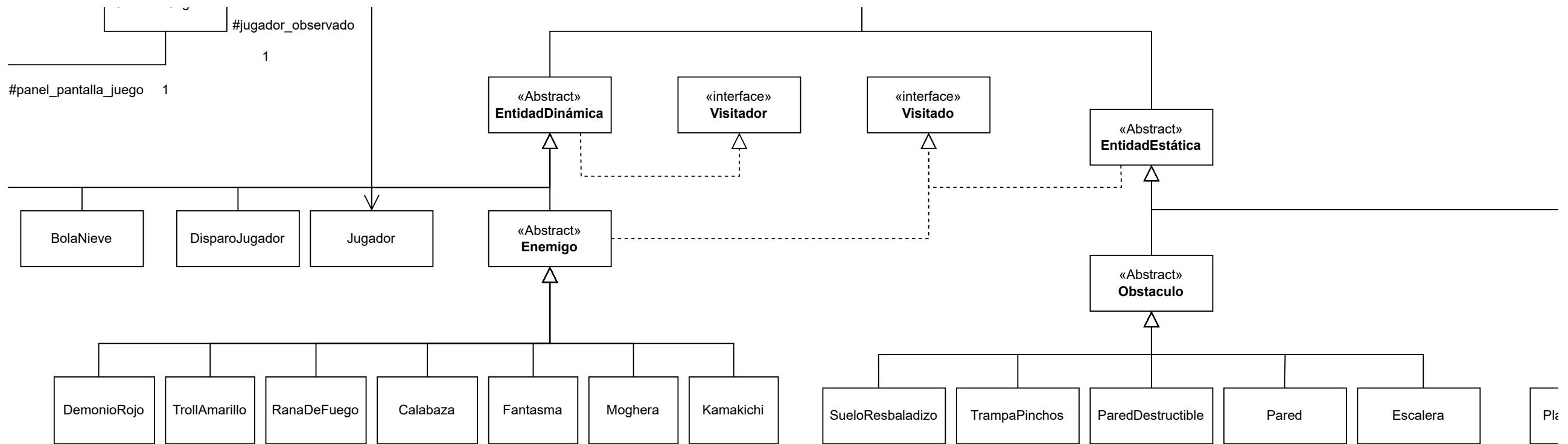


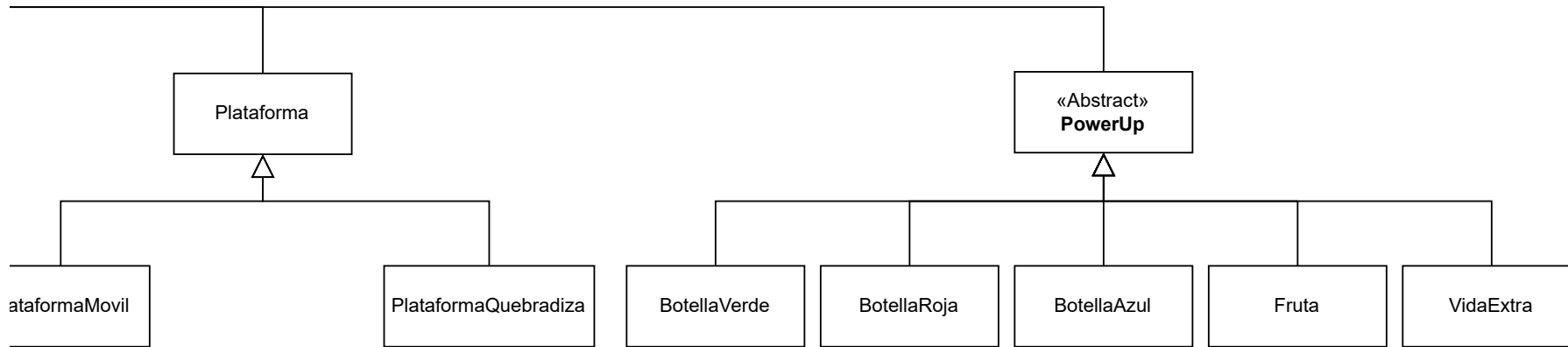


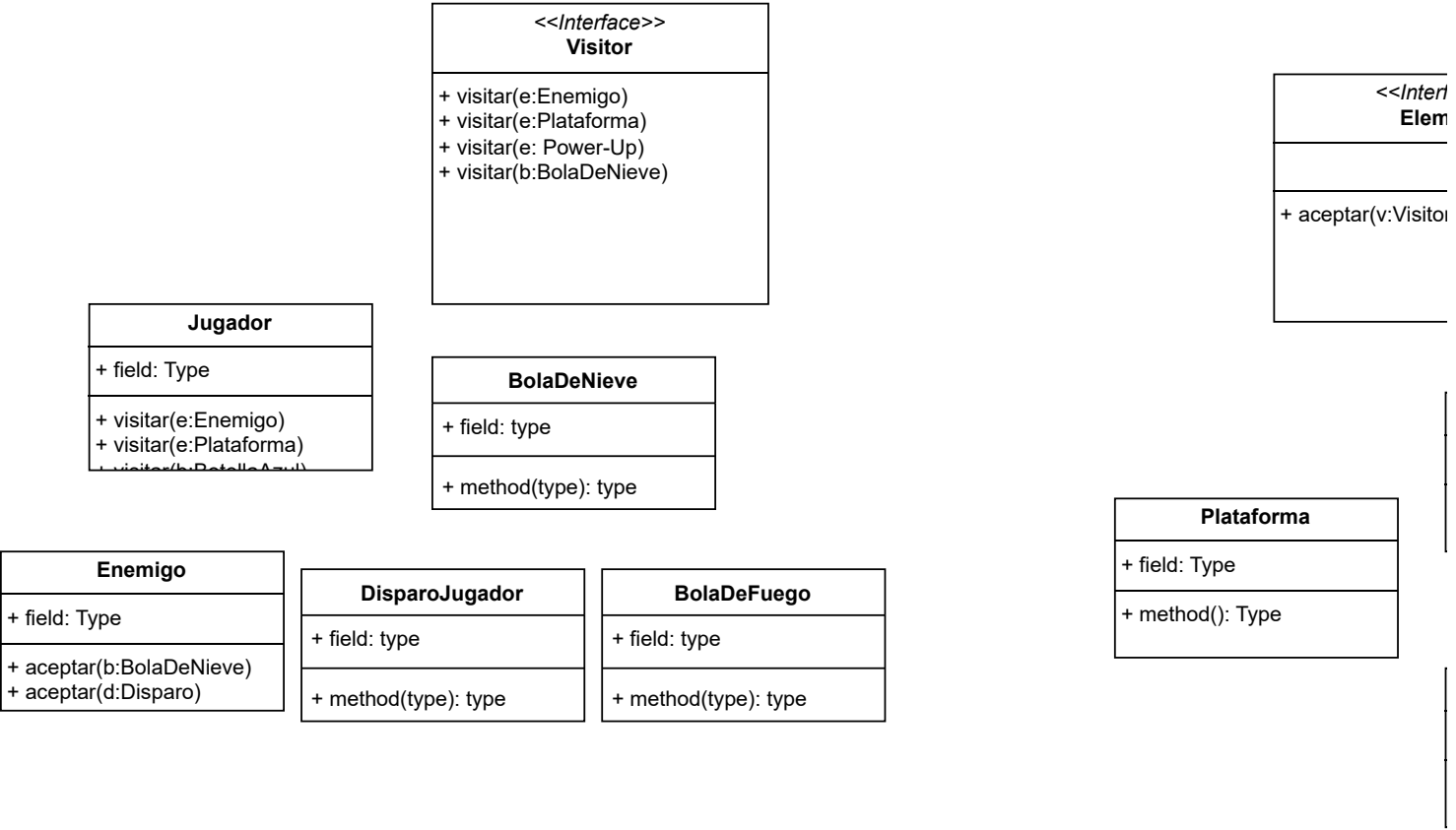












```
class Plataforma extends Entidad {

    void aceptar(Visitor v) {

        // Lógica propia de la Plataforma

        this.reproducirSonidoColision();

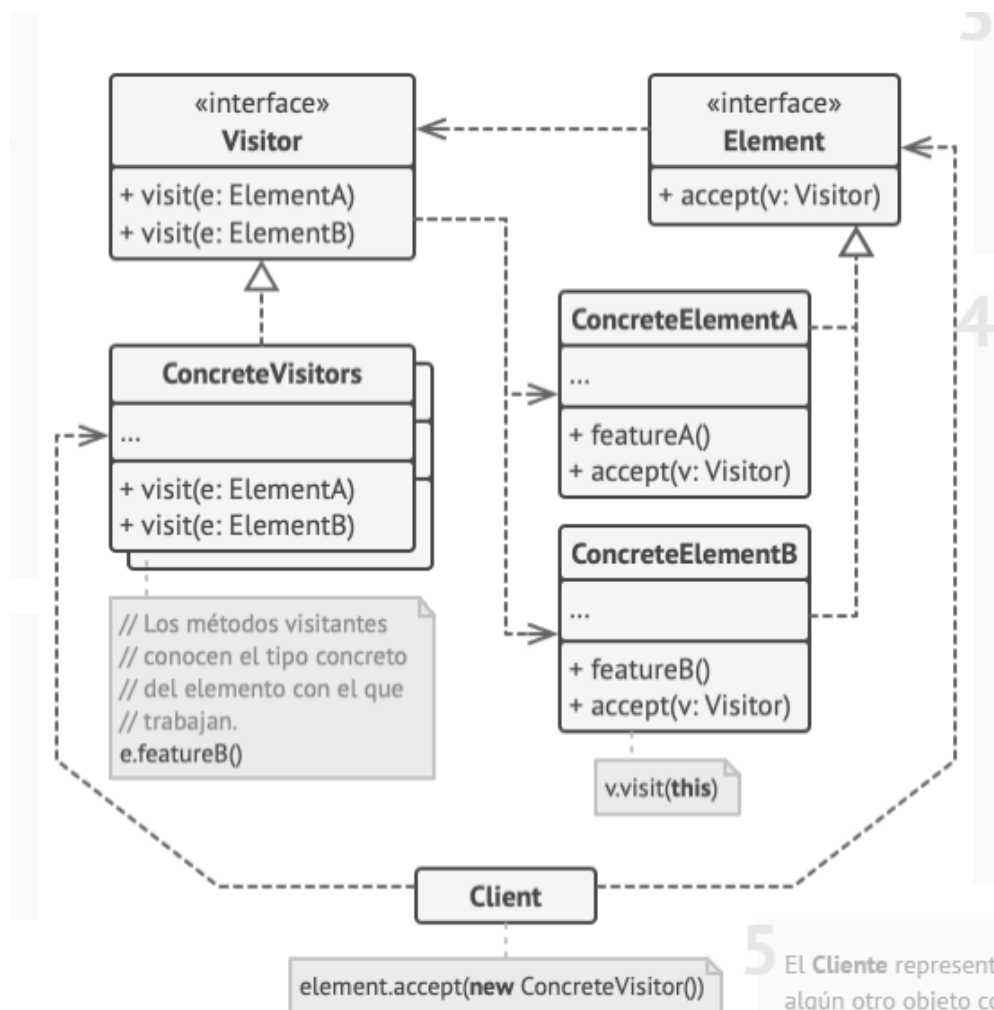
        // Delego en el visitante

        v.visitarPlataforma(this);
```

face>>
rent
r)

Enemigo
+ field: Type
+ aceptar(b:BolaDeNieve)
+ aceptar(d:Disparo)

PowerUp
+ field: Type
+ method(): Type



5 El Cliente representa algún otro objeto con el que se interactúa. A menudo, el Cliente es el que crea los objetos de la interfaz Visitor y los objetos de la interfaz Element.

```

class ExportVisitor implements Visitor is
    method doForCity(City c) { ... }
    method doForIndustry(Industry f) { ... }
    method doForSightSeeing(SightSeeing ss) { ... }
    // ...
  
```

```

// Código cliente
foreach (Node node in graph)
    node.accept(exportVisitor)

// Ciudad
class City is
    method accept(Visitor v) is
        v.doForCity(this)
    // ...

// Industria
class Industry is
    method accept(Visitor v) is
  
```

```
// Lógica adicional si quiero  
this.marcarComoUsada();  
}  
}
```

```
        v.doForIndustry(this)  
    // ...
```

Sonido
- instancia:Sonido - mapeo: Map <String,Clip>
-Sonido() //Builder privado + getInstancia(): Sonido (static) + reproducir(s: String): void - cargarSonido(r:String) :Clip

```

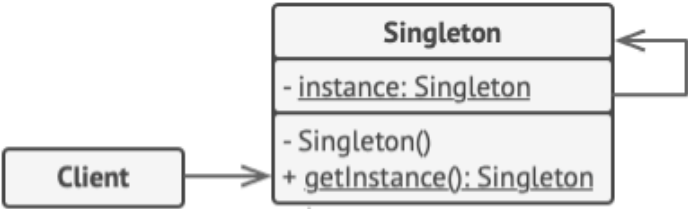
+-----+
| Musica |
+-----+
| - stream : SourceDataLine
| - archivo : String
+-----+
| + reproducir(loop:boolean) : void
| + detener() : void
+-----+

```

```

+-----+
| Sonido | <<Singleton>>
+-----+
| - instance : Sonido [static]
| - sonidos : Map<String, Clip> // efectos precargados
| - musicaActual : Musica // pista en curso
+-----+
| + getInstance() : Sonido [static]
| + reproducirSonido(ruta:String) : void
| + reproducirMusica(ruta:String, loop:boolean) : void
| + detenerMusica() : void
| - cargarSonido(ruta:String) : Clip
+-----+

```



1 La clase **Singleton** declara el método estático `obtenerInstancia` que devuelve la misma instancia de su propia clase.

El constructor del Singleton debe ocultarse del código cliente. La llamada al método `obtenerInstancia` debe ser la única manera de obtener el objeto de Singleton.

```

if (instance == null) {
    // Nota: si estás creando una aplicación
    // que soporte el multihilo, debes
    // colocar un bloqueo de hilo aquí.
    instance = new Singleton()
}
return instance

```