Algoritmos y Programacion III

Primer cuatrimestre de 2021

Trabajo Práctico 2: A.L.T.E.G.O

Alumno	Número de padrón	Email
BALMACEDA, Fernando	105525	fbalmaceda@fi.uba.ar
BOCACCIO, Agustina	106393	abocaccio@fi.uba.ar
CASTRO, Nahuel	106551	ncastro@fi.uba.ar
PINTO, Nicolas	105064	npinto@fi.uba.ar
PRESEDO, Maria del Rosario	106338	mpresedo@fi.uba.ar

Curso: Suarez
Corrector: GOMEZ, Joaquin

2021

Índice

1. Introducción

El presente informe sera utilizado para explicar y especificar lo realizado en el Trabajo Practico 2 de Algoritmos y Programación III. El objetivo fue desarrollar una aplicación, la cual consiste en una versión del juego T.E.G.

El trabajo se realizó de manera grupal aplicando todos los conceptos vistos en el curso, utilizando un lenguaje de tipado estático (Java) con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua.

2. Supuestos

A continuación se especificaran los supuestos que se hicieron a la hora de desarrollar el juego

- Los países se dividirán uniformemente entre los jugadores. En caso de tener una cantidad de jugadores que no sea divisor de 50, los países se repartirán hasta que se acaben, dejando jugadores con un país mas que otros.
- Al inicializar el juego y asignarle un país a un jugador automáticamente se le coloca un ejercito sobre este.
- Debido a que la única forma de diferenciar los países son sus nombres, no se permitirá que haya dos países llamados de la misma manera.
- Debido a que la única forma de diferenciar a los jugadores son sus nombres, no se permitirá que haya dos jugadores con el mismo nombre.
- El jugador debe colocar todas sus fichas en la etapa de colocación para poder comenzar con la siguiente etapa.
- Solo se puede activar una carta en la fase de colocación.
- Cuando se conquista a un país se traslada un ejercito al mismo automáticamente.

3. Modelo de Dominio

El trabajo entregado es una implementación de una versión del juego **T.E.G** (Plan Táctico y Estratégico de Guerra), el cual plantea un conflicto bélico y tiene como fin ir conquistando países para cumplir con el objetivo individual secreto y asignado al azar que posee cada jugador, o el objetivo común, que se basa en apoderarse de 30 países. En el caso de que se cumpla con el objetivo del juego, este finalizará.

El juego se basa en rondas que tienen distintas etapas. Cada ronda comienza con la etapa de colocación, que consta de solo una fase, la fase de colocación. En esta fase cada jugador debe colocar sus fichas. Además, se puede aprovechar para utilizar las cartas (es en la única fase donde se pueden realizar los canjes).
Luego, sigue la etapa de batalla, que consta de dos fases: la fase de ataque y la fase de agrupación. La fase de ataque es para que un país ataque a otro, y una vez determinado el perdedor, se pasa a la fase de agrupación, en donde el jugador puede mover ejércitos (fichas) de un país a otro.

Uno de los principales objetivos del trabajo fue cumplir con lo enseñando en la materia como los pilares de programación orientada a objetos: abstracción, encapsulamiento, herencia y polimorfismo.

- Con la creación de cada clase la idea fue abstraerse de la realidad implementando solo los atributos relevantes del objeto. Un ejemplo es la clase País, el mismo tiene más características que solo un nombre o sus países limítrofes, es decir, puede tener la cantidad de habitantes,

una cámara de diputados, un monumento, entre otras cosas. Pero la decisión fue usar lo importante para el trabajo: nombre, un dueño, la cantidad de ejércitos y sus países limítrofes.

- La ventaja de usar encapsulamiento en el trabajo es que, si se quiere cambiar algo en una clase, se puede hacer sin afectar al que esta usandola. En donde más se puede ver reflejado este pilar es en las clases de etapas. Hay dos etapas y cada una de ellas tiene su/s fase/s. Cada fase conoce cual es la fase que le sigue, al igual que cada etapa conoce la etapa que le sigue. Por lo tanto, el juego solo debe encargarse de decirle a las etapas que pasen de etapa.
- La herencia fue utilizada en varias ocasiones. Por ejemplo, hay tres lectores encargados de distintas lecturas: paises, cartas y objetivos. Dichos lectores leen archivos en formato JSON, y para hacerlo, extienden su lector correspondiente (p.ej. el lector de objetivos JSON extiende el lector de objetivos) y de esta manera, puede reutilizar el comportamiento de lectura.
- Por último, el polimorfismo nos permitió dejar encapsuladas las etapas y fases. El juego tiene una etapa (la cual desconoce) y como las mismas entienden los mismos mensajes, juego solo se encarga de utilizar sus métodos. Lo mismo ocurre con las etapas y sus fases.

Respecto a la interfaz gráfica, se aplico **MVC** (modelo-vista-controlador). La idea detrás de MVC es que cada sección del código tiene un propósito, y esos propósitos son diferentes.Parte del código contiene los datos de la aplicación (modelo), parte del código hace que la aplicación se vea bien (vistas) y parte del código controla cómo funciona la aplicación (controladores).

- **Modelo**: contiene mecanismos para acceder a la información y también para actualizar su estado.
- **Vistas**: contienen el código que produce la visualización de las interfaces de usuario. En las vistas nada más tenemos el código que nos permite mostrar la salida.
- **Controladores**: contiene el código que responde a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar un movimiento, hacer un canje, etc.

4. Excepciones

Las excepciones utilizadas en el trabajo práctico se pueden encontrar en la carpeta de excepciones del modelo. Las mismas, con su respectivo comportamiento son:

- **AtaqueAPaisPropioException**
  Esta excepción es creada para evitar que un jugador pueda atacar a un país que le pertenece. Para ello, luego de chequear que el país atacante le pertenece, verificamos dentro de la batalla que el país a atacar no sea suyo, si lo es se levantara la excepción
- **AtaqueConCantidadInvalidaException**
  El objetivo de esta excepción es evitar que el jugador pueda atacar con mas ejercitos de los que tiene o sin dejar ningún ejercito en el pais atacante. Se chequea en la creacion de una nueva batalla
- **CartaYaActivadaException**
  Esta excepción se lanza cuando el estado de una carta es .ªctivadoz se quiere activar nuevamente.
- **FichasInsuficientesException**
  Se lanza cuando el jugador no tiene la cantidad de fichas suficientes que quiere colocar.
- **MovimientoConCantidadInvalidaException**
  Esta excepción sirve para cuando el usuario quiere mover una cantidad de ejercitos de un pais de origen a otro, pero el pais de origen no tiene esa cantidad de fichas.

- **NoSePudoLeerExcepcion**
  Se lanza en el caso de que haya un error al momento de leer un archivo, ya sea de cartas, paises u objetivos.
- **NoSePuedeActivarCartaEnLaBatallaException**
  Se utiliza cuando se intenta activar una carta cuando la etapa es la de batalla y no la de colocación.
- **NoSePuedeCanjearEnEtapaBatallaException**
  Se utiliza cuando se intenta canjear una carta cuando la etapa es la de batalla y no la de colocación.
- **PaisNoTePerteneceException**
  La excepcion se levantara si el jugador intenta realizar una jugada con un pais que no es suyo, ya sea atacar a otro pais, mover ejercitos de ese pais a uno limitrofe o agregar ejercitos a ese pais.
- **PaisNoExisteException**
  Se utiliza para verificar que el pais exista dentro del juego cada vez que se quiera realizar una accion con este.
- **PaisNoLimitrofeException**
  Se lanza si el pais a atacar o al que se van a trasladar los ejércitos no es limítrofe al país atacante o que mueve el ejercito.
- **QuedanFichasPorColocarException**
  Se lanza cuando se quiere pasar a la fase que le sigue a la de colocación pero aún quedan fichas por colocar.
- **SimbolosInvalidosException**
  Se lanza cuando las cartas no son canjeables entre ellas. Esto, por ejemplo, puede pasar porque las cartas no son iguales o no son comodines.

5. Patrones de diseño

5.1. State

Este patrón de diseño de comportamiento permite a un objeto alterar su comportamiento cuando su estado interno cambia. Por ejemplo, se utilizó al momento de implementar la lógica de fases o el estado del jugador. En el segundo ejemplo, el jugador tiene un estado que va cambiando en base a las circunstancias del juego.

5.2. Observer

Este patrón de diseño de comportamiento permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando. En el trabajo, Juego (quien forma parte del modelo) es el objeto que tiene un estado relevante para otros objetos, es por eso que implementa la interfaz Observable. El mismo tiene dos observadores (ObservadorJuego y ObservadorPaisesSeleccionados, ambas clases son vistas), por lo tanto, cuando le sucede un evento importante a Juego, recorre sus suscriptores y llama al método de notificación específico de sus objetos.

5.3. Strategy

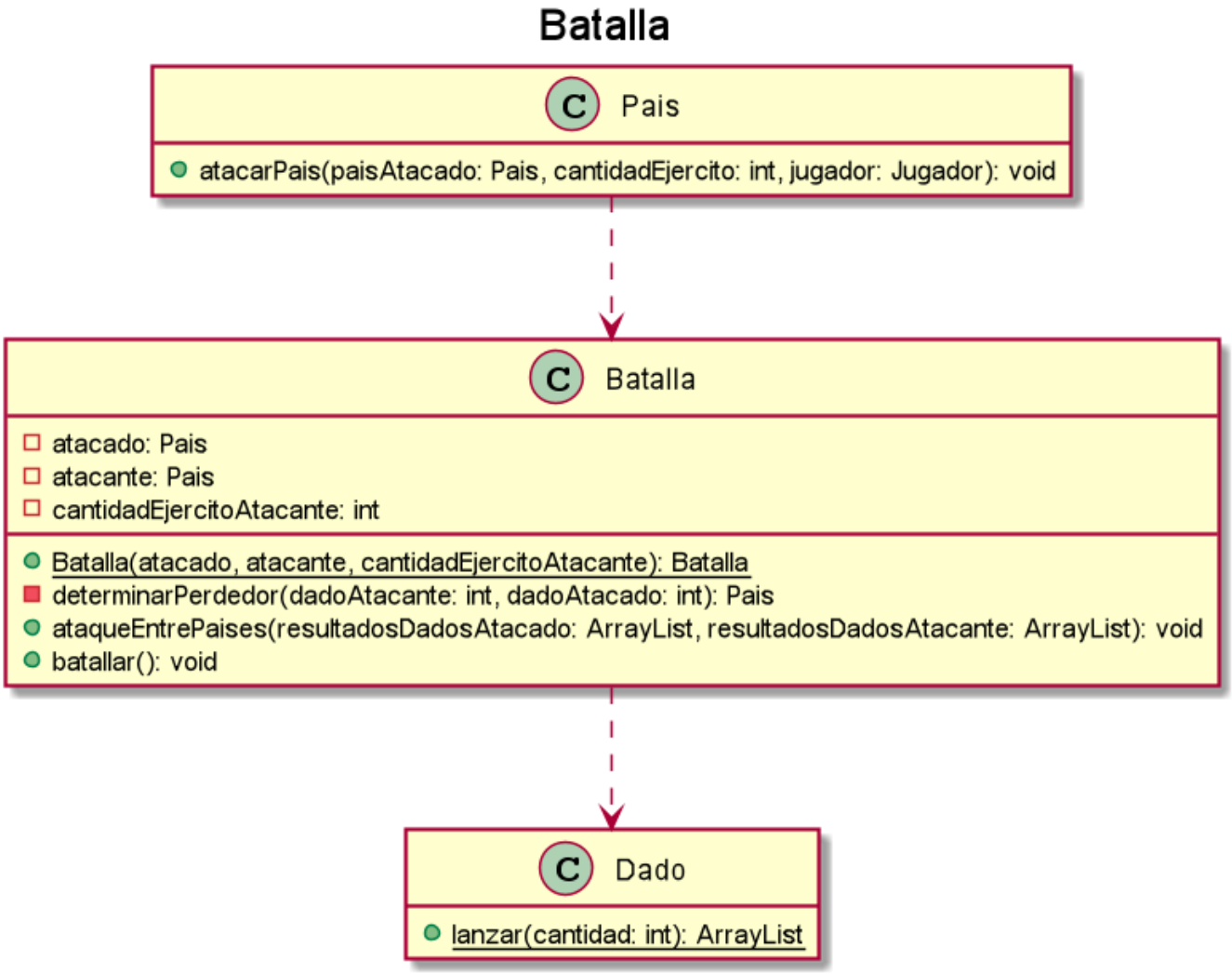
Este patrón de diseño de comportamiento permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables. Hay varios ejemplos de la implementación de este patrón, uno de ellos es que un jugador tiene un canje. Este canje puede

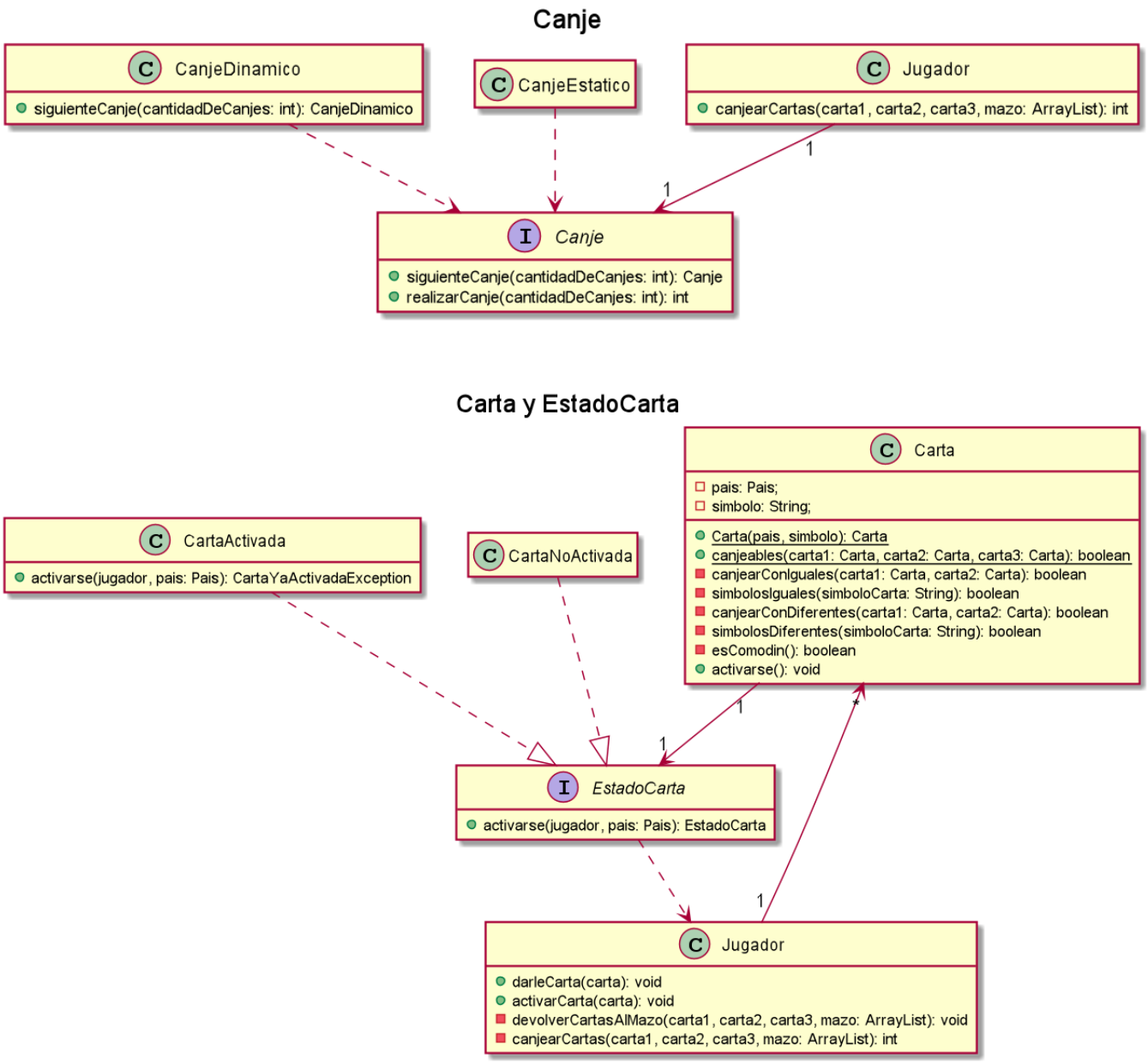
ser estático o dinámico. Pero el jugador utiliza su canje sin importar de que tipo sea. Los tipos de canje se podrían considerar las estrategias y el jugador el contexto.
Otro ejemplo es el estado del jugador. Todos los estados entienden el siguiente mensaje:

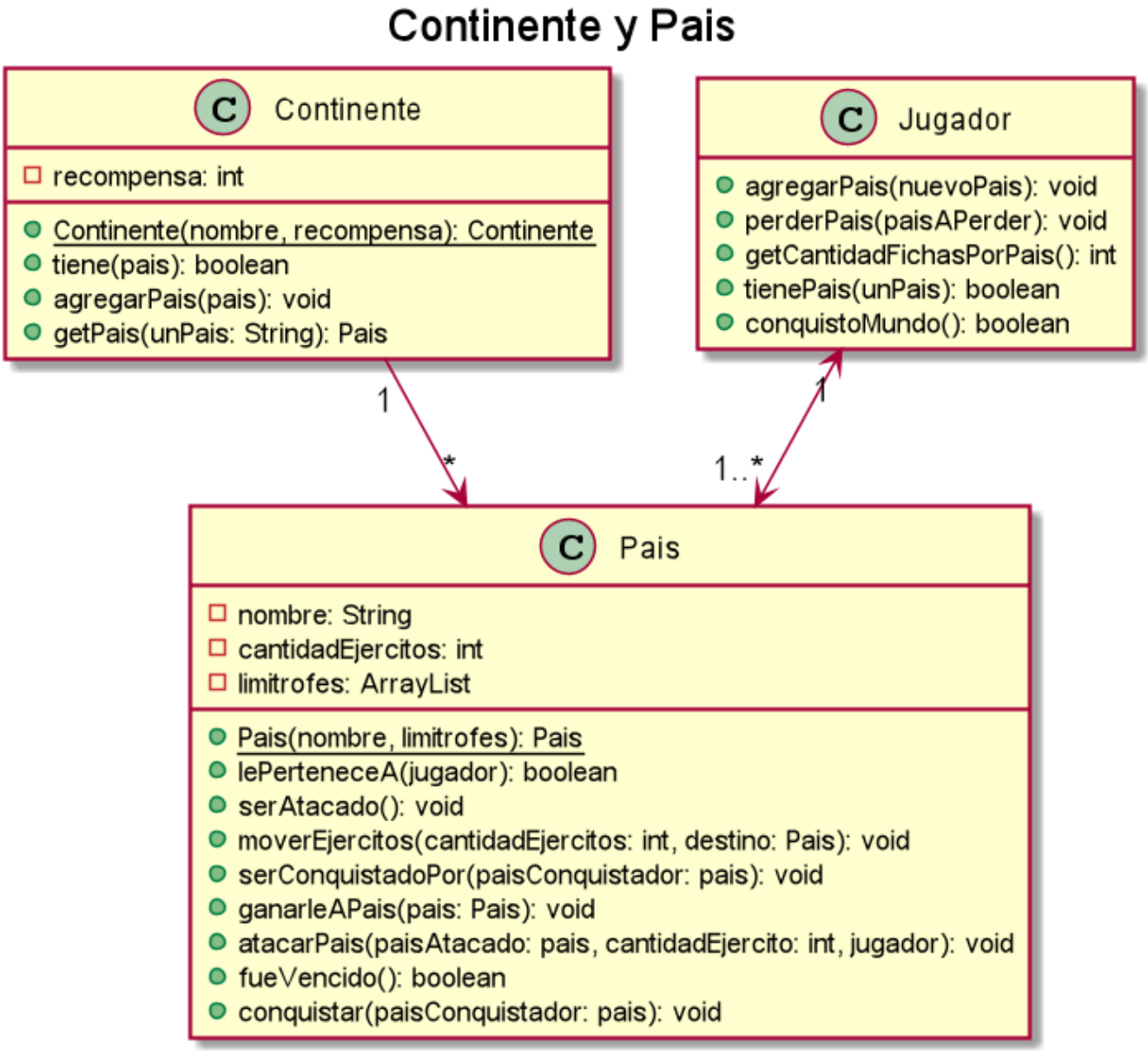
```
derrotadoPor(Jugador unJugador)
```

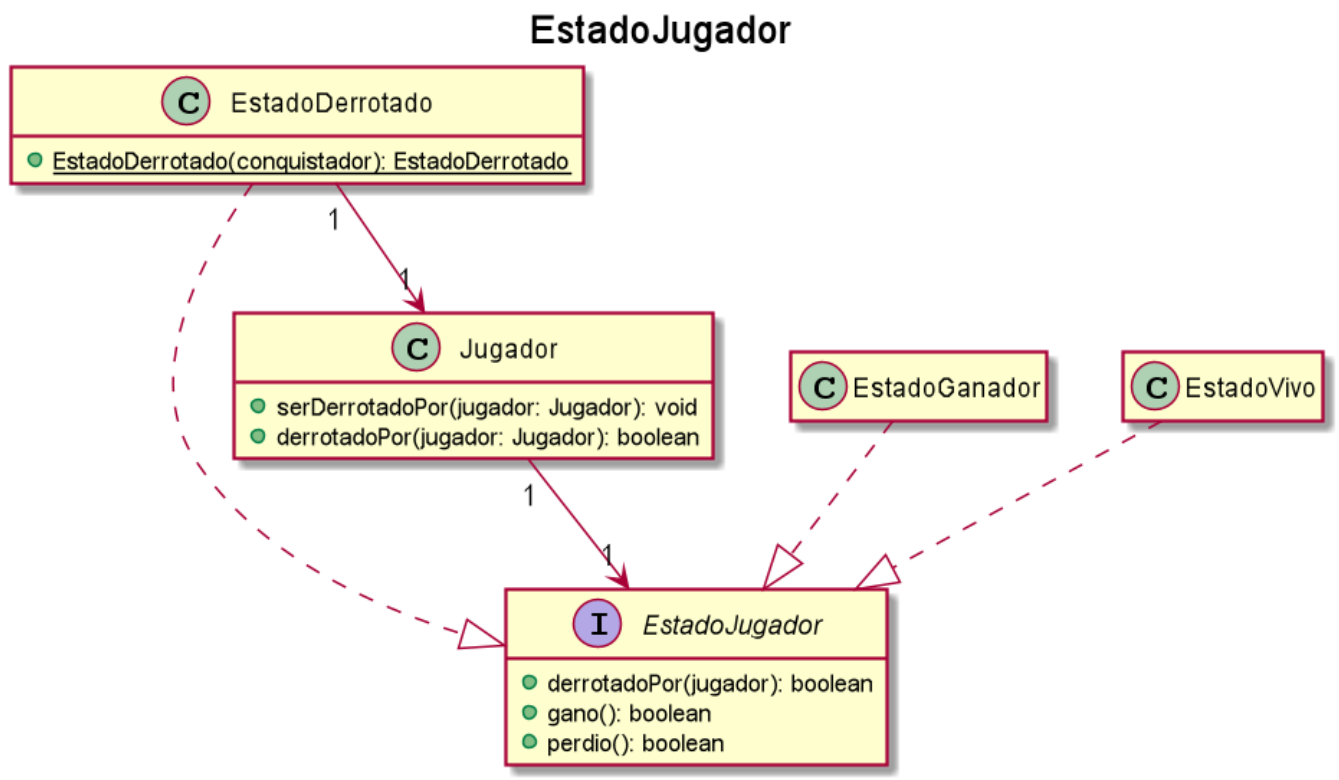
esto, vuelve a los estados intercambiables entre sí. En este caso, la interfaz estrategia sería EstadoJugador y las estrategias concretas serían EstadoVivo, EstadoGanador y EstadoPerdedor.

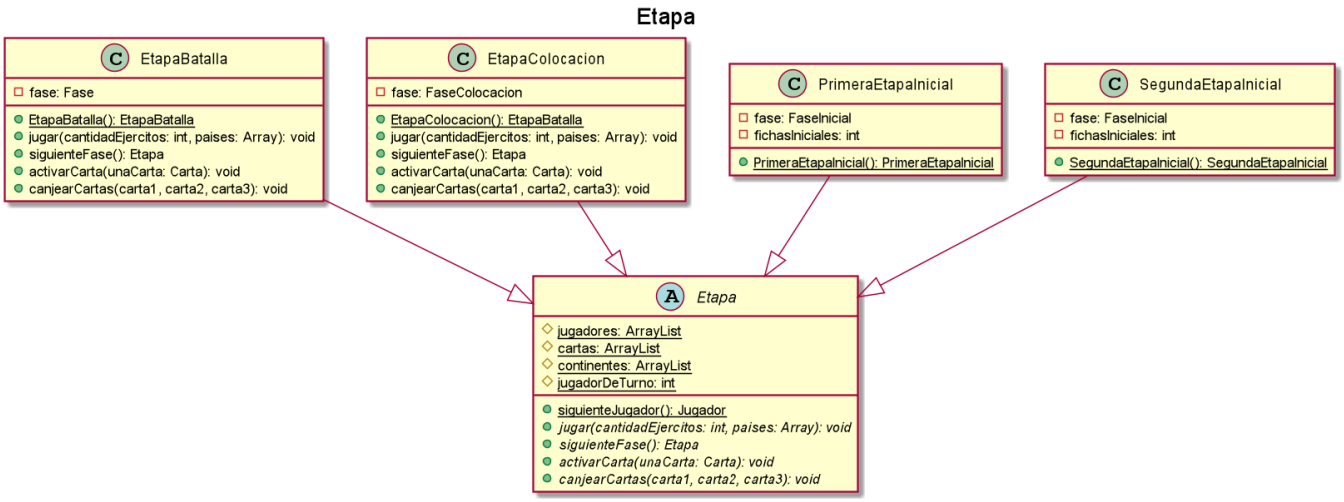
6. Diagramas de clase

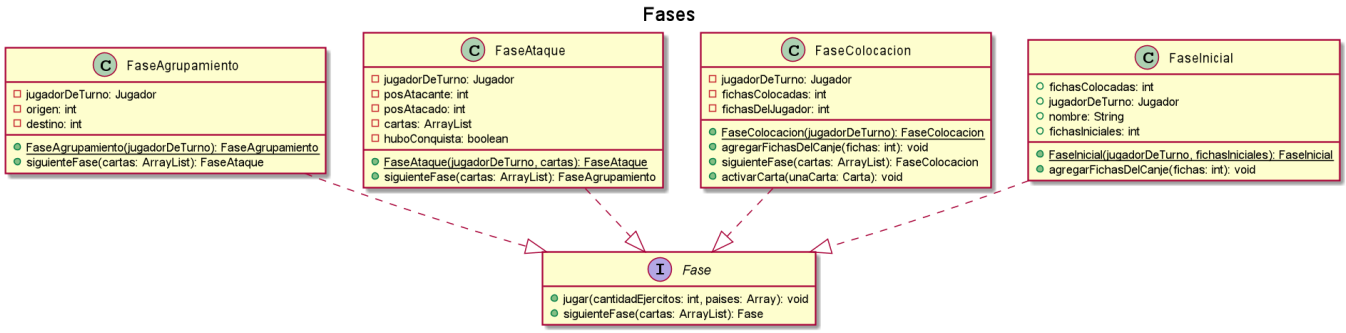


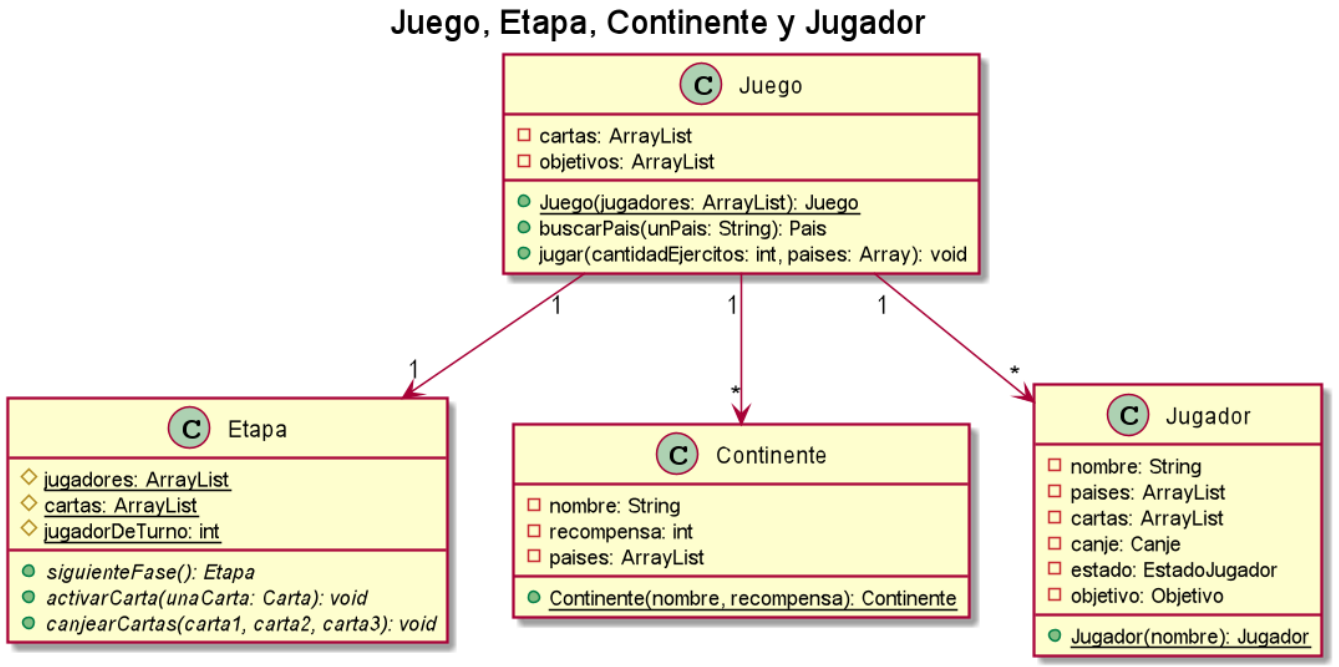

Canje
CanjeDinamico
siguienteCanje(cantidadDeCanjes: int): CanjeDinamico
CanjeEstatico
Jugador
canjearCartas(carta1, carta2, carta3, mazo: ArrayList): int
1
1
Canje
siguienteCanje(cantidadDeCanjes: int): Canje
realizarCanje(cantidadDeCanjes: int): int
Carta y EstadoCarta
Carta
pais: Pais;
simbolo: String;
Carta(pais, simbolo): Carta
canjeables(carta1: Carta, carta2: Carta, carta3: Carta): boolean
canjearConIguales(carta1: Carta, carta2: Carta): boolean
simbolosIguales(simboloCarta: String): boolean
canjearConDiferentes(carta1: Carta, carta2: Carta): boolean
simbolosDiferentes(simboloCarta: String): boolean
esComodin(): boolean
activarse(): void
CartaActivada
activarse(jugador, pais: Pais): CartaYaActivadaException
CartaNoActivada
1
1
*
EstadoCarta
activarse(jugador, pais: Pais): EstadoCarta
1
Jugador
darleCarta(carta): void
activarCarta(carta): void
devolverCartasAlMazo(carta1, carta2, carta3, mazo: ArrayList): void
canjearCartas(carta1, carta2, carta3, mazo: ArrayList): int

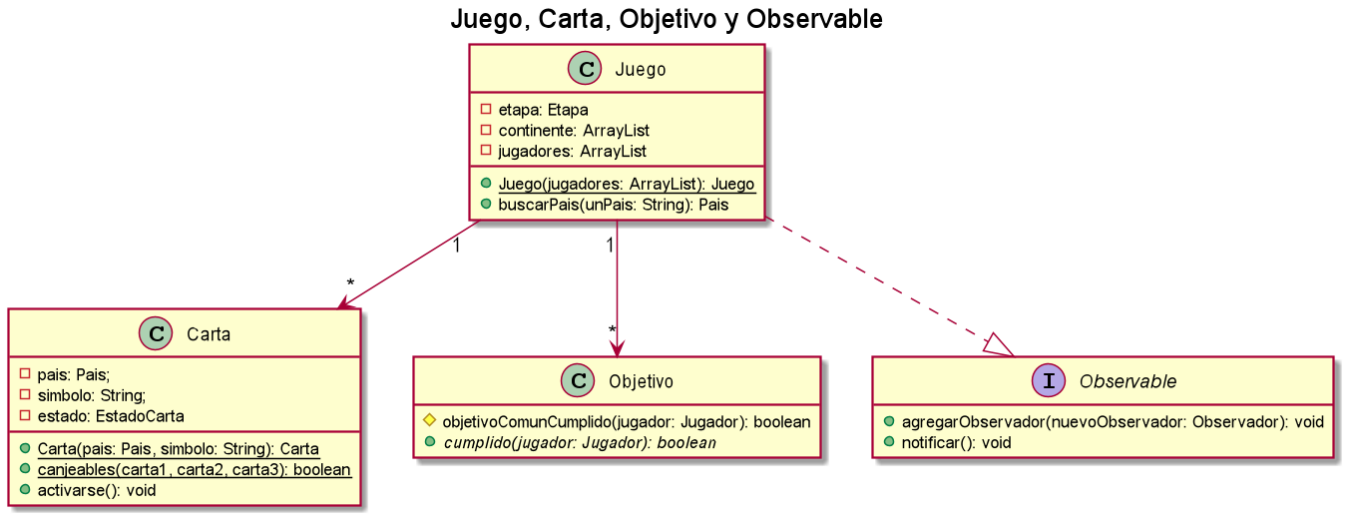

Continente y Pais
Continente
recompensa: int
Continente(nombre, recompensa): Continente
tiene(pais): boolean
agregarPais(pais): void
getPais(unPais: String): Pais
Jugador
agregarPais(nuevoPais): void
perderPais(paisAPerder): void
getCantidadFichasPorPais(): int
tienePais(unPais): boolean
conquistoMundo(): boolean
1
*
1..*
1
Pais
nombre: String
cantidadEjercitos: int
limitrofes: ArrayList
Pais(nombre, limitrofes): Pais
lePerteneceA(jugador): boolean
serAtacado(): void
moverEjercitos(cantidadEjercitos: int, destino: Pais): void
serConquistadoPor(paisConquistador: pais): void
ganarleAPais(pais: Pais): void
atacarPais(paisAtacado: pais, cantidadEjercito: int, jugador): void
fueVencido(): boolean
conquistar(paisConquistador: pais): void

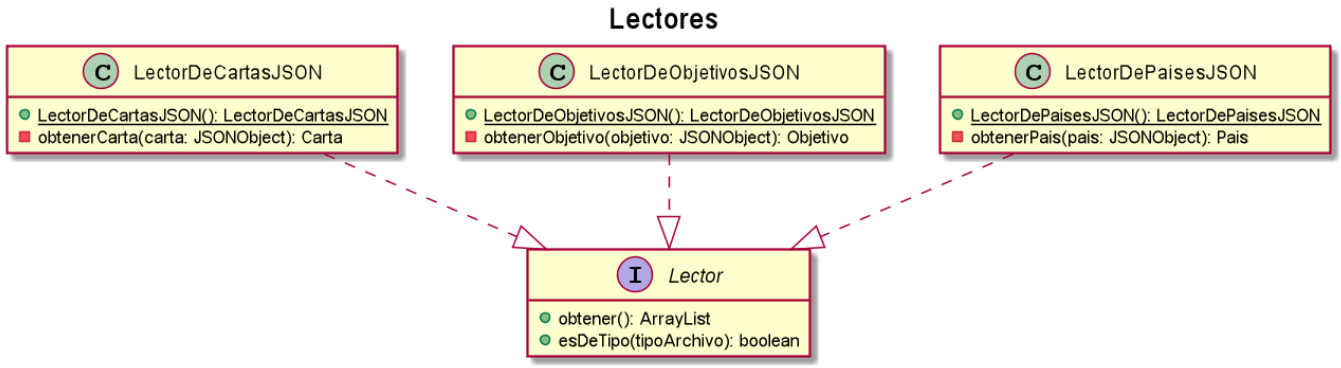

EstadoJugador
EstadoDerrotado
EstadoDerrotado(conquistador): EstadoDerrotado
1
1
Jugador
serDerrotadoPor(jugador: Jugador): void
derrotadoPor(jugador: Jugador): boolean
1
1
EstadoGanador
EstadoVivo
EstadoJugador
derrotadoPor(jugador): boolean
gano(): boolean
perdio(): boolean

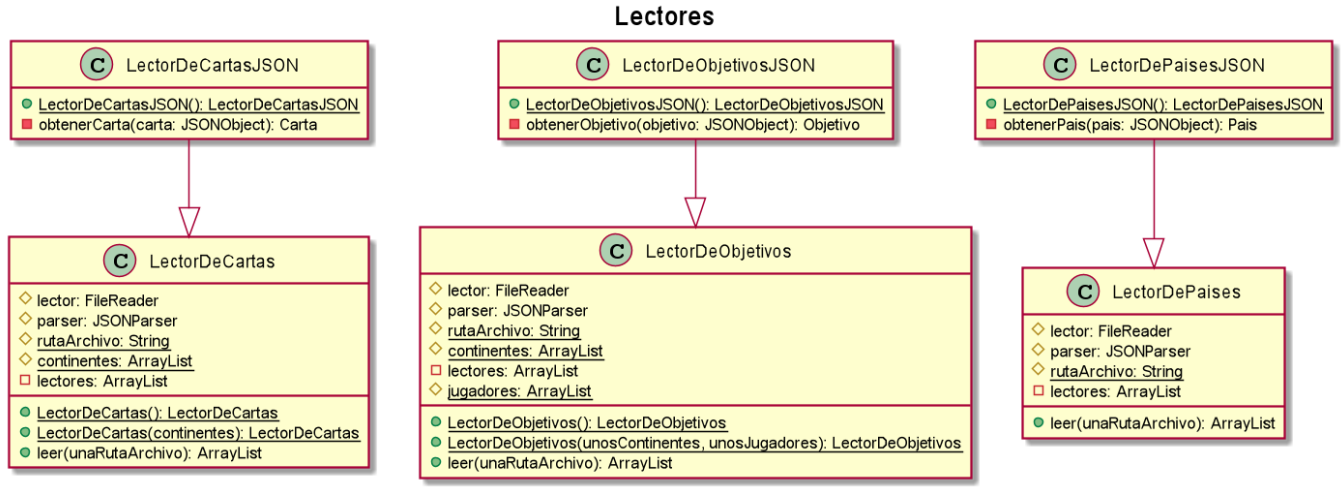

Etapa
EtapaBatalla
fase: Fase
EtapaBatalla(): EtapaBatalla
jugar(cantidadEjercitos: int, paises: Array): void
siguienteFase(): Etapa
activarCarta(unaCarta: Carta): void
canjearCartas(carta1, carta2, carta3): void
EtapaColocacion
fase: FaseColocacion
EtapaColocacion(): EtapaBatalla
jugar(cantidadEjercitos: int, paises: Array): void
siguienteFase(): Etapa
activarCarta(unaCarta: Carta): void
canjearCartas(carta1, carta2, carta3): void
PrimeraEtapaInicial
fase: FaseInicial
fichasIniciales: int
PrimeraEtapaInicial(): PrimeraEtapaInicial
SegundaEtapaInicial
fase: FaseInicial
fichasIniciales: int
SegundaEtapaInicial(): SegundaEtapaInicial
Etapa
jugadores: ArrayList
cartas: ArrayList
continentes: ArrayList
jugadorDeTurno: int
siguienteJugador(): Jugador
jugar(cantidadEjercitos: int, paises: Array): void
siguienteFase(): Etapa
activarCarta(unaCarta: Carta): void
canjearCartas(carta1, carta2, carta3): void

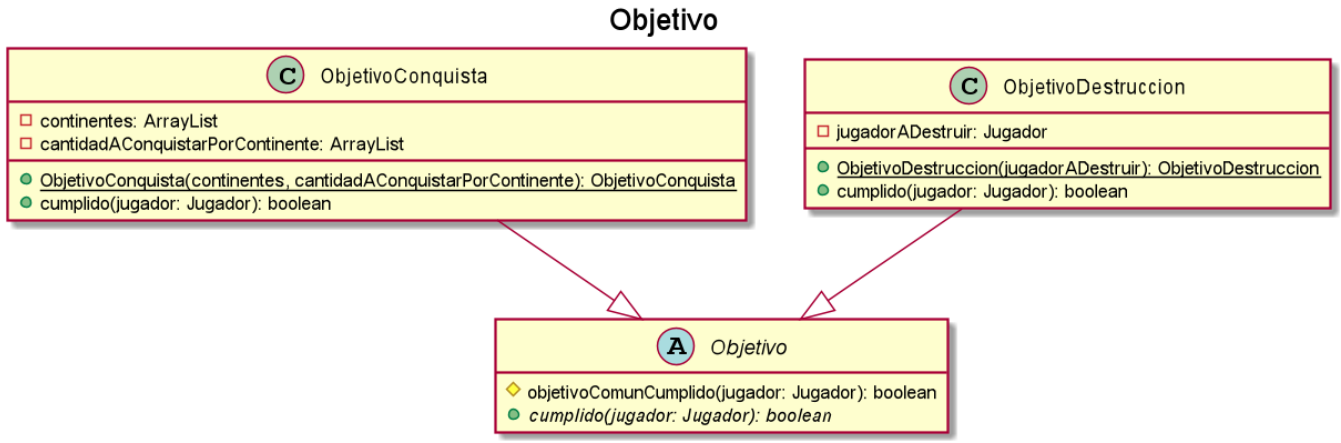

Fases
FaseAgrupamiento
jugadorDeTurno: Jugador
origen: int
destino: int
FaseAgrupamiento(jugadorDeTurno): FaseAgrupamiento
siguienteFase(cartas: ArrayList): FaseAtaque
FaseAtaque
jugadorDeTurno: Jugador
posAtacante: int
posAtacado: int
cartas: ArrayList
huboConquista: boolean
FaseAtaque(jugadorDeTurno, cartas): FaseAtaque
siguienteFase(cartas: ArrayList): FaseAgrupamiento
FaseColocacion
jugadorDeTurno: Jugador
fichasColocadas: int
fichasDelJugador: int
FaseColocacion(jugadorDeTurno): FaseColocacion
agregarFichasDelCanje(fichas: int): void
siguienteFase(cartas: ArrayList): FaseColocacion
activarCarta(unaCarta: Carta): void
FaseInicial
fichasColocadas: int
jugadorDeTurno: Jugador
nombre: String
fichasIniciales: int
FaseInicial(jugadorDeTurno, fichasIniciales): FaseInicial
agregarFichasDelCanje(fichas: int): void
Fase
jugar(cantidadEjercitos: int, paises: Array): void
siguienteFase(cartas: ArrayList): Fase

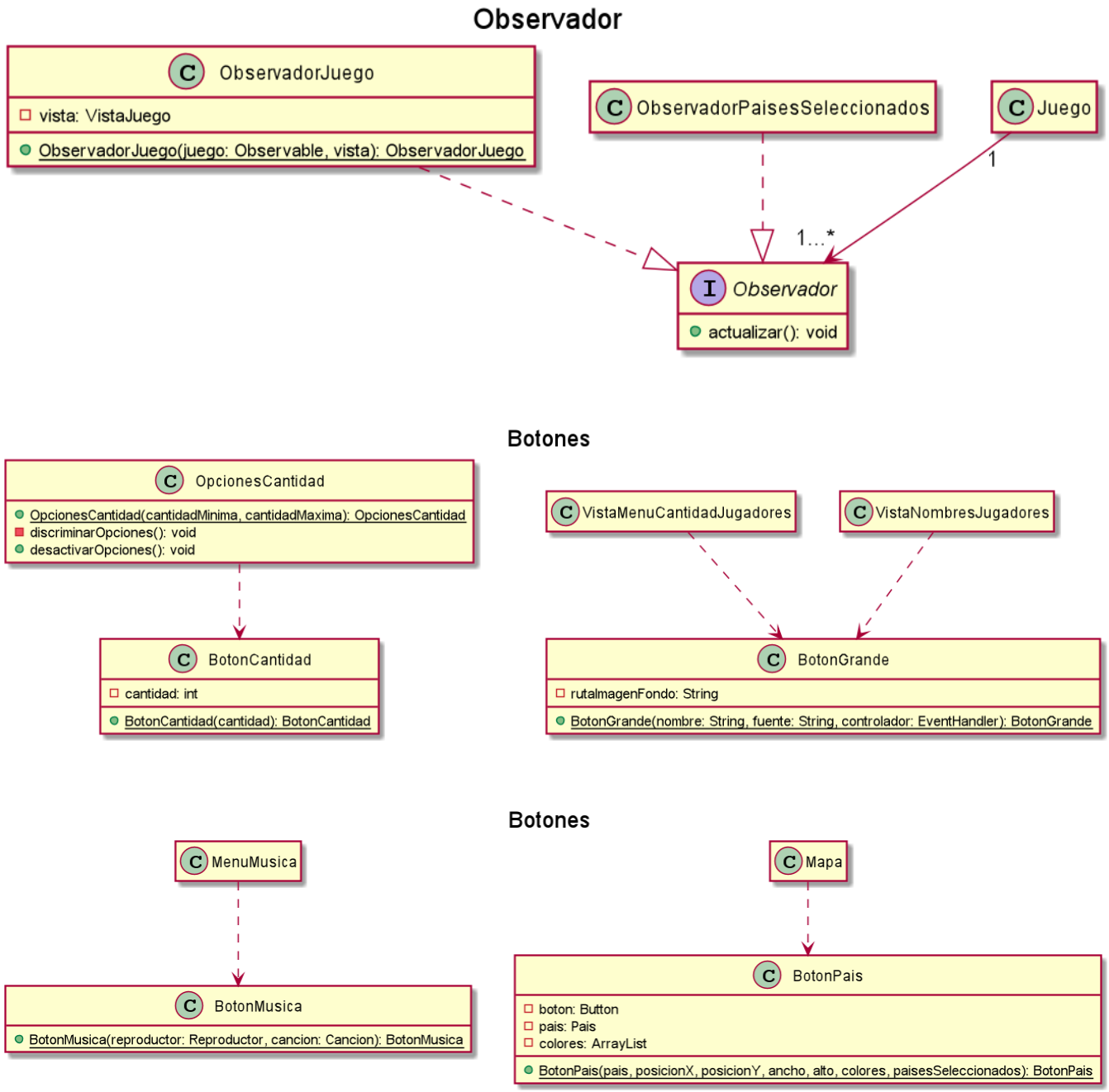

Juego, Etapa, Continente y Jugador
Juego
cartas: ArrayList
objetivos: ArrayList
Juego(jugadores: ArrayList): Juego
buscarPais(unPais: String): Pais
jugar(cantidadEjercitos: int, paises: Array): void
1
1
1
1
*
*
Etapa
jugadores: ArrayList
cartas: ArrayList
jugadorDeTurno: int
siguienteFase(): Etapa
activarCarta(unaCarta: Carta): void
canjearCartas(carta1, carta2, carta3): void
Continente
nombre: String
recompensa: int
paises: ArrayList
Continente(nombre, recompensa): Continente
Jugador
nombre: String
paises: ArrayList
cartas: ArrayList
canje: Canje
estado: EstadoJugador
objetivo: Objetivo
Jugador(nombre): Jugador


Juego, Carta, Objetivo y Observable
Juego
etapa: Etapa
continente: ArrayList
jugadores: ArrayList
Juego(jugadores: ArrayList): Juego
buscarPais(unPais: String): Pais
1
1
*
*
Carta
pais: Pais;
simbolo: String;
estado: EstadoCarta
Carta(pais: Pais, simbolo: String): Carta
canjeables(carta1, carta2, carta3): boolean
activarse(): void
Objetivo
objetivoComunCumplido(jugador: Jugador): boolean
cumplido(jugador: Jugador): boolean
Observable
agregarObservador(nuevoObservador: Observador): void
notificar(): void


Lectores
LectorDeCartasJSON
LectorDeCartasJSON(): LectorDeCartasJSON
obtenerCarta(carta: JSONObject): Carta
LectorDeObjetivosJSON
LectorDeObjetivosJSON(): LectorDeObjetivosJSON
obtenerObjetivo(objetivo: JSONObject): Objetivo
LectorDePaisesJSON
LectorDePaisesJSON(): LectorDePaisesJSON
obtenerPais(pais: JSONObject): Pais
Lector
obtener(): ArrayList
esDeTipo(tipoArchivo): boolean

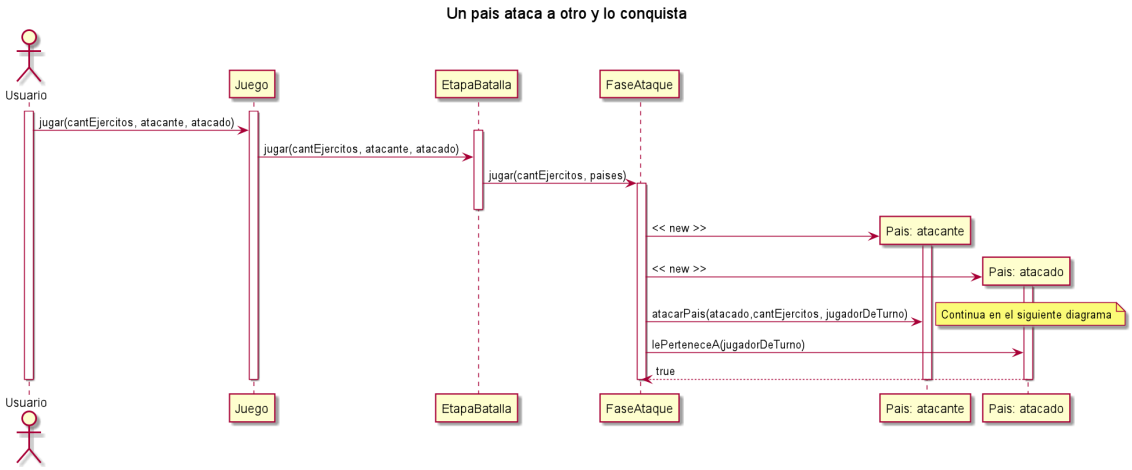

Lectores
LectorDeCartasJSON
LectorDeCartasJSON(): LectorDeCartasJSON
obtenerCarta(carta: JSONObject): Carta
LectorDeObjetivosJSON
LectorDeObjetivosJSON(): LectorDeObjetivosJSON
obtenerObjetivo(objetivo: JSONObject): Objetivo
LectorDePaisesJSON
LectorDePaisesJSON(): LectorDePaisesJSON
obtenerPais(pais: JSONObject): Pais
LectorDeCartas
lector: FileReader
parser: JSONParser
rutaArchivo: String
continentes: ArrayList
lectores: ArrayList
LectorDeCartas(): LectorDeCartas
LectorDeCartas(continentes): LectorDeCartas
leer(unaRutaArchivo): ArrayList
LectorDeObjetivos
lector: FileReader
parser: JSONParser
rutaArchivo: String
continentes: ArrayList
lectores: ArrayList
jugadores: ArrayList
LectorDeObjetivos(): LectorDeObjetivos
LectorDeObjetivos(unosContinentes, unosJugadores): LectorDeObjetivos
leer(unaRutaArchivo): ArrayList
LectorDePaises
lector: FileReader
parser: JSONParser
rutaArchivo: String
lectores: ArrayList
leer(unaRutaArchivo): ArrayList

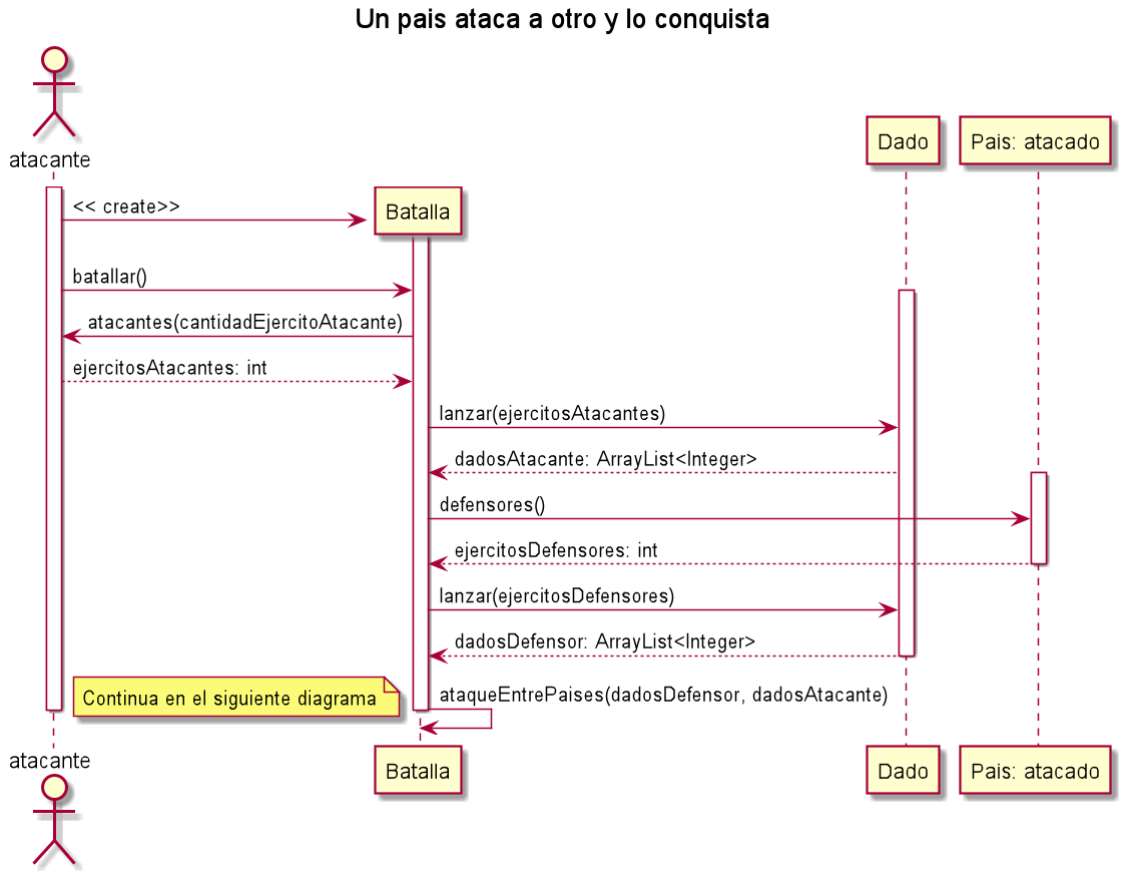

Objetivo
ObjetivoConquista
continentes: ArrayList
cantidadAConquistarPorContinente: ArrayList
ObjetivoConquista(continentes, cantidadAConquistarPorContinente): ObjetivoConquista
cumplido(jugador: Jugador): boolean
ObjetivoDestruccion
jugadorADestruir: Jugador
ObjetivoDestruccion(jugadorADestruir): ObjetivoDestruccion
cumplido(jugador: Jugador): boolean
Objetivo
objetivoComunCumplido(jugador: Jugador): boolean
cumplido(jugador: Jugador): boolean

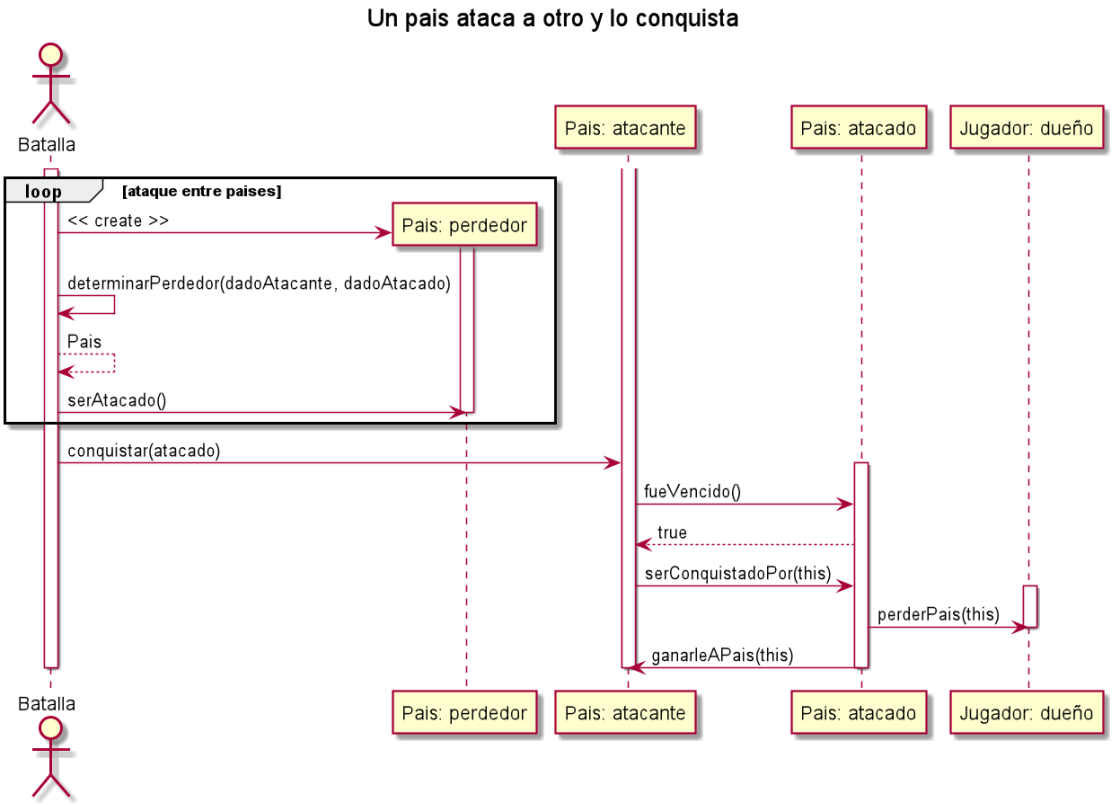


7. Diagramas de secuencia

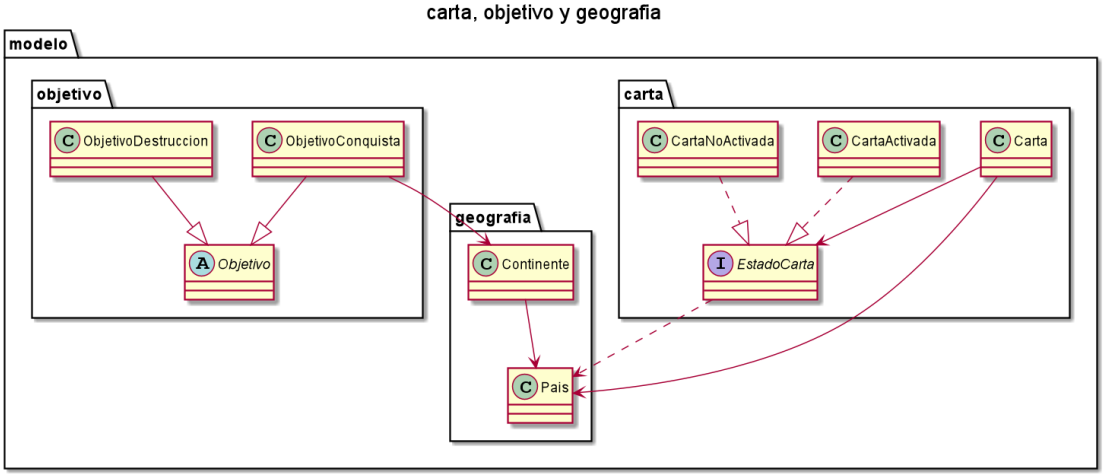
Diagrama que muestra como un país es conquistado por otro:

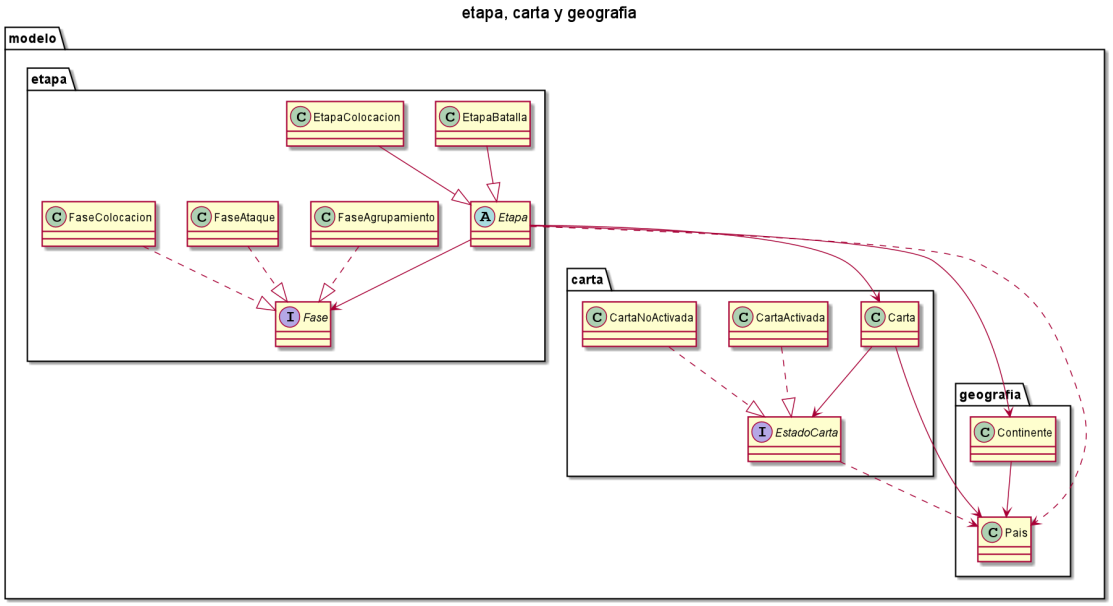

Un pais ataca a otro y lo conquista
Usuario
Juego
EtapaBatalla
FaseAtaque
jugar(cantEjercitos, atacante, atacado)
jugar(cantEjercitos, atacante, atacado)
jugar(cantEjercitos, paises)
<< new >>
Pais: atacante
<< new >>
Pais: atacado
atacarPais(atacado,cantEjercitos, jugadorDeTurno)
Continua en el siguiente diagrama
lePerteneceA(jugadorDeTurno)
true
Usuario
Juego
EtapaBatalla
FaseAtaque
Pais: atacante
Pais: atacado

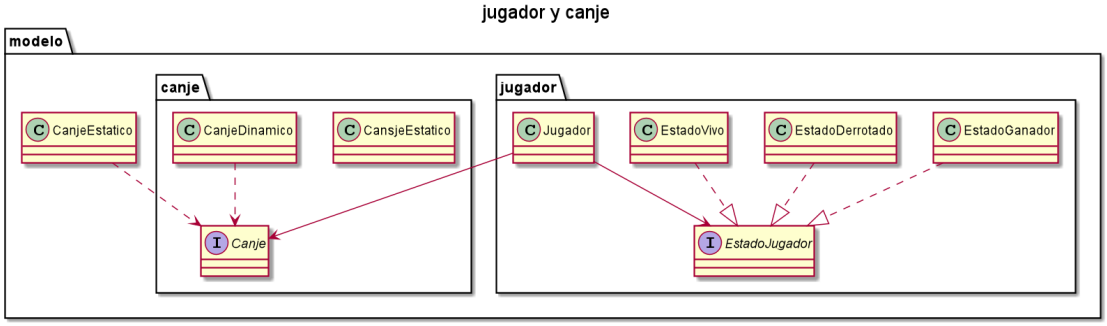

Un pais ataca a otro y lo conquista
atacante
Dado
Pais: atacado
<< create>>
Batalla
batallar()
atacantes(cantidadEjercitoAtacante)
ejercitosAtacantes: int
lanzar(ejercitosAtacantes)
dadosAtacante: ArrayList<Integer>
defensores()
ejercitosDefensores: int
lanzar(ejercitosDefensores)
dadosDefensor: ArrayList<Integer>
Continua en el siguiente diagrama
ataqueEntrePaises(dadosDefensor, dadosAtacante)
atacante
Batalla
Dado
Pais: atacado

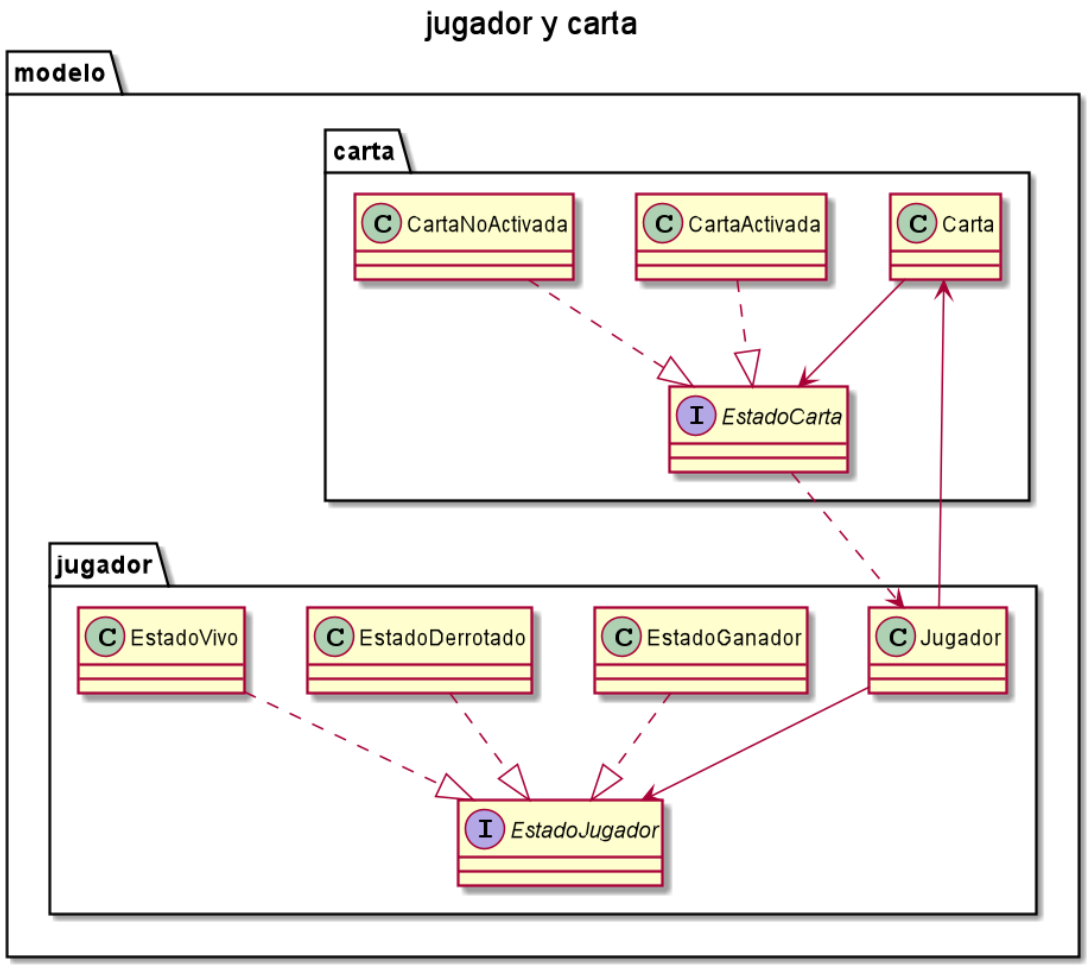


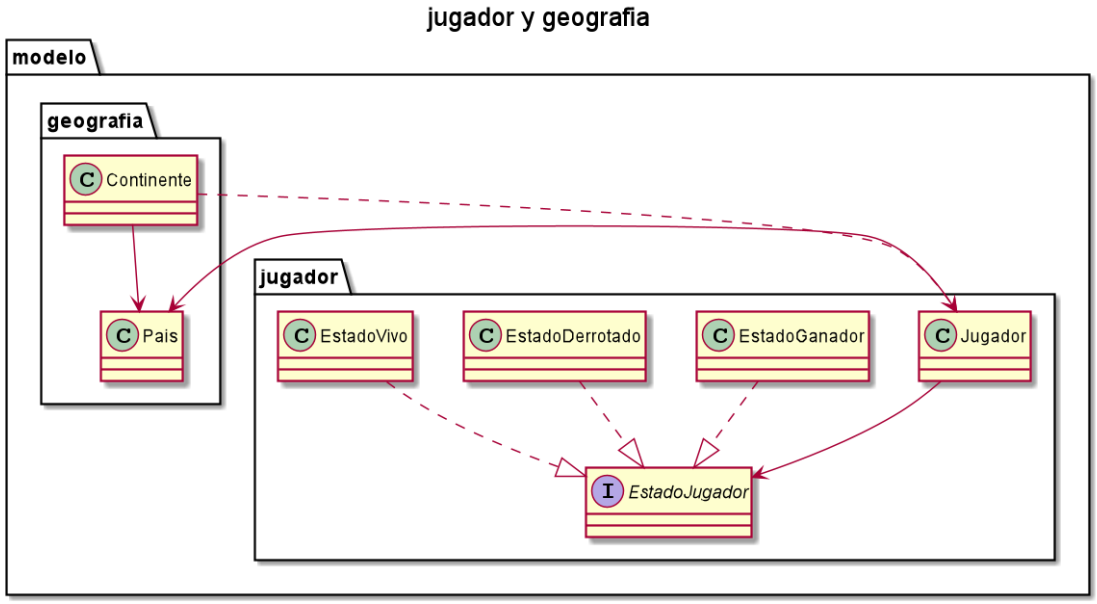
8. Diagramas de paquetes

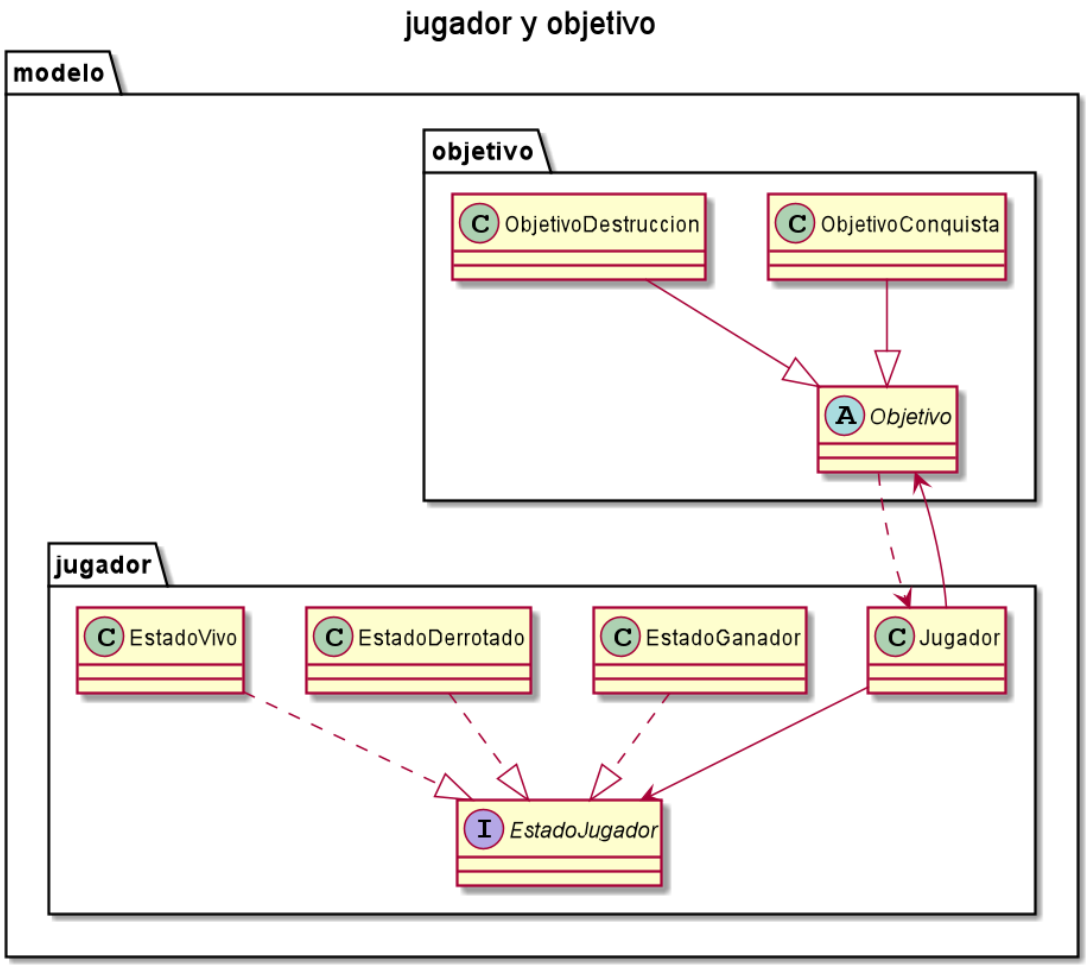


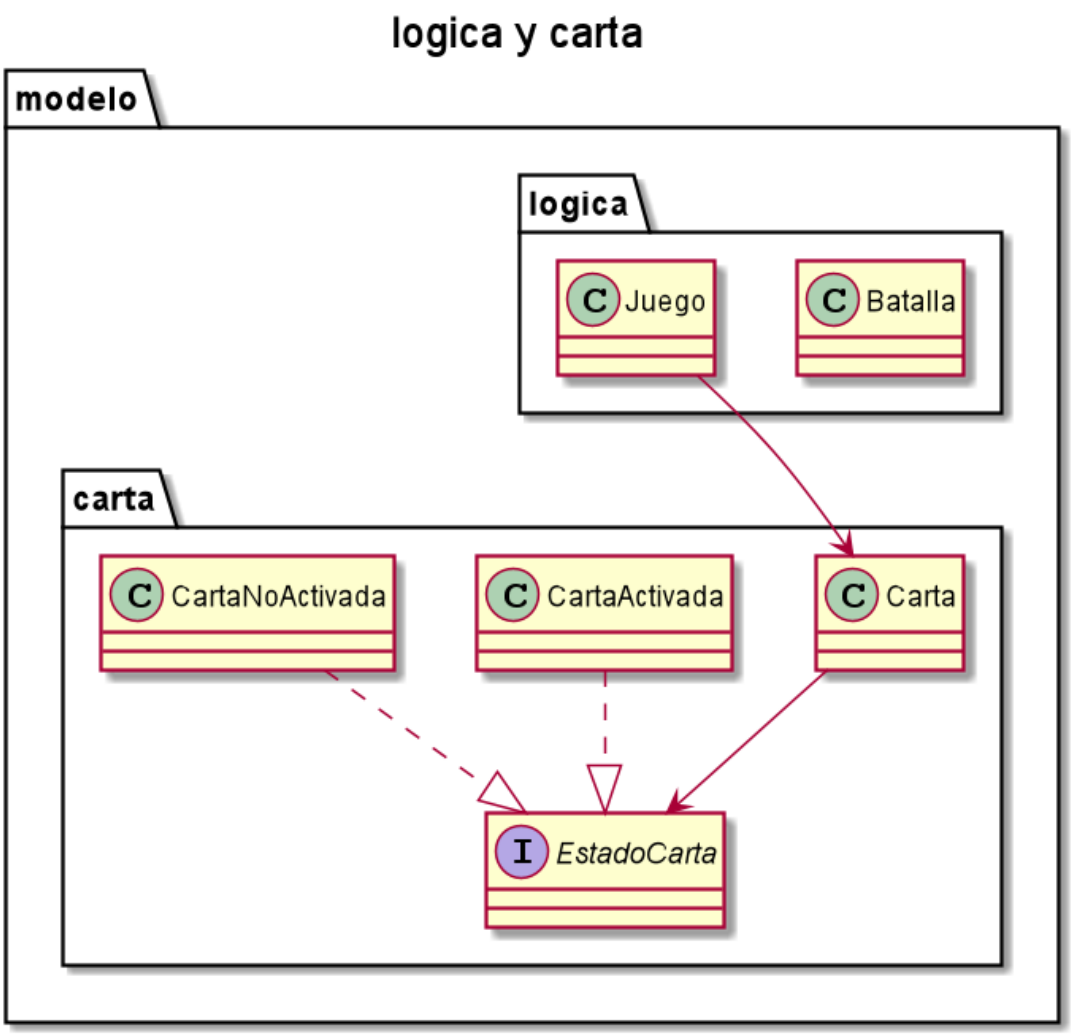

etapa, carta y geografia
modelo
etapa
EtapaColocacion
EtapaBatalla
FaseColocacion
FaseAtaque
FaseAgrupamiento
Etapa
Fase
carta
CartaNoActivada
CartaActivada
Carta
EstadoCarta
geografia
Continente
Pais

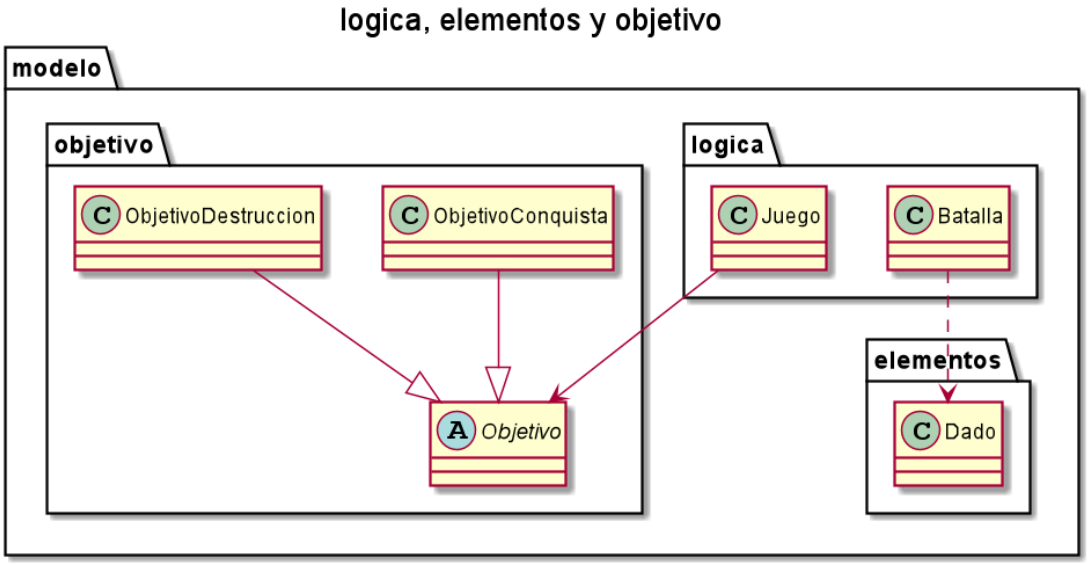

jugador y canje
modelo
canje
CanjeEstatico
CanjeDinamico
CansjeEstatico
Canje
jugador
Jugador
EstadoVivo
EstadoDerrotado
EstadoGanador
EstadoJugador

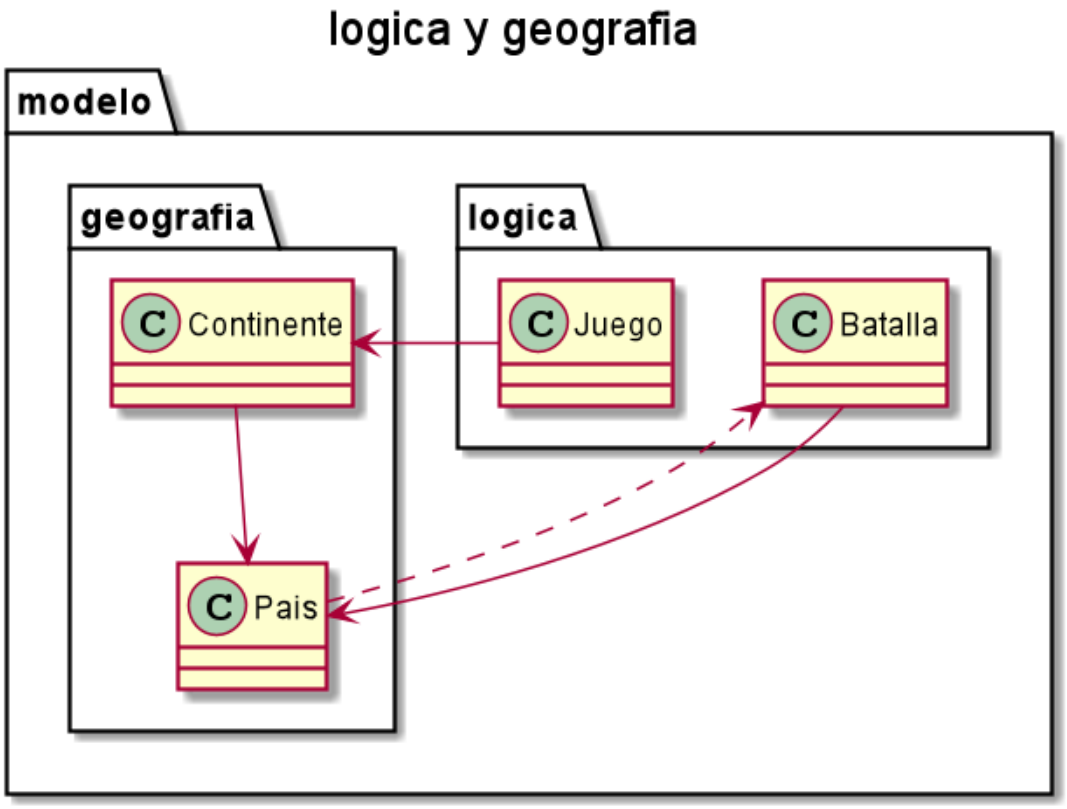

jugador y carta
modelo
carta
CartaNoActivada
CartaActivada
Carta
EstadoCarta
jugador
EstadoVivo
EstadoDerrotado
EstadoGanador
Jugador
EstadoJugador

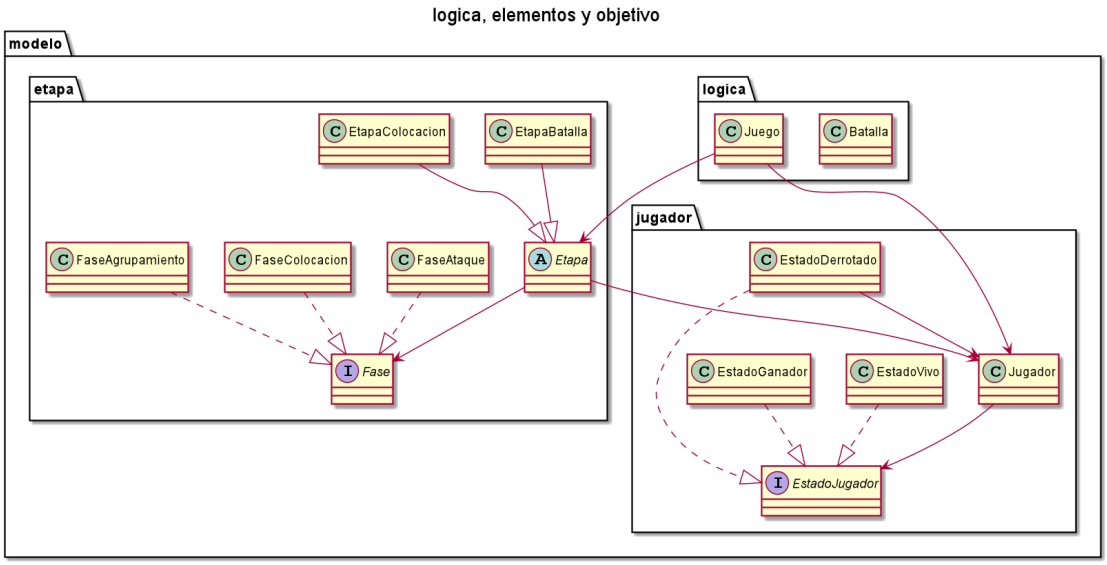

jugador y geografia
modelo
geografia
Continente
Pais
jugador
EstadoVivo
EstadoDerrotado
EstadoGanador
Jugador
EstadoJugador


jugador y objetivo
modelo
objetivo
ObjetivoDestruccion
ObjetivoConquista
Objetivo
jugador
EstadoVivo
EstadoDerrotado
EstadoGanador
Jugador
EstadoJugador


logica y carta
modelo
logica
Juego
Batalla
carta
CartaNoActivada
CartaActivada
Carta
EstadoCarta


logica, elementos y objetivo
modelo
objetivo
ObjetivoDestruccion
ObjetivoConquista
Objetivo
logica
Juego
Batalla
elementos
Dado


logica y geografia
modelo
geografia
logica
Continente
Juego
Batalla
Pais


logica, elementos y objetivo
modelo
etapa
EtapaColocacion
EtapaBatalla
FaseAgrupamiento
FaseColocacion
FaseAtaque
Etapa
Fase
logica
Juego
Batalla
jugador
EstadoDerrotado
EstadoGanador
EstadoVivo
Jugador
EstadoJugador

9. Diagramas de estado