FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Algoritmos y Programación 3
Trabajo Práctico - Etapa 2
Solitario

Cátedra Essaya - Corsi

Alumnos :
Lara Daniela Converso - 107632
Lucas Ezequiel Villarrubia - 108230

Etapa 2:

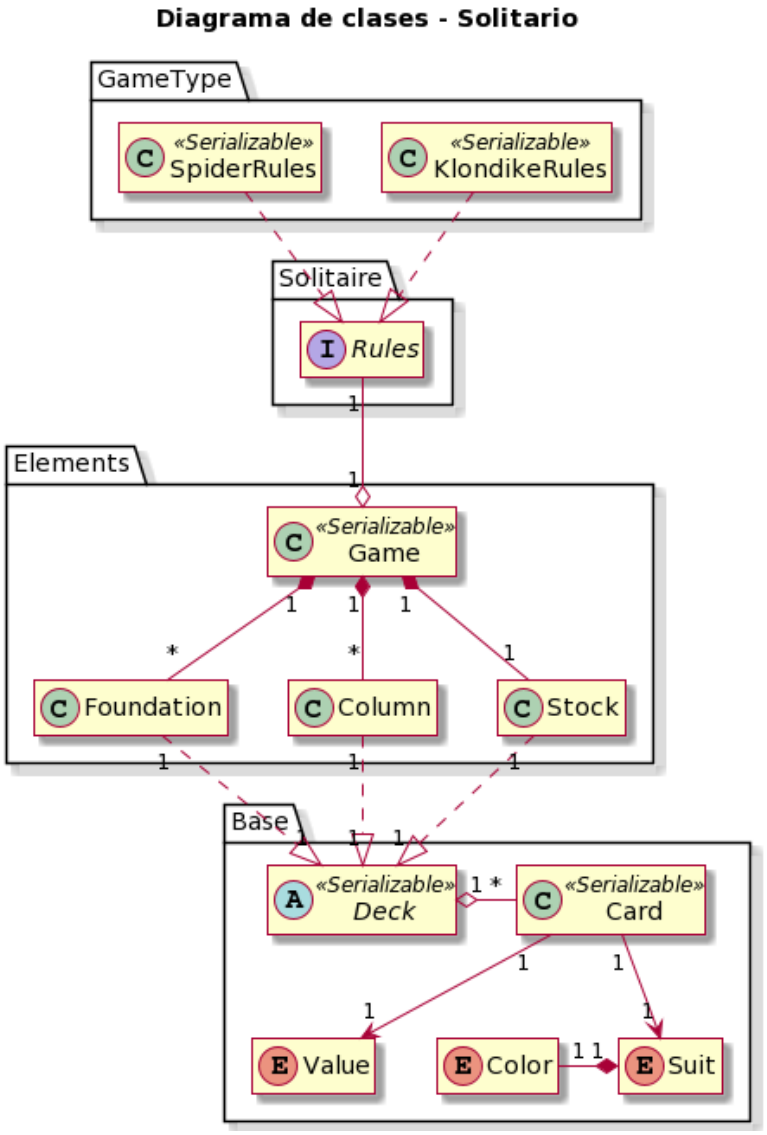
El repositorio:

Trabajamos en el repositorio https://github.com/lucasvillarrubia/algo3_tp .Para la entrega de esta etapa utilizamos una rama llamada ‘wip-etapa2’ (work in progress etapa 2).

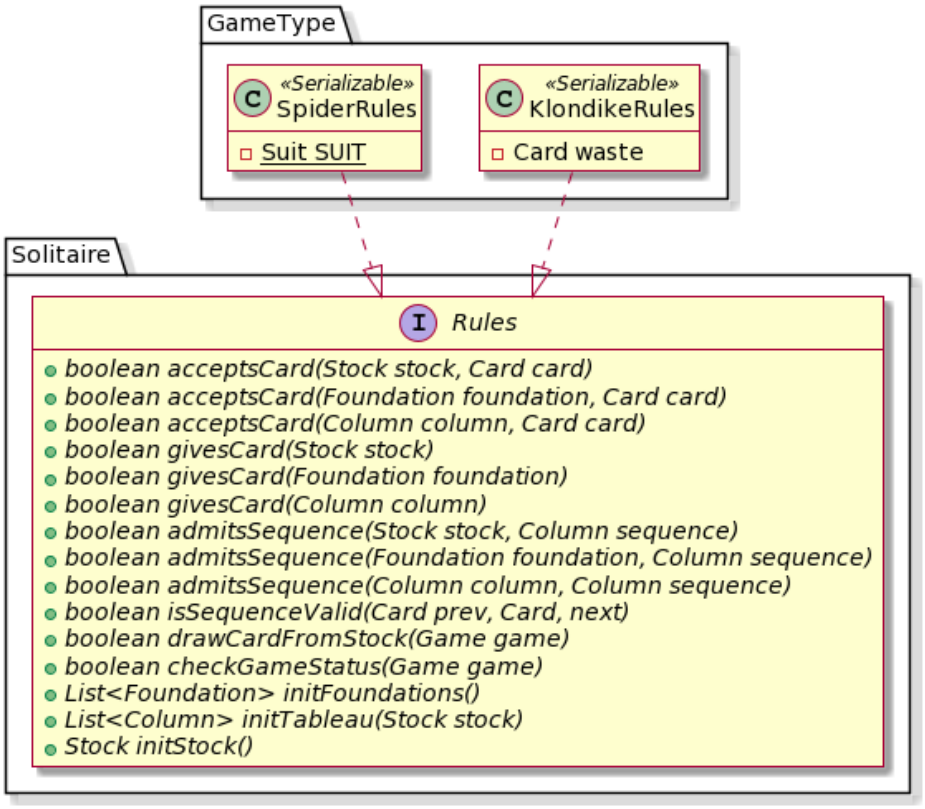
Programa:

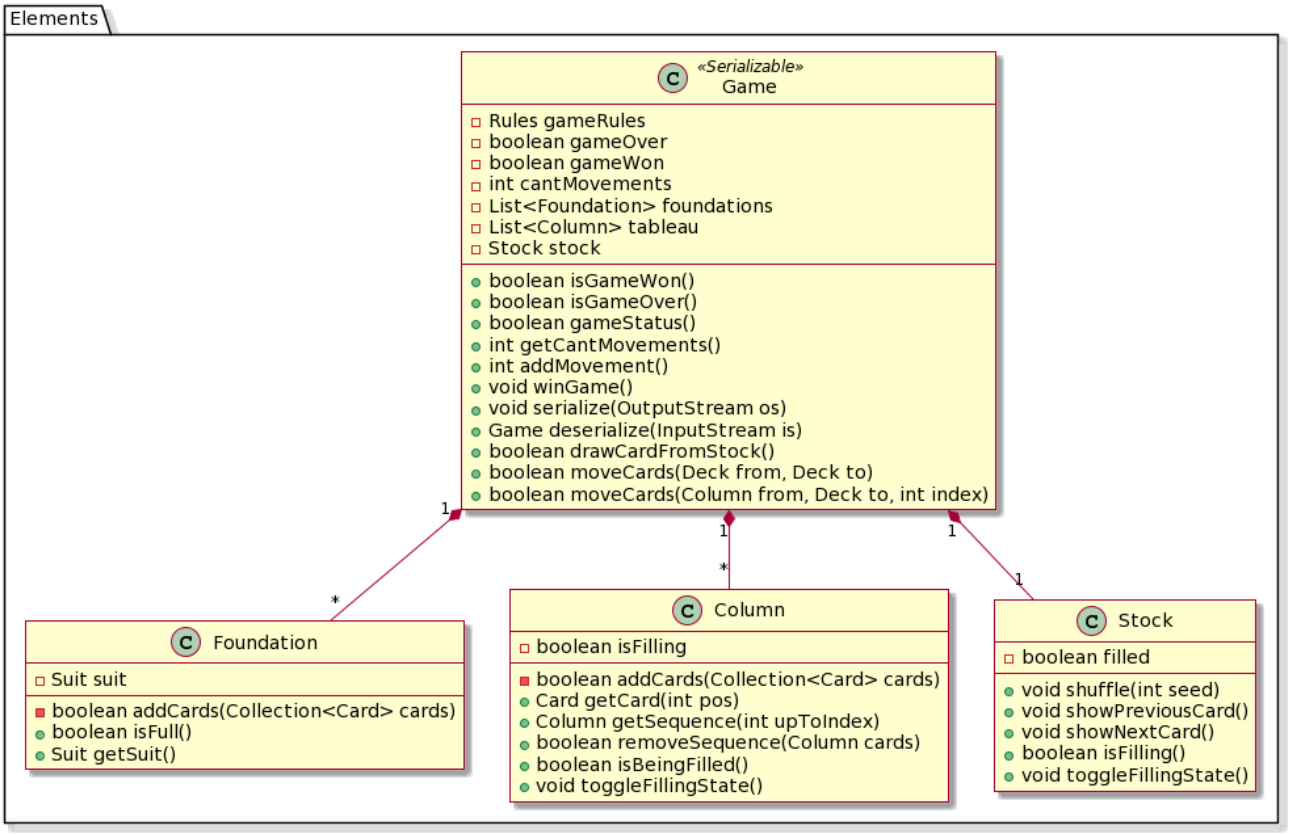
El programa contiene cuatro paquetes, el paquete de la Base del juego donde se encuentra la lógica base de un mazo de cartas. En segundo lugar el paquete de Elements, donde se encuentran los elementos que componen un solitario (el tableau, las foundations y el stock). Luego está el paquete Solitaire que contiene una interfaz con las reglas y luego el paquete GameType que contiene las clases que implementan la interfaz de reglas según el tipo de juego (Klondike y Spider).

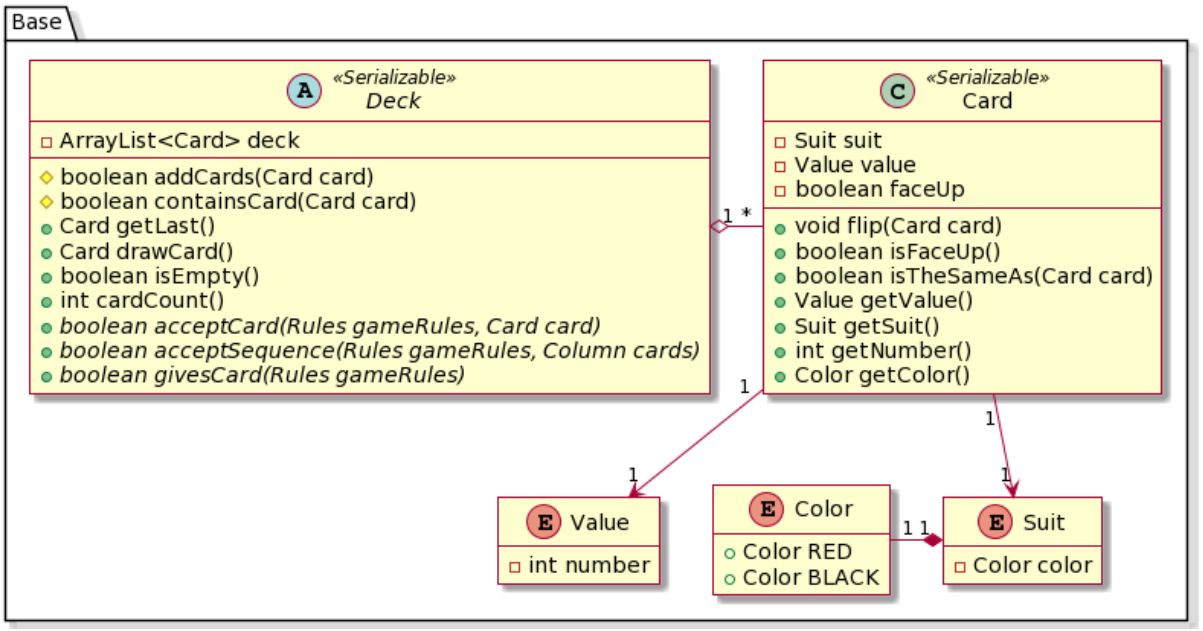
Diagramas:



Diagramas de cada paquete:


GameType
«Serializable»
SpiderRules
Suit SUIT
«Serializable»
KlondikeRules
Card waste
Solitaire
Rules
boolean acceptsCard(Stock stock, Card card)
boolean acceptsCard(Foundation foundation, Card card)
boolean acceptsCard(Column column, Card card)
boolean givesCard(Stock stock)
boolean givesCard(Foundation foundation)
boolean givesCard(Column column)
boolean admitsSequence(Stock stock, Column sequence)
boolean admitsSequence(Foundation foundation, Column sequence)
boolean admitsSequence(Column column, Column sequence)
boolean isSequenceValid(Card prev, Card, next)
boolean drawCardFromStock(Game game)
boolean checkGameStatus(Game game)
List<Foundation> initFoundations()
List<Column> initTableau(Stock stock)
Stock initStock()


Elements
«Serializable»
Game
Rules gameRules
boolean gameOver
boolean gameWon
int cantMovements
List<Foundation> foundations
List<Column> tableau
Stock stock
boolean isGameWon()
boolean isGameOver()
boolean gameStatus()
int getCantMovements()
int addMovement()
void winGame()
void serialize(OutputStream os)
Game deserialize(InputStream is)
boolean drawCardFromStock()
boolean moveCards(Deck from, Deck to)
boolean moveCards(Column from, Deck to, int index)
1
1
1
*
*
1
Foundation
Suit suit
boolean addCards(Collection<Card> cards)
boolean isFull()
Suit getSuit()
Column
boolean isFilling
boolean addCards(Collection<Card> cards)
Card getCard(int pos)
Column getSequence(int upToIndex)
boolean removeSequence(Column cards)
boolean isBeingFilled()
void toggleFillingState()
Stock
boolean filled
void shuffle(int seed)
void showPreviousCard()
void showNextCard()
boolean isFilling()
void toggleFillingState()



Enums:

- Suit: define los palos que puede tener una carta. Estructura: NOMBRE(color)
- Value: están definidos los valores posibles de una carta. Estructura: NOMBRE(valor)
- Color: contiene los dos colores (rojo y negro) que pueden tomar las cartas.

Clases:

Card: representa las cartas del juego.

- Atributos: suit, value, faceUp (booleano que define si la carta esta de frente o no)
- Métodos:
    - flip: se encarga de dar vuelta la carta, transformando el estado actual del atributo faceUp.
    - isFaceUp : consulta si la carta está de frente.
    - isTheSameAs: evalúa si las cartas tienen el mismo valor y el mismo número.
    - getValue : obtiene el valor de la carta.
    - getSuit: obtiene el palo de la carta.
    - getColor: retorna el color de la carta.
    - getNumber: obtiene el número del valor de la carta.

Deck: clase abstracta que representa los mazos de cartas

- Atributos: deck de tipo ArrayList de Cartas.
- Métodos:
    - addCards: agrega una carta al deck.
    - getLast: muestra la última carta del deck y la pone de frente, si el deck está vacío retorna null.
    - drawCard: saca la última carta del deck y la pone de frente, si el deck está vacío retorna null.
    - isEmpty: chequea si el deck está vacío.
    - cardCount: retorna la cantidad de cartas agregadas.
    - containsCard: evalúa si la instancia de esa carta ya se encuentra creada en el mismo.

- acceptSequence: método abstracto.
- acceptCard: método abstracto.
- givesCard: método abstracto.

Foundation: extiende de Deck representa la pila de cartas en donde se van acumulando de menor a mayor según su valor dependiendo de su palo.

- Atributos: suit.
- Métodos:
  - isFull: chequea si la foundation está completa.
  - getSuit: obtiene el suit asignado.
  - addCards: agrega a la foundation la colección de cartas pasada por parámetro, evaluando que no sea nula.
  - acceptCard: evalúa según las reglas pasadas por parámetro si la foundation puede agregar la carta, de ser positivo, la agrega.
  - acceptSequence: según las reglas pasadas por parámetro evalúa si la secuencia puede ser agregada a la foundation, de ser así la agrega.
  - givesCard: según las reglas recibidas por parámetro, evalúa si la Foundation puede dar una carta.

Column: extiende de Deck y representa las columnas del tableau. Cada columna inicia de una forma distinta dependiendo del tipo de juego.

- Atributos: isFilling un booleano que representa si a la columna se le están agregando cartas, comienza en true.
- Métodos:
  - toggleFillingState:
  - isBeingFilled: obtiene el valor del atributo isFilling;
  - getCard: retorna la carta que se encuentra en la posición pasada por parámetro.
  - getSequence: retorna una secuencia de cartas en forma de Column seleccionando desde el índice 0 hasta el recibido por parámetro
  - addCards: agrega a la Column la colección de cartas pasada por parámetro, evaluando que no sea nula.
  - drawCard: retorna la última carta agregada a la Column
  - removeSequence: elimina una secuencia de cartas de la Column.
  - acceptCard: evalúa según las reglas recibidas por parámetro si puede la carta se puede agregar.
  - acceptSequence: según las reglas pasadas por parámetro evalúa si la secuencia puede ser agregada a la Columna, de ser así la agrega.
  - givesCard: según las reglas recibidas por parámetro, evalúa si la Columna puede dar una carta.

Stock: extiende de Deck y representa el mazo del juego.

- Atributos: filled un booleano que comienza en true.
- Métodos:
  - showPreviousCard: muestra la carta anterior.
  - showNextCard: muestra la carta anterior.
  - toggleFillingState: modifica el valor de su atributo.
  - shuffle: a partir de una semilla recibida por parámetro mezcla el deck con la función shuffle de Collections del paquete java util.

- isFilling: retorna el valor del atributo.
- addCards: agrega al Stock la carta pasada por parámetro.
- acceptCard: evalúa según las reglas recibidas por parámetro si puede la carta se puede agregar.
- acceptSequence: según las reglas pasadas por parámetro evalúa si la secuencia puede ser agregada a la Columna, de ser así la agrega.
- givesCard: según las reglas recibidas por parámetro, evalúa si la Columna puede dar una carta.

Game: clase abstracta que contiene la inicialización y métodos básicos del juego.

- Atributos:
  - gameRules: del tipo Rules
  - gameOver: un booleano
  - gameWon: un booleano
  - cantMovements: un contador para los movimientos
  - foundations: una Lista de Foundations
  - tableau: compuesto por una Lista de Columnas
  - stock: de tipo Stock.
- Métodos:
  - isGameWon: retorna el valor del atributo gameWon.
  - isGameOver: retorna el valor del atributo gameOver.
  - winGame: modifica el valor gameWon y gameOver si cumple con las condiciones de juego ganado (el tableau está vacío, las foundations llenas y el mazo de cartas vacío) y termina el juego.
  - gameStatus: retorna el valor del atributo gameOver.
  - getCantMovements: devuelve la cantidad de movimientos..
  - addMovements: suma un movimiento.
  - getFoundationBySuit: busca la foundation según el suit (palo) pasado por parámetro.
  - getColumn: retorna la columna con el índice pasado por parámetro del tableau del game.
  - getStock: retorna el stock del game.
  - areAllFoundationsFull: recorre la lista de las foundations y chequea si están completas.
  - serialize: utiliza la interfaz Serializable para escribir un objeto del tipo Game y lo guarda en un objectOutputStream.
  - deserialize: lee el objeto recibido por parámetro y lo escribe en un game.
  - drawCardFromStock: realiza la acción de sacar una carta del mazo dependiendo de las reglas del juego.
  - moveCards: realiza el movimiento de un elemento del juego a otro, agregando un movimiento y evaluando si el juego se ganó.
  - moveCards: realiza el movimiento de una secuencia de cartas desde una columna a algún elemento del juego. Con el índice pasado por parámetro se puede saber desde donde va esta secuencia.

  Esta clase tiene tres constructores, en uno donde es requerida la semilla para mezclar el mazo, otro donde permite inicializar el juego en un estado particular, y el tercero sirve en el caso de que se serializa un juego, ya que contiene todos sus atributos.

KlondikeRules: implementa la interfaz Rules respetando las reglas del solitario Klondike.

- Atributos:
    - *AMOUNT_COLUMNS*: una constante que representa la cantidad de columnas del tableau.
    - waste: de tipo Card.

SpiderRules: implementa la interfaz Rules respetando las reglas del solitario Spider de un solo palo.

- Atributos:
    - *AMOUNT_COLUMNS*: una constante que representa la cantidad de columnas del tableau.
    - *AMOUNT_CARDS_SHORT/LONG*: una constante que representa la cantidad de cartas de las columnas del tableau.
    - *AMOUNT_COLUMNS_LONG*: una constante que representa la cantidad de columnas de cinco cartas.
    - *AMOUNT_FOUNDATIONS:* una constante que representa la cantidad de foundations.
    - *SPADES:* constante que indica el palo utilizado en este tipo de juego.

Interfaces:

Rules: es una interfaz que define los métodos que serán implementados por cada tipo de regla.

- Métodos:
    - acceptsCard: recibe siempre una carta y también recibe o un stock, o una foundation o una Column, según las reglas permite o no que se acepte la carta al tipo de deck recibido por parámetro.
    - givesCard: evalúa según las reglas si el elemento recibido por parámetro puede dar una carta o no.
    - admitsSequence: evalúa según las reglas si la secuencia recibida por parámetro puede ser incorporada al elemento también pasado por parámetro.
    - initStock: inicializa el stock según las reglas.
    - initTableau: inicializa el tableau, con las cartas necesarias en cada columna según el tipo de solitario.
    - initFoundations: inicializa las fundaciones según el tipo de solitario.
    - drawCardFromStock: realiza la acción de pedirle una carta al mazo según el tipo de solitario.
    - isSequenceValid: evalúa si la secuencia es válida según el tipo de solitario.

Pruebas:

Las pruebas se encuentran en la carpeta : …src/test/java…
Dentro de esta carpeta se replica la misma estructura del programa. Cada clase tiene su test en donde se prueba el funcionamiento de sus métodos y el funcionamiento de las clases en conjunto.
En la clase Game se encuentran los tests relacionados con la persistencia de un juego mismo y también incluimos tests dedicados a ambos tipos de solitario, que fueron realizados utilizando el debugger de IntelliJ.

Cobertura obtenida:

Element	Class, %	Method, %	Line, %
all	100% (11/11)	100% (103/103)	100% (300/300)
Solitaire	100% (0/0)	100% (0/0)	100% (0/0)
Rules	100% (0/0)	100% (0/0)	100% (0/0)
GameType	100% (2/2)	100% (29/29)	100% (123/123)
SpiderRules	100% (1/1)	100% (15/15)	100% (64/64)
KlondikeRules	100% (1/1)	100% (14/14)	100% (59/59)
Elements	100% (4/4)	100% (49/49)	100% (137/137)
Stock	100% (1/1)	100% (11/11)	100% (18/18)
Game	100% (1/1)	100% (18/18)	100% (69/69)
Foundation	100% (1/1)	100% (8/8)	100% (18/18)
Column	100% (1/1)	100% (12/12)	100% (32/32)
Base	100% (5/5)	100% (25/25)	100% (40/40)
Value	100% (1/1)	100% (4/4)	100% (7/7)
Suit	100% (1/1)	100% (4/4)	100% (8/8)
Deck	100% (1/1)	100% (7/7)	100% (12/12)
Color	100% (1/1)	100% (2/2)	100% (2/2)
Card	100% (1/1)	100% (8/8)	100% (11/11)

Comentarios:

Aplicamos el patrón de diseño Strategy para implementar diferentemente los dos tipos de solitario que elegimos: Klondike y Spider (variante de un palo). Para esto, la interfaz *Rules* fue creada para determinar qué comportamiento debe tener un nuevo tipo de solitario al agregarse.

Junto con este cambio, se sumaron modificaciones a las clases ya existentes en la Etapa 1 para ampliar su comportamiento polimórfico:

- ➔ La clase *Tableau* fue suprimida, su comportamiento se ejecuta desde la clase *Game*.
- ➔ La clase *Deck* es ahora abstracta, con la mayoría de sus métodos concretos.
- ➔ Las clases *Foundation* y *Column* pasaron a ser subclases de *Deck*, implementando su comportamiento abstracto y otro adicional específico de cada una.
- ➔ Se agregó la clase *Stock*, otro tipo de *Deck* que también define su propio comportamiento adicional.

Para la persistencia del estado de juego, utilizamos la interfaz *Serializable* de Java importada desde el paquete …java.io… y su comportamiento se aplica desde la clase *Game*.