FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Algoritmos y Programación 3
Trabajo Práctico - Etapa 3
Solitario

Cátedra Essaya - Corsi

Alumnos :

Lara Daniela Converso - 107632

Lucas Ezequiel Villarrubia - 108230

Etapa 3 versión 2:

Correcciones:

En el informe se representan los metodos eliminados con color rojo y los métodos agregados o modificados con color azul.

El repositorio:

Trabajamos en el repositorio https://github.com/lucasvillarrubia/algo3_tp .Para la entrega de esta etapa utilizamos una rama llamada ‘wip-etapa3’ (work in progress etapa 3).

Programa:

El programa contiene cinco paquetes, el paquete de la Base del juego donde se encuentra la lógica base de un mazo de cartas de poker. En segundo lugar el paquete de Elements, donde se encuentran los elementos que componen un solitario (el tableau, las foundations y el stock) junto con las interfaces necesarias para que sean accesibles a la lógica de un juego. Luego está el paquete Solitaire que contiene una interfaz con las reglas, los movimientos y el solitario en si. Luego el paquete GameType que contiene las clases que implementan la interfaz de reglas según el tipo de juego (Klondike y Spider). Y por último el paquete de la UI (interfaz de usuario en inglés) que contiene las clases relacionadas a la vista del programa.

Finalmente, una clase Main donde se ejecuta la aplicación con la vista.

Ejecución:

Para poder visualizar la ejecución del programa con su interfaz gráfica. En el caso de haberlo abierto con IntelliJ. Se debe correr desde la pestaña de Maven (usualmente ubicada a la derecha de la pantalla) desde la carpeta de Plugins, desde el plugin de javafx con [javafx:run].

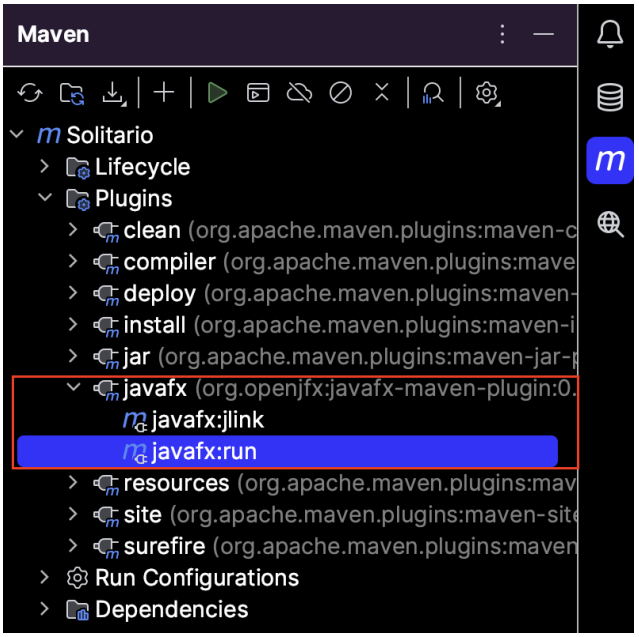
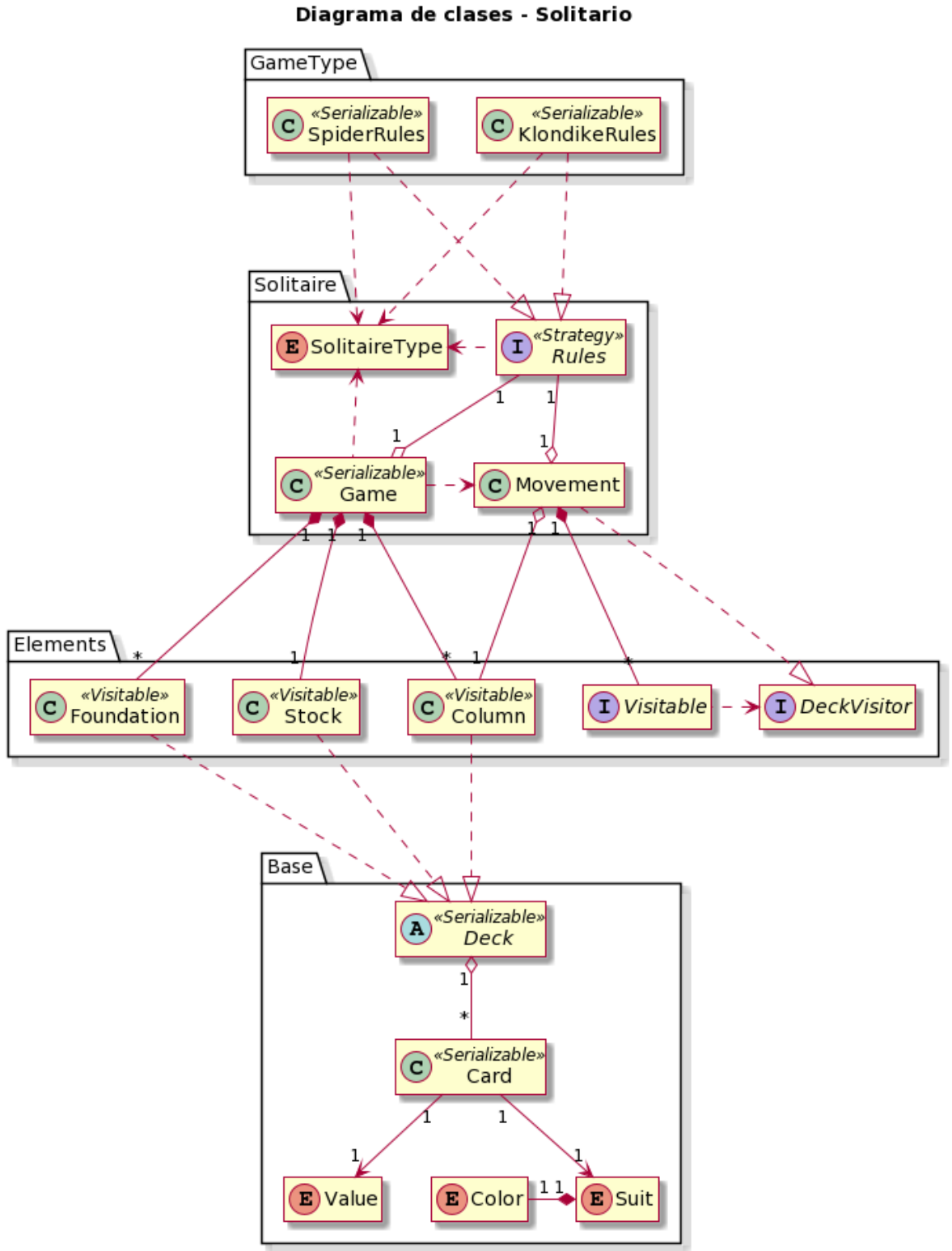


Diagrama de clases:

En este diagrama se encuentra la abstracción principal del programa. Es un diagrama simplificado con las relaciones principales omitiendo dependencias (todas unidireccionales).



Objetos desarrollados en la etapa 3:

Enums:

UI:

- ClickState: define el estado del click : *NO_CLICK* o *CLICKED,* utilizado en los distintos tipos de juegos.

Interfaces:

Elements:

Visitable y DeckVisitor: hacen referencia al patrón de visitor puro. La primera se encarga de proveer accesibilidad a cada elemento desde un *visitor* y la segunda de acceder a cada elemento y realizar acciones según la clase de instancia recibida en las sobrecargas del método *visit()* que posee.

UI:

Clickable: interfaz utilizada para cada elemento del juego que recibe clicks.

- Métodos:
  - getIndex: devuelve el índice del StackPane en el que se encuentra.
  - getDeck: devuelve el mazo del juego asociado a la vista.
  - getClickedCardIndex: revisa si posee una carta clickeada y devuelve el índice de esa carta.
  - turnOffSelectedCard: indica a la carta clickeada que posee que setee su efecto visual a “no seleccionada” (nulo).

Clases:

Solitaire:

Movement: representa un movimiento entre dos mazos visitables

- Atributos: mazos visitables de orígen y destino (from, to), sequenceToMove, sourceChecked, validMove, cardIndex, rules.
- Métodos: además de heredar e implementar los métodos *visit()* de DeckVisitor, posee:
  - moveCards: retorna true si se ejecuta exitosamente el movimiento de una carta entre dos mazos.
  - moveSequence: retorna true si se ejecuta exitosamente el movimiento de una secuencia entre dos mazos.
  - checkMoveByRules: retorna el valor de validMove una vez que se aplicaron las reglas del juego actual sobre el mazo de salida y llegada involucrados en el movimiento, implica chequear que el mazo de salida puede otorgar una carta o secuencia y el de llegada puede recibirla.

UI:

CardView: representa la vista de las cartas del juego. Extiende de ImageView e implementa la interfaz Clickeable.

- Atributos: Map de cartas con la imagen, cardIndex, clickState y utilizados como constantes IMAGE_LOCATION, IMAGE_SUFFIX, BACK_IMAGE_NAME para la obtener las imágenes. Y W y H para alto y ancho de la carta. Y SELECTED EFFECT para el efecto de seleccionado.
- Métodos:
  - getImage: retorna una imageView de la carta dependiendo de si se encuentra boca arriba o hacia abajo.
  - getPhoto: busca la foto indicada para la carta dentro de la carpeta de /resources/images/cards

- getBack: busca la imagen correspondiente para la parte de atrás de una carta.
- getFoundationImage: setea la imagen para la foundation con el suit correspondiente.
- handleCardClick: cada vez que la carta es clickeada se llama a este método.
- toggleCardClick: cambia el estado del atributo clickState.
- unselectCard: modifica la vista de la carta y setea el atributo clickState a falso.
- isClicked: retorna el valor del atributo clickState.
- setIndex: setea el índice;
- getIndex: retorna el índice;

ColumnView: extiende de StackPane, implementa Clickable y representa la vista de las columnas del tableau.

- Atributos: column, index, clickState y offset utilizado como constante para el desplazamiento de cartas. Constante de ERROR (-1)
- Métodos:
  - getColumn: retorna la columna asignada a la vista.
  - buildColumn: arma la visual de la columna pasada por parámetro.
  - getCardView: retorna la vista de una carta a partir de un índice.
  - toggleColumnClick: cambia el estado del atributo clickState.
  - handleColumnClick: cada vez que la columna es clickeada se llama a este método.
  - estaClickeado: retorna el valor del atributo clickState.
  - setIndex: setea el índice;

FoundationVIew: extiende de StackPane, implementa Clickable y representa la vista de las foundations.

- Atributos: cardView, suit, foundation, clickState, index, constante de ERROR (-1) y OK (0).
- Métodos:
  - buildFoundation: evalúa si la foundation está vacía y le asigna la imagen que le corresponde.
  - toggleFoundationClick: cambia el estado del atributo clickState.
  - handleFoundationClick: cada vez que la foundation es clickeada se llama a este método.
  - get Foundation: retorna la foundation que le corresponde.
  - estaClickeado: retorna el valor del atributo clickState.
  - setIndex: setea el índice.

StockView: extiende de HBox, implementa Clickable y representa la vista del stock.

- Atributos: cardView, stock, clickState, number (haciendo referencia al indice).
- Métodos:
  - showStock: crea la vista para un stock.
  - showEmptyStock: crea la visa para un stock vacio.
  - eventHandler: como stock funciona como un botón, al darle click modifica su posición.

WasteView: extiende de StackPane, implementa Clickable y representa la vista de la pila de "desecho" en los tipos de solitario que la implementen.

- Atributos: stock, index.
- Métodos:
    - update: actualiza la vista de la waste, mostrando la carta que se ve al frente del stock que posee.

GameUI: clase abstracta que contiene los métodos necesarios para setear la vista de un juego y recibir las acciones que realiza el usuario durante la ejecución del programa.

- Atributos: game, clickstate, columnView, clickedFoundation, clickedCard y los siguientes hacen referencia a elementos de los FXML tableau, foundations, stockPile, moves. En las correcciones, se cambiaron algunos atributos por dos Clickable sourceDeck y goalDeck, un entero clickedCardIndex y un booleano showingEmptyStock.
- Métodos:
    - setUpGame: método abstracto. construye la vista de un juego a partir de un juego pasado por parámetro y lo muestra en el stage que también es un parámetro.
    - handleColumnClick: responde al click en una columna y de ser posible llama el método
    - handleFoundaitonClick: reacciona al evento del mouse (click) y llama al método acceptMoveToFoundation.
    - handleStockClick: método abstracto, a partir del evento realiza movimientos dependiendo del tipo de solitario.
    - acceptMoveToColumn: verifica las condiciones para realizar movimientos entre columnas.
    - acceptMoveToFoundation: método abstracto. Dependiendo del tipo de solitario realiza el movimiento a la fundación.
    - setEventHandlers: setea para los elementos del solitario los handlers que responden a los clicks que realice el usuario.
    - getClickedCard: recorre la columna recibida por parámetro y retorna la carta clickeada.
    - updateTableauView: setea la vista de la tableau, creando la cantidad de columnViews pasadas por parámetro.
    - updateFoundations: setea la vista de las foundations, creando la cantidad pasada por parámetro.
    - updateColumnView: a partir de la vista de columnas pasada por parámetro, la actualiza según lo que haya ocurrido en el juego.
    - updateStockButton: método abstracto, dependiendo del solitario y las condiciones de este se setea el botón.
    - showWinScene: recibe un stage en el cual se muestra la escena de victoria.
    - checkWiningCondition: chequea la condición de victoria.
    - updatePane: a partir del Pane pasado por parámetro, actualiza la vista con el parámetro de updatedView.
    - updateDeckView: dependiendo de si es una columna o una foundation actualiza la vista de esta como corresponde.
    - handleClick: maneja el evento de click del mouse, con las evaluaciones pertinentes para cada Deck.

KlondikeUI: extiende de GameUI, muestra la interfaz de usuario para el solitario de tipo Klondike.

- Atributos:
  - stage: donde se muestra la escena.
  - wasteIsClicked: booleano que indica si la waste ha sido clickeada.
  - *FILE_PATH:* contiene un String con la ruta del archivo, en donde se guarda el juego.
  - *AMOUNT_COLUMNS:* constante con la cantidad de columnas del juego (7).
  - *AMOUNT_FOUNDATIONS*: constante con la cantidad de foundations del juego (4).
  - W y H: son constantes utilizadas para setear el alto y ancho de la ventana donde se mostrará el juego.
  - waste: es un elemento del fxml, donde será ubicada la carta sacada del stock.
- Métodos:
  - setUpGame: construye la vista de un juego a partir de un juego pasado por parámetro y lo muestra en el stage que también es un parámetro.
  - acceptMoveToFoundation: realiza los movimientos que involucren foundations, realizando los chequeos necesarios y con el resultado de estos actualiza la vista del solitario.
  - acceptMoveToColumn: realiza los movimientos que involucren columnas, realizando los chequeos necesarios y con el resultado de estos actualiza la vista del solitario.
  - updateStockButton: setea la vista para un stock y asigna que en el evento click de este, se ejecute alguno de los handlers, dependiendo del caso.
  - handleEmptyStockClick: setea la vista para un stock vacio.
  - handleStockClick: realiza el movimiento de sacar una carta del stock, cumpliendo las reglas de klondike.
  - handleWasteCardClick: actualiza el estado a clicked del atributo clickState y el de wasteIsClicked a true.
  - updateWaste: setea la vista del waste.
  - setEventHandlers: setea los handlers utilizando el método de la clase abstracta padre y el del waste.

SpiderUI: extiende de GameUI, muestra la interfaz de usuario para el solitario de tipo Spider.

- Atributos:
  - *FILE_PATH:* contiene un String con la ruta del archivo, en donde se guarda el juego.
  - *AMOUNT_COLUMNS:* constante con la cantidad de columnas del juego (10).
  - *AMOUNT_FOUNDATIONS*: constante con la cantidad de foundations del juego (8).
  - W y H: son constantes utilizadas para setear el alto y ancho de la ventana donde se mostrará el juego.
- Métodos:
  - setUpGame: construye la vista de un juego a partir de un juego pasado por parámetro y lo muestra en el stage que también es un parámetro.

- acceptMoveToFoundation: realiza el movimiento a la foundation, si cumple con las condiciones y actualiza la vista del solitario.
- handleStockClick: cuando se realiza el evento (en este caso click) se reparten las cartas del stock respetando las reglas de este tipo de solitario.
- updateStockButton: setea la vista del Stock para este tipo de solitario.

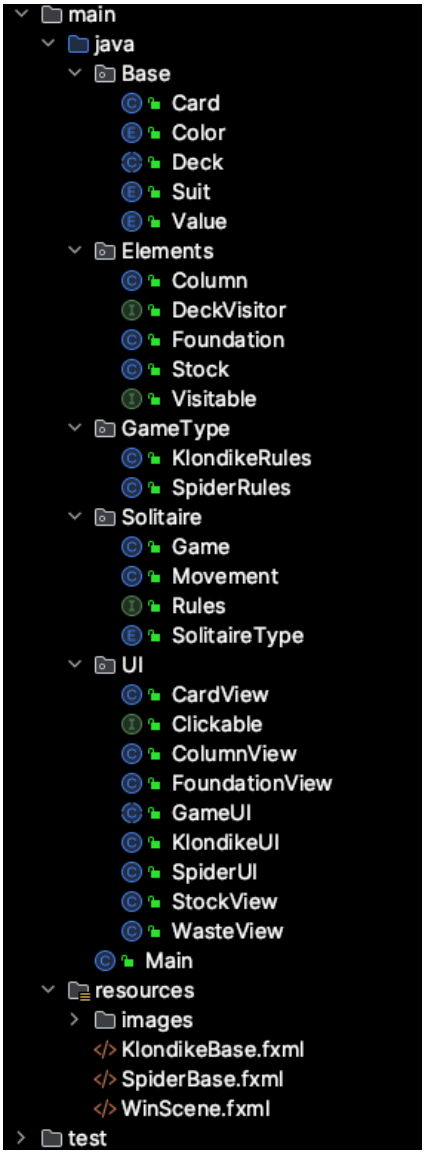
Para la creación de la parte estática de la vista utilizamos Scene Builder donde armamos los archivos xml donde ubicamos la vista de cada elemento dependiendo del solitario, estos archivos se pueden encontrar en resources.

¿Cómo jugar?

Luego de correr el programa, se elige el tipo de solitario al que se va a jugar. En este caso las opciones son Klondike y Spider. Luego aparecerá una pantalla con el juego iniciado con una semilla aleatoria.
Para realizar un movimiento se debe clickear la carta que se quiere mover y luego donde va a ser desplazada. En caso de querer mover una secuencia de cartas, se debe clickear la carta desde donde inicia esta misma.
Los movimientos se realizan de a pares de clicks, es decir, si clickeo una carta y me arrepiento de querer moverla, simplemente la puedo volver a clickear para seguir con el movimiento siguiente.
La condición de victoria es que todas las cartas del juego se encuentren apiladas en la foundations, dejando vacío el stock y el tableau.
Al ganar aparecerá una pantalla con la cantidad de movimientos realizados en la partida.
En el caso de cerrar el juiego mientras se encuentra en curso, al volver a abrir el programa, este te permite elegir si iniciar uno nuevo o continuar con el anterior.

Comentarios:

Realizamos una reestructuración y modificaciones al modelo para no tener dependencias cíclicas.
Aplicamos el patrón de diseño Visitor para solucionar el problema de accesibilidad entre clases del modelo anterior. Este patrón produjo una violación al principio DRY al implementar los métodos para ‘visitar’ cada tipo de clase (mazo) visitable, pero mantuvimos esta alternativa de solución por cuestión y decisión de diseño.
Por último se modificó la serialización del juego, para que se guarde en un archivo binario.
La estructura del programa actualizado quedo asi:



Referencias:

Las imágenes de las cartas utilizadas fueron obtenidas de https://code.google.com/archive/p/vector-playing-cards/ , publicadas por Byron Knoll.