# Solitaire

Introduction

Solitaire is a popular single-player card game that is played with a standard deck of 52 cards. The objective of the game is to sort all the cards into their respective suits, arranged from Ace to King, in an ascending order. The game is challenging, requiring patience, strategy, and a bit of luck.

Rules

1. Set up the game by arranging the cards into seven piles, with the first pile containing one card, the second pile containing two cards (with the second card overlapping the first), and so on, until the seventh pile contains seven cards.
2. The remaining cards from the stock, which is placed face down.
3. The top card of the stock pile is turned over to start the discard pile or talon.
4. To build the foundations, Aces are the base, and cards must be placed in the same suit, in ascending order (Ace, 2, 3, ..., King).
5. The tableau piles (the seven piles initially dealt) can be built down in alternating colors (red suits on black suits, and vice versa), with cards ranked in descending order (King, Queen, Jack, ..., Ace).
6. Empty spaces in the tableau can be filled with a King or a series of cards with the King at the bottom.
7. Cards from the discard pile can be played to the foundations or the tableau piles, following the rules above.
8. When no more moves are possible from the discard pile, the stockpile is turned over, and the cycle continues.
9. The game is won when all cards are built onto the foundations, from Ace to King, in each suit.

- You can reveal one stock card at a time by moving it to the waste. You can only add the top card of the waste to the tableau or the foundation.
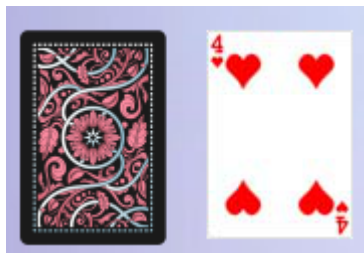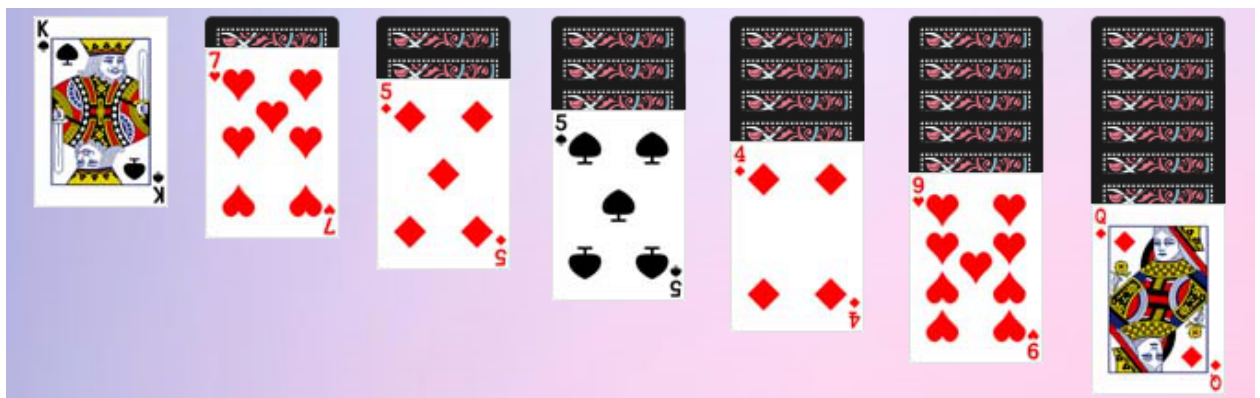

- The four panels on the top right of the playing field are empty at the beginning of the round. They are the foundation piles, where you want all sorted cards to end up. Each pile can only contain one suit. The ranks of each pile must be placed there in ascending order

- At the beginning of a round, 28 cards are arranged in the tableau. The first of seven piles in the middle of the table contains one card. The other six piles each contain one more card than the previous pile. Each top card is revealed.
  You can move cards from pile to pile. After removing the revealed cards from a pile, the new top card is revealed. In order to move several cards in the tableau, you must sort them, and you can also add cards from the waste to the tableau.
  - You must sort the cards uninterrupted in descending order, starting with the King at the bottom, continuing Queen, Jack, 10, 9, 8, 7, 6, 5, 4, 3, 2, and Ace.
  - Alternating red and black suits: A card of a red suit, Hearts or Diamonds, must follow a black suit, Spades or Clubs, and vice versa.
  - You can only place a King on an empty tableau pile

- Undo button and new game button

Homepage
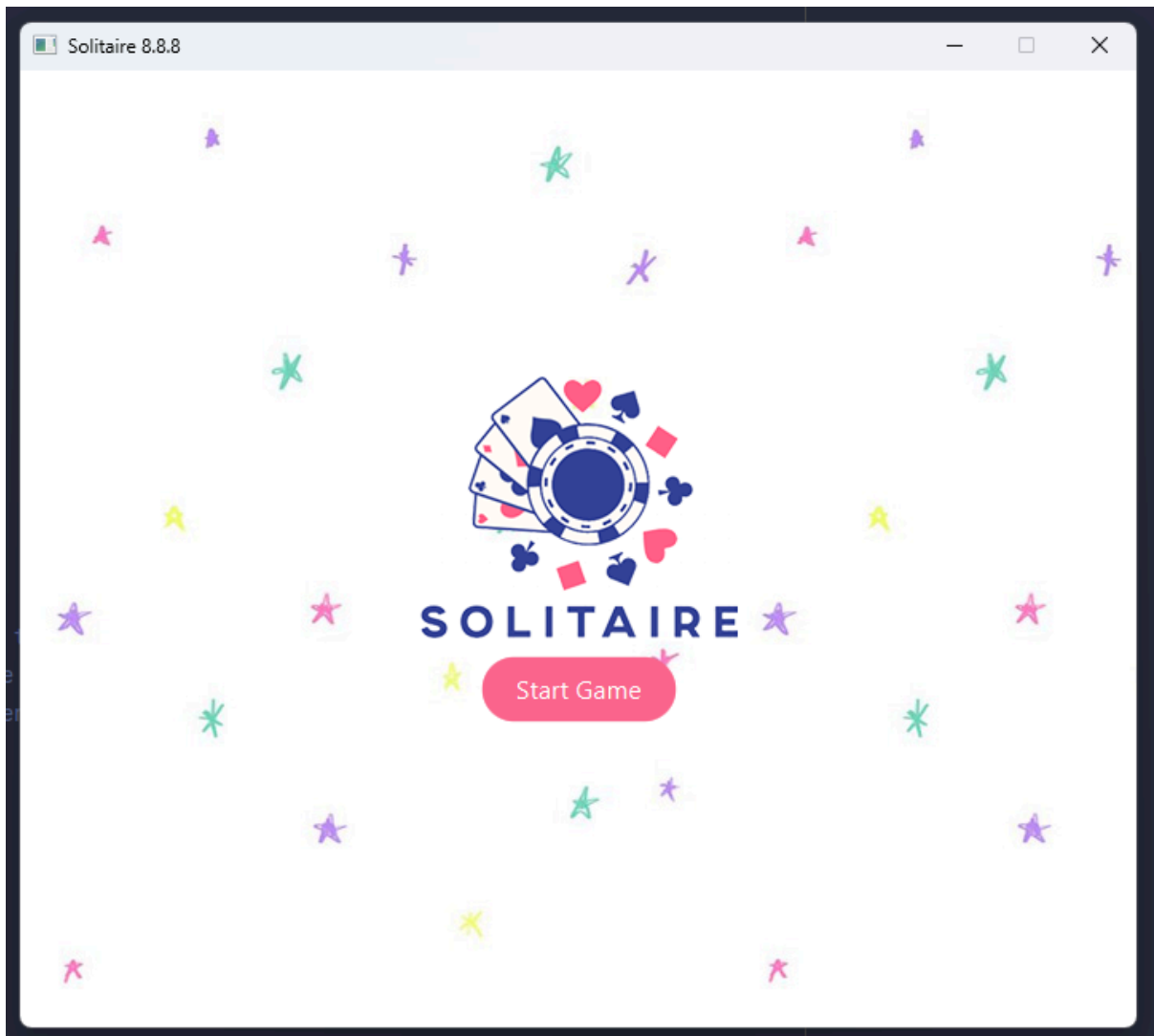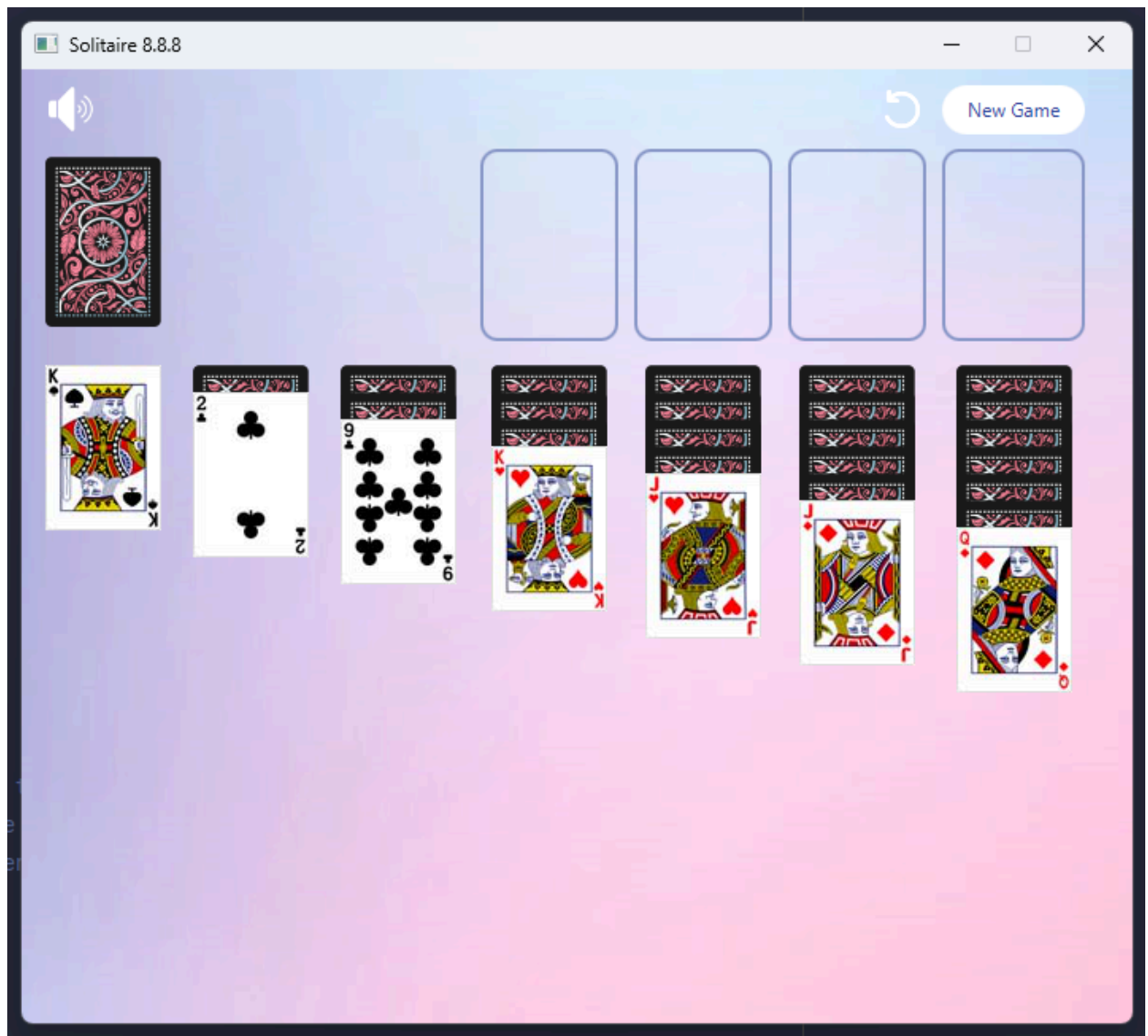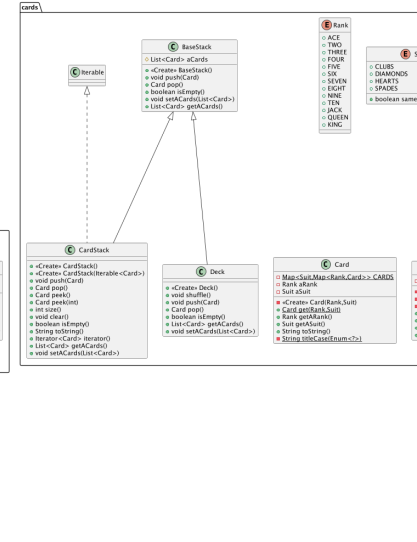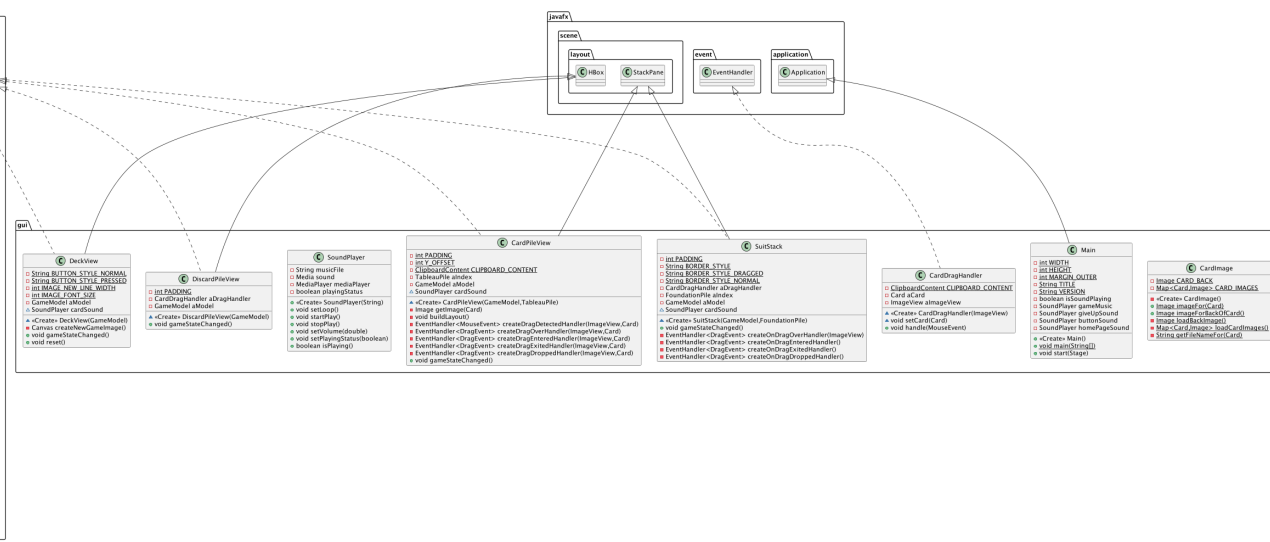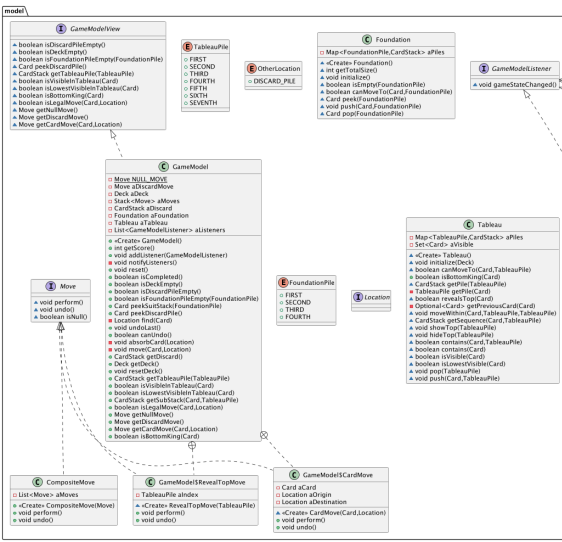
Game Page

UML Class Diagram (Solitaire application)

Package: model

GameModelView «interface»
- boolean isDiscardPileEmpty()
- boolean isDeckEmpty()
- boolean isFoundationPileEmpty(FoundationPile)
- Card peekDiscardPile()
- CardStack getTableauPile(TableauPile)
- boolean isVisibleInTableau(Card)
- boolean isLowestVisibleInTableau(Card)
- boolean isBottomKing(Card)
- boolean isLegalMove(Card,Location)
- Move getNullMove()
- Move getDiscardMove()
- Move getCardMove(Card,Location)

TableauPile «enum»
- FIRST
- SECOND
- THIRD
- FOURTH
- FIFTH
- SIXTH
- SEVENTH

OtherLocation «enum»
- DISCARD_PILE

Foundation
- Map<FoundationPile,CardStack> aPiles
- «Create» Foundation()
- int getTotalSize()
- void initialize()
- boolean isEmpty(FoundationPile)
- boolean canMoveTo(Card,FoundationPile)
- Card peek(FoundationPile)
- void push(Card,FoundationPile)
- Card pop(FoundationPile)

GameModelListener «interface»
- void gameStateChanged()

GameModel
- Move NULL_MOVE
- Move aDiscardMove
- Deck aDeck
- Stack<Move> aMoves
- CardStack aDiscard
- Foundation aFoundation
- Tableau aTableau
- List<GameModelListener> aListeners
- «Create» GameModel()
- int getScore()
- void addListener(GameModelListener)
- void notifyListeners()
- void reset()
- boolean isCompleted()
- boolean isDeckEmpty()
- boolean isDiscardPileEmpty()
- boolean isFoundationPileEmpty(FoundationPile)
- Card peekDiscardPile()
- Optional<Card> getPreviousCard(Card)
- void moveWithin(Card,TableauPile,TableauPile)
- CardStack getSequence(Card,TableauPile)
- void hideTop(TableauPile)
- boolean contains(Card,TableauPile)
- boolean contains(Card)
- boolean isVisible(Card)
- boolean isLowestVisible(Card)
- void move(Card,Location)
- Deck getDeck()
- void resetDeck()
- CardStack getTableauPile(TableauPile)
- boolean isVisibleInTableau(Card)
- boolean isLowestVisibleInTableau(Card)
- CardStack getSubStack(Card,TableauPile)
- boolean isLegalMove(Card,Location)
- Move getNullMove()
- Move getDiscardMove()
- Move getCardMove(Card,Location)
- boolean isBottomKing(Card)

Move «interface»
- void perform()
- void undo()
- boolean isNull()

FoundationPile «enum»
- FIRST
- SECOND
- THIRD
- FOURTH

Location «interface»

Tableau
- Map<TableauPile,CardStack> aPiles
- Set<Card> aVisible
- «Create» Tableau()
- void initialize(Deck)
- boolean canMoveTo(Card,TableauPile)
- boolean isBottomKing(Card)
- CardStack getPile(TableauPile)
- TableauPile getPile(Card)
- boolean revealsTop(Card)
- Optional<Card> getPreviousCard(Card)
- void moveWithin(Card,TableauPile,TableauPile)
- CardStack getSequence(Card,TableauPile)
- void hideTop(TableauPile)
- boolean contains(Card,TableauPile)
- boolean contains(Card)
- boolean isVisible(Card)
- boolean isLowestVisible(Card)
- void pop(TableauPile)
- void push(Card,TableauPile)

CompositeMove
- List<Move> aMoves
- «Create» CompositeMove(Move)
- void perform()
- void undo()

GameModel$RevealTopMove
- TableauPile aIndex
- «Create» RevealTopMove(TableauPile)
- void perform()
- void undo()

GameModel$CardMove
- Card aCard
- Location aOrigin
- Location aDestination
- «Create» CardMove(Card,Location)
- void perform()
- void undo()

Package: javafx

scene
- layout: HBox, StackPane
- event: EventHandler
- application: Application

Package: gui

DeckView
- String BUTTON_STYLE_NORMAL
- String BUTTON_STYLE_PRESSED
- int IMAGE_NEW_LINE_WIDTH
- int IMAGE_FONT_SIZE
- GameModel aModel
- SoundPlayer cardSound
- «Create» DeckView(GameModel)
- Canvas createNewGameImage()
- void gameStateChanged()
- void reset()

DiscardPileView
- int PADDING
- CardDragHandler aDragHandler
- GameModel aModel
- «Create» DiscardPileView(GameModel)
- void gameStateChanged()

SoundPlayer
- String media
- Media sound
- MediaPlayer mediaPlayer
- boolean playingStatus
- «Create» SoundPlayer(String)
- void setLoop()
- void startPlay()
- void stopPlay()
- void setVolume(double)
- void setPlayingStatus(boolean)
- boolean isPlaying()

CardPileView
- int PADDING
- int Y_OFFSET
- ClipboardContent CLIPBOARD_CONTENT
- TableauPile aIndex
- GameModel aModel
- SoundPlayer cardSound
- «Create» CardPileView(GameModel,TableauPile)
- Image getImage(Card)
- void buildLayout()
- EventHandler<MouseEvent> createDragDetectedHandler(ImageView,Card)
- EventHandler<DragEvent> createDragOverHandler(ImageView,Card)
- EventHandler<DragEvent> createDragEnteredHandler(ImageView,Card)
- EventHandler<DragEvent> createDragExitedHandler(ImageView,Card)
- EventHandler<DragEvent> createDragDroppedHandler(ImageView,Card)
- void gameStateChanged()

SuitStack
- int PADDING
- String BORDER_STYLE
- String BORDER_STYLE_DRAGGED
- String BORDER_STYLE_NORMAL
- FoundationPile aIndex
- CardDragHandler aDragHandler
- GameModel aModel
- SoundPlayer cardSound
- «Create» SuitStack(GameModel,FoundationPile)
- void gameStateChanged()
- EventHandler<DragEvent> createOnDragOverHandler(ImageView)
- EventHandler<DragEvent> createOnDragEnteredHandler()
- EventHandler<DragEvent> createOnDragExitedHandler()
- EventHandler<DragEvent> createOnDragDroppedHandler()

CardDragHandler
- ClipboardContent CLIPBOARD_CONTENT
- Card aCard
- ImageView aImageView
- «Create» CardDragHandler(ImageView)
- void setCard(Card)
- void handle(MouseEvent)

Main
- int WIDTH
- int HEIGHT
- int MARGIN_OUTER
- String TITLE
- String VERSION
- boolean isSoundPlaying
- SoundPlayer playMusic
- SoundPlayer gameUpSound
- SoundPlayer buttonSound
- SoundPlayer homePageSound
- «Create» Main()
- void main(String[])
- void start(Stage)

Package: cards

Iterable «interface»

BaseStack
- List<Card> aCards
- «Create» BaseStack()
- void push(Card)
- Card pop()
- Card peek()
- boolean isEmpty()
- void setEmpty()
- void setCards(List<Card>)
- List<Card> getCards()

Rank «enum»
- ACE
- TWO
- THREE
- FOUR
- FIVE
- SIX
- SEVEN
- EIGHT
- NINE
- TEN
- JACK
- QUEEN
- KING

Suit «enum»
- CLUBS
- DIAMONDS
- HEARTS
- SPADES
- boolean sameColor...

CardStack
- «Create» CardStack()
- «Create» CardStack(Iterable<Card>)
- void push(Card)
- Card pop()
- Card peek()
- Card peek(int)
- int size()
- boolean isEmpty()
- void clear()
- List<Card> getCards()
- void setCards(List<Card>)

CardImage
- Image CARD_BACK
- Map<Card,Image> CARD_IMAGES
- «Create» CardImage()
- Image imageFor(Card)
- Image imageForBackOf(Card)
- Image loadBackImage()
- Map<Card,Image> loadCardImages()
- String getFileNameFor(Card)

Deck
- «Create» Deck()
- void shuffle()
- void push(Card)
- Card pop()
- boolean isEmpty()
- List<Card> getCards(List<Card>)

Card
- Map<Suit,Map<Rank,Card>> CARDS
- Rank aRank
- Suit aSuit
- «Create» Card(Rank,Suit)
- Rank getRank()
- Suit getSuit()
- String toString()
- String titleCase(Enum<?>)

1. Package cards

  1.1.   Class Card

    1.1.1.   Fields

- static final Map<Suit, Map<Rank, Card>> CARDS	A map of Card objects keyed by Suit and then by Rank.
- final Rank aRank	The rank of the card.
- final Suit aSuit	The suit of the card.

    1.1.2.   Constructor

- Card(Rank pRank , Suit pSuit)	Creates a new Card object with the specified Rank and Suit.

    1.1.3.   Methods

+ static Card get(Rank pRank, Suit pSuit)	return a Card object from a collection based on the provided Rank and Suit
+ Rank getARank()	Returns the rank of the card.
+ Suit getASuit()	Returns the suit of the card.
+ String toString()	Returns a string representation of the card in the format "Rank of Suit" (e.g., "Ace of Hearts").
- static String titleCase(Enum<?> pEnum)	A helper method to convert an enum value to title case (e.g., "ACE" becomes "Ace").

Static Block: Initializes the CARDS map with all possible combinations of Rank and Suit, creating Card objects for each.

  1.2.   Class BaseStack

    1.2.1.   Fields

# List<Card> aCards	A list of card of a stack

    1.2.2.   Constructor

+ BaseStack()	Initialize a new ArrayList

1.2.3. Methods

+ void push(Card pCard)	Add pCard to aCards if pCard is not equal to null
+ Card pop()	Remove a top card of a stack and return a new top card of the stack
+ boolean isEmpty()	Return true is the stack is empty and return false if it is not
+ void setACards(List<Card>)	Set aCards list
+ List<Card> getACards()	Return aCards list

1.3. Class CardSerializer

1.3.1. Fields

- static final String SEPARATOR	A separator string used to separate serialized card representations.

1.3.2. Constructor

- CardSerializer()	A private constructor

1.3.3. Methods

+ static String serialize(Card pCard)	Serializes a single card into a string representation.
+ static String serialize(CardStack pCards)	Serializes a cardStack into a string representation.
+ static CardStack deserialize(String pCards)	Deserializes a card stack from a string representation.
+ static Card deserializeBottomCard(String pCards)	Deserializes a card stack from a string representation and returns the top card of the stack.
- static String cardToInt(Card pCard)	A helper method to convert a card to an integer representation.

<u>- static Card intToCard(int pNumber)</u>	A helper method to convert an integer representation to a card.

1.4.   Class CardStack extends BaseStack implements Iterable<Card>

1.4.1.   Constructor

+ CardStack()	Initializes the CardStack
+ CardStack(Iterable<Card> pCards)	Initializes the CardStack with the cards provided in the iterable pCards, adding them from bottom to top.

1.4.2.   Methods

+ void push(Card pCard)	Adds a card to the top of the stack. Preconditions: pCard must not be null and must not already be in the stack.
+ Card pop()	Removes and returns the card at the top of the stack. Precondition: The stack must not be empty.
+ Card peek()	Returns the card at the top of the stack without removing it from the stack. Precondition: The stack must not be empty.
+ Card peek(int pIndex)	Returns the card at the specified index pIndex within the stack without removing it. Preconditions: pIndex must be between 0 and the size of the stack minus one.
+ int size()	Returns the number of cards in the stack.
+ void clear()	Removes all cards from the stack.
+ boolean isEmpty()	Checks whether the stack is empty and returns true if it is, otherwise false.
+ String toString()	Provides a string representation of the stack in the format of a list of cards.
+ Iterator<Card> iterator()	Returns an iterator over the cards in the stack, from the bottom of the stack to the top.
+ void setACards(List<Card>)	Set aCards list
+ List<Card> getACards()	Return aCards list

1.5.   Class Deck

1.5.1.   Constructor

+ Deck()	Creates a new deck of 52 cards and shuffles them immediately.

1.5.2.   Methods

+ void shuffle()	Reinitializes the deck with all 52 standard playing cards, and shuffles them randomly.
+ void push(Card pCard)	Places a card on top of the deck.
+ Card push()	Draws and removes the top card from the deck.
+ boolean isEmpty()	returns true if there are no cards left in the deck; otherwise, returns false.
+ void setACards(List<Card>)	Set aCards list
+ List<Card> getACards()	Return aCards list

1.6.   Enum Rank

Represents the rank of a playing card.

ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING

1.7.   Enum Suit

Represents the suit of a playing card.

CLUBS, DIAMONDS, HEARTS, SPADES

Method

+ boolean sameColorAs(Suit pSuit)	return True if this suit and pSuit are of the same color.

2.   Package gui

2.1.   Class CardDragHandler implements EventHandler<MouseEvent>

2.1.1.   Fields

- static final ClipboardContent CLIPBOARD_CONTENT	A static clipboard content object used to hold the string representation of the card being dragged.
- Card aCard	object that represents the card being dragged.

- ImageView aImageView	object that displays the graphical representation of the card being dragged.

2.1.2.   Constructor

~ CardDragHandler(ImageView pView)	creates a new CardDragHandler object and initializes the aImageView variable with the provided ImageView object.

2.1.3.   Methods

~ setCard(Card pCard)	sets the aCard variable to the provided Card object.
+ void handle(MouseEvent pMouseEvent)	handles the drag event for the CardDragHandler object. This method creates a Dragboard object, sets its content to the string representation of the card being dragged, and sets the drag view to the image of the card. The method then consumes the mouse event to prevent it from being handled by any other event handlers.

2.2.   Class CardImage
2.2.1.   Fields

- Image CARD_BACK	The image for the back of a card
- static Map<Card, Image> CARD_IMAGES	A map that stores images for each cards

2.2.2.   Constructor

- CardImage()	Initializes the CardImage

2.2.3.   Methods

+ static Image imageFor(Card pCard)	Returns the image of a specified card
+ static Image imageForBackOfCard()	Returns the image for the back of a card
- static Image loadBackImage()	Loads the image for the back of the card
- static Map<Card, Image> loadCardImages()	Loads images for all 52 cards in a map.

- static String getFileNameFor(Card pCard)	Generates the file name for a card image based in its rank and suit

2.3. Class CardPileView extends StackPane implements GameModelListener

2.3.1. Fields

- static final int PADDING	The padding between the cards and the edge of the component.
- static final int Y_OFFSET	The vertical offset between each card in the stack.
- static final ClipboardContent CLIPBOARD_CONTENT	The clipboard content used for transferring card data between piles.
- TableauPile aIndex	The index of the tableau pile being displayed.
- final GameModel aModel	The game model that the component is listening to for state changes.
- SoundPlayer cardSound	An instance of the SoundPlayer class for playing card sounds.

2.3.2. Constructor

~ CardPileView(GameModel pModel, TableauPile pIndex)	Constructs a new CardPileView component with the given game model and tableau pile index.

2.3.3. Methods

- Image getImage(Card pCard)	Returns the Image object for the given Card object.
- void buildLayout()	Builds the layout of the component by adding ImageView objects for each card in the tableau pile.
- EventHandler<MouseEvent> createDragDetectedHandler(final ImageView pImageView, final Card pCard)	Creates a drag detected event handler for the given ImageView and Card objects.
- EventHandler<DragEvent> createDragOverHandler(final ImageView pImageView, final Card pCard)	Creates a drag over event handler for the given ImageView and Card objects.

- EventHandler<DragEvent> createDragEnteredHandler(final ImageView pImageView, final Card pCard)	Creates a drag entered event handler for the given ImageView and Card objects.
- EventHandler<DragEvent> createDragExitedHandler(final ImageView pImageView, final Card pCard)	Creates a drag exited event handler for the given ImageView and Card objects
- EventHandler<DragEvent> createDragDroppedHandler(final ImageView pImageView, final Card pCard)	Creates a drag dropped event handler for the given ImageView and Card objects
+ void gameStateChanged()	Called when the game state changes. Rebuilds the layout of the component.

2.4.   Class DeckView extends HBox implements GameModelListener

2.4.1.   Fields

- static final String BUTTON_STYLE_NORMAL	The style for the button in its normal state.
- static final String BUTTON_STYLE_PRESSED	The style for the button when it is pressed.
- static final int IMAGE_NEW_LINE_WIDTH	The width of the border around the reset image.
- static final int IMAGE_FONT_SIZE	The font size for the reset image text.
- final GameModel aModel	The game model that the component is listening to for state changes.
- SoundPlayer cardSound	An instance of the SoundPlayer class for playing card sounds.

2.4.2.   Constructor

~ DeckView(GameModel pModel)	Constructs a new DeckView component with the given game model.

2.4.3.   Methods

- Canvas createNewGameImage()	Creates a new game image that displays the "Reset Deck" button when the game is completed.

+ void gameStateChanged()	Called when the game state changes. Updates the component to show the deck or the "Reset Deck" button.
+ void reset()	Resets the deck and makes it visible

2.5. Class DiscardPileView extends HBox implements GameModelListener

2.5.1. Fields

- static final int PADDING	The padding between the edge of the component and the card image.
- CardDragHandler aDragHandler	The card drag handler for the top card of the discard pile.
- final GameModel aModel	The game model that the component is listening to for state changes.

2.5.2. Constructor

~ DiscardPileView(GameModel pModel)	Constructs a new DiscardPileView component with the given game model.

2.5.3. Methods

+ void gameStateChanged()	Called when the game state changes. Updates the component to show the top card of the discard pile or hide it if the pile is empty.

2.6. Class Main

2.6.1. Fields

- static final int WIDTH	Initializes width to 700
- static final int HEIGHT	Initializes height to 600
- static final int MARGIN_OUTER	Initializes margin outer to 10
- static final String TITLE	Initializes title to "Solitaire"
- static final String VERSION	Initializes version to "8.8.8"

- boolean soundPlaying	Initializes soundPlaying to true to check if the game sound is playing or not
- SoundPlayer gameMusic	Initializes new SoundPlayer
- SoundPlayer giveUpSound	Initializes new SoundPlayer
- SoundPlayer buttonSound	Initializes new SoundPlayer
- SoundPlayer homePageSound	Initializes new SoundPlayer

2.6.2. Constructor

+ Main()	Initializes Main

2.6.3. Methods

<u>+ static void main(String[] pArgs)</u>	Launch the JavaFX application
+ void start(Stage pPrimaryStage)	- Set homePageSound volume to 1.00 - Set homePageSound to be loop - Start playing homePageSound - Set title of pPrimaryStage to TITLE + VERSION - Create homepageLayout and set background of homepageLayout with WIDTH and HEIGHT field - Set spacing of homepageLayout to 10 - Set padding of homepageLayout to 20 - Set alignment of homepageLayout to center - Create startButton and logo image - Add function to startButton when click to change scene from homepageLayout to root and change sound from homePageSound to gameMusic - Add created button and logo to homepageLayout - show homepageLayout - Create root layout with GridPane and set background to file background.jpg - Set Hgap, Vgap and Padding of root to MARGIN_OUTER - Create new Gamemodel as model - Create new Deckview with model - Create DiscradPileView with model

	- Create newGameButton, soundButton and undoButton - Set style and image to newGameButton, soundButton and undoButton - Add newGameButton, soundButton, undoButton, deckView and discardPileView to root - Create new SuitStack and CardPileView with loop - Add SuitStack and CardPileView to root - Add function when releasing newGameButton to play giveUpSound and gameMusic then call model.reset() to reset game - Add function when clicking newGameButton to wait 2.5 second after called then stop playing giveUpSound and play gameMusic - Add function when not hover newGameButton to show text "New Game" -Add function when hover the newGameButton if the player finished the game show the text "again", if not then show the text "don't give up" - Add function to soundButton when press to stop/play the sound - Add function to undoButton when click to undo last move - Add effect hover style to undoButton, startButton, soundButton and newGameButton

2.7.   Class SoundPlayer

2.7.1.   Fields

- String musicFile	Path of the resource sound
- Media sound	Media of the sound
- MediaPlayer mediaPlayer	Media player
- boolean playingStatus	Checking if the sound is playing or not

2.7.2.   Constructor

+ SoundPlayer(String musicFileURL)	Set musicFile to musicFileURL Set sound with musicFile Set playingStatus to false

	And Set mediaPlayer with sound

2.7.3. Methods

+ void setLoop()	Set mediaPlayer to playing with loop
+ void startPlay()	Set playingStatus to true and make the mediaPlayer playing
+ void stopPlay()	Set playingStatus to false and make the mediaPlayer stop playing
+ void setVolume(double volume)	Set volume to mediaPlayer
+ void setPlayingStatus(boolean status)	Set playing status
+ boolean isPlaying()	return true if the mediaPlayer is playing and return false if the mediaPlayer isn't playing

2.8. Class SuitStack extends StackPane implements GameModelListener

2.8.1. Fields

- static final int PADDING	The padding for SuitStack
- static final String BORDER_STYLE	The styles for border of the SuitStack
- static final String BORDER_STYLE_DRAGGED	The styles for dragged state of the SuitStack
- static final String BORDER_STYLE_NORMAL	The styles for normal state of the SuitStack
- CardDragHandler aDragHandler	CardDragHandler for representing the card dragged
- FoundationPile aIndex	Location of the FoundationPile
- final GameModel aModel	GameModel for tracking the current state of the game
- SoundPlayer cardSound	An instance of the SoundPlayer class for playing card sounds.

2.8.2. Constructor

~ SuitStack(GameModel pModel, FoundationPile pIndex)	Constructs a SuitStack with the given GameModel and FoundationPile

2.8.3.   Methods

+ void gameStateChanged()	Updates the state of the SuitStack based on changes in the game model
- EventHandler<DragEvent> createOnDragOverHandler(final ImageView pView)	Event handlers for drag and drop functionality
- EventHandler<DragEvent> createOnDragEnteredHandler()	Event handlers for drag and drop functionality
- EventHandler<DragEvent> createOnDragExitedHandler()	Event handlers for drag and drop functionality
- EventHandler<DragEvent> createOnDragDroppedHandler()	Event handlers for drag and drop functionality

3.   Package model

    3.1.   Class CompositeMove implements Move

        3.1.1.   Fields

- final List<Move> aMoves	The list of individual moves that are part of the composite move.

        3.1.2.   Constructor

+ CompositeMove(Move ... pMoves)	Constructs a new CompositeMove object with the given individual moves.

        3.1.3.   Methods

+ void perform()	Performs all the individual moves in the composite move.
+ void undo()	Undoes all the individual moves in the composite move in reverse order.

    3.2.   Enum FoundationPile implements Location

3.3. Class Foundation

3.3.1. Fields

- final Map<FoundationPile, CardStack> aPiles	The map of foundation piles, where each key is a FoundationPile object and each value is a CardStack object.

3.3.2. Constructor

+ Foundation()	Constructs a new Foundation object with four empty piles.

3.3.3. Methods

+ int getTotalSize()	Returns the total number of cards in all the foundation piles.
+ void initialize()	Initializes the Foundation object to reset it to four empty piles.
+ boolean isEmpty(FoundationPile pLocation)	Returns true if the pile at pLocation is empty.
+ boolean canMoveTo(Card pCard, FoundationPile pLocation)	Returns true if pCard can be moved to the top of pLocation.
+ Card peek(FoundationPile pLocation)	Returns the card on top of the pile at pLocation.
+ void push(Card pCard, FoundationPile pLocation)	Place pCard onto the pile at pLocation.
+ Card pop(FoundationPile pLocation)	Removes the card at the top of the pile at pLocation and returns it.

3.4. Class GameModel implements GameModelView

3.4.1. Fields

- static final Move NULL_MOVE	Move interface that isNull() return true and does nothing
- final Move aDiscardMove	Defines Move interface and set actions for discard card

- final Deck aDeck	A Deck object representing the deck of cards.
- final Stack<Move> aMoves	A Stack of Move objects representing the moves made in the game.
- final CardStack aDiscard	A CardStack object representing the discard pile.
- final Foundation aFoundation	A Foundation object representing the foundation piles.
- final Tableau aTableau	A Tableau object representing the tableau piles.
- final List<GameModelListener> aListeners	A List of GameModelListener objects representing the listeners that are registered to receive notifications about changes to the game state.

3.4.2.   Constructor

+ GameModel()	Creates a new GameModel object initialized to a new game.

3.4.3.   Methods

+ int getScore()	Returns the number of cards in the foundations
+ void addListener(GameModelListener pListener)	Registers an observer for the state of the game model
- void notifyListeners()	Notifies an observer for the state of the game model
+ void reset()	Restores the model to the state corresponding to the start of a new game
+ boolean isCompleted()	Checks if the game is completed or not
+ boolean isDeckEmpty()	Checks if the deck is empty or not
+ boolean isDiscardPileEmpty()	Checks if the discard pile is empty or not
+ boolean isFoundationPileEmpty(FoundationPile pPile)	Checks if the foundation pile is empty or not
+ Card peekSuitStack(FoundationPile pPile)	Obtains the card on top of the foundation pile without removing it
+ Card peekDiscardPile()	Obtains the card on top of the discard pile without removing it

- Location find(Card pCard)	Returns the game location where this card currently is
+ void undoLast()	Undoes the last move
+ boolean canUndo()	Checks if there is a move to undo
- void absorbCard(Location pLocation)	Removes the moveable card from specified location
- void move(Card pCard, Location pDestination)	Moves the card to specified location
+ CardStack getDiscard()	Gets current discard pile
+ Deck getDeck()	Gets current deck
+ void resetDeck()	Reset the deck back to the first index of remaining discard pile
+ CardStack getTableauPile(TableauPile pIndex)	Gets the card pile at specified index
+ boolean isVisibleInTableau(Card pCard)	Checks if a card is visible in the tableau
+ boolean isLowestVisibleInTableau(Card pCard)	Checks if a card is the lowest visible in the tableau
+ CardStack getSubStack(Card pCard, TableauPile pPile)	Retrieves a sequence of cards from the tableau
+ boolean isLegalMove(Card pCard, Location pDestination)	Checks if the move is legal
+ Move getNullMove()	Retrieves the null move
+ Move getDiscardMove()	Retrieves the discard move
+ Move getCardMove(Card pCard, Location pDestination)	Retrieves a move for moving a card to a destination
+ boolean isBottomKing(Card pCard)	Checks if a card is the bottom king in the tableau

3.4.4.   Private Class CardMove

3.4.4.1.   Fields

- Card aCard	The selected card
- Location aOrigin	The location before moving the card

- Location aDestination	The location where the card will be placed

3.4.4.2. Constructor

+ CardMove(Card pCard, Location pDestination)	Initializes the CardMove with provided information

3.4.4.3. Methods

+ void perform()	Action for moving the card
+ void undo()	Action for undoing the moved card

3.4.5. Private Class RevealTopMove
3.4.5.1. Fields

- final TableauPile aIndex	Holds a reference to a specific tableau pile from which the top card will be revealed or hidden.

3.4.5.2. Constructor

+ RevealTopMove(TableauPile pIndex)	Initializes a new instance of RevealTopMove with a specific tableau pile.

3.4.5.3. Methods

+ void perform()	Action for revealing the top card of the stack
+ void undo()	Action for undoing after revealed the top card of the stack

3.5. Interface GameModelListener
3.5.1. Methods

+ void gameStateChanged()	Called whenever the state of the game model changes.

3.6. Interface GameModelView

3.6.1. Methods

+ boolean isDiscardPileEmpty()	Returns true if the discard pile has no card in it.
+ boolean isDeckEmpty()	Returns true if the deck has no card left in it.
+ boolean isFoundationPileEmpty(FoundationPile pIndex)	Returns true if the suit stack for the given index is empty.
+ Card peekDiscardPile()	Return the card on top of the discard pile.
+ CardStack getTableauPile(TableauPile pIndex)	Returns a copy of the stack at the given index.
+ boolean isVisibleInTableau(Card pCard)	Returns true if the card's value is visible in the tableau.
+ boolean isLowestVisibleInTableau(Card pCard)	Returns true if the card is in the tableau and is the lowest card visible in its pile, including the case where it's the only card in the pile.
+ boolean isBottomKing(Card pCard)	Returns true if the card is a visible king located at the bottom of the pile.
+ boolean isLegalMove(Card pCard, Location pDestination)	Determines if the given card can be moved to the given destination according to the rules of the game and given the current game state.
+ Move getNullMove()	Returns an instance of the Move interface that represents no move at all.
+ Move getDiscardMove()	Returns an instance of the Move interface that represents discarding a card from the deck.
+ Move getCardMove(Card pCard, Location pDestination)	Returns an instance of the Move interface that represents moving the given card (and possibly all cards below it if applicable) to the given destination.

3.7. Interface Location

3.8. Interface Move

3.8.1. Methods

+ void perform()	Performs the move.

+ void undo()	Undoes the move by reversing its effect.
+ boolean isNull()	Returns true if the move is not a move that advances the game. Returns false by default.

3.9. Enum OtherLocation implements Location

DISCARD_PILE

3.10. Class Tableau
3.10.1. Fields

- final Map<TableauPile, CardStack> aPiles	A Map of CardStack objects representing the piles in the tableau.
- final Set<Card> aVisible	A Set of Card objects representing the visible cards in the tableau.

3.10.2. Constructor

+ Tableau()	Creates an empty tableau.

3.10.3. Methods

~ initialize(Deck pDeck)	Fills the tableau by drawing cards from the given deck.
~ boolean canMoveTo(Card pCard, TableauPile pPile)	Determines if it is legal to move the given card to the given pile.
+ boolean isBottomKing(Card pCard)	Returns true if the given card is a king at the bottom of a pile, and false otherwise.
~ CardStack getPile(TableauPile pPile)	Returns a copy of the pile at the specified position in the tableau.
- TableauPile getPile(Card pCard)	returns the TableauPile index of the pile that contains the given card.
~ boolean revealsTop(Card pCard)	Returns true if moving the given card away reveals the top of the card, and false otherwise.

- Optional<Card> getPreviousCard(Card pCard)	returns the previous card to the given card in the same pile.
~ void moveWithin(Card pCard, TableauPile pOrigin, TableauPile pDestination)	Moves the given card and all the cards below it to the given destination pile.
~ CardStack getSequence(Card pCard, TableauPile pPile)	Returns a copy of the sequence of cards starting at the given card and including all cards on top of it.
~ void showTop(TableauPile pIndex)	Makes the top card of the given pile visible.
~ void hideTop(TableauPile pIndex)	Makes the top card of the given pile not visible.
~ boolean contains(Card pCard, TableauPile pIndex)	Returns true if the given pile contains the given card, and false otherwise.
~ boolean contains(Card pCard)	Returns true if any pile contains the given card, and false otherwise.
~ boolean isVisible(Card pCard)	Returns true if the given card is visible, and false otherwise.
~ boolean isLowestVisible(Card pCard)	Returns true if the given card is visible and there is no visible card below it in its pile, and false otherwise.
~ void pop(TableauPile pIndex)	Removes the top card from the given pile.
~ void push(Card pCard, TableauPile pIndex)	Place the given card on top of the given pile.

3.11.   Enum TableauPile implements Location

FIRST, SECOND, THIRD, FOURTH, FIFTH, SIXTH, SEVENTH