Model Diagram Description

**Hanabi Game**

**Description:** Hanabi Game is a public class that is responsible for storing and manipulating the game state.

**Fields:**

* int numPlayers: Integer representing the number of players when game is launched.
* Player players: Player objects, each representing a unique player.
* DiscardPile discardPile: DiscardPile container to store discared cards during the game.
* FireWorksPile fireworks: FireWorksPile container to store the stacks of cards played.
* Token fuseTokens: Token object to keep track of fuse tokens.
* Token informationTokens: Token object to keep track of information tokens.
* Log log: Log container to store all actions throughout the game.
* int playerTurn: Integer representing which players turn it is.
* bool DiscardVisible: Boolean to represent whether or not the player has selected the discarded card pile.
* bool logVisible: Boolean to represent whether or not the player has selected the game log.

**Methods:**

* void addPlayer(): Adds player to the Hanabi game.
* void removePlayer(): Removes player from Hanabi game.
* void startGame(hands): Method to initialize the Hanabi game.
  + param hands:a 2d list of strings to represent a player and his/her hand accordingly.
* void playCard(int handIndex, String drawColor, int drawRank): Method to play a certain players card.
  + param handIndex: Integer representing the current players hand.
  + param drawColor: String representing the color of the current players card that is to be played.
  + param drawRank: Integer representing the rank of the current players card that is to be played.
* void discardCard(int handIndex, String drawColor, int drawRank): Method to discard a certain players card.
  + param handIndex: Integer representing the hand of the current player.
  + param drawColor: String representing the color of the current players card that is to be discarded.
  + param drawRank: Integer representing the rank of the current players card that is to be discarded.
* void giveInfo(int playerIndex, String property): Method that allows player to give information to another player
  + param playerIndex: Integer representing the player in the array of players
  + param property: String representing the property by which to give info (rank or color)
* void toggleDiscradView(): Public Method that allows player to all cards that have been discarded during the game.
* void toggleLogView(): Method that allows player to view the log.
* void addActionToLog(String move, String[\*] params): Method to add log actions to log collection.
  + param move: The move that was completed (give info, play card, discard card)
  + param params: String array to represent the description of a move.
* void endgame(): Method to end the game and perform all necessary actions.

**Class Card**

**Description:** Card is a public class that is responsible for holding information for Rank and Color of a given card. It sets respective values for each card at game launch and provides access to them throughout the game.

**Fields:**

* int rank: Integer data member of Card that describes the value rank of the given card.
* String colour: String data member of Card that describes the color type of the given card.
* boolean rankKnown: Keep track if a Card’s rank is known to the player.
* boolean colourKnown: Keep track if a Card’s colour is known to the player.

**Methods:**

* int getRank(): Returns an integer value that represents the rank of a given card.
* String getColour(): Returns a string that represents the colour of a given card.
* Boolean[] getInfo(): Returns a list that contains rankKnown and colourKnown.

**Class Hand**

**Description:** Hand is a public class that is responsible for holding information for the cards for a given player.

**Fields:**

* Card[4-5] cards: Collection of cards for a player. Its size depends on the number of players in the game, thus either of size four or five.
* int size: Integer value representing the size of the hand.

**Methods:**

* int getSize(): Returns an integer value representing the amount of cards in the hand.
* Card[4-5] getCards(): Returns the collection of cards.
* Card getCard(int i): Returns specific card from the collection of cards.
* void addCard(Card c, int i): Adds a Card to the collection of cards at a specific index.
* void removeCard(int i): Removes a Card from the collection, from specific index.

**Class DiscardPile**

**Description:** Public class that contains a collection of cards that have been discarded. It provides access to getting the collection, and adding/removing cards to it.

**Fields:**

* Card[\*] cards: Collection of cards, representing the discarded cards.

**Methods:**

* Card[\*] getCards(): Returns the collection of discarded cards.
* void addCard(Card c): Add a new card to the discard pile collection.
* void clear(): Empty the collection of cards in the discard pile.

**Class Token**

**Description:** Public class that handles tokens. Their count, adding and removing tokens.

**Fields:**

* int numTokens: Integer value representing the total number of tokens.

**Methods:**

* int getCount(): Returns integer value representing the number of tokens.
* void addToken(): Increases the number of tokens by one.
* void removeToken(): Decreases the number of tokens by one.

**Class Player**

**Description:** Public class that represents specific player.

**Fields:**

* Hand hand: Hand reference that contains information for the cards of a player.

**Methods:**

* Hand getHand(): Returns hand information of the current player.

**Class Log**

**Description:** Public class that contains a collection of actions. These actions describe what moves have been made by players.

**Fields:**

* Action[\*] actions: Collection of actions.

**Methods:**

* Action[\*] getActions(): Returns the collection of existing actions.
* void addAction(Action act): Add a new action reference to the collection of existing actions.
* void clear(): Clear the collection of actions.

**Class Action**

**Description:** Public class that handles what moves have been made and by whom.

**Fields:**

* int actingPlayer: Integer value that represents a specific player.
* String moveType: Represents the type of move being made – Play/Discard/Give Information.
* String[\*] moveParams: Describes everything about a move. Which player has performed what type of move – for example play/discard and what card, or information given.

**Methods:**

* int getPlayer(): Returns the id of the current player.
* String getMoveType(): Returns the type of move made by the player.
* String[\*] getMoveParams(): Returns a full description of the move made by a player.

**Class FireWorksPile**

**Description:** FireWorksPile is a public class that is responsible for holding card objects to represent a fire works pile in the Hanabi game.

**Fields:**

* Cards[0-5] cards: A card array to represent a pile.
* String colour: String to represent the colour of the pile.

**Methods:**

* String getColour(): Method to get the colour of the fireworks pile. Returns a string that represents the colour of the pile.
* String getCards(): Method to get the cards in a fireworks pile. Returns a card array.
* void addCard(Card c): Method to add a card to the fireworks pile.
  + param c: the card object
* void clear(): clears the fireworks pile.

**Notes:**

* is the colour method in FireWorksPile needed? it is aggregating card which already has a colour field.