# TicTacToe and War

# 1. Description

Develop a GUI program that can switch between War and TicTacToe without Dialogue Boxes.

# 2. Specification

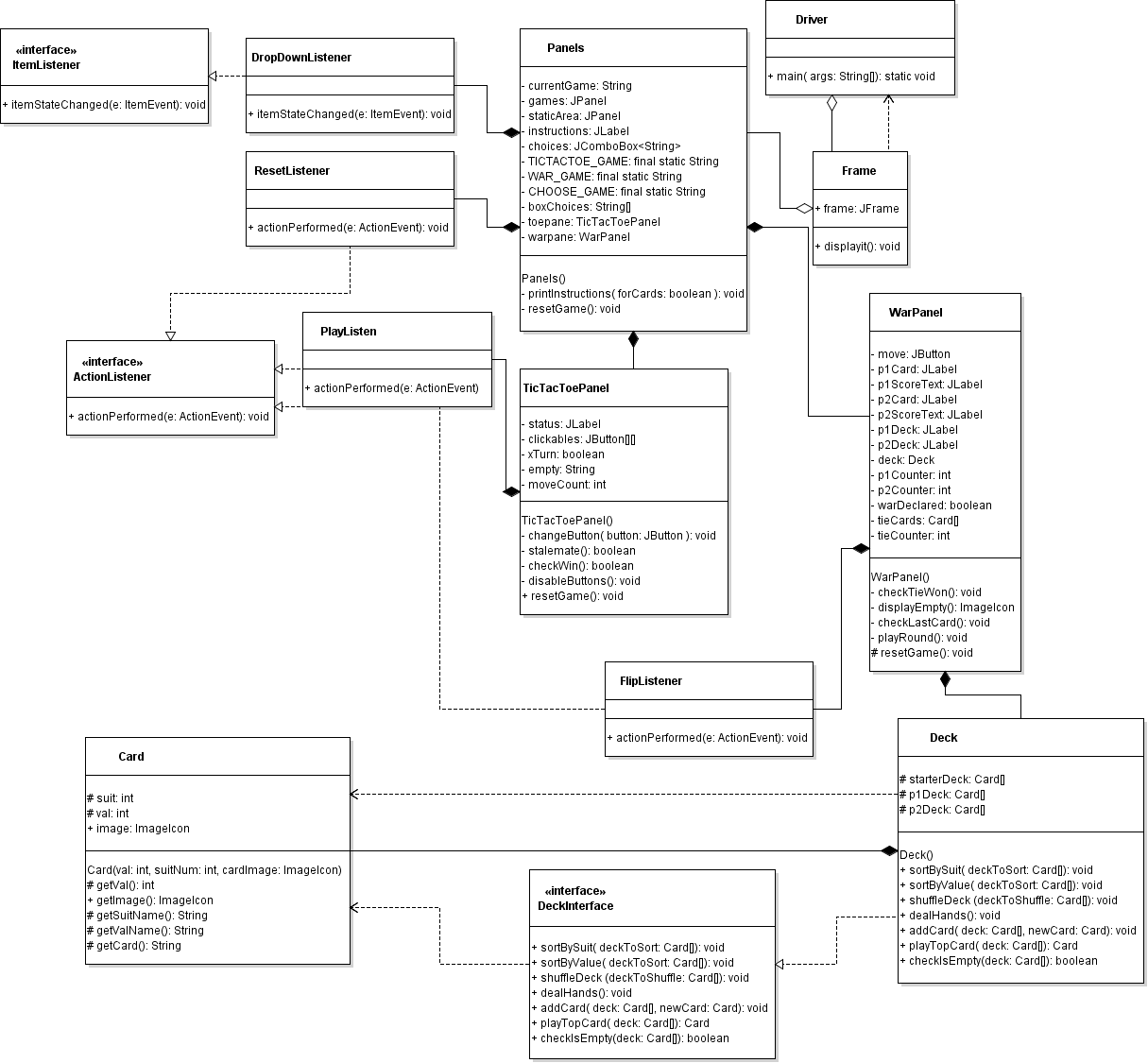
* GUI version
* Two Players
* Switching between two games in one app

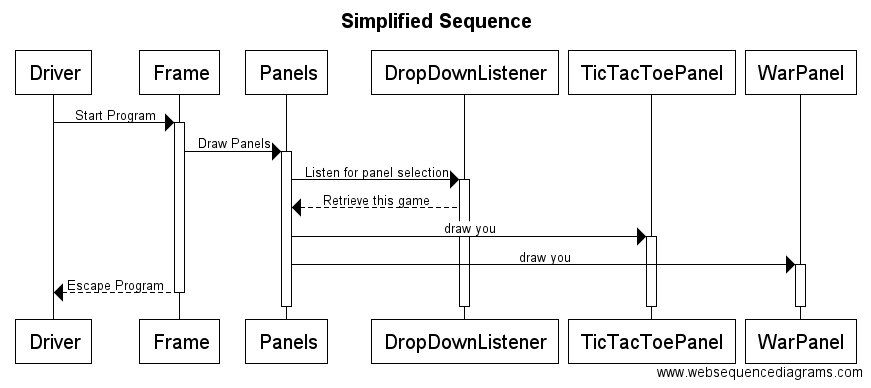
# 3. Dependencies

Program calls for java compiler and ability to run Java GUI.

Class Dependencies: Program has a near linear dependency. Driver calls the Frame, Frame calls the Panels which implements both TicTacToePanels and WarPanels. TicTacToePanels is a dead end, whereas WarPanels then uses Deck, which implements the cards. Panels contains the logic for playing and displaying the game as it goes on. Deck cannot function without cards (and has an interface). Deck COULD be used without a game loop due to its functionality.

# 4. Design





# 5. Test plan

[Table showing how each method/system function is tested]

|  |  |  |
| --- | --- | --- |
| Frame Class | | |
| Test# | Requirement | Test |
| 1. | Does Program run | Run program |
| 2. | Does Frame show correctly | Does bar read correctly and window match set dimensions |
| 3. | Does initial panel show | Does it> |
| 4. | Can you switch between panels | Change between each of dropdown menu option |
| 5. | Do instructions change to correct game | Change between dropdown menu options |
| 6. | Can x’s and o’s be place in TicTacToe | Click buttons |
| 7. | Does a full (non winning) board cause a stalemate | Message pops up and buttons disabled |
| 8. | Can either side win | Win with x’s, x win statement pops up. Win with o, o statement pops up. Both disable buttons |
| 9. | Does game reset on restart | Restart button clicked allows for new playthrough |
| 10 | Is game “saved” when switched over to another option | Switch to another option, switch back, game still there without changes |
| 11. | Restart button is only good for currently active game | Switch to another option, restart button, switch back, game still there without changes |
| 12. | Winning can be won in all 8 possible configurations at tic tac toe | Try winning 8 different ways. |
| WarPanel Class | | |
| Test# | Requirement | Test |
| 1. | Do the panels display correctly | Run program |
| 2. | | | Buttons use their listeners correctly | Flip button flips cards/increases card counts, restart resets game |
| 3. | Game can be won | When player has zero cards, other player has 52, win message appears |
| 4. | War works correctly | Ties allocate the correct amount of cards and counter ticks. |
| 5. | Correct counts are updated | When one person wins the hand they gain count while other loses. |
| Card Class | | |
| Test# | Requirement | Test |
| 1. | Make sure card can be created. | Call Card constructor. |
| 2. | Make sure value on card can be retrieved | getVal() returns this.val |
| 3. | Make sure suit name can be retrieved | getSuitName() returns String associated with the numeric Value of the card. |
| 4. | Can return name of card value | getValName() returns String with associated name. |
| 5. | Can return the full name of the card | getCard() returns the coordinating value and suit names for the card |
| 6. | ImageIcon matches the value and suit | Check that image visually represents the value and suit name that the card is set with |
| Deck Class/Deck Interface | | |
| Test# | Requirement | Test |
| 1. | Deck takes the deck interface | Deck includes interface methods |
| 2. | Starter deck has 52 slots. | starterDeck.length = 52, query above spits out error. |
| 3. | P1 and P2 decks have 52 available slots | Check that deck lengths = 52. |
| 4. | Deck can be shuffled | Check starterDeck with shuffle method, print out cards. |
| 5. | Check that all suits and numbers of a deck are completed | Print out a full deck, use sort methods with each other to make it easier to see what is there. |
| 6. | Check that once dealt, p1 and p2 decks only have 26 cards. | Check and see what the first null index is, if it is 26, there are 26 cards. |
| 7. | Check that decks p1 and p2 have different cards | Compare |
| 8. | Check that decks p1 and p2 make a full deck | Combine into one array, call sort methods and make sure no element is the same as another. |
| 9. | Check that adding a card increases the number of non null elements by 1 | Cycle through until null value found, if more than start length, good to go |
| 10. | Check that adding a card correctly puts it in the array at the index | Check new index and print out card, is it correct? |
| 11. | Check that when card is played, it is in index 0 | Print index 0, then play hand and print played card. |
| 12. | Check that non null elements decrease by 1. | Print out non null length by counter cycling through until null found and compare after playTopCard() has been called. |
| 13. | Check that when deck is empty | Check that if element 0 is empty, returns true via checkIsEmpty() |

# 6. Source Code

[Softcopy only]

# 7. JavaDoc

[zipped together with this document]

# Revisions

|  |  |
| --- | --- |
| Date | Brief description |
| Nov 18 | Started UML, started planning how to switch. Remembered CardLayout |
| Nov 19 | Wrote bulk of program |
| Nov 20 | Tweaked. Made solving function for tic tac toe better. Found minor bug that didn’t display moves correctly. Turned in. |
|  |  |
|  |  |