Object-Oriented Programming

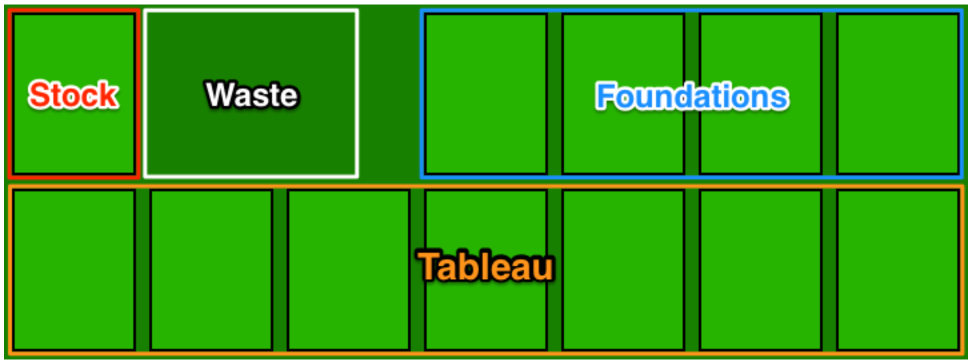
Project Statement

Academic Year 2015-2016

This project is individual and must be submitted to `blaugraud@ulg.ac.be` *for May 9th at the latest. Projects returned after the deadline will not be corrected. Plagiarism is not tolerated, in line with the policy of the university (*`http://www.ulg.ac.be/plagiat`*).*

Klondike solitaire is a single player game played with a deck of 52 cards on a board composed of

- the *stock*, a pile of hidden cards,
- the *waste*, a pile of upturned cards,
- the *foundations*, composed of 4 piles of up to 13 cards,
- the *tableau*, composed of 7 piles of cards.



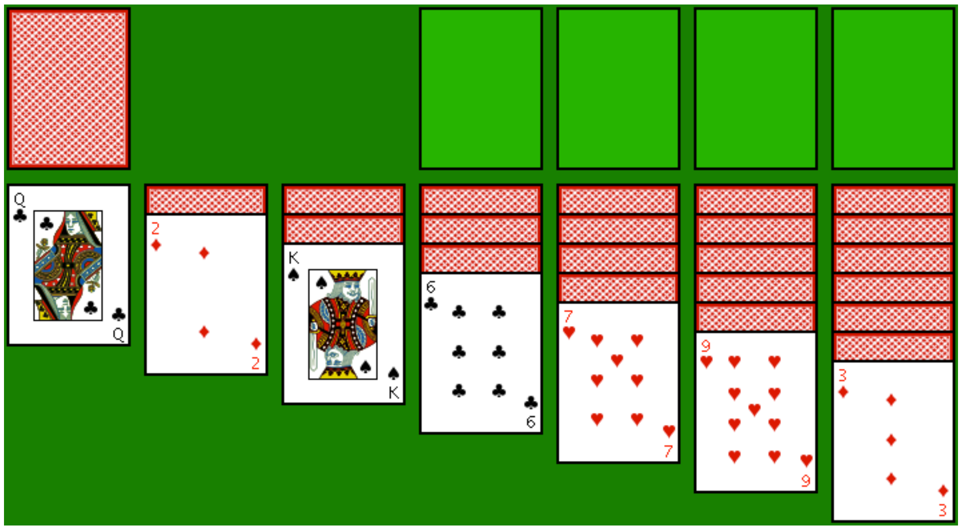
Components of the board

Rules In our version of Klondike, the following rules must be followed:

- The goal of the game is to complete the four foundations, in such a way that each of them contains a full sequence of cards of a unique suit from Ace to King. The suits assigned to foundations can be freely chosen by the player.
- The tableau piles are built by alternating red and black cards. A card can cover another one only if its value immediately precedes the one of the covered card. For instance, a Four of Hearts can be placed on top of either a Five of Spades or a Five of Clubs.
- The player is allowed to move a card from either the top of a foundation or the top of the waste onto a tableau pile.
- An upturned card or a pile of upturned cards can be moved from the top of a tableau pile onto another one.
- If a hidden card of a tableau pile becomes the top card after a move, then it is automatically upturned.
- An empty foundation can be filled with any Ace taken from the waste or from the top of a tableau pile.

- An empty tableau pile can be filled with a pile of upturned cards with a King as bottom card[1].
- Among the cards in the waste, only the three topmost ones are visible to the player. If the waste contains less than three cards, then it is entirely visible. Recall that only the top card of the waste can be moved.
- The remainder of the stock can be dealt by turning three cards at once to the waste. If less than three cards are remaining in the stock, all these cards are moved to the waste.
- When the stock is empty, the content of the waste can be transferred to the stock by turning the cards over.

New Game At the beginning of a new game, the deck of 52 cards is randomly shuffled, then 28 cards at the top of the deck are distributed among the piles composing the tableau, in increasing order of size: The first pile contains one card, the second two, and so on. In each pile, all cards are hidden except the topmost one. After the tableau piles have been created, the rest of the deck is used as the stock.



Example of initial game configuration

Statement The project consists in developing a Java program for playing Klondike solitaire through a graphical user interface *(GUI)*. In this program, the player must be able to undo its last move (up to the initial state of the game), as well as to quit the game or start a new game at any time. The program should always display the number of moves made by the player since the game has been started.

The GUI is to be handled by a library that will be made available to you (in other words, you do not have to program it by yourself). This library contains mechanisms for signaling that the player performs game actions, to which your program should respond in the appropriate way. For instance, when the player moves a card from the waste to the tableau, the program receives a notification, which should then trigger operations such as checking whether the move is valid, carrying out this move by updating the state of the piles according to the rules of the game, and refreshing the GUI display.

GUI Library The library discussed above is provided as a file called `graphics.jar`[2]. The classes contained in this file are only given as bytecode (in other words, their source code is not available). The documentation of this library is given in the appendix.

[1]As a particular case, such a pile may be composed of a single King.
[2]The technical details of this file format will be explained in an upcoming exercise session.

Mouse Events In order to manage the game, your program has to react to the following events:

- The player starts or quits the game.
- The player clicks on the stock in order to deal three cards, or to return the cards to the waste in the case of an empty stock.
- The player moves a card from the waste onto a tableau pile or a foundation.
- The player moves a card from a tableau pile onto a foundation.
- The player moves a card from a foundation to a tableau pile.
- The player moves cards between two tableau piles.
- The player undoes the last move.

The GUI is implemented in such a way that it is impossible to move a hidden card.

Program Organization Each class of your program should belong to the `be.ac.ulg.montefiore.oop` package. You are allowed to create new packages within this one, provided that they do not conflict with the `be.ac.ulg.montefiore.oop.graphics` package used by the GUI library. Finally, your program must implement the `main()` method in a public class called `Klondike`.

Program Execution You program should not expect to receive any runtime argument, and should return with an error message if some are provided.

Evaluation Criteria The evaluation of the project will take into account the fact that the program compiles successfully and is functionally correct, as well as the appropriate usage of object-oriented mechanisms, the structure and the readability of the source code. To avoid bad surprises, you are strongly recommended to check that your program compiles and works properly on the university computers *("Network 8")*.

Submission Rules Your program must be submitted using a **ZIP** archive named:

```
oop_lastname_firstname.zip
```

where `lastname` and `firstname` have been respectively replaced by your last and first names. This archive must be sent to `blaugraud@ulg.ac.be` in an email message with the subject:

```
OOP - Project 1 - 2016 - lastname firstname
```

.

At the root of the archive, you must place:

- An Apache Ant file `build.xml` for building the project.
- An empty `bin` folder in which the resulting bytecode should appear after a build.
- A `src` folder containing the source code of your program.
- The provided `graphics.jar` file.

Note Developing your own GUI instead of the provided library, or using any classes provided by other graphical libraries (such as Swing, AWT, JavaFX, . . .) is **prohibited**!

Appendix to the Project Statement

The GUI library is provided on the exercises website (`http://www.montefiore.ulg.ac.be/~blaugraud`), as a `graphics.jar` file to be included into your project. This GUI is able to render all the graphical elements of the game in a window, to handle mouse events, and to detect when the player closes the window.

Contents of the library The GUI library contains the following items, in addition to exception classes and internal components:

- `KlondikeEventsHandler:` An interface that must be implemented in order to react to player actions such as moving cards.
- `KlondikeSwingView:` A class that represents a graphical view of the game. It must be instantiated once by your program, passing to its constructor a reference to an object whose class implements `KlondikeEventsHandler`. This object will then automatically receive messages signaling that the player performs game actions.
- `KlondikeView:` An interface that must be employed for accessing the instance of `KlondikeSwingView`, in order to update the view displayed in the game window.

Interface constants The interfaces `KlondikeEventsHandler` and `KlondikeView` define constants that must be used for referring to card values and suits (a hidden card being assigned a special value), as well as to the locations of foundations and tableau piles:

-
  ```
  int SPADES_A;      // Ace of Spades
  int SPADES_2;      // Two of Spades
  ...
  int SPADES_10;     // Ten of Spades
  int SPADES_J;      // Jack of Spades
  int SPADES_Q;      // Queen of Spades
  int SPADES_K;      // King of Spades

  int HEARTS_A;      // Ace of Hearts
  ...
  int HEARTS_K;      // King of Hearts

  int DIAMONDS_A;    // Ace of Diamonds
  ...
  int DIAMONDS_K;    // King of Diamonds

  int CLUBS_A;       // Ace of Clubs
  ...
  int CLUBS_K;       // King of Clubs

  int HIDDEN;        // Hidden (face down) card
  ```

-
  ```
  int FOUNDATION_1;  // First (leftmost) foundation
  ...
  int FOUNDATION_4;  // Last (rightmost) foundation
  ```

-
  ```
  int TABLEAU_1;     // First (leftmost) tableau pile
  ...
  int TABLEAU_7;     // Last (rightmost) tableau pile
  ```

Documentation of the `KlondikeEventsHandler` interface The complete documentation of this interface can be found in the `KlondikeEventsHandler.java` file inside `graphics.jar`. The most relevant parts of this documentation are summarized here:

-
  ```
  void newGame();
  ```

  Signals that the player starts a new game.

-
  ```
  void clickStock();
  ```

  Signals that the player clicked on the stock in order to deal cards.

-
  ```
  void moveWasteToTableau(int tableau);
  ```

  Signals that the player moved a card from the waste to the tableau pile `tableau`. The argument `tableau` corresponds to a tableau pile constant.

-
  ```
  void moveWasteToFoundation(int foundation);
  ```

  Signals that the player moved a card from the waste to the foundation `foundation`. The argument `foundation` corresponds to a foundation constant.

-
  ```
  void moveTableauToFoundation(int tableau, int foundation);
  ```

  Signals that the player moved a card from the tableau pile `tableau` to the foundation `foundation`.

-
  ```
  void moveTableauToTableau(int tableauSrc, int numCards, int tableauDst);
  ```

  Signals that the player moved `numCards` card(s) from the tableau pile `tableauSrc` to another pile `tableauDst`.

-
  ```
  void moveFoundationToTableau(int foundation, int tableau);
  ```

  Signals that the player moved a card from the foundation `foundation` to the tableau pile `tableau`.

-
  ```
  void undo();
  ```

  Signals that the player wishes to undo its last action.

-
  ```
  String getName();
  ```

  When invoked, this method should return the name of the student implementing the project. This name will be credited in the "About" box.

Exceptions The methods above do not throw exceptions.

Documentation of the `KlondikeView` interface The complete documentation of this interface can be found in the `KlondikeView.java` file inside `graphics.jar`. The most relevant parts of this documentation are summarized here:

-
  ```
  void stockEmpty(final boolean isStockEmpty);
  ```

  Updates the state of the displayed stock. If `isStockEmpty` is true (resp. false), an empty frame (resp. a hidden card) will be drawn.

-
  ```
  void wasteCards(final int[] cards)
    throws NullArrayException, TooManyCardsException, UnknownCardException;
  ```

  Updates the state of the displayed waste. The new state is described by an array containing up to three cards constants, from bottom to top.

-
  ```
  void tableauCards(final int tableau, final int[] cards)
    throws UnknownTableauException, NullArrayException, TooManyCardsException,
           UnknownCardException;
  ```

  Updates the state of a displayed tableau pile, identified by a tableau pile constant. The new state is described by an array of cards constants, from bottom to top.

-
  ```
  void foundationCards(final int foundation, final int[] cards)
    throws UnknownFoundationException, NullArrayException, TooManyCardsException,
           UnknownCardException;
  ```

  Updates the state of a displayed foundation, identified by a foundation constant. The parameter `cards` describes the two topmost cards of the foundation (or less if the foundation does not have enough cards), from bottom to top. (Specifying the card underneath the top one is necessary, since it has to be displayed when the player moves the top card.)

-
  ```
  void updateMoves(final int moves)
    throws NegativeNumberException;
  ```

  Updates the number of moves made by the player since the beginning of the game. Note that this number must be decreased whenever a move is undone.

-
  ```
  void win();
  ```

  Announces to the player that the game has been won. After invoking this method, mouse events are ignored until the player starts a new game or presses the `Undo` button.

-
  ```
  void refreshWindow();
  ```

  Updates the graphical representation of the game according to all the modifications that have been performed since the last update. This method must be invoked in order for modifications to become visible to the player.

Exceptions The methods above and the constructor of the `KlondikeSwingView` class can throw the following exceptions:

Exception	**Context**
`NegativeNumberException`	The integer parameter is invalid.
`TooManyCardsException`	The array contains too many cards.
`NullArrayException`	The array parameter is `null`.
`NullHandlerException`	The handler parameter is `null`.
`UnknownCardException`	The integer card parameter is invalid.
`UnknownTableauException`	The tableau pile parameter is invalid.
`UnknownFoundationException`	The foundation parameter is invalid.