Solitaire is a popular card game that heavily utilizes the concept of stacks. It is not hard to see that stacks are formed when cards are placed on top of each other. For this homework assignment, we will imitate a game of Solitaire, called Stackotaire. You will be required to print a graphical display of the game board on the console, and, for extra credit, implement the game with GUI.

If you are unfamiliar with the game, here is an excellent website where you can try out the game.

General rules your game must enforce:

1. When the player draws from the stock pile, draw 1 card at a time.
2. In the tableau piles, all faced up cards can be placed under another faced up card that has a value of exactly 1 more than itself with an opposite colored suit, or a faced down card.
3. In the tableau piles, you can only move a 'K' to an empty pile.
4. After moving card(s) from a tableau pile, flip the top card face up if it is not.
5. In the foundation piles, cards should start at 'A', and subsequently stack cards with the same suits in order of value.

1. Write a fully-documented class named Card that contains the suit of the card (int), value of the card (int), and a boolean value isFaceUp. The following is a partial specification and it is up to you to fill in the details:

   public class Card

   • Constructor for Card (you may include a constructor with parameters)
     public Card()
   • set/get methods for suit and value
     include setters and getters for suit and value
   • isFaceUp
     public boolean isFaceUp()
   • setFaceUp
     public void setFaceUp(boolean faceUp)
   • isRed
     public boolean isRed() // odd values represents red, and even represents black. Look at hints below.
   • toString
     public String toString() // returns the card in the following format: [3♠]
HINT: You may use the following code to store values and suits of the cards. The value and suit data fields in each card should store the integer index corresponding to the following arrays.

```java
char suits[] = {' ', '\u2666', '\u2663', '\u2665', '\u2660'}; // {'
System.out.println("[" + values[cardValue]+suits[cardSuit] + "]"); // print a card
```

NOTE: The index position 0 contains a dummy value which will not be used.

2. Write a fully-documented class named CardStack that will represent a stack of cards in the game. It is recommended to extend the Stack API when implementing this class. We can keep track of the different types of stacks using a char variable type. The following stack types are: 's' - stock, 'w' - waste, 'f' - foundations, and 't' - tableau. Each stack has the following functionalities:

```java
public class CardStack

    • Constructor for CardStack
        public CardStack(char type)
    • push
        public void push(Card newCard)
        Adds a new Card on top of the Stack.
    • pop
        public Card pop()
        Removes the Card that is on top of the Stack
    • isEmpty
        public boolean isEmpty()
        Returns an indicator to whether the stack is empty or not
    • size
        public int size()
        Returns the number of cards in the stack
    • printStack
        public void printStack()
        Renders the stack visually, depending on the type of the stack.
        Type 's': Print the top card of the stack facing down - [XX]
        Type 'w': Print the top card of the stack facing up - [9♣]
Type 'f': Print the whole stack on a line - [XX][XX][XX][4♠][3♥][XX]
Type 't': Print the top card of the stack facing up - [A♥]

3. Write a fully-documented class named Stackotaire. This class will contain a main method that presents a menu that allows the user interact with the game UI. This will be a simple command line like UI where the player specifies the moves to play.

In terms of data storage, you should have an array of stacks that represent the tableau stacks, an array of stacks that represent the foundation stacks, and 2 individual stacks that represent the stock and waste stacks respectively. In addition, include a main stack that initially contains all 52 cards of a deck, though this stack is not used directly in the game. Use a double for-loop to create 52 distinct cards; these will be the cards you will use throughout the program.

There are methods that you should implement in addition to main:

- **initialize game**
  Initializes the game by distributing all cards into the proper stacks. First, move all 52 cards from the game stacks into the deck stack. Then, shuffle the deck by using the static method `shuffle()` in the Collections class (or you can implement your own). Finally, distribute the correct number of cards into the tableau and stock piles.

- **display game board**
  Renders all stacks to produce an image of the game board. Use the print stack method you have written to render each stack. The foundation stacks should be displayed on the top-left of the board; the waste stack should be displayed on the next to the foundation stacks on their right; the stock stack should be places right next to it on its right; a number should be displayed next to the stock stack that represents total cards left in the stack; the tableau stacks should be displayed horizontally under the previous stacks. Check the sample input/output for a visual understanding.
  
  *This method should be invoked after every move a player makes.*

- **check winning board**
  If all cards are faced up in the board, the player has the ability to manually move all cards to the foundation stacks. In other words, this is a guaranteed win. Iterate through the games stacks to determine the face value of all the cards. If all cards are faced up, display a win message, and prompt the user to play again.
  
  *HINT: Create another list to hold a shallow copy of all cards. This way, you don't have to check the 7 stacks individually.*

The player is able to interact with your game via a command line - like interface. When prompted for a move, the player can supply the move in any one of the following formats:
• draw
  Removes the top card from the stock pile and places it face up in the waste pile.
• move W1 T2
  Removes the top card from the waste pile, and places it on top of tableau pile #2. The third argument of this command can range from T1-T7, and F1-F4.
• move F1 T5
  Removes the top card from foundation pile #1, and places it on top of tableau pile #5. The second argument of this command can range from F1-14. The third argument of this command can range from T1-T7.
• move T1 F1
  Removes the top card from the tableau pile #1, and places it on top of foundation pile #1. The second argument of this command can range from T1-T7. The third argument of this command can range from F1-F4.
• moveN T3 T2 3
  Removes the top n cards, where n is the value of the fourth argument, from the tableau pile #3, and places them on top of tableau pile #2. The second and third arguments of this command can range from T1-T7.
• restart
  Prompts the player to end the game and start a new one. If the player chooses yes, initialize a new game board, else, continue with the current.
• quit
  Print a loss message and terminate the program.

HINT: Read the command one token at a time. If the command is draw, restart or quit, do as instructed. If the command is move, expect 2 more arguments. If the command is moveN, expect 3 more arguments.

4. Supply any exception handling class(es) that you need in addition to the classes above.

INPUT FORMAT

• All commands and pile labels are case-insensitive (e.g. 'move F1 t5' and 'MOVe f1 T5' are the same).
• You may assume that all input will be of the correct type. For example, if the input is supposed to be an integer, we will not test your program with a non-integer String instead.

EXTRA CREDIT

• (6 points) autoMove - your program should recognize all available cards that can be moved to a foundation pile. Automatically move these cards to the appropriate foundation pile without the user having to enter the "move" command. Display the commands for all cards that have been moved.
• (7 points) undo - you much create another stack that should record all the moves that the player has done so far. When this command is called, the previous move should be undone, and the game should return to the previous state.

• (up to 18 points) turnOnAI - computer auto plays the game! Your program should implement an artificial intelligence such that it recognizes all available moves while attempting to win the game. Be aware of move recycling!

• The best program in class receives up to 6 extra points and will be posted as the sample answer. (The author may remove his/her name before posting the program)

• Use GUI for all the user interface (input/output). [up to 12 points depending on how well you incorporate the GUI options]

NOTE 1: The "autoMove" command can be toggled on and off. Enter the command while it is already on to turn it off.

NOTE 2: If the AI gets stuck with no more available moves, turn of AI mode, and give the user a chance to undo. Continue the game accordingly.

NOTE 3: You will get maximum of 20 points extra credit if you complete both the GUI and the turnOnAI option.