Problem Set 2
Due: 6pm Tuesday, February 17

Due Date:
Because this assignment was posted late, its due date has been extended from Fri. Feb. 13 to Tue. Feb 17. However, you are encouraged to complete as much of the assignment as possible before Fri. Feb 13 since most help from instructors and tutors will be available between Tue. Feb 10 and Fri. Feb. 13. This will also give you more time to enjoy the long weekend.

Overview:
The purpose of this assignment is to give you practice with Java arrays, vectors, and loops (including nested loops). You will also get some practice with writing your own test cases and writing a class from scratch.

Reading:
- Read Chapters 10–12 of Downey’s online Java book (accessible from the Resource Links section of the CS230 home page).

Submission:
Each team should turn in a single hardcopy submission packet for all problems by slipping it under Lyn’s office door by 6pm on the due date. The packet should include:

1. a team header sheet (see the end of this assignment for the header sheet) indicating the time that you (and your partner, if you are working with one) spent on the parts of the assignment.
2. your final version of `SetTableau.java` from Problems 1 and 2.
3. a transcript of your tests from Problem 2.
4. your final version of `SetGameHistogram.java` from Problem 3.
5. a transcript of your histograms from Problem 3.

Each team should also submit a single softcopy (consisting of your final `ps2` directory) to the drop directory `~cs230/drop/ps2/username`, where `username` is the username of one of the team members (indicate which drop folder you used on your hardcopy header sheet). To do this, execute the following commands in Linux in the account of the team member being used to store the code.

```
cd /students/username/cs230
cp -R ps2 ~/cs230/drop/ps2/username/
```
Problem 0: The Game of Set

In this assignment, you will use the SetCard and SetHand classes you worked with in PS1 to implement a simple version of the Game of Set. Before starting the assignment, you should familiarize yourself with the rules of the game and some other classes by reading the rest of this problem.

a. : The Rules of CS230 Set

In this assignment, we will play a variant of the Game of Set that is somewhat simpler than the real game. We will call our variant CS230 Set. Although the usual Game of Set has any number of players, CS230 Set will have only one player. Our variant is thus a kind of solitaire.

CS230 Set is played with a deck of the 81 possible Set cards. The game starts by dealing some number of these cards face up to form what is called the tableau. The size of the tableau remains constant as long as there are still cards left in the deck, but may shrink after the deck is empty. The game can be played with any size of tableau, but 12 cards is the default.

At each step of the game, the player tries to find three cards in the tableau that form a set. Sometimes there may not be any sets in the tableau, so the player is allowed to declare that there are no sets in the tableau. We have the following cases:

- If the player selects three cards that form a set, the player “wins” the hand consisting of the three cards. These three cards are removed from the tableau. If the deck has three or more cards, three cards are dealt from the deck to replace the three taken by the player. If the deck has fewer than three cards, all remaining cards from the deck (if any) are added to the tableau.

- If the player selects three cards from the tableau that do not form a set, the game ends.

- If the player declares that there are no sets in the tableau, the game ends, regardless of whether the declaration is true or not. In the case where the player is incorrect, the player is penalized by not being able to continue the game to win more hands. In the case where the player is correct, the game ends because there are no more sets.

When the game ends, how well the player did is measured by the total number of sets the player won. This number may range between 0 (no hands won) to 27 (the maximum number of winnable sets). In CS230 Set, this number is determined by both luck and skill – luck, because a set-less tableau can end a game early; and skill, because the player advances by finding a set in a tableau that has one.

b. : The Contracts

In addition to the SetCard and SetHand contracts from PS1, the implementation of CS230 Set involves contracts for three new classes:

1. The SetTableau class represents a tableau of Set cards. See Appendix A for the contract for this class. This is the class that you will implement in Problems 1 and 2.

2. The SetDeck class represents a deck of Set cards. See Appendix B for the contract for this class.

---

1 For more about the real game, see http://www.setgame.com/set/.

2 In the real version of the game, when a player correctly declares that there are no more sets, the size of the tableau is increased (if possible) by dealing three more cards from the deck.
3. The SetGame class represents a CS230 Set game. See Appendix C for the contract for this class.

You should study all three of the above contracts before proceeding.

c. Game Modes

There are two modes in which the CS230 Set game can be played.

1. In automatic mode, a computer player automatically plays the game. Using the leastSet method of the SetTableau class, the computer player chooses the least set from the current tableau (if there is one) at every step of the game. This continues until the tableau contains no more sets. Fig. 2 shows a transcript of a sample automatic game.

```
[lyn@jaguar ps2] java SetGame automatic 12 true
----------------------------------------
1hgc  2ebt  2hbs  2hbt
2hgc  2hrs  2hrt  3fbt
3fgs  3frt  3hgc  3hrt
Found set [1hgc,2hbs,3hrt]
Automatic player has 1 sets
----------------------------------------
1fbs  2ebt  2egc  2hbt
2hgc  2hrs  2hrt  3fbt
3fgs  3frs  3frt  3hgc
Found set [2hbt,2hgc,2hrs]
Automatic player has 2 sets
----------------------------------------
1egc  1fbs  2ebt  2egc
2hrt  3egc  3fbs  3fbt
3fgs  3frs  3frt  3hgc
Found set [1egc,2egc,3egc]
Automatic player has 3 sets
----------------------------------------
1fbs  1fbt  2ebt  2hrt
3ebt  3fbs  3fbt  3fgs
3frs  3frc  3hgc  3hrc
Found set [3ebt,3fgs,3hrc]
Automatic player has 4 sets
----------------------------------------
1ebt  1fbs  1fbt  1frc
1hrc  2ebt  2hrt  3fbt
3fbs  3frc  3frc  3hgc
There are no more sets.
Automatic player won 4 sets: [1hgc,2hbs,3hrt],[2hbt,2hgc,2hrs],[1egc,2egc,3egc],[3ebt,3fgs,3hrc]
```

Figure 1: Transcript of a sample automatic 12-card game. In this game, the automatic player wins only 4 sets.
2. In **interactive mode**, the user plays the game. At each step of the game, the user can enter one of four inputs:

(a) A string representation of a Set hand (e.g., \[1fbt,2fgc,3frs\]) that specifies a set in the tableau.
   i. if the hand is a set in the tableau, the player wins this hand;
   ii. if the hand is in the tableau, but is not a set, the game ends;
   iii. if the hand string is ill-formed or not all of the cards are in the tableau, the input is assumed to be a mistake, and the player is given another chance to give an input.

(b) The input **none** is used to declare that there are no sets in the tableau. As explained above, this ends the game.

(c) The input **done** ends the game when the player does not wish to continue.

(d) The input **help** displays instructions for playing the game.

Fig. 2 shows a transcript of a short game played in interactive mode.

### d. Implementations

You have been provided with implementations of the **SetDeck** and **SetGame** classes in the files `ps2/SetDeck.java` and `ps2/SetGame.java`. You are encouraged to study these implementations before you proceed with the implementation of **SetTableau** in Problems 1 and 2. There are two reasons to study the implementations:

1. They include some programming idioms that are handy to know. In particular, they include array, vector, and loop idioms that may be useful to you for implementing **SetTableau**.

2. A better understanding for the structure of the game and how the **SetTableau** class is used within the game may help you in your implementation.
Welcome to the CS230 Set Game!

When a Set tableau is displayed, you have the following options:
1. Type in a set appearing in the tableau -- e.g. [lebc,2fgs,3hrt].
   * If the hand is a set in the tableau, you win it;
   * If the hand is in the tableau, but not a set, the game ends;
   * If the hand is illegal or not in the tableau,
     you will be given a chance to type in another set.
2. Type "none" (without the quotes) if you think there are no sets in the tableau.
   This ends the game whether you are right or wrong.
3. Type "done" (without the quotes) if you want to exit the game.
4. Type "help" (without the quotes) to see these instructions.

Type a set hand representation, "done", "none", or "help" (w/o quotes):
[1fbs,1fgt,1frc]
Good! You now have 1 sets

Type a set hand representation, "done", "none", or "help" (w/o quotes):
[1ers,2ebs,3egs]
Good! You now have 2 sets

Type a set hand representation, "done", "none", or "help" (w/o quotes):

none
Nope -- you missed the following sets:
[1fbt,2fgc,3frs]

Thanks for playing the CS230 Set Game!
You won 2 sets:
[[1fbs,1fgt,1frc], [1ers,2ebs,3egs]]

Figure 2: Transcript of a sample automatic 12-card game. In this game, the automatic player wins only 4 sets.
Problem 1 [65]: Implementing SetTableau

Flesh out the skeleton implementation of the SetTableau class in ps2/SetTableau.java so that it satisfies the contract in Appendix A. This class contains a single private instance variable named cards that holds a vector of the cards in the tableau. In your implementation, you should maintain the invariant that the cards in cards are always stored in sorted card order from least to greatest. This simplifies the implementation of the cards, toString, tableString, sets, and leastSet methods.

Notes:

- To begin this problem, execute the following in Linux:
  ```
  cd ~/cs230
  cvs update -d
  ```
- To compile your class, use `javac SetTableau.java`.
- Although Problem 1 and Problem 2 are listed as separate problems, you will probably want to interleave them.
- Use VectorOps.isort to sort a vector of SetCards or a vector of SetHands. (This sorts any vector of Comparable objects.)
- The VectorOps.binarySearch method introduced in class is an easy way to find the index or insertion index of an element in a sorted vector. This method is handy in several of the SetTableau methods.
- Do not use the contains(Object elem) or remove(Object elem) methods of the Vector class. But you may use the remove(int index) method.
- In tableString, the Math.sqrt and Math.ceil functions are handy. Note that Math.ceil takes a double and returns a double. You should use an (int) cast to convert it to an integer. To get a double result from dividing two integers, you first need to cast one of the integers to a double using (double). For instance, in Java, 7/2 is 3, but ((double) 7)/2, 7/((double)2), and ((double)7)/((double)2) all return 3.5.
- The hardest methods to implement are sets and leastSet. The most straightforward implementation involves a triply nested loop with indices i, j, and k such that i < j < k. Think carefully about these problems before you implement them.
- Your leastSet method should not invoke sets and extract the smallest element of the result. Although it’s often a good idea to implement one method in terms of another, the key idea behind leastSet is that it should be cheaper than sets because it only finds one set rather than all of them. Implementing leastSet in terms of sets would violate this expectation.
- It is likely that your leastSet method will look very much like your sets method except for a few tweaks. This should cause some queasiness – after all, shouldn’t we abstract over common code patterns? Yes, but not yet in this case. Soon we will learn about an abstraction (called an Enumeration) that is the right way to abstract over the commonalities between leastSet and sets.
- The Vector toArray method is helpful for returning a SetCard array in cards and SetHand array in sets. See examples of its use in SetGame.java.
• Once you have successfully implemented SetTableau, you should be able to compile SetGame (via javac SetGame.java). Then you can play CS230 Set — in automatic or interactive mode — as described in Appendix C.

Problem 2 [15]: Testing SetTableau

Add a main method and testing code to the SetTableau class to test your implementation from Problem 1. You needn’t implement anything as fancy as the testing code in PS1, though you are welcome to do so if you like. At the very least, you should have a way of testing each of your methods by giving appropriate arguments to main.

For this problem you should turn in a transcript of the tests that you perform. The easiest way to generate a transcript is to perform all of your tests within a shell buffer inside Emacs, and then save this buffer away to a file.

Problem 3 [20]: Game Histograms

How does the size of the tableau influence the number of sets that are typically collected in a CS230 Set game? Intuitively, it’s hard to find sets in small tableaux, so few sets will be collected when the tableau size is small. As the tableau size increases, we expect the the number of sets collected in a typical game will rise.

In this problem, you will investigate how the distribution of the number of sets collected in the CS230 Set Game depends on the size of the tableau. You will do this by writing a program that makes a histogram of the number of sets collected in each of $t$ trials of an automatic game with a tableau of size $n$. A histogram is a table with indexed slots. Here, the indices range from 0 (the smallest number of sets that can be won in a game) up to and including 27 (the largest number of sets that can be won in a game). The histogram stores in index slot $i$ the number of games out of the $t$ trials in which exactly $i$ sets were won.

For example, Fig. 3 shows a sample histogram for 100 trials of a game using a tableau size of 10. The first line of the histogram says that in 14 of the 100 games played, no sets were won in the game; the second says that exactly one game was one in 13 of the 100 games; and so on. The total number of games in the histogram is 100.

In this problem, you should write, from scratch, a class named SetGameHistogram in a file named ps2/SetGameHistogram.java. The main method of this class should produce histograms like the one in Fig. 3. Your class should have as many methods as you need to solve the problem.

You should test your class by computing histograms for all tableau sizes $n$ between 9 and 15, inclusive. In each histogram, use 100 trials. Note that as $n$ grows larger, the time to compute the histogram grows larger as well; be patient. For example, on my laptop, computing the histogram for $n = 15$ and $t = 100$ takes about a minute. (Yours may take more or less time, depending on the machine you are using and the efficiency of your SetTableau implementation.) Turn in a transcript showing your histograms.

Also, answer this question: based on your histogram results, what do you think is the most sensible default size for a game of CS230 Set? There is no right or wrong answer here – I want to hear your reasoning!

Notes:

• You should use the automatic method of the SetGame class to play each game. Using the verbose flag false will prevent the details of each game from being displayed on the screen.

• It does not take very much code to solve this problem. It is possible to solve it using one or two short methods, though you are welcome to use more.
Figure 3: Sample histogram of 100 trials for a game with tableau size 10.
Going Further
You are encouraged to explore improvements and extensions to the game presented in this assignment. There are many ways to improve the CS230 Set Game. The text-based interface is very clunky and could be made more user-friendly. A graphical user interface (GUI) would be even nicer. (You will learn how to implement GUIs later in the semester.) The rules of the game could be modified to be more like those in the real Game of Set. For example, the real game involves multiple players, and the tableau size is increased (if possible) when all players agree that the tableau contains no sets.

Appendix A: SetTableau Contract

The SetTableau class models the collection of cards face up on the table in the Game of Set.

Public Constructor Methods:

public SetTableau ()
Creates a tableau with 0 cards.

public SetTableau (String [] cardStrings)
Creates a tableau whose cards are determined by the string representations of cards in cardStrings. Throws a RuntimeException if any of the strings are not legal card representations or if the same card appears more than once in cardStrings.

public SetTableau (SetCard [] scards)
Creates a tableau whose cards are the cards in scards. Throws a RuntimeException if the same card appears more than once in scards.

Public Instance Methods:

public int size ()
Returns the number of cards in this tableau.

public SetCard [] cards ()
Returns all the cards of this tableau in an array. In the resulting array, the cards should be sorted from least to greatest in the card ordering.

public void add (SetCard sc)
Adds sc to this tableau. Throws a RuntimeException if sc is already in the tableau.

public boolean contains (SetCard sc)
Returns true if sc is in this tableau and false otherwise.

public boolean contains (SetHand sh)
Returns true if all the cards in sh are in this tableau and false otherwise.

public void remove (SetHand sh)
If sh is a set, removes the three cards in sh from this tableau. Throws a RuntimeException if sh is not a set or the hand is not contained in this tableau.

public String toString ()
Returns a string representation of the cards in this tableau. This should be a comma-separated sequence of card strings (in sorted order from least to greatest) delimited by squiggly brackets. For example, here is the string representation of a 12-card tableau we’ll call sample:

{1ebc,1erc,1fbc,1fbt,1hbc,1hbs,1hrs,2erc,2fgc,3fgt,3frc,3erc}
public String tableString ()
Returns a string representing the cards of this tableau as newline-separated rows of tab-
separated columns of card strings. If \( n \) is the number of cards in this tableau, then the
number of columns \( c \) should be \( \lceil \sqrt{n} \rceil \) and the number of rows \( r \) should \( \lceil n/c \rceil \) (where \( n/c \) is
mathematical division on real numbers, not Java's / operator).\(^3\) The cards should be shown in
card order from least to greatest. For example, Fig. 4 shows the result of displaying tableaux
with 9 through 13 cards.

```java
// 9-card tableau
1ebc 1erc 1fbc
1fbt 1hbc 1hbs
1hrs 2erc 2fgc

// 10-card tableau
1ebc 1erc 1fbc 1fbt
1hbc 1hbs 1hrs 2erc
2fgc 3fgt

// 11-card tableau
1ebc 1erc 1fbc 1fbt
1hbc 1hbs 1hrs 2erc
2fgc 3fgt 3frs

// 12-card tableau
1ebc 1erc 1fbc 1fbt
1hbc 1hbs 1hrs 2erc
2fgc 3fgt 3frs 3erc

// 13-card tableau
1ebc 1erc 1fbc 1fbt
1hbc 1hbs 1hrs 2erc
2fgc 3fgt 3frs 3erc
3hrt
```

Figure 4: Results of calling the `tableString` method with tableaux ranging in size from 9 to 13.

public SetHand [] sets ()
Returns an array of all sets in this tableau. The sets in the returned array should be ordered
from least to greatest according to the `SetHand` ordering. For example, for the 12-card sample
tableau introduced in the `toString` specification, `sets()` should return an array of 6 hands:

```
[\{1ebc,1fbc,1hbc\}, \{1ebc,1fbt,1hbs\}, \{1erc,2erc,3erc\},
 \{1fbt,2fgc,3frs\}, \{1hbc,2fgc,3erc\}, \{1hbs,2erc,3fgt\}]
```

public SetHand leastSet ()
Returns the least set (by the `SetHand` ordering) in this tableau if there is one. Otherwise
returns `null`. For example, for the 12-card sample tableau introduced in the `toString`
specification, `leastSet()` should return the hand \{1ebc,1fbc,1hbc\}.

---

\(^3\)The notation \([x]\), pronounced “ceiling of \(x\)”, denotes the smallest integer \(\geq x\). The notation \([x]\), pronounced
“floor of \(x\)”, denotes the largest integer \(\leq x\). For example, \([6.001]\) = \([6.821]\) = 7, while \([6.001]\) = \([6.821]\) = 6.
Appendix B: SetDeck Contract

Public Constructor Methods:

```
public SetDeck ();
Creates a new deck with all 81 cards from the Game of Set.

public SetDeck (String [] cardStrings);
Assumes that cardStrings is a array of string representations of cards that has all 81 cards from the Game of Set exactly once. Returns a deck that will deal the cards in the order specified by the cardStrings array. I.e., the first call to deal() returns the card specified by cardStrings[0], the second call to deal() returns the card specified by cardStrings[1], etc. Throws a RuntimeException if cardStrings does not contain the string representations of all 81 cards exactly once.

This constructor is useful for testing purposes, since it allows complete control over the order in which cards are dealt from the deck. Otherwise, the cards are dealt in random order.
```

Public Instance Methods:

```
public SetCard deal ();
Returns a randomly chosen card from the deck after removing it from the deck.

public int size ();
Returns the number of cards left to deal in this deck.

public boolean isEmpty ();
Returns true if there are no cards left to deal in this deck. Otherwise returns false.

public String toString ();
Returns a string representation of this deck in the form <SetDeck:size=n>, where n is the number of cards left in the deck.
```
Appendix C: SetGame Contract

A SetGame instance represents a game of CS230 Set. Conceptually, a the state of a game consists of a deck and a tableau with a specified number of cards. The game can be played for any size of tableau between 0 and 81.

```java
public SetGame (int n);
Creates a new game with a tableau of n cards dealt from a new SetDeck.
```

```java
public SetGame (int n, SetDeck d);
Creates a new game using a tableau of n cards dealt from the give deck d.
```

**Public Instance Methods:**

```java
public SetHand [] interactive ();
Plays an interactive version of CS230 Set with the user via a crude text-based interface. Returns the array of sets won during the game (in the order they were won, not in hand order).
```

```java
public SetHand [] automatic (boolean verbose);
Plays an automatic version of CS230 Set with a computer player that always chooses the least set from the available tableau (if there is one). Returns the array of sets won during the game (in the order they were won, not in hand order). The verbose flag controls the display of information during the game. If verbose is true, the tableaux and automatic player choices are displayed at each point of the game. If verbose is false, no text is displayed during the game.
```

**Public Class Methods:**

```java
public static void main (String [] args);
The SetGame main method can be invoked as follows from a Linux shell:

- java SetGame interactive n invokes new SetGame(n).interactive().
- java SetGame interactive invokes new SetGame(12).interactive().
- java SetGame automatic n bool invokes new SetGame(n).automatic(bool).
- java SetGame automatic n invokes new SetGame(n).automatic(true).
- java SetGame automatic invokes new SetGame(12).automatic(true).
```
Names of Team Members:

Date & Time Submitted:

Collaborators (anyone you or your team collaborated with):

By signing below, I/we attest that I/we have followed the collaboration policy as specified in the Course Information handout.
Signature(s):

In the Time column, please estimate the time you or your team spent on the parts of this problem set. Team members should be working closely together, so it will be assumed that the time reported is the time for each team member. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the Score column when grading your problem set.

<table>
<thead>
<tr>
<th>Part</th>
<th>Time</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Reading</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem 1 [65]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem 2 [15]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem 3 [20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>