Problem Set 3
Due: 6pm Friday, February 27

Revisions:
Feb 25: In Appendix A, in the constructor with the cardStrings argument, the type of the argument should be String [], not SetCard [].
Feb 26: In figures 5 and 6, the strings passed to the -s option were missing explicit squiggly braces.

Overview:
The purpose of this assignment is to give you practice with enumerations and input/output in Java. You will also get more practice with writing and testing Java programs from scratch.

Reading:
- Handout #11: Enumerations

Working Together:
Reminder: if you worked with a partner on PS1 or PS2 and want to work with a partner on this assignment, you must choose a different partner.

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 Triples.java from Problems 1.
3. a transcript of your tests from Problem 1.
4. your final version of SetEnum.java from Problems 2 and 3.
5. a transcript of your tests from Problem 3.

Each team should also submit a single softcopy (consisting of your final ps3 directory) to the drop directory ~cs230/drop/ps3/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 ps3 ~cs230/drop/ps3/username/

Problem 1 [20]: Triples
In this problem you will define a class Triples that enumerates instances of the IntTriple class shown in Fig. 1. (This has already been defined for you in ~/cs230/ps3/IntTriple.java). In a new file named ~/cs230/ps3/Triples.java, define a Triples class that implements the Enumeration interface. Your class should have one constructor method:
public class IntTriple {
    public int i1, i2, i3;

    public IntTriple (int n1, int n2, int n3) {
        i1 = n1;
        i2 = n2;
        i3 = n3;
    }

    public String toString () {
        return "<" + i1 + "," + i2 + "," + i3 + ">";
    }
}

Figure 1: The IntTriple class.

public Triples (int n);
Creates a new enumeration that enumerates all IntTriple instances of the form <i,j,k>,
where 0 ≤ i < j < k ≤ n. The triples should be enumerated in lexicographic order (i.e.,
dictionary order), where i is the most significant number, j is next, and k is least significant.
See Fig. 2 for examples.

As shown in Fig. 2, the main method of your Triples class should read a non-negative integer
n supplied on the command line and show all the triples enumerated by new Triples(n).

Notes:

• Begin this problem by performing a cvs update -d in your ~/cs230 directory. This will
  create a ~/cs230/ps3 subdirectory containing the files that you need for the rest of the
  assignment.

• Make sure that Triples.java begins with the line import java.util.*;. This makes all
classes from the java.util package (including Enumeration and Vector) visible in the rest
of the file.

• The header for your class should be
  public class Triples implements Enumeration
This indicates that Triples satisfies the Enumeration interface – i.e., it has the following
two public instance methods:
    public boolean hasMoreElements();
    public Object nextElement();

• This problem essentially asks you to modularize the calculation of indices in the triply nested
  loops that you wrote for the isset and leastSet methods in PS2. Think carefully about
what state (i.e. instance variables) a Triples instance must maintain in order to generate
the next triple. How do you know when you’ve run out of triples?

• As shown in Handout #11, an easy way to test an enumeration e is to use EnumTest.test(e);
  You should use this in your main method.
[lyn@jaguar ps3] java Triples 0
Total number of elements: 0

[lyn@jaguar ps3] java Triples 1
Total number of elements: 0

[lyn@jaguar ps3] java Triples 2
<0,1,2>
Total number of elements: 1
[lyn@jaguar ps3] java Triples 3
<0,1,2>
<0,1,3>
<0,2,3>
<1,2,3>
Total number of elements: 4

[lyn@jaguar ps3] java Triples 4
<0,1,2>
<0,1,3>
<0,1,4>
<0,2,3>
<0,2,4>
<0,3,4>
<1,2,3>
<1,2,4>
<1,3,4>
<2,3,4>
Total number of elements: 10

[lyn@jaguar ps3] java Triples 5
<0,1,2>
<0,1,3>
<0,1,4>
<0,1,5>
<0,2,3>
<0,2,4>
<0,2,5>
<0,3,4>
<0,3,5>
<0,4,5>
<1,2,3>
<1,2,4>
<1,2,5>
<1,3,4>
<1,3,5>
<1,4,5>
<2,3,4>
<2,3,5>
<2,4,5>
<3,4,5>
Total number of elements: 20

Figure 2: Examples showing the Triples class in action.
• As part of your hardcopy submission, you should turn in a transcript showing the result of invoking `java Triples` on inputs from 0 through 6.
Problem 2 [60]: SetEnum

In this problem, you will define a SetEnum class that enumerates all the sets (in lexicographic order) in a given collection of cards from the Game of Set. The contract for the SetEnum class is given in Appendix A. The SetEnum class modularizes the enumeration of sets from the sets and leastSet methods of the SetTableau class that you wrote in PS2. As shown in Fig. 3, it is easy to use SetEnum to implement both sets and leastSet. Moreover, because an enumeration only generates as many elements as requested, the implementation of leastSet in terms of SetEnum is efficient – something that would not be the case if leastSet were written in terms of sets.

```java
public SetHand [] sets () {
    Vector sets = new Vector();
    SetEnum se = new SetEnum(this);
    while (se.hasMoreElements()) {
        sets.add(se.nextElement());
    }
    return (SetHand []) sets.toArray(new SetHand [sets.size()]);
}

public SetHand leastSet () {
    SetEnum se = new SetEnum(this);
    return (SetHand) se.nextElement(); // According to SetEnum spec, returns
    // null if there aren't any more elements.
}
```

Figure 3: Elegant implementations of the SetTableau sets and leastSet methods written in terms of SetEnum.

Your task is to define a SetEnum class satisfying the contract in Appendix A in the file ~/cs230/ps3/SetEnum.java, which you should create from scratch.

Notes:

- Make sure that SetEnum.java begins with the following lines:
  import java.util.*;
  import java.io.*;

  This makes all classes from the java.util package (including Enumeration and Vector) and the java.io package (including FileWriter and IOException) visible in the rest of the file.

- The header for your class should be

  public class SetEnum implements Enumeration

  This indicates that Triples satisfies the Enumeration interface – i.e., it has the following two public instance methods:

  public boolean hasMoreElements();
  public Object nextElement();

- Think carefully about what instance variables a SetEnum instance needs to have. Hint: Using a Triples instance from Problem 1 will greatly simplify your program.

- In order to know whether there are more sets in the enumeration, you will need to use the buffering idiom described in Handout #11. That is, you should have a buffer instance variable that is either empty (i.e., holds the null pointer) or full (i.e., holds the next set to
be enumerated). You should also define a `void peekElement()` method that fills an empty buffer with the next set in the enumeration, if there is one, but does nothing if the buffer is already full.

- If you are clever, you constructor methods will not need to do any sorting of cards, checking for duplicate cards, or checking for invalid card representations. *Hint:* Use `SetTableau` constructors to do this work for you!

- For the `fromFile` method, you should use the `FileLines` CS230 class to enumerate lines from a file. See Appendix B for details. You should also use the Java SDK `StringTokenizer` class discussed in lecture to extract card strings from the file. The file `deck.txt` specifies a collection of all 81 set cards, and the files `cards1.txt`, `cards2.txt`, `cards3.txt`, `cards4.txt`, and `cards5.txt` specify the same collection of 15 cards in different formats.

- For the `toFile` method, you should use the `FileWriter` Java SDK class. See the Java SDK documentation for details. You will need to create a `FileWriter` instance from a filename using a `FileWriter` constructor. You will also need to use the `write` method to write strings and/or characters, and the `close` method to close the file when you are done.

- For the `interactive` method, you will need to read a string of characters typed by the user in the Linux shell. The expression `(char) System.in.read()` reads the next character from the shell; you should collect all such characters into a string until you read the newline character (the end of the input line typed by the user). For an example of this idiom, study the `getInput` method in `~/cs230/ps2/SetGame.java`.

- As you write your methods, you should test them via the `SetEnum` main method. In Problem 3, you will replace your simple tests with a very general command-line interface for testing your `SetEnum` methods.

**Problem 3 [20]: Using `SetEnum` with Command-Line Options**

Many Linux commands take so-called command-line options that modify the meaning of the command. For instance, the default behavior of the `ls` command is to list the filenames in the current directory in tab-separated columns:

```
[lyn@jaguar ps3] ls cards*.txt
cards1.txt cards2.txt cards3.txt cards4.txt cards5.txt
```

However, specifying the `-l` command-line option displays the files one per line with extra information:

```
[lyn@jaguar ps3] ls -l cards*.txt
-rw-rw---- 1 lyn lyn  77 Feb 22 11:14 cards1.txt
-rw-rw---- 1 lyn lyn  77 Feb 21 09:37 cards2.txt
-rw-rw---- 1 lyn lyn  79 Feb 22 11:16 cards3.txt
-rw-rw---- 1 lyn lyn  75 Feb 22 11:17 cards4.txt
-rw-rw---- 1 lyn lyn 150 Feb 22 11:33 cards5.txt
```

As another example, the `a2ps` command normally converts a text file into PostScript form and sends it to a printer to be printed. However, specifying the `-o` option followed by a filename will write the converted output to a file rather than send it to the printer:

```
[lyn@jaguar ps3] a2ps -o cards1.ps cards1.txt
[cards1.txt (plain): 1 page on 1 sheet]
[Total: 1 page on 1 sheet] saved into the file `cards1.ps`
```
In this problem, you will modify whatever testing behavior your `main` method had in Problem 2 to have the behavior described in Fig. 4.

The default behavior of `main` is to use `EnumTest.test()` to display the sets for the cards specified in the file `cards1.txt`. This default behavior can be overridden by the following command-line options:

- `-s string`: Uses `string` as the specification of the cards from which the sets are drawn, where `string` has a format acceptable for the `String` constructor for the `SetEnum` class.

- `-f infilename`: Reads the cards from the file `infilename` using `SetEnum.fromFile`.

- `-p n`: Assume that `n` is a positive number. Indicates that only the first `n` elements in the set enumeration should be used. An error message is printed if `n` is not a positive number.

- `-o outfilename`: Rather than displaying any sets on the screen, writes them to the file named `outfilename` using `SetEnum.toFile()`.

- `-i`: Uses interactive mode (via `SetEnum.interactive()`) to display the sets in the enumeration.

Figure 4: Specification of the behavior of the `SetEnum main` method.

A transcript showing sample interactions with the `SetEnum` program is shown in Figs. 5–6. As shown in the transcript, it is possible to use several command line options in the same command, and the order of their use does not matter. However, the same option may not be used more than once in a single command, and certain other combinations are not allowed. For example, input can only come from either a string or a file, so only one of `-s` or `-f` is allowed in a single line. Sets can be processed interactively or sent to a file, so only one of `-i` or `-o` is allowed in a single line.

Notes:

- You may want to first simplify the problem by handling only one option at a time or multiple options in a fixed order.

- Think carefully about how to handle multiple options in arbitrary order and how to catch option errors.

- For error handling, it is helpful to know the following:
  - you can immediately exit a `void` method by executing `return;`.
  - to handle an exception raised in the execution of `statements`, use the following template:

    ```java
    try {
      statements
    } catch (exception-class-1 exn) {
      statements to handle exception-class-1
    } catch (exception-class-2 exn) {
      statements to handle exception-class-2
    } ...
    ```

- In your hardcopy submission, turn in a transcript of invoking `SetEnum` program with numerous combinations of command-line options. You should show how your program deals with option errors as well as correctly used options.
[fturbak@teddy ps3] java SetEnum
[1ebc,1fbc,1hbc] Default is to display sets from cards1.txt.
[1ebc,1fbc,1hbs]
[1ebc,2erc,3egc]
[1ebc,2hrs,3fgt]
[1erc,2erc,3erc]
[1fbt,2fgc,3frs]
[1fbt,2hrs,3egc]
[1hbc,2fgc,3erc]
[1hbs,2erc,3fgt]
[1hrs,2hrs,3hrs]
Total number of elements: 10

[fturbak@teddy ps3] java SetEnum -p 3
[1ebc,1fbc,1hbc] Displays the first 3 sets in cards1.txt.
[1ebc,1fbc,1hbs]
[1ebc,2erc,3egc]
Total number of elements: 3

[fturbak@teddy ps3] java SetEnum -s "{1ebc,1ebs,1ebt,2ebc,3ebc}"
[1ebc,1ebs,1ebt] Displays all 2 sets in the given cards.
[1ebc,2ebc,3ebc]
Total number of elements: 2

[fturbak@teddy ps3] java SetEnum -s "{1ebc,1ebs,1ebt,2ebc,2ebs,2ebt,3ebc,3ebs,3ebt}"
[1ebc,1ebs,1ebt] Displays all 12 sets in the given cards.
[1ebc,2ebc,3ebc]
[1ebc,2ebs,3ebs]
[1ebc,2ebt,3ebc]
[1ebs,2ebc,3ebs]
[1ebs,2ebs,3ebs]
[1ebs,2ebt,3ebc]
[1ebt,2ebc,3ebs]
[1ebt,2ebs,3ebc]
[1ebt,2ebt,3ebs]
[2ebc,2ebs,2ebt]
[3ebc,3ebs,3ebs]
Total number of elements: 12

[fturbak@teddy ps3] java SetEnum -p 3 -s "{1ebc,1ebs,1ebt,2ebc,2ebs,2ebt,3ebc,3ebs,3ebt}"
[1ebc,1ebs,1ebt] Displays first 3 sets in the given cards.
[1ebc,2ebc,3ebc]
[1ebc,2ebs,3ebc]
Total number of elements: 3

[fturbak@teddy ps3] java SetEnum -s "{1ebc,1ebs,1ebt,2ebc,2ebs,2ebt,3ebc,3ebs,3ebt}" -p 2
[1ebc,1ebs,1ebt] Order of options does not matter.
[1ebc,2ebc,3ebc]
Total number of elements: 2

Figure 5: Transcript of interactions with SetEnum command line interface, part 1.
Figure 6: Transcript of interactions with `SetEnum` command line interface, part 2.
Appendix A: SetEnum Contract

A SetEnum instance enumerates all the sets from a given collection of cards in lexicographic order.

Public Constructor Methods:

public SetEnum (SetTableau st);
Creates a new set enumeration from the cards in the tableau st.

public SetEnum (SetCard [] scs);
Creates a new set enumeration from the cards in the array scs. Throws a RuntimeException if the same card appears twice in scs.

public SetEnum (String [] cardStrings);
Creates a new set enumeration from the cards specified by the string representations in the string array scs. Throws a RuntimeException if the same card is specified twice in cardStrings or if any of the strings is not a valid string representation of a card.

public SetEnum (String cardsString);
Creates a new set enumeration from the cards specified by the string cardsString, which should be a string of comma-separated card representations delimited by squiggly braces. String representations in the string array scs. Throws a RuntimeException if the same card is specified twice in cardsString or if any of the card representations is not a valid string representation of a card.

Public Instance Methods:

public boolean hasMoreElements ();
Returns true if there are more sets in the enumeration and false otherwise.

public Object nextElement ();
Returns the next set (in lexicographic) order if there is one. Otherwise, returns the null pointer.

Public Class Methods:

public static SetEnum fromFile (String filename);
Returns an instance of SetEnum that enumerates all the cards specified in the file named filename. Each card specification should be a 4-character string representation as specified in the SetCards contract. Such 4-character strings may be separated by any number of whitespace characters, commas, squiggly braces, and single and double quotes, all of which are ignored. A RuntimeException should be thrown if any other characters, an invalid card representation, or a duplicate card are encountered. Fig. 7 shows the contents of five text files in the ~/cs230/ps3 directory, all of which denote the same collection of fifteen cards. The file deck.txt (not shown in the figure) contains all 81 cards in the Game of Set.

public static void toFile (Enumeration e, String filename);
Writes the string representation of every element of any enumeration e, one per line, to the file named filename. For example, if se is an instance of SetEnum created by the invocation SetEnum.fromFile("cards1.txt"), then toFile(se, "out1.txt") creates a file with the contents shown in Fig. 8.
public static void interactive (Enumeration e);
Enters an interactive mode in which the user is prompted for the number $n$ of elements she wants to see displayed from the given enumeration $e$. If there are $n$ or more elements remaining in the enumeration, displays each element, one per line, after a number $i$, a colon, and a tab, where $i$ is the index of the element since interactive mode was entered. If there are less than $n$ elements, displays all of the the elements, one per line as described above, followed by the line No more elements. If the input $n$ is $\leq 0$ or is not a number, exits interactive mode after displaying the line Bye!. For example, Fig. 9 shows a transcript from interactive mode that was entered via the invocation `SetEnum.interactive(SetEnum.fromFile("cards1.txt"))`.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>cards1.txt</td>
<td><code>{1ebc,1erc,1fbc,1fht,1hbc,1hrs,2erc,2frc,2hrs,3egc,3erc,3fgc,3hrs,3hrs}</code></td>
</tr>
<tr>
<td>cards2.txt</td>
<td><code>{3egc,2frc,1ebc,3frs,1fbc,3fgc,1hbc,2erc,1erc,2hrs,1hrs,3erc,1fht,1hrs,1hrs}</code></td>
</tr>
<tr>
<td>cards3.txt</td>
<td><code>&quot;{3egc,2frc,1ebc,3frs,1fbc,3fgc,1hbc,2erc,1erc,2hrs,1hrs,3erc,1fht,1hrs,1hrs}&quot;</code></td>
</tr>
</tbody>
</table>

| cards4.txt | `3egc 2fgc 1ebc 3frs 1fbc 3fgt 1hbc 2erc 1erc 2hrs 1hrs 3erc 1fht 3hrs 1hrs` |
| cards5.txt | `3egc, "2fgc" 1ebc {3frs} 1fbc 3fgt , 1hbc 2erc "1erc" '2hrs' "1hrs' '3erc" 1fht 3hrs 1hrs` |

Figure 7: The contents of five different card files, all of which specify the same fifteen cards.

Figure 8: Contents of the file `out1.txt`. 
<table>
<thead>
<tr>
<th></th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1ebc,1fbc,1hbc]</td>
</tr>
<tr>
<td>2</td>
<td>[1ebc,1fbt,1hbs]</td>
</tr>
<tr>
<td>3</td>
<td>[1ebc,2erc,3egc]</td>
</tr>
<tr>
<td>4</td>
<td>[1ebc,2hrs,3fgt]</td>
</tr>
<tr>
<td>5</td>
<td>[1erc,2erc,3erc]</td>
</tr>
<tr>
<td>6</td>
<td>[1fbt,2fgc,3frs]</td>
</tr>
<tr>
<td>7</td>
<td>[1fbt,2hrs,3egc]</td>
</tr>
<tr>
<td>8</td>
<td>[1hbc,2fgc,3erc]</td>
</tr>
<tr>
<td>9</td>
<td>[1hbs,2erc,3fgt]</td>
</tr>
<tr>
<td>10</td>
<td>[1hrs,2hrs,3hrs]</td>
</tr>
</tbody>
</table>

Figure 9: Contents of the file `out1.txt`.
Appendix B: FileLines Contract

The FileLines class is an implementation of the Enumeration interface that yields the lines of a given file one by one. Each line includes the terminating newline character, if present, unless the enumeration is created with the two-argument constructor with a second argument of false. In addition to the hasMoreElements and nextElement instance method required by implementations of Enumeration, FileLines supports the following two constructor methods and main testing method:

    public FileLines (String filename);
    Create an enumeration that enumerates the lines of the file named by filename. Each line includes the terminating newline.

    public FileLines (String filename, boolean includeNewlines);
    Create an enumeration that enumerates the lines of the file named by filename. The includeNewlines flag controls whether newlines are included in each line.

    public static void main (String [] args);
    If there is one argument, invokes EnumTest.test on new FileLines(args[0]). If there are two arguments, and the second is false, invokes EnumTest.test on the result of the constructor invocation new FileLines(args[0],false). If there are two arguments, and the second is a non-negative number n, displays the first n lines of file named by args[0].

For example, suppose that out1.txt is the file specified in Fig. 8. Then here is the result of the invocation of java FileLines out1.txt:

    [1ebc,1fbc,1hbc]
    [1ebc,1fbt,1hbs]
    [1ebc,2erc,3egc]
    [1ebc,2hrs,3fgt]
    [1erc,2erc,3erc]
    [1fbt,2fgc,3frs]
    [1fbt,2hrs,3egc]
    [1hbc,2fgc,3erc]
    [1hbs,2erc,3fgt]
    [1hrs,2hrs,3hrs]

    Total number of elements: 10

In the above example, each pair of lines is separated by a blank line because each line includes a terminating newline in addition to the newline introduced for each line by EnumTest.test. Had we instead invoked java FileLines out1.txt false, the blank lines would go away.

---

1The contract given here is slightly different than for the FileLines implementation described in Handout #11. In particular, the version in Handout #11 does not allow controlling whether newlines are kept in each line.
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 [20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem 2 [60]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem 3 [20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>Total</strong></td>
</tr>
</tbody>
</table>