Problem Set 2

Procedural Abstraction and Using Abstract Datatypes

Out: 30 August
Due: **Friday, 8 September**
(beginning of class)

Collaboration Policy - Read Carefully: For this problem set, you may work alone or with one other student in the class. If you work with another student, you should both participate fully in all parts of the assignments and turn in one assignment with both of your names on it. Either way, feel free to ask other students for help and offer help to other students.

Purpose

- Learn to reason about and develop good procedural specifications.
- Learn to use an abstract data type by reading its specification.
- Think about how to test a procedure.

Reading: read Chapters 3 and 4, Chapter 5 through section 5.2, Chapter 9, and Chapter 10 through 10.2.

This assignment is longer than PS1 and you have over a week to complete it. Please don't wait to get started. You are strongly encouraged to take advantage of the scheduled lab hours in Small Hall (Sundays, 7-8; Mondays, 6-7; and Thursdays, 5-6).

Specifying Procedures

It is necessary for technical reasons that these warheads be stored upside down, that is, with the top at the bottom and the bottom at the top. In order that there be no doubt as to which is the bottom and which is the top, for storage purposes, it will be seen that the bottom of each warhead has been labeled 'TOP'.
Instructions accompanying a shipment of ballistic missiles from British Admiralty (reported in The Humus Report)

For the next question, consider these two specifications for the `sort` procedure:

A. From the Java 2 Platform API documentation (`java.util.Arrays`):

```
public static void sort(int[ ] a)
```

Sorts the specified array of ints into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance.

Parameters:
`a` - the array to be sorted.

B. From the textbook (p.46):

```
public static void sort(int[ ] a)
```

MODIFIES: `a`
EFFECTS: Rearranges the elements of `a` into ascending order.
e.g., if `a = [3, 1, 6, 1]`, `a_post = [1, 1, 3, 6]`

1. Describe three advantages of specification B over specification A.

2. Describe one scenario where specification A is more useful.

Consider the `histogram` prodecure defined (but not specified!) below:

```
public static int [] histogram (int [] a)
// unspecified
{
   int maxval = 0;
   for (int i = 0; i < a.length; i++) {
      if (a[i] > maxval) {
         maxval = a[i];
      }
   }
   int histo [] = new int [maxval + 1];
   for (int i = 0; i < a.length; i++) {
      histo[a[i]]++;
   }
   return histo;
}
```

For example,

```
   int [] test = { 1, 2, 3, 4, 6, 4, 3, 3, 0 } ;
   int [] hist = histogram (test);
```

the value of `hist` will be `[1, 1, 1, 3, 2, 0, 1]`.

3. Write a complete, declarative specification of `histogram`. Your specification should be enough for a client to safely use the implementation provided above.

4. Write an alternative specification for the `histogram` procedure that places less burden on the client by using exceptions. What are the advantages and disadvantages of this specification compared to your answer to question 3?

5. Write an alternative specification for the `histogram` procedure that is total. A client should be able to pass any array of integers into this procedure and obtain a meaningful return value. What are the advantages and disadvantages of this specification compared to your answers to questions 3 and 4?

Dependency Graphs

Download
http://www.cs.virginia.edu/cs250/ps/ps2/ps2.zip to your home directory and extract all files to your `cs205\` directory (this will create the `cs205\ps2` subdirectory).

A *dependency graph* is a structure for showing dependencies between things. It could be used, for example, to represent dependencies between events ("opening the book" has to happen before "reading the book") or between modules (the table top depends on the table legs). For this assignment, you will read in a file that describes a set of items and their dependencies, and produce a schedule in which each item is done only after all the items on which it depends have already been completed.

The input file lists a task, followed by a number (which could represent the time required to complete the task), followed by a list of tasks upon which this dask depends. Lines that start with a # are treated as commments.

For example,

```
# Based on Figure 13.9 (p. 323) of textbook
Engine 30 { Comm.getDocs TitleTable Doc Query WordTable }
Comm.getDocs 50 { }
TitleTable 60 { Doc }
Doc 15 { }
Query 30 { WordTable }
WordTable 20 { Doc }
```

describes the dependency graph shown in Figure 13.9 of the textbook.

The first task listed in the file (`Engine` in the example) is the main task. The output of your program should be a schedule of tasks that can be used to complete the main task, and the total time needed to complete it. For the example there are many possible valid schedules; one valid output would be:

```
Schedule: [Comm.getDocs, Doc, TitleTable, WordTable, Query, Engine]
Completion time: 205
```

Note that every task in the list is preceeded by all tasks upon which it depends.

We have provided you with several abstract datatypes that you should find helpful in building your implementation. Their specifications are at the end of this document. Note that we provide only the specifications and class file implementations, not the source code. Your code should work with any implementations of these datatypes that satisfy the provided specifications.

6. Impement the task schedule as described above. Your program should take a file name as input, and output a valid schedule and completion time for completing the first task in the file. You implementation should be total: no matter what input it is given, it should behave in a *sensible* way. You should use procedural abstraction to avoid code duplication and enhance the clarity of your implementation.

You should implement your program by creating a new `TaskScheduler` class (in the ps2 package) with a `main` method.

7. Write a specification for your program. Your specification should be total: it should describe what the program does on all possible inputs.

8. Describe a testing strategy for an implementation of the task schedule program. Your answer should include a list of black-box test cases, and any additional glass-box test cases.

9. How confident are you that your program will always work as intended? (Where "as intended" means as you specified it in question 7, except if there are inputs that are not covered by your specification it must behave as the course staff intended.) Express your answer as a bet of between 0 and 20 points. If the customer (grader) agrees that your program always works as intended, you get the points. If not, you lose twice your bet. You should assume that your program will run with implementations of the provided datatypes that satisfy their specifications, but not necessarily the same implementations as were provided.

Turn-in Checklist: You should turn in your answers to questions 1-9 on paper at the beginning of class on Friday, 8 September. Also, submit your `TaskSchedule.java` code electronically by email to *evans@cs.virginia.edu*.

Specifications

DirectedGraph

The `DirectedGraph` datatype is a *generic* datatype, meaning it is parameterized with a type parameter. The parameter *T* is the type of the nodes in the graph. For example, `DirectedGraph<String>` would denote a `DirectedGraph` where the nodes has time `String`.

```
public class DirectedGraph<T>
  OVERVIEW: A DirectedGraph is a directed graph  where
      V is a set of nodes (of type T), and E is a set of edges.
      Each edge is a pair (v1, v2), representing an edge
      from v1 to v2 in G.

  public DirectedGraph()
    EFFECTS: Creates a new, empty DirectedGraph: < {}, {} >

  public void addNode(T s) throws DuplicateException
     MODIFIES: this
     EFFECTS: If s is the name of a node in this, throws
       DuplicateNodeException. Otherwise, adds
       s to the nodes in this, with no adjacent nodes:
           $G_{post}$ = < $V_{pre}$ U { s }, $E_{pre}$ >

  public void addEdge(T s, T t) throws NoNodeException, DuplicateException
     MODIFIES: this
     EFFECTS: If s and t are not names of nodes in
        this, throws NoNodeException.  If there is already an edge
        between s and t, throws DuplicateEdgeException.
        Otherwise, adds an edge between s and t to this:
           $G_{post}$ = < $V_{pre}$, $E_{pre}$ U  >

  public Set<T> getAdjacent(T s) throws NoNodeException
     EFFECTS: If s is not the name of a node, throws NoNodeException.
        Otherwise, returns an array of the nodes adjacent to s
        That is, returns the set of nodes
            { e | <s, e> is in E }
```

Set

Another datatype we provide is `Set`, specified below (`Set` provides some additional methods, not specified here; you should only need to use the specified methods):

```
public class Set<T> implements Iterable<T>, Collection<T> {
   OVERVIEW: A Set is a mutable, unbounded set of objects of type T.
        A typical Set is {x_1, ..., x_n }.

    public Set()
       EFFECTS: Initializes this to an empty set: { }.

    public Set(Set<T> s)
       EFFECTS: Initializes this to a set containing the same elements
          as the set s (a shallow copy).

    public boolean add(T el)
       MODIFIES: this
       EFFECTS: Adds el to the elements of this:
                   this_post = this_pre U { el }
          Returns true iff el was not an element of
         this_pre.

    public void union(Set<T> t)
       MODIFIES: this
       EFFECTS: this_post = this_pre U t

    public Iterator<T> iterator()
       REQUIRES: this must not be modified while the
                    iterator is in use.
       EFFECTS: Returns an iterator that yields each element of
                    this.

    public T choose()
       REQUIRES: this has at least one element
       EFFECTS: Returns an element of this.

    public boolean contains(Object el)
       EFFECTS: Returns true iff el is an element of this.

    public int size()
       EFFECTS: Returns the number of elements in this.

    public boolean isEmpty()
       EFFECTS: Returns true iff this has no elements.

    public boolean remove(Object el)
       MODIFIES: this
       EFFECTS: Removes el from this:
                    this_post = this_pre - { el }
         Returns true iff el is in this_pre
```

Task

```
public class Task
  OVERVIEW: A typical Task is < name, time, dependencies > where
     name is the name of this task, time is the time it takes
     to complete it (in minutes), and dependencies is a set
     of tasks that must be completed before this task can
     be done.

public Task (String p_name, int p_time, String [] p_dependencies)
   EFFECTS: Initializes this to the task .

public String getName ()
   EFFECTS: Returns the name of this.

public int getTime ()
   EFFECTS: Returns the time of this.

public String [] getDependencies ()
   EFFECTS: Returns the dependencies of this.

public String toFullString ()
   EFFECTS: Returns a detailed string description of this.

public String toString ()
   EFFECTS: Returns a short string description of this (just the name).
```

Vector

You may also find the `java.util.Vector` datatype provided by the Java API useful. It provides an unbounded, ordered array of objects. Like the `Set` datatype, it is generic and is parameterized with the type of the object element. For example, `Vector<String>` denotes a vector of `String` objects.

The most useful `java.util.Vector` methods are specified below:

```
public class Vector<T> {
   OVERVIEW: A Vector is a mutable, unbouded, ordered collection of
      objects of type T.  A typical Vector is [x_0, ..., x_n].

   public Vector()
      EFFECTS: Initializes this to an empty vector, [].

   public boolean add(T el)
      MODIFIES: this
      EFFECTS: Appends el to the end of this.  Returns true.
         If this_pre = [x_0, ..., x_n], this_post = [x_0, ..., x_n, el].

   public void add(int index, T el)
       REQUIRES: 0 <= index <= size
       MODIFIES: this
       EFFECTS: Inserts el at location index in this.
          All elements before index are preseved unchanged, and
          all elements after index are advanced one position.
   public boolean remove(Object el)
      MODIFIES: this
      EFFECTS: Removes the first occurance of an element whose value is
         equal to el (as matched using equals) from
         this.  Returns true there is a matching occurance; otherwise,
         returns false and leaves this unchanged.

   public boolean contains(Object el)
      EFFECTS: Returns true iff an object with the same value as
         el is an element of the Vector as determined by equals.
```

The `Vector` class also implements the `Iterator` interface, so you can iterate through the elements of a Vector (in order) using:

```
    Vector v = new Vector ();
    ...
    for (String el : v) {
       ... // Do something on each element
    }
```

Scanning

The Java API provides the `java.util.Scanner` class that makes it easier to process structured data. For this problem set, the most important `Scanner` constructors and methods you are likely to need are specified below. Note that Scanner objects can be created from both files and Strings. The specification below is quite vague — it does not specify things like what a line separator is, what character sequences form valid integers, and what separator characters are.

```
public class Scanner {
  OVERVIEW: A Scanner provides an abstract interface to textual data.
     A typical Scanner is < c1, c2, c3, ..., cn >
                                ^
     where ci is the ith character in the text, and the cursor (^)
     points to the next character to process.  A Scanner may be
     open or closed; a closed Scanner has no cursor.


  public Scanner(File source) throws FileNotFoundException
    EFFECTS: If source is not a readable file, throws FileNotFoundException.
       Otherwise, initializes this to a scanner containing the characters in
       the file source with the cursor pointing to the first character.

  public Scanner(String source)
    EFFECTS: Initializes this to a scanner containing the characters in
       the source string with the cursor pointing to the first character.

  public boolean hasNextLine()
    REQUIRES: this is an open scanner
    EFFECTS: Returns true iff there is another line in the text for
                this scanner.

  public String nextLine()
    REQUIRES: this is an open scanner and the cursor is not at the end
                 of the input
    MODIFIES: this
    EFFECTS: Returns the next line in the text (from the current cursor
       to either the end of the text or a line separator
       character).  The line separator is not included in the
       return value.  The text of the scanner is unchanged.
       The cursor advances to the position immediately
       following the line separator if one was reached, or
       to the end of the text.

  public boolean hasNextInt()
    REQUIRES: this is an open scanner
    EFFECTS: Returns true if the characters starting from the current
       cursor can be interpreted as a valid integer.

  public int nextInt()
    REQUIRES: this is an open scanner and the cursor points to a
       sequence of characters that can be interpreted as an integer.
    MODIFIES: this
    EFFECTS: Return the value of the next integer in the file (using
       as many characters as possible to form a valid integer), and
       advances the cursor to the position immediately following
       the input used.

  public boolean hasNext()
    REQUIRES: this is an open scanner
    EFFECTS: Returns true if this scanner has another token.

  public String next()
    REQUIRES: this is an open scanner with at least one more token.
    MODIFIES: this
    EFFECTS: Returns the next token in this scanner, and advances the
      cursor to point to the position immediately following the
      returned token.  The next token is as many non-whitespace characters
      as can be read until the next whitespace character.
```