CSE 143, Winter 2009
Programming Assignment #3: Stable Marriage (20 points)
Due Thursday, January 29, 2009, 11:30 PM

This program focuses on using `Set` and `Map` collections. Turn in two files named `MatchMaker.java` and `couples.txt` from the Homework section of the course web site. You will also need the support files `Person.java` and `StableMarriageMain.java` from the Homework section of the course web site; place them in the same folder as your program.

Program Description:

The "stable marriage" problem is a classic computer science problem of matching men and women into married couples. The problem examines a set of input data for an equal number of men and women, where each person has an ordered queue of preferences for whom that person would like to marry. Your task is to pair up the men with the women in such a way that no one is unsatisfied with their partner. You will use a particular algorithm to pair up ("engage") men to women and repeatedly improve the pairings as necessary until every person is satisfied.

Consider an engagement (M, W) between a man M and woman W. We say that (M, W) is *unstable* if either M or W has another person they would rather marry (and who would rather marry them as well). In other words, an unstable engagement meets one or more of the following conditions:

- There is another woman W_2 such that M and W_2 both prefer each other to their current partners; that is, M prefers the pair (M, W_2) to (M, W) and W_2 is either single or is engaged to a man she likes less than M; or,
- There is another man M_2 such that M_2 and W both prefer each other to their current partners; that is, W prefers the pair (M_2, W) to (M, W) and M_2 is either single or is engaged to a woman he likes less than W.

Gale-Shapley Algorithm for Stable Marriages:

In 1962 David Gale and Lloyd Shapley (mathematics/economics professors from Berkeley and UCLA) created an algorithm for computing a set of stable marriages for any set of *N* men's and women's preferences. This Gale-Shapley algorithm guarantees that within N passes over the data you will find a way to marry everyone with no unstable pairings.

The input into the algorithm is a set of men and a set of women, where each person carries an associated queue of preferred people to marry.

```
MEN   := set of all men.   (initially all are single)
WOMEN := set of all women. (initially all are single)
```

The algorithm consists of a series of match-making "rounds" in which pairs become engaged. A round is considered "stable" if there are no single men in the set and if no one changes their partner during the entire round. Generally match-making rounds are performed until a stable round occurs, at which point the solution is stable and the algorithm stops.

```
Algorithm for One Match-Making Round:
    for each man M in MEN:
        if M is single and still has at least one woman left in his preferences queue:
            let W = remove M's most-preferred woman remaining from the preferences queue.

            if W is single:
                (M, W) become engaged.
            else:
                W must be currently engaged to some other man M2.
                if W prefers M over M2:
                    (M, W) become engaged.
                    M2 becomes single.
```

Consider the input data on the next page. The men are listed first, then the women. Each line consists of a person's name, followed by a list of whom that person would like to marry, from most to least. For example, Jerry would most like to marry Miranda and would least like to marry Charlotte. Miranda desires Newman and would be least happy with George.

```
George:Charlotte,Carrie,Miranda,Samantha
Jerry:Miranda,Samantha,Carrie,Charlotte
Kramer:Samantha,Charlotte,Carrie,Miranda
Newman:Samantha,Charlotte,Miranda,Carrie

Carrie:Newman,Jerry,Kramer,George
Charlotte:Jerry,George,Kramer,Newman
Miranda:Newman,Jerry,Kramer,George
Samantha:Jerry,Kramer,George,Newman
```

If you run the Gale-Shapley algorithm on the above data, the algorithm produces the following results:

Round 1:
- George proposes to Charlotte.
- Jerry proposes to Miranda.
- Kramer proposes to Samantha.
- Newman's preference is Samantha, but she's engaged to Kramer and prefers him, so he does nothing.

Couples so far: (George, Charlotte), (Jerry, Miranda), (Kramer, Samantha).

Round 2:
- George is already engaged, so he does nothing.
- Jerry is already engaged, so he does nothing.
- Kramer is already engaged, so he does nothing.
- Newman's next preference is Charlotte, but she's engaged to George and prefers him, so he does nothing.

Couples so far: (George, Charlotte), (Jerry, Miranda), (Kramer, Samantha).

Round 3:
- George is already engaged, so he does nothing.
- Jerry is already engaged, so he does nothing.
- Kramer is already engaged, so he does nothing.
- Newman's next preference is Miranda; she's engaged to Jerry but prefers Newman. Jerry gets dumped.

Couples so far: (George, Charlotte), (Kramer, Samantha), (Newman, Miranda)

Round 4:
- George is already engaged, so he does nothing.
- Jerry's next preference is Samantha; she's engaged to Kramer but prefers Jerry. Kramer gets dumped.
- Kramer's next preference is Charlotte, but she's engaged to George and prefers him, so he does nothing.
- Newman is already engaged, so he does nothing.

Couples so far: (George, Charlotte), (Jerry, Samantha), (Newman, Miranda)

Round 5:
- George is already engaged, so he does nothing.
- Jerry is already engaged, so he does nothing.
- Kramer proposes to Carrie.
- Newman is already engaged, so he does nothing.

Couples so far: (George, Charlotte), (Jerry, Samantha), (Kramer, Carrie), (Newman, Miranda)

Round 6:
- No changes; algorithm has stabilized.

Provided Files:

You are given two files to use to help you write this assignment:

- `StableMarriageMain.java`: The client program to run the stable marriage simulation. Performs all file I/O.
- `Person.java`: A class representing each person in the simulation, along with their fiancee and preferences.

A set of provided data files to use as input for the algorithm is also posted on the Homework section of the web site. We will not provide testing code for you other than the `StableMarriageMain.java` program, and we do not guarantee that this program is an exhaustive test. You should perform additional testing of your own before you submit your program.

Your class will need to interact with `Person` objects to solve the overall problem. A `Person` has the following methods:

`public void engageTo(Person other)`
Sets this person to be engaged to the given other person. Subsequent calls to `getFiancee` on this person will return other, and subsequent calls to `getFiancee` on `other` will return this person. If either this person or `other` were previously engaged, their previous engagement is called off and the previous partner is set to be single.

`public String getName()`
Returns this person's name, such as `"George"`.

`public Person getFiancee()`
Returns the person to whom this person is currently engaged. If this person is single, returns `null`.

`public Queue<String> getPreferences()`
Returns a queue of the names of people this person would like to marry, in order from most preferred (front) to least preferred (back of the queue). You can and should modify the contents of this queue as your program is running. For example, you can remove a person from the queue to indicate that you have already considered proposing to that person.

`public Map<String, Integer> getRankings()`
Returns a map of this person's rankings of all other potential partners. The keys of the map are people's names, and the values are `int`s representing this person's ranking for that potential partner. You can pass a person's name to the map's get method to find out the ranking for that name. For example, if you have a `Person` object stored in a variable named `boy` and you want to know the boy's ranking for the girl named `"Carrie"`, you could write:

```
int rank = boy.getRankings().get("Carrie");
```

`public boolean isSingle()`
Returns `true` if this person has no current engagement partner (if this person's fiancee is `null`); otherwise returns `false`.

`public String toString()`
Returns a string representation of this `Person` that includes the person's name and fiancee status if any, such as `"Newman: single"` or `"George: engaged to Charlotte (rank 1)"`.

Implementation Details:

For this assignment you are to write a class called `MatchMaker` that performs the Gale-Shapley algorithm on a given set of men and women. You must write the following methods:

`public MatchMaker(Set<Person> men, Set<Person> women)`
In this method you should initialize a new match maker over the given sets of men and women. You may assume that all men and women in both sets are initially single, that each set contains at least one person, and that the sets are the same size, that no two people have exactly the same name, that the sets and their elements are not `null`, and that every `Person` object has every member of the opposite sex in his/her rankings and preferences collections.

`public Person getFiancee(String name)`
This method should return the Person object representing the current fiancee of the person with the given name. For example, if passed `"Joe"`, you should return the `Person` object representing Joe's fiancee. If the person in question is single, you should return `null`. You may assume that the name matches some `Person`'s name from the original data sets.

`public int getRound()`
In this method you should return the number of the current round of match making. When a `MatchMaker` is first created, it is on round 0. Each time the `nextMatchRound` method is called, the round number increases by 1.

`public boolean isStable()`
In this method you should return true if the `MatchMaker` is in a stable state; that is, if the Gale-Shapley algorithm has finished running to produce a stable set of engagements between the men and women. When the `MatchMaker` is initially created, it is not stable, so this method should return `false`. If `nextMatchRound` is called and the round is a "stable" round as defined previously, you should remember this and return `true` for subsequent calls to `isStable`.

`public void nextMatchRound()`
In this method you should perform a single round of the Gale-Shapley algorithm as previously described to match men and women. As you perform each round you should also make note of whether the round was "stable" as previously defined, because you will need to report this information from your `isStable` method.

`public String toString()`
In this method you should return a string representing the current state of the men and women and their engagements, with each man on one line followed by each woman on one line. For example, if a client program created a `MatchMaker` using the example data on the previous page, performed one round, then printed the `MatchMaker`, the output would be:

```
George: engaged to Charlotte (rank 1)
Jerry: engaged to Miranda (rank 1)
Kramer: engaged to Samantha (rank 1)
Newman: single
Carrie: single
Charlotte: engaged to George (rank 2)
Miranda: engaged to Jerry (rank 2)
Samantha: engaged to Kramer (rank 2)
```

Notice that the appearance of each line in the output corresponds to the `toString` representation of the `Person` object.

A major part of this assignment is using sets and maps. Some places in the code you will find that you have a `String` of a person's name, and you want to get the `Person` object that has that name. To facilitate this, each `MatchMaker` should maintain a map from names to `Person` objects. You can fill this map with data as the `MatchMaker` object is created and use it in other parts of your code. Use the map as appropriate to look up `Person` objects by name efficiently.

Creative Aspect (`couples.txt`):

In addition to your `MatchMaker.java`, turn in a text file named `couples.txt` containing your own data set of men and women and their preferences, in the same format as the provided data files. This will form a small part of your grade. To get the credit, you must have an equal number of men and women, at least 3 men and 3 women, and the data must be valid (every line in proper format, every man/woman must have every member of the opposite list in their preferences list, etc.). We will also soon provide a place online where you can submit creative data to share with others.

Development Strategy:

We suggest that you develop the program in the following stages:

1. Create your class and write "stub" versions of the methods so your code compiles and runs in a testing program.
2. Decide some of the data fields you'll want it to keep track of. Write your constructor and set up these fields.
3. Write an initial version of the `nextMatchRound` code that just pairs up each person with their first choice. We suggest that you use debugging `println` statements to print a message each time one person proposes to another.
4. Refine your `nextMatchRound` code to deal with each person's single/engaged status, preferences, etc.

Style Guidelines and Grading:

Part of your grade will come from correctly following the Gale-Shapley algorithm as described in this document. Another part of your grade will come from appropriately utilizing the collections (lists, sets, and maps) described previously. Redundancy is another major grading focus; avoid redundancy and repeated logic as much as possible.

Properly encapsulate your objects by making any data fields in your class `private`. Avoid unnecessary fields; use fields to store important data of your objects but not to store temporary values only used within a single call to one method.

You should follow good general style guidelines such as: appropriately using control structures like loops and `if`/`else` statements; avoiding redundancy using techniques such as methods, loops, and `if/else` factoring; properly using indentation, good variable names, and proper types; and not having any lines of code longer than 100 characters.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, and pre/post-conditions. For reference, our solution is around 80 lines long including comments, though you do not have to match this exactly.