Homework 1: Coding and Tooling Warm-up

1 Overview

This assignment's purpose is exercise your developer skills and to give experience with two of the tools we use in the course: version control via `git` and `github`, and builds via `maven`. By the end of this homework, you will have cloned a repository and established a maven project. You will also have made changes to files, committed them, worked on branches, merged these, and then pushed the results back to origin repository.

This assignment will be graded on your answers to the questions and the quality of your software.

If you have worked with `git` and github, please pay attention to how we use it versus policies you may have used elsewhere. For example, if you did a co-op, they may have a different policy for managing branches. Not following the directions will cause you to lose points – so please follow the directions.

You will be creating a document called `homework_1` as well as code. Any written or pasted answers should go in `homework_1`, which may be a txt, doc, or pdf. Code goes into the project you'll be working on. You will be turning in your assignment by pushing to `origin`.

There are some questions where you will cut and paste what you get in the terminal and some where you have to write out an answer. For each questions, label each answer with a bold **STEP** X where X is the step number in which the question was asked. If the answer requires something from the terminal, you can either cut and paste what's in the terminal, or you can take a snapshot of the terminal and insert that picture. In either case, **remember to include the command along with the results.**

Please keep your written answers short and sweet.

2 Assignment Prep

Prep Step One: Get `git`

If you are unsure if you have `git`, you can check this in the terminal via the command: `git --version`. If your system does not have `git` installed, install `git` from `https://git-scm/downloads`. We suggest you accept the defaults.

Prep Step Two: Get `maven`

If you are unsure if you have `maven`, you can check this in the terminal via the command: `mvn --version`. If your system does not have `maven` installed, install `maven` from `http://maven.apache.org/download.cgi`.

Prep Step Three: Open a shell/terminal window

1. If you have a mac or use linux, open a terminal shell.

2. If you use Windows, open `git bash`. Cortana can be a helpful shortcut.

All work in this assignment must be done through the terminal. While there are several nice graphical clients for managing your repository, including your IDE, all local work must be done through the command line. Knowing how to use `git` from the command line will cement an understanding of how to use `git`.

3 Assignment

3.1 Git

This section is worth 10 points overall. Each step is worth 2.5 points.

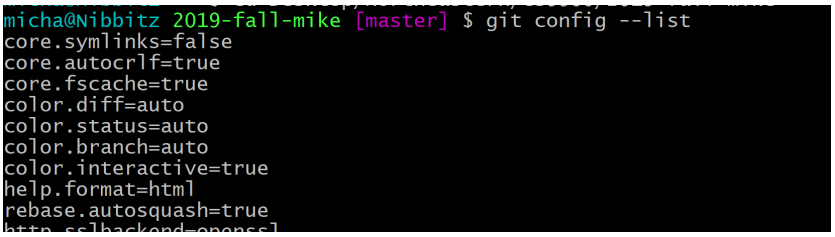
Do the following tasks in order:

Step 1: Configure git with your name, email, and favorite editor. Verify `git` knows your name, email, and editor by having `git` list your configuration.

Turn-in Task: List your `git` config's.

Hint: you should use the command `git config --list` and copy both the command and results into `homework_1`. Please follow this pattern whenever possible for the rest of the assignment. *Do not answer with an essay.*

Here's an example of exactly what you should turn in as either a cut and paste or as a snap:



Remember to turn in the entire `git` config. The example is a truncated view.

Step 2: For the rest of the assignment, you will be working with files in a repository. Since you are readying this, you found the repo. Clone this repository to your machine using git's `clone` command.

Turn-in Task: Create a listing of the local copy of the cloned repository. Copy both the command and results into `homework_1`.
Bonus turn-in task: (1/4 pt.) Create a listing of the local repo's `.git` directory and put both the command and results into `homework_1`.

Step 3: Edit the repository's README.md to describe your repository. Remember, the README.md is for people who are looking for a repo, not the author. Be creative and try something beyond: *this is the repository for Your-Name-Goes-Here*. You will be using this repository for your homework submissions. See `https://guides.github.com/features/mastering-markdown/` for formatting commands.

Turn-in Task: Alter your repo's homepage to include some descriptive text about what the repo is for and include a link to the course's web site. We'll check your github repo for this work.

Step 4: Decide which files you do not want committed to source control. Set up your `.gitignore` appropriately. You may use templates others have contributed as a guide (e.g. you can use `https://github.com/github/gitignore` for ideas). Commit your rules for this repo (meaning, your `.gitingore` should be put under `git` control). Please insure files like `Thumbs.db` or `.DS_Store` do not appear in your repo.

Turn-in Task: Put a copy of your `.gitignore` in `homework_1`.

3.2 Current Code Base

Part 1 Overview

Included in this directory is code that assigns partners. The algorithm uses a simple greedy approach to assign one set of people to another. It does not guarantee there will be a match and the matching is rather one-sided.

The code includes two classes, `Person` and `MatchMaker`, and one enumeration, `Attributes`. Here's an overview. For a complete understanding, you'll need to review the source.

`Person` — represents a person. Has four properties:

1. `name`, an English name for the person, which should be unique,
2. `preferenceList`, a `List` of `Person`,
3. `match`, a link to the person with whom this `Person` is matched,
4. `attribute`, an enum of kinds of people (e.g. male/female, advisor/student). The enum is named `Attributes`.

Each property has getters and setters.

`MatchMaker` — plays the role of matchmaker. It has three public methods.

1. `setUpGroups(List<Person> people, Attributes proposerType, Attributes proposeeType)`
   `people` is a list of `Person` and the two `Attributes` define the two categories of `Person` to be matched.
2. `makeMatches()`
   Creates the pairs. That is, `Person.match` will point at `Person` with whom they are matched.
3. `getList()`
   Lists all the `Person` the `MatchMaker` is considering as proposers or proposees.

There is also a set of tests defined using `junit4`.

This project uses maven and you can inspect the POM. You can change it if you want.

Part 2 Assignment

This section is worth 45 points. Tasks 1, the reflection, and 3, refactoring, are each worth 15 points. The remaining 15 points are distributed evenly among the other questions.

1. Review the code and write a critique in no more than two pages of any compliments or concerns. You may express this as a list. Please pin-point where in the code you see an issue unless it's a general concern. The critique should be added to `homework_1` in a separate section called Part 2.

   You need to be critical on all aspects of the code:

- correctness (does it work?),
- readability,
- expressiveness and clarity (is it clear what the code is doing and is it easy to understand),
- extensibility,
- does it follow good coding practice,
- documentation.

2. Create a branch called `refactor`.
3. Refactor the code. In less than one page, describe why the refactored code is better. Limit the type of refactoring done to what we described in class.
4. Commit the final draft of your branch and push the branch to `origin`.
5. Merge `refactor` to `master`. Push `master` to `origin`.

3.3 New Functionality

Part 1 Overview

The product owner wants a fairer approach to the matches. That is, the preferences of both groups should be respected to the greatest extent possible. If you have taken algorithms, you may recognize this as the stable-matching problem and remember the Gale-Shapley algorithm (aka the Stable Match Algorithm).

In case you don't remember or never heard of this algorithm, the following pseudocode describes it:[1]

```
while there is an unpaired man do
  pick an unpaired man X and the first woman w on X's list;
  remove w from X's list; {so it won't be picked again}
  if w is engaged then
    if w prefers X more than her current partner Y then
      set (X, w) as married;
      set (Y, w) as unmarried; {so now Y is unpaired }
    else
      X is still unpaired; {since w is happier with Y }
    end if
  else
    set (X, w) as married; {the woman was not previously paired so accept}
  end if
end while
```

Part 2 Assignment

This section is worth 45 points. Task 5, the reflection, is worth 15 points. The coding portion (tasks 2, 3, and 4) is worth 20 points. The `git` work is worth five (5) points.

Your task is to adapt/refactor either the original code or your work on the `refactor` branch so it implements the Gale-Shapley algorithm.

Here are your instructions:

[1] Adapted from `http://community.wvu.edu/~krsubramani/courses/fa01/random/lecnotes/lecture5.pdf`

1. You should work on a second branch called `feature2`.
2. You may extend or refactor the code however you think best, but you may not throw out the entire code and redo it from scratch.

   Largely you should concentrate on refactoring `MatchMaker.makeMatches()`. It may become necessary to modify `MatchMaker.setUpGroups()`. It may even be necessary to modify `Person`. You may add categories to match to `Attributes`.
3. You should add a report() method to `MatchMaker` that will list all the matches and who didn't get matched. There are no formatting requirements.
4. You may need to update or add tests to the test suite.
5. You should describe in no more than two pages why you think this project is good work. Add this answer to `homework_1` under the label Section 3.
6. You should merge your final working draft to `master` and push both `feature2` and `master` to `origin`.

How Your Code Will Be Graded

1. We will test for functioning code by running `mvn test` from the command line. The code must compile and all tests must run successfully.

   Some advice: your IDE may try to be helpful and fill in dependencies missing in your POM. Make sure you run `mvn test` in a terminal locally before you push to `origin`. Code that doesn't work from the command line will be graded as non-functioning.
2. Your code will be reviewed on all aspects of the code: readability, expressiveness and clarity, extensibility, good coding practice, and documentation.