COMP 30070 Assignment #3: Mixed Doubles Tennis Agency

Preamble: Parts 1-4 of this assignment are based on a past programming exam, so they give you an idea of the level of challenge you can expect in the 3-hour programming exam this year. This assignment may look very long, but this more due to the amount of detail provided.

Standard Exam Preamble: Attempt Parts 1-5 in sequence. Submit only one Ruby program finally. Aim to get each part working correctly before moving on to the next. The bulk of the marks will be awarded for how much you get working, so be sure to submit a working solution, commenting out sections of code if necessary. Marks are also awarded for clear design and coding, indentation and (light) commenting.

1. The MDT (Mixed Doubles Tennis) Agency provides a service whereby it matches players with other players of the opposite sex to create teams of two players. Every player has a name, a proficiency (integer, 1..10 where 1=beginner and 10=elite player) and the minimum desired proficiency of their teammate (also integer, 1..10). A sample data set for the women players is provided in `women.txt`, and for the men in `men.txt`. The format of each line in each input file is:

   `<name><proficiency><minimum desired proficiency>`

   You may assume there is always the same number of men as women.

   Implement classes in Ruby to model the MDT Agency and its players, and write a script in `main.rb` that loads the agency with data from `women_players.txt` and `men_players.txt` and then outputs the state of the agency. The expected output for the given input files is:

```
MALE Players:
Luke, proficiency: 6, seeks proficiency >= 5.
...
Sean, proficiency: 4. Seeks proficiency >= 4.

FEMALE Players:
Michelle, proficiency: 8, seeks proficiency >= 8.
...
Mary, proficiency: 5, seeks proficiency >= 5.
```

   Advice: Simplest solution is to create two classes, `Agency` and `Player`, and to use arrays to store the male and female players in the `Agency` class.

2. A *team* is a male and female player that the agency has paired together. The *satisfaction* of each player on the team is zero (perfect satisfaction) if the proficiency of their teammate is greater than, or equal to, the minimum proficiency they require in a teammate. Otherwise it's the gap between these two numbers, which is always negative. The satisfaction of a team is defined as the average satisfaction of its two members.

Add a `create_teams` method to the `Agency` class that pairs each man and woman together based on proficiency, i.e. the most proficient man with the most proficient woman, and so on. Add also a `teams_to_s` method and hence extend `main.rb` to output the teams. Expected output for the given input files is:

```
(Mary, Bob) Satisfaction: -1.5
(Alice, Sean) Satisfaction: 0.0
(Lucy, Mike) Satisfaction: -2.0
(Michelle, Luke) Satisfaction: -1.0
(Patricia, John) Satisfaction: 0.0
```

3. Add an `each_player` iterator to the `Agency` class that passes each player to the given block. Use this iterator in `main.rb` to output the names of the elite (i.e., proficiency greater than or equal to 9) players.

4. The `create_teams` method will not produce the best set of teams in the general case. The *fitness* of a set of teams is defined as the sum of the satisfaction of all the teams. Implement a *steepest-ascent hill climb* to try to improve overall team fitness as follows. Starting with the set of teams created by the `create_teams` method of (2) above, try swapping each man with every other man in turn. Apply the best swap found, and then start the swapping process afresh. Stop when no further useful swaps can be found. The `create_teams` method achieves a fitness of -4.5 on the sample data given. Steepest-ascent hill climbing can improve this to -2.0. Update `main.rb` to output the new, improved set of teams.

5. A colleague tells you that they are implementing steepest-ascent hill climb to solve an optimisation problem they have involving the placement of electrical transformers, and asks for a copy of your code. You (after groaning inwardly) explain to them what a **Template Method** is, and then refactor your solution to (4) so that the algorithm for performing a steepest-ascent hill climb is separated from the details of tennis teams, and can be reused (through inheritance) in another context.

Submission

As usual, develop your solution under Git source code control, and push regularly to your COMP30070 Assignment 3 repository in GitHub. Unit tests are expected, but 'light' testing will not be penalised. Read the assignment guidelines to make sure you're not throwing away marks needlessly.