Assignment 0219
Now that working with time and money has gotten you all warmed up, let’s go a little further. This assignment is designed to expose you to additional coding as well as introduce you to Java’s classes and objects—both built-in to the language (e.g., Math) and created by you (e.g., this assignment).

Outcomes
This assignment will affect your proficiency measures for outcomes 1a–1c, 2a–2c, and 3a–3f.

Not for Submission
Of particular relevance to this assignment is the “Classes and Objects” lesson from the Java tutorials website:
http://docs.oracle.com/javase/tutorial/java/javaOO
Sections to read or re-read, in order of importance, are: Classes, Objects, Enum Types, and More on Classes. Nested Classes is interesting but beyond the scope of this assignment.

For Submission
Your work will consist of three classes, all of which have been started for you. Your task is to implement the stubbed-out methods according to the specification that is given here.

While implementing the stubs, do also study the rest of the code so that you can build upon your knowledge of Java, particularly with respect to classes and objects. I will be happy to field questions that clear up or explain various parts of the code—including, maybe, the parts marked advanced, if you dare!

I Stub You
For PicomonCard, implement the beats method. This method accepts an opponent’s card, returning true if the receiving card wins against the opponent’s card and false otherwise. When the cards are tied, beats returns false in both directions.

“Beating” a card involves taking the card’s power, applying a multiplier to it based on its element and the opponent’s element, then comparing that product to the corresponding product for the opponent’s card.

The following rules provide the element-vs.-element multipliers for the card game:
• Fire is thrice as effective against air
• Air is twice as effective against water and earth
• Water is twice as effective against fire and earth
• Earth is four times as effective against fire

For PicomonDeck, implement the shuffle and orderedEquals methods.

Shuffling the deck should guarantee a new card order while, of course, retaining the same cards from before. Recommendation: use a standard riffle shuffle, making sure to accommodate decks with both an odd or even number of cards.

The provided PicomonDeck implements equals such that two decks are considered equal if they have the same cards, regardless of the order of the cards. The orderedEquals methods is stricter, returning true if and only if two decks have the same cards and the cards are in the same order.

Most of the action lives in PicomonGame. The game’s rules are simple: given two decks, cards are compared in order. When one card beats the other, the winning deck stays on that card while the losing deck moves to the next one. In the event of a tie, both decks move to the next card. The “match” is over when one deck runs out of cards. For this class, implement the following:

• public boolean isMatchOver() returns whether one of the decks has been depleted.
• public Player getLeader() returns the deck with more cards remaining. When both decks have the same number of cards, the game is considered tied and getLeader should return null.
• public Round playRound() returns the result of one “round”—i.e., a single comparison of the current cards from each deck. The state of the game should be updated so that the loser of the round moves to the next card in the deck.
• public Round[] playMatch() returns the complete sequence of round results from beginning to end (i.e., when one deck runs out).

The existing code in PicomonGame, especially its instance variables, constructors, and inner classes, provide clues for implementing these methods. They are there by design—use them!

**Gotta Call Them All**

Complete your work by implementing the main method of PicomonGame so that it (auto-)plays a game at the command line. The user should see the “play-by-play” for the match—the beginning decks, which cards were drawn at each round, who won each round, and finally the overall winner.

Two types of matches should be possible: a “random” mode where the match uses two randomly-created decks, and a “preset” mode where the match is based on shuffled versions of the same deck, to be specified on the command line.

- If the user does not supply any arguments, a “random” match is played. The program generates two random decks and plays the decks against each other. (note that random deck generation has already been done for you)

- If the user supplies arguments, the arguments are interpreted as the definition of a deck (see the format below). The program generates two decks with the same set of cards, shuffles each deck a random number of times, then plays the decks (with the same cards) against each other.

- If the user-supplied arguments diverge in any way from the prescribed “deck of cards” format, the program should display the message below then exit:

```java
Cannot create a deck based on the supplied arguments.
```

The command line format should be a sequence of string pairs of the form `<element> <power>`, each pair representing a card in the preset deck to be played. Elements can be `fire`, `earth`, `water`, or `air`, while power must be an integer greater than zero. Display the aforementioned error message and quit if the arguments diverge from these conditions in any way. Note that an odd number of arguments can be dismissed immediately as invalid. Other problems will only be caught as you process the arguments (e.g., integers less than 1).

For example, the following command plays two decks of the same five cards: a fire card with power 5, an earth card with power 10, a water card with power 20, an earth card with power 2, and an air card with power 18.

```
java PicomonGame fire 5 earth 10 water 20 earth 2 air 18
```

Finally, here is some sample output for a full game played in random mode, just so you can get a feel for the final product. Pardon the small font—just zoom in to see it :-P

```
$ java PicomonGame
Gym leader's deck: [Carleen (AIR, 80), Josh (AIR, 12), Jake (AIR, 3), Evan (WATER, 68), Christopher (FIRE, 80), Jake (AIR, 48), Matthew (EARTH, 1), Savannah (AIR, 82), Evan (WATER, 53), Mary (WATER, 35)]
Trainer's deck: [Carleen (AIR, 59), Justin (FIRE, 64), Josh (AIR, 11), Ryan (EARTH, 7), Evan (WATER, 44), Ryan (EARTH, 92), Filip (WATER, 96), Isabella (WATER, 21), Brent (EARTH, 53), Jake (FIRE, 65)]
Gym leader's Carleen (AIR, 80) beats Trainer's Carleen (AIR, 59)!
Trainer's Justin (FIRE, 64) beats Gym leader's Carleen (AIR, 80)!
Trainer's Justin (FIRE, 64) beats Gym leader's Josh (AIR, 12)!
Gym leader's Evan (WATER, 68) beats Trainer's Justin (FIRE, 64)!
Gym leader's Evan (WATER, 68) beats Trainer's Josh (AIR, 12)!
Gym leader's Evan (WATER, 68) beats Trainer's Ryan (EARTH, 7)!
Gym leader's Evan (WATER, 68) beats Trainer's Evan (WATER, 44)!
Gym leader's Evan (WATER, 68) beats Trainer's Ryan (EARTH, 92)!
Trainer's Filip (WATER, 96) beats Gym leader's Evan (WATER, 68)!
Trainer's Filip (WATER, 96) beats Gym leader's Christopher (FIRE, 80)!
It's a tie between Gym leader's Jake (AIR, 48) and Trainer's Filip (WATER, 96)!
Trainer's Isabella (WATER, 21) beats Gym leader's Matthew (EARTH, 1)!
Gym leader's Savannah (AIR, 82) beats Trainer's Isabella (WATER, 21)!
Gym leader's Savannah (AIR, 82) beats Trainer's Brent (EARTH, 53)!
Trainer's Jake (FIRE, 65) beats Gym leader's Savannah (AIR, 82)!
Gym leader's Evan (WATER, 93) beats Trainer's Jake (FIRE, 65)!
```

As before, a test program is available on the course website. Feel free to expand the number of test cases covered by this program.

You are also not restricted solely to the methods provided or stubbed in the code. If you think that a new method will help you implement the assigned ones, then go ahead and write it.

**How to Turn It In**

Upload your code to your GitHub repository. Don't forget to commit as you go.