CS 1410: Word Zap!

Game rules: each player begins the game with seven letters. Players take turns eliminating their letters by forming words that use their letters (but only their letters), and the first player to eliminate all of their letters wins. If a player cannot form a valid word using their letters, they may choose to pass their turn and receive an additional letter. Letters are selected randomly by the game.

Assignment

Your assignment is to create a Python program that uses a class that represents a player to play the game described above for two or more players.

Your program will start by asking how many players will play the game, and then ask for the name of each of the players. Seven random letters should be assigned to each player, and then game play should begin.

For each player, in turn, your program will print the player's name and their letters, and then ask the player for a word. Your program should check to ensure that the word entered by the player consists only of the player's letters (and also ensure that the player does not use a letter more times than that letter occurs in the player's letters).

If the word contains a letter that the player does not have, then this word should be discarded without any effect to the game, and the player should be asked for another word. Otherwise, all letters used by the word should be eliminated from the player's letters.

If the player does not enter a word, then their turn should be considered a pass, and an additional random letter should be added to the player's letters.

After the last player completes their turn, game play should continue with the first player and proceed until a player eliminates all of their letters. At this point, the name of the winner should be printed and your program should terminate. (You can choose if the game waits to the end of a round to check for wins, or if it checks after every player's turn.)

Sample

Program execution:

```
Welcome! Time to play! Try to use all of your letters.
The first player that uses all of their letters wins!

How many players will be playing? 2
Enter the name for player #1: Luke
Enter the name for player #2: Leia

Great! Now we can play!

Luke, it is your turn!
Your letters are: s n s a w n o
Enter a word to play (or press enter to pass) saw
Great job!

Leia, it is your turn!
Your letters are: t a h e e d b
Enter a word to play (or press enter to pass) heed
Great job!

Okay! Next round!

Luke, it is your turn!
Your letters are: n s n o
Enter a word to play (or press enter to pass) sons
Check your letters and try again!

Luke, it is your turn!
Your letters are: n s n o
Enter a word to play (or press enter to pass)
```

```
You get another letter, "l"!

Leia, it is your turn!
Your letters are: t a b
Enter a word to play (or press enter to pass) bat
Great job!

Leia wins!!
```

Instructions

For this assignment you will need to make 2 files: `player.py` and `main.py`. Unlike the previous assignments we are only going to provide unit tests for your Player class. Your files need to be in the same folder as the unittest files.

You must follow the specifications exactly, but may choose your own method for solving the problem described for each.

Instructions for `player.py`

Create a new class `Player`

Data Members

`name`

A string containing the user's entered name.

`letters`

A list containing the letters in the player's hand.

Methods

`__init__`

The constructor takes 1 parameter `name` a string. It should store the name and initialize a data member `letters` a list to store the users letters.

```
p = Player('Luke')
```

`getName`

This method receives no additional parameters and returns the value of the `name` data member.

```
p = Player('Leia')
p.getName() -> 'Leia'
```

`getLetters`

This method receives no additional parameters and returns the value of the `letters` data member. This should currently return an empty list until the next method is created.

```
p = Player('Luke')
p.getLetters() -> []
```

`drawLetter`

This method adds a randomly chosen letter and adds it to the player's list of letters, it returns the letter the player drew. You might consider using the code below to choose your letter from (it is the character frequency of letters in a popular anagram game)

```
letters =
'aaaaaaaaabbccddddeeeeeeeeeeeeeffgggghhiiiiiiiiijkllllmmnnnnnnnooooooooppqrrrrrrssssttttttuuuuvvwwxyyz'
```

example usage:

```
p = Player('Leia')
p.getLetters() -> []
p.drawLetter() -> 'a' # random letter
p.getLetters() -> ['a']
p.drawLetter() -> 'n' # random letter
p.getLetters() -> ['a', 'n']
```

After you finish this method, update your constructor to call it 7 times so a player will start out with 7 random letters in their hand. *Note: because the letters are chosen at random your output will be different than the examples.*

```
p = Player('Luke')
p.getLetters() -> ['n', 'e', 'f', 'g', 'e', 'a', 'z']

p2 = Player('Leia')
p2.getLetters() -> ['u', 's', 'b', 'a', 'c', 'e', 'n']
```

`printLetters`

This method takes the user's letters and formats them into a pretty printed form. It should return a string of the letters separated by a single space. There should be no trailing space.

```
p2 = Player('Leia')
p2.getLetters() -> ['u', 's', 'b', 'a', 'c', 'e', 'n']
p2.printLetters() -> 'u s b a c e n'
```

`checkWord`

This method takes 1 additional parameter `word` a string. If the word contains a letter that the player does not have, or the player does not have enough copies of the letter it should return `False`. Otherwise, update the player's current letters removing the letters used to play the word and then return `True`.

```
p = Player('Luke')
# you might consider temporarily forcing the letters while testing
# make sure to reference your data member storing the letters.
p.letters = ['n', 'e', 'f', 'g', 'e', 'a', 'z']
p.getLetters() -> ['n', 'e', 'f', 'g', 'e', 'a', 'z']
p.checkWord('feed') -> False
p.getLetters() -> ['n', 'e', 'f', 'g', 'e', 'a', 'z']
p.checkWord('gag') -> False
p.getLetters() -> ['n', 'e', 'f', 'g', 'e', 'a', 'z']
p.checkWord('gene') -> True
p.getLetters() -> ['f', 'a', 'z']
```

Instructions for `main.py`

We are not providing unittests for `main.py` because it is too difficult to test a program this dynamic, we will have to rely on Acceptance Testing to verify it works as expected. It is up to you to figure out how to finish the game using the player class. Here is a list of functions that might be useful to you. You are not required to create any of them. *As you work on this if you see any other functions that might be useful to other students feel free to suggest them.*

- `getUserInt` - Similar to `getUserFloat` from the previous 2 assignments but instead of casting the user input to a `float`, cast it to an `int`
- `getUserString` - Similar to `getUserString` from previous assignments, except an empty string should be ok (so a user can skip their turn)
- `getPlayers` - Ask the user how many players will be playing (consider using `getUserInt`). Ask the user for a name and create an instance of `Player` for the number of entered players. Return a list of the `Player` instances.
- `convertToLower` - Convert a user's entered `word` to lowercase. So if they type in an `A` it counts as `a`.

Your file should consist of of `main()` function which is called to execute your program.

Pass-off instructions

1. To pass off this assignment you need to show your completed program to the lab assistants.
   - Show them your `player.py` and `main.py` code
   - Run `test_all.py` - All tests MUST pass!
   - Run `main.py`
   - *The lab assistants may have additional tests they want you to run.*
2. Upload your `player.py` and `main.py` files to canvas (you may zip them), please add a comment to the top of the files with your name and time your class meets.