Programming Exercise: Generating Random Text

For the following assignments, you will start with the files provided that include several data files.

First there are two classes provided:

- The class **MarkovZero**, from the lesson, has two private variables, a String **myText**, for the training text, and **myRandom**, a random number generator. It also has a constructor to initialize **myRandom** and three methods:
    - The void method **setRandom** has one integer parameter named **seed**. Using this method will allow you to generate the same random text each time, which will help in testing your program.
    - The void method **setTraining** has one String parameter named **s**. The String **s** is used to initialize the training text. It is important that you DO NOT change this line or it may affect your output:

      ```
      myText = s.trim();
      ```

    - The **getRandomText** method has one integer parameter named **numChars.** This method generates and returns random text that is **numChars** long. Remember, for MarkovZero, this class generates each letter by randomly choosing a letter from the training text.
- The class **MarkovRunner** has two methods.
    - The void method **runMarkovZero** has no parameters. This method reads in a file the user chooses, creates a MarkovZero object, and then generates three sets of randomly generated text using the file read in to choose the random characters from.
    - The void method **printOut** is called by **runMarkovZero** to print out the random text that was generated with around 60 characters per line. DO NOT CHANGE THIS METHOD**.** You'll need output generated in this format for some of the quiz questions.

For this assignment, you will create several other Markov classes to generate text that use characters that follow other characters to predict the next possible character. The more characters you use in the prediction, the more likely the text will appear like the training text.

Assignment 1: MarkovZero and MarkovOne

For this assignment you will first experiment with the MarkovZero class, and then create a MarkovOne class.

Specifically, for this assignment, you will:

- Create MarkovZero generated texts by running the method **runMarkovZero** in MarkovRunner. Run the program twice and note that the output is different each time you run it.
- Modify the **runMarkovZero** method to call the **setRandom** method with the seed 42. Run this method at least twice. What do you observe? Now change to seed to 101. Run it at least twice. You should get different text than you got with the seed 42, but every time you run it you get the same text.
- Create a new class called **MarkovOne**. Copy the body of MarkovZero into MarkovOne. You'll only need to change the name of the constructor to MarkovOne and add the same import that MarkovZero had, and then it should compile. Right now, MarkovOne is only doing what MarkovZero did, since it is a copy of it. We will fix it shortly to use one character to predict text.
- In the class **MarkovRunner**, make a copy of the method **runMarkovZero**, and name this method **runMarkovOne**. Then change the line

  ```
  MarkovZero markov = new MarkovZero();
  ```

  to

  ```
  MarkovOne markov = new MarkovOne();.
  ```

  Try running the **runMarkovOne** method. It should compile and do exactly what **runMarkovZero** did.

Now let's fix up the class **MarkovOne** to generate text randomly by predicting the next character based on one character that follows somewhere in the training text.

- In the class MarkovOne, write the method **getFollows** that has one String parameter named **key**. This method should find all the characters from the private variable **myText** in MarkovOne that follow **key** and put all these characters into an ArrayList and then return this ArrayList. This algorithm for this method was described in "Finding Follow Sets." For example, if **myText** were "this is a test yes this is a test.", then the call

getFollows("t") should return an ArrayList with the Strings "h", "e", " ", "h", "e", "." as "t" appears 6 times. The call **getFollows("e")** should return an ArrayList with the Strings "s", "s", "s". Your method should work even if **key** is a word. Thus, **getFollows("es")** should return an ArrayList with the Strings "t", " ", "t". Next we will write a method to test this method.

- Create a new class in this project named **Tester** and a void method in this class named **testGetFollows** with no parameters. This method should create a MarkovOne object, set the training text as "this is a test yes this is a test.". Then have it call **getFollows** and print out the resulting ArrayList and also its size. Be sure to test it on the three examples above and also on the Strings "." and "t.", which occur at the end of the String.
- Now let's test **getFollows** on a file. In the Tester class, write the void method **testGetFollowsWithFile** with no parameters. This method should create a MarkovOne object, set the training text to a file the user selects (similar to the methods in MarkovRunner), and then call **getFollows**. Run your program on **confucius.txt** and look for the characters that follow "t". You should get 11548.
- In the class MarkovOne modify the method **getRandomText** so that it works for the way it should for MarkovOne. It should predict the next character, by finding all the characters that follow the current character in the training text, and then randomly picking one of them as the next character.
- You already modified the **runMarkovOne** method in the class MarkovRunner. Run this method with the random seed as 42 and the file **confucius.txt**. The first line of MarkovOne random text generated starts with:

```
nd are, Prevedowalvism n thastsour tr ndsang heag ti. the ffinthe
```

Assignment 2: MarkovFour and MarkovModel

For this assignment you will randomly generate text by predicting possible next characters based on all the characters that follow a substring in the training text, where the substring is made up of more than one character.

Specifically, for this assignment, you will:

- Create the class **MarkovFour** to use four characters to predict the next character. Copy and paste in MarkovOne and then modify it. You can watch the video “Implementing Order-Two” on how to create MarkovTwo from MarkovOne.
- In the **MarkovRunner** class, create the method **runMarkovFour** to generate random text using the MarkovFour class. If you set the random seed with 25 and run this method on **confucius.txt**, the first line of text should start with:
  `ouses the people Minister said the that a many Project of it`
- Create the class **MarkovModel** to use N characters to predict the next character. An integer should be passed in with the constructor to specify the number of characters to use to predict the next character. Copy and paste in MarkovFour and then modify it.
- In the **MarkovRunner** class, create the method **runMarkovModel** to generate random text using the MarkovModel class. If you set the random seed with 38 and run this method with N = 6 on **confucius.txt**, the first line of text should start with:
  `sters I could thrice before downloading, and his lord, might`