Word N-Grams

WordGram Class Implementation

Using WordGram

- As part of designing class, think of uses
  - How will WordGram be used in program?
- Create WordGram of N words from String[]
  - Analog of random String of N characters
- Add new String to last part of WordGram
  - Analog of adding follow char to make new key

```
String current = myText.substring(index, index + myOrder);
```

```
current = current.substring(1)+ follows.get(index);
```

Initial Design of WordGram

- State: Array of Strings
    - Stored in instance variable, construct from array
- Simple Behavior
    - get method to get length, like String
    - get method to get String at index, like String
- Behave like other classes
    - **`.toString()`** method for printing
    - **`.equals()`** method for finding follows
    - Do we need **`.compareTo()`**? Comparable?

Constructor

- Initialize state, an array of strings

```
public class WordGram {

  private String[] myWords;

  public WordGram(String[] source, int start, int size) {
      myWords = new String[size];
     System.arraycopy(source, start, myWords, 0, size);
  }
```

Duke
UNIVERSITY

Constructor

- Initialize state, an array of strings

```
public class WordGram {

  private String[] myWords;

  public WordGram(String[] source, int start, int size) {
      myWords = new String[size];
     System.arraycopy(source, start, myWords, 0, size);
  }
```

Constructor

- Initialize state, an array of strings
  - Copy from source array

```
public class WordGram {

  private String[] myWords;

  public WordGram(String[] source, int start, int size) {
      myWords = new String[size];
     System.arraycopy(source, start, myWords, 0, size);
  }
```

Duke UNIVERSITY

Constructor

- Initialize state, an array of strings
  - Copy from source array
  - Specified number of Strings from start index

```
public class WordGram {

  private String[] myWords;

  public WordGram(String[] source, int start, int size) {
      myWords = new String[size];
     System.arraycopy(source, start, myWords, 0, size);
  }
```

Constructor

- Initialize state, an array of strings
    - Copy from source array
    - Specified number of Strings from start index

```
public class WordGram {

  private String[] myWords;

  public WordGram(String[] source, int start, int size) {
      myWords = new String[size];
     System.arraycopy(source, start, myWords, 0, size);
  }
```

Simple Behavior

- Analogs of behavior in String class

```
public String wordAt(int index) {
   if (index < 0 || index >= myWords.length) {
     throw new IndexOutOfBoundsException("bad index "+index);
   }
   return myWords[index];
}

public int length(){
   return myWords.length;
}
```

Simple Behavior

- Analogs of behavior in String class
  - Get word **`.wordAt(i)`**, error for bad indexes

```
public String wordAt(int index) {
   if (index < 0 || index >= myWords.length) {
     throw new IndexOutOfBoundsException("bad index "+index);
   }
   return myWords[index];
}

public int length(){
   return myWords.length;
}
```

Simple Behavior

- Analogs of behavior in String class
  - Get word **`.wordAt(i)`**, error for bad indexes

```
public String wordAt(int index) {
   if (index < 0 || index >= myWords.length) {
     throw new IndexOutOfBoundsException("bad index "+index);
   }
   return myWords[index];
}

public int length(){
   return myWords.length;
}
```

Duke
UNIVERSITY

Simple Behavior

- Analogs of behavior in String class
  - Get word **`.wordAt(i)`**, error for bad indexes

```
public String wordAt(int index) {
   if (index < 0 || index >= myWords.length) {
     throw new IndexOutOfBoundsException("bad index "+index);
   }
   return myWords[index];
}

public int length(){
   return myWords.length;
}
```

Simple Behavior

- Analogs of behavior in String class
  - Get word **.wordAt(i)**, error for bad indexes
  - Get number of words stored, **.length()**

```
public String wordAt(int index) {
   if (index < 0 || index >= myWords.length) {
     throw new IndexOutOfBoundsException("bad index "+index);
   }
   return myWords[index];
}

public int length(){
   return myWords.length;
}
```

Behave Well with Other Classes

- Create `.toString()` method
  - Helps with print debugging
  - Needed to append to StringBuilder
- Create `.equals()` method
  - When is this WordGram equal to another?
  - Lengths are the same
  - `x.wordAt(i) == y.wordAt(i)`

Duke
UNIVERSITY

Behave Well with Other Classes

- Create `.toString()` method
    - Helps with print debugging
    - Needed to append to StringBuilder
- Create `.equals()` method
    - When is this WordGram equal to another?
    - Lengths are the same
    - ~~`x.wordAt(i) == y.wordAt(i)`~~
    - `x.wordAt(i).equals(y.wordAt(i))`

Duke UNIVERSITY

Changes in WordGram Markov

- Generating random key from index
  - Markov 2 character: "th", "er", …

```
int index = myRandom.nextInt(myText.length() - myOrder);
String key = myText.substring(index, index + myOrder);
```

  - Markov 2 words: "how long", "no such"

```
int index = myRandom.nextInt(myText.length - myOrder);
WordGram key = new WordGram(myText,index, myOrder);
```