List.java :

```
/** A linked list of character data objects.
 *  (Actually, a list of Node objects, each holding a reference to a character data
object.
 *  However, users of this class are not aware of the Node objects. As far as they
are concerned,
 *  the class represents a list of CharData objects. Likwise, the API of the class
does not
 *  mention the existence of the Node objects). */
public class List {

    // Points to the first node in this list
    private Node first;

    // The number of elements in this list
    private int size;

    /** Constructs an empty list. */
    public List() {
        first = null;
        size = 0;
    }

    /** Returns the number of elements in this list. */
    public int getSize() {
          return size;
    }

    /** Returns the first element in the list */
    public CharData getFirst() {
        return first.cp;
    }

    /** GIVE Adds a CharData object with the given character to the beginning of
this list. */
    public void addFirst(char chr) {
        Node newNode = new Node(new CharData(chr));
        newNode.next = first;
        first = newNode;
        size++;
    }

    /** GIVE Textual representation of this list. */
    public String toString() {
        StringBuilder builder = new StringBuilder();
        double cumulativeProbability = 0.0;
        Node current = first;
        builder.append("(");
        while (current != null) {
            cumulativeProbability += current.cp.p;
```

```
            String probability = String.format("%.4f",
current.cp.p).replaceAll("0*$", "").replaceAll("(\\.)$", "$1");
            String cumulative = String.format("%.4f",
cumulativeProbability).replaceAll("0*$", "").replaceAll("(\\.)$", "$1");
            probability = probability.contains(".") ? probability : probability +
".0";
            cumulative = cumulative.contains(".") ? cumulative : cumulative + ".0";
            if (cumulative.endsWith(".")) {
        cumulative += "0";
        }
            builder.append(String.format("(%c %d %s %s)", current.cp.chr,
current.cp.count, probability, cumulative));
            if (current.next != null) {
                builder.append(" ");
            }
            current = current.next;
        }
        builder.append(")");
        return builder.toString();
    }

    /** Returns the index of the first CharData object in this list
     *  that has the same chr value as the given char,
     *  or -1 if there is no such object in this list. */
    public int indexOf(char chr) {
        Node current = first;
        int index = 0;
        while (current != null) {
            if (current.cp.chr == chr) return index;
            current = current.next;
            index++;
        }
        return -1;
    }

    /** If the given character exists in one of the CharData objects in this list,
     *  increments its counter. Otherwise, adds a new CharData object with the
     *  given chr to the beginning of this list. */
    public void update(char chr) {
        Node current = first;
        while (current != null) {
            if (current.cp.chr == chr) {
                current.cp.count++;
                return;
            }
            current = current.next;
        }
        addFirst(chr);
    }

    /** GIVE If the given character exists in one of the CharData objects
```

```
     *  in this list, removes this CharData object from the list and returns
     *  true. Otherwise, returns false. */
    public boolean remove(char chr) {
        Node prev = null;
        Node current = first;
        while (current != null) {
            if (current.cp.chr == chr) {
                if (prev == null) first = current.next;
                else prev.next = current.next;
                size--;
                return true;
            }
            prev = current;
            current = current.next;
        }
        return false;
    }

    /** Returns the CharData object at the specified index in this list.
     *  If the index is negative or is greater than the size of this list,
     *  throws an IndexOutOfBoundsException. */
    public CharData get(int index) {
        if (index < 0 || index >= size) throw new IndexOutOfBoundsException("Index
invalide: " + index);
        Node current = first;
        for (int i = 0; i < index; i++) current = current.next;
        return current.cp;
    }

    /** Returns an array of CharData objects, containing all the CharData objects
in this list. */
    public CharData[] toArray() {
        CharData[] arr = new CharData[size];
        Node current = first;
        int i = 0;
        while (current != null) {
            arr[i++]  = current.cp;
            current = current.next;
        }
        return arr;
    }

    /** Returns an iterator over the elements in this list, starting at the given
index. */
    public ListIterator listIterator(int index) {
        // If the list is empty, there is nothing to iterate
        if (size == 0) return null;
        // Gets the element in position index of this list
        Node current = first;
        int i = 0;
        while (i < index) {
```

```
            current = current.next;
            i++;
        }
        // Returns an iterator that starts in that element
        return new ListIterator(current);
    }
}
```

LanguageModel.java :

```java
import java.util.HashMap;
import java.util.Random;

public class LanguageModel {

    // The map of this model.
    // Maps windows to lists of charachter data objects.
    HashMap<String, List> CharDataMap;

    // The window length used in this model.
    int windowLength;

    // The random number generator used by this model.
    private Random randomGenerator;

    /** Constructs a language model with the given window length and a given
     *  seed value. Generating texts from this model multiple times with the
     *  same seed value will produce the same random texts. Good for debugging. */
    public LanguageModel(int windowLength, int seed) {
        this.windowLength = windowLength;
        randomGenerator = new Random(seed);
        CharDataMap = new HashMap<String, List>();
    }

    /** Constructs a language model with the given window length.
     * Generating texts from this model multiple times will produce
     * different random texts. Good for production. */
    public LanguageModel(int windowLength) {
        this.windowLength = windowLength;
        randomGenerator = new Random();
        CharDataMap = new HashMap<String, List>();
    }

    /** Builds a language model from the text in the given file (the corpus). */
    public void train(String fileName) {
        In input = new In(fileName);
        String text = input.readAll();
        for (int i = 0; i <= text.length() - windowLength - 1; i++) {
            String window = text.substring(i, i + windowLength);
            char nextChar = text.charAt(i + windowLength);
            List charList = CharDataMap.getOrDefault(window, new List());
            charList.update(nextChar);
            CharDataMap.put(window, charList);
        }
    }

    // Computes and sets the probabilities (p and cp fields) of all the
    // characters in the given list. */
    public void calculateProbabilities(List probs) {
```

```
        int total = 0;
        for (int i = 0; i < probs.getSize(); i++) {
            total += probs.get(i).count;
        }
        for (int i = 0; i < probs.getSize(); i++) {
            CharData charData = probs.get(i);
            charData.p = (double) charData.count / total;
        }
    }
    //

    // Returns a random character from the given probabilities list.
    public char getRandomChar(List probs) {
        double p = randomGenerator.nextDouble();
        double cumulativeProbability = 0.0;
        for (int i = 0; i < probs.getSize(); i++) {
            cumulativeProbability += probs.get(i).p;
            if (p <= cumulativeProbability) {
                return probs.get(i).chr;
            }
        }
        return ' ';
    }

    /**
     * Generates a random text, based on the probabilities that were learned during
training.
     * @param initialText - text to start with. If initialText's last substring of
size numberOfLetters
     * doesn't appear as a key in Map, we generate no text and return only the
initial text.
     * @param numberOfLetters - the size of text to generate
     * @return the generated text
     */
    public String generate(String initialText, int textLength) {
        StringBuilder generatedText = new StringBuilder(initialText);
        for (int i = 0; i < textLength; i++) {
            String window = generatedText.substring(generatedText.length() -
windowLength);
            if (!CharDataMap.containsKey(window)) break;
            List charList = CharDataMap.get(window);
            calculateProbabilities(charList);
            char nextChar = getRandomChar(charList);
            generatedText.append(nextChar);
        }
        return generatedText.toString();
    }

    /** Returns a string representing the map of this language model. */
    public String toString() {
        StringBuilder str = new StringBuilder();
```

```
        for (String key : CharDataMap.keySet()) {
            List keyProbs = CharDataMap.get(key);
            str.append(key + " : " + keyProbs + "\n");
        }
        return str.toString();
    }

    public static void main(String[] args) {

    }
}
```