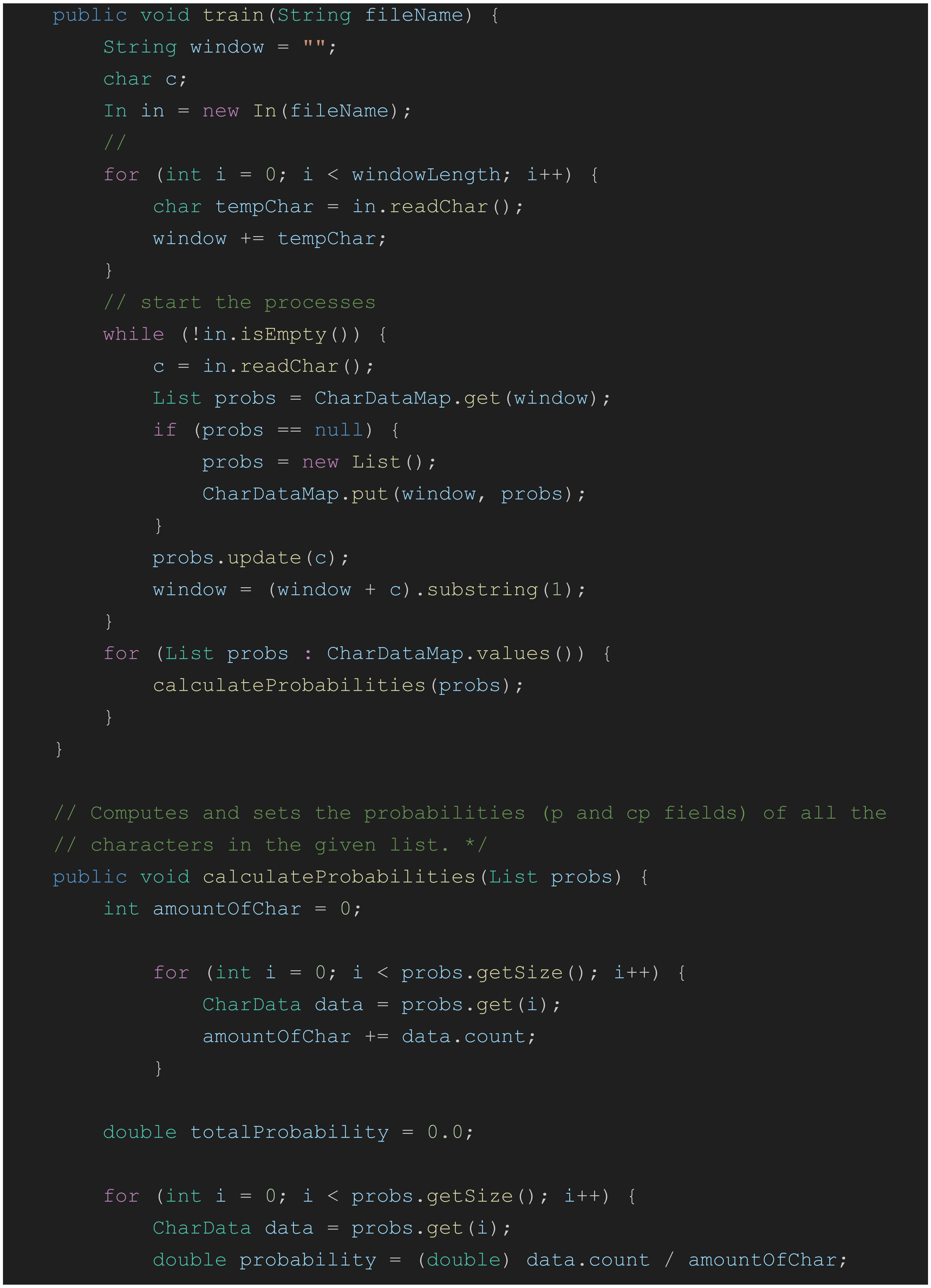
HW09 Code

List.java:

```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) {
        CharData data = new CharData(chr);
        Node newNode = new Node(data);
        newNode.next = first;
        first = newNode;
        size++;
    }

    /** GIVE Textual representation of this list. */
    public String toString() {
        Node current = first;
        String text = "(";
        while(current != null){
            if(current.next == null){
                text += current.toString()+")";
            }else{
                text += current.toString()+" ";
            }
            current = current.next;
        }
        return text;
    }

    /** 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) {
        if(size == 0){
            return -1;
        }else{
            int count = 0;
            int index = -1;
            Node current = first;
            while(current != null){
                if(current.cp.chr == chr){
                    return count;
                    //current = current.next;
                }else{
                    current = current.next;
                }
                count++;
            }
            return index;
        }
    }

    /** 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) {
        if(indexOf(chr) != -1){
            int count = 0;
            int index = indexOf(chr);
            Node current = first;
            while(current != null){
                if(count == index){
                    current.cp.count++;
                    break;
                }else{
                    current = current.next;
                }
                count++;
            }
        }else{
            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) {
        if(size == 0){
            return false;
        }else{
            int count = 0;
            int secondCount = 0;
            int inList = 0;
            Node current = first;
            while(current != null){
                count++;
                if(current.cp.chr == chr){
                    inList = 1;
                    break;
                }else{
                    current = current.next;
                }
            }
            if(inList == 1){
                current = first;
                while(current != null){
                    secondCount++;
                    if(secondCount == count - 1){
                        current.next = (current.next).next;
                        size--;
                        return true;
                    }else{
                        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.
     * @throws Exception */
    public CharData get(int index) throws IndexOutOfBoundsException {
        if(size == 0){
            throw new
IndexOutOfBoundsException("IndexOutOfBoundsException");
        }else{
            int count = 0;
            Node current = first;
            while(current != null){
                if(count == index){
                    return current.cp;
                }else{
                    current = current.next;
                }
                count++;
            }
            throw new
IndexOutOfBoundsException("IndexOutOfBoundsException");
        }
    }

    /** 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:

```
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). */
```



```java
    public void train(String fileName) {
        String window = "";
        char c;
        In in = new In(fileName);
        //
        for (int i = 0; i < windowLength; i++) {
            char tempChar = in.readChar();
            window += tempChar;
        }
        // start the processes
        while (!in.isEmpty()) {
            c = in.readChar();
            List probs = CharDataMap.get(window);
            if (probs == null) {
                probs = new List();
                CharDataMap.put(window, probs);
            }
            probs.update(c);
            window = (window + c).substring(1);
        }
        for (List probs : CharDataMap.values()) {
            calculateProbabilities(probs);
        }
    }

    // Computes and sets the probabilities (p and cp fields) of all the
    // characters in the given list. */
    public void calculateProbabilities(List probs) {
        int amountOfChar = 0;

            for (int i = 0; i < probs.getSize(); i++) {
                CharData data = probs.get(i);
                amountOfChar += data.count;
            }

        double totalProbability = 0.0;

        for (int i = 0; i < probs.getSize(); i++) {
            CharData data = probs.get(i);
            double probability = (double) data.count / amountOfChar;
            data.p = probability;

            totalProbability += probability;
            data.cp = totalProbability;
        }

        if (probs.getSize() > 0) {
            CharData lastData = probs.get(probs.getSize() - 1);
            lastData.cp = 1.0;
        }
    }

    // Returns a random character from the given probabilities list.
    public char getRandomChar(List probs) {

        double random = randomGenerator.nextDouble();
        for (int i = 0; i < probs.getSize(); i++) {
            CharData data = probs.get(i);
            if (data.cp >= random) {
                return data.chr;
            }
        }
        return probs.get(probs.getSize() - 1).chr;
    }

    /**
     * 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) {

        if (initialText.length() < windowLength) {
            return initialText;
        }
```

```java
        String window = initialText.substring(initialText.length() -
windowLength);
        String generatedText = window;

        int numberOfLetters = textLength + windowLength;
        while ((generatedText.length() < numberOfLetters)) {
            List currList = CharDataMap.get(window);

            if (currList == null) {
                break;
            }

            generatedText += getRandomChar(currList);

            window = generatedText.substring(generatedText.length() -
windowLength);
        }
        return generatedText;
    }

    /** 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) {
    }
}
```