

```

import java.util.HashMap;
import java.util.Random;
public class LanguageModel {
    // The map of this model.
    // Maps windows to lists of character 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) {
        String window = "";
        char chr = ' ';
        In in = new In(fileName);
        for (int i = 0; i < windowLength; i++)
        {
            chr = in.readChar();
            window += chr;
        }
        while (!in.isEmpty())
        {
            chr = in.readChar();
            if (CharDataMap.containsKey(window))
            {
                CharDataMap.get(window).update(chr);
            }
            else

```

```

        {
            List probs = new List();
            probs.addFirst(chr);
            CharDataMap.put(window, probs);
        }
        window = window.substring(1) + chr;
    }
    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 totalCountOfChar = 0;
    for (int i = 0; i < probs.getSize(); i++) {
        CharData data = probs.get(i);
        totalCountOfChar += data.count;
    }
    double cumulativeProb = 0.0;
    for (int i = 0; i < probs.getSize(); i++) {
        CharData data = probs.get(i);
        double probability = (double) data.count / totalCountOfChar;
        data.p = probability;
        cumulativeProb += probability;
        data.cp = cumulativeProb;
    }
    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 r = randomGenerator.nextDouble();
    for (int i = 0; i < probs.getSize(); i++) {
        CharData data = probs.get(i);
        if (data.cp >= r) {
            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) {
    String window = "";
    String mytext = initialText;
    char chr;
    if (windowLength > initialText.length() || initialText.length() >= textLength)
    {
        return initialText;
    }
    else
    {
        window = initialText.substring(initialText.length() - windowLength);
        while (mytext.length() - windowLength < textLength)
        {
            if (CharDataMap.containsKey(window))
            {
                chr = getRandomChar(CharDataMap.get(window));
                mytext += chr;
                window = window.substring(1) + chr;
            }
            else
            {
                return mytext;
            }
        }
        return mytext;
    }
}
```

*/** 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) {  
}  
}
```

```

/** 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. Likewise, 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 cd = new CharData(chr);
        Node newNode = new Node(cd);
        newNode.next = first;
        first = newNode;
        this.size++;
    }
    /** GIVE Textual representation of this list. */
    public String toString() {
        if (size == 0) {
            return "()";
        }
        String str = "(";
        Node current = first;
        while (current != null) {
            str += current.cp.toString() + " ";

```

```

        current = current.next;
    }
    return str.substring(0, str.length()-1) + "));
}
/** 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;
    boolean found = false;
    while (current != null) {
        if (current.cp.chr == chr) {
            current.cp.count++;
            found = true;
            break;
        }
        current = current.next;
    }
    if (!found) {
        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) { //lecture 8-2
    if (size == 0) {
        return false;
    }
    Node current = first;

```

```

Node prev = null;
while (current != null && current.cp.chr != chr) {
    prev = current;
    current = current.next;
}
if (current == null) {
    return false;
}
if (prev == null) {
    first = first.next;
} else {
    prev.next = current .next;
}
size--;
return true;
}
/** 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 + ":This index is out of
bounds");
    }
    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);
}
}
```