

CSC 221: Computer Programming I

Fall 2005

Repetition and Text Processing

- Java Strings: declaration, assignment, printing, concatenation
- String traversal, construction, `Character.toLowerCase`
- String methods: `length`, `charAt`, `indexOf`, `substring`
- applications: Pig Latin translator
- file input: `Scanner`, `File`, exceptions
- `Scanner` methods: `next`, `hasNext`, `nextLine`, `hasNextLine`, `nextInt`, `hasNextInt`

1

Java strings

recall: `String` is a Java class that is automatically defined for you

- a `String` object encapsulates a sequence of characters
- you can declare a `String` variable and assign it a value just like any other type

```
String firstName = "Dave";
```

- you can display `Strings` using `System.out.print` and `System.out.println`

```
System.out.println(firstName);
```

- the '+' operator concatenates two strings (or string and number) together

```
String str = "foo" + "lish";  
str = str + "ly";
```

```
int age = 19;  
System.out.println("Next year, you will be " + (age+1));
```

2

String methods

in addition, there are many useful methods defined for Strings

`int length()`
returns the length of the String str

```
String str = "foobar";
System.out.println( str.length() );
System.out.println( str.charAt(0) );
System.out.println( str.charAt(1) );
System.out.println( str.charAt(str.length()-1) );
```

`char charAt(int index)`
returns the character at specified index

- first index is 0
- last index is `str.length()-1`

```
String str = "foobar";
for (int i = 0; i < str.length(); i++) {
    System.out.print(str.charAt(i));
}
```

if `index < 0` Or `index >= str.length()`,
an error occurs

```
String str = "foobar";
for (int i = str.length()-1; i >= 0; i--) {
    System.out.print(str.charAt(i));
}
```

3

Traversing & constructing Strings

since the length of a String can be determined using the `length` method, a for loop can be used to traverse the String

- as you access individual characters, can test and act upon values
- can even construct a new string out of individual characters

```
String str = "zaboomofoo";
int count = 0;
for (int i = 0; i < str.length(); i++) {
    if (str.charAt(i) == 'o') {
        count++;
    }
}
```

```
String copy = "";
for (int i = 0; i < str.length(); i++) {
    copy = copy + str.charAt(i);
}
```

```
String copy = "";
for (int i = 0; i < str.length(); i++) {
    copy = str.charAt(i) + copy;
}
```

4

String utilities

we can define and encapsulate additional string operations in a class

- `StringUtils` will not have any fields, it simply encapsulates methods
- can define methods to be *static* – static methods can be called directly on the class

```
public class StringUtils
{
    /**
     * Reverses a string.
     * @param str the string to be reversed
     * @return a copy of str with the order of the characters reversed
     */
    public static String reverse(String str)
    {
        String copy = "";
        for (int i = 0; i < str.length(); i++) {
            copy = str.charAt(i) + copy;
        }
        return copy;
    }
    .
    .
    .
}
```

5

Stripping a string

consider the task of removing spaces from a string

- need to traverse the string and check each char to see if it is a space
- if it is not, add that char to the copy string
- if it is a space?

```
/**
 * Strips all spaces out of a string.
 * @param str the string to be stripped
 * @return a copy of str with each space removed
 */
public static String stripSpaces(String str)
{
    String copy = "";
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) != ' ') {
            copy += str.charAt(i);
        }
    }
    return copy;
}
```

6

Censoring a string

consider the task of censoring a word, i.e., replacing each vowel with '*'

- need to traverse the string and check each char to see if it is a vowel
- if it is a vowel, add '*' to the copy string
- if it is not, add the char to the copy string

```
/**
 * Censors a string by replacing all vowels with asterisks.
 * @param str the string to be censored
 * @return a copy of str with each vowel replaced by an asterisk
 */
public static String censor(String str)
{
    String copy = "";
    for (int i = 0; i < str.length(); i++) {
        if (isVowel(str.charAt(i))) {
            copy += '*';
        }
        else {
            copy += str.charAt(i);
        }
    }
    return copy;
}

/**
 * Determines if a character is a vowel (either upper or lower case).
 * @param ch the character to be tested
 * @return true if ch is a vowel, else false
 */
public static boolean isVowel(char ch)
{
    ?????????
}
```

7

Testing for a vowel

a brute force approach would be to test every vowel separately

TEDIOUS!

```
public static boolean isVowel(char ch)
{
    if (ch == 'a') {
        return true;
    }
    else if (ch == 'A') {
        return true;
    }
    else if (ch == 'e') {
        return true;
    }
    else if (ch == 'E') {
        return true;
    }
    .
    .
    .
    else if (ch == 'u') {
        return true;
    }
    else if (ch == 'U') {
        return true;
    }
    else {
        return false;
    }
}
```

8

Testing for a vowel (cont.)

we could simplify the code using the `Character.toLowerCase` method

- `toLowerCase` is a static method of the `Character` class
- it takes a character as input, and returns the lower case equivalent
- if the input is not a letter, then it simply returns it unchanged

```
public static boolean isVowel(char ch)
{
    ch = Character.toLowerCase(ch);

    if (ch == 'a') {
        return true;
    }
    else if (ch == 'e') {
        return true;
    }
    else if (ch == 'i') {
        return true;
    }
    else if (ch == 'o') {
        return true;
    }
    else if (ch == 'u') {
        return true;
    }
    else {
        return false;
    }
}
```

9

Testing for a vowel (cont.)

could simplify using `||`

```
public static boolean isVowel(char ch)
{
    ch = Character.toLowerCase(ch);

    if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
        return true;
    }
    else {
        return false;
    }
}
```

since the code returns the same value as the test, can avoid the `if` altogether

```
public static boolean isVowel(char ch)
{
    ch = Character.toLowerCase(ch);

    return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');
}
```

recall: boolean expressions involving `||` or `&&` are evaluated intelligently via *short-circuit evaluation*

- expressions are evaluated left to right
- can stop evaluating `||` expression as soon as a part evaluates to true
→ entire expression is true
- can stop evaluating `&&` expression as soon as a part evaluates to false
→ entire expression is false

10

Testing for a vowel (cont.)

best solution involves the `String` method:

```
int indexOf(char ch)
int indexOf(String str)
```

returns the index where `ch/str` first appears in the string (-1 if not found)

for `isVowel`:

- create a `String` that contains all the vowels
- to test a character, call `indexOf` to find where it appears in the vowel `String`
- if return value `!= -1`, then it is a vowel

```
public static boolean isVowel(char ch)
{
    String VOWELS = "aeiouAEIOU";

    return (VOWELS.indexOf(ch) != -1);
}
```

11

Substring

the last `String` method we will consider is `substring`:

```
String substring(int start, int end)
```

returns the substring starting at index `start` and ending at index `end-1`

```
e.g.,   String str = "foobar";
        str.substring(0,3)           → "foo"
        str.substring(3, str.length()) → "bar"
```

```
/**
 * Capitalizes the first letter in the string.
 * @param str the string to be capitalized
 * @return a copy of str with the first letter capitalized
 */
public static String capitalize(String str)
{
    return Character.toUpperCase(str.charAt(0)) + str.substring(1, str.length());
}
```

12

Pig Latin

suppose we want to translate a word into Pig Latin

- simplest version

nix → ixnay
latin → atinlay

pig → igpay
banana → ananabay

- to translate a word, move the last letter to the end and add "ay"

```
/**
 * Translates a string into Pig Latin
 * @param str the string to be converted
 * @return a copy of str translated into Pig Latin
 */
public static String pigLatin(String str)
{
    return str.substring(1, str.length()) + str.charAt(0) + "ay";
}
```

13

opsoay?

using our method,

oops → opsoay

apple → ppleaay

for "real" Pig Latin, you must consider the first letter of the word

- if a consonant, then translate as before (move first letter to end then add "ay")
- if a vowel, simply add "way" to the end

oops → oopsway

apple → appleway

```
public static String pigLatin(String str)
{
    if (isVowel(str.charAt(0))) {
        return str + "way";
    }
    else {
        return str.substring(1, str.length()) + str.charAt(0) + "ay";
    }
}
```

14

reightoncay?

using our method,

creighton → reightoncay

thrill → hrilltay

for "real" Pig Latin, if the word starts with a sequence of consonants,
must move the entire sequence to the end then add "ay"

creighton → eightoncay thrill → illthray

so, we need to be able to find the first occurrence of a vowel

HOW?

15

Handling multiple consonants

```
/**
 * Finds the first occurrence of a vowel in a string.
 * @param str the string to be searched
 * @return the index where the first vowel in str occurs (-1 if no vowel)
 */
private static int findVowel(String str)
{
    for (int i = 0; i < str.length(); i++) {
        if (isVowel(str.charAt(i))) {
            return i;
        }
    }
    return -1;
}

public static String pigLatin(String str)
{
    int firstVowel = findVowel(str);

    if (firstVowel <= 0) {
        return str + "way";
    }
    else {
        return str.substring(firstVowel, str.length()) +
            str.substring(0, firstVowel) + "ay";
    }
}
```

16

In-class exercise

modify `pigLatin` so that it preserves capitalization

computer → omlpulercaY

science → iencesay

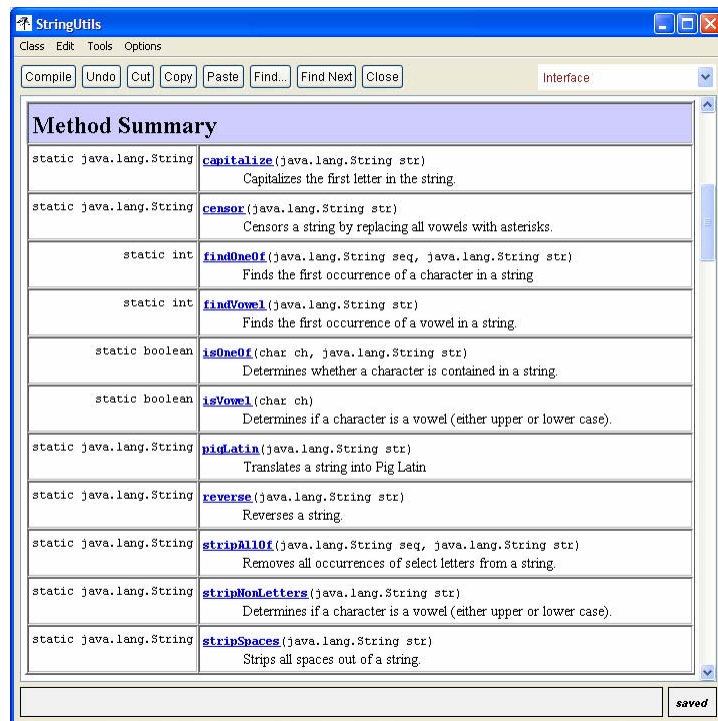
Creighton → Eightoncray

Nebraska → Ebraskanay

Omaha → Omahaway

17

Final version of StringUtils



18

String method summary

| | |
|---|---|
| <code>int length()</code> | returns number of chars in String |
| <code>char charAt(int index)</code> | returns the character at the specified index (indices range from 0 to str.length()-1) |
| <code>int indexOf(char ch)</code> <code>int indexOf(String str)</code> | returns index where the specified char/substring first occurs in the String (-1 if not found) |
| <code>String substring(int start, int end)</code> | returns the substring from indices start to (end-1) |
| <code>String toUpperCase()</code> <code>String toLowerCase()</code> | returns copy of String with all letters uppercase returns copy of String with all letters lowercase |
| <code>boolean equals(String other)</code> <code>int compareTo(String other)</code> | returns true if other String has same value returns -1 if less than other String, 0 if equal to other String, 1 if greater than other String |

ALSO, from the Character class:

| | |
|---|--|
| <code>char Character.toLowerCase(char ch)</code> | returns lowercase copy of ch |
| <code>char Character.toUpperCase(char ch)</code> | returns uppercase copy of ch |
| <code>boolean Character.isLetter(char ch)</code> | returns true if ch is a letter |
| <code>boolean Character.isLowerCase(char ch)</code> | returns true if lowercase letter |
| <code>boolean Character.isUpperCase(char ch)</code> | returns true if uppercase letter 19 |

Testing code

when you design and write code, how do you know if it works?

- run it a few times and assume it's OK?

to be convinced that code runs correctly in all cases, you must analyze the code and identify special cases that are handled

- then, define a test data set (inputs & corresponding outputs) that covers those cases
- e.g., for Pig Latin,
 - words that start with single consonant: "foo" → "oofay" "banana" → "ananabay"
 - words that start with multiple consonants: "thrill" → "illthray" "cheese" → "eesechay"
 - words that start with vowel: "apple" → "appleway" "oops" → "oopsway"
 - words with no vowels: "nth" → "nthway"
 - words that are capitalized: "Creighton" → "Eightoncray" "Omaha" → "Omahaway"

Palindrome

suppose we want to define a method to test whether a word is a palindrome (i.e., reads the same forwards and backwards)

```
isPalindrome("bob") → true           isPalindrome("madam") → true
isPalindrome("blob") → false        isPalindrome("madame") → false
```

download `StringUtils.java`

define `isPalindrome` method returns true if the word is a palindrome

extend your `isPalindrome` method to handle phrases

- need to ignore spaces, punctuation & capitalization

```
isPalindrome("Madam, I'm Adam.") → true
isPalindrome("Able was I ere I saw Elba.") → true
isPalindrome("A man, a plan, a canal: Panama.") → true
```

21

File input

so far, our programs have required little input from the user

- sufficient to specify parameters when constructing an object or calling a method
- what if we want to process lots of data?

Java 5.0 introduced the `Scanner` class that has methods for reading input from a stream (e.g., from the console window or a file)

```
Scanner input = new Scanner(System.in);           // creates a Scanner for reading
                                                    // from the console window

Scanner infile = new Scanner(new File("in.txt")); // creates a Scanner for reading
                                                    // from the file in.txt
```

`Scanner` methods exist for reading data values from the stream

```
public String next();           // reads & returns the next String (up to whitespace)
public boolean hasNext();       // returns true if a String remains to be read

public String nextLine();       // reads & returns the next line (up to <CR>)
public boolean hasNextLine();   // returns true if a line remains to be read

public int nextInt();           // reads & returns the next int value
public boolean hasNextInt();    // returns true if an int remains to be read

public int nextDouble();        // reads & returns the next int value
public boolean hasNextDouble(); // returns true if an int remains to be read
```

22

Revisiting HW3

```
public class ComparePlans
{
    private ServicePlan plan1;
    private ServicePlan plan2;
    private ServicePlan plan3;
    private ServicePlan plan4;
    private ServicePlan plan5;

    public ComparePlans()
    {
        plan1 = new ServicePlan("Sprint PCS Fair and Flexible America 400",
            24, 35.00, 34.99, 400, 0.05);
        plan2 = new ServicePlan("Sprint PCS Fair and Flexible America 700",
            24, 35.00, 49.99, 700, 0.05);
        plan3 = new ServicePlan("U.S. Cellular National 800",
            24, 30.00, 49.95, 800, 0.40);
        plan4 = new ServicePlan("Verizon Wireless America's Choice 450",
            24, 35.00, 39.99, 450, 0.45);
        plan5 = new ServicePlan("Verizon Wireless America's Choice 900",
            24, 35.00, 59.99, 900, 0.40);
    }

    public void compareCosts(int minutesUsed)
    {
        System.out.println("Your monthly cost for each plan would be:");
        System.out.println("-----");
        System.out.printf("%s: $%.2f\n", plan1.getName(),
            plan1.monthlyCost(minutesUsed));
        System.out.printf("%s: $%.2f\n", plan2.getName(),
            plan2.monthlyCost(minutesUsed));
        System.out.printf("%s: $%.2f\n", plan3.getName(),
            plan3.monthlyCost(minutesUsed));
        System.out.printf("%s: $%.2f\n", plan4.getName(),
            plan4.monthlyCost(minutesUsed));
        System.out.printf("%s: $%.2f\n", plan5.getName(),
            plan5.monthlyCost(minutesUsed));
    }
}
```

recall the
ComparePlans
class

creates a field for
each of the five
service plans

plan details are
hard-coded into
the constructor
calls

23

Service plan input file

```
Sprint PCS Fair and Flexible America 400
24
35.00
34.99
400
0.05
Sprint PCS Fair and Flexible America 700
24
35.00
49.99
700
0.05
U.S. Cellular National 800
24
30.00
49.95
800
0.40
Verizon Wireless America's Choice 450
24
35.00
39.99
450
0.45
Verizon Wireless America's Choice 900
24
35.00
59.99
900
0.40
```

a more general solution might
involve a text file:

- store the details for each of the five plans in a text file
- can then read the details in when you want to create a ServicePlan object

24

```

import java.util.Scanner;
import java.io.File;

public class ComparePlans
{
    private ServicePlan plan1;
    private ServicePlan plan2;
    private ServicePlan plan3;
    private ServicePlan plan4;
    private ServicePlan plan5;

    public ComparePlans(String filename) throws java.io.FileNotFoundException
    {
        Scanner infile = new Scanner(new File(filename));
        plan1 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan2 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan3 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan4 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan5 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

    }
    . . .
}

```

HW3 using an input file

Scanner is defined in
java.util.Scanner

File is defined in
java.io.File

- in order to use these classes, must import the files

you must specify what is to happen if the file is not found

in Java, you either have to define code to handle such errors (later), or simply "throw an exception"

25

```

import java.util.Scanner;
import java.io.File;

public class ComparePlans
{
    private ServicePlan plan1;
    private ServicePlan plan2;
    private ServicePlan plan3;
    private ServicePlan plan4;
    private ServicePlan plan5;

    public ComparePlans(String filename) throws java.io.FileNotFoundException
    {
        Scanner infile = new Scanner(new File(filename));
        plan1 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan2 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan3 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan4 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

        infile.nextLine();
        plan5 = new ServicePlan(infile.nextLine(), infile.nextInt(),
                                infile.nextDouble(), infile.nextDouble(),
                                infile.nextInt(), infile.nextDouble());

    }
    . . .
}

```

HW3 using an input file

ComparePlans is now more general

can change the plans being compared simply by changing the input file

WHY THE EXTRA nextLine() ?

26

File processing

recall: `Scanner` has methods for detecting whether more input is available

- `hasNext`, `hasNextLine`, `hasNextInt`, `hasNextDouble`

can combine these with a `while` loop to process arbitrarily large files

```
Scanner infile = new Scanner(new File("words.txt"));

int wordCount = 0;
while (infile.hasNext()) {
    String nextWord = infile.next();
    wordCount++;
}

System.out.println("The file contains " + wordCount + " words.");
```

```
Scanner infile = new Scanner(new File("nums.txt"));

int sum = 0;
int count = 0;
while (infile.hasNextInt()) {
    int num = infile.nextInt();
    count++;
    sum += num;
}

System.out.println("The average is " + (double)sum/count);
```

27

Document class

```
import java.util.Scanner;
import java.io.File;

public class Document
{
    private int numWords;

    public Document(String filename) throws java.io.FileNotFoundException
    {
        numWords = 0;

        Scanner infile = new Scanner(new File(filename));
        while (infile.hasNext()) {
            String nextWord = infile.next();
            numWords++;
        }
    }

    public int getNumWords()
    {
        return numWords;
    }
}
```

this simple class processes a document

- it reads words one at a time (delineated by whitespace)
- it counts the words as they are read in
- the word count field is accessible via `getNumWords`

28

Document class

```
import java.util.Scanner;
import java.io.File;

public class Document
{
    private int numWords;
    private int longest;

    public Document(String filename) throws java.io.FileNotFoundException
    {
        numWords = 0;
        longest = 0;

        Scanner infile = new Scanner(new File(filename));
        while (infile.hasNext()) {
            String nextWord = infile.next();
            numWords++;
            if (longest < nextWord.length()) {
                longest = nextWord.length();
            }
        }
    }

    public int getNumWords()
    {
        return numWords;
    }

    public int longestWordLength()
    {
        return longest;
    }
}
```

here, add an additional field is used to keep track of the longest word length

could easily envision other file attributes being stored

- number of capitalized words
- number of vowels

29

Document class

```
import java.util.Scanner;
import java.io.File;

public class Document
{
    private int numWords;
    private int longest;
    private int numCaps;
    private int numVowels;

    public Document(String filename) throws java.io.FileNotFoundException
    {
        numWords = 0;
        longest = 0;
        numCaps = 0;
        numVowels = 0;

        Scanner infile = new Scanner(new File(filename));
        while (infile.hasNext()) {
            String nextWord = infile.next();
            numWords++;

            if (longest < nextWord.length()) {
                longest = nextWord.length();
            }

            if (nextWord.charAt(0) == Character.toUpperCase(nextWord.charAt(0))) {
                numCaps++;
            }

            for (int i = 0; i < nextWord.length(); i++) {
                if (StringUtils.isVowel(nextWord.charAt(i))) {
                    numVowels++;
                }
            }
        }
    }
}
```

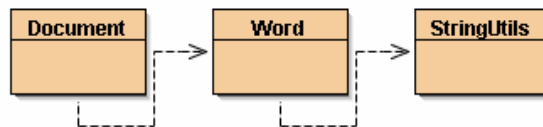
this is getting messy!

30

Object-oriented design

OOD is all about abstraction

- a *cohesive* class models one type of thing, with minimal complexity
- *loosely coupled* classes separate out independent behaviors, minimal interaction
- a better design would be to separate out the operations on words
get # of characters, determine if capitalized, count vowels
- from the overall document processing
read the file, get the longest, get # of capitals, get # of vowels



31

Word class

```
public class Word
{
    private String str;

    public Word(String word)
    {
        str = word;
    }

    public int getNumChars()
    {
        return str.length();
    }

    public int getNumVowels()
    {
        int count = 0;
        for (int i = 0; i < str.length(); i++) {
            if (StringUtils.isVowel(str.charAt(i))) {
                count++;
            }
        }
        return count;
    }

    public boolean isCapitalized()
    {
        return str.charAt(0) == Character.toUpperCase(str.charAt(0));
    }
}
```

the `word` class can encapsulate all the messy operations on a word

does this simple job & does it well!

32

Document class

```
import java.util.Scanner;
import java.io.File;

public class Document
{
    private int numWords;
    private int longest;
    private int numCaps;
    private int numVowels;

    public Document(String filename) throws java.io.FileNotFoundException
    {
        numWords = 0;
        longest = 0;
        numCaps = 0;
        numVowels = 0;

        Scanner infile = new Scanner(new File(filename));
        while (infile.hasNext()) {
            Word nextWord = new Word(infile.next());
            numWords++;

            if (longest < nextWord.getNumChars()) {
                longest = nextWord.getNumChars();
            }

            if (nextWord.isCapitalized()) {
                numCaps++;
            }

            numVowels += nextWord.getNumVowels();
        }
    }
}
```

the Document class is then much simpler

can deal with an abstract word without worrying how it works

33

HW5: Flesch Readability Index

HW5 involves processing a text file and calculating a readability index

Flesch Index = $206.835 - 84.6 * (\text{avg \# syllables per word}) - 1.015 * (\text{avg \# words per sentence})$

you will define two classes:

- word class encapsulates a String, with additional operations
 - numChars*: number of characters, excluding punctuation
 - numSyllables*: (estimated) number of syllables in the word
 - endsWithPunctuation*: returns true if word ends with a punctuation mark
- Document class encapsulates a text document
 - numWords*: total number of words in the document
 - numSyllables*: total number of syllables in the document
 - numSentences*: total number of sentences in the document
 - fleschIndex*: the calculated Flesch readability index
 - gradeLevel*: the education level corresponding to that index

| Flesch Index | Grade Level | Examples |
|--------------|-----------------------|---------------------|
| 90 - 100 | 5 th grade | comics |
| 80 - 90 | 6 th grade | consumer ads |
| 70 - 80 | 7 th grade | Alice in Wonderland |
| 65 - 70 | 8 th grade | Sports Illustrated |
| 50 - 65 | high school student | Time Magazine |
| 30 - 50 | college student | New York Times |
| 0 - 30 | college graduate | Auto insurance |
| < 30 | law school graduate | IRS tax code |

34