String Methods (Outsource: 7-1 – 7-13)

The following is an abridged list of methods provided by the String class. For a more detailed description of the String methods see the JDK Help files.

Method Summary – String class	
`char`	**`charAt`**`(int index)` Returns the `char` value at the specified index.
`boolean`	**`equals`**`(Object anotherObject)` Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.
`boolean`	**`equalsIgnoreCase`**`(String anotherString)` Compares this `String` to another `String`, ignoring case considerations.
`int`	**`indexOf`**`(int ch, int fromIndex)` Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
`int`	**`indexOf`**`(String str)` Returns the index within this string of the first occurrence of the specified substring.
`int`	**`indexOf`**`(String str, int fromIndex)` Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
`int`	**`length`**`()` Returns the length of this string.
`String`	**`replace`**`(char oldChar, char newChar)` Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.
`String[]`	**`split`**`(String regex)` Splits this string around matches of the given regular expression.
`String[]`	**`split`**`(String regex, int limit)` Splits this string around matches of the given regular expression.
`boolean`	**`startsWith`**`(String prefix)` Tests if this string starts with the specified prefix.
`String`	**`substring`**`(int beginIndex)` Returns a new string that is a substring of this string.
`String`	**`substring`**`(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.
`char[]`	**`toCharArray`**`()` Converts this string to a new character array.
`String`	**`toLowerCase`**`()` Converts all of the characters in this `String` to lower case using the

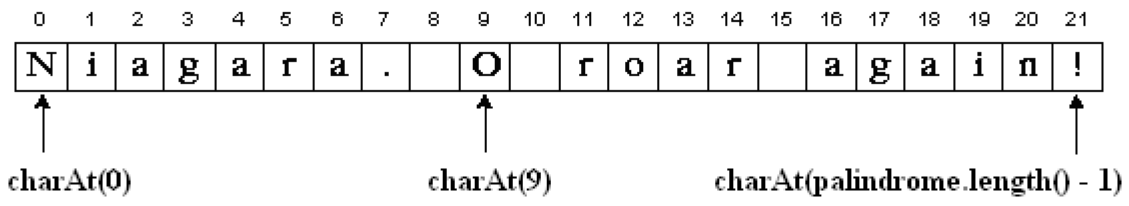
	rules of the default locale.
`String`	`toString()` This object (which is already a string!) is itself returned.
`String`	`toUpperCase()` Converts all of the characters in this `String` to upper case using the rules of the default locale.
`String`	`trim()` Returns a copy of the string, with leading and trailing whitespace omitted.
`static String`	`valueOf(char c)` Returns the string representation of the `char` argument.
`static String`	`valueOf(char[] data)` Returns the string representation of the `char` array argument.
`static String`	`valueOf(double d)` Returns the string representation of the `double` argument.
`static String`	`valueOf(int i)` Returns the string representation of the `int` argument.

Getting Characters By Index From A String

You can get the character at a particular index within a string by using the charAt method. The index of the first character is 0; the index of the last is length()-1. For example, the following code gets the character at index 9 in a string:

```
String palindrome = "Niagara. O roar again!";
char aChar = palindrome.charAt(9);
```

Indices begin at 0, so the character at index 9 is 'O', as illustrated in the following figure:



indexOf

The indexOf method searches inside the source string for a "target" string. The indexOf method returns the index number where the target string is first found (searching left to right), or -1 if the target is not found. The search is case-sensitive - upper and lowercase letters must match exactly.

```
String string = "Here there everywhere";
int a = string.indexOf("there");  // a is 5
int b = string.indexOf("er");     // b is 1
int c = string.indexOf("eR");     // c is -1, "eR" is not found
```

length

The length method returns the number of chars in a string. An empty string "" returns a length of 0.

```
String a = "Hello";
int len = a.length();  // len contains 5
```

replace

Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

```
String str1 = “Hexxo”;
String str2 = str1.replace(‘x’, ‘l’);        // str2 contains “Hello”
```

split

The split method splits a string according to the given regular expression. The number of resultant substrings by splitting the string is controlled by limit argument (if used).

```
String str1 = "one-two-three";
String[] list = str.split(“-“);       // list contains {“one”, “two”, “three”}
```

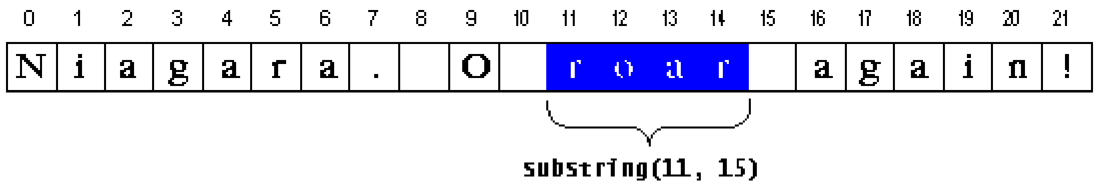
substring (Outsource: 7-12)

The substring method returns part of a string. It uses index numbers to identify the part of the string you want. The simplest version of substring takes a single int index value and returns a new string made of the chars starting at that index and extending through the end of the string.

The more complex substring(int start, int end) method takes both start and end index numbers, returning a string of the chars between start and end – 1.

The following code gets the substring that extends from index 11 to index 15, which is the word "roar", from the string palindrome:

```
String palindrome = "Niagara. O roar again!";
String roar = palindrome.substring(11, 15);
```



Remember that indices begin at 0. Also remember that substring goes up to but does not include the end index. The length of the result is (end - start). It's easy to make off-by-one errors with index numbers in any algorithm. Make a little drawing of a sample string with its index numbers to get your code exactly right.

toLowerCase

The toLowerCase method returns a new string which is a lowercase copy of the source string. It does not change the source string.

```
String a = "Hello";
String b = a.toLowerCase();  // b contains "hello"
```

toUpperCase

The toUpperCase method is like toLowerCase, but returns an all uppercase copy of the source string. It does not change the source string.

```
String a = "Hello"
String b = a.toUpperCase();  // b contains "HELLO"
```

trim

The trim method returns a copy of the source string but with whitespace chars (like space, tab, newline) removed from the start and end of the string. It does not remove whitespace everywhere, only at the front and end. It does not change the source string
.

```
String a = "   Hello   There   ";
String b = a.trim();  // b contains "Hello There"
```

Comparing Strings (Outsource: 7-15)

To test a string for ***equality*** call the string's equals method:

```
String str = "JVHS";
If  ( str.equals("JVHS") )     /* if str contains "JVHS" */
{
     /*  process the variable str  */
}
```

Comparing Strings For Inequality

To test a string for ***inequality*** (NOT EQUAL TO) place a *not* operator before the call to the string's equals method.

```
String str = reader.next();
if ( !str.equals("JVHS" )      /* if str DOES NOT contain "JVHS" */
{
     /*  process the variable str  */
}
```

LAB 14 - ASSIGNMENT

Lab 14A - 70 points

OBJECTIVE

WAP to read the names of cities from a data file (**“lab14a.dat”**). Input the name of a city from the keyboard and determine if it is equal to any of the city names contained in the data file, paying attention to case sensitivity. Uppercase letters *are not the same* as lowercase letters.

FIELD SUMMARY

- **String[ ] list** – an array of fifteen city names
- **String city** – the name of a city to be compared with all the values in *cities*.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to input and output.
- **fileInput** – declare a Scanner object and read the names of fifteen cities from a data file (**“lab14a.dat”**) storing them in ***list***. Each city name is on a separate line in the data file.
- **kybdInput** – declare a Scanner object and read the name of a city from the keyboard.
- **output** – use a loop to compare the city read from the keyboard with every city in ***list*** and output whether the cities are the same or not.

SAMPLE DATA FILE INPUT

<Data File Contents Not Shown>

SAMPLE KEYBOARD INPUT

Enter the name of a city: **Miami**

SAMPLE OUTPUT

```
Miami is Not equal to Houston.
Miami is Not equal to Phoenix.
Miami is equal to Miami.
Miami is Not equal to Chicago.
Miami is Not equal to Dayton.
Miami is Not equal to Los Angeles.
Miami is Not equal to Buffalo.
Miami is Not equal to New York City.
Miami is Not equal to San Francisco.
Miami is Not equal to Denver.
Miami is Not equal to Seattle.
Miami is Not equal to Boston.
Miami is Not equal to Montgomery.
Miami is Not equal to New Orleans.
Miami is Not equal to Dallas.
```

Lab 14B - 80 points

OBJECTIVE

WAP that will determine if a word is spelled correctly involving "ie" and "ei." If the word is spelled incorrectly, then the program will correct the spelling. (You may assume that there are no exceptions to the phonics rule, "'I' before 'E' except after 'C.'") All data will be input from a data file (**"lab14b.dat"**).

At first this appears to be a very complex problem, however if we break it down into parts it turns out to be fairly simple as you will see in the following *pseudo* code.

1. read the words from the data file <store them in a String[ ] variable>
2. for every word in the array of words:
    1. search for the index location where **"ei"** occurs in the word. ***While*** the index location is not -1 replace the **"ei"** with **"ie"** using the substring method. Search again for the index location where **"ei"** occurs in the word. (The while loop should replace every instance of **"ei"** with **"ie"**.)
    2. Using the same technique descriped in step 1 replace every instance of **"cie"** with **"cei"**.
    3. Compare the original word with the corrected word. If the two words are the same, the word was spelled correctly, otherwise it was misspelled.

Note: For this problem you are required to use **`indexOf`** and **`substring`** to solve the problem. You are not allowed to use the **`replace`** method of the string class. That would be too easy!

FIELD SUMMARY

- **String[] word** – the words read from the data file.
- **int count** – a counter to determine the number of words read from the data file.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to input and output.
- **input** – declare a Scanner object and read a series of words from the data file (**"lab14b.dat"**) storing them in the array **`word`**.
- **correctSpelling** – Arguments: a String – the word to be corrected. Return: a String. Correct the spelling of the String received as an argument by following steps 2-1 and 2-2 in the pseudo code shown above and return the corrected word. This method should be **static**.
- **output** – For every string in the array word: call **`correctSpelling`** passing the current string as an argument and storing the results in a local String variable. If the original word and the corrected words are the same then output that the original word was spelled correctly, otherwise output the correctly spelled word. Step 2-3 in the pseudo code shown above.

SAMPLE DATA FILE INPUT

```
receive
conceive
sieve
believe
recieve
```

```
concieve
seive
believe
```

SAMPLE OUTPUT 1

```


The word "receive" is spelled correctly.

The word "conceive" is spelled correctly.

The word "sieve" is spelled correctly.

The word "believe" is spelled correctly.

The word "recieve" is spelled incorrectly.
It should be spelled "receive".

The word "concieve" is spelled incorrectly.
It should be spelled "conceive".

The word "seive" is spelled incorrectly.
It should be spelled "sieve".

The word "beleive" is spelled incorrectly.
It should be spelled "believe".
```

Lab 14C - 90 points

OBJECTIVE

WAP that will rearrange a string of numbers according to the following rule:

If two even numbers are next to each other, insert an X or a Z between them. If the larger even number comes first, insert an X; if the numbers are the same or the second is larger, insert a Z. (You may assume that only digits 1-9 will be used.)

The following *pseudo* code might help in solving this problem.

```
for every character in the String up to but NOT INCLUDING the last character
  • If this character is an even number and the character immediately following is an even
    number then
      ➢ If this character is greater than the character immediately following it then insert an 'X'
        between this character and the character immediately following it using the substring
        method of the String class.
      ➢ Otherwise (this character is less than or equal to the character immediately following it)
        insert a 'Z' between this character and the character immediately following it using the
        substring method of the String class.
```

Make this program run three times.

FIELD SUMMARY

- **String sequence** – a sequence of numbers read from the keyboard.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to input and output.
- **input** – declare a Scanner object and read a series of numbers in the form of a String from the keyboard using an appropriate prompt.
- **process** – follow the pseudo code shown above to rebuild the word using the String's substring method.
- **output** – display the modified word.
- **isEven** – **Arguments:** a char. **Return:** true if the argument is an even number; otherwise return false. This method should be **static**.

SAMPLE KEYBOARD INPUT 1

Enter a sequence of numbers: **432413649**

SAMPLE OUTPUT 1

```
432Z4136X49
```

SAMPLE KEYBOARD INPUT 2

Enter a string of numbers: **22381463**

SAMPLE OUTPUT 2

```
2Z23814Z63
```

Lab 14D - 100 points - THIS CLASS ALREADY EXISTS -

OBJECTIVE

Pig Latin has nothing to do with pigs and remarkably little in common with Latin, other than being a way of communicating. It is a coded way of talking, based on English and used chiefly by children who think or believe that this system allows them to speak without being understood by others. Parents whose children don't know pig Latin have also been known to use it in order to speak "privately" in their children's presence. The reference to Latin may simply be because using pig Latin has some of the same feel and effects of speaking another language in terms of secrecy, or because of the sound of the language.

In Pig Latin each word of a sentence is changed by moving the first letter of the word to the end and adding "ay." For example, the word *hello* would be written and pronounced *ellohay*.

However, there are two exceptions to this rule.

- If a word begins with a vowel simply add yay to the end of the word. For example the word *and* would become *andyay*.

- If a word begins with a blend, such as "th" or "str", then the blend is moved to the end of the word and "ay" is added. For example the word *the* would become *ethay*.

Two-character Consonant Blends

bl	br	ch	cl	cr	dr	dw	fl	fr	gl
gr	kl	ph	pl	pr	qu	sc	sh	sk	sl
sn	sm	sp	sq	st	sw	th	tr	tw	wh
wr									

Three-character Consonant Blends

sch	scr	spl	spr	squ	str	thr

`translate(String word)` receives a String argument and returns a Pig Latin translation of the argument (also a String). You can include all the private helper methods you need to perform the translation. For example:

```
String s = "Happy ";
String pigLatin = translate(s);
System.out.println(pigLatin);
```

would result in the following output: **appyhay**

You can assume that NO punctuation will be used. You may implement as many helper methods you want.

FIELD SUMMARY

- **static String[ ] threeLetterBlends** – an array of strings containing three letter blends.
- **static String[] twoLetterBlends** – an array of strings containing two letter blends.

METHOD SUMMARY

- **startsWithThreeLetterBlend(String word)** – Returns true if the argument starts with a three letter blend.
- **startsWithTwoLetterBlend(String word)** – Returns true if the argument starts with a two letter blend.
- **startsWithVowel(String word)** – Returns true if the argument starts with a vowel.
- **translate(String word)** –Translates the argument into its corresponding Pig Latin following the rules as described above. Return the translated word as a String.

SAMPLE KEYBOARD INPUT

Enter a sentence: **The Life And Times Of Judge Roy Bean**

SAMPLE OUTPUT

```
The original sentence was:
The Life And Times Of Judge Roy Bean

Translated into Pig Latin:
ethay ifelay andyay imestay ofyay udgejay oyray eanbay
```