CSCI 161
Introduction to Computer Science

Est. 1888
UNIVERSITY of PUGET SOUND

Department of Mathematics
and Computer Science

Lecture 5
Strings

Exploring Strings

- Speaking of *abstraction*, one of the classes/objects you've been using the *whole semester* are **`Strings`**.

- **`Strings`** are objects that represent a sequence of **chars**:
  - Recall that a **char** is a primitive data type, that can hold a single symbol.
  - Each character in the string corresponds to a position (or address, or index)

- A String **`"Hello World!\n"`** is represented in the machine as:

	H	e	l	l	o		W	o	r	l	d	!	\n
Addr	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]

Strings Are Objects?

- Yes, **`String`** is a class in Java!
  - Each string also has access to various methods.

- Then **`Strings`** must have a constructor? Yes they do!
  - We've never seen the **`new`** keyword being used to instantiate strings.
  - Java hides it from us though.

```java
String message = new String("Hello World!");    // This works!
```

```java
String message = "Hello World";    // But this syntax is more convenient!
```

Strings Are Special Objects

- Strings are the most commonly used objects in Java.
  - Java had to make them super convenient to use!

- **#1:** Strings have a short-hand constructor
  - (Use of the **new** keyword is not necessary for creating string objects)

```
String message = "Hello World";    // This construction syntax is convenient!
```

- **#2:** Strings have their own handy-dandy operator: **+**
  - To "concatenate" two strings

```
"Hello" + " " + "World"
```

Are Strings "Mutable?"

- We've been calling methods on objects to change (mutate) their state.
  - For instance:

```
Circle c = new Circle();
c.changeColor("blue");
c.moveUp();
c.moveLeft();
c.slowMoveVertical(-40);
```

```
OrcaCard myCard = new OrcaCard();
myCard.topUp(10);
myCard.buyTrip(5);
```

- How about Strings?

```
String str = "hello";
str.toUpperCase();
System.out.println(str);    // what will this print?
```

Immutability of Strings

- **#3** Strings are *"immutable."*
  - Unlike objects we've seen, their methods do not change their internal state!
- Then what good are their methods??
  - They can return a new string, though.
- In the previous example, how do you capture the upper-case version?
  - (You need capture or re-capture its return value)

```
String str = "hello";
str = str.toUpperCase();   // re-capture the upper-case version in str
System.out.println(str);   // This prints HELLO
```

== cannot be used to compare Strings!

- **#4:** Strings cannot be compared using **==** and **!=**
  - Because Strings are objects, these comparison operators cannot be used reliably.
  - They sometimes work, but not always.

```
String s1 = "hello";
String s2 = "he";
s2 += "llo";

if (s1 == s2) {
   // Doesn't work! s1 and s2 point to different things
   System.out.println("Does this work?");
}

if (s1.equals(s2)) {
   // This works!
   System.out.println("Does this work?");
}
```

The String API (Selected Methods)

Length

`public boolean isEmpty()`	Returns true if and only if the `length()` is zero.
`public int length()`	Gets the length of the String.

Comparison

`public int compareTo(String other)`	Returns 0 if strings are equal, -1 if current string is "less than" input; positive value if current string is "greater than" input.
`public boolean equals(String other)`	Tests if two strings are equal. Case sensitive.

Extraction

`public char charAt(int pos)`	Returns the character at the given position, `pos`.
`public String[] split(String delimiter)`	Splits the string into substrings around the given delimiter.
`public String substring(int begin)`	Returns a copy of the String starting from position `begin` to the end.
`public String substring(int begin, int end)`	Returns a copy of the String starting from position `begin`, ending at position `end – 1`.

The String API (2)

Manipulation

`public String toLowerCase()`	Returns a copy of the String in lower case.
`public String toUpperCase()`	Returns a copy of the String in upper case.
`public String trim()`	Returns a copy of the String omitting any leading and trailing spaces.

Search and Replace

`public int indexOf(String str)`	Returns starting position of str if found, or -1 if not found in the current string
`public String replace(String key, String rep)`	Returns a copy of the String after replacing all occurrences of key with rep

Examples using String Methods

- Getting the length of a **`String`**
  - This method is widely used

```
String school = "University of Puget Sound";
int size = school.length(); // size gets 25
```

- Search and Replace (case-sensitive)

```
String name = "Adam A. Smith";
String shortenedName = name.replace("A. ", "");
System.out.println(shortenedName);  //Adam Smith
```

Extraction Examples

- Extracting a Substring
  - Keep in mind that Java subtracts 1 from the given **end** index.

- Example: Extract first name:

```java
String fullname = "Brad Richards";
String firstname = fullname.substring(0, 4);
```

- Example: Extract last name:

```java
String fullname = "Brad Richards";
String lastname = fullname.substring(5, fullname.length());
```

Example: Taiwan Phone Numbers

- In Taiwan, a landline phone number follows these patterns:
  - `"(02)xxx-xxxx":` Taipei; `"(04)xxx-xxxx":` Taichung; `"(07)xxx-xxxx":` Kaohsiung

- Write **public void** location(String phoneNumber) that prints the location of the given phone number.
  - Anything longer or shorter is badly formatted.
  - The hyphen should be in position 7.
  - There should only be 2 digits for the "area code."

```
location("(04)439-9182");
> Taichung

location("(07)312-0992");
> Kaohsiung

location("(2079)2-0282");
> Bad format
```

Example: Create a class, StringExercise

- Puget Sound email addresses are formed using the first initial appended to the last name appended to @thu.edu.tw

- **Write** an email creation method called **`createEmail()`**:
  - Two input parameters: first name & last name
  - Returns a Puget Sound email address in lowercase
    - But if *either* input is an **empty string ("")**, return an **empty string**.

- Example Usage:

```
String myEmail = createEmail("David", "Chiu");
System.out.println(myEmail); // outputs "dchiu@thu.edu.tw"
```

```
String myEmail = createEmail("David", "");
System.out.println(myEmail); // outputs ""
```

Email Solution

```java
/**
 * Creates a puget sound email
 * @param first The first name of student
 * @param last  The last name of student
 * @return the email address, or empty string if either name is not given
 */
public String createEmail(String first, String last) {
    if (first == null || last == null || first.isEmpty() || last.isEmpty()) {
       // one or both inputs were empty
       return "";
    }

    // convert both to lower case
    first = first.toLowerCase();
    last = last.toLowerCase();
    return first.charAt(0) + last + "@thu.edu.tw";
}
```

Your Turn: Pig-Latin-fying Words

- Write a method `pigLatin(String word)` that inputs a word and returns the Pig Latin version of the word.
  - If a word starts with a consonant, swap that letter to the back, hyphenate, and concatenate **"ay"** to it.
  - If a word starts with a vowel, just concatenate **"-way"** to that word

```
System.out.println(pigLatin("hello"));
> ello-hay

System.out.println(pigLatin("Mice"));
> ice-May

System.out.println(pigLatin("circle"));
> ircle-cay

System.out.println(pigLatin("apple"));
> apple-way

System.out.println(pigLatin(""));
>
```

Pig Latin Solution

```
/**
 * Pig Latinfies a word.
 * @param word
 * @return the pig-latin version of the specified word
 */

public String pigLatin(String word) {
    if (word == null || word.isEmpty()) {
        return "";
    }

    if (isVowel(word.charAt(0))) {
        // first letter is a vowel
        return word + "-way";
    }
    // first letter is a consonant
    return word.substring(1) + "-" + word.charAt(0) + "ay";
}


/**
 * @return true if the given character is a vowel; false otherwise
 */
private boolean isVowel(char c) {
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}
```

Your Turn!

- Write a method called vowelsAtEnds:
  - Inputs a word (String)
  - Returns **true** if word starts *and* ends with vowel; **false** otherwise
    - Assume standard vowels only: a,e,i,o,u
    - Don't assume word will be given in lower case

- Example Usage:

```
System.out.println(vowelsAtEnds("Ada"));             // true
System.out.println(vowelsAtEnds("ice cream"));       // false
System.out.println(vowelsAtEnds("  UMBRELLA   "));   // true (oooh, spaces)
System.out.println(vowelsAtEnds(""));                // false
```

Solution

```java
/**
 * Tests whether a string starts and ends with a vowel.
 *
 * @param  s Some given string to test
 * @return true if the given string starts and ends with a vowel
 *         false otherwise, or if string is empty.
 */
public boolean vowelsAtEnds(String str) {
    if (str == null || str.isEmpty()) {
        return false;
    }
    // trim leading and trailing spaces and convert to lower case
    str = str.trim().toLowerCase();
    return isVowel(str.charAt(0)) && isVowel(str.charAt(str.length()-1));
}

/**
 * @return true if the given character is a vowel; false otherwise
 */
private boolean isVowel(char c) {
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}
```