CS 106A, Lecture 9
Problem-Solving with Strings

suggested reading:

Java Ch. 8.5
Plan For Today

• Announcements
• Recap: Characters and Strings
• More Strings
• Practice: Reversing a String
• Practice: Palindromes
• Practice: Caesar Cipher
Plan For Today

• Announcements
• Recap: Characters and Strings
• More Strings
• Practice: Reversing a String
• Practice: Palindromes
• Practice: Caesar Cipher
Announcements

• Assignment 2 is due tomorrow!
  – Please submit ahead of time, just in case

• Questions during lecture:
  – Keep asking!
  – I will start directing some to Piazza 😊
Plan For Today

• Announcements
• Recap: Characters and Strings
• More Strings
• Practice: Reversing a String
• Practice: Palindromes
• Practice: Caesar Cipher
A **char** is a variable type that represents a single character or “glyph”.

```java
char letterA = 'A';
char plus = '+';
char zero = '0';
char space = ' ';
char newLine = '\n';
char tab = '\t';
char singleQuote = '\'';
char backSlash = '\\';
```
Under the hood, Java represents each `char` as an `integer` (its “ASCII value”).

- Uppercase letters are sequentially numbered
- Lowercase letters are sequentially numbered
- Digits are sequentially numbered

```java
char uppercaseA = 'A';    // Actually 65
char lowercaseA = 'a';    // Actually 97
char zeroDigit = '0';    // Actually 48
```
Char Math!

We can take advantage of Java representing each `char` as an `integer` (its “ASCII value”):

```java
boolean areEqual = 'A' == 'A';  // true
boolean earlierLetter = 'f' < 'c';  // false
char uppercaseB = 'A' + 1;
int diff = 'c' - 'a';  // 2
int numLettersInAlphabet = 'z' - 'a' + 1;
// or
int numLettersInAlphabet = 'Z' - 'A' + 1;
```
Side Note: Type-casting

If we want to force Java to treat an expression as a particular type, we can also cast it to that type.

```java
'A' + 1  // evaluates to 66 (int)
(char)('A' + 1)  // evaluates to 'B' (char)

1 / 2  // evaluates to 0 (int)
(double)1 / 2  // evaluates to 0.5 (double)
1 / (double)2  // evaluates to 0.5 (double)
```
# Character Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character.isDigit(ch)</td>
<td>true if ch is '0' through '9'</td>
</tr>
<tr>
<td>Character.isLetter(ch)</td>
<td>true if ch is 'a' through 'z' or 'A' through 'Z'</td>
</tr>
<tr>
<td>Character.isLetterOrDigit(ch)</td>
<td>true if ch is 'a' through 'z', 'A' through 'Z' or '0' through '9'</td>
</tr>
<tr>
<td>Character.isLowerCase(ch)</td>
<td>true if ch is 'a' through 'z'</td>
</tr>
<tr>
<td>Character.isUpperCase(ch)</td>
<td>true if ch is 'A' through 'Z'</td>
</tr>
<tr>
<td>Character.toLowerCase(ch)</td>
<td>returns lowercase equivalent of a letter</td>
</tr>
<tr>
<td>Character.toUpperCase(ch)</td>
<td>returns uppercase equivalent of a letter</td>
</tr>
<tr>
<td>Character.isWhitespace(ch)</td>
<td>true if ch is a space, tab, new line, etc.</td>
</tr>
</tbody>
</table>

**Remember:** toLowerCase and toUpperCase **return** the new char; they cannot modify an existing char!
## Strings

A **String** is a variable type representing a sequence of characters.

```java
String text = "Hi parents!";
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>'H'</td>
<td>'i'</td>
<td>' '</td>
<td>'p'</td>
<td>'a'</td>
<td>'r'</td>
<td>'e'</td>
<td>'n'</td>
<td>'t'</td>
<td>'s'</td>
<td>'!'</td>
</tr>
</tbody>
</table>

- Each character is assigned an *index*, going from 0 to length-1
- There is a **char** at each index
Strings vs. Chars

**Remember**: chars and length-1 strings are different!

```java
char ch = 'A'  // DIFFERENT FROM String str = “A”

'A' + 1       // evaluates to 66 (int)
“A” + 1        // evaluates to “A1” (String)
```
String str = "Hello, world!";
String empty = "";
println(str);

// Read in text from the user
String name = readLine("What is your name? ");

// String concatenation (using "+")
String message = name + " is " + 2 + " cool.";
char c1 = 'a';
char c2 = 'b';

// How do we concatenate these characters?

String str = c1 + c2; // ERROR: this is an int!

String str = "" + c1 + c2; //
Plan For Today

- Announcements
- Recap: Characters and Strings
- More Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher
A *substring* is a subset of a string.

String `str = "Hello, world!";`  
String `hello = str.substring(0, 5);`
A *substring* is a subset of a string.

String `str = "Hello, world!";`
String `worldExclm = str.substring(7, 13);`
Substrings

A *substring* is a subset of a string.

String `str = "Hello, world!";`
String `worldExclm = str.substring(7);` // to end
String Methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.length()</code></td>
<td>number of characters in this string</td>
</tr>
<tr>
<td><code>s.charAt(index)</code></td>
<td>char at the given index</td>
</tr>
<tr>
<td><code>s.indexOf(str)</code></td>
<td>index where the start of the given string appears in this string (-1 if not found)</td>
</tr>
<tr>
<td><code>s.substring(index1, index2)</code> or <code>s.substring(index1)</code></td>
<td>the characters in this string from <code>index1</code> (inclusive) to <code>index2</code> (exclusive); if <code>index2</code> is omitted, goes until end</td>
</tr>
<tr>
<td><code>s.toLowerCase()</code></td>
<td>a new string with all lowercase letters</td>
</tr>
<tr>
<td><code>s.toUpperCase()</code></td>
<td>a new string with all uppercase letters</td>
</tr>
</tbody>
</table>

- These methods are called using **dot notation**:  
  ```java
  String className = "CS 106yay!";
  println(className.length()); // 10
  ```
Strings are Immutable

Once you create a String, its contents cannot be changed.

// Cannot change individual chars in the string
String typo = "Hello, warld!";
typo.charAt(8) = 'o'; // Error! Will not run.

To change a String, you must create a new String containing the value you want (possibly using String methods).

String corrected = typo.substring(0, 8) + 'o' + typo.substring(9);
Strings are Immutable

String className = "cs 106a";
className.toUpperCase(); // does nothing!

className = className.toUpperCase(); //
println(className); // CS 106A
String greeting = "Hello!";
if (greeting == "Hello!") {  // Doesn't work!
    ...
}

// Instead:
if (greeting.equals("Hello!")) {
    ...
}

Always use .equals instead of == and !=
## Comparing Strings

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s1.equals(s2)</code></td>
<td>whether two strings contain the same characters</td>
</tr>
<tr>
<td><code>s1.equalsIgnoreCase(s2)</code></td>
<td>whether two strings contain the same characters, ignoring upper vs. lower case</td>
</tr>
<tr>
<td><code>s1.startsWith(s2)</code></td>
<td>whether <code>s1</code> contains <code>s2</code>’s characters at start</td>
</tr>
<tr>
<td><code>s1.endsWith(s2)</code></td>
<td>whether <code>s1</code> contains <code>s2</code>’s characters at end</td>
</tr>
<tr>
<td><code>s1.contains(s2)</code></td>
<td>whether <code>s2</code> is found within <code>s1</code></td>
</tr>
</tbody>
</table>
A common String programming pattern is looping over a string and operating on each character.

String str = "Hello!";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    // Do something with ch here
}
A common String programming pattern is looping over a string and operating on each character.

// Prints out each letter on a separate line
String str = "Hello!";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    println(ch);
}
A common String programming pattern is looping over a string and operating on each character.

// Creates a new String in all caps
String str = "Hello!";
String newStr = "";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    newStr = newStr + Character.toUpperCase(ch);
}
println(newStr);     // HELLO!
A common String programming pattern is looping over a string and operating on each character.

// Creates a new String in all caps
String str = "Hello!";
String newStr = "";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    newStr += Character.toUpperCase(ch);
}
println(newStr);    // HELLO!
Another common String programming pattern is building up a new string by adding characters to it over time.

```java
// Creates a new String in all caps
String str = "";
for (int i = 0; i < 5; i++) {
    str += i;
}
println(str);  // 01234
```
Plan For Today

• Announcements
• Recap: Characters and Strings
• More Strings

• Practice: Reversing a String
• Practice: Palindromes
• Practice: Caesar Cipher
Exercise: Reversing a String

Let’s write a method called `reverseString` that takes one String parameter, and returns a new String with the characters in the opposite order.

`reverseString("Hello!") -> "!olleH"`
Reversing a String

Hello!
Reversing a String
Reversing a String

Hello!
Reversing a String

Hello!

! o
Reversing a String

Hello!

! o
Reversing a String

Hello!

! o l
Reversing a String

Hello!

! o l
Reversing a String

Hello!

! o l l
Reversing a String

Hello!

! o l l
Reversing a String

Hello!

!ollel
Reversing a String

Hello!

!ollel
Reversing a String

Hello!

!olleH
Reversing a String

Hello!

!olleH
String str = "Hello!";
String newStr = "";
for (??? ; ??? ; ???) {
    ...
}

String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; ??? ; ???) {
    ...
}

Hello!
String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; i >= 0; ???) {
    ...
}

Hello!
!olleH
String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; i >= 0; i--) {
    ...
}
String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; i >= 0; i--) {
    newStr += str.charAt(i);
}
Reversing a String

Hello!
Reversing a String

Hello!

H
Reversing a String

Hello!

eH
Reversing a String

Hello!

ileH
Reversing a String

Hello!

Hello! → !olleH
Reversing a String

Hello!

olleH
Reversing a String

Hello!

!olleH
public void run() {
    String str = readLine("Enter a string: ");
    String rev = reverseString(str);
    println(str + " spelled backwards is " + rev);
}

private String reverseString(String str) {
    String result = "";
    for (int i = 0; i < str.length(); i++) {
        result = str.charAt(i) + result;
    }
    return result;
}
Plan For Today

- Announcements
- Recap: Characters and Strings
- More Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher
Let’s write a method called `isPalindrome` that takes one String parameter, and returns whether or not that String is a palindrome (the same forwards and backwards).

`isPalindrome("racecar")` -> true
`isPalindrome("hi there")` -> false
`isPalindrome("kayak")` -> true
Plan For Today

• Announcements
• Recap: Characters and Strings
• More Strings
• Practice: Reversing a String
• Practice: Palindromes
• Practice: Caesar Cipher
Exercise: Caesar Cipher

• Rotate alphabet by $n$ letters ($n = 3$ in below)
  – $n$ is called the **key**
• Wrap-around at the end
• Substitute letters based on this mapping

| original | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| encrypt  | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |
Exercise: Caesar Cipher

- Rotate alphabet by a certain key, with wrapping

| original | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| encrypt  | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |
Recap

• Recap: Characters and Strings
• More Strings
• Practice: Reversing a String
• Practice: Palindromes
• Practice: Caesar Cipher

Next time: reading text files