Building Java Programs

Chapter 4
Lecture 4-2: Strings

**reading:** 3.3, 4.3 - 4.4

**self-check:** Ch. 4 #12, 15

**exercises:** Ch. 4 #15, 16

**videos:** Ch. 3 #3
Objects and classes

- **object**: An entity that contains:
  - *data* (variables), and
  - *behavior* (methods).

- **class**: A program, or a type of objects.

**Examples:**
- The class `String` represents objects that store text.
- The class `DrawingPanel` represents graphical window objects.
- The class `Scanner` represents objects that read information from the keyboard, files, and other sources.
Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is not created with `new`.

  ```java
  String name = "text";
  String name = expression;
  ```

- **Examples:**

  ```java
  String name = "Marla Singer";
  int x = 3;
  int y = 5;
  String point = "(" + x + ", " + y + ")";
  ```
Indexes

- Characters of a string are numbered with 0-based *indexes*:

  ```java
  String name = "P. Diddy";
  ```

<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>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>P</td>
<td>.</td>
<td>D</td>
<td>i</td>
<td>d</td>
<td>d</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

- The first character's index is always 0
- The last character's index is 1 less than the string's length
- The individual characters are values of type `char` (seen later)
### String methods

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

- These methods are called using the dot notation:

  ```java
  String gangsta = "Dr. Dre";
  System.out.println(gangsta.length());  // 7
  ```
String method examples

```java
// index 012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());  // 12
System.out.println(s1.indexOf("e"));  // 8
System.out.println(s1.substring(7, 10));  // "Reg"

String s3 = s2.substring(2, 8);
System.out.println(s3.toLowerCase());  // "rty st"
```

- Given the following string:

  ```java
  // index 0123456789012345678901
  String book = "Building Java Programs";
  ```

- How would you extract the word "Java"?
- How would you extract the first word from any string?
Modifying strings

- Methods like `substring`, `toLowerCase`, etc. create/return a new string, rather than modifying the current string.

```java
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);  // lil bow wow
```

- To modify a variable, you must reassign it:

```java
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);  // LIL BOW WOW
```
Strings as parameters

```java
public class StringParameters {
    public static void main(String[] args) {
        sayHello("Marty");

        String teacher = "Helene";
        sayHello(teacher);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

**Output:**
Welcome, Marty
Welcome, Helene
Strings as user input

- **Scanner's next method reads a word of input as a String.**

  ```java
  Scanner console = new Scanner(System.in);
  System.out.println("What is your name? ");
  String name = console.next();
  name = name.toUpperCase();
  System.out.println(name + " has " + name.length() + " letters and starts with " + name.substring(0, 1));
  
  Output:
  What is your name? **Madonna**
  MADONNA has 7 letters and starts with M
  ```

- **The nextLine method reads a line of input as a String.**

  ```java
  System.out.println("What is your address? ");
  String address = console.nextLine();
  ```
Comparing strings

- Relational operators such as < and == fail on objects.

```java
Scanner console = new Scanner(System.in);
System.out.println("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!"));
}
```

- This code will compile, but it will not print the song.

- == compares objects by references (seen later), so it often gives false even when two Strings have the same letters.
The `equals` method

- Objects are compared using a method named `equals`.

```java
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.nextLine();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- Technically this is a method that returns a value of type `boolean`, the type used in logical tests.
String test methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals(str)</td>
<td>whether two strings contain the same characters</td>
</tr>
<tr>
<td>equalsIgnoreCase(str)</td>
<td>whether two strings contain the same characters,</td>
</tr>
<tr>
<td></td>
<td>ignoring upper vs. lower case</td>
</tr>
<tr>
<td>startsWith(str)</td>
<td>whether one contains other's characters at start</td>
</tr>
<tr>
<td>endsWith(str)</td>
<td>whether one contains other's characters at end</td>
</tr>
<tr>
<td>contains(str)</td>
<td>whether the given string is found within this one</td>
</tr>
</tbody>
</table>

String name = console.next();
if (name.startsWith("Dr.")) {
    System.out.println("Are you single?");
} else if (name.equalsIgnoreCase("LUMBERG") ) {
    System.out.println("I need your TPS reports.");
}
Strings question

- Write a program that reads a person's name and converts it into a "gangsta name."

Output (run 1):
Type your name, playa: Peter Griffin
(M)ale or (F)emale? m
Your gangsta name is "P. GRIFFIN Daddy Peter-izzle"

Output (run 2):
Type your name, playa: Marge Simpson
(M)ale or (F)emale? F
Your gangsta name is "M. SIMPSON Goddess Marge-izzle"
// This program prints your "gangsta" name.
import java.util.*;

public class GangstaName {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Type your name, playa: ");
        String name = console.nextLine();
        System.out.print("(M)ale or (F)emale: ");
        String gender = console.next();

        // split name into first/last name and initials
        String first = name.substring(0, name.indexOf(" "));
        String last = name.substring(name.indexOf(" ") + 1);
        last = last.toUpperCase();
        String fInitial = first.substring(0, 1);

        String title;
        if (gender.equalsIgnoreCase("m")) {
            title = "Daddy";
        } else {
            title = "Goddess";
        }

        System.out.println("Your gangsta name is \"" + fInitial + ". "
                + last + " " + title + " " + first + "-izzle\"");
    }
}
Type `char`

- **char**: A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as 'a' or '4' or '\n' or '\'`

- It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter);    // s
```

- `char` values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");  // P Diddy
```
The `charAt` method

- The **chars in a String** can be accessed using the `charAt` method.

```java
String food = "cookie";
char firstLetter = food.charAt(0);    // 'c'
System.out.println(firstLetter + " is for " + food);
System.out.println("That's good enough for me!");
```

- You can use a **for loop** to print or examine each character.

```java
String major = "CSE";
for (int i = 0; i < major.length(); i++) {
    char c = major.charAt(i);
    System.out.println(c);
}
```

**Output:**

C
S
E
**char VS. int**

- All `char` values are assigned numbers internally by the computer, called *ASCII* values.
  
  - Examples:
    
    | Character | Value |
    |-----------|-------|
    | 'A'       | 65    |
    | 'B'       | 66    |
    | ' '       | 32    |
    | 'a'       | 97    |
    | 'b'       | 98    |
    | '*'       | 42    |

  - Mixing `char` and `int` causes automatic conversion to `int`.
    
    | Expression | Value  |
    |------------|--------|
    | 'a' + 10   | 107    |
    | 'A' + 'A'  | 130    |

- To convert an `int` into the equivalent `char`, type-cast it.
  
  `(char) ('a' + 2)` is 'c'
char ** char VS. String

- "h" is a String
  'h' is a char (the two behave differently)

- String is an object; it contains methods

  ```java
  String s = "h";
  s = s.toUpperCase();  // 'H'
  int len = s.length();   // 1
  char first = s.charAt(0);   // 'H'
  ```

- char is primitive; you can't call methods on it

  ```java
  char c = 'h';
  c = c.toUpperCase();  // ERROR: "cannot be dereferenced"
  ```

- What is s + 1 ? What is c + 1 ?
- What is s + s ? What is c + c ?
Comparing `char` values

- You can compare `char` values with relational operators:
  `'a' < 'b'` and `'X' == 'X'` and `'Q' != 'q'`

- An example that prints the alphabet:
  ```java
  for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
  }
  ```

- You can test the value of a string's character:
  ```java
  String word = console.next();
  if (word.charAt(word.length() - 1) == 's') {
    System.out.println(word + " is plural.");
  }
  ```
String/char question

- A Caesar cipher is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

  Your secret message: Brad thinks Angelina is cute
  Your secret key: 3
  The encoded message: eudg wklqnv dqjholqd lv fxwh
Strings answer 1

// This program reads a message and a secret key from the user and
// encrypts the message using a Caesar cipher, shifting each letter.

import java.util.*;

public class SecretMessage {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Your secret message: ");
        String message = console.nextLine();
        message = message.toLowerCase();

        System.out.print("Your secret key: ");
        int key = console.nextInt();

        encode(message, key);
    }

    ...
}
// This method encodes the given text string using a Caesar cipher, shifting each letter by the given number of places.
public static void encode(String text, int shift) {
    System.out.print("The encoded message: ");
    for (int i = 0; i < text.length(); i++) {
        char letter = text.charAt(i);

        // shift only letters (leave other characters alone)
        if (letter >= 'a' && letter <= 'z') {
            letter = (char) (letter + shift);

            // may need to wrap around
            if (letter > 'z') {
                letter = (char) (letter - 26);
            } else if (letter < 'a') {
                letter = (char) (letter + 26);
            }
        }
        System.out.print(letter);
    }
    System.out.println();
}