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* (*fields*), and
  - *behavior* (methods).

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

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

- Non-examples: `int` and `double` are not classes
  - No methods (can't say `34.someMethod(…)`)

Strings

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

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

  - Examples:

```
String name = "Barack Obama";

int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Indexes

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

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

index	0	1	2	3	4	5	6	7
char	P	.		D	i	d	d	y

  - The first character's index is always 0
    - "How many away from the beginning"
    - "offset"
  - 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

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if it is not there)
`length()`	number of characters in this string

`substring(`**`index1`**`, `**`index2`**`)`
or
`substring(`**`index1`**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String formalName = "Mr. Combs";
System.out.println(formalName.length());    // 9
```

`String` method examples

```
//      index 012345678901
String s1 = "Dan Grossman";
String s2 = "Alan Borning";
System.out.println(s1.length());          // 12
System.out.println(s1.indexOf("a"));      // 1
System.out.println(s1.substring(7, 10)); // "ssm"

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

`String` mini-exercises

- Given the following string:

```
//          index 01234567890123456789012
String book = "Building Java Programs";
```

How would you pick out the word `"Java"` as a substring of `book`?

- Write a method to take any String and return its first word

Method name	Description
`indexOf(`**str**`)`	index where the start of the given string appears in this string (-1 if it is not there)

`substring(`**index1**`, `**index2**`)`
or
`substring(`**index1**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

Answers

- `book.substring(9,13);`

- ```
  public static String getFirstWord(String s){
     int firstSpacePlace = s.indexOf(" ");
     if(firstSpacePlace == -1) {
        return s;
     } else {
        return s.substring(0,firstSpacePlace);
     }
   }
  ```
  ```

Modifying strings

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

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

- To change which String a variable refers to, you must assign to the variable

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

Strings as parameters

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

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

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

Output:

```
Welcome, Dan
Welcome, Alan
```

Strings as user input

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

```
Scanner console = new Scanner(System.in);
System.out.print("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`.

```
System.out.print("What is your address? ");
String address = console.nextLine();
```

`String` mini-exercises (2)

- Write a method 'shout' that takes a string and prints it out in all upper-case
- Using 'shout', write a Java program that reads in a line of text and prints it out all upper case.

‘shout’ mini-exercise

```
Import java.util.*;
public class LoudMouth {

  public static void main (String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("say what? ");
    String s = console.nextLine();
    shout(s);
  }

  public static void main shout (String u) {
    System.out.println(u.toUpperCase());
  }
}
```

Comparing strings

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

```
Scanner console = new Scanner(System.in);
System.out.print("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 `String`s have the same letters.
    - Two String objects with the same contents may not be the same String and that's what == is asking
    - == works "as you expect" for non-objects (int)

The `equals` method

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

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
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

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one

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 `char`s in a `String` can be accessed using the `charAt` method.

```
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.

```
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:
    `'A'` is 65, `'B'` is 66, `' '` is 32
    `'a'` is 97, `'b'` is 98, `'*'` is 42
  - Conveniently, the alphabet is in order (`'b' < 'j'`)
  - Occasionally convenient to convert to/from `int`
    - Adding a char and an int, makes an int: `'a' + 7 // 104`
    - To convert back to a char, use a cast: `(char)('a' + 7) // 'h'`

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:

```
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

- You can test the value of a string's character:

```
String word = console.next();
if (word.charAt(word.length() - 1) == 's') {
    System.out.println(word + " is likely plural.");
}
```

char vs. String

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

- String is an object; it contains methods

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

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

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

`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: 10
The encoded message: lbkn drsxuc kxqovsxk sc medo
```

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);
    }

    ...
```

Strings answer 2

```
// This method encodes the given text string using a Caesar
// cipher, shifting each letter by the given number of places.
// Assumes shift is between -26 and +26
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();
}
}
```