

Building Java Programs

Chapter 4
Lecture 4-3: Strings; `char`; procedural design

reading: 3.3, 4.3, 4.5

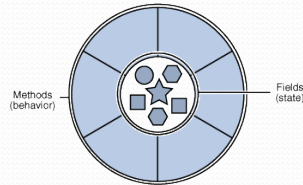
1

Strings

reading: 3.3

Objects

- **object:** An entity that contains data and behavior.
 - *data:* variables inside the object
 - *behavior:* methods inside the object
 - You interact with the methods; the data is hidden in the object.
 - A **class** is a *type* of objects.



- Constructing (creating) an object:
`<type> <objectName> = new <type> (<parameters>);`
- Calling an object's method:
`<objectName> .<methodName> (<parameters>);`

3

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 with String value>;
```

- Examples:

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

4

Indexes

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

```
String name = "M. Mouse";
```

index	0	1	2	3	4	5	6	7
character	M	.		M	o	u	s	e

- First character's index : 0
- Last character's index : 1 less than the string's length
- The individual characters are values of type `char` (seen later)

5

String methods

Method name	Description
<code>indexOf(<string>)</code>	index where the start of the given string appears in this string (-1 if not found)
<code>length()</code>	number of characters in this string
<code>substring(<index1>, <index2>)</code> or <code>substring(<index1>)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (<u>exclusive</u>); if <i>index2</i> is omitted, grabs until end of string
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String popStarz = "Prince vs. Michael";  
System.out.println(popStarz.length()); // 18
```

6

String method examples

```
// 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(1, 7);
System.out.println(s3.toLowerCase());     // "arty s"
```

- Given the following string:

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

- How would you extract the word "Java" ?

7

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "Mumford & Sons";
s.toUpperCase();
System.out.println(s);    // Mumford & Sons
```

- To modify a variable's value, you must reassign it:

```
String s = "Mumford & Sons";
s = s.toUpperCase();
System.out.println(s);    // MUMFORD & SONS
```

8

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? Bono
BONO has 4 letters and starts with B
```

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

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

9

Strings question

- Write a program that outputs "The Name Game" with a person's first and last name.

Example Output:

```
What is your name? James Joyce
```

```
James, James, bo-bames
Banana-fana fo-fames
Fee-fi-mo-mames
JAMES!
```

```
Joyce, Joyce, bo-boyce
Banana-fana fo-foyce
Fee-fi-mo-moyce
JOYCE!
```

10

Strings answer

```
// This program prints "The Name Game".
import java.util.*;

public class TheNameGame {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("What is your name? ");
        String name = console.nextLine();

        int spaceIndex = name.indexOf(" ");
        String firstName = name.substring(0, spaceIndex);
        String lastName = name.substring(spaceIndex + 1);

        singSong(firstName);
        singSong(lastName);
    }
}
```

11

Strings answer (cont.)

```
public static void singSong(String name) {
    System.out.println();
    String allButLast = name.substring(1);
    System.out.println(name + ", " + name + ", bo-b" + allButLast);
    System.out.println("Banana-fana fo-f" + allButLast);
    System.out.println("Fee-fi-mo-m" + allButLast);
    System.out.println(name.toUpperCase() + "!");
}
}
```

12

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 `Strings` have the same letters.

13

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.

14

String test methods

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

```
String name = console.nextLine();
if (name.endsWith("Yeats")) {
    System.out.println("Say my glory was I had such friends.");
} else if (name.equalsIgnoreCase("OSCAR WILDE")) {
    System.out.println("A true friend stabs you in the front.");
}
```

15

char

reading: 4.3

Type char

- **char** : A primitive type representing single characters.

- A String is stored internally as an array of char

```
String s = "nachos";
```

index	0	1	2	3	4	5
value	'n'	'a'	'c'	'h'	'o'	's'

- It is legal to have variables, parameters, returns of type char
 - surrounded with apostrophes: 'a' or '4' or '\n' or '\\'

```
char initial = 'J';  
System.out.println(initial); // J  
System.out.println(initial + " Joyce"); // J Joyce
```

17

The charAt method

- The chars in a String can be accessed using the charAt method.
 - accepts an int index parameter and returns the char at that index

```
String food = "cookie";  
char firstLetter = food.charAt(0); // 'c'  
System.out.println(firstLetter + " is for " + food);
```

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

```
String major = "CSE";  
for (int i = 0; i < major.length(); i++) { // output:  
    char c = major.charAt(i); // C  
    System.out.println(c); // S  
} // E
```

18

Comparing char values

- You can compare chars with ==, !=, and other operators:

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

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

19

char VS. int

- Each char is mapped to an integer value internally
 - Called an **ASCII value**

'A' is 65	'B' is 66	' ' is 32
'a' is 97	'b' is 98	'*' is 42

- Mixing char and int causes automatic conversion to int.
'a' + 10 is 107, 'A' + 'A' is 130
- To convert an int into the equivalent char, type-cast it.
(char) ('a' + 2) is 'c'

20

char VS. String

- "h" is a String, but 'h' is a char (they are different)
- A 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'
```

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

```
char c = 'h';  
c = c.toUpperCase();           // ERROR  
s = s.charAt(0).toUpperCase(); // ERROR
```

- What is `s + 1`? What is `c + 1`?
- What is `s + s`? What is `c + c`?

21

printf

reading: 4.3

Formatting text with `printf`

```
System.out.printf("<format string>", <parameters>);
```

- A format string can contain *placeholders* to insert parameters:

- `%d` integer
- `%f` real number
- `%s` string
- `%c` character

(these placeholders are used instead of `+` concatenation)

- Example:

```
int x = 3;
int y = -17;
System.out.printf("x is %d and y is %d!\n", x, y);
// x is 3 and y is -17!
```

Note: `printf` does not drop to the next line unless you write `\n`

23

`printf` precision

- `%.<D>f` real number, rounded to `<D>` digits after decimal

```
double gpa = 3.253764;
System.out.printf("your GPA is %.1f\n", gpa);
```

Output:

```
your GPA is 3.3
```

24

Procedural design

reading: 4.5

Recall: BMI program

Formula for body mass index (BMI):

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
below 18.5	underweight
18.5 - 24.9	normal
25.0 - 29.9	overweight
30.0 and up	obese

- Write a program that produces output like the following:

```
This program reads data for two people and
computes their body mass index (BMI).
```

```
Enter next person's information:
height (in inches)? 70.0
weight (in pounds)? 194.25
```

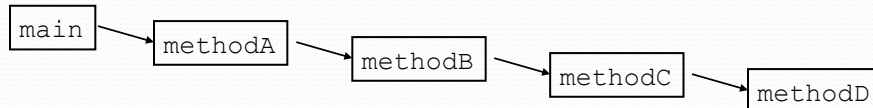
```
Enter next person's information:
height (in inches)? 62.5
weight (in pounds)? 130.5
```

```
Person 1 BMI = 27.868928571428572
overweight
Person 2 BMI = 23.485824
normal
Difference = 4.3831045714285715
```

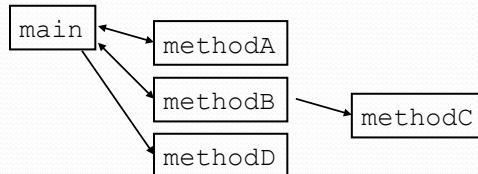
26

"Chaining"

- `main` should be a concise summary of your program.
 - It is bad if each method calls the next without ever returning (we call this *chaining*):



- A better structure has `main` make most of the calls.
 - Methods must return values to `main` to be passed on later.



27

Bad "chain" code

```
public class BMI {
    public static void main(String[] args) {
        System.out.println("This program reads ... (etc.)");
        Scanner console = new Scanner(System.in);
        person(console);
    }

    public static void person(Scanner console) {
        System.out.println("Enter next person's information:");
        System.out.print("height (in inches)? ");
        double height = console.nextDouble();
        getWeight(console, height);
    }

    public static void getWeight(Scanner console, double height) {
        System.out.print("weight (in pounds)? ");
        double weight = console.nextDouble();
        computeBMI(console, height, weight);
    }

    public static void computeBMI(Scanner s, double h, double w) {
        ...
    }
}
```

28

Procedural heuristics

1. Each method should have a clear set of responsibilities.
2. No method should do too large a share of the overall task.
3. Minimize coupling and dependencies between methods.
4. The main method should read as a concise summary of the overall set of tasks performed by the program.
5. Data should be declared/used at the lowest level possible.