

Building Java Programs

Chapter 4
Lecture 4-3: Procedural design; Strings

reading: 3.3; 4.3; 4.5

Copyright 2010 by Pearson Education

Nested if/else question

Formula for body mass index

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

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) and weight status.

```
Enter next person's information:  
height (in inches)? 73.5  
weight (in pounds)? 230  
BMI = 29.93
```

Copyright 2010 by Pearson Education

2

One-person, no methods

```
import java.util.*;

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

        System.out.println("Enter next person's information:");
        System.out.print("height (in inches)? ");
        double height = console.nextDouble();

        System.out.print("weight (in pounds)? ");
        double weight = console.nextDouble();

        double bmi = weight * 703 / height / height;

        System.out.printf("BMI = %.2f\n", bmi);
        if (bmi < 18.5) {
            System.out.println("underweight");
        } else if (bmi < 25) {
            System.out.println("normal");
        } else if (bmi < 30) {
            System.out.println("overweight");
        } else {
            System.out.println("obese");
        }
    }
}
```

Copyright 2010 by Pearson Education

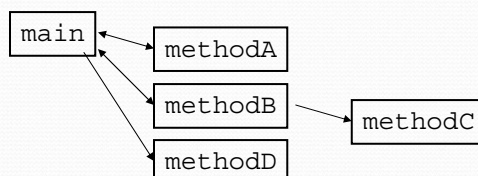
3

"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.



Copyright 2010 by Pearson Education

4

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) {
        ...
    }
}
```

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.

Better solution

```
// This program computes two people's body mass index (BMI) and
// compares them. The code uses Scanner for input, and parameters/returns.

import java.util.*; // so that I can use Scanner

public class BMI {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);
        double bmi1 = person(console);
        double bmi2 = person(console);

        // report overall results
        report(1, bmi1);
        report(2, bmi2);
        System.out.println("Difference = " + Math.abs(bmi1 - bmi2));
    }

    // prints a welcome message explaining the program
    public static void introduction() {
        System.out.println("This program reads ...");
        // ...
    }
    ...
}
```

Better solution, cont'd.

```
// reads information for one person, computes their BMI, and returns it
public static double person(Scanner console) {
    System.out.println("Enter next person's information:");
    System.out.print("height (in inches)? ");
    double height = console.nextDouble();

    System.out.print("weight (in pounds)? ");
    double weight = console.nextDouble();
    System.out.println();

    return bmi(height, weight);
}

// Computes/returns a person's BMI based on their height and weight.
public static double bmi(double height, double weight) {
    return weight * 703 / height / height;
}

// Outputs information about a person's BMI and weight status.
public static void report(int number, double bmi) {
    System.out.printf("BMI = %.2f\n", number, bmi);
    if (bmi < 18.5) {
        System.out.println("underweight");
    } else if (bmi < 25) {
        System.out.println("normal");
    } else if (bmi < 30) {
        System.out.println("overweight");
    } else {
        System.out.println("obese");
    }
}
}
```

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

Indexes

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

```
String name = "R. Kelly";
```

index	0	1	2	3	4	5	6	7
character	R	.		K	e	l	l	y

- 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)

String methods

Method name	Description
<code>indexOf(str)</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>replace(str1, str2)</code>	replaces occurrences of <i>str1</i> with <i>str2</i>
<code>substring(index1, index2)</code> or <code>substring(index1)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (exclusive); if <i>index2</i> is omitted, grabs till 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 gangsta = "Dr. Dre";  
System.out.println(gangsta.length()); // 7
```

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 "Building" ?
(Write code that can extract the first word from any string.)

Modifying strings

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

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

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

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

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 first name? ");  
String name = console.next();  
System.out.println(name + " has " + name.length() +  
    " letters and starts with " + name.substring(0, 1));
```

Output:

```
What is your first name? Chamillionaire  
Chamillionaire has 14 letters and starts with C
```

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

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

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.

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
<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.next();
if (name.startsWith("Prof")) {
    System.out.println("When are your office hours?");
} else if (name.equalsIgnoreCase("BENSON")) {
    System.out.println("Call me Mommy... Whatever!");
}
```

Strings question

- Write a program that outputs a person's "gangsta name."
 - last initial
 - *Diddy*
 - first name (all caps)
 - *-izzle*
 - whether the name starts with a vowel or consonant

Example Output:

```
Type your name, playa: Marge Simpson
Your gangsta name is "S. Diddy MARGE-izzle"
Your name starts with a consonant.
```

