CS101

Java Programming

Fall 2012

Thursdays 10:00-12:00noon

RF Academy

Announcements

- Any Questions?!?!

Nested `if/else` question

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) and weight status.

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

Enter next person's information:
height (in inches)? 71
weight (in pounds)? 220.5
BMI = 30.75
obese

Difference = 0.82
```

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");
        }
    }
}
```

Extending to two people

- If we want our program to process two people, we could copy & paste the code a second time
  - Poor solution as it results in redundant code
- Thus will define methods to do the work for us
  - Eliminates redundancy
  - Simplifies the program via structural decomposition
- But what methods and what decomposition?
  - Many choices are possible
  - Some lead to a good design while others are poor

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.

Good 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 ...");
        // ...
    }
...
```

Good 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("Person %d: 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");
        }
    }
}
```

Announcements

- Read sections 5.1-5.2
- Any Questions?!?!

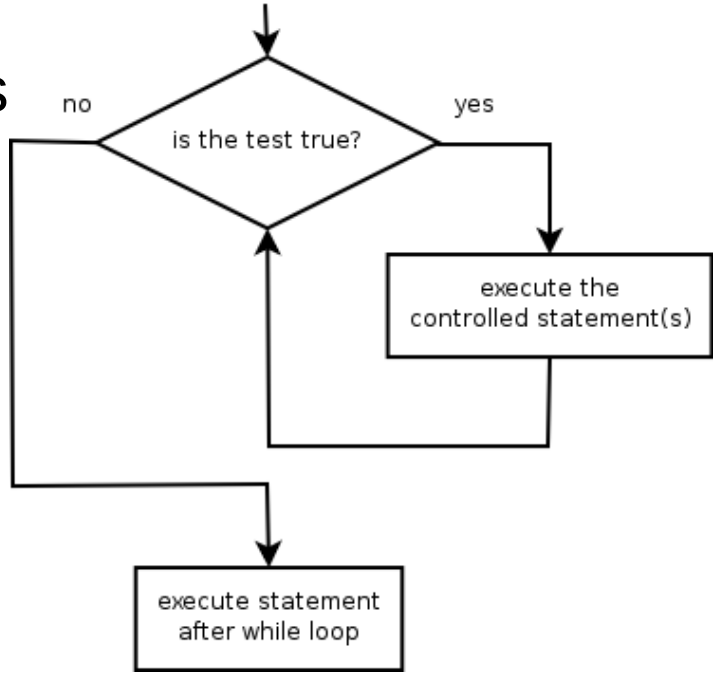
Categories of loops

- **definite loop**: Executes a known number of times.
  - The `for` loops we have seen are definite loops.
  - Examples:
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.
- **indefinite loop**: One where the number of times its body repeats is not known in advance.
  - Examples:
    - Prompt the user until they type a non-negative number.
    - Print random numbers until a prime number is printed.
    - Repeat until the user types "q" to quit.

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

```
while (test) {
    statement(s);
}
```



- Example:

```
int num = 1;                          // initialization
while (num <= 200) {                  // test
    System.out.print(num + " ");
    num = num * 2;                    // update
}
```

  - OUTPUT: `1 2 4 8 16 32 64 128`

Example `while` loop

```
// finds a number's first factor other than 1
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int number = console.nextInt();
int factor = 2;
while (number % factor != 0) {
    factor++;
}
System.out.println("First factor: " + factor);
```

- Example log of execution:

```
Type a number: 91
First factor: 7
```

- `while` is better than `for` here because we don't know how many times we will need to increment to find the factor.

`for` vs. `while` loops

- The `for` loop is just a specialized form of the `while` loop.
  - The following loops are equivalent (more or less):

```
for (int num = 1; num <= 200; num = num * 2) {
      System.out.print(num + " ");
}
```

```
// actually, not a very compelling use of a while loop
// (a for loop is better because the # of reps is definite)
int num = 1;
while (num <= 200) {
    System.out.print(num + " ");
    num = num * 2;
}
```

Mini-exercise

- What `while` loop is essentially equivalent to the following `for` loop?

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

Solution:

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
}
```

`while` loop vs. `if` statement

- The `while` loop and the `if` statement look a lot alike
  - They only differ by a single keyword

```
while (test) {
    statement(s);
}
```

```
if (test) {
    statement(s);
}
```

- But they act very differently
  - You need to be careful to chose the one that is appropriate for a given situation

`while` and `Scanner`

- `while` loops are often used with `Scanner` input.
  - You don't know many times you'll need to re-prompt the user if they enter bad data. (an indefinite loop!)

- Write code that repeatedly prompts until the user types a non-negative number, then computes its square root.
  - Example log of execution:

```
Type a non-negative integer: -5
Invalid number, try again: -1
Invalid number, try again: -235
Invalid number, try again: -87
Invalid number, try again: 121
The square root of 121 is 11.0
```

`while` loop answer

```
System.out.print("Type a non-negative integer: ");
int number = console.nextInt();

while (number < 0) {
    System.out.print("Invalid number, try again: ");
    number = console.nextInt();
}

System.out.println("The square root of " + number +
                   " is " + Math.sqrt(number));
```

- Notice that `number` has to be declared outside the loop.
- Common newbie mistake is to use an if-statement here
  - But what if the next number they input is also negative?

While loop question

- Write a method named `digitSum` that accepts an integer as a parameter and returns the sum of the digits of that number.
  - `digitSum(29107)` returns `2+9+1+0+7` or `19`
  - You may assume that the number is non-negative.
  - Hint: does it matter what order we process the digits?
  - Is there an order that makes this problem easier?
  - How do we extract the last digit from the number?
  - Once extracted, how do we delete the last digit from the number?
  - If we do this repeatedly, when should we stop?

While loop answer

- The following code implements the method:

```
public static int digitSum(int n) {
    int sum = 0;
    while (n > 0) {
        sum += n % 10;   // add last digit to sum
        n = n / 10;      // remove last digit
    }
    return sum;
}
```

- But this code changes the value of n, the parameter we received! Is that okay?
  - Yes, because of call-by-value semantics.

A deceptive problem

- Problem: Write a static method named `printNumbers` that prints each number from 1 to a given maximum, separated by commas; i.e., a comma separated list.

  For example, the method call:

  ```
  printNumbers(5)
  ```

  should print:

  ```
  1, 2, 3, 4, 5
  ```

Flawed solutions

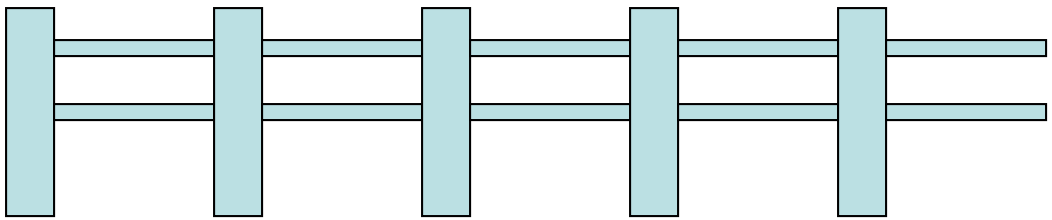
- ```
  public static void printNumbers(int max) {
      for (int i = 1; i <= max; i++) {
          System.out.print(i + ", ");
      }
      System.out.println();  // to end the line of output
  }
  ```
  - Output from `printNumbers(5)`: `1, 2, 3, 4, 5,`
- ```
  public static void printNumbers(int max) {
      for (int i = 1; i <= max; i++) {
          System.out.print(", " + i);
      }
      System.out.println();  // to end the line of output
  }
  ```
  - Output from `printNumbers(5)`: `, 1, 2, 3, 4, 5`

Fence post analogy

- We print *n* numbers but need only *n* - 1 commas.
- This problem is similar to the task of building a fence with lengths of wire separated by posts.
  - often called a *fencepost problem*
  - If we repeatedly place a post and wire, the last post will have an extra dangling wire.

  - A flawed algorithm:

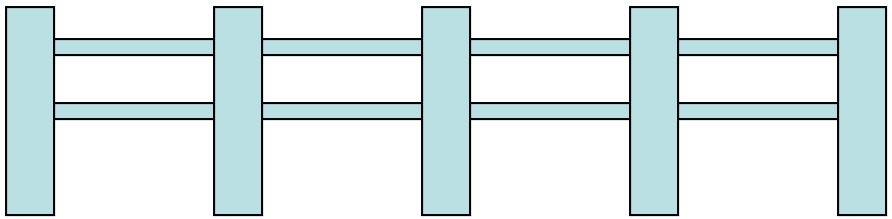
    for (length of fence) {
    place a post.
    attach some wire.
    }



Fencepost loop

- The solution is to add an extra statement outside the loop that places the initial "post."
  - This is sometimes also called a *fencepost loop* or a "loop-and-a-half" solution.
  - The revised algorithm:

place a post.
for (length of fence ***- 1****) {*
    attach some wire.
    place a post.
}



Fencepost method solution

```
public static void printNumbers(int max) {
    System.out.print(1);
    for (int i = 2; i <= max; i++) {
        System.out.print(", " + i);
    }
    System.out.println();     // to end the line
}
```

- Alternate solution: Either first or last "post" can be taken out:

```
public static void printNumbers(int max) {
    for (int i = 1; i <= max - 1; i++) {
        System.out.print(i + ", ");
    }
    System.out.println(max);  // to end the line
}
```

Another fencepost question

- Write a method `printPrimes` that prints all prime numbers up to a given maximum in the following format.
  - Example: `printPrimes(50)` prints
    `[2 3 5 7 11 13 17 19 23 29 31 37 41 43 47]`

- To find primes, write a method `countFactors` which returns the number of factors of an integer.
  - `countFactors(60)` returns 12 because
    1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60 are factors of 60.

- How can we tell if one number is a factor of another?
- Given `countFactors()`, how do we know if a number is prime?

Fencepost answer

```
public class Primes {
    public static void main(String[] args) {
        printPrimes(50);
        printPrimes(1000);
    }

    // Prints all prime numbers up to the given max.
    // Precondition: max is >= 2
    public static void printPrimes(int max) {
        System.out.print("[2");
        for (int i = 3; i <= max; i++) {
            if (countFactors(i) == 2) {   // is i prime?
                System.out.print(" " + i);
            }
        }
        System.out.println("]");
    }
```

Fencepost answer, continued

```
    // Returns how many factors the given number has.
    public static int countFactors(int number) {
        int count = 0;
        for (int i = 1; i <= number; i++) {
            if (number % i == 0) {
                count++;  // i is a factor of number
            }
        }
        return count;
    }
}
```