ICS 111
Introduction to Computer Science I

Nikki Manuel
Leeward Community College

Java Coding Standard, & Input for Interactive Programs

Week 3
Fall 2019

Java Coding Standard

Coding Standard

Why care about neat code?

- Ensures readability (by future you and others)
- Shows your coding ability
- Because you should

Standards differ between organizations

- You must get into the habit of adhering to a standard

Coding Standard

- Naming Conventions
- Comments
- Formatting Conventions

Naming Conventions

We already know:

- Class names (Names of programs)
- Variable names
- Constants
- Upper and lower camel case
- Using _ to separate words

Comments

In-line Comments

- Begin with // and continues to the end of line
- Add a space after // to ensure readability
- Placed above the code they are commenting

Documentation Comments

- Describe a class or method
- Should be 2 - 3 sentences

Comments: In-Line Comments

Placed above the code they are commenting:

Instead of:

```
System.out.print("The area is: " + area);
// Print the result of the calculation.
```

Do this:

```
// Print the result of the calculation.
System.out.print("The area is: " + area);
```

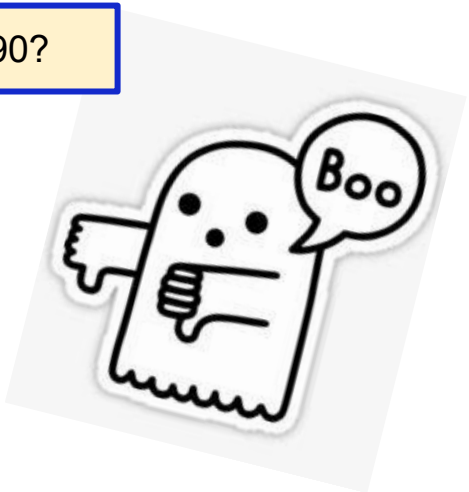
Comments: In-Line Comments - BAD

```
// Apply the discount.
if (totalCost > 1000.0) {

    // Do the multiplication.
    totalCost = totalCost * 0.90;
}
```

How do you know when to apply the discount?

Why is it 0.90?



Comments: In-Line Comments - GOOD

```
// Apply the discount to all invoices over $1000.
if (totalCost > 1000.0) { // :TODO: Use constant?

    // Calculate the cost with the 10% discount,
    // 0.90 is the percentage of how much the customer will pay
    totalCost = totalCost * 0.90;

}
```

You may end up writing more comments than code! But it's necessary!

Comments: Documentation Comments

- Starts with /** ends with */
- Subsequent lines start with *
- Align the * with the first asterisk in the /**
- Used to document classes and methods
- Contains a description and JavaDoc tags
  - JavaDoc tags: @author, @date, @etc.

Comments: Documentation Comments

Asterisks line up.

```
/**
 * Asks the user for a number of seconds (an int).
 * Converts the number of seconds into minutes, hours, and days.
 * Prints the results of the conversions.
 *
 * @author Manuel, Nikki
 */

public class InputText {
```

Program description should be in third person and directly describes what the program does.

Formatting Conventions: Indentation

- Indent nested code with 2 or 3 spaces
- Curly brackets {} indicate another level of code
- Use the CSD button in jGRASP to indent for you
- Be sure to align comments with the same indentation

Formatting Conventions: Program Spacing

Have a space before and after:

- arithmetic operators
- concatenation
- =

```
int number = 100 / 60 + (6 * 9);
System.out.print("The answer is: " + number);
```

Input for Interactive Programming

Why do we want an interactive program?

- Makes programs more dynamic
- Makes code reusable
- Adds usability
  - Tells the user what is going on
  - Explains to the user what to do

Creating an Interactive Program

1. Setup the program skeleton
2. Import the Scanner class
3. Declare your variables
4. Create a Scanner object
5. Create a prompt for the user
6. Get the input
7. Output some feedback

1. Setup the Program Skeleton

```
public class ProgramName {
    public static void main(String[] args) {



    }
}
```

2. Import the Scanner Class

```
import java.util.Scanner;

public class ProgramName {
    public static void main(String[] args) {



    }
}
```

3. Declare Your Variables

```
import java.util.Scanner;

public class ProgramName {
    public static void main(String[] args) {

        String name = “”;

    }
}
```

4. Create a Scanner Object

```
import java.util.Scanner;

public class ProgramName {
    public static void main(String[] args) {

        String name = "";
        Scanner reader = new Scanner(System.in);

    }
}
```

5. Create a Prompt for the User

```
import java.util.Scanner;

public class ProgramName {
    public static void main(String[] args) {

        String name = "";
        Scanner reader = new Scanner(System.in);
        System.out.println("What is your name?");

    }
}
```

6. Retrieve the Input

```
import java.util.Scanner;

public class ProgramName {
    public static void main(String[] args) {

        String name = "";
        Scanner reader = new Scanner(System.in);
        System.out.println("What is your name?");
        name = reader.nextLine();

    }
}
```

7. Output Some Feedback to the User

```
import java.util.Scanner;

public class ProgramName {
    public static void main(String[] args) {

        String name = “”;
        Scanner reader = new Scanner(System.in);
        System.out.println(“What is your name?”);
        name = reader.nextLine();
        System.out.println("Hello, " + name + "!");

    }
}
```

Reading Numbers as Input

Instead of:

```
name = reader.nextLine();
```

To read an **integer** we use:

```
variableName = reader.nextInt();
```

To read a **decimal** we use:

```
variableName = reader.nextDouble();
```

Reading an Integer as Input

```
import java.util.Scanner;

public class PracticeIntegerInput {
    public static void main(String[] args) {

        int age = 0;
        Scanner reader = new Scanner(System.in);
        System.out.println("How old are you?");
        age = reader.nextInt();
        System.out.println("Ok! Next year you'll be: " + ++age);

    }
}
```

Reading a Real Number (Decimal) as Input

```
import java.util.Scanner;

public class PracticeDecimalInput {
    public static void main(String[] args) {

        double wage = 0;
        Scanner reader = new Scanner(System.in);
        System.out.println("How much do you make an hour?");
        wage = reader.nextDouble();
        System.out.println(cost + "? You need a raise!");

    }
}
```