PROGRAMMING STYLE

KEEP IN MIND THAT I’M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMMIE SEE—I’M SURE IT’S FINE.

...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.

IT’S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.

IT’S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I’LL READ A STYLE GUIDE.
Documentation and Style: Outline

- Meaningful Names
- Comments
- Indentation
- Named Constants
- Whitespace
- Compound Statements
Most programs are modified over time to respond to new requirements.
Programs which are easy to read and understand are easy to modify.
Even if it will be used only once, you have to read it in order to debug it.
And when we talk about javadoc, if your comments are meaningful, your code will write its own documentation!!
Meaningful Variable Names

• A variable's name should suggest its use
  • e.g. taxRate

• Boolean variables should suggest a true/false value
  • Choose names such as isPositive or systemsAreOk.
  • Avoid names such as numberSign or systemStatus.
Style: Naming Things

- **Variable names**
  - Begin with lowercase, uppercase each new word
  - `int totalWidgets;

- **Class names**
  - Begin uppercase, then lowercase except for new words
  - `public class InventoryTracker`
  - Name exactly as in assignment description

- **Constants**
  - All upper case, use _ between words
  - `double SPEED_LIGHT = 3.0e8;`
Comments

• The best programs are self-documenting.
  • Clean style
  • Well-chosen names

• Comments are written into a program as needed to explain the program.
  • They are useful to the programmer, but they are ignored by the compiler.
Style: Comments

- **Comments help reader/grader** understand your program
  - Good comments explain **why** something is done
  - Write comments before coding tricky bits
    - Helps you formulate a plan
  - Don't comment the obvious:
    - `i++; // Increment i by one`
Comments

- A comment can begin with `//`.
- Everything after these symbols and to the end of the line is treated as a comment and is ignored by the compiler.

```c
double radius; // in centimeters
```
• A comment can begin with `/*` and end with `*/`
• Everything between these symbols is treated as a comment and is ignored by the compiler.

    /**
     * This program should only
     * be used on alternate Thursdays,
     * except during leap years, when it should
     * only be used on alternate Tuesdays.
     */
Comments

• A javadoc comment, begins with /** and ends with */.

• It can be extracted automatically from Java software.

  /** method change requires the number of coins to be nonnegative */

  We will talk about javadoc later.
When to Use Comments

- Begin each program file with an explanatory comment
  - What the program does
  - The name of the author
  - Contact information for the author
  - Date of the last modification.
- Provide only those comments which the expected reader of the program file will need in order to understand it.
- Each method you write should have a comment about it’s input parameters, it’s return value, and it’s purpose
- Tricky bits of logic should also be commented
**Indentation**

- Indentation should communicate nesting clearly.
- A good choice is four spaces for each level of indentation.
- Indentation should be consistent.
- Indentation should be used for second and subsequent lines of statements which do not fit on a single line.
- Indentation does not change the behavior of the program (in Java).
- Proper indentation helps communicate to the human reader the nested structures of the program.
- Eclipse will help you indent correctly.
  - Eclipse can fix automatically, `ctrl-a` then `ctrl-i`

This is not indented.
This is indented.
This is not indented.
Using Named Constants

• To avoid confusion, always name constants (and variables).
  area = PI * radius * radius;
  is clearer than
  area = 3.14159 * radius * radius;

• Place constants near the beginning of the program.

• Once the value of a constant is set (or changed by an editor), it can be used (or reflected) throughout the program.
  public static final double INTEREST_RATE = 6.65;

• If a literal (such as 6.65) is used instead, every occurrence must be changed, with the risk that another literal with the same value might be changed unintentionally.
Declaring Constants

• Syntax

   public static final Variable_Type Variable_Name = Constant_Value;

• Examples:

   public static final double PI = 3.14159;
   public static final String MOTTO = "The customer is always right.";

• By convention, uppercase letters are used for constants.
public class StarTriangle {
    public static void main(String[] args) {
        int limit = Integer.parseInt(args[0]);
        for (int i=0; i<limit; i++) {
            for (int j = 0; j <= i; j++)
                System.out.print("*");
        System.out.println();
    }
}

- Indent each level of conditionals/loops
  - Indent a fixed number of spaces (3-4)
- Use blank lines to separate logical sections
- Only one statement per line
• **Use spaces between**
  - Statements in for loop
  - Operators in math expressions
  - After the // starting a comment

• But don’t use “too much” whitespace (next slide)
Style: Whitespace

Math. random();

vs.

Math.random();

args [0];

vs.

args[0];

i = i + 1;

vs.

i = i + 1;

• Do NOT use spaces between
  • method class, dot, name, or ()'s
  • array name and []'s
  • statement and ending semicolon
Style: Whitespace

• Use **spaces to align parallel code** if it makes it more readable
  • Often helps to spot mistakes

```java
int numPoints = Integer.parseInt(args[0]);
int startX = Integer.parseInt(args[0]);
int startY = Integer.parseInt(args[2]);
double velX = Integer.parseInt(args[3]);
double velY = Integer.parseInt(args[4]);
```
public class HelloWorld
{
    public static void main(String [] args)
    {
        System.out.println("Hello world!");
    }
}

public class HelloWorld
{
    public static void main(String [] args)
    {
        System.out.println("Hello world!");
    }
}

public class HelloWorld
{
    public static void main(String [] args)
    {
        System.out.println("Hello world!");
    }
}
Summary

• Meaningful Names
• Comments
• Indentation
• Named Constants
• Whitespace
• Compound Statements