CS 351
Design of Large Programs
Coding Standards

Brooke Chenoweth

University of New Mexico

Spring 2024

CS-351 Coding Standards

- All projects and labs must follow the great and hallowed CS-351 coding standards.
- These standards do not necessarily represent the best nor the only good way to write Java code.
- If you have experience programming, then these standards may not be the standards you are used to using.
- However, in this class, these are the standards we will use.

Primary Reasons for Defined Standard

1. A standard makes it easier for the instructors to read your code.
2. A class standard makes it easier for a grader to recognize when a program does not use a consistent standard.
   Often when each student is allowed to define his or her own standard, students switch standards multiple times in a single project. It is tedious for a grader to deduce each person's standard and then check for self-consistency.
3. It is good practice to learn to follow a standard.

Coding Standard: Naming

- Multi-word names are generally written in mixed-case (also known as `camelCase`). Internal words start with a capital letter.
- All variables not declared **final** shall begin with a lowercase letter.
- Static variables that do not ever change value (that is, constants) shall be declared **final** and shall be all uppercase with words separated by underscores.
- All class and member variables (non-local variables) will be given descriptive names.
- Never use the single letter **l** nor the single letter **O** as a name.

Coding Standard: Naming

- Method names must be descriptive (usually verbs) and start with a lowercase letter.
- All class and interface names shall be descriptive (usually nouns) and begin with an uppercase letter.

Coding Standard: Indenting

- Code blocks will be indented to show the block structure with **four spaces** per level.
  (Note: I often use only two spaces in order to fit on the slides. Do as I say, not as I do.)
- Tab characters shall **not** be used for indenting.
- All statements within a block must be indented to the same level.
- You should be able to configure your text editor to use spaces to indent to the correct level when you type a tab. (IntelliJ IDEA uses spaces by default.)

Coding Standard: Curly Braces

- Open brace is last non-space character on a line.
- Closing brace begins a line and is indented to the beginning of the block.
- Exceptions: Empty/single statement class/method body.

```
public class ExampleClass {
  private int myNumber;

  public static void main(String[] args) {
    // statements go here
  }

  private void emptyMethod() {}
  private void singleStatementBody() { myNumber++; }
}
```

Coding Standard: Blocks and { }

Whenever a structure spans more than one line, curly braces *must* be used. For example:

```
/* OK */
if (x == 5) y=y+1;
```

```
/* OK */
if (x == 5) { y=y+1; }
```

```
/* OK */
if (x == 5) {
  y=y+1;
}
```

```
/* Not CS-351 standard */
if (x == 5)
  y=y+1;
```

Comments

- At the top of every .java file, there must be a Javadoc comment block with your name, a description of the what the class/interface is used for and how to use it.
- Before every *public* method, there must be a Javadoc comment block describing what the method does, its parameters, and its return value. (You may comment non-public methods, too, if you wish.)
- There must be a Javadoc comment describing any public variables, enums, nested classes, etc. (Again, good idea to comment non-public stuff too.)

Coding Standard: 80 Character Line Max

- No line shall be more than 80 characters.
- The best way to avoid overly long statements is by not doing too much in a single statement.
- Use temporary variables to break up large expressions.
- If you must keep a long statement, the statement should be broken in a logical place and each line over which the long statement is continued must be indented at least as much as the first line of the statement. (Use more indentation if it improves readability.)

Principle of Least Privilege

In computer science, the *Principle of Least Privilege* requires that in a particular abstraction layer of a computing environment, every module must be able to access only such information and resources that are necessary for its legitimate purpose.

- Variables used in only one method shall be local variables.
- Don't make a non-final instance or class variable public without good reason. (Note: "I couldn't make it work otherwise" is not a good reason!)

Access Modifiers

- All methods, variables, nested classes, etc. contained within a class shall have an explicit access modifier.
- In other words, do not use package-private access in this course.
- Package-private access sometimes has a place in complex projects, but in this course, it is generally a sign that you forgot the access modifier.

Getters and Setters

- In Java, when outside access is needed for a field, the class should provide a getter and/or setter for the field.
- DO NOT blindly auto generate a getter and setter for every field in your class.
- Only create a getter or setter when there is actually a use for that getter or setter.

Package Names

- With the complexity of the projects you will write in this course, your code should be organized into named packages.
- Package names should be all lower case.
- Do not use long chains of sparsely populated directories

  `edu.unm.cs.cs351.spring2024.project1.gui.abc.xyz.Foo.java`
- Don't make a separate package just to hold a single class. (Unless the entire project exists in one class.)
- Initially, you'll probably just have your entire project in one package.

Write Self-Documenting Code

- Self-documenting code uses well-chosen names and has a clear style.
- The program's purpose and workings should be obvious to any programmer who reads the program, even if the program has no comments.
- To the extent that is possible, strive to make your programs self-documenting.

Clean up Debug Output

- When your program is run, it is okay if a few lines of status/debug info is displayed. For example, if the user resizes the window, you might want to have your program print the new window size.
- However, your program should *not* be printing pages of debug spew.
- Either comment out your debug statements or protect them with: `if (DEBUG_GUI)` or some equivalent. Be sure to turn in a version with all the debug flags set to false.

Java Code Conventions

For examples of how to format specific constructs and guidance on issues not covered in these slides, see Java code conventions online.