Code Style and Conventions

CSE219, Computer Science III

Stony Brook University

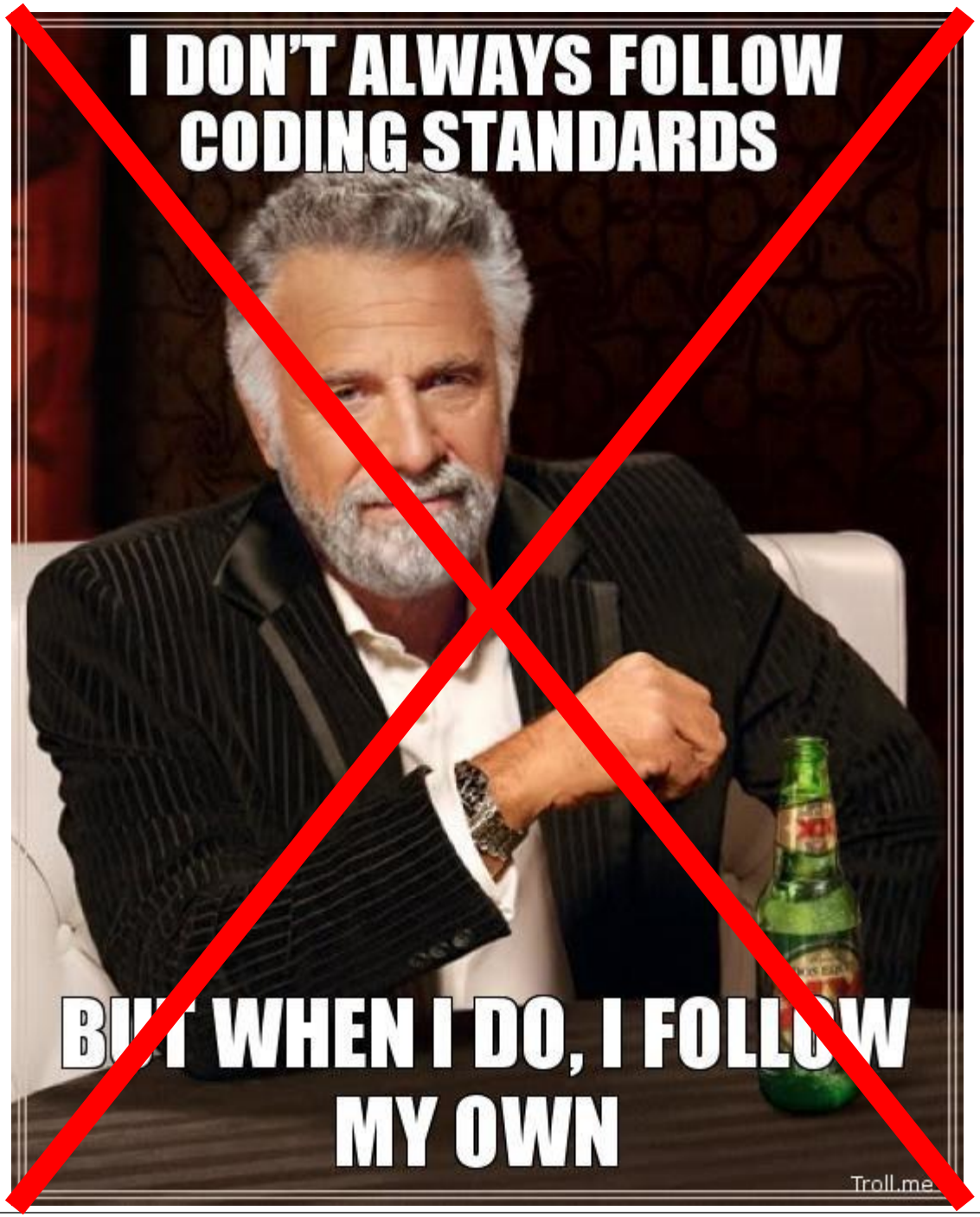
http://www.cs.stonybrook.edu/~cse219

Next three lectures

- Next three lectures are about *"What you always wanted to know about Java (but you never dared to ask)"*
  - Documentation, conventions and code style
    - Includes annotations and reflection
  - Compiling source code into bytecode
    - Includes profiling and optimization

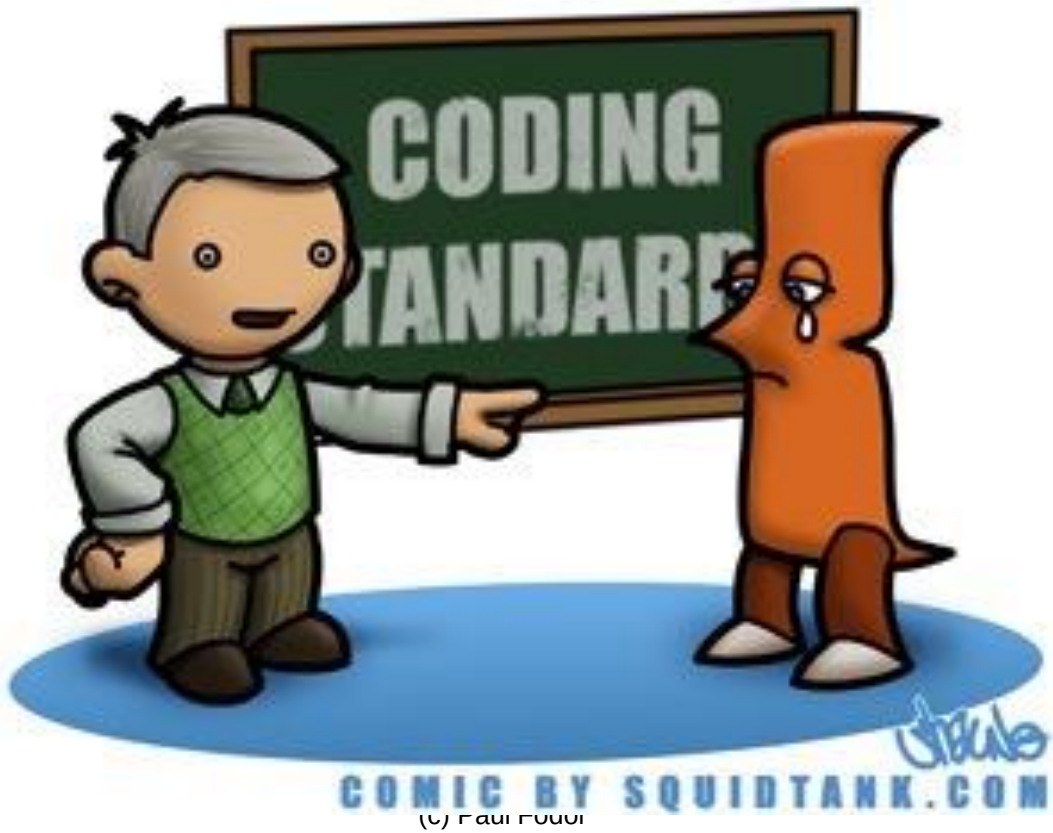
What are code conventions?

- A common style standard
- Encouraged, not enforced
- Think programmer etiquette
- Vary between languages


I DON'T ALWAYS FOLLOW CODING STANDARDS
BUT WHEN I DO, I FOLLOW MY OWN
Troll.me

Why have code conventions?

- Why have code conventions?
  - ~80% of the lifetime cost of software is maintenance
  - rarely maintained by the original author



Java Code Conventions

- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
  - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

What are the benefits of code conventions?

- Improve readability
- Make learning curve less steep
- Ship neatly packaged, clean code

Java Recommendations

- http://www.oracle.com/technetwork/java/codeconvtoc-136057.html
  - Files (file names and extensions)
    - No source files more than 2000 lines of code.
- Order of appearance:
  1. Class/interface documentation comment
  2. Class or interface statement
  3. Class/interface implementation comment (/*...*/), if necessary
  4. Class (static) variables
  5. Instance variables
  6. Constructors
  7. Methods


```
/**
 * The Example class
 * provides ...
 */
public class Example { ...
```

First public, then protected, then package level (no access modifier), and then private.

group these by functionality (those that work together)

More Conventions

- Avoid lines longer than 70 characters
  - not handled well by many terminals and tools.
- When an expression will not fit on a single line, break it according to these general principles:
  - Break after a comma.
  - Break before an operator.
  - Prefer higher-level breaks to lower-level breaks.
  - Align the new line with the beginning of the expression at the same level on the previous line.
  - If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Declaration Conventions

- One declaration per line is recommended since it encourages commenting. In other words,

```
int level; // indentation level
int size; // size of table
```

is preferred over

```
int level, size;
```

- Do not put different types on the same line, Ex:

```
int foo[], bar; //WRONG!
```

Class & Method Conventions

- No space between a method name and the parenthesis "(" starting its parameter list
- Open brace "{" appears at the end of the same line as the declaration statement
- Closing brace "}" starts a line by itself indented to match its corresponding opening statement,
  - when it is a empty method the "}" should appear immediately after the "{"

```
class Sample extends Object {
    int ivar1;
    int ivar2;
    Sample(int i, int j) {
      ivar1 = i;
      ivar2 = j;
    }
    int emptyMethod() {}
    ...
}
```

If, Loop, & Try/Catch conventions

```
if (condition) {
 statements;
}

if (condition) {
 statements;
} else {
 statements;
}

for (initialization; condition; update) {
 statements;
}

try {
 statements;
} catch (ExceptionClass e) {
 statements;
}
```

Additional Conventions

- Avoid using an object to access a class (static) variable or method -> Use a class name instead. For example:

```
classMethod(); //OK in the same class
AClass.classMethod(); //OK
anObject.classMethod(); //AVOID!
    /* It gives the wrong impression
     that the method is dynamic */
```

Javadoc

- Javadoc collects HTML comments from the code into HTML files
  - The comments may contain HTML tags

```
/**
  * Graphics is the abstract base class for all graphics contexts
  * which allow an application to draw onto components realized on
  * various devices or onto off-screen images.
  * A Graphics object encapsulates the state information needed
  * for the various rendering operations that Java supports.  This
  * state information includes:
  * <ul>
  * <li>The Component to draw on …
```

  - The comments contain Javadoc tags

Javadoc Tag Conventions

- Javadoc tags:
  - Order of Tags - include tags in the following order:
    - @author (classes and interfaces only, required)
    - @version (classes and interfaces only, required)
    - @param (methods and constructors only)
    - @return (methods only)
    - @exception (@throws is a synonym added in Javadoc 1.2)
    - @see
    - @since
    - @serial (or @serialField or @serialData)
    - @deprecated

Example

```
/**
 * @param ch the character to be tested

 * @since 1.2

 * @throws IOException  If an input or output
 *         exception occurred

 * @deprecated  As of JDK 1.1, replaced by
 *         setBounds
 * @see #setBounds(int,int,int,int)
 * ...
 */
```