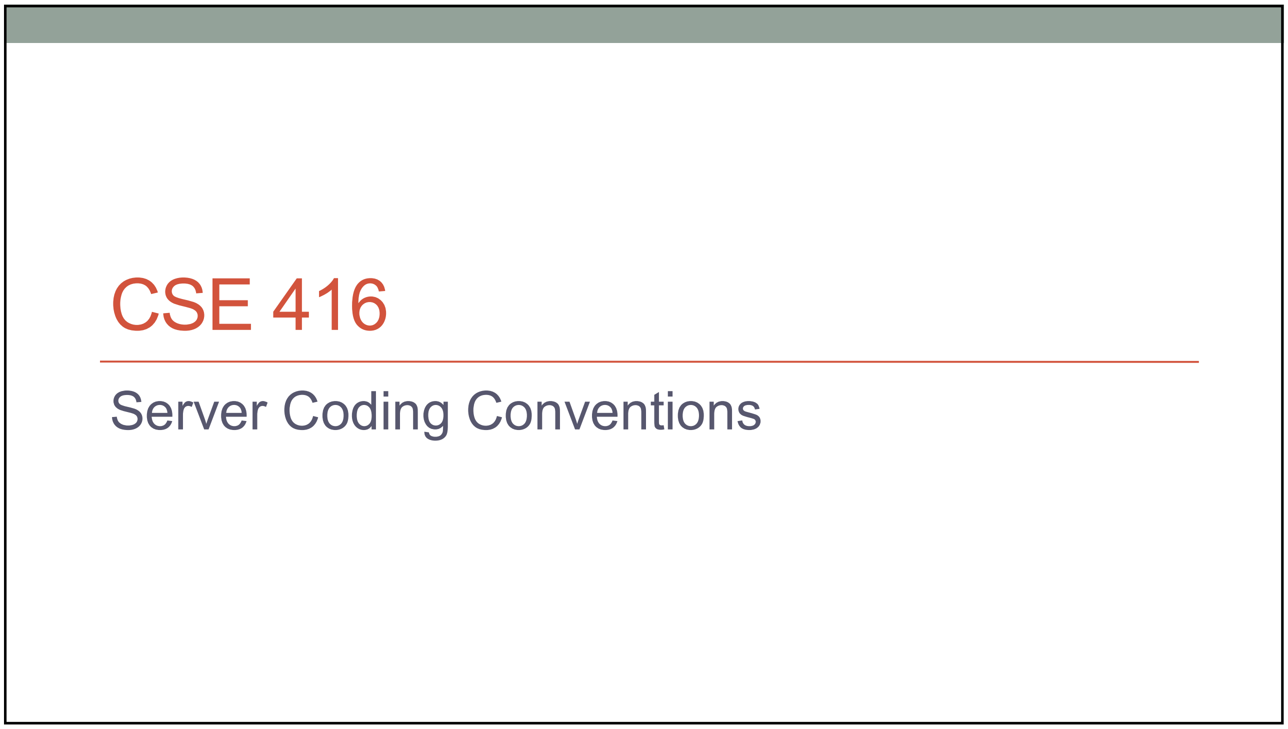


CSE 416

Server Coding Conventions

1

2

Reading

- Python coding style
- https://www.python.org/dev/peps/pep-0008/

2

Code Reviews

- 30-minute Zoom session (either in-class or not in-class)
- Project team picks the starting use case
  - Start by briefly showing the GUI (does not need to be fully functional)
  - Trace the use case logic step-by-step in the code, starting with the HTML
- Plan to respond to requests to review any use case or algorithm / pre-processing code
- Scoring
  - Oral communications (maximum of 10 points)
  - Technical (maximum of 100 points)

Minimal review of pre-processing code and MGGG code

3

Oral Communications Evaluation Criteria

- Voice Projection
- Proper use of vocabulary
- Effectively managing time
- Handling questions

All team members must enable video in Zoom

Communication links / microphone may not be the highest quality, so please speak carefully

4

Technical Quality Evaluation Criteria - Overall

- Code is readable and maintainable
- Code is logical
- Code follows coding conventions
- Completed code shows progress in all aspects of the system
- Team demonstrates an understanding of libraries, frameworks, and language features
- DB is partially populated and in 3rd normal form (supplemental criterion)

Technical Evaluation Criteria

- Absence of logic flaws
- Use of appropriate data structures
- Use of a proper DB persistence layer
- Normalized DB
- Correct structure
- Proper style (e.g., JPA naming)
- Consistency of coding style
- Appropriately named identifiers
- Modular code
- Appropriate use of tools
- Comprehensive RESTful API
- Proper client event handling
- Robust set of SW to date
- Comments only when needed
- Avoiding “magic” numbers
- Import of configuration data
- Code to enable testing

Why Do We Need Coding Conventions?

- Reduce software maintenance
- Improve readability of the SW
  - Easier code walkthroughs and design reviews
  - Short methods

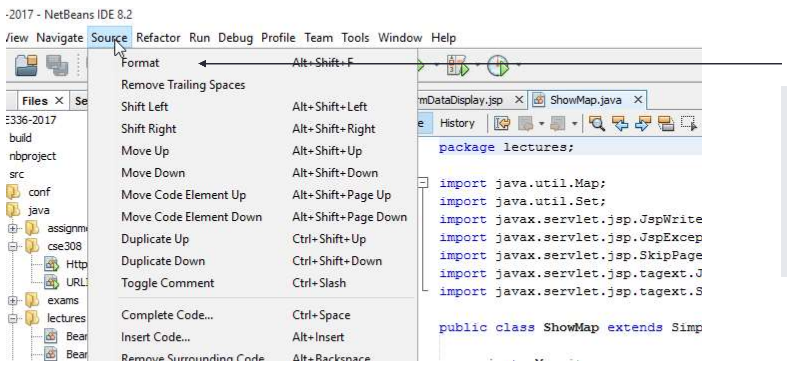
Comments

- Comments will be
  - Implementation - /* … */
  - Javadoc - /** … */

- Implementation comments are for commenting out code or describing particular implementation issues
- Comments should provide only info that is not available in the code (don't document trivial issues)
- Don't use special characters, boxes, etc.
- Block comments should be indented to the same level as the code
- Trailing comments (same line) should be shifted away from code

Appearance

- Indentation
  - 2 spaces is recommended (4 is OK)
  - Use the formatting feature of your IDE (tailor your settings)


-2017 - NetBeans IDE 8.2
/iew Navigate Source Refactor Run Debug Profile Team Tools Window Help
Format Alt+Shift+F
Remove Trailing Spaces
Shift Left Alt+Shift+Left
Shift Right Alt+Shift+Right
Move Up Alt+Shift+Up
Move Down Alt+Shift+Down
Move Code Element Up Alt+Shift+Page Up
Move Code Element Down Alt+Shift+Page Down
Duplicate Up Ctrl+Shift+Up
Duplicate Down Ctrl+Shift+Down
Toggle Comment Ctrl+Slash
Complete Code... Ctrl+Space
Insert Code... Alt+Insert
Remove Surrounding Code Alt+Backspace
Files Se
E336-2017
build
nbproject
src
conf
java
assignm
cse308
Http
URL
exams
lectures
Bear
Bear
mDataDisplay.jsp
ShowMap.java
History

```
package lectures;

import java.util.Map;
import java.util.Set;
import javax.servlet.jsp.JspWrite
import javax.servlet.jsp.JspExcep
import javax.servlet.jsp.SkipPage
import javax.servlet.jsp.tagext.J
import javax.servlet.jsp.tagext.S

public class ShowMap extends Simp
```

Helpful to use the IDE format feature regularly as you are coding – it helps you to see errors

Declarations

- Declarations at the beginning of a block
- One declaration per line
- You can either use a space or a tab between the type and the identifier
  - int level   //authorization level
  - int level   //authorization level
- No space between a method name and the (
- { at the end of the line
- {} when there is a null

Left alignment improves readability

Helpful if identifier names line up on multiple lines

Annotations

- Annotations applying to a class, method or constructor appear immediately after the documentation block
- Each annotation is listed on a line of its own (that is, one annotation per line)

Annotation of method, not instance variable

```
@Override
@Nullable
public String getNameIfPresent() { ... }
```

Blank Lines

- Between methods
- Between local variables in a method and the first statement
- Between logical sections

Do not use unnecessary blank lines – remember, a code module should be readable on a screen without scrolling

Java Naming Conventions

- Packages – lower case (not CC)
- Classes – should be nouns in upper camel case
  - First letter of each internal word is capitalized
  - Use whole words – avoid acronyms and abbreviations
- Methods – should be verbs in lower camel case
- Variables – lower camel case
  - Don't use _ or $
- Constants – all uppercase

Do not use default package

Worthless Documentation

```
/**
 * Represents a command history
 */
public class CommandHistory {
 /**
  * Get the command history for a given user
     */
public static CommandHistory
 getCommandHistory(String user) {
}
}
```

Python Coding Convention Highlights

- PeP 8 – style guide
- Naming
  - Variables, lower case, using a _ for separation
  - Classes – upper camel case
- 4 spaces per indentation level
- # Arguments on first line discouraged, as in
- No mixing of tabs and spaces for indentation
- Lines limited to 79 characters
- Imports at the top of the file on separate lines

```
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```

Selected JavaScript Style Suggestions

- Use const and let, not var
- Do not use the Array constructor – use a literal instead
- Do not use non-numeric properties on an array
- Preference for arrow functions
- Do not use the with keyword

https://google.github.io/styleguide/jsguide.html#language-features