CSE 308
Coding Conventions

Code Reviews

- 30 minutes (either in-class or not in-class)
- Project team picks the starting use case
  - Start by showing the GUI (does not need to be fully functional)
  - Trace the use case logic step-by-step, starting with the HTML
- Plan to respond to requests to review any use case or set-up code (e.g., pre-processing)
- Scoring
  - Oral communications (maximum of 15 points)
  - Technical (maximum of 100 points)
Oral Communications Evaluation Criteria

- Voice Projection
- Eye contact with audience
- Proper use of vocabulary
- Effectively managing time
- Handling questions

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
- Appropriately named identifiers
- Modular code
- Appropriate use of tools
- Robust set of SW to date
- Comments only when needed
- Avoiding “magic” numbers
- Import of configuration data

Emphasis on code maintainability
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
- Source files for classes should start with a comment that includes class name, version (i.e., build), date, and copyright notice
Appearance

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

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

Line Wraps

• Rules:
  • Break after a ,
  • Break before an operator
  • Line wrapping – indent at least 4 spaces (2 is OK)

\[
X = a + b;
\]
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

Declarations

- Some logical ordering
- No C++ array declarations

```
String[] args  // not String args[]
```
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)
- These line breaks do not constitute line-wrapping

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

Statements

- One statement per line
- Return statements should not use ()

```java
Return myObject;
```
If Statements

• Examples

```java
if (condition) {
    statements
}
```

```java
if (condition) {
    statements
} else {
    statements
}
```

Always use {} in your if statements

Iteration Statements

• Examples

```java
while (condition) {
    statements
}
```

```
do {
    statements
} while (condition);
```

```
for (initialization, condition, update) {
    statements
}
```

Always use {} in your if statements
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

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
### Ambler Name Suggestions

- Use full descriptors (e.g., firstName)
- Use domain terminology
- Use abbreviations sparingly
- Avoid long names (e.g., 15 character max)
- Avoid names that are too similar

### Worthless Documentation

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

• One class per file
• Use packages

Do not use default package