**JabberPoint Code Conventions**

Ensuring consistency, readability, and maintainability in the JabberPoint project through standardized coding conventions. By following these conventions, developers can work more efficiently and collaborate seamlessly without confusion.

**Source File Organization**

**Structure Guidelines:**

1. **Package Statements** – Placed at the top of the file, without line wrapping, ensuring that all related files are categorized properly.
2. **Import Statements**

-Organized in a logical sequence for better clarity and consistency:

-No wildcard imports, as they can lead to unexpected dependencies and naming conflicts.

-Group imports in this order to maintain a clean import structure:

* + 1. Static imports, which should be used sparingly for readability.
    2. Java standard library imports to differentiate built-in functionality from external dependencies.
    3. Third-party library imports to isolate external tools and frameworks.
    4. Project-specific imports, ensuring internal references remain distinct.

-Maintain alphabetical ordering within each group for quick reference and easy debugging.

**Formatting**

**Formatting Rules:**

1. **Indentation** – Use 4 spaces per indentation level (no tabs), ensuring code alignment remains uniform across all editors.
2. **Line Length** – Maximum of 100 characters per line, preventing excessive horizontal scrolling and improving readability.
3. **Braces Style** – Follow "Egyptian" style, keeping opening braces on the same line as control structures:

A white background with black and white clouds

AI-generated content may be incorrect.

This style improves code readability and reduces unnecessary vertical space.

1. **White Space Usage**:
   * Add a single space after keywords (**if**, **for**, **while**) to separate them from parentheses for readability.
   * Use spaces around operators (**+**, **-**, **\***, **/**, **=**) to improve expression clarity.
   * No space between a method name and its opening parenthesis, maintaining a compact method declaration style.

**Naming Conventions**

**Naming Rules:**

1. **Classes & Interfaces** – PascalCase (e.g., **SlideViewerFrame**), ensuring that class names are easily distinguishable.
2. **Methods** – camelCase (e.g., **loadFile**), keeping function names descriptive and intuitive.
3. **Variables** – camelCase (e.g., **slideNumber**), following a standard that aligns with method names.
4. **Constants** – ALL\_CAPS\_WITH\_UNDERSCORES (e.g., **DEFAULT\_API\_TO\_USE**), making immutable values immediately recognizable.
5. **Packages** – Lowercase, no underscores (e.g., **jabberpoint**), maintaining a simple and uniform package structure.

**Documentation**

**Documentation Guidelines:**

1. **Javadoc** – Required for all public classes, interfaces, methods, and fields, providing detailed explanations of their purpose and usage.
2. **Comment Format** – Use complete sentences with correct punctuation, ensuring readability and clarity.
3. **Code Comments** – Focus on explaining complex logic while avoiding redundant remarks on obvious operations, making comments more meaningful.

**Error Handling**

**Error Handling Best Practices:**

1. **Use Specific Exceptions** – Catch precise exceptions rather than generic ones to provide more targeted error handling.
2. **Descriptive Error Messages** – Provide informative messages for debugging purposes, ensuring that errors are easily traceable.

**File Structure**

**Structure Guidelines:**

1. Each Java file should contain only one top-level class or interface to maintain modularity and separation of concerns.
2. The filename must match the class or interface name exactly, preventing confusion when locating files.

**Automated Enforcement**

**Enforcement Process:**

1. **Checkstyle Integration** – Style rules are enforced via Checkstyle in the CI/CD pipeline, automatically flagging violations.
2. **Pre-Commit Validation** – Run the following command before committing code to check for errors:

A close up of a word

AI-generated content may be incorrect.

This ensures that only properly formatted code is pushed to the repository.

1. **Merge Restrictions** – Code that fails style checks cannot be merged into the acceptance branch, enforcing strict compliance with conventions.

**Conclusion**

By following these coding conventions, we ensure JabberPoint remains a clean, maintainable, and scalable project. Adhering to these standards promotes best software engineering practices and simplifies collaboration among developers, fostering a more efficient development environment.