**CS 139 Program Style Evaluation Chart**

1 point off each occurrence up to maximum of 2 per line. Maximum5 points off for each section.

|  |  |  |
| --- | --- | --- |
| A. Names | | |
|  | 1. Names should be descriptive and readable |  |
|  | 2. Multi word names should use underscore or capital letter (camel case) |  |
|  | 3. Variable and method names should begin with a lowercase letter. |  |
|  | 4. Class names should begin with a capital letter and use title case. |  |
|  | 5. Constant names should be all caps with an underscore separator. |  |
| B. Declarations | | |
|  | 1. Constants should be named and initialized at the top of the method or class |  |
|  | 2. All variables should be declared directly after the constants (white space sep ok) |  |
|  | 3. There must be a line of white space directly after the variable declarations. |  |
|  | 4. It is NOT permissible to initialize a variable as you are declaring it. |  |
|  | 5. Each declaration should be on its own line. Comment to the right if not clear |  |
| C. Indentation | | |
|  | 1. Subsections of code should be indented a consistent 2-4 spaces |  |
|  | 2. Statements that are too long should be indented 2-4 spaces for 2nd + lines |  |
|  | 3. Blocks of code should be surrounded by curly braces using one one of formats |  |
|  | 4. You should not mix formats within the same file |  |
| D. Literals and constants | | |
|  | 1. Numeric literals must be of the correct type for the context in which they are used. |  |
|  | 2. Constants should be used for meaningful values. |  |
| E. Operators | | |
|  | 1. Binary operators should be separated from their operands by a single space |  |
|  | 2. Unary operators should NOT be separated by a space. |  |
|  | 3. An exception, the dot (.) operator should not have spaces surrounding |  |
|  | 4. In for loop headers, you may compress the operators |  |
| F. Structure | | |
|  | 1. One line of white space should follow the declaration of variables. |  |
|  | 2. Use white space to separate segments of code. |  |
|  | 3. Lines should be kept to a short length (<-80 characters) |  |
|  | 4. Methods should be no longer than 25 lines in length. |  |
|  | 5. If a method returns a value it should have a single return statement. |  |
|  | 6. Break statements are not permitted except for their use in a switch. |  |
|  | 7. You must NOT have any unused variables, constants, or do nothing lines of code (a = a;) |  |
| G. Class structure | |  |
|  | 1. In a class declaration, constructors should precede all other methods. |  |
|  | 2. Methods should be defined in order by visibility modifier, then alphabetically |  |
|  | 3. All class variables should be private unless there is a documented reason for public |  |
|  | 4. All class constants should be public unless there is a documented reason for private |  |
|  | 5. All visibility modifiers should be explicitly stated |  |
| G. Comments | | |
|  | 1. Should use normal English spelling and grammar. Phrases okay. |  |
|  | 2. They must come before the code that they are describing. |  |
|  | 3. Inline comments should only describe major structures or steps in a method |  |
| I. Class Comment | | |
|  | 1. Every class must contain a javadoc comment in the correct format |  |
|  | 2. It must include an @author tag (with both partner’s names if done in pairs) |  |
|  | 3. It must include an @version tag |  |
|  | 4. It must describe the purpose of the class |  |
| H. Acknowledgements | | |
|  | 1. All PAs must have an Acknowledgement section referencing help |  |
|  | 2. This section must come at the top of each file just beneath the class comment. |  |
| J. Method Comment | | |
|  | 1. Every method must have a javadoc style comment in the correct format |  |
|  | 2. It must include a description of the purpose of the class |  |
|  | 3. It must have an @param tag for EACH parameter and must describe each |  |
|  | 4. It must have an @return tag describing the return value (if the method returns) |  |
|  | 5. It must NOT include an @param if no parameters or @return if no return value |  |

**CS 139 Program Style Evaluation Chart**

1 point off each occurrence up to maximum of 2 per line. Maximum5 points off for each section.

|  |  |  |
| --- | --- | --- |
| A. Names | | |
|  | 1. Names should be descriptive and readable |  |
|  | 2. Multi word names should use underscore or capital letter (camel case) |  |
|  | 3. Variable and method names should begin with a lowercase letter. |  |
|  | 4. Class names should begin with a capital letter and use title case. |  |
|  | 5. Constant names should be all caps with an underscore separator. |  |
| B. Declarations | | |
|  | 1. Constants should be named and initialized at the top of the method or class |  |
|  | 2. All variables should be declared directly after the constants (white space sep ok) |  |
|  | 3. There must be a line of white space directly after the variable declarations. |  |
|  | 4. It is NOT permissible to initialize a variable as you are declaring it. |  |
|  | 5. Each declaration should be on its own line. Comment to the right if not clear |  |
| C. Indentation | | |
|  | 1. Subsections of code should be indented a consistent 2-4 spaces |  |
|  | 2. Statements that are too long should be indented 2-4 spaces for 2nd + lines |  |
|  | 3. Blocks of code should be surrounded by curly braces using one one of formats |  |
|  | 4. You should not mix formats within the same file |  |
| D. Literals and constants | | |
|  | 1. Numeric literals must be of the correct type for the context in which they are used. |  |
|  | 2. Constants should be used for meaningful values. |  |
| E. Operators | | |
|  | 1. Binary operators should be separated from their operands by a single space |  |
|  | 2. Unary operators should NOT be separated by a space. |  |
|  | 3. An exception, the dot (.) operator should not have spaces surrounding |  |
|  | 4. In for loop headers, you may compress the operators |  |
| F. Structure | | |
|  | 1. One line of white space should follow the declaration of variables. |  |
|  | 2. Use white space to separate segments of code. |  |
|  | 3. Lines should be kept to a short length (<-80 characters) |  |
|  | 4. Methods should be no longer than 25 lines in length. |  |
|  | 5. If a method returns a value it should have a single return statement. |  |
|  | 6. Break statements are not permitted except for their use in a switch. |  |
|  | 7. You must NOT have any unused variables, constants, or do nothing lines of code (a = a;) |  |
| G. Class structure | |  |
|  | 1. In a class declaration, constructors should precede all other methods. |  |
|  | 2. Methods should be defined in order by visibility modifier, then alphabetically |  |
|  | 3. All class variables should be private unless there is a documented reason for public |  |
|  | 4. All class constants should be public unless there is a documented reason for private |  |
|  | 5. All visibility modifiers should be explicitly stated |  |
| G. Comments | | |
|  | 1. Should use normal English spelling and grammar. Phrases okay. |  |
|  | 2. They must come before the code that they are describing. |  |
|  | 3. Inline comments should only describe major structures or steps in a method |  |
| I. Class Comment | | |
|  | 1. Every class must contain a javadoc comment in the correct format |  |
|  | 2. It must include an @author tag (with both partner’s names if done in pairs) |  |
|  | 3. It must include an @version tag |  |
|  | 4. It must describe the purpose of the class |  |
| H. Acknowledgements | | |
|  | 1. All PAs must have an Acknowledgement section referencing help |  |
|  | 2. This section must come at the top of each file just beneath the class comment. |  |
| J. Method Comment | | |
|  | 1. Every method must have a javadoc style comment in the correct format |  |
|  | 2. It must include a description of the purpose of the class |  |
|  | 3. It must have an @param tag for EACH parameter and must describe each |  |
|  | 4. It must have an @return tag describing the return value (if the method returns) |  |
|  | 5. It must NOT include an @param if no parameters or @return if no return value |  |