|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Level 1** | **Level 2** | **Level 3** | **Level 4** |
| **Thinking: Correctness 4/60** |  |  |  |  |
| Correctness and Specification: How correct is the output of the program. | * The program does not execute due to errors * Incorrect results for most or all input * The program performs tasks that it was not supposed to perform * No evidence of testing | * The program produces correct results but does not display them correctly * Program works and completes most tasks appropriately * Program executes without errors for pre-specified test cases * Program fails to work for special cases | * The program works and produces the correct results, correctly handles most special cases and displays them correctly. * It also meets most of the specifications. * Thorough testing has been completed. | * The program works and meets all the specifications. * The program runs and completes all required tasks. * Program correctly handles all specified test cases and additional special cases. * Program contains error checking code. |
| **Thinking: Quality of Code - 16/60** |  |  |  |  |
| Code Organization and Readability | * No organization of code or no coding conventions seem to exist. * Misleading indentation; too much or too little white space * Very difficult to read | * Disorganized tabs and spaces * Only readable by someone who knows what it is supposed to be doing | * Code is formatted using common tabbing. Multiple coding conventions (inconsistencies) are being used. * Some inadequate or wasted space. | * Code is correctly tabbed, and all elements are neatly organized. * Coding conventions are consistently followed throughout the program. * Consistent indentation; good use of white space |
| Identifier Names | * Meaningless or misleading names. * Variables/methods are used inappropriately. | * Some poor choices of naming. * Most identifiers explained where appropriate. | * Unambiguous naming. | * Meaningful identifier names [some single letter names are OK, such as i, for indices]. * Explanations of identifiers where appropriate. |
| Reusability | * The code is not organized for reusability. No reuse potential. * Too much redundancy in code. * Long code segments, improper usage of functions, functions with side effects. | * Some parts of the code could be reused in other programs (individual methods). * Occasional code redundancy. | * Some of the code could be reused in other programs (possibly a class). | * The code was developed in such a way that either the whole or some of the supporting classes could be reused. * Code structure has perfectly followed guidelines. Short code segments, proper use of functions. No code redundancy. |
| Efficiency | * Program uses a difficult and inefficient solution. Programmer has not considered alternate solutions. * The code is huge and appears to be patched together. | * Program uses a logical solution that is easy to follow but it is not the most efficient. Programmer has considered alternate solutions. | * Program uses an efficient and easy to follow solution (i.e., no confusing tricks), without sacrificing readability and understanding. * Programmer has considered alternate solution and has chosen the most efficient. | * Program uses solution that is easy to understand and maintain. * Programmer has analyzed many alternate solutions and has chosen the most efficient. Programmer has included the reasons for the solution chosen. |
| **Application: Documentations//Graphical User Interface Front End 16/60** |  |  |  |  |
| Logical Blocks | * Few or no logical blocks documented | * Half to average number of blocks documented | * Most logical blocks documented | * Documentation for each function and loop and logical block |
| Commenting | * No name, student ID, date, or assignment title included * No comments included or other documentation provided * Comments are too general | * Header is appropriate * Comments are included but poorly described. * Most methods are commented. | * The purpose of all variables is clearly explained. * The purpose of each method is described. * Comments provide some assistance with understanding the code. | * Includes name, student ID, date, and assignment title. * Clear and complete documentation. Contains required heading with program description. * Appropriate use of comments. * The purpose and constraints of every variable and method are described. Comments for code segments make the code easy to follow. * Specific purpose is noted for each control structure, input requirements, and output results. |
| **Communication: Demonstration/Graphical User Interface Front End 24/60** |  |  |  |  |
| Demonstration | * Program will not run or crashes unexpectedly. Submitted code seems different from that demonstrated. Programmer seemed confused by the program they were demonstrating. | * Program runs but does not match design | * Program runs reasonably well. * Submitted code matches’ demo. | * Demo went well; major features shown and explained. * Demo shows that the author clearly understood their work. |
| **Communication:** Testing Document | * Very little tests shown. * Hardly any attempt at testing any part of the software | * Some testing but does not encompass entire software | * Testing covers most of the software | * All tests cover all use cases |
| **Communication:** “How-To” Document | * Very basic description of functionality of the program with hardly any detail | * Some but few steps shown demonstrating functionality | * Most steps are explained very well | * Thorough steps explained with all use cases covered |
| **Communication:** Video Demonstration | * Very short and/or basic demo | * Some or very few features of program shown. | * Most features of shown. | * Very thorough demo showing all major and minor features. |