```
#+STARTUP: showeverything
#+TITLE: Rubric: pMenu
#+SUBTITLE: CIS 205 Computer Science III
#+DATE: Fall 2023
#+AUTHOR: Brian C. Ladd
#+EMAIL: laddbc@potsdam.edu

<student>
DUE:       2023-09-01T09:00:00-05:00
COMMITTED: <commit>
<late>

*REASONS TO STOP GRADING*
- Failure to compile.
- No =README=
  - No ID block in =README=
  - No instructions on compiling and running
- Source code file with no ID block
*REASONS TO STOP GRADING*

=====================================================================
- Documentation /30
=====================================================================
- Good type, function, and variable names
  - Descriptive names
  - Name length directly proportional to size of scope:
    larger scope -_ longer name

  - Appropriate level of in-line and header comments
    - Comments *do not* replicate information better communicated by reading the
 code.
    - Header comments on files, classes, functions
      - Document /intention/: what is the purpose of this *thing*
      - Functions: what does it do; what are the parameters and their permitted
values?
      - Class/Type: how and where is one constructed; why is one made; how long
should it live?
    - In-line comments when necessary to explain/support the code
    - Comments use *complete sentences*, *standard spelling*, and *proper Englis
h grammar*
- Style
  - Indentation and Whitespace
    - Use only SPACE characters for indent, not TAB
    - Consistent indentation that improves readability
      - Indent between curly braces to show nesting
    - Good use of blank lines to break code into sections
  - Curly-brace placement
    - Opening curly-brace should NEVER start a line
      - It ends the line with the if/while/for/else on it, doesn't start the nex
t line
    - Closing curly-brace aligns with the BEGINNING of the if/while/for/else lin
e
    - else if is on one line
      - Starts with } (if necessary)
      - else if (<boolean>) follows
      - ends with { (if necessary)
- README
  - One of the acceptable formats: =.txt=, =.md=, =.org=
  - Problem Statement: Make the submission self-contained for future readers
  - Compiling and Running Instructions
```

  - Testing Criteria
    - Indicates what testing was done
      - *How* was the code tested?
        - What "happy path" tests do you do to show things work?
        - What error conditions does the code detect and how did you test that?
      - *When* were the tests developed? (/Before/ or /after/ code was written.)
      - *What* artifacts (files, input sequences, etc.) were used to test?
        - Include artifacts in the repository.
        - Explain how to test the project.
    - *IF* the solution is incomplete
      - What incorrect behavior do you see?
      - What do you think is causing it?

==========
- Design /20
==========
  - Good use of Java standard library classes
    - Read as, "Uses the right implementation of =List= interface"
  - Does not leak the MenuEntry class
  - Menu has the correct public interface: ctor, match, help
  - MenuItem properly hides its three fields
==========
- Implementation /20
==========
  - Includes necessary files in =git=
    - And *excludes* all the files that are not necessary (through =.gitignore=)
  - Menu checks against the values in the entries for a match
  - match does not return until there is a match
  - match shows an appropriate error message on a missed match
  - help output is formatted correctly
==========
- Testing /20
==========
  - What I should type at each prompt is provided in docs
  - Expected response is documented and is the output of the test
  - Tests at least zero, one, and two entry menus.
==========
- Aesthetics 5/10
==========
  - Starts at 5.
  - Pros:
  - Cons: