Linter Project Wiki

1. ConfusingMethodNames (called MethodStyleCheck in the project)
   The MethodStyleCheck identifies whether or not there are method names in a class that differ only by capitalization and adds to a report to let the user know if their code requires more revision.

   Methods:
   - populateMethodName
     - Inputs:
       - None
     - Outputs:
       - None
     - Algorithm
       - Loop through the list of MyMethodNodes and add all method names to a list
       - Ignore the method name if it is the constructor
     - Data Needed:
       - MyClassNode with a list<MyMethodNode>
   - confusingMethodNames
     - Inputs:
       - None
     - Outputs:
       - None
     - Algorithm:
       - Loop through method names
       - Temp variable holds lowercase version of the method name
       - Compare temp to the keys of the hashmap of lowercase method names to original method names.
       - If a name is unique, add it to the HashMap, otherwise add it to a list to be dealt with later.
       - If there were no errors, return a success message, otherwise loop through all problematic names and give details in the user report.
     - Data Needed:
       - HashMap<String,String> for storing unique method names
       - ArrayList<String> for storing improper method names
   - IsInitMethod
     - Input:
       - MyMethodNode
     - Output:
       - Boolean
     - Algorithm
       - Return if “<init>” is equal to the method name
     - Data Needed:

            - MyMethodNode.name()
    - performCheck
        - Input:
            - MyClassNode
        - Output:
            - String
        - Algorithm:
            - Call other functionality in the class to compute the check
        - Data Needed:
            - N/A
2. InformationHiding

Methods:

- performCheck
    - Input:
        - MyClassNode
    - Output:
        - String
    - Algorithm:
        - Call other functionality in the class to compute the check
    - Data Needed:
        - N/A
- checkModifiers
    - Input:
        - None
    - Output:
        - String
    - Algorithm:
        - Create a StringBuilder
        - Parse through all fields in the class. Use Opcodes to determine access modifier
            - Increment appropriate fields
        - Parse through all methods in the class. Use Opcodes to determine access modifier
            - Increment appropriate fields
        - Call another method to compute a result
- checkAccessFlag
    - Input:
        - Int: code
        - Int: inputType
    - Output:
        - None
    - Algorithm

            - Use a switch case to determine what type of input is being dealt with as well as determine what the access modifier is.
            - Increment appropriate fields
        - Data Needed:
            - Opcodes for access modifiers
- finalCheck
    - Input:
        - int: number of fields
        - Int: number of methods
        - StringBuilder: string builder
    - Output:
        - None
    - Algorithm
        - Perform calculations to determine a relative percentage of fields/methods that are public.
        - Based on calculation,, suggest, or don't, revisions to reduce public methods
        - Add info to String to be added to a report for the user.
- resetCounter
    - Input:
        - None
    - Output:
        - None
    - Algorithm:
        - Reset all incrementors in the class to 0
    - Data Needed:
        - Incrementors in the class.

3. Strategy Pattern
    - performCheck
        - Input:
            - MyClassNode: classNode
        - Output:
            - String
        - Algorithm
            - Check if the given node has implemented an interface
            - If a node has an interface, do checks to see if there are potential strategies
            - Determine if there is a high likelihood that a strategy exists
            - Create report for user
        - Data Needed:
            - N/A