Chapter 1

Code Inspection Checklist

In this appendix, the list of all points checked in our Code Inspection's analysis is provided split into the same sections of the **??**.

- **Naming Conventions**
  - All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.
  - If one-character variables are used, they are used only for temporary throwaway variables, such as those used in for loops.
  - Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: class Raster; class ImageSprite;
  - Interface names should be capitalized like classes.
  - Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().
  - Class variables, also called attributes, are mixed case, but might begin with an underscore ('_') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: _windowHeight, timeSeriesData.
  - Constants are declared using all uppercase with words separated by an underscore. Examples: MIN_WIDTH; MAX_HEIGHT;

- **Indention**
  - Three or four spaces are used for indentation and done so consistently.
  - No tabs are used to indent.
- **Braces**
  - Consistent bracing style is used, either the preferred *Allman* style (first brace goes underneath the opening block) or the *Kernighan and Ritchie* style (first brace is on the same line of the instruction that opens the new block).
  - All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. Example:

    ```
    if ( condition )
        doThis();
    ```

    The correct version is:

    ```
    if ( condition ) {
        doThis();
    }
    ```

- **File Organization**
  - Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).
  - Where practical, line length does not exceed 80 characters.
  - When line length must exceed 80 characters, it does NOT exceed 120 characters.
- **Wrapping Lines**
  - Line break occurs after a comma or an operator.
  - Higher-level breaks are used.
  - A new statement is aligned with the beginning of the expression at the same level as the previous line.
- **Comments**
  - Comments are used to adequately explain what the class, interface,

methods, and blocks of code are doing.
  - Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.
- **Java Source Files**
  - Each Java source file contains a single public class or interface.
  - The public class is the first class or interface in the file.
  - Check that the external program interfaces are implemented consistently with what is described in the javadoc.
  - Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).
- **Package and Import Statements**
  - If any package statements are needed, they should be the first non-comment statements. Import statements follow.
- **Class and Interface Declarations**
  - The class or interface declarations shall be in the following order:
    - class/interface documentation comment
    - class or interface statement
    - class/interface implementation comment, if necessary
    - class (static) variables
      - first public class variables
      - next protected class variables
      - next package level (no access modifier)
      - last private class variables
    - instance variables
      - first public instance variables
      - next protected instance variables
      - next package level (no access modifier)
      - last private instance variables
    - constructors
    - methods
  - Methods are grouped by functionality rather than by scope or acces-

sibility.
  - Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.
- **Initialization and Declarations**
  - Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).
  - Check that variables are declared in the proper scope.
  - Check that constructors are called when a new object is desired.
  - Check that all object references are initialized before use.
  - Variables are initialized where they are declared, unless dependent upon a computation.
  - Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces {and }). The exception is a variable can be declared in a 'for'loop.
- **Method Calls**
  - Check that parameters are presented in the correct order.
  - Check that the correct method is being called, or should it be a different method with a similar name.
  - Check that method returned values are used properly.
- **Arrays**
  - Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).
  - Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
  - Check that constructors are called when a new array item is desired.
- **Object Comparisons**
  - Check that all objects (including Strings) are compared with *equals* and not with '=='.
- **Output Format**
  - Check that displayed output is free of spelling and grammatical errors.

  - Check that error messages are comprehensive and provide guidance as to how to correct the problem.
  - Check that the output is formatted correctly in terms of line stepping and spacing.
- **Computation, Comparisons and Assignments**
  - Check that the implementation avoids *brutish programming*.
  - Check order of computation/evaluation, operator precedence and parenthesizing.
  - Check the liberal use of parenthesis is used to avoid operator precedence problems.
  - Check that all denominators of a division are prevented from being zero.
  - Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.
  - Check that the comparison and Boolean operators are correct.
  - Check throw-catch expressions, and check that the error condition is actually legitimate.
  - Check that the code is free of any implicit type conversions.
- **Exceptions**
  - Check that the relevant exceptions are caught.
  - Check that the appropriate action are taken for each catch block.
- **Flow of Control**
  - In a switch statement, check that all cases are addressed by break or return.
  - Check that all switch statements have a default branch.
  - Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.
- **Files**
  - Check that all files are properly declared and opened.
  - Check that all files are closed properly, even in the case of an error
  - Check that EOF conditions are detected and handled correctly
  - Check that all file exceptions are caught and dealt with accordingly