Java Code Inspection Checklist

1. Variable and Constant Declaration Defects (VC)

1. Are descriptive variable and constant names used in accord with naming conventions?

   Sí. Todas las variables descriptivas están en minúsculas (p.e en la clase Entry línea 16) y si son una palabra compuesta la segunda está con mayúscula (p.e en la clase Entry línea 23). Las constantes también están acorde a las convenciones, ya que todas las letras del nombre están en mayúscula (p.e. en la clase Parser línea 18).

2. Are there variables with confusingly similar names?
   No, porque aunque en la clase Entry y en la clase Parser hay variables cuyo objetivo del nombre de la variable es el mismo, no se repiten los nombres. Por ejemplo, String telephone (clase Entry) y final String TFN (clase Parser).

3. Is every variable properly initialized?
   Los strings están inicializados a cadena vacía (línea 27 de la clase Entry), los int a 0 (línea 34 de la clase Entry) y los nodos a null (línea 27 de la clase Agenda). En conclusión, están todos bien inicializados.

4. Could any non-local variables be made local?
   No, ya que todas las variables no locales, son usadas por varios o todos los métodos de la clase.

5. Are there literal constants that should be named constants?
   No, ya que las constantes de la clase Parser tienen sentido porque siempre que se quiera crear una entrada en la agenda se crea SIEMPRE una línea que empieza con lo que se ha inicializado la constante. (P.e. la constante NOM de la clase Parser está inicializada a NOMBRE=).

6. Are there macros that should be constants?
   No procede.

7. Are there variables that should be constants?
   No. Porque las variables que están definidas como no constantes, no siempre tienen el mismo valor (cada persona de la agenda tendrá o no un nombre distinto a los demás). Por ejemplo en la clase Entry de las líneas 16-23. La misma explicación para la clase Agenda, que sus variables tampoco tienen el mismo valor siempre.

2. Function Definition Defects (FD)

8. Are descriptive function names used in accord with naming conventions?

   Sí, todos los nombres de las funciones en todas las clases son correctos.

9. Is every function parameter value checked before being used?
   En los setters de la clase Entry no se comprueba ningún valor de los que se le pasa por parámetro, aunque en estos casos no es necesario chequearlos.
   En el método "addEntry" de la clase Agenda (línea 39) no se comprueba el estado de la variable de tipo Entry que se le pasa por parámetro.

10. For every function: Does it return the correct value at every function return point?
    En todas las funciones de todas las clases se devuelve correctamente el valor adecuado (por ejemplo, en la función "addEntry" de la clase Agenda (línea 39) se devuelve true si se añade correctamente la entrada, y false si no es posible).

3. Class Definition Defects (CD)

11. Does each class have an appropriate constructor and destructor?

    Todas las clases tienen constructor, aunque ninguna tiene explícito el destructor.

12. For each member of every class: Could access to the member be further restricted?
    No, porque los métodos que deberían ser privados porque solo se llaman dentro de esa clase ya lo están. Por ejemplo en la clase Parser los métodos createEntry() y createLine(). Es cierto que en la clase Agenda, el método podríamos restringirlo a privado si pusiéramos una llamada al mismo en el constructor, pero como no la hay, la restricción no está mal (es solo una sugerencia).

13. Do any derived classes have common members that should be in the base class?
    No procede.

14. Can the class inheritance hierarchy be simplified?
    No procede.

4. Computation/Numeric Defects (CN)

15. Is overflow or underflow possible during a computation?

    No, no es posible que se origine underflow u overflow en una computación del programa.

16. For each expressions with more than one operator: Are the assumptions about order of evalu- ation and precedence correct?
    No hay expresiones con más de un operador.

17. Are parentheses used to avoid ambiguity?
    No son necesarios los paréntesis para evitar ambigüedad.

5. Comparison/Relational Defects (CR)

18. Are the comparison operators correct?

    Sí, están todos correctos. Solo se usan el == y el != en la clase Agenda (p.e. en la línea 43) y están escritos correctamente.

19. Is each boolean expression correct?
    En la clase Agenda sí (línea 41, 126).

20. Are there improper and unnoticed side-effects of a comparison?
    Haciendo un repaso general de la clase Agenda que es la que usa comparaciones, no hemos encontrado ningún efecto secundario de estas que afecte al código.

6. Control Flow Defects (CF)

21. For each loop: Is the best choice of looping constructs used?

    Únicamente existen 3 whiles y 1 do-while en la clase Agenda (líneas 45, 74, 133 y 164), y son la opción más adecuada puesto que no se está recorriendo ninguna estructura de datos compleja (tipo arrays).

22. Will all loops terminate?
    El bucle while de la función “saveAgenda” en la clase Agenda (línea 133) no terminará nunca, pues la condición de seguimiento es que cur no sea null y dentro del bucle no se modifica o itera en ningún momento dicha variable cur, por lo que si se cumple en un primer momento dicha condición, no acabará nunca la ejecución del bucle.

23. When there are multiple exits from a loop, is each exit necessary and handled properly?
    En los bucles donde son posibles varias salidas (whiles de la clase Agenda en las líneas 45 y 74), estas son necesarias y tratadas correctamente.

24. Does each switch statement have a default case?

    No hay sentencias switch.

25. Are missing switch case break statements correct and marked with a comment?
    No hay sentencias switch.

26. Is the nesting of loops and branches too deep, and is it correct?
    No hay bucles anidados.

27. Can any nested if statements be converted into a switch statement?
    No hay bucles anidados.

28. Are null bodied control structures correct and marked with braces or comments?
    No existen estructuras con cuerpo nulo.

29. Does every function terminate?
    A excepción de la función “saveAgenda()” que contiene el bucle que puede que no termine nunca (clase Agenda, línea 133), todas las funciones terminarán.

30. Are goto statements avoided?
    No existen sentencias “goto”, por lo que sí se evitan.

7. Input-Output Defects (IO)

31. Have all files been opened before use?

    Sí. En la única clase que se usan ficheros es en Agenda en el método loadAgenda() (línea 153) y sí se abre el fichero de forma correcta con las líneas 154-155.

32. Are the attributes of the open statement consistent with the use of the file?
    Sí, porque se ha abierto como un FileReader y solo se lee de fichero, no se modifica.

33. Have all files been closed after use?
    No. El de la línea 172 de la clase Agenda sí, pero fichero que se abre en el método saveAgenda() no se cierra nunca.

34. Is buffered data flushed?
    No.

35. Are there spelling or grammatical errors in any text printed or displayed?
    No, todos están correctos. En la clase Parser y la de Agenda. Están puestos los espacios correctos entre palabras y también puntos para separar.

36. Are error conditions checked?
    No, ya que se lanzan las excepciones en los métodos de Agenda, pero no se controlan con el try y el catch.

8. Module Interface Defects (MI)

37. Are the number, order, types, and values of parameters in every function call in agreement with the called function’s declaration?
    Sí, por ejemplo la línea 55 de la clase Agenda.java.

38. Do the values in units agree (e.g., inches versus yards)?
    No se utilizan unidades.

9. Comment Defects (CM)

39. Does every function, class, and file have an appropriate header comment?
    Sí, por ejemplo la función addEntry de Agenda.java.

40. Does every variable or constant declaration have a comment?
    No porque se dan por entendidos con el nombre de la variable.

41. Is the underlying behavior of each function and class expressed in plain language?
    Sí, por ejemplo en la función addEntry de Agenda.java explica perfectamente lo que hace aunque no tengas nociones de programación.

42. Is the header comment for each function and class consistent with the behavior of the function or class?
    Sí, es consistente con el funcionamiento de las funciones y clases.

43. Do the comments and code agree?
    Sí, los comentarios son coherentes con el Código.

44. Do the comments help in understanding the code?
    Sí, explican de manera clara el funcionamiento del Código.

45. Are there enough comments in the code
    Sí, existen comentarios que explican las funciones más difíciles de entender.

46. Are there too many comments in the code?
    No, en las funciones que se sobreentienden no existen comentarios.

10. Packaging Defects (LP)

47. For each file: Does it contain only one class?
    No, son 4 clases.

48. For each function: Is it no more than about 60 lines long?
    Sí, la más larga tiene 46 líneas.

49. For each class: Is no more than 2000 lines long (Sun Coding Standard) ?
    Sí, no llegan a 200 líneas.

11. Modularity Defects (MO)

50. Is there a low level of coupling between packages (classes)?
    No, se necesitan unas a otras, por ejemplo la clase Entry.java proporciona todas las funciones necesarias para las clases Parser y Agenda.

51. Is there a high level of cohesion within each package?
    Sí, las clases Agenda y Parser dependen de Entry.

52. Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?
    No.

53. Are framework classes used where and when appropriate?
    No.

slightly adapted from the C++ Inpsection Checklist of
Christopher Fox,
http://www.cs.jmu.edu/users/foxcj/cs555/StdDoc/CppChk.htm.