Java Code Inspection Checklist

1. Variable and Constant Declaration Defects (VC)

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

No, en el caso de neigh el significado no es claro, en el caso de significar neighbour no está especificado.

2. Are there variables with confusingly similar names?

No.

3. Is every variable properly initialized?

Sí, están bien inicializadas (líneas 42,43 y 57-61).

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.

6. Are there macros that should be constants?

No, no hay macros en el código.

7. Are there variables that should be constants?

nVertices pues es una variable que nunca cambia de valor.

2. Function Definition Defects (FD)

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

El nombre del método “initializeDataStructures” (línea 50) es bastante largo y redundante. Cambiándolo por “inic” o “initialize” mejoraríamos la comprensión del programa.

El nombre de la función “nextCur” (l.73) quizás es un poco confuso de entender.
El nombre de la función “computeShortestPath” (l.107) es bastante largo y se podría resumir como “shortestPath”.

El método “hasErrorHappened” (l.179) se podría reescribir como “getError”, pues básicamente devuelve el contenido de la variable “error”.

Los demás métodos los consideramos adecuados.
Además, todos los nombres empiezan en minúscula y si tiene varias palabras las siguientes se ponen en mayúscula, siguiendo la convención de Java.

9. Is every function parameter value checked before being used?

En la función “computeShortestPath” (l.107), no se evalúan ninguno de sus dos parámetros de tipo Integer, “ini” y “end”.
Igualmente ocurre lo mismo con el método “getPath” (l.153).

10. For every function: Does it return the correct value at every function return point?

En el método “getPath” (l.153) devuelve “null” dentro de una condición if, algo que no es muy conveniente.

3. Class Definition Defects (CD)

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

La clase Dijkstra tiene un constructor (línea 41) pero no dispone de un destructor explícito.

12. For each member of every class: Could access to the member be further restricted?

Existen 3 métodos públicos (computeShortestPath, getPath y hasErrorHappened) pero consideramos que su acceso no debería ser más restringido.

13. Do any derived classes have common members that should be in the base class?

No hay clases derivadas.

14. Can the class inheritance hierarchy be simplified?

En este código no hay jerarquía de clases.

4. Computation/Numeric Defects (CN)

15. Is overflow or underflow possible during a computation?

No es posible, pues ninguna de las variables se ven aumentadas o disminuidas hasta tal punto que sobrepase su mínimo o máximo valor del rango.

16. For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?

Todas estas suposiciones son correctas, como podemos ver en las líneas 118 y 120.

17. Are parentheses used to avoid ambiguity?

Quizás en la línea 118 podría ser conveniente utilizar paréntesis para evitar confusiones, aunque no sería necesario.

5. Comparison/Relational Defects (CR)

18. Are the comparison operators correct?

Sí, se ha empleado el > (mayor que) y el < (menor que) en las líneas 57, 77, 78, 115, 118 y 123 de forma correcta y el != (distinto que) en las líneas 113, 118, 165.

19. Is each boolean expression correct?

Sí, vemos un uso adecuado del comparador && (and) en las líneas 78 y 118, y del ! (not) en las líneas 154 y 165.

20. Are there improper and unnoticed side-effects of a comparison?

No se ha notado ninguna.

6. Control Flow Defects (CF)

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

Sí, porque cada bucle realiza su función de la manera más eficiente, como vemos en el bucle de la línea 57, donde se recorre varios arrays para inicializarlos con un for.

22. Will all loops terminate?

Gracias a que las condiciones de los bucles están bien definidas, todos los bucles terminarán tarde o temprano.

23. When there are multiple exits from a loop, is each exit necessary and handled properly?

Según los bucles del programa, no podrán existir múltiples salidas, pues la condición de continuidad es unaria.

24. Does each switch statement have a default case?

No existen sentencias switch.

25. Are missing switch case break statements correct and marked with a comment?

No existen sentencias switch.

26. Is the nesting of loops and branches too deep, and is it correct?

Únicamente existe un bucle for dentro de un bucle while (línea 113). No consideramos que sea demasiado profundo ya que solo está anidado un nivel, siendo tratado de manera correcta.

27. Can any nested if statements be converted into a switch statement?

No es posible. Lo máximo que existe son dos ifs anidado uno dentro del otro, por lo que no es posible transformarlo eficientemente en un switch (tampoco existen sentencias if – else if – else if – else…)

28. Are null bodied control structures correct and marked with braces or comments?

No existen estructuras con cuerpo nulo.

29. Does every function terminate?

Sí, todas las funciones terminan a través de la sentencia return, aunque en el caso de las funciones void (como la de la línea 50), no sería necesario esta sentencia, pues no devuevle nada.

30. Are goto statements avoided?

No existen sentencias goto en la programa, por lo que sí.

7. Input-Output Defects (IO)

31. Have all files been opened before use?

No se emplean ficheros.

32. Are the attributes of the open statement consistent with the use of the file?

No se emplean ficheros.

33. Have all files been closed after use?

No se emplean ficheros.

34. Is buffered data flushed?

No.

35. Are there spelling or grammatical errors in any text printed or displayed?

No hay sentencias print, y no se ha encontrado ningún fallo.

36. Are error conditions checked?

Sí, en la línea 173 se comprueba.

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, hemos comprobado todos los métodos y se corresponden con lo llamado.

38. Do the values in units agree (e.g., inches versus yards)?

Sí, se tratan correctamente las mismas unidades (línea 120, se suman double+double).

9. Comment Defects (CM)

39. Does every function, class, and file have an appropriate header comment?

Sí, tanto la clase como todas sus funciones tienen un comentario previo explicando su propósito (líneas 4, 22, 47…)

40. Does every variable or constant declaration have a comment?

La variable neigh no se explica en ningún momento y se usa en el for (líneas 115-125), las variables cost y next del método nextCur() (líneas 64 y 65) tampoco se explican, aunque su significado es explícito, constantes no hay.

41. Is the underlying behavior of each function and class expressed in plain language?

Sí, el lenguaje es entendible y directo.

42. Is the header comment for each function and class consistent with the behavior of the function or class?

Sí, el comportamiento coincide con lo especificado.

43. Do the comments and code agree?

Sí, los comentarios dentro del código coinciden y además ayudan a aclarar su significado.

44. Do the comments help in understanding the code?

Sí, su comportamiento queda claro.

45. Are there enough comments in the code

Sí, hay suficientes comentarios.

46. Are there too many comments in the code?

Sí, algunos sobran (línea 134 por ejemplo).

10. Packaging Defects (LP)

47. For each file: Does it contain only one class?

Sí, Dijkstra.java contiene solo la clase Dijkstra.

48. For each function: Is it no more than about 60 lines long?

Sí, no sobrepasa las 60 líneas en ningún momento.

49. For each class: Is no more than 2000 lines long (Sun Coding Standard)?

Solo existe una clase con 182 líneas en total (contando comentarios), por lo que no.

11. Modularity Defects (MO)

50. Is there a low level of coupling between packages (classes)?

No hay paquetes.

51. Is there a high level of cohesion within each package?

Sólo hay una clase, por tanto, no podemos hablar de la cohesión.

52. Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?

No, porque el código es breve y está bien optimizado.

53. Are framework classes used where and when appropriate?

Sí, se emplean los arraylist y sus métodos en el momento adecuado.