Portada

Indice

Enunciado

## Ojear documentación:

**Pts. 1, 2, 3**

**¿Cuál es tu primera impresión sobre el sistema? ¿Dónde centrarías tus esfuerzos de refactorización?**

La clase LANTests se podría incluir dentro de lanSimulation, eliminando así un paquete (lanSimulation.test), y facilitando las referencias entre clases. En su lugar, como ambas clases se encargan de realizar pruebas con la red, se podrían pasar ambas clases al mismo paquete lanSimulation.test.

También se podría crear una clase Workstation, la cual estaría en el paquete lanSimulation.internals, que se encargara de la función “requestWorkstationPrintsDocument”. Además, podría tener otros métodos de utilidad para saber si está ocupado, y desde la clase Network se podría saber si un Workstation pertenece a dicha red o no.

A la hora de imprimir autor y título (según el fichero toDoList\_es en la versión 1.3), se podría incluir dicha información como parámetros adicionales, externos al mensaje, origen y destino, para evitar que, dependiendo si es ASCII o postcript, se deban leer de forma diferente.

En la versión 2.0, la configuración de la red debería constar dentro de la propia red, con un método que nos permita consultarla más fácilmente que desde un fichero. Se puede mantener un método que permita cambiar dicha configuración desde fichero en caso de que alguno de sus parámetros de problemas o tenga que ser sobreescrito.

En la versión 3.0, la GUI debería ir en una clase aparte dentro del paquete lanSimulation, y debería estar relacionada únicamente con LANSimulation que a su vez se comunica con el resto de clases (hace de intermediario) o incluso, si dicha interfaz no es excesivamente grande, podría ser uncluida en la misma clase.

## Leer código en 5 minutos:

**Pto. 4**

**¿Cuál es la segunda impresión sobre el sistema? ¿Estás de acuerdo con la impresión inicial? Con este nuevo conocimiento sobre el código, ¿dónde centrarías tus esfuerzos de factorización?**

A primera vista se podrían cambiar los métodos dentro de LANSimulation, excepto el main, a la clase LANTest, ya que es la clase encargada de realizar los tests, dejando LANSimulation exclusivamente para el programa principal.

La clase network.java contiene varios métodos similares, que son los printOn, printHTMLOn y printXMLOn. Se podrían factorizar en un solo método en función del tipo de lenguaje en el que tenga que sacar la información, el cual se pasará como parámetro.

Las variables autor y title en el método printDocument de la clase network, como se comentó en el apartado anterior, podrían estar almacenadas en la clase packet, de maner aque resultaría más sencillo accededr a ellas.

Con un solo constructor de Node sería suficiente, pasando null como nextNode en caso de querer usar el primero en el código actual.

Estas clases (node y packet) deberían tener los atributos privados, de manera que solo se pueda acceder a ellos con los correspondientes get() y set().

## Hacer una instalación de prueba:

**Pts. 5, 6, 7, 8**

**¿Crees que el código base está ya refactorizado? ¿Qué puedes decir de la calidad de los tests: puedes empezar a refactorizar de manera segura?**

Tiene ciertos elementos refactorizados, aunque se podría refactorizar un poco más.

En los tests, según hemos comentado los cambios en los constructores y cambios de clases, habría que cambiar un par de líneas (cuando se crea un nodo nuevo o cuando llamamos a la función requestWorkstationPrintsDocument).

Además, hay muchas líneas de código en las que se usa el método write() de lo que se llama en el código ‘report’, de manera que muchas de ellas se pueden extraer a métodos comunes.

## Habla con los de mantenimiento:

**Desarrolla un plan de proyecto listando a) los riesgos, b) las oportunidades de refactorización (detección de defectos), c) las actividades (plan de refactorizaciones).**

## Extract method:

**Pts. 10, 11, 12**

**¿Estás seguro de que estas refactorizaciones no rompen el código? ¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorización hace un buen trabajo?**

No, no rompen el código ya que, lo único que se está haciendo es pasar las líneas de código duplicadas a un método común, pasando los elementos que sean diferentes por parámetro, de manera que siguen teniendo la misma funcionalidad.

En estos casos puede que no perezcan excesivamente la pena al ser pocas líneas de código, aunque hace la labor de mantenimiento más asequible, pudiendo cambiar un solo método con 4 líneas que cambiar esas 4 líneas en cada sitio del código conde lo usemos.

La herramienta cumple con su trabajo, sacando elementos comunes y encontrando las coincidencias en el código sustituyéndolas automáticamente, de manera que facilita bastante el trabajo.

## Mover el comportamiento cerca de los datos:

**Pts. 13, 14, 15**

**¿Estás seguro de que estas refactorizaciones no rompen el código? ¿Crees que estas refactorizaciones merecen la pena?** **¿La herramienta de refactorización hace un buen trabajo?**

A la hora de crear el report en un sitio (network) y rellenarlo en otro (node) puede quedar poco claro, y el tener que cambiar de clase para ver el camino que sigue dicho report puede no ser la mejor idea, aunque de esta manera se evita acceder a variables o campos de estas clases (nodo) de forma indirecta, pudiendo cambiarlas a privadas en caso de necesitarlo.

La herramienta hace un buen trabajo como en el caso anterior, excepto que añade parámetros que no queremos añadir (en nuestro caso añade la red sobre la que estamos trabajado (Network) y se la pasa como parámetro con ‘this’). Se ha eliminado manualmente porque era algo que no se usaba en este caso.

Además, ha habido un caso en el que no nos dejaba mover un método (accounting) a otra clase (packet), de manera que hemos tenido que moverlo a mano, cambiando las referencias en el método print de la clase packet.

NOTA: en este punto se ha extraído otro método al que hemos llamado “acceptBroadcastPackage()”, que a su vez hemos movido a la clase Node. Le pasamos como parámetro el report en el que vamos a escribir el mensaje, y se encarga de escribir dicho mensaje y el nombre del nodo.

## Eliminar código de navegación:

**Pts. 16, 17**

**¿Estás seguro que estas refactorizaciones no rompen el código? ¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorización hace un buen trabajo?**

Realizando estas refactorizaciones, se consigue aumentar la atomicidad de nuestros objetos, ya que se llaman a los métodos de la clase Nodo en vez de acceder directamente a sus atributos. Por otra parte, también se eliminan líneas repetidas de código en la clase principal (Network), moviendo los switch a un método común. No rompen el código y aumentan la atomicidad, reduciendo las líneas de código.

A la hora de realizar la refactorización de los bucles while() de los métodos requestWorkstationPrintsDocument() y requestBroadcast(), primero se ha realizado una reestructuración de uno de ellos para que ambos fueran do-while. Una vez hecho esto se ha añadido una condición boolean (broadcast) para añadir al report dicho mensaje en caso de que fuera true. Además, se ha añadido una condición dentro de las condiciones del bucle para que evite comprobar si ha llegado de nuevo al destino si no es necesario (con una relación ternaria).