DISEÑO AVANZADO DE SISTEMAS SOFTWARE

Práctica Final: Sesión de Refactoring

Autor:

Víctor de Castro Hurtado

Contents

Contexto

Estas manteniendo un sistema software que representa una simulación de una red de área local (LAN). El equipo de desarrollo ha sido muy rápido en adaptar los requisitos iniciales para el sistema entregando una versión 1.4 que contiene la funcionalidad para el primer hito. El cliente solicita añadir una nueva funcionalidad y el equipo de desarrollo se percata que el diseño no está preparado.

Saben que eres un experto en refactorización, por eso te prestan su código para que lo refactorices apropiadamente. No esperan un diseño perfecto, esperan un diseño que permita añadir la nueva funcionalidad fácilmente. Además, tienen disponibles pruebas del sistema desarrollado.

Enunciado

Abordar la refactorización de un sistema que simula una red de área local (LAN).

La documentación del sistema está contenida en el repositorio público:

https://github.com/clopezno/reafactoring_lab_session

Repositorio

https://github.com/victorcas04/reafactoring_lab_session

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

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

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

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).

Transformar códigos de tipo

Pts. 18, 19, 20, 21, 22, 23

Primero se ha movido el contenido de los métodos 'printOn', 'printXMLOn' y 'printHTMLOn' a la clase Nodo, para después extraer la condición de salida del bucle while en una nueva función llamada 'checkCurrentNode'. Además, se ha extraído el switch de 'printHTMLOn' al mismo método 'switchPrintTypeNode' que los otros dos.

A continuación, se procede a la creación de las subclases 'Workstation' y 'Printer', haciendo que extiendan de Nodo para poder usarlas como si fueran un nodo normal en caso necesario (al establecer el 'nextNode_' en la clase 'Network' por ejemplo).
Se pasan los métodos creados anteriormente a dichas clases, y se cambian las referencias del constructor genérico de la clase 'Node' al constructor específico de cada tipo de nodo en caso necesario.

En nuestro caso las llamadas al nuevo contructor pasaron de ser, por ejemplo: de 'new Node(Node.Printer, "Andy")' a 'new Node().new Printer("Andy")'. Para finalizar, se eliminan todas las referencias al atributo 'type_' del nodo.

NOTA: se han tenido que crear un par de métodos adicionales, el primero que checkea el tipo de nodo que es, ya que no tenemos la variable 'type_', tenemos que comprobar que sea una instancia de una clase determinada (o Workstation, o Printer o Node).

Además, se ha decidido meter el mensaje de nodo desconocido en un método para que tenga una estructura similar a las opciones de Workstation, Printer y Node normal.

IMPORTANTE: se tiene constancia que lo que se pide en este apartado es tener dos subclases independientes, cada una de ellas con tres métodos implementados ('printOn', 'printHtmlOn'y 'printXmlOn'), pero, ya que son tipos especiales de nodos ('Workstation' y 'Printer'), se ha decidido hacer que estas subclases extiendan de la clase original: 'Nodo', de manera que puedan usar métodos comunes, como son los especificados.

Con este cambio se pretende reducir las líneas de código repetido, así como facilitar la reusabilidad en caso de querer añadir nuevos tipos de nodo.

El resultado de esta manera es: los métodos printOn se han dejado en la clase principal Node, y se ha añadido un método 'checkNodeType', que comprueba el tipo de instancia del que es el nodo del que se va a imprimir la información (con un 'node instanceof Subclass'). De esta manera sólo hay que llamar al método 'printOnSwitch' de la subclase en cuestión, el cual se encarga de introducir la información específica, ya sea esa subclase de Workstation o de Print.

Conclusión

Pto. 24

Chequea el fichero "toDoList_es" y argumenta para cada uno de los futuros requisitos cómo tu diseño soportará los cambios.

**** version 2.0: READ FROM FILE***

Lee la configuración de la red y los trabajos que deben ser impresos desde un fichero en formato XML.

- Esta funcionalidad habría que implementarla de 0, ya que en la versión actual no hay implementado nada similar. Por supuesto, habría que hacerlo de tal manera que pueda ser reutilizado en el futuro para leer la configuración desde otro tipo de ficheros.

**** version 2.1: GATEWAY NODE***

Introduce un nuevo nodo "gateway", el cuál pueda reconocer todas las direcciones en su subred actual mirando los 3 caracteres de la dirección.

Si un paquete pasa a través del gateway y la dirección no está en la subred actual, el mensaje es reenviado a través del gateway.

- Esta funcionalidad estaría parcialmente implementada debido a que ya tenemos una opción 'broadcast' en el método 'send' de la clase 'network', de modo que, en este caso, se ha realizado una buena labor de diseño.
  Además, habría que implementar una función para comprobar la dirección de un nodo, y si pertenece o no a la subred.

**** version 2.1: COMPILE LIST OF NODES***

Usando el BROADCAST PACKET, el gateway envia periodicamente una petición a todos los nodos para que contesten con su nombre. Cada gateway puede verificar si los nodos tienen el prefijo correcto.

- Esta funcionalidad, a falta del apartado periódico, estaría implementada en la función 'requestBroadcast'. Por supuesto, habría que seguir checkeando al final que cada nodo tenga el prefijo correcto, algo que no se hace actualemente, pero se podría utilizar la función mencionada en el apartado anterior.

**** version 3.0: GUI***

El sistema debería tener una GUI que debería mostrar una animación del sistema ejecutándose.

- Aparte de implementar la GUI propiamente dicha, en cuanto al diseño no habría problema, ya que hemos hecho los métodos lo suficientemente genéricos como para permitir la inclusión de este tipo de elementos. Por ejemplo, en vez de llamar a las funciones printXMLOn, habría que llamar a la función GUI, peor en cuanto a diseño no haría falta retocar nada.

¿Hay asuntos que no has considerado? ¿Hay refactorizaciones que parecen innecesarias?

En general si que se han considerado útiles las refactorizaciones, aunque hay excepciones, como mover el método 'printDocument' como 'print' a la clase 'Packet'. Si que es cierto que debería pertenecer a esa clase, ya que todo lo que imprime viene de dicha clase, pero aparte de eso no se ha retocado para nada, y es quizá el método más largo del proyecto (después de refactorizar el resto), por lo que habría sido conveniente extraer algún método o buscar formas de reducirlo un poco.
Además, la última parte de eliminar la variable 'type_' y usar el tipo de clase para el acceso a métodos, puede que no hiciera falta. Si que es cierto que, cuantas menos variables tengamos mejor, ya que habrá menos posibilidades de equivocarse con ellas, pero la variable eliminada era bastante útil y funcionaba bien.

Preguntas de Reflexión

- ¿Se puede automatizar completamente el proceso de refactorización a través de herramientas?

La mayor parte si, aunque hay ocasiones en las que se tiene que refactorizar a mano, como es el caso de las subclases Workstation y Printer.

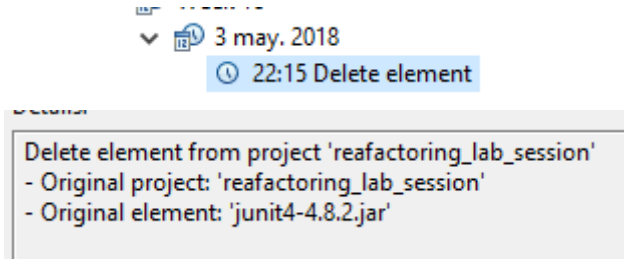
- ¿Qué relación encuentras entre el proceso de refactorización y la utilización de sistemas de control de tareas y versiones?

Cada proceso de refactorización debería tener asignado una tarea específica, de manera que se pueda controlar mediante las diferentes versiones que refactorización funciona mejor que otra, o si una falla por cualquier motivo, saber en qué tarea ha dejado de tener el comportamiento deseado para saber a qué tipo de refactorización correspondía esa tarea.

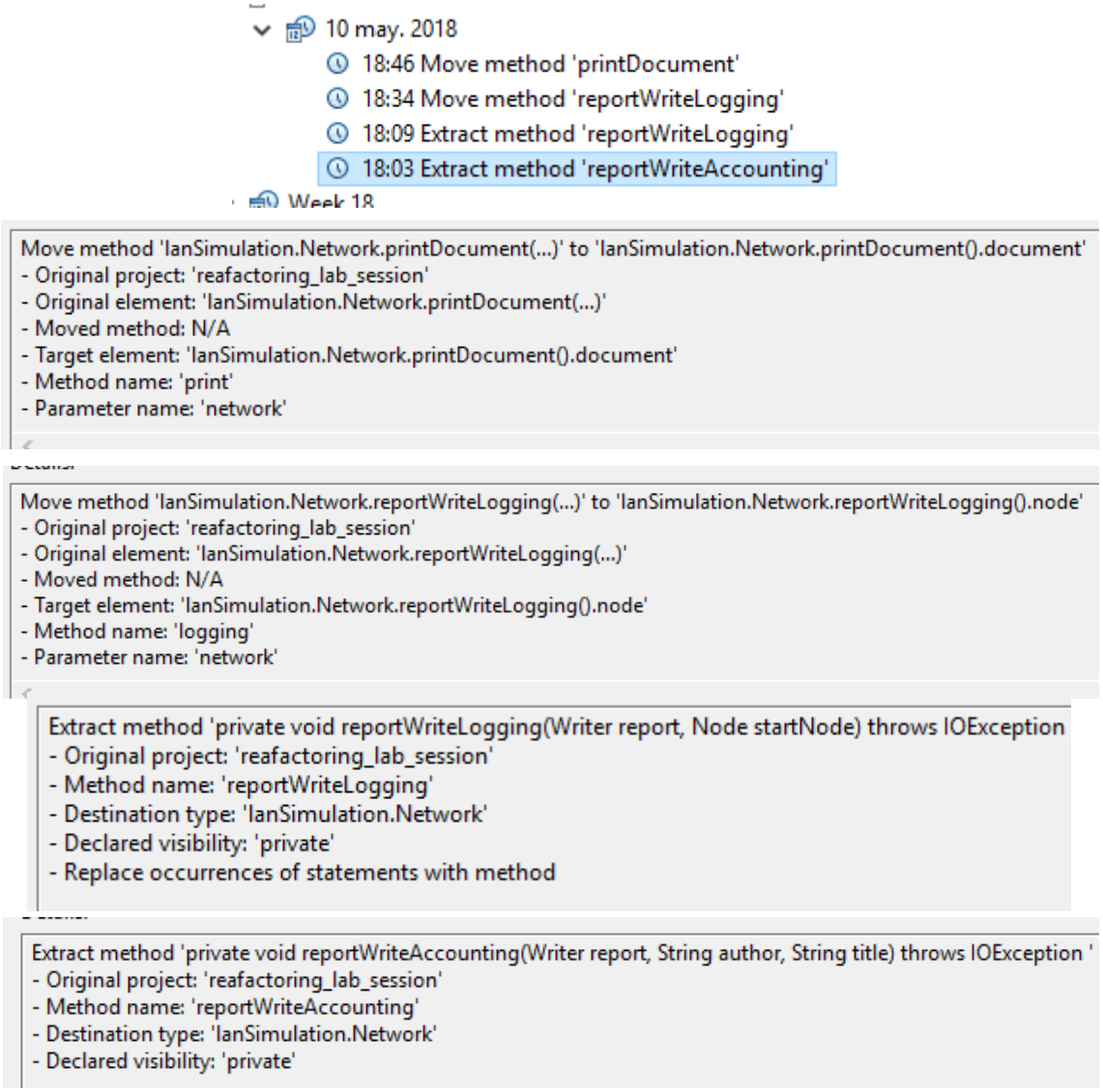
Histórico de Refactorizaciones

A continuación, se muestra la información contenida en el histórico de factorizaciones. Se pueden distinguir 4 grandes agrupaciones de refactorización, divididas por fechas:

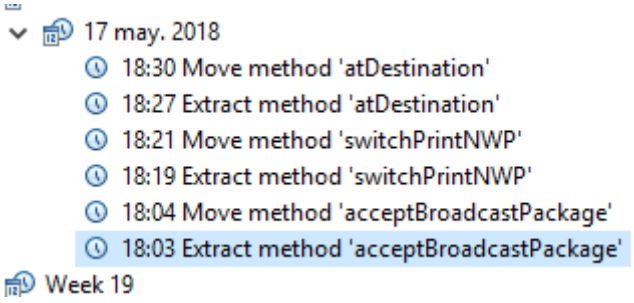
1.- 3 mayo:



2.- 10mayo:



2.- 17 mayo:



Move method 'lanSimulation.Network.atDestination(...)' to 'lanSimulation.Network.atDestination().currentNode'
- Original project: 'reafactoring_lab_session'
- Original element: 'lanSimulation.Network.atDestination(...)'
- Moved method: N/A
- Target element: 'lanSimulation.Network.atDestination().currentNode'
- Method name: 'atDestination'
- Parameter name: 'network'

Extract method 'private boolean atDestination(Node currentNode, Packet packet)' from 'lanSimulation.Network.requestBroadcast()' to 'lanSimulation.Network'
- Original project: 'reafactoring_lab_session'
- Method name: 'atDestination'
- Destination type: 'lanSimulation.Network'
- Declared visibility: 'private'
- Replace occurrences of statements with method

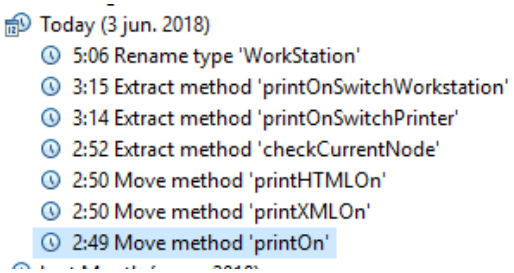
Move method 'lanSimulation.Network.switchPrintNWP(...)' to 'lanSimulation.Network.switchPrintNWP().currentNode'
- Original project: 'reafactoring_lab_session'
- Original element: 'lanSimulation.Network.switchPrintNWP(...)'
- Moved method: N/A
- Target element: 'lanSimulation.Network.switchPrintNWP().currentNode'
- Method name: 'switchPrintTypeNode'
- Parameter name: 'network'

Extract method 'private void switchPrintNWP(StringBuffer buf, Node currentNode)' from 'lanSimulation.Network.printOn()' to 'lanSimulation.Network'
- Original project: 'reafactoring_lab_session'
- Method name: 'switchPrintNWP'
- Destination type: 'lanSimulation.Network'
- Declared visibility: 'private'
- Replace occurrences of statements with method

Move method 'lanSimulation.Network.acceptBroadcastPackage(...)' to 'lanSimulation.Network.acceptBroadcastPackage().currentNode'
- Original project: 'reafactoring_lab_session'
- Original element: 'lanSimulation.Network.acceptBroadcastPackage(...)'
- Moved method: N/A
- Target element: 'lanSimulation.Network.acceptBroadcastPackage().currentNode'
- Method name: 'acceptBroadcastPackage'
- Parameter name: 'network'

Extract method 'private void acceptBroadcastPackage(Writer report, Node currentNode) throws IOException '
- Original project: 'reafactoring_lab_session'
- Method name: 'acceptBroadcastPackage'
- Destination type: 'lanSimulation.Network'
- Declared visibility: 'private'

2.- 3 junio:



Rename type 'lanSimulation.internals.Node.WorkStation' to 'Workstation'
- Original project: 'reafactoring_lab_session'
- Original element: 'lanSimulation.internals.Node.WorkStation'
- Renamed element: 'lanSimulation.internals.Node.Workstation'
- Update references to refactored element
- Update textual occurrences in comments and strings

Extract method 'private void printOnSwitchWorkstation(StringBuffer buf, boolean t)'
- Original project: 'reafactoring_lab_session'
- Method name: 'printOnSwitchWorkstation'
- Destination type: 'lanSimulation.internals.Node'
- Declared visibility: 'private'

Extract method 'private void printOnSwitchPrinter(StringBuffer buf, boolean t)'
- Original project: 'reafactoring_lab_session'
- Method name: 'printOnSwitchPrinter'
- Destination type: 'lanSimulation.internals.Node'
- Declared visibility: 'private'

Extract method 'private boolean checkCurrentNode(Node currentNode)' from 'lanSimulation.internals.Node.printOn()' to 'lanSimulation.internals.Node'
- Original project: 'reafactoring_lab_session'
- Method name: 'checkCurrentNode'
- Destination type: 'lanSimulation.internals.Node'
- Declared visibility: 'private'
- Replace occurrences of statements with method

Move method 'lanSimulation.Network.printHTMLOn(...)' to 'lanSimulation.Network.firstNode_'
- Original project: 'reafactoring_lab_session'
- Original element: 'lanSimulation.Network.printHTMLOn(...)'
- Moved method: N/A
- Target element: 'lanSimulation.Network.firstNode_'
- Method name: 'printHTMLOn'
- Parameter name: 'network'

Move method 'lanSimulation.Network.printXMLOn(...)' to 'lanSimulation.Network.firstNode_'
- Original project: 'reafactoring_lab_session'
- Original element: 'lanSimulation.Network.printXMLOn(...)'
- Moved method: N/A
- Target element: 'lanSimulation.Network.firstNode_'
- Method name: 'printXMLOn'
- Parameter name: 'network'

Move method 'lanSimulation.Network.printOn(...)' to 'lanSimulation.Network.firstNode_'
- Original project: 'reafactoring_lab_session'
- Original element: 'lanSimulation.Network.printOn(...)'
- Moved method: N/A
- Target element: 'lanSimulation.Network.firstNode_'
- Method name: 'printOn'
- Parameter name: 'network'