Curso Spring – RichFaces – Desarrollando una Aplicación desde POJOs

Marzo 2011 – JoeDayz

Susan Inga

Contenido

- ✓ Introducción
- ✓ ¿Qué aprenderemos?
- ✓ Ganaremos experiencia en…
- ✓ Instrucciones
- ✓ Creando la aplicación
- ✓ Paso 1: Crear implementación de RewardNetwork
- ✓ Paso 2: Implementando la configuración de RewardNetworkImpl
- ✓ Paso 3: Implementando la lógica para RewardNetworkImpl
- ✓ Paso 4: Probando la lógica de RewardNetworkImpl con Unit Test
- ✓ También viene…

Reward Dining

Desarrollando una Aplicación desde Plain Java Objects (POJOs)

Introducción

- ✓ En este laboratorio implementarás y probarás la aplicación Reward Dinnig haciendo uso de Plain Java Objects (POJOs) configurados a través de inyección de dependencias. No vas a codificar todo tú solo, pero lo harás en las **piezas centrales de la aplicación** con **pruebas** que verifiquen que todo funcione correctamente. Esto te dará la oportunidad de familiarizarte con el dominio que se usará en el resto del curso.
- ✓ Cuando lo hayas hecho, podrás observar que la lógica es limpia y no está acoplada con la infraestructura del API.
- ✓ Entenderás que no necesitas Spring solo para desarrollar y ejecutar pruebas unitarias para la lógica de negocio. Es más, lo que desarrolles en este laboratorio puede ser directamente corrido en un entorno Spring sin cambiarlo, y lo comprobarás en futuros laboratorios.

¿Qué aprenderemos?

- ✓ El dominio de un negocio real que aplicará Spring en el resto del curso.
- ✓ ¿Cómo usar una arquitectura en capas para dividir una aplicación en componentes con roles definidos?
- ✓ ¿Cómo usar la inyección de dependencias para pasar un componente que requieres para trabajar?
- ✓ ¿Cómo programar interfaces para encapsular la complejidad de la implementación?
- ✓ ¿Cómo usar JUnit para probar el comportamiento de la aplicación?

Ganaremos experiencia en…

- ✓ Eclipse
- ✓ Plain Old Java Objects (POJOs)
- ✓ Inyección de Dependencias por constructor
- ✓ JUnit

Instrucciones

- ✓ Antes de empezar este laboratorio, se recomienda leer la referencia del curso para entender el contexto de la aplicación.

Creando la aplicación

- ✓ Antes que uses Spring para configurar la aplicación, las piezas de la aplicación deben ser definidas.
- ✓ La aplicación de contribuciones consiste de muchas piezas que trabajan junto con las cuentas de contribución para la recaudación en las cenas de los restaurantes.
- ✓ En este laboratorio, la mayoría de piezas han sido implementadas para ti. Sin embargo, la pieza central, la interfaz `RewardNetwork`, no.
- ✓ En los siguientes pasos tu implementarás esta pieza faltante.

Paso 1: Crear implementación de `RewardNetwork`

- ✓ La interface `RewardNetwork` es el principal responsable de aprobar el ingreso de las operaciones `rewardAccountFor(Dining)`.
- ✓ En este paso crearás una clase que implementará esta interface. A continuación la implementación que debes realizar:

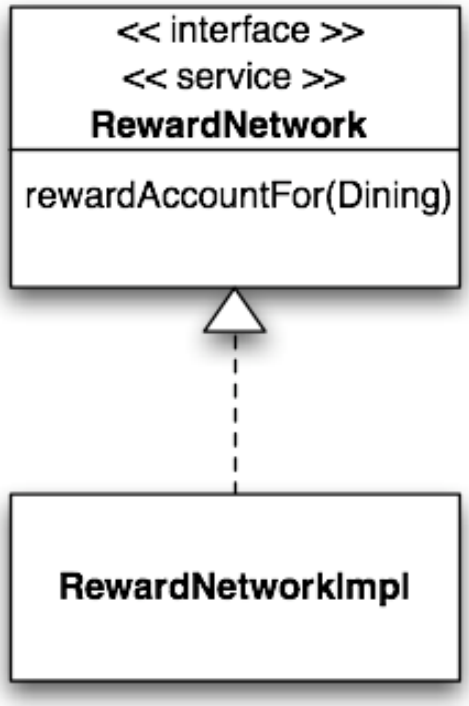


Figura 1: `RewardNetworkImpl` implementa la interface `RewardNetwork`

Paso 1: Crear implementación de `RewardNetwork`

- ✓ Dale una mirada al proyecto `01-spring-intro-1` en el IDE Eclipse.
- ✓ Navega dentro de `src/main/java` y encontrarás el paquete raíz `rewards`. Dentro de este paquete encontrarás la definición de la interface `RewardNetwork`:

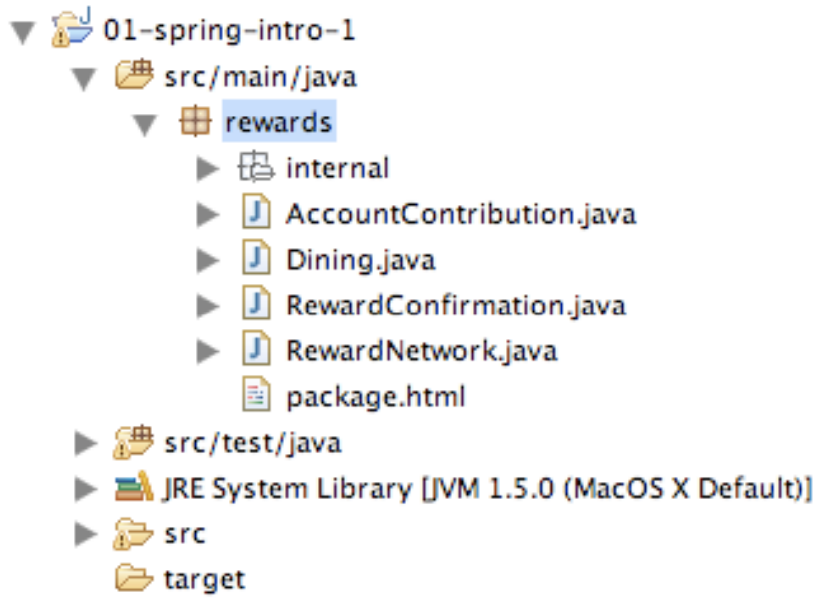


Figura 2: El paquete `rewards`

Esta imagen usa la vista de Presentación "Jerárquica" de Paquetes ("Hierarchical" Package Presentation) en lugar de la vista "Plana" que siempre va por defecto.

Paso 1: Crear implementación de `RewardNetwork`

- ✓ Las clases dentro del paquete `rewards` están definidas como públicas para la aplicación, siendo `RewardNetwork` la parte central.
- ✓ Abrir `RewardNetwork.java` y revisarla.
- ✓ Ahora debes expandir el paquete `rewards.internal` y notarás que no hay clases aún. Aquí es donde los artefactos implementados de la aplicación deben ser definidos.

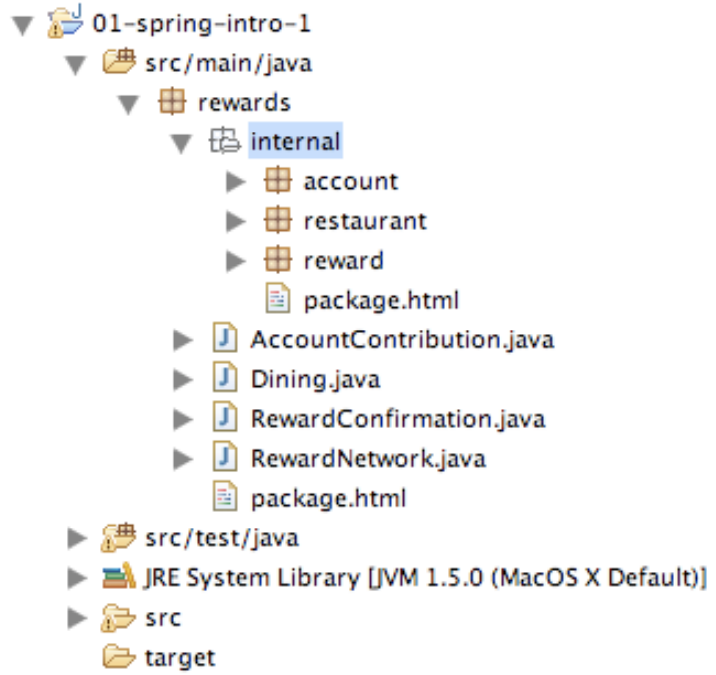


Figura 3: El paquete `rewards.internal`

Paso 1: Crear implementación de `RewardNetwork`

- ✓ Crear la clase `RewardNetworkImpl` dentro del paquete `rewards.internal`, la cual implementará la interface `RewardNetwork`:

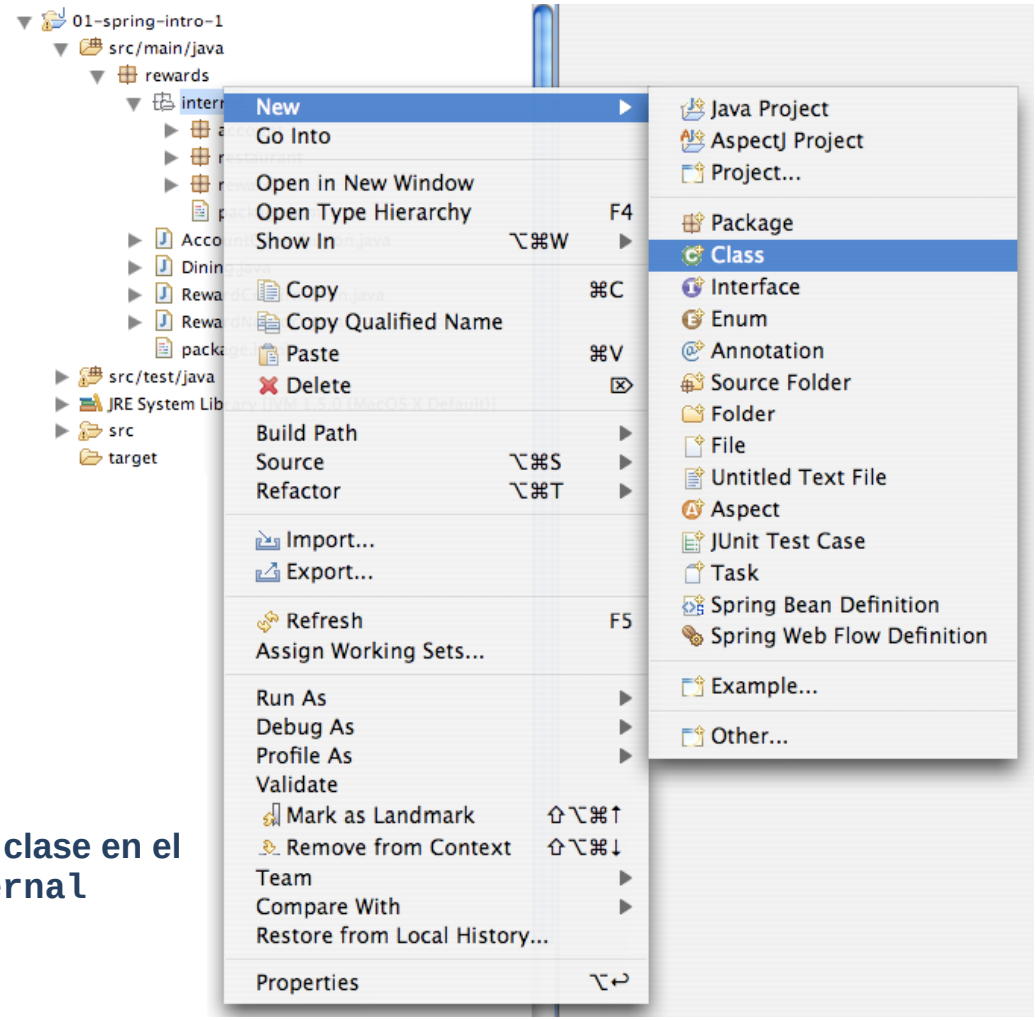


Figura 4: Creando una nueva clase en el paquete `rewards.internal`

Paso 1: Crear implementación de `RewardNetwork`

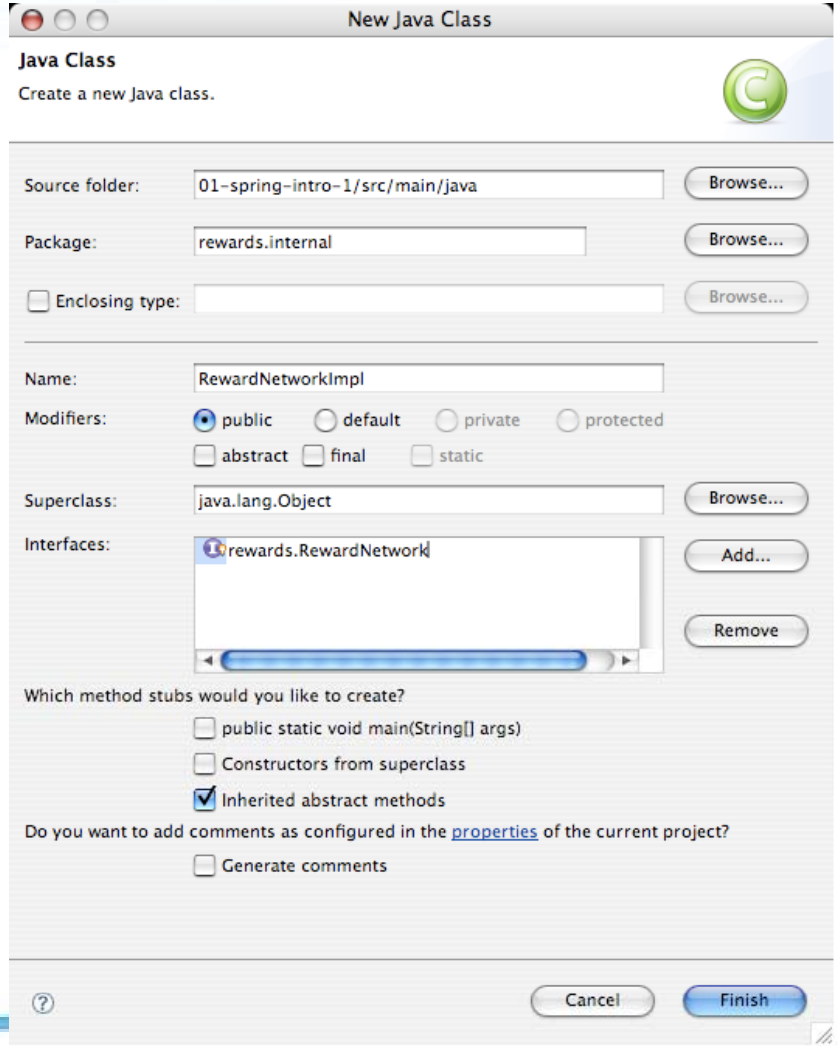


Figura 5: Creando `RewardNetworkImpl`, clase que implementa la interface `RewardNetwork`

Paso 1: Crear implementación de `RewardNetwork`

- ✓ Elegir finish. Eclipse generará una implementación como la siguiente:

RewardNetworkImpl.java

```
1  package rewards.internal;
2
3  import rewards.Dining;
6
7  public class RewardNetworkImpl implements RewardNetwork {
8
9      public RewardConfirmation rewardAccountFor(Dining dining) {
10         // TODO Auto-generated method stub
11         return null;
12     }
13
14 }
```

Figura 6: Cáscara de la implementación de `RewardNetworkImpl`

- ✓ Una vez que tienes el equivalente de la Figura 6 en tu Eclipse Java editor, debes pasar al siguiente paso!

Paso 2: Implementando la configuración de RewardNetworkImpl

- ✓ Para que tu RewardNetworkImpl se encuentre disponible a realizar su trabajo de **recaudar las contribuciones de las cenas**, se requiere algunos servicios de soporte para delegar las responsabilidades. En este paso expresarás una dependencia sobre cada uno de los servicios pre-existentes en RewardNetworkImpl y se generará un constructor para que puedan ser inyectados.

- ✓ Específicamente, RewardNetworkImpl necesita tres servicios de acceso a datos llamados 'Repositories' para realizar este trabajo. Estos servicios son:

  - Un AccountRepository para cargar objetos de tipo Account para realizar las contribuciones a los beneficiarios.
  - Un RestaurantRepository para cargar objetos Restaurant que calculen los beneficios a abonar en las cuentas.
  - Un RewardRepository para marcar las transacciones que confirmen las contribuciones, para propósitos de contabilidad y reportes.

Paso 2: Implementando la configuración de RewardNetworkImpl

- ✓ Esta relación se muestra gráficamente a continuación:

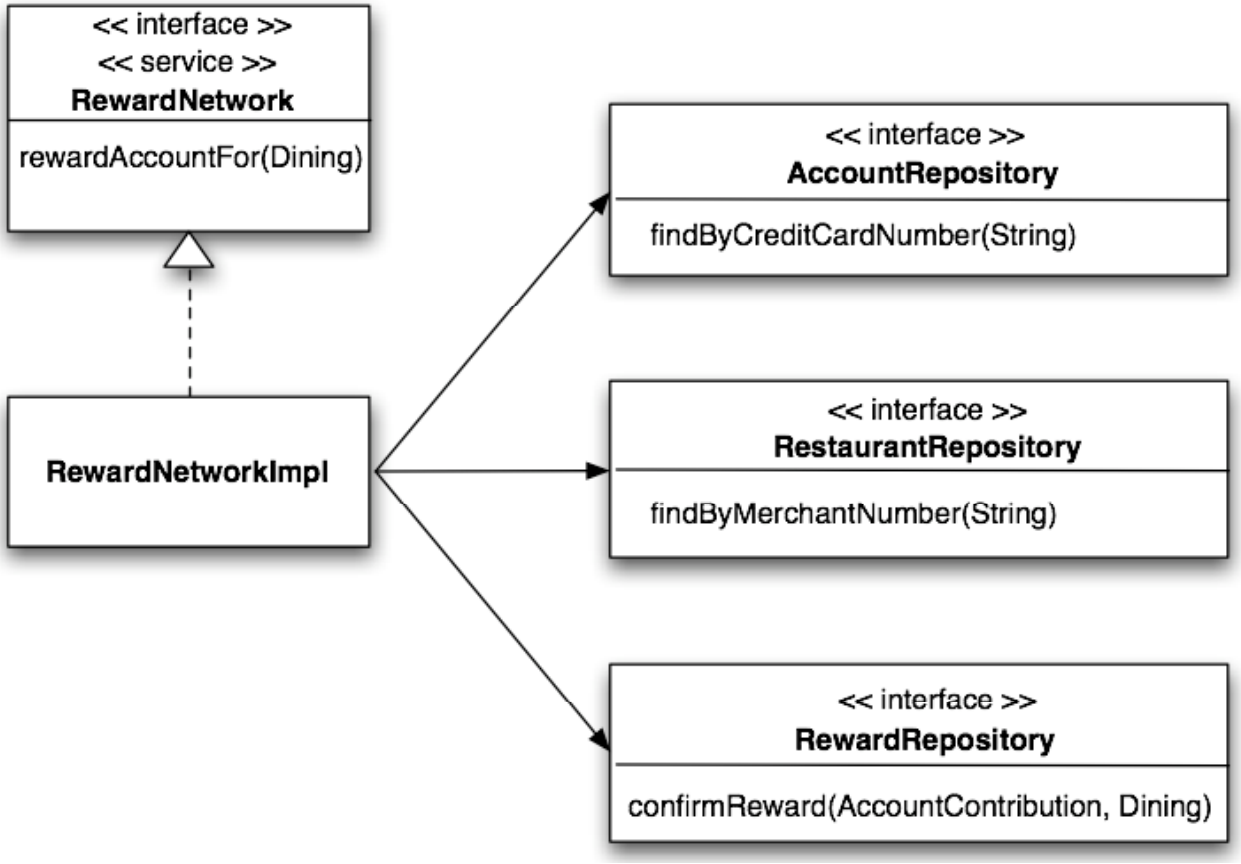


Figura 7: Diagrama de clase RewardNetworkImpl

Paso 2: Implementando la configuración de RewardNetworkImpl

- ✓ En tu proyecto, puedes encontrar cada una de estas interfaces repositorios en la perspectiva domain packages dentro del paquete rewards.internal. Como se ve a continuación:

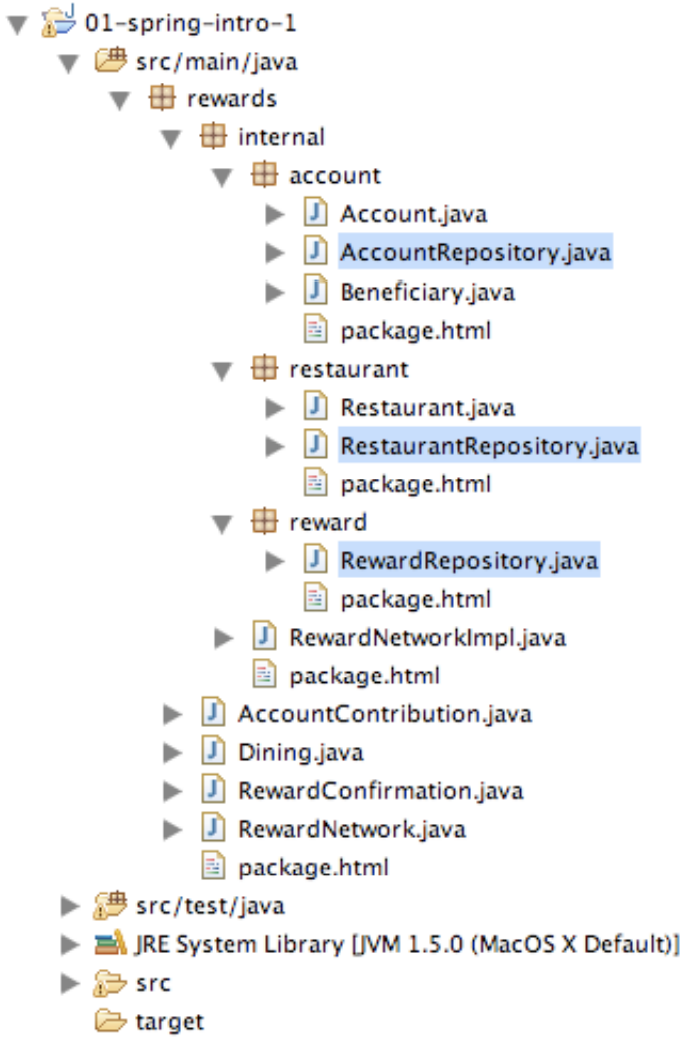
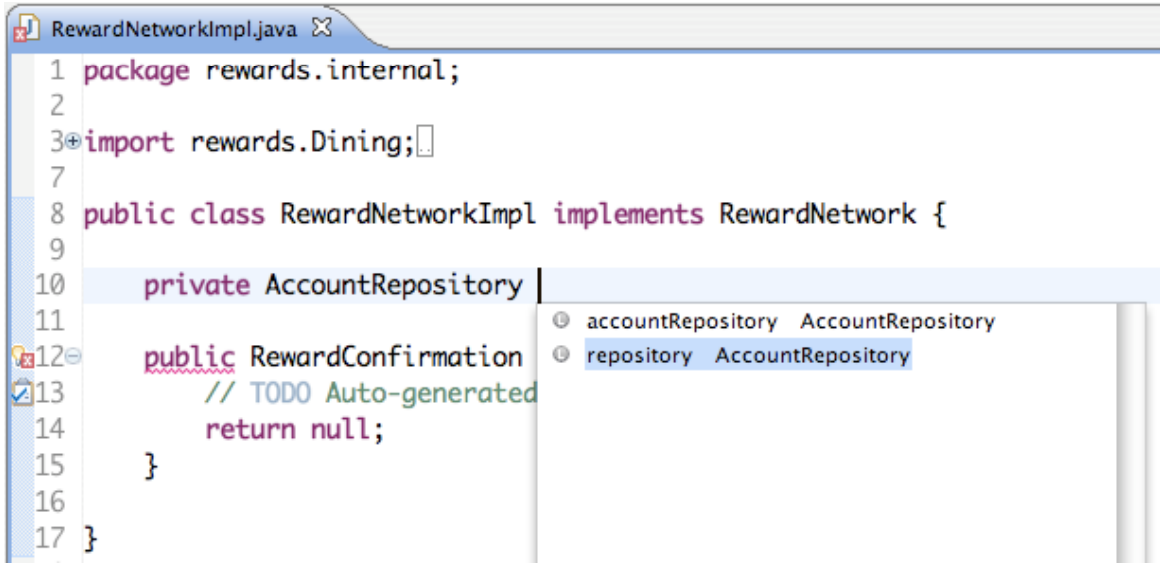


Figura 8: Repository packaging

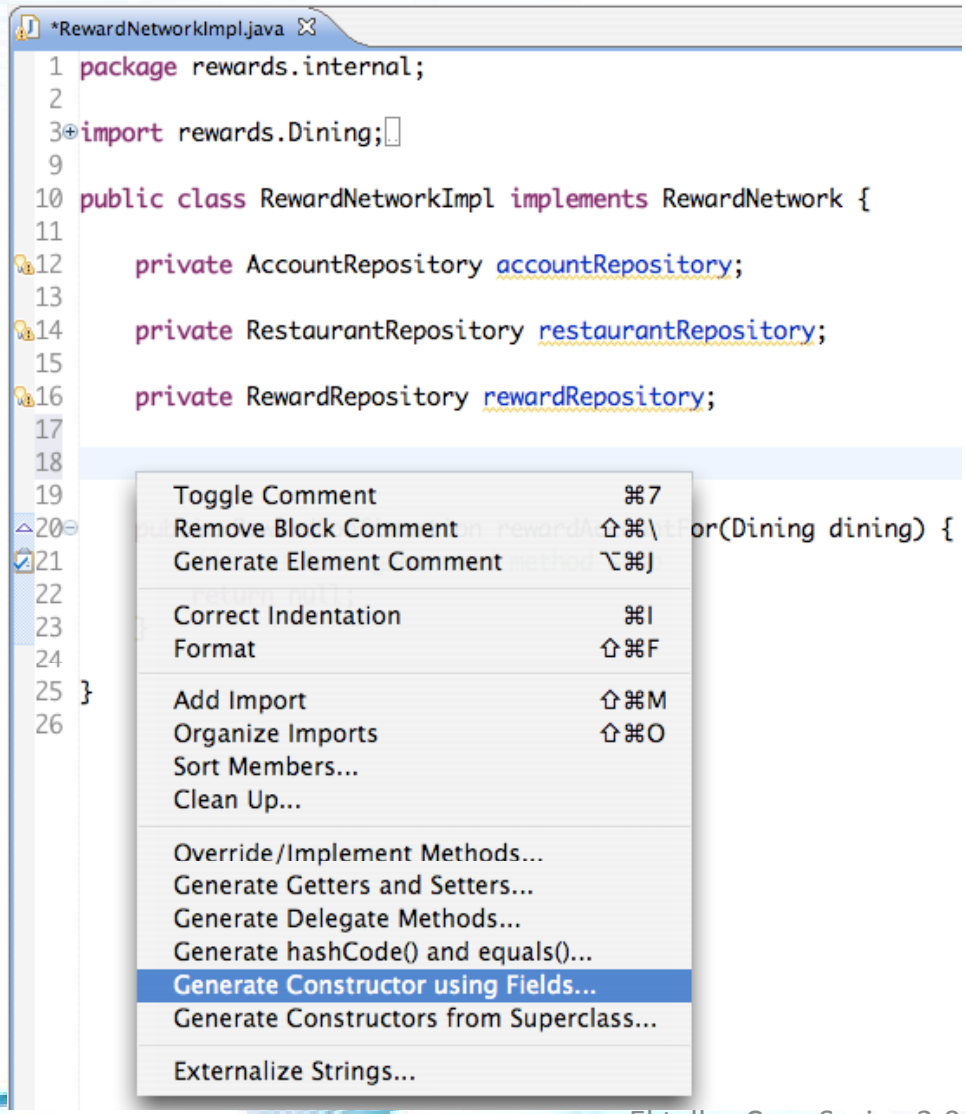
Paso 2: Implementando la configuración de RewardNetworkImpl

- ✓ Para cada dependencia de repositorio, debes agregar un atributo privado a RewardNetworkImpl para expresar la asociación en código Java. Luego, definir un constructor para las dependencias que serán inyectadas cuando RewardNetworkImpl se construya. Inicializar cada atributo desde su argumento respectivo en el constructor.



Usa el autocomplete de Eclipse para ayudarte a definir cada atributo.

Paso 2: Implementando la configuración de RewardNetworkImpl



Después de definir tus atributos, usa el comando Generate Constructors From Fields para generar un constructor para los tres atributos que deben inicializarse.

- ✓ Cuando sientas que ya has terminado exitosamente esta parte, puedes pasar al siguiente paso!

Paso 3: Implementando la lógica para RewardNetworkImpl

- ✓ Más adelante tienes que definir la lógica de la configuración requerida para darle a RewardNetworkImpl lo que necesita para trabajar, pero aún no lo necesitas hacer...por ahora.
- ✓ En este paso implementarás la lógica necesaria para completar la operación rewardAccountFor(Dining), delegando tus dependencias.
- ✓ Por cierto, no te preocupes rewardAccountFor(Dining) es una simple unidad de trabajo, el trabajo pesado se delegará a los servicios de soporte.
- ✓ Seguiremos por volver a revisar la implementación rewardAccountFor(Dining) de RewardNetworkImpl, que aún no hace nada:

```
public RewardConfirmation rewardAccountFor(Dining dining) {
    // TODO Auto-generated method stub
    return null;
}
```

Figura 9: TODO - Abonar una cuenta por el concepto de una cena

Paso 3: Implementando la lógica para RewardNetworkImpl

- ✓ Ahora debes codificar los siguientes paso para completar la implementación del método:

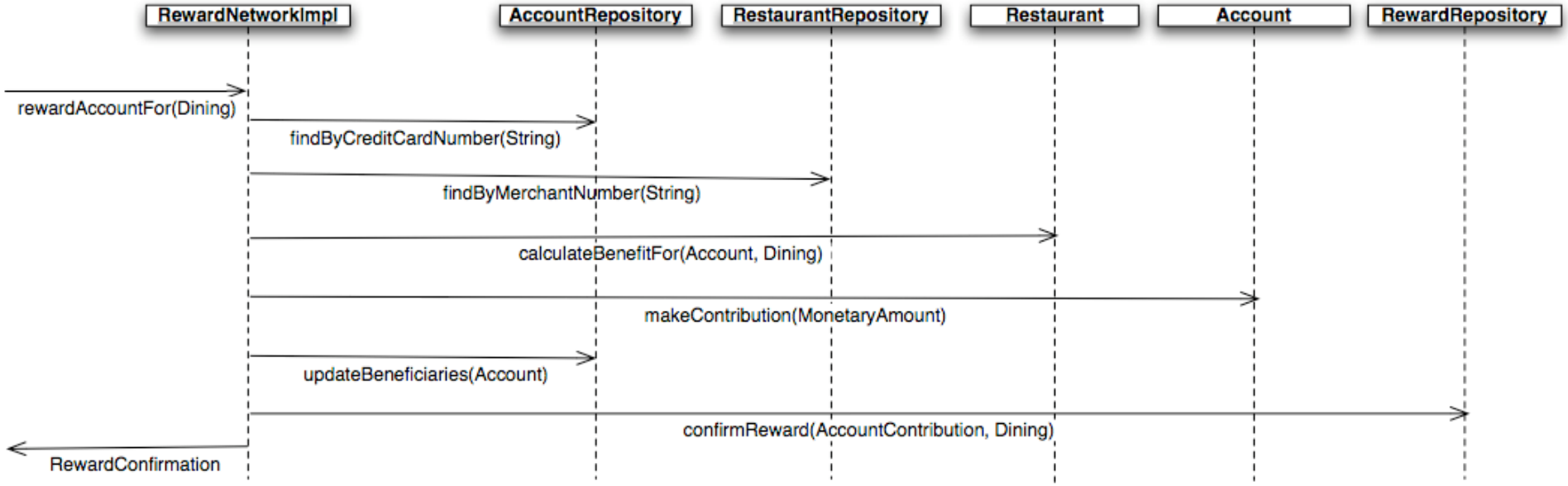


Figura 10: La secuencia del método rewardAccountFor(Dining)

Los números de tarjetas de crédito y de restaurantes los puedes sacar desde el objeto Dining.

- ✓ Una vez que te sientas bien con tu implementación, continua con los siguientes pasos del laboratorio!

Paso 4: Probando la lógica de RewardNetworkImpl con Unit Test

- ✓ Cómo es que sabes que la lógica de la aplicación que acabas de codificar funciona? No lo sabes, no sin pruebas que lo demuestren. En este paso correrás una prueba automatizada con JUnit para verificar que lo codificado es correcto.

- ✓ Navega dentro de src/test/java y podrás observar el paquete rewards. Todas las pruebas para la Rewards Application residen dentro de este árbol al mismo nivel del código del ejercicio. Revisando dentro del paquete rewards.internal puedes encontrar RewardNetworkImplTests, el JUnit TestCase para la clase RewardNetworkImpl.

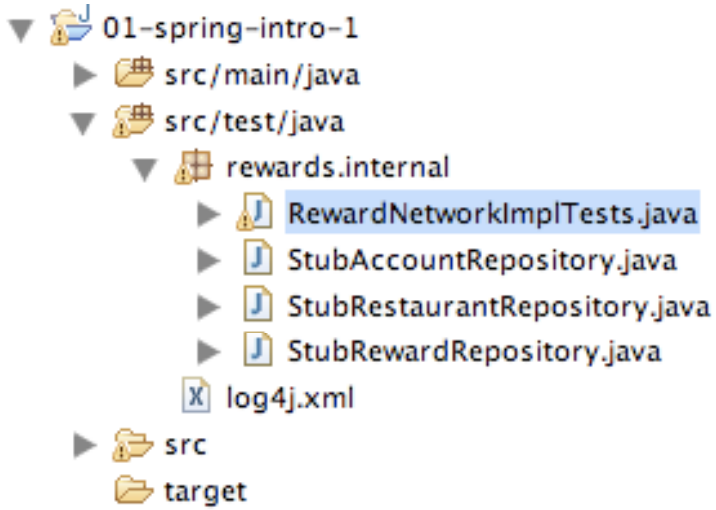


Figura 11: El árbol de pruebas de las contribuciones

Paso 4: Probando la lógica de RewardNetworkImpl con Unit Test

- ✓ Antes que vayas más lejos, primero corre el test para ver que pasa. Para correrlo, has click derecho sobre RewardNetworkImplTests y elige Run As -> JUnit Test. Verás que una barra roja aparecerá en la vista de JUnit, lo que nos indica que el test ha fallado.

- ✓ Ahora abre RewardNetworkImplTests y rápidamente revísalo. Notarás las dos declaraciones TODO. Si revisas la vista de tareas de Eclipse (Window -> Show View -> Tasks) podrás ver estas dos declaraciones que faltan completar. Elige alguna de ella para ir al código de pruebas.

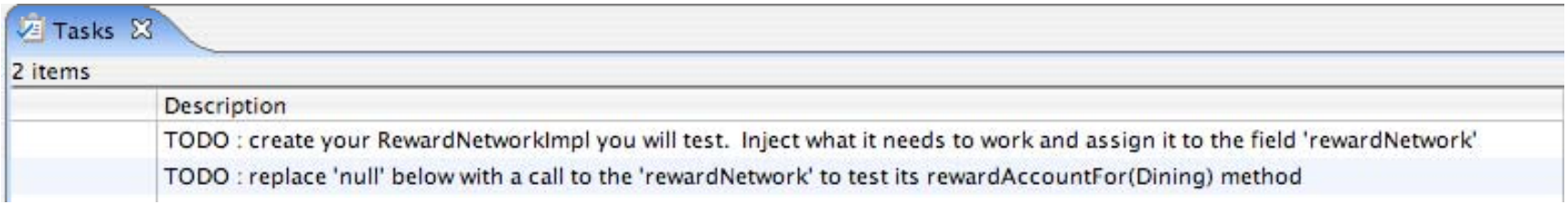


Figura 12: La vista de tareas en la parte inferior del área de trabajo

Paso 4: Probando la lógica de RewardNetworkImpl con Unit Test

- ✓ El objetivo de este paso es pasar la prueba completando los TODOs marcados. Específicamente:
  - ➢ El primer TODO te invita a configurar una instancia de RewardNetworkImpl para que la pruebes. Puedes ver que los 'stub' repositories ya han sido creados para ti. Entonces tu nueva clase simplemente debe inyectar esas dependencias, y asignar los objetos a atributos privados que puedas probar.
  - ➢ El segundo TODO te invita a probar el método rewardAccountFor(Dining). Podrás ver la prueba de la lógica que ha sido, en su mayoría, escrita por ti. Lo único que debes hacer es invocar el método y asignar el valor de retorno a la variable de confirmación, entonces las aserciones podrán ser ejecutadas nuevamente.
- ✓ Cuando te sientas bien con la lógica de las pruebas que has realizado puede volver a correr el test. Esta vez debe aparecer una barra verde.

Paso 4: Probando la lógica de RewardNetworkImpl con Unit Test

- *Usa tu debugger step by step a través de tu prueba y en la lógica de aplicación para ayudarte a entender como es que los componentes trabajan. Nota como es que cada pieza de la aplicación trabaja en conjunto para ejecutar el comportamiento de la aplicación, y como tu RewardNetwork realiza la coordinación para que el trabajo sea hecho.*

- *Revisa las implementaciones repositorios stub para ver como se prueban las aserciones, verificando las expectativas basadas en el estado de los stubs.*

- ✓ Cuando hayas obtenido la barra verde, felicitaciones! Has completado el laboratorio! Solo has tenido que desarrollar y probar un componente de una gran aplicación, ejecutándola exitosamente en un ambiente de pruebas dentro del IDE.
- ✓ Usaste stubs para probar la lógica de aplicación de manera aislada, sin involucrar dependencias externas, tales como una base de datos o Spring. Y la lógica de la aplicación quedó limpia y desacoplada de la infraestructura del API.

También viene…

- ✓ En el siguiente laboratorio, usarás Spring para configurar esta misma aplicación con componentes reales, incluyendo implementaciones reales para los accesos a base de datos!
- ✓ Entonces podrás correr un sistema de pruebas top-down para verificar que todos los componentes trabajen juntos. De esta manera, podrás observar el valor que Spring provee a la configuración y pruebas del sistema.

Enjoy it!