Curso Spring – RichFaces – Usando Spring y sus anotaciones para configurar una aplicación

Marzo 2011 – JoeDayz
Susan Inga

Contenido

Reward Dining

Usando Spring y sus anotaciones para configurar una aplicación

Introducción

- ✓ En este laboratorio ganarás experiencia usando Spring para configurar la Reward Application completamente.
- ✓ Usarás Spring para configurar las piezas de la aplicación usando el soporte para anotaciones de la versión 3.0 en la aplicación de contribuciones.
- ✓ Usarás una instalación existente y la transformarás usando anotaciones tales como @Autowired y @Component para configurar las piezas de la aplicación.

¿Qué aprenderemos?

- ✓ La figura completa: ¿cómo Spring "ataca" en la arquitectura de una de una típica aplicación Java EE?
- ✓ ¿Cómo se usa Spring para configurar Plain Java Objects (POJOs)?
- ✓ ¿Cómo se usa Spring para organizar la configuración de archivos efectivamente?
- ✓ ¿Cómo crear un Spring ApplicationContext?
- ✓ ¿Cómo usar anotaciones de inyección de dependencia de Spring tales como @ Component y @Autowired?
- ✓ Ventajas y desventajas de estas anotaciones.
- ✓ ¿Cómo se usa el Spring IDE para visualizar la configuración de la aplicación?
- ✓ ¿Cómo Spring se combina con modernas herramientas de desarrollo para facilitar el proceso de pruebas?

Ganaremos experiencia en…

- ✓ Inyección de dependencias vía constructor
- ✓ Inyección de dependencias vía setter
- ✓ Inyección de dependencias basadas en anotaciones
- ✓ Sintaxis de la configuración XML Spring
- ✓ Spring IDE
- ✓ Factory Beans

Instrucciones

- ✓ Las instrucciones para este laboratorio están divididas en dos secciones.
- ✓ En la primera sección, usarás Spring para configurar las piezas de las aplicación de contribuciones.
- ✓ En la segunda sección, correrás un sistema de pruebas para verificar todas que todas las piezas trabajen juntas ejecutando el comportamiento de la aplicación correctamente. Diviértete!

Creando la configuración de la aplicación

- ✓ Mucho más adelante codificarás tu RewardNetworkImpl, la pieza central de esta aplicación de contribuciones. La probarás y verificarás que trabaje de manera aislada con su respectivos repositorios stub.
- ✓ Ahora es tiempo para atar juntas todas las piezas reales de aplicación, integrando tu código con los servicios de soporte que se te han proveído.
- ✓ En los siguientes pasos usarás Spring para configurar la aplicación completa...pero la configurarás por partes. Esto incluye las implementaciones de las clases de acceso a datos para las cuales se usará un JDBC data source para acceder a la base de datos relacional!

Creando la configuración de la aplicación

- ✓ A continuación el diagrama de configuración que muestra los componentes de la aplicación de contribuciones, tu mismo configurarás y que deben estar relacionados:

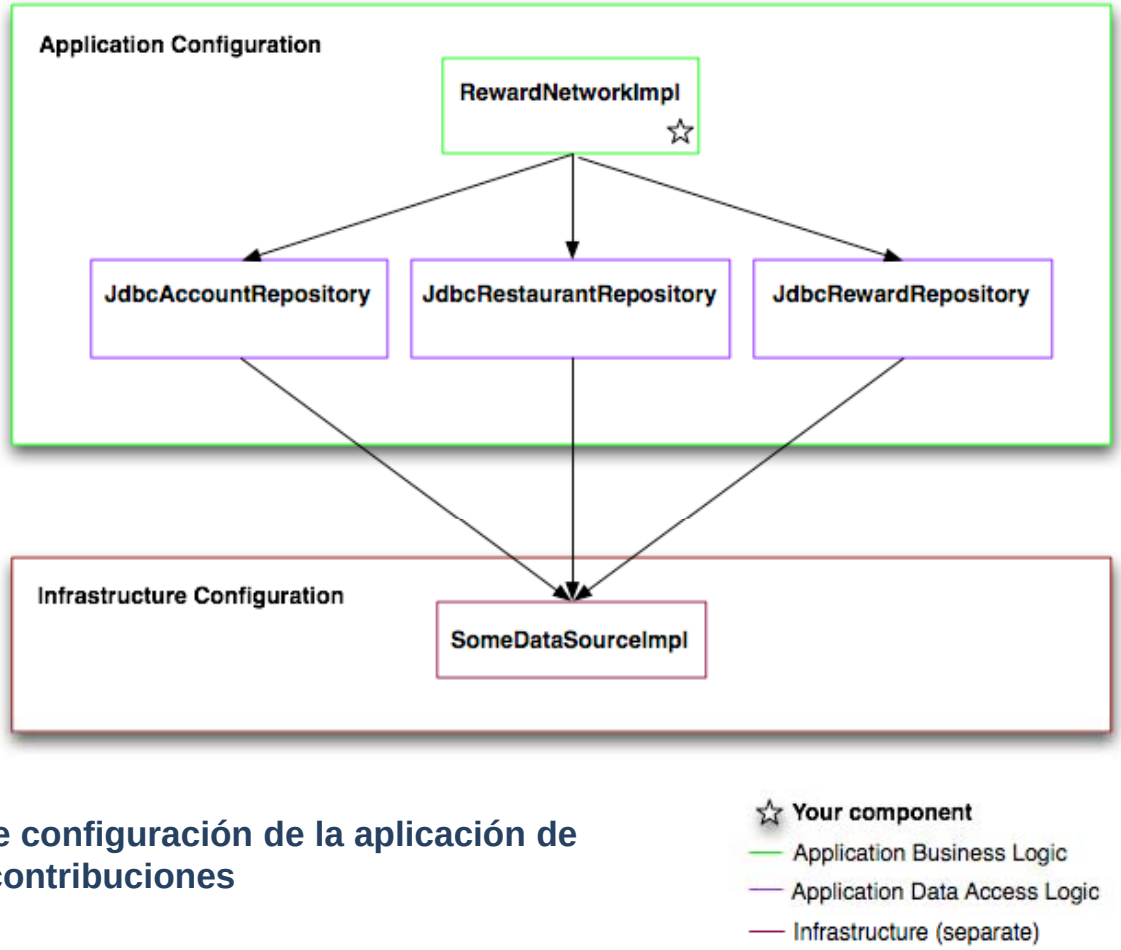
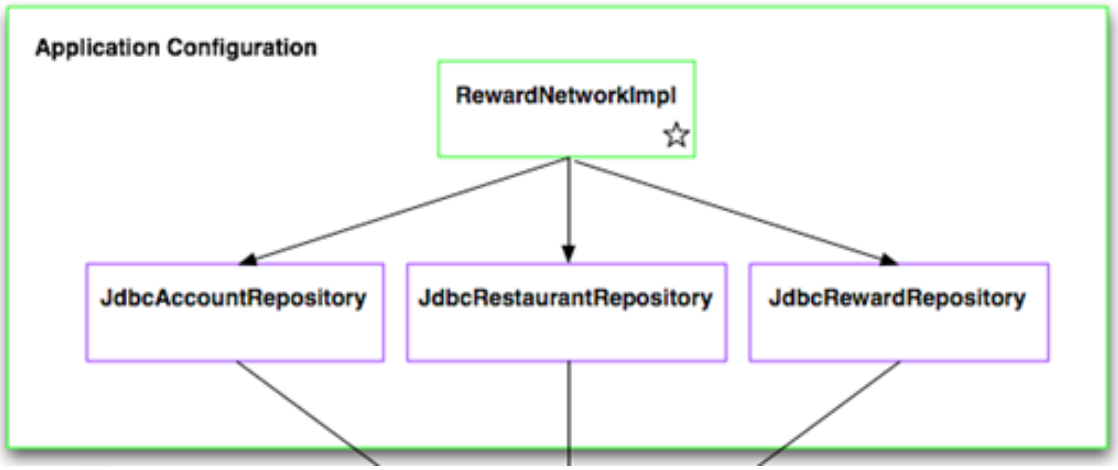


Figura 1: Diagrama de configuración de la aplicación de contribuciones

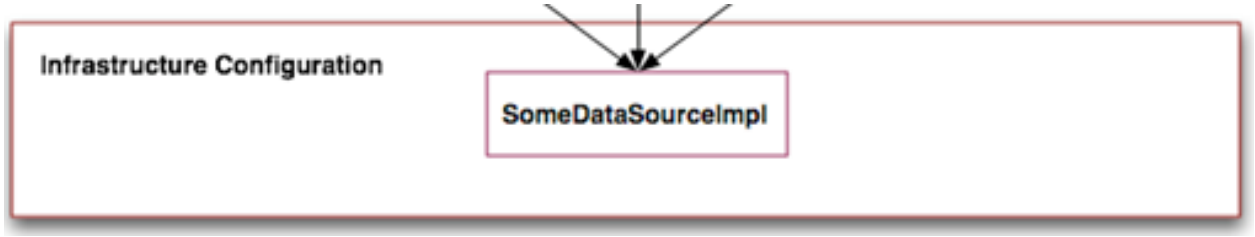
Creando la configuración de la aplicación

- ✓ La figura 1 muestra la configuración dividida en dos categorías: la configuración de la aplicación y la configuración de la infraestructura.
- ✓ Los componentes en la caja de la configuración de la aplicación están desarrollados para ti y maquillados por la lógica de la aplicación.



Creando la configuración de la aplicación

- ✓ Los componentes en la caja de la configuración de la infraestructura no están desarrollados por ti y brindan los servicios de bajo nivel que usa la aplicación.



Creando la configuración de la aplicación

- ✓ En algunos de los siguientes pasos te enfocarás en la parte de la configuración de la aplicación. Y definirás la parte de la infraestructura después.
- ✓ En tu proyecto, encontrarás al familiar RewardNetworkImpl en el paquete rewards.internal.
- ✓ También encontrarás que cada implementación repositorio está basada en JDBC y que necesitan estar en los paquete de dominio rewards.internal.
- ✓ Cada una de estas implementaciones usa el JDBC API para ejecutar sentencias SQL contra un DataSource que es parte de la infraestructura de la aplicación.
- ✓ La implementación del DataSource que usarás no es importante en este momento, pero lo será más adelante.

Paso 1: Creando el archivo de configuración de la aplicación

- ✓ La información de la configuración Spring es típicamente tercerizada del código Java, para quedar particionada entre uno o muchos archivos XML.
- ✓ En este paso puedes crear un archivo XML sencillo que le diga a Spring como se configuran los componentes de tu aplicación. Validarás tu configuración visualmente usando el visualizador gráfico del Spring IDE.
- ✓ Primero, en Eclipse navega hacia los templates del proyecto e ingresa al directorio spring. Verás un archivo llamado bean-definitions.xml. Ábrelo y notarás que es una plantilla simple que puedes usar para crear rápidamente un nuevo archivo de configuración Spring para ti.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springf
        http://www.springframework.org/schema/context http://www.springframework.org/s
        http://www.springframework.org/schema/jdbc http://www.springframework.org/sche

</beans>
```

Figura 1: Plantilla de configuración de contexto de Spring

Paso 1: Creando el archivo de configuración de la aplicación

- ✓ Copia bean-definitions.xml en el paquete src/main/java rewards.internal. Renombra el archivo como application-config.xml para indicar que aquí donde la configuración de tu aplicación va a residir.
- ✓ Ahora, en application-config.xml crea la configuración ilustrada en la caja de 'application-config.xml' que se ve a continuación:

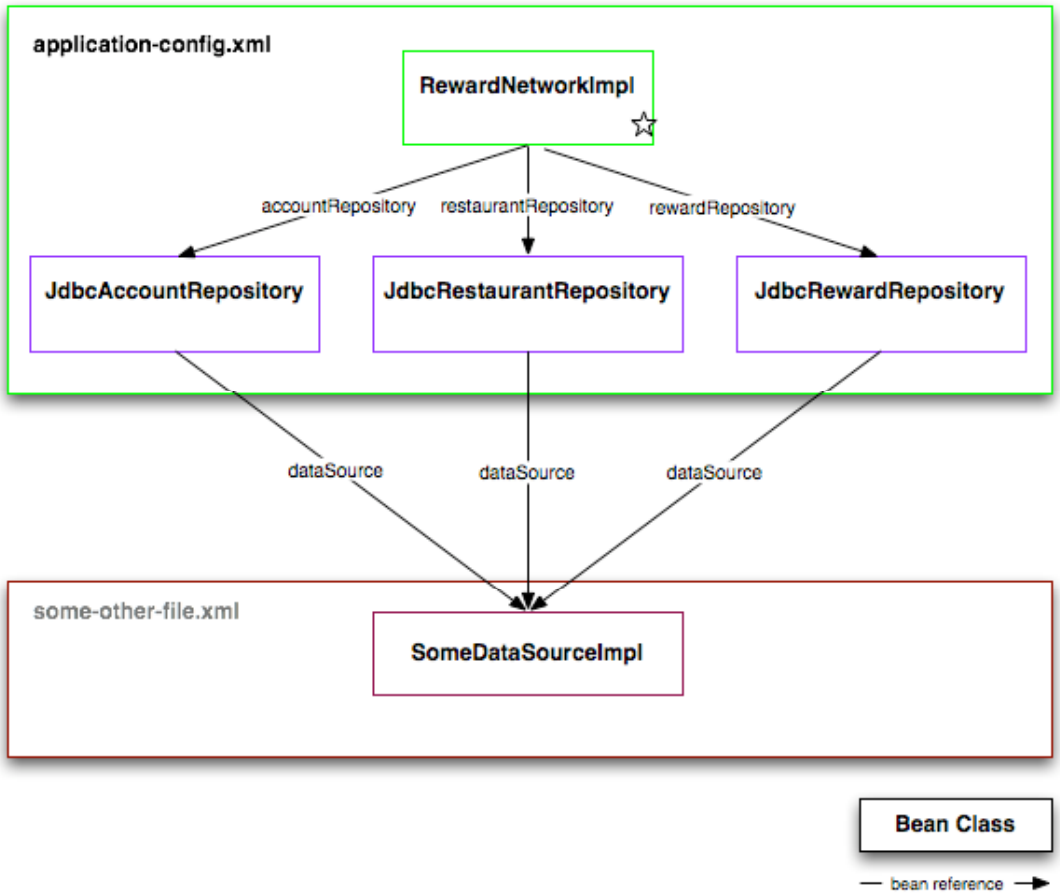


Figura 3: Configuración de la aplicación

Paso 1: Creando el archivo de configuración de la aplicación

- ✓ En la Figura 3, las cajas rectangulares coloreadas representan definiciones de beans, y las flechas etiquetadas entre ellas representan referencias a otros beans. La configuración completa se muestra dividida entre dos archivos application-config.xml y some-other-file.xml.
- ✓ Esto es debido a que generalmente es una buena práctica para separar la configuración de la aplicación y la de la infraestructura, entendiendo típicamente como infraestructura varios entornos continuos. Por ejemplo, en un entorno de pruebas podrías usar un simple data source en memoria, pero en producción deberás usar un pool de conexiones compartidas que se comunican con el servidor de base de datos.
- ✓ Para mover tu infraestructura a otro archivo, puedes cambiarlo sin afectar la configuración de tu aplicación.
- ✓ Por ahora no te preocupes de la configuración de la infraestructura. Ahora, vas a configurar los beans de los repositorios.
- ✓ Si usáramos versiones anteriores de Spring deberíamos hacer las declaraciones como las observamos en la siguiente figura:

Paso 1: Creando el archivo de configuración de la aplicación

```
<bean id="rewardNetwork" class="rewards.internal.RewardNetworkImpl">
</bean>

<bean id="accountRepository" class="rewards.internal.account.JdbcAccountRepository">
</bean>

<bean id="restaurantRepository" class="rewards.internal.restaurant.JdbcRestaurantRepository">
</bean>

<bean id="rewardRepository" class="rewards.internal.reward.JdbcRewardRepository">
</bean>
```

Paso 1: Creando el archivo de configuración de la aplicación

- ✓ Sin embargo, en la versión actual de Spring las cosas han cambiado y solo usaremos una directiva que nos ayudará a obviar toda esta tediosa configuración en el archivo application-config.xml. Esta directiva es <context:component-scan base-package="rewards"/>, la cual le dice a Spring que estas usando la inyección de dependencias basadas en anotaciones, escanea todo el paquete buscando clases que han sido registradas como beans de Spring y con ello elimina los <bean> y la inyección vía elementos <property> en los XML.
- ✓ La directiva mencionada deber quedar así:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/.
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context.
        http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/sprin

    <context:component-scan base-package="rewards" />

</beans>
```

Paso 1: Creando el archivo de configuración de la aplicación

- ✓ De esta manera Spring queda configurado para que automáticamente encuentre los beans y los declare por ti. Ya no se usa <bean>!

- ✓ Pero como es que <context:component-scan> sabe cuales son las clases que debe registrar como beans de Spring? Lo hace buscando clases que se encuentran anotadas con un estereotipo especial de anotaciones, tales como:

  - @Component
  - @Controller
  - @Service
  - @Repository

Paso 1: Creando el archivo de configuración de la aplicación

- ✓ Entonces debemos marcar con alguna de estas anotaciones los cuatro beans que íbamos a declarar el nuestro archivo, de esta forma Spring los auto detectará. Usaremos la anotación @Service y @Repository para marcar los cuatro beans:

RewardNetworkImpl.java | JdbcAccountRepository.java | JdbcRestaurantRepository.java

```
package rewards.internal;

import org.springframework.beans.factory.annotation.Autowired;

/**
 * Rewards an Account for Dining at a Restaurant.
 *
 * The sole Reward Network implementation. This object is an appli
 * the domain-layer to carry out the process of rewarding benefits
 *
 * Said in other words, this class implements the "reward account
 */
@Service
public class RewardNetworkImpl implements RewardNetwork {

    private AccountRepository accountRepository;

    private RestaurantRepository restaurantRepository;

    private RewardRepository rewardRepository;
```

Paso 1: Creando el archivo de configuración de la aplicación

- ✓ Ahora tenemos que relacionar los componentes individuales disponibles en application-config.xml usando la anotación @Autowired. En application-config.xml, ya no usaremos el constructor con argumentos y ni las definiciones de las propiedades de todos los beans. En otras palabras: menos que escribir :D
- ✓ Abre la clase RewardNetworkImpl y anota el constructor con la anotación @Autowired. Esta le dice a Spring que debe intentar encontrar automáticamente los beans que corresponden con los tipos del constructor con argumentos para instanciar el tipo.
- ✓ Ahora abre JdbcRewardRepository y anota DataSource con la misma anotación @Autowired. Esto le dirá a Spring que inyecte vía atributo una instancia de un bean de tipo DataSource.
- ✓ A continuación abre JdbcRestaurantRepository y JdbcAccountRepository, entonces has lo mismo que hiciste con el repository de reward. Ahora has terminado de agregar instrucciones para automáticamente relacionar las dependencias.
- ✓ Una vez que has definido los cuatro beans y los referencias como se muestra en la figura, muévete al siguiente paso!

Paso 2: Visualizando el archivo de configuración de la aplicación

- ✓ Ahora que has definido tus cuatro beans en application-config.xml, cómo puedes verificar que están correctamente definidos? Un camino es usar el Spring IDE para visualizar tu configuración. En el siguiente paso usarás el Spring IDE para graficar los beans que están en application-config.xml.

- ✓ Para habilitar los gráficos del Spring IDE, primero debes decirle a Eclipse que tu proyecto es un proyecto Spring IDE. Para ellos, has clic derecho en el proyecto 02-container-1 y elige Spring Tools -> Add Spring Project Nature.

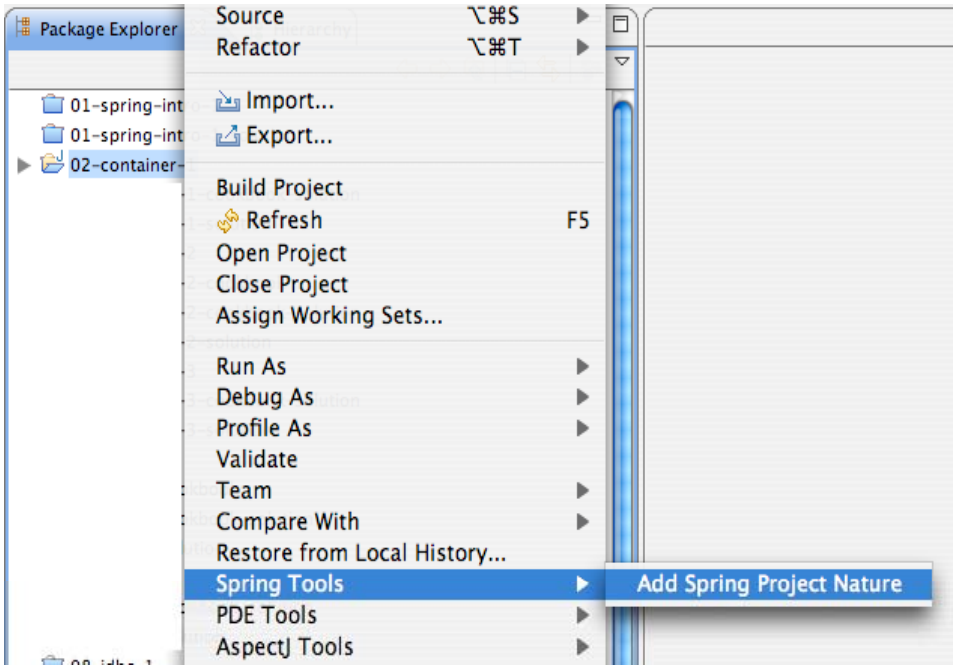


Figura 4: Habilitando tu proyecto como un proyecto Spring IDE

Paso 2: Visualizando el archivo de configuración de la aplicación

- ✓ Ahora verás que tu proyecto tiene un icono "S", la cual es un indicador que es un proyecto Spring IDE:

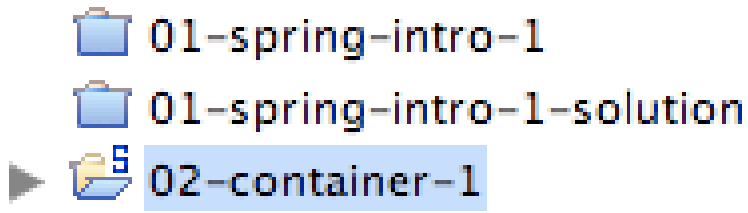


Figura 5: 02-container-1 es un proyecto Spring IDE

Paso 2: Visualizando el archivo de configuración de la aplicación

- ✓ A continuación, necesitas decirle al Spring IDE acerca de los archivos donde has definido tus beans. Para hacer esto, has clic derecho de nuevo sobre el proyecto 02-container-1, elige Properties, y navega hacia el tab Spring -> Beans Support.

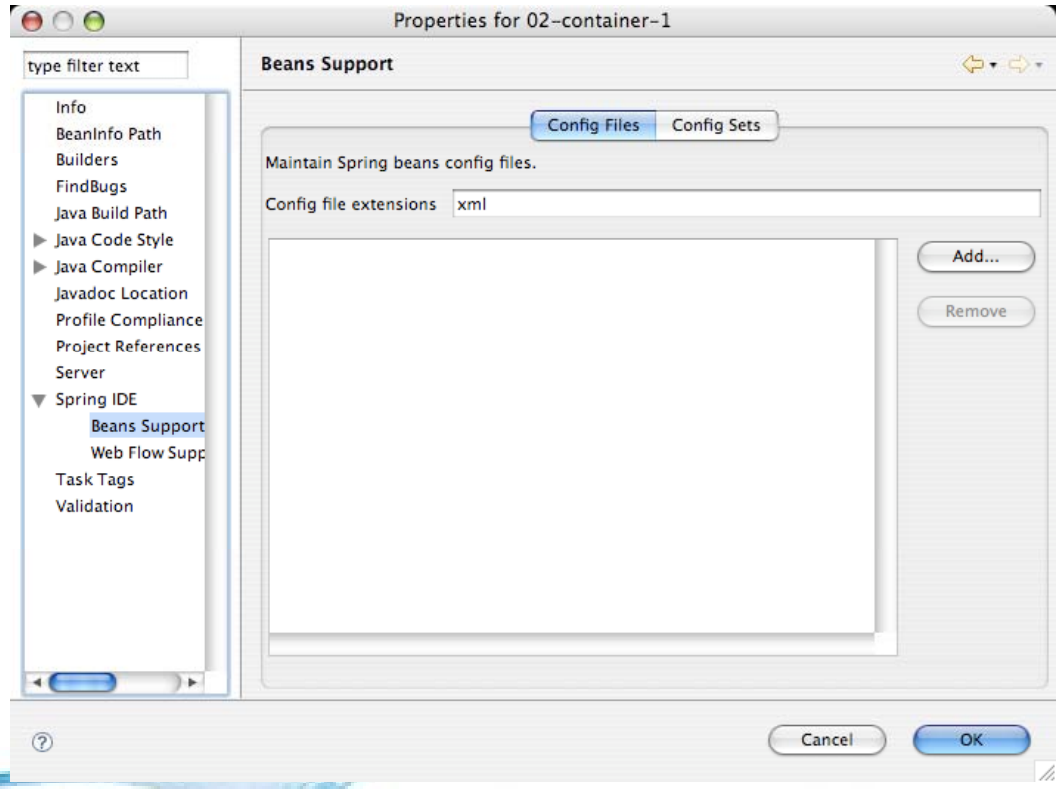


Figura 6: El tab Spring

Paso 2: Visualizando el archivo de configuración de la aplicación

- ✓ Elige Add... y agrega tu application-config.xml de la lista. una vez agregado, elige OK para terminar.
- ✓ Con el archivo agregado, ahora dile al Spring IDE que lo visualice. Para hacer esto, desde la barra de menú elige Window -> Show View -> Other... Expande el nodo de Spring y elige Spring Explorer. Verás la vista Spring Explorer aparecer en la parte inferior del área de trabajo. En esta vista, expande el proyecto 02-container-1, has click derecho sobre application-config.xml y elige Open Graph.

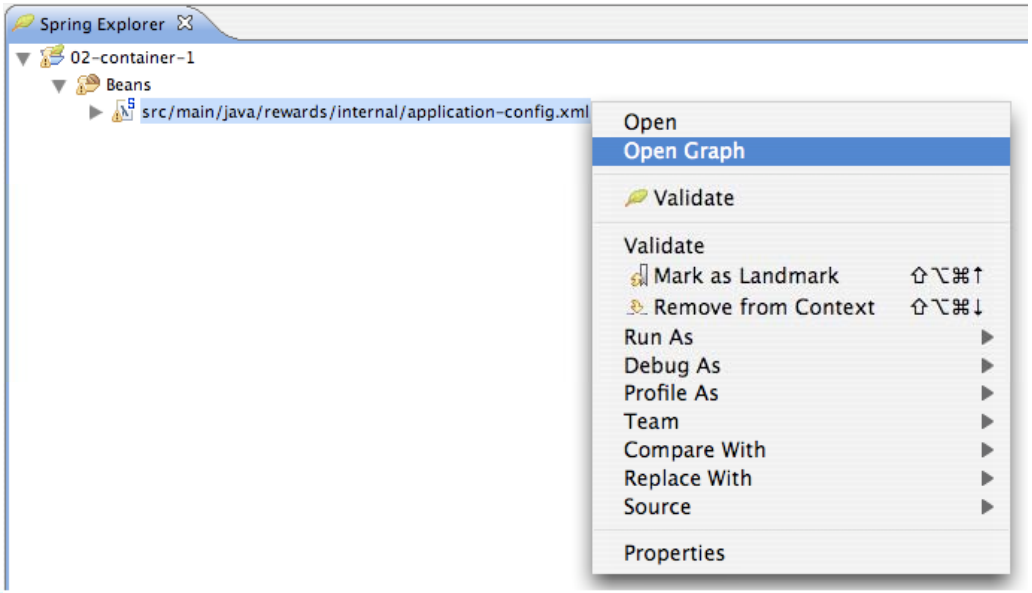


Figura 7: Open Graph

Paso 2: Visualizando el archivo de configuración de la aplicación

- ✓ El resultado debe verse así:

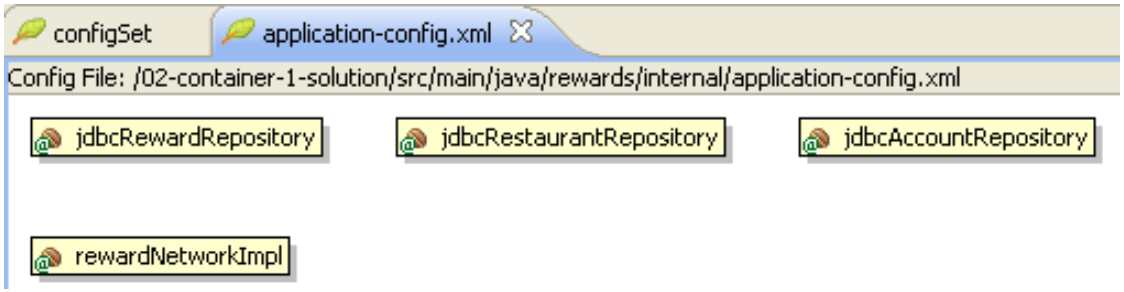


Figure 8: application-config.xml visualizado con Spring IDE

- ✓ Cuando tu veas el equivalente de la Figura 8 en tu gráfico de application-config.xml, sigue con el siguiente paso! Si ves algo diferente, regresa a tu configuración para encontrar el error. Cuando hagas cambios en la configuración, refresca tu gráfico.

Paso 3: Creando la configuración de la infraestructura para pruebas

- ✓ En el paso previo creaste y visualizaste las definiciones de los beans para los componentes de tu aplicación. En este paso crearás la configuración de la infraestructura necesaria para probar tu aplicación. Entonces, podrás visualizar la configuración de pruebas completa.
- ✓ ¿Qué es lo que nos faltaría para probar nuestra aplicación? Recordar que cada repositorio basado en JDBC necesita un DataSource para trabajar. Por ejemplo, JdbcRestaurantRepository necesita un DataSource para cargar objetos Restaurant por su merchant numbers desde las filas que se encuentran en la tabla T_RESTAURANT. Sin embargo, aún no has definido ninguna implementación para DataSource. En este paso definirás el DataSource en el archivo de configuración en el árbol de pruebas.
- ✓ En el folder src/test/java, navega hacia el paquete rewards. ahí encontrarás un archivo llamado test-infrastructure-config.xml. Ábrelo.

test-infrastructure-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring


    <!-- TODO:  Importar application-config.xml -->
    <!--
         Infrastructure configuration to support system testing the rewards ap

         These beans are defined in a separate file to isolate infrastructure
         as infrastructure often varies between environments.  For example, in
         might use a lightweight in-memory DataSource, while in production you
         with a connection pool.
    -->

    <!-- Creates an in-memory "rewards" database populated with test data for
    <!-- TODO: Define a bean named 'dataSource' that configures a test DataSou

</beans>
```

Figura 9: test-infrastructure-config.xml

Paso 3: Creando la configuración de la infraestructura para pruebas

- ✓ Puedes ver un TODO invitándote a completar la codificación. Aquí es donde se define el DataSource que tu aplicación usará para obtener las conexiones a base de datos en un ambiente de pruebas.
- ✓ ¿Qué implementación de DataSource podríamos usar? Seguro quieres algo simple que te permita probar rápidamente tu aplicación dentro de Eclipse. Además, necesitas asegurarte que la base de datos es creada y llenada con datos de pruebas antes que tu aplicación se inicialice. Sino tus pruebas fallarán al no encontrar datos.
- ✓ Afortunadamente, Spring nos da la opción de utilizar una base de datos embebida. La declarativa a usar, encapsula la creación de una implementación sencilla de un DataSource que se conecta al una base de datos Hypersonic (HSQLDB) llenada en memoria con datos de pruebas.

Paso 3: Creando la configuración de la infraestructura para pruebas

- ✓ Esto está bien, pero ¿cómo es que podemos exponer este DataSource como un bean para que sea inyectado en tu aplicación? **Fácil**: simplemente define un bean llamado dataSource. Spring expondrá el DataSource que la fábrica creará automáticamente.

```
<import resource="classpath:rewards/internal/application-config.xml"/>
<!--
    Infrastructure configuration to support system testing the rewards application.
    These beans are defined in a separate file to isolate infrastructure config from application config,
    as infrastructure often varies between environments.  For example, in a test environment you
    might use a lightweight in-memory DataSource, while in production you connect to a database server
    with a connection pool.
-->

<!-- Embedded H2 Database -->
<jdbc:embedded-database id="dataSource" type="H2">
    <jdbc:script location="classpath:rewards/testdb/schema.sql"/>
    <jdbc:script location="classpath:rewards/testdb/test-data.sql"/>
</jdbc:embedded-database>

<!-- Local, JDBC-based TransactionManager -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

Paso 3: Creando la configuración de la infraestructura para pruebas

- ✓ ¿Cómo es que esto trabaja? La clave de todo esto es que se implementa la interface FactoryBean, una interface especial de Spring. Cuando un FactoryBean es creado por Spring, este mira a la fábrica y le pregunta cómo "obtener ese objeto". El objeto retornado, DataSource en este caso, puede ser referenciado por otro beans, mientras que la fábrica se oculta a si misma.

Paso 3: Creando la configuración de la infraestructura para pruebas

- La fábrica de beans es usada para llamar a código Java definido para obtener el bean. Esta característica es muy útil cuando existen paso complicados al obtener un bean que puede ser llamado por constructor y por propiedades setter. Este es el caso de jdbc:embedded-database, donde estos pasos son:
  - Construir la implementación del DataSource.
  - Configurar la implementación del DataSource.
  - Exporta el esquema de base de datos (para crear tablas, etc.)
  - Cargar los datos de prueba
- jdbc:embedded-database encapsula estos pasos y Spring se asegura que ellos corran antes que el DataSource pueda ser referenciado por otros beans. Adicionalmente, debido que el DataSource es requerido por la aplicación, todo esto debe ocurrir antes que cualquier bean de la aplicación sea creado. Spring siempre asegura que los beans sean creados en el orden correcto a sus dependencias.
- Con este conocimiento, ahora a jdbc:embedded-database se le asigna un id dataSource del tipo H2 en test-infrastructure-config.xml.

Paso 3: Creando la configuración de la infraestructura para pruebas

- ✓ Ahora configura la fábrica del bean dataSource con las siguientes propiedades:

Tabla 1. Propiedades de TestDataSourceFactory

Nombre	Valor
schemaLocation	classpath:rewards/testdb/schema.sql
testDataLocation	classpath:rewards/testdb/test-data.sql

- ✓ Usando la configuración de la Tabla 2.1 se crea una implementación sencilla de un DataSource que se conecta en memoria a una base de datos Hypersonic (HSQLDB) llamada 'rewards' definida por schema.sql y llenada con datos contenidos en test-data.sql. DataSource es retornado por la fábrica cuando sea referenciada por otros beans por su nombre dataSource.

  - *Dale una mirada a los archivos schema.sql y test-data.sql en el paquete rewards.testdb para revisar el esquema de base de datos y los datos de pruebas cargados por TestDataSourceFactory.*

  - *Nota como la fábrica encapsula la construcción y el llenado de las pruebas del DataSource de prueba. Por implementar la interface especial FactoryBean, el DataSource creado es espuesto como un bean de Spring automáticamente. La fábrica se oculta asimisma.*

- ✓ **Una vez el bean dataSource se ha definido en test-infrastructure-config.xml con la configuración de la Tabla 2.1, sigue con el siguiente paso!**

Paso 4: Visualizando la configuración de las pruebas

- Más adelante definirás la configuración para tu aplicación y la infraestructura necesaria para el sistema de pruebas de tu aplicación. En este paso usarás el Spring IDE para visualizar la configuración del sistema de pruebas completo dividido en dos archivos.

- Has click derecho sobre tu proyecto 02-container-1 y elige Properties. Elige el tab Spring IDE -> Beans Support y Add... tu test-infrastructure-config.xml a la lista de Config Files.

Paso 4: Visualizando la configuración de las pruebas

- Ahora debes crear un grupo de dos archivos juntos dentro de 'systemTest' en el tab Config Set lógico.
- Para hacer esto, elige el tab Config Sets y elige New...En el dialogo, ingresa el nombre del Config Set, elige los config files para incluirlos, y elige OK:

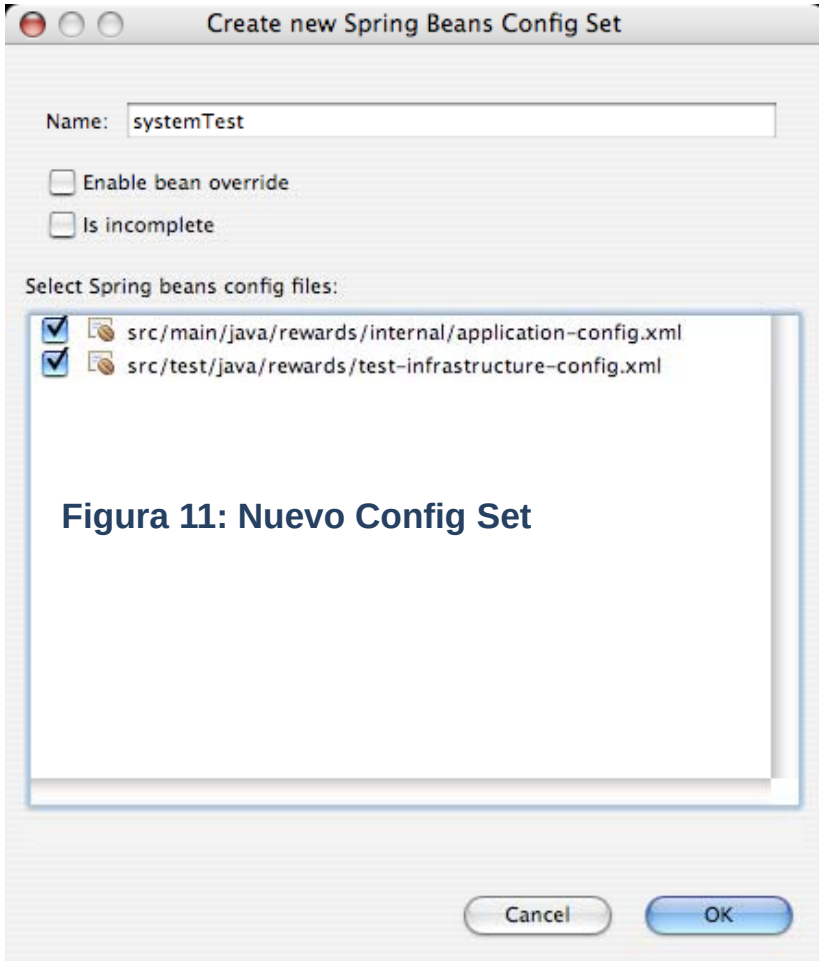


Figura 11: Nuevo Config Set

Paso 4: Visualizando la configuración de las pruebas

- Ahora gráfica tu nuevo 'systemTest' Config Set. En tu view Explorer, has click derecho sobre el set y elige Open Graph. Tu gráfico debe verse así:

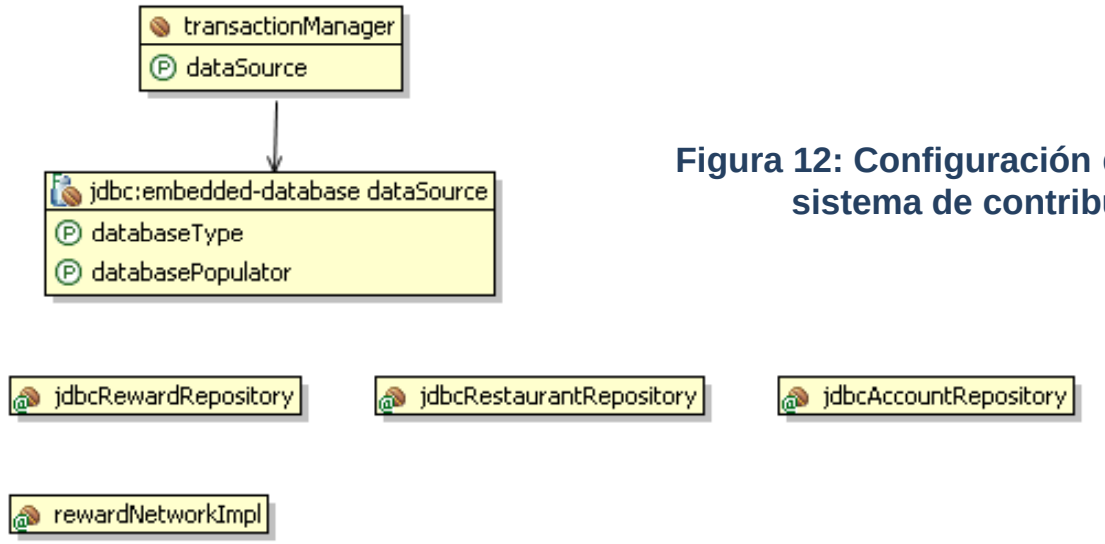


Figura 12: Configuración de pruebas del sistema de contribuciones

- Nota como el bean dataSource es parte del gráfico, como es que el gráfico ahora visualiza la configuración de los archivos application-config.xml y test-infrastructure-config.xml.
- Cuando obtengas un gráfico equivalente al de la Figura 12 en tu application-config.xml, sigue con el siguiente paso!

Probando la aplicación con Spring y JUnit

- ✓ En esta sección final probarás la aplicación de contribuciones con Spring y JUnit.
- ✓ Primero implementarás la lógica de construcción de pruebas para crear un Spring ApplicationContext que "escuche" tu aplicación.
- ✓ Entonces implementarás la lógica de pruebas para ejecutar y verificar el comportamiento de la aplicación.

Paso 5: Creando la clase de pruebas

- Empieza por crear un nuevo JUnit TestCase llamado RewardNetworkTests en el paquete rewards dentro de src/test/java. usa el wizard New -> JUnit Test Case para ayudarte:

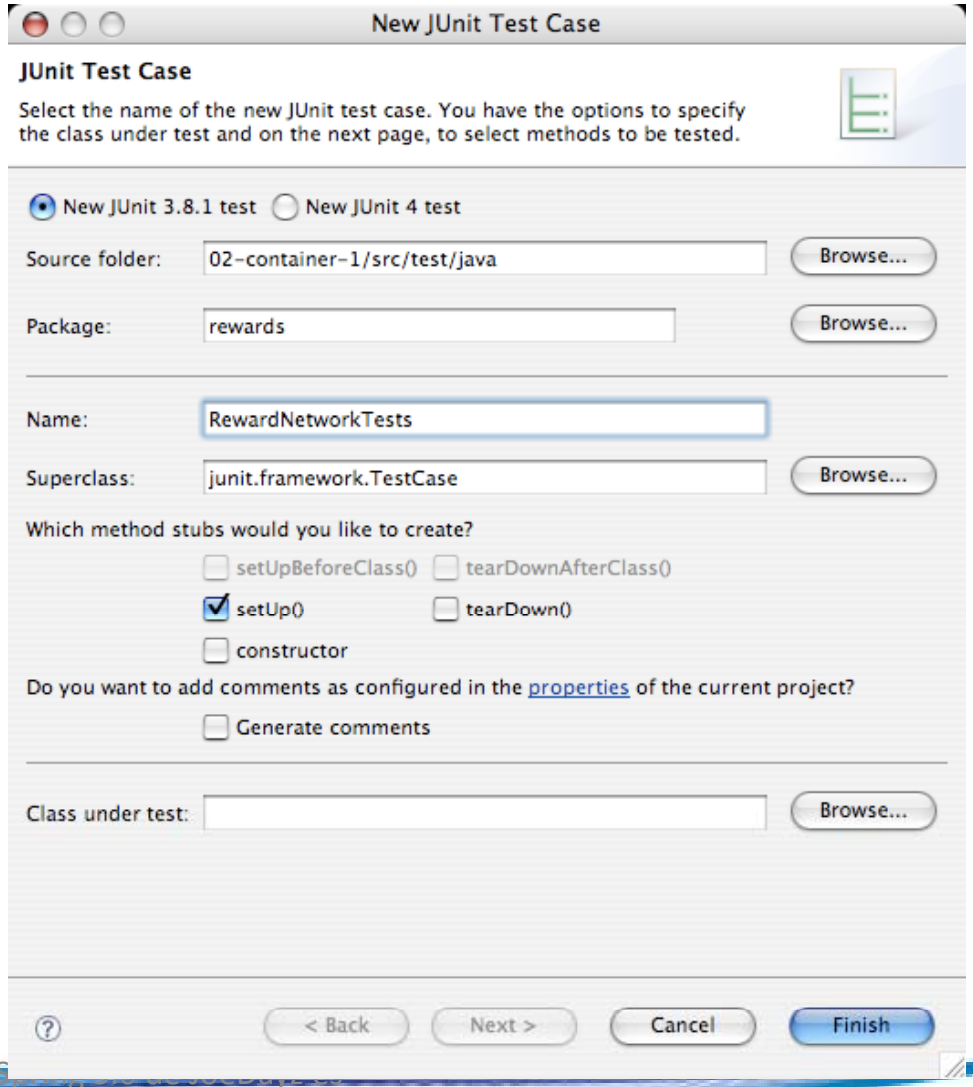


Figura 13: Creando RewardNetworkTests TestCase usando el wizard JUnit Test Case

- Una vez que tu clase RewardNetworkTests ha sido creada, **continúa ya falta poco!**

Paso 6: Implementando la lógica de pruebas

- ✓ En este paso implementarás la lógica necesaria para correr tus pruebas de sistema. Primero debes crear un Spring ApplicationContext que escuchará tu aplicación, luego busca el bean que usarás para invocar a la aplicación.
- ✓ Primero, sobreescribe el método setUp() del TestCase, si es que aún no lo has hecho aún. Para hacer esto, has click derecho en tu editor y elige Source -> Override/Implement methods... (además puedes usar ALT + SHIFT + S, atajo para el 'Source Menu', seguido del mnemonico 'V'
- ✓ Una vez dentro de setUp(), crea un nuevo ClassPathXmlApplicationContext, el que proveerá el camino hacia los archivos application-config.xml y test-infrastructure-config.xml. Esto hará que se oiga la aplicación dado que es Spring quien se encargará de crear, configurar y ensamblar todos los beans definidos en esos dos archivos.
- ✓ Luego pregunta al contexto para obtener el bean rewardNetwork, el cual representa el punto de entrada a la aplicación de contribuciones. Asigna este bean a un atributo privado del tipo RewardNetwork para que lo puedas referenciar desde los métodos de prueba.

Paso 6: Implementando la lógica de pruebas

- *Asegúrate de asignar la referencia hacia del bean rewardNetwork al atributo de tipo **RewardNetwork y NO a RewardNetworkImpl**. Un Spring ApplicationContext encapsula el conocimiento necesario para que la implementación correcta sea elegida cuando se hace referencia a un determinado componente ha sido elegido en entorno. Por el hecho de trabajar con bean a través de interfaces desacoplamos las complejidades y volatilidades de implementaciones.*

- *No le preguntes al contexto por los beans internos de la aplicación. RewardNetwork es el punto de entrada, marca el límite para la aplicación definido por una sencilla interface pública. Preguntarle al contexto por un bean interno como un repositorio o data source es cuestionable.*

Paso 6: Implementando la lógica de pruebas

- ✓ Ahora verifica que Spring pueda crear exitosamente tu aplicación sobre la prueba setUp. Para hacer esto, crea un método publico llamado testRewardForDining(). deja el cuerpo del método vacio por ahora. Luego corre, tu clase de prueba desde Run -> Run As -> JUnit Test de la barra de menú. Luego de correr tu test, deberías de ver una barra verde indicando que setUp corrió sin lanzar ninguna excepción.

- ✓ Sin embargo, vas a obtener una barra roja y esto se debe a que este tipo de test case se usa si se ha utilizado la antigua forma de configuración de beans: usando <bean> en el archivo de configuración. Puesto que con el ApplicationContext se obtienen justamente beans declarados de forma **explicita** en application-config.xml y test-infrastructure-config.xml.

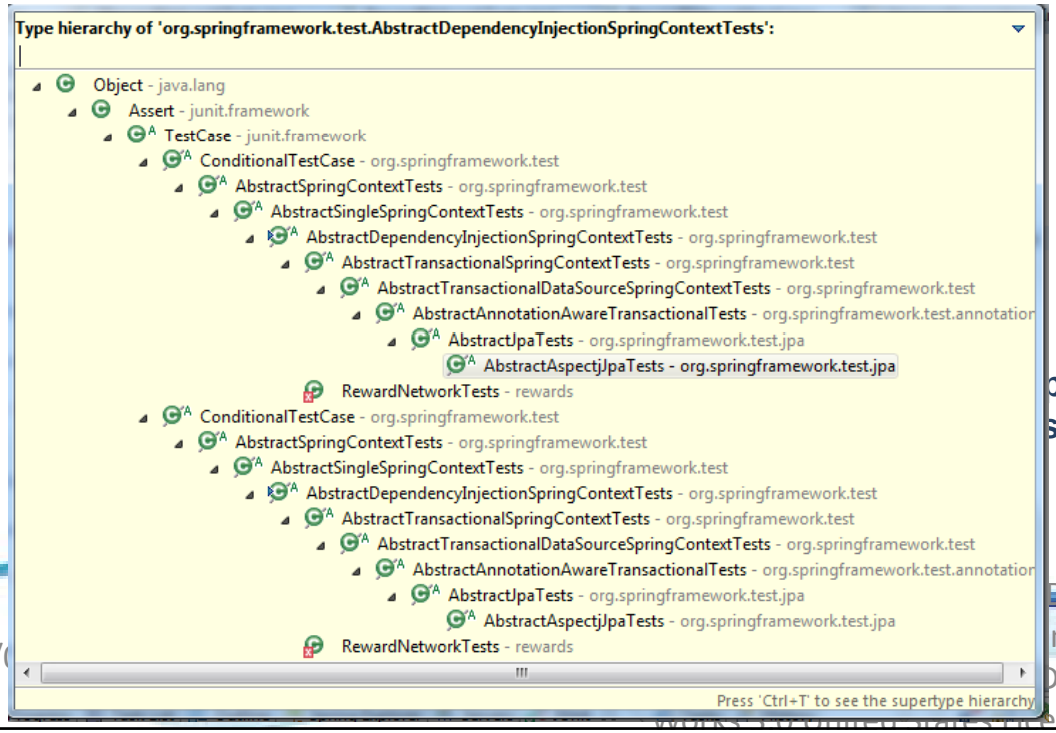
- ✓ Entonces como podríamos hacerlo?

Mejorando la performance de las pruebas del sistema

- ✓ Spring provee clases de soporte para pruebas del sistema. En esta sección, refactorizarás la clase RewardNetworkTests para tomar ventaja de este soporte. Verás como este soporte simplifica la codificación de la configuración de las pruebas, además de mejorar la performance de las pruebas.

Paso 7: Refactorizando para usar AbstractDependencyInjection_ SpringContextTests

- ✓ Una de las clases más útiles en la librería de soporte de pruebas es AbstractDependencyInjectionSpringContextTests. En este paso, aplicarás la herencia de esta clase, y refactorizarás tu prueba para que puedas trabajar con ella.
- ✓ Primero navega hasta esta clase desde el editor de Eclipse. Elige Ctrl+Shift+T y escribe ADISCT para localizarla. Ábrela. Una vez en el editor elige Ctrl+T para ver la jerarquía de los subtipos, ahora si eliges Ctrl+T de nuevo podrás ver la jerarquía de supertipos. Podrás observar que esta clase hereda del TestCase estándar de JUnit:



pendencyInjectionSpringContextTests
stCase estándar de JUnit

Paso 7: Refactorizando para usar AbstractDependencyInjection_SpringContextTests

- ✓ Ahora regresa a tu clase RewardNetworkTests y modifícala haciéndola heredar de AbstractDependencyInjectionSpringContextTests. Asegúrate de usar Ctrl+Space para usar el auto-complete con este nombre tan largo (por ejemplo, tipeando extends ADISCT y presionando Ctrl+Space)
- ✓ Una vez que has hecho la herencia de esta clase de soporte de pruebas de Spring, lo primero que notarás es que tu método setUp no está compilando debido a que esta marcado como final en clase superior en el árbol de jerarquías. Esto está bien. Como podrás ver, ya no requieres este método.
- ✓ Para completar la refactorización necesitas hacer dos cambios. Primero, sobreescribir el método getConfigLocations.

Paso 7: Refactorizando para usar AbstractDependencyInjection_SpringContextTests

- En el cuerpo del método se retorna el mismo String[] que definiste en el método setUp. Borra el original setUp pues no lo necesitaremos más. La superclase automáticamente lo llamará sobreescribiendolo con el método getConfigLocations para crear (y guardar en caché) un ApplicationContext. Esto se puede apreciar gráficamente a continuación:

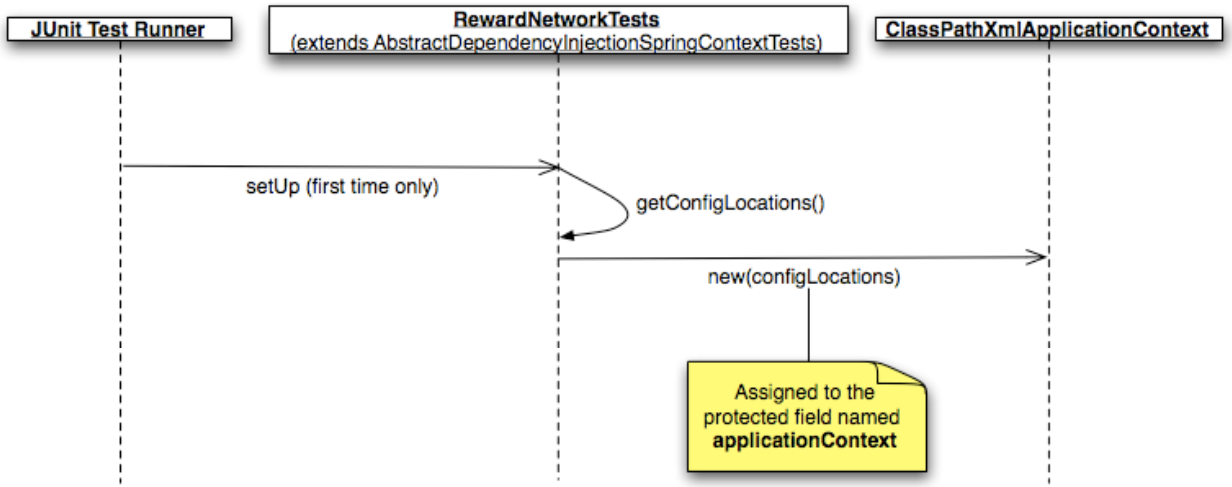


Figura 11: AbstractDependencyInjectionSpringContextTests crea un ApplicationContext en la nueva configuración

Paso 7: Refactorizando para usar AbstractDependencyInjection_SpringContextTests

- ✓ Ahora cuando corras tu prueba la lógica configurada de la superclase usará una forma especial de auto-wiring que busca los atributos que debe inyectar en tu clase de prueba y los llama para asignarles valores desde el ApplicationContext vía autoreflexión. Esto significa que el bean RewardNetwork pasó desde el contexto automáticamente! El autowiring se basa en el tipo. El comportamiento de la configuración de la prueba completa se muestra gráficamente a continuación:

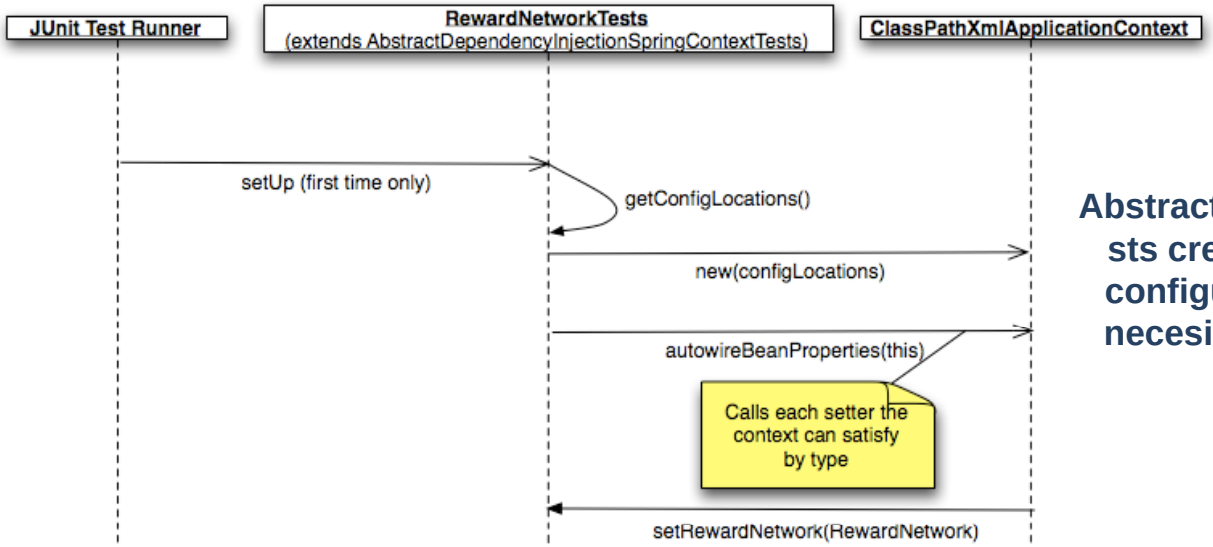


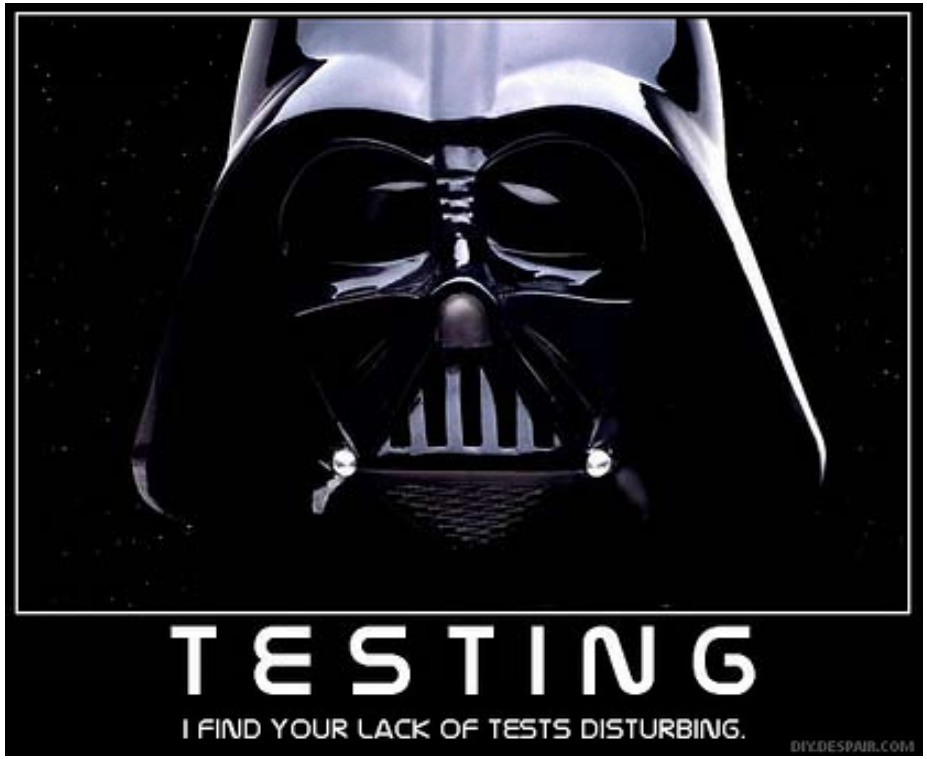
Figura 12: AbstractDependencyInjectionSpringContextTests crea un ApplicationContext en la nueva configuración, después pasa la prueba que necesita desde el contexto por el llamado a setters

Paso 7: Refactorizando para usar AbstractDependencyInjection_SpringContextTests

- ✓ Vuelve a correr tu prueba en Eclipse y verifica que obtengas la barra verde. **Si es así, has asignado apropiadamente el atributo rewardNetwork.** Si no ves la barra verde, intenta averiguar donde esta el problema. Si no logras encontrarlo, pídele al instructor ayuda para solucionar el problema.
- ✓ Cuando hayas obtenido la barra verde, felicitaciones, has entendido e implementado este test case! Has integrado exitosamente una mejora a la aplicación, y al mismo tiempo has simplificado tu sistema de prueba agregando el soporte para pruebas de Spring. Además, la performance de tu sistema ha sido mejorada dado que ahora el ApplicationContext es creado cuando se corre (y se guarda en caché) el test case en lugar de crearlo por cada método.
- ✓ Sin embargo, hay algo que seguramente te está inquietando. Si, el warning *"The type AbstractDependencyInjectionSpringContextTests is deprecated"*. Esto sucede porque en Spring 3 ya "no se ve tan bien" usar AbstractDependencyInjectionSpringContextTests, debido a que hay formas más simplificadas y modernas de realizar las pruebas al sistema sin utilizar herencias.

Usando el soporte de Spring para pruebas con anotaciones

- ✓ En los siguientes pasos veremos como usar el soporte de Spring para realizar pruebas usando anotaciones y JUnit 4.



Paso 8: Refactorizando el test usando JUnit 4

- ✓ JUnit 4 no usa más la herencia para identificar test cases. En su lugar usa anotaciones. Además, no tiene que usar prefijos test en los métodos de pruebas.
- ✓ Abre RewardNetworkTests y elimina la declaración de la super clase (AbstractTransactionDataSourceSpringContextTests). Nota que la clase ya no compila. Esto ha ocasionado la perdida de los métodos de asertos, dejandolos no disponibles. Dejaremos esto para ahora.
- ✓ Lo siguiente es eliminar el método getConfigLocations(). Ahora esencialmente, nos quedamos con una clase Java que no compila más. Ahora trabajaremos sobre el sistema de pruebas de JUnit4.
- ✓ Anota el método testRewardForDining con @Test. Esta anotación marca el método como método test.
- ✓ Como esto es un sistema de prueba, necesitamos decirle que cargue archivo de configuración de Spring. Podemos hacer esto agregando la anotación @ContextConfiguration. Anota la clase con lo siguiente: @ContextConfiguration(locations={"classpath:rewards/test-infrastructure-config.xml"}).

Paso 8: Refactorizando el test usando JUnit 4

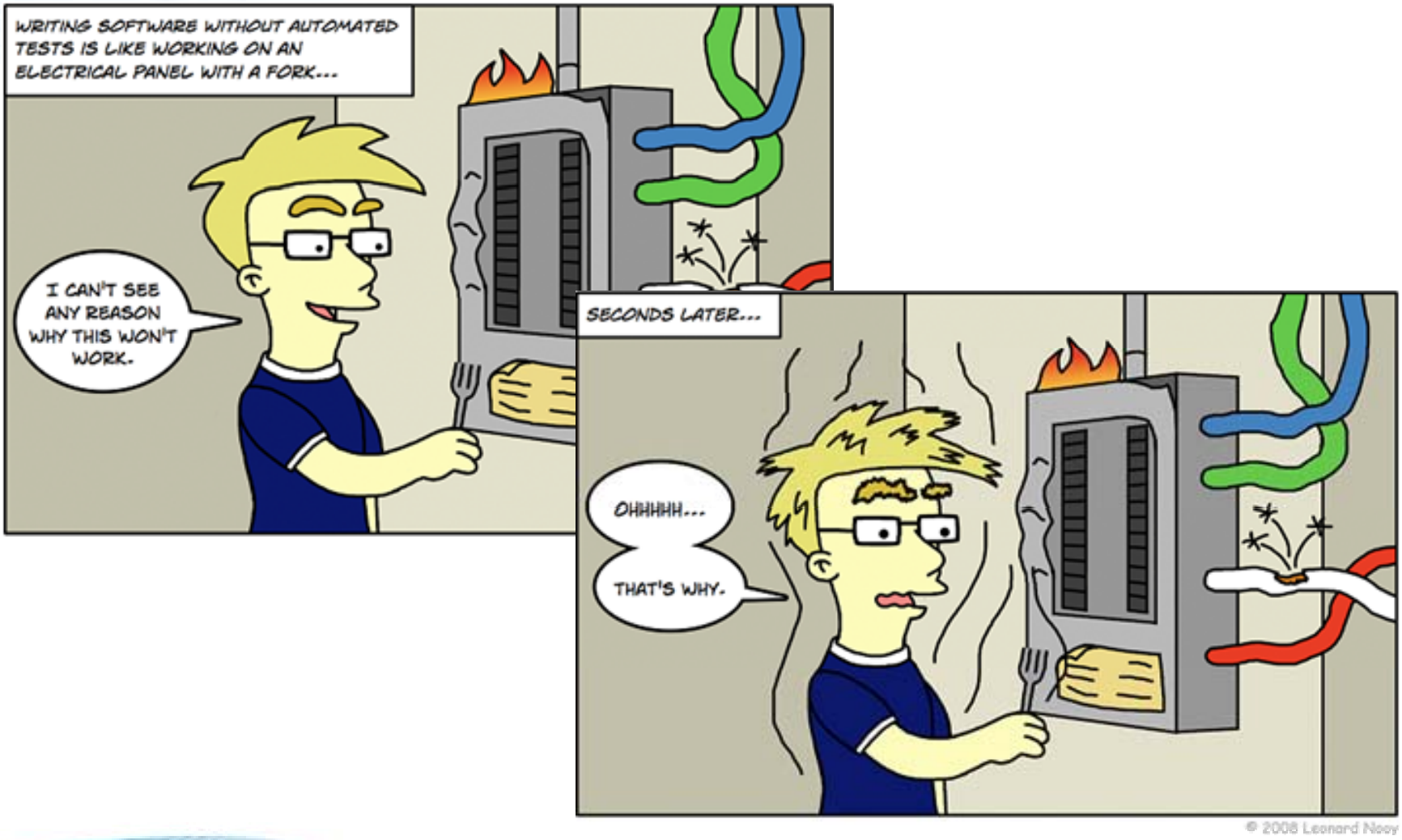
- ✓ A continuación, necesitamos decirle a JUnit que es un test de integración de Spring. Podemos usar la notación @RunWith. Entonces, adicionalmente a la anotación @ContextConfiguration, también anota @RunWith(SpringJUnit4ClassRunner.class).

- ✓ Nota que la clase aun no compila. Algunos de los assertos al parecer no están disponibles. Usaremos imports estáticos (una característica de Java 5) para resolver esto. Esto se realiza de la siguiente manera import static org.junit.Assert.*;. Ahora el test debe compilar.

- ✓ Intenta correrlo de nuevo. Ahora el test debe correr exitosamente.

- ✓ Cuando obtengas la barra verde, felicitaciones! Has completado este laboratorio! Has usado Spring para configurar componentes de una aplicación real y has ejercitado el comportamiento de la aplicación en un entorno de pruebas dentro de tu IDE...de tres formas diferentes! :D

Finalmente…recuerda…



Enjoy it!