Séquence injection de dépendances

istic

Objectifs

- Être capable de décrire le but et le fonctionnement de l'injection de dépendances
- Être capable d'employer le framework Spring à cet usage

istic

Justification

- Comment créez-vous les différents objets qui constituent votre application en cours d'exécution ?
- Très probablement par une ribambelle de new
- Parfois par emploi de patrons de création (Factory Method, Abstract Factory, Singleton, …)

istic

Configuration traditionnelle

Un fragment de code instancie les objets et les connecte

- Généralement dans des constructeurs

Appel à new ou beaucoup mieux à des opérations de fabrication

Ensuite connexion des objets créés

- par des paramètres de constructeur
- par des get/set

Limites de ce système

La préoccupation « configuration » est mêlée au code des différents objets

Dans le cas où on utilise un objet configurateur

- le procédé de configuration peut être délicat à coder
- difficile à réutiliser pour des variantes car trop dépendant des propriétés des éléments du système

Solution

Externaliser la configuration

Comporte les deux étapes

- création des objets
- connexion des objets

Mécanisme déclaratif de la configuration

Mécanisme permettant à un objet de déclarer des dépendances de façon bien séparée

Que sont les dépendances ?

Pour un objet au sens Java

des références vers d'autres objets

Pour un objet au sens UML

des associations vers d'autres objets

Pour un composant logiciel

des ports

Exemple de dépendance Java

```
class Afficheur {
  private Compteur cpt;
  public  Afficheur(Compteur c) {
    this.cpt = c;
  }
}
```

Dépendance

Point d'injection

Principe général de configuration

Un fichier externe

contient des déclarations d'instances à créer

des valeurs des dépendances

Plusieurs formats possibles

fichier XML : vraie séparation des préoccupations configuration/exécution

annotations Java

Exemple 1

- Une classe Compteur
- Une classe Afficheur qui observe un compteur
- On crée une instance de chaque et on les associe

istic

Exemple en Spring (1)

```
<beans>
<bean name="monAfficheur"
  class="Afficheur">
   <constructor-arg>
      <ref bean="monCompteur" />
   </constructor-arg>
</bean>
```

Déclaration d'une instance

Déclaration d'une association

Injection via le constructeur

istic

Exemple en Spring (2)

```
<bean name="monCompteur"
  class="Compteur">
   <constructor-arg>
      <ref bean="monAfficheur" />
   </constructor-arg>
</bean>
</beans>
```

istic

Problème dans l'exemple montré

Dépendance circulaire entre constructeurs

Afficheur(Compteur c)

Compteur(Afficheur a)

- Spring lance alors : BeanCurrentlyInCreationException

Solution

utiliser des set/get (propriétés)

Variante de l'exemple (emploi de propriétés)

```
class Afficheur {
  private Compteur cpt;
  public Compteur setCpt(Compteur c) {
    this.cpt = c;
  }
}
```


Suite variante

```
<beans>
<bean name="monAfficheur"
  class="Afficheur">
  <property name="cpt">
    <ref bean="monCompteur" />
  </property>
</bean>
```

Suite variante

```
<bean name="monCompteur"
  class="Compteur">
   <property id="afficheur"
   ref="monAfficheur" />
</bean>
</beans>
```


Définition d'autres propriétés

Le mécanisme d'injection de dépendances permet de définir des valeurs autres que des références à des objets

par exemple des constantes de configuration, etc

Exemple avec Properties

La classe java.util.Properties permet de stocker des valeurs de configuration

```
Object setProperty(String key, String value);
String getProperty(String key);
void storeToXML(OutputStream os, String
  info);
void loadFromXML(InputStream is);
```

Exemple avec Properties (2)

```
<bean name="unBean" class="master2.SuperBean">
  <property id="lesParametres">
    <props>
    <prop key="truc">Un gros truc</prop>
    </props>
  </property>
</bean>
```


Exemple avec Properties (3)

```
package master2;
import java.util.Properties;
class SuperBean {
   private Properties lesParametres;
   // setLesParametres et getLesParametres ...
}
```


Activation de la gestion de l'injection

- ApplicationContext beanFactory = new ClassPathXmlApplicationContext(
- "test_configuration.xml");
- // Accès aux instances créées
- c = (Compteur) beanFactory.getBean("compteur");

Le fichier de configuration

- <bean name="afficheur" class="Afficheur">
- <property name="cpt" ref="compteur"></property>
- <property name="seuils"><util:map><entry key="mini" value="3" /></util:map>
- </property>
- </bean>
- <bean name="compteur" class="Compteur">
- <property name="aff" ref="afficheur"></property></bean>

istic

En conclusion

- Spring propose un modèle de « micro composant »
- Spring offre des moyens de gérer le cycle de vie
  - possibilité de déclarer des destroy-method

istic