Úvod

Jste v roli Spring konzultanta a zákazník vás požádal o refaktoring rozpracované bankovní aplikace. Od refaktoringu si slibuje lepší udržovatelnost a testovatelnost.

Aplikace je tvořena 3 samostatnými projekty:

- SpringDemo-Banking – jádro bankovní aplikace
- SpringDemo-Commons – společná rozhraní
- SpringDemo-Server – služby poskytované vzdáleným systémem 3. strany

Aplikace zatím nemá hotové žádné uživatelské rozhraní. To nám však nebrání ve vývoji, místo stylu práce „kóduju-nasadím-proklikám“ píšeme automatizované testy. Těmi ověřujeme že aplikace dělá to co má a zároveň můžeme odhalit možné regrese během refaktoringu.

Jediná stávající funcionalita spočívá v přidání nového zákazníka do databáze a vylistování všech stávajících zákazníku.

Design aplikace

Srdcem je rozhraní *BankService* poskytující business služby *getAllCustomers()* a *createNewCustomer()*. Toto rozhraní má jedinou implementaci *BankServiceImpl.* Samotná *BankServiceImpl* neumí přistupovat k databází, ale tuto činnost deleguje na třídu *PureJdbcCustomerDaoImpl*.

Ověření správné funčnosti dosavadní aplikace

Rozbalte všechny 3 projekty a otevřete ve vývojovém prostředí NetBeans. Klikněte pravým tlačítkem na projekt SpringDemo-Commons, zvolte Custom → Goals, do dialogu vyplňte políčko Goals jako *install* a potvďte.

Nyní otevřete třídu *BankServiceTest*, klikněte na ni pravým tlačítkem a zvolte *Test File*. Tester by se měl rozsvítit zeleně. Pokud ne, kontaktujte lektora.

Úkol č. 1 – Uvolňujeme vazby

BankServiceImpl je závislá na třídě *PureJdbcCustomerDaoImpl*. Našim prvním úkolem je tuto vazbu rozbít.

Nápověda pro implementaci:

Springová beana se v XML deskriptoru definuje takto:

<bean id=”myFooBean” class=”xx.yy.Foo” />

tím se vytvoří instance *xx.yy.Foo* jako beana pojmenovaná myBean.

Pokud mám třídu *xx.yy.Bar*, která má metodu *setFoo(Foo foo)*, tak beanu typu *Bar* zhotovím následujícím zápisem v XML:

<bean id=”myBarBean” class=”xx.yy.Bar” >

<property name=”foo” ref=”myFooBean”/>

</bean>

Spring vytvoří instanci třídy *xx.yy.Bar* a po vytvoření do ní přes setter injektne beanu

pojmenovanou *myFooBean.*

Postup implementace:

Rozbití vazby se skládá z těchto kroků:

1. Místo aby si *BankServiceImpl* sama vytvářela instanci PureJdbcCustomerDaoImpl při každém voláni business metody, tak v souboru *applicationContext.xml* definujeme beanu typu *PureJdbcCustomerDaoImpl* s ID *customerDao*.
2. Do *BankServiceImpl* připišeme setter pro *PureJdbcCustomerDaoImpl* a upravíme *applicationContext.xml* tak, aby při vytvoření beany *BankService* zavolal tento setter a vložil skrze něj beanu *customerDao.*
3. Z *PureJdbcCustomerDaoImpl* extrahujeme rozhraní *CustomerDao*, které bude obsahovat metody *findAll()* a *save()*. Signatury metod ponechte stejné jako jsou *PureJdbcCustomerDaoImpl.*
4. Upravte *PureJdbcCustomerDaoImpl* tak, že implementuje rozhraní *CustomerDao.* Za název třídy stačí připsat *implements CustomerDao* a provést příslušný import.
5. Upravte *BankServiceImpl* tak, že všechny výskyty třídy *PureJdbcCustomerDaoImpl* nahradíte rozhraním *CustomerDao.*
6. Nyní je možno z *BankServiceImpl* zrušit setter na *dataSource* a odstranit příslušnou property z *applicationContext.xml*
7. Spusťte testy z *BankServiceTest.* Pokud jste postupovali správně, tak by měly procházet. Pokud ne, kontaktujte lektora.

Pokud jste úspěšně došli až sem, tak gratuluji. Právě jste odstranili závislost třídy *BookServiceImpl* na konkretní implementaci přístupu k databázi. Nyní nám nic nebrání použít jinou implementaci a to pouze změnou konfiguračního souboru.

Úkol č. 2 – Použití JdbcTemplate

Protože *BookServiceImpl* nyní nezávísí na *PureJdbcCustomerDaoImpl*, tak si můžeme dovolit napsat alternativní implementaci rozhraní *CustomerDao.* Využijeme k tomu springovskou třídu *JdbcTemplate*, která nám výrazně usnadní práci s databází.

Nápověda při implementaci:

- Dokumentace JdbcTemplate na http://static.springsource.org/spring/docs/3.0.x/javadoc-api/
- JdbcTemplate potřebuje setnout beanu typu *dataSource*.

- Kostra implementace metody findAll() :

```
List<Customer> customerList =
jdbcTemplate.query(SQL_DOTAZ_STEJNY_JAKO_V_PUVODNI_IMPLEMENTACI,
        new BeanPropertyRowMapper<Customer>(Customer.class));
```

- Kostra metody save:

```
KeyHolder keyHolder = new GeneratedKeyHolder();
    jdbcTemplate.update(new PreparedStatementCreator() {
        public PreparedStatement createPreparedStatement(Connection connection) throws
```

```
SQLException {
        PreparedStatement stmt = connection.prepareStatement(SQL_PRO_INSERT,
            Statement.RETURN_GENERATED_KEYS);
        stmt.setString(1, customer.getFirstname());
        stmt.setString(2, customer.getLastname());
        return stmt;
      }
  }, keyHolder);
  customer.setId(keyHolder.getKey().longValue());
```

Postup implementace:

1. Vytvořte novou třídu *JdbcTemplateCustomerDaoImpl*, která implementuje rozhraní *CustomerDao.*
2. Tato třída bude mít private field typu JdbcTemplate a k němu přislušný setter.
3. Implementujte obě metody rozhraní podle výše uvedené nápovědy
4. V *applicationContext.xml* vytvořte beanu typu *JdbcTemplateCustomerDaoImpl* a pojmenujte ji *springJdbcCustomerDao*
5. Dále upravte *applicationContext.xml* tak, aby beaně bookService byla setnuta beana *springJdbcCustomerDao* místo dosavadní *customerDao.*
6. Můžete z *applicationContext.xml* zrušit definici beany *customerDao*.
7. Spusťte testy, opět by měli procházet.

Nyní srovnejte třídy *JdbcTemplateCustomerDaoImpl* a *PureJdbcCustomerDaoImpl*. Funkčně jsou totožné, ale Springová varianta je výrazně úspornější a méně náchylná k chybám.

Úkol č. 3 – AoP

Prohlédněte si třídu *BankServiceLoggingAspect.* Jedná se o aspekt, který loguje všechna volání metody *createNewCustomer()* z *BankService* a zapisuje je na standardní výstup. Výhodou tohoto řešení je, že logovací kód je oddělen od samotné business logiky.

Vytvořte v tomto aspektu novou metodu, která bude logovat volání *getAllCustomers()* z BankService. Spusťe test a sledujte standardní výstup.

Úkol č. 4 – Separation of Concerns

BankServiceLoggingAspect nyní loguje volání obou business metod. Ale umí logovat jen na standardní výstup. Od zákazníka přišel požadavek na flexibilnější logovací mechanismus. Upravte tedy *BankServiceLoggingAspect* tak, aby dostaval setterem instanci třídy implementující rozhraní *Notifier* a místo zápisu zprávy na standardní výstup volal metodu *notify* tohoto rozhraní. Následně upravte *applicationContext.xml,* tak aby do *BankServiceLoggingAspect* setnul instanci

SysOutNotifier, která toto rozhraní implementuje. Následně spusťte testy a sledujte standardní výstup, zda dochází k logování.

Úkol č. 5 – Remoting

Od zákazníka přišel další požadavek – volání business metod je nyní třeba logovat do vzdáleného systému auditora. Systémy auditor podporují vzdálené volání přes rozhraní Hessian. Upravte tedy aplikaci tak, aby místo posílání logu na standardní výstup volala službu Notify na vzdáleném serveru.

Jako vzdálený systém nám bude sloužit projekt SpringDemo-server. V souboru *remoting-servlet.xml* vytvořte beanu typu *SysOutNotifier,* pojmenujte ji *notifier* a následně pomocí tohoto kusu XML vystavte pro vzdálený přístup:

```
<bean name="/NotifierService"
class="org.springframework.remoting.caucho.HessianServiceExporter">
  <property name="service" ref="notifier"/>
  <property name="serviceInterface"
value="eu.ibacz.swsc.spring.commons.springdemocommons.Notifier"/>
</bean>
```

Z příkazové řadky z adresáře SpringDemo-server spusťte *mvn jetty:run* a ověřte že poslední řádka končí *[INFO] Started Jetty Server*

V projektu SpringDemo-banking upravte *applicationContext.xml*, tak aby Spring do *BankServiceLoggingAspect* setoval proxy pro vzdálené volání služby přes protokol Hessian.

Vytvoření proxy pro vzdálený přístup ke službě:

```
<bean id="remoteNotifier"
class="org.springframework.remoting.caucho.HessianProxyFactoryBean">
        <property name="serviceUrl" value="http://localhost:8080/SpringDemo-
Server/remoting/NotifierService" />
        <property name="serviceInterface"
value="eu.ibacz.swsc.spring.commons.springdemocommons.Notifier"/>
    </bean>
```

Spusťte testy a ověřte že do konzole s jetty probíhá logování business akcí.

Úkol č. 6 – Odstranění konfiguračních parametrů z XML

Zákazník nyní požaduje flexibilnější konfiguraci. Není akceptovatelné, aby URL pro vzdálenou službu bylo zadrátované natvrdo v XML. Použijte *PropertyPlaceholderConfigurer* a vytáhněte toto URL do properties souboru. Viz http://www.mkyong.com/spring/spring-propertyplaceholderconfigurer-example/