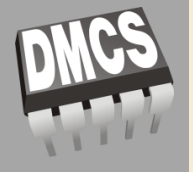


SPRING FRAMEWORK

Katedra Mikroelektroniki i Technik Informatycznych
Politechniki Łódzkiej
ul. Wólczanska 221/223 budynek B18,
90-924 Łódź

dr inż. Jakub Chłapiński

3. Spring Data Access

Zarządzanie transakcjami

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Spring umożliwia zarządzanie transakcjami w oparciu o spójny i jednolity model z wykorzystaniem różnych implementacji (np. JTA, JDBC, Hibernate, JPA, JDO)
- Możliwe jest deklaratywne definiowanie kodu transakcyjnego
- Transakcyjność opiera się w Spring na implementacjach interfejsu PlatformTransactionManger

```
public interface PlatformTransactionManager {

  TransactionStatus getTransaction(TransactionDefinition definition)
    throws TransactionException;

  void commit(TransactionStatus status) throws TransactionException;

  void rollback(TransactionStatus status) throws TransactionException;

}
```

Zarządzanie transakcjami (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Interfejs TransactionDefinition zawiera metody określające:
  - Poziom izolacji (isolation)
  - Propagację (propagation)
  - Maksymalny czas trwania (timeout)
  - Czy transakcja tylko odczytuje dane (read-only)
- Interfejs TransactionStatus określa stan transakcji

Zarządzanie transakcjami (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- W zależności od sposobu dostępu do danych stosuje się różne implementacje PlatformTransactionManager
  - DataSourceTransactionManager dla JDBC
  - JtaTransactionManager dla JTA
  - HibernateTransactionManager dla Hibernate
  - itd.

Deklaratywne definiowanie transakcji

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przykład deklaracji w pliku XML

```
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="*"/>
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut id="metodaSerwisowa"
     expression="execution(* example.service.*Service.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="metodaSerwisowa"/>
</aop:config>

<bean id="txManager"
  class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
  destroy-method="close">
  ...
</bean>
```

Deklaratywne definiowanie transakcji (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

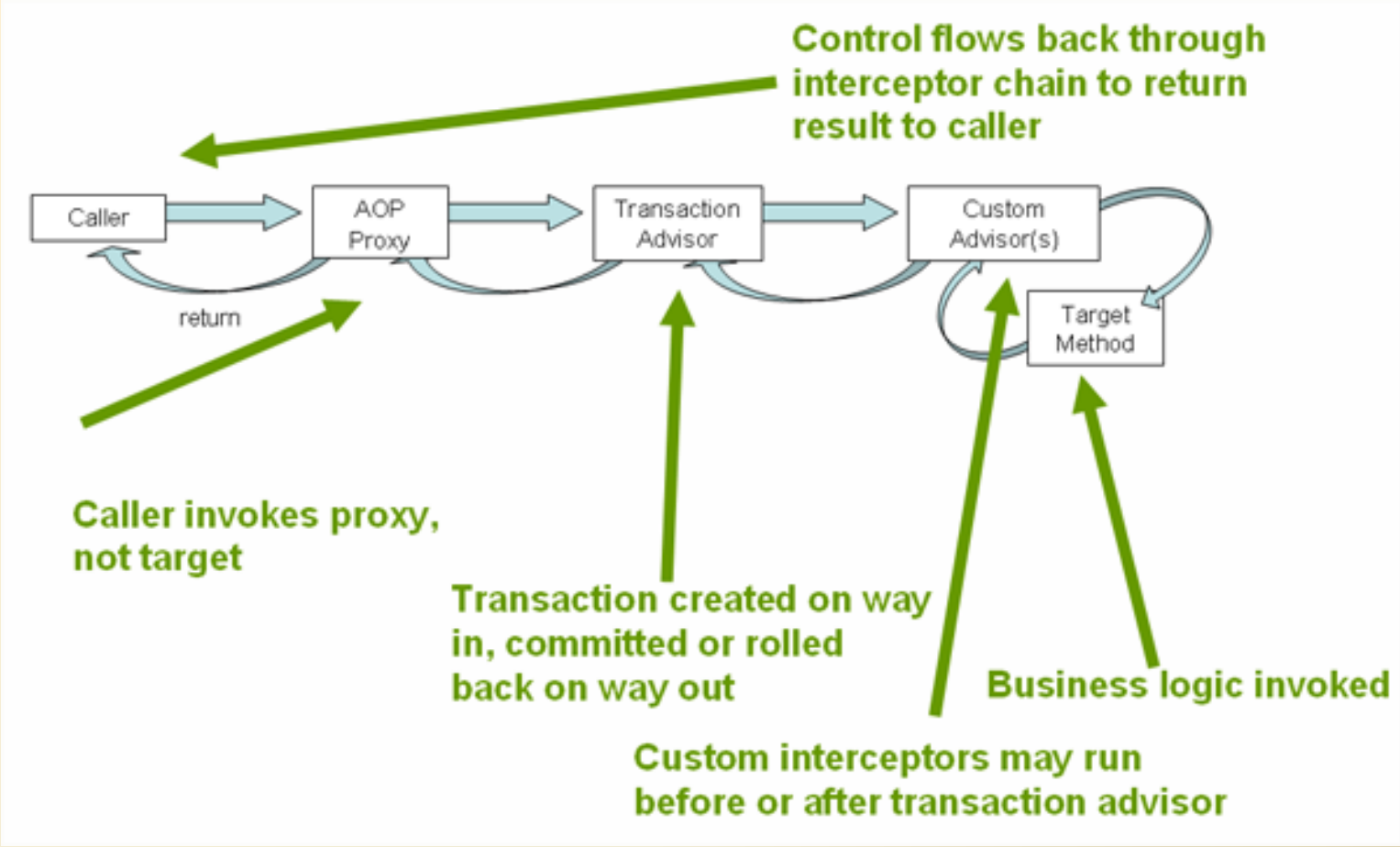
- Przykład deklaracji przy użyciu @Transactional

```
<tx:annotation-driven transaction-manager="txManager"/>
```

```
@Transactional(propagation=Propagation.REQUIRED)
public class OsobaServiceImpl implements OsobaService {

  @Transactional(readOnly=true)
  public Osoba get(Long id);

  public void save(Osoba osoba);

}
```

Mechanizm transakcji

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl



Propagacja transakcji

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Required - transakcja jest wymagana dla wykonania metody, jeżeli metody transakcyjne wywołują się wzajemnie, cały kod będzie wykonany w obrębie tej samej transakcji.
- RequiresNew - dla wykonania danej metody konieczne jest uruchomienie nowej transakcji
- Nested - wykorzystanie jednej transakcji z wieloma punktami SavePoint (tylko dla JDBC) do których transakcja może się cofać

Data Access Object (DAO)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Spring posiada jednolity i spójny model dla obiektów DAO, pozwalający na łączenie w jednej aplikacji wielu heterogenicznych źródeł danych
- Wyjątki wykorzystywane przez poszczególne technologie dostępu do danych zebrane zostały w spójną hierarchię
- Dla poszczególnych rodzajów źródeł danych przygotowane zostały klasy (np. HibernateDaoTemplate) ale ich stosowanie nie jest konieczne
- Obecnie Spring opakowuje wyjątki HibernateException

Data Access Object (DAO) (2)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przykładowa implementacja klasy DAO dla Hibernate

```
@Repository
public class ProductDaoImpl implements ProductDao {

  @Autowired
  private SessionFactory sessionFactory;

  public Collection loadProductsByCategory(String category) {
    return this.sessionFactory.getCurrentSession()
      .createQuery("from test.Product product where product.category=?")
      .setParameter(0, category)
      .list();
  }

  public void setSessionFactory(SessionFactory sessionFactory) {
    this.sessionFactory = sessionFactory;
  }
}
```

Data Access Object (DAO) (3)

dr inż. Jakub Chłapiński, jchlapi@dmcs.pl

- Przykładowa konfiguracja Spring dla Hibernate

```
<bean id="dataSource"
  class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
  <property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/>
  <property name="username" value="sa"/>
  <property name="password" value=""/>
</bean>

<bean id="sessionFactory"
  class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
  <property name="dataSource">
    <ref bean="dataSource"/>
  </property>
  <property name="annotatedPackages">
    <list>
      <value>example.domain</value>
    </list>
  </property>
</bean>

<bean id="txManager"
  class="org.springframework.orm.hibernate.HibernateTransactionManager">
  <property name="sessionFactory" ref="sessionFactory" />
</bean>
```