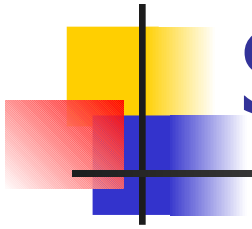
Corso di “Sviluppo di applicazioni Web”

Docente: Giovanni Grasso

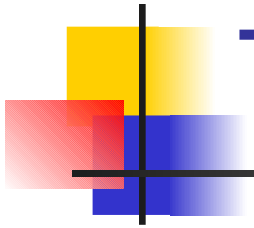
- Transazionalità in Spring

Pluggable Transaction Strategy



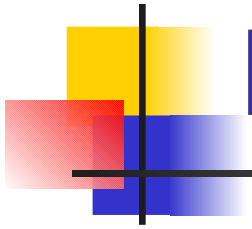
- Spring PlatformTransactionManager
- Transaction management provided by implemenation
- Implemenations include
  - JDBC
  - Hibernate
  - JTA
  - Others

Creating a TransactionManager



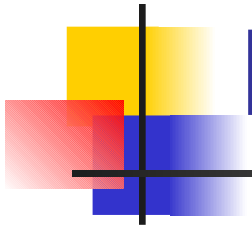
- Each implementation will require a different set of properties to configure
  - For example, the HibernateTransactionManager requires a reference to a Session Factory

```
<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```



Programmatic Transactions

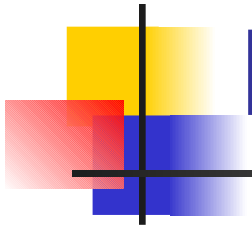
- TransactionTemplate
- Works just like other Spring Template classes
  - HibernateTemplate
  - JDBCTemplate
- Primary method:
  - doInTransaction



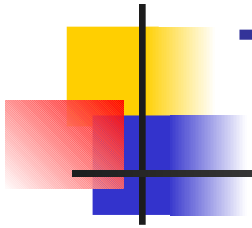
Declarative Transactions

- Modeled after EJB declarative transactions
- Transactions are declared not programmed
  - Configuration-based
  - Annotation-based
- Transaction support supplied by AOP
  - Several options exist
  - TransactionProxyFactoryBean
  - AspectJ
- Define a transaction manager

Spring Declarative Transaction Features

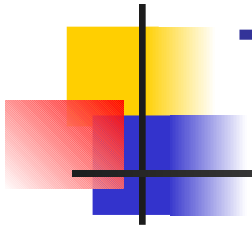


- POJO-baesd
- Supports Interfaces or Classes
- Rollback rules on certain exceptions
- Customizable using AOP features



Transaction Attributes

- Spring supports the following configuration of transactions:
  - Propagation behavior
  - Isolation levels
  - Read-only hints
  - Transaction timeout period

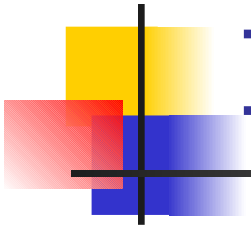


Transaction Propagation

- Defines the transaction boundry between client and method
- Supported modes
  - PROPAGATION_MANDATORY
  - PROPAGATION_NESTED
  - PROPAGATION_NEVER
  - PROPAGATION_NOT_SUPPORTED
  - PROPAGATION_REQUIRED
  - PROPAGATION_REQUIRES_NEW
  - PROPAGATION_SUPPORTS

Transaction Propagation

- PROPAGATION_REQUIRED
  - Support a current transaction, create a new one if none exists.
- PROPAGATION_SUPPORTS
  - Support a current transaction, execute non-transactionally if none exists.
- PROPAGATION_MANDATORY
  - Support a current transaction, throw an exception if none exist
- PROPAGATION_REQUIRES_NEW
  - Create a new transaction, suspend the current transaction if or exists.
- PROPAGATION_NOT_SUPPORTED
- PROPAGATION_NEVER
- PROPAGATION_NESTED

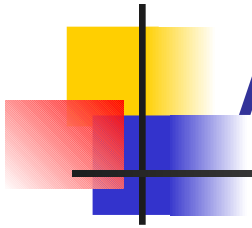


Isolation Levels

- Spring supports the following isolation levels on transactions:
  - ISOLATION_DEFAULT (datastore)
  - ISOLATION_READ_UNCOMMITTED
  - ISOLATION_READ_COMMITTED
  - ISOLATION_REPEATABLE_READ
  - ISOLATION_SERIALIZABLE

Transaction Isolation Levels

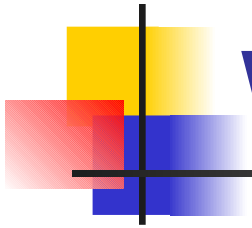
- ISOLATION_DEFAULT
- ISOLATION_READ_UNCOMMITTED
  - Dirty reads, non-repeatable reads and phantom reads can occur.
- ISOLATION_READ_COMMITTED
  - Dirty reads are prevented; non-repeatable reads and phantom reads can occur.
- ISOLATION_REPEATABLE_READ
  - Dirty reads and non-repeatable reads are prevented; phantom reads can occur.
- ISOLATION_SERIALIZABLE
  - Dirty reads, non-repeatable reads and phantom reads are prevented



Annotation vs. Configuration

- Previous versions of Spring did not support Annotation-based declarative transactions
- Configuration-based resulted in highly complex application context configuration
- Annotation-based is preferred approach
  - Much simpler configuration
  - More obvious in source code

Enabling Spring Transactions with Annotations



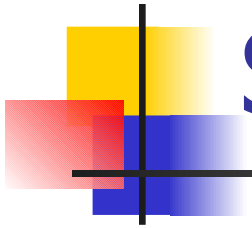
- Step 1: Enable Feature

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

- Step 2: Annotate Classes or Interfaces

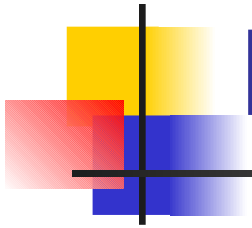
```
public interface StockService {

    @Transactional
    public Stock listStock(Stock stock);
```



Setting Transaction Attributes

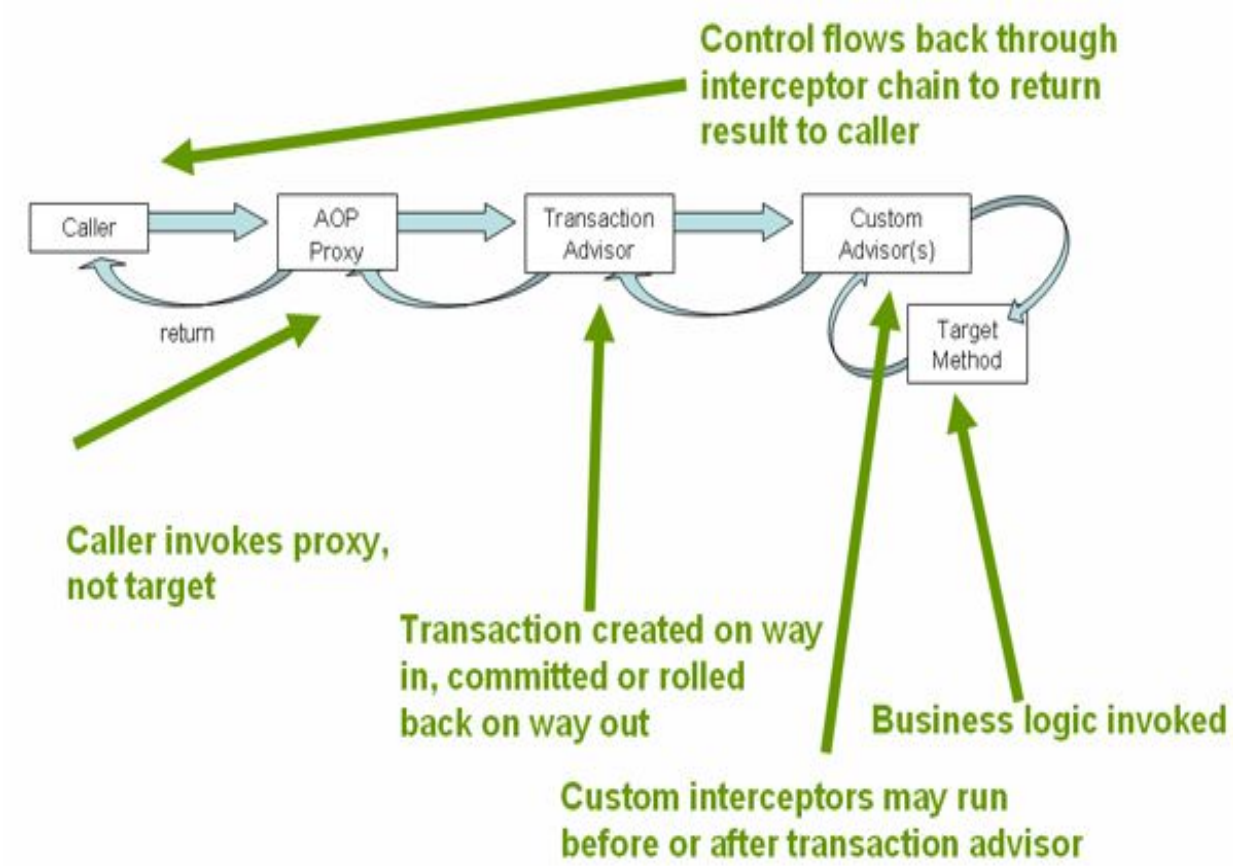
- @Transactional supports annotation parameters for:
  - Propagation
  - Isolation
  - Timeout
  - Read only
  - Rollback rules



Rollback Rules

- Spring supports declaring under what conditions a rollback should automatically take place
- Default behavior
  - Automatic rollback on RuntimeException

How Spring Transactions Work



From Spring 2.0 Reference

Esercizio

- Scaricare il codice della scorsa esercitazione e definire un nuovo bean implementando la classe PersistenceLayer che ha come dipendenze gli oggetti DAO giò disponibili.

- PersistenceLayer deve offrire metodi TRANSAZIONALI come:
  - saveUser, findUserById, deleteUser, updateUser

per ogni entità del modello. Questi metodi delegano il loro comportamento agli oggetti DAO specifici.

- Si adattino i test al nuovo setting