MODULE 13

# TRANSACTION

Françoise Dubisy

# TABLE OF CONTENT

## Transaction Definition

- A sequence of operations performed as a single logical unit of work
  - Commit
  - Rollback
- Keep a database consistent even in cases of system failure
  - Allow correct recovery from failures

# Session and Transaction

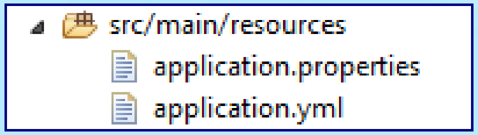
- A transaction is associated with a **session**

  = a **conversation** between the application and the datastore

- ⇨ Needs a SessionFactory
  - session = ***sessionFactory.getCurrentSession()***
- Create and begin a new transaction on a session
  - ***session.beginTransaction()***
- Commit the transaction
  - ***session.getTransaction().commit()***

# Session Factory Configuration

- Definition of current session context
  - Create ***application.properties*** in src/main/resources



```
src/main/resources
    application.properties
    application.yml
```

  - Containing
    - Property
      - ***spring.jpa.properties.hibernate.current_session_context_class***
    - Value
      - ***org.springframework.orm.hibernate4.SpringSessionContext***

```
spring.jpa.properties.hibernate.current_session_context_class=org.springframework.orm.hibernate4.SpringSessionContext
```

## Session Factory Configuration

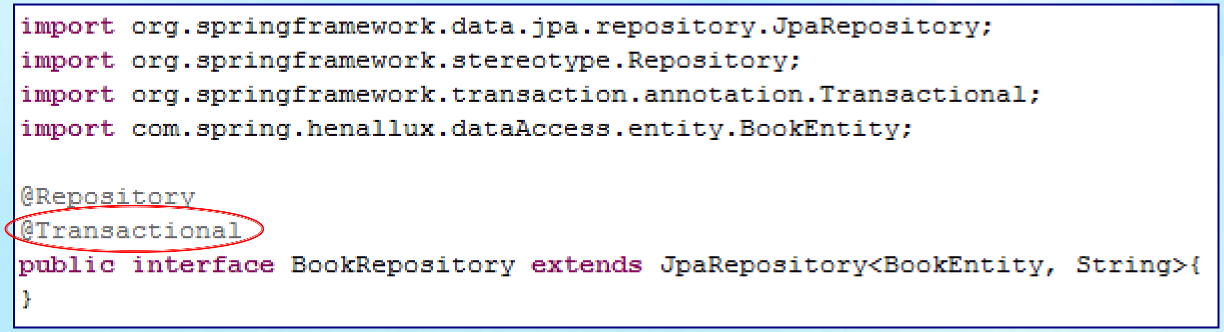
- Add a bean declaration in a configuration class

```
import org.springframework.orm.jpa.vendor.HibernateJpaSessionFactoryBean;
```

```
@Bean
public HibernateJpaSessionFactoryBean sessionFactory() {
    return new HibernateJpaSessionFactoryBean();
}
```

## @Transactional Annotation

- Add *@Transactional* to the repository

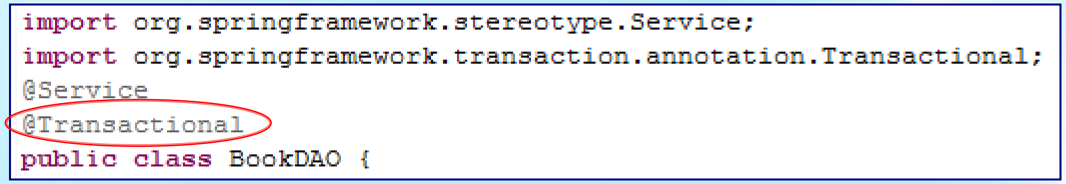


```java
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import com.spring.henallux.dataAccess.entity.BookEntity;

@Repository
@Transactional
public interface BookRepository extends JpaRepository<BookEntity, String>{
}
```

## @Transactional Annotation

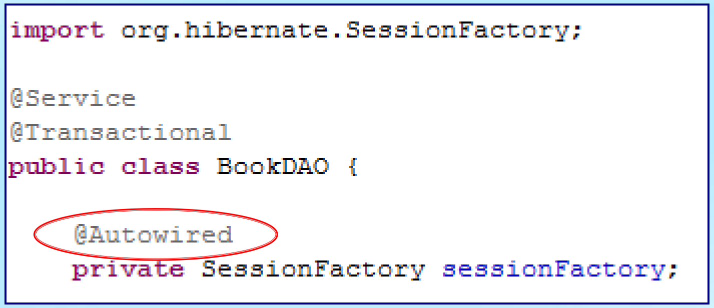
- Add ***@Transactional*** to the DAO class



```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
@Service
@Transactional
public class BookDAO {
```

# SessionFactory Injection

- Inject SessionFactory using ***@Autowired***
- E.g, in *BookDAO*



```
import org.hibernate.SessionFactory;

@Service
@Transactional
public class BookDAO {

    @Autowired
    private SessionFactory sessionFactory;
```

## Some Methods of Session

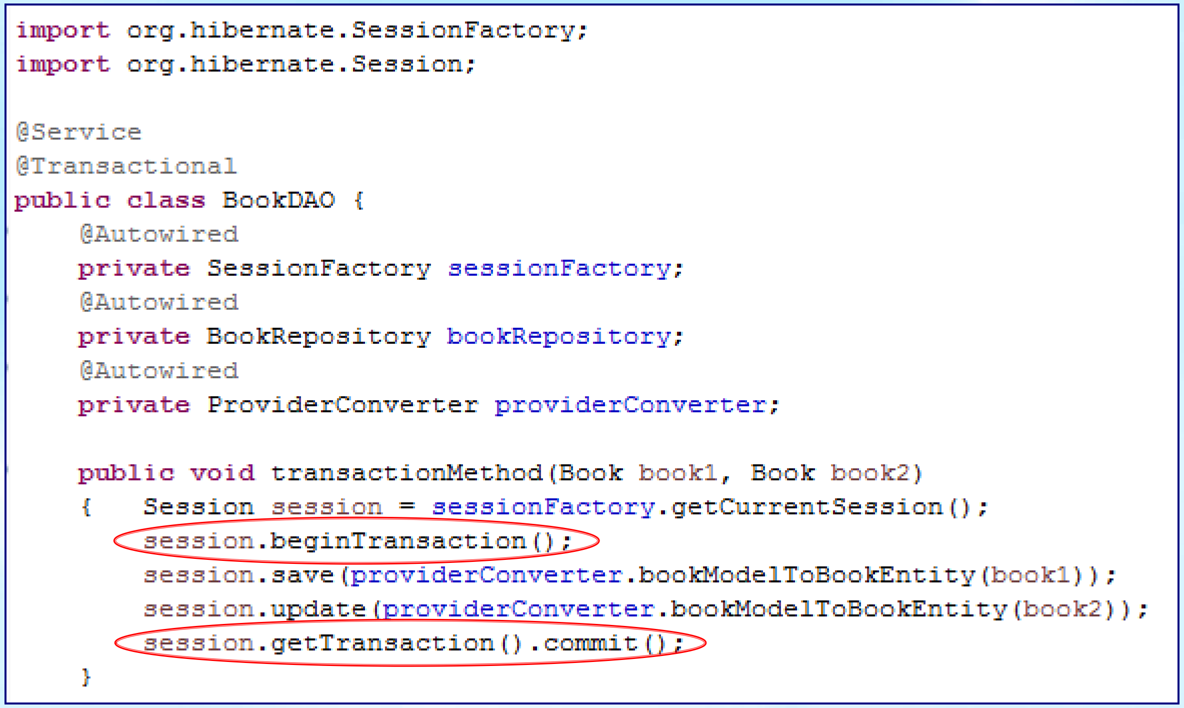
- Some methods of ***org.hibernate.Session***
  - ***beginTransaction()***
    - Begins a unit of work and returns the associated Transaction object
  - ***getTransaction()***
    - Gets the Transaction instance associated with this session
  - ***getNamedQuery(String queryName)***
    - Obtains an instance of Query for a named query string
    - *See Query module*

# Some Methods of Session

- Some methods of ***org.hibernate.Session*** (continue)
  - ***save(Object object)***
    - Persists the given transient instance
  - ***delete(Object object)***
    - Removes a persistent instance from the datastore
  - ***update(Object object)***
    - Updates the persistent instance with the identifier of the given detached instance

# Method Using Transaction

- Create method using transaction
- E.g, in *BookDAO*



```
import org.hibernate.SessionFactory;
import org.hibernate.Session;

@Service
@Transactional
public class BookDAO {
    @Autowired
    private SessionFactory sessionFactory;
    @Autowired
    private BookRepository bookRepository;
    @Autowired
    private ProviderConverter providerConverter;

    public void transactionMethod(Book book1, Book book2)
    {   Session session = sessionFactory.getCurrentSession();
        session.beginTransaction();
        session.save(providerConverter.bookModelToBookEntity(book1));
        session.update(providerConverter.bookModelToBookEntity(book2));
        session.getTransaction().commit();
    }
```