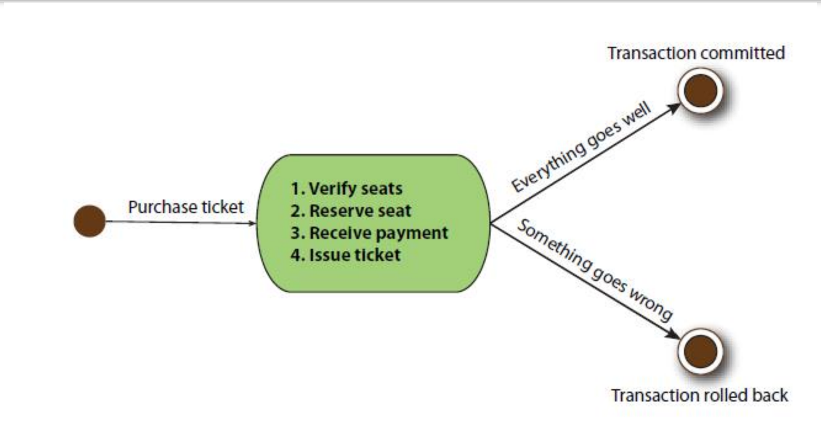
SPRING TRANSACTIONS

TRANSACTIONS SUPPORT ARCHITECTURE

MARCH, 2015

What transaction is?



Atomicity – All or Nothing

Consistent – Only valid data

Isolated – No interference

Durable – Data is recoverable

Transaction model

- Local
  - Work across single transactional resource
  - Resource-specific
  - Easier to use
- Global
  - Work across multiple transactional resources

Spring transaction support benefits

- Consistent programming model across different transaction APIs
- Support for declarative transaction management
- Simpler API for programmatic transaction management than complex transaction APIs such as JTA
- Excellent integration with Spring's data access abstractions

Simple transaction implementation

```
@Transactional(readOnly = true)
public class DefaultFooService implements FooService {
  public Foo getFoo(String fooName) {
    // do something
  }

@Transactional(readOnly = false,
                          propagation = Propagation.REQUIRES_NEW)
  public void updateFoo(Foo foo) {
    // do something
  }
}
```

Application context configuration

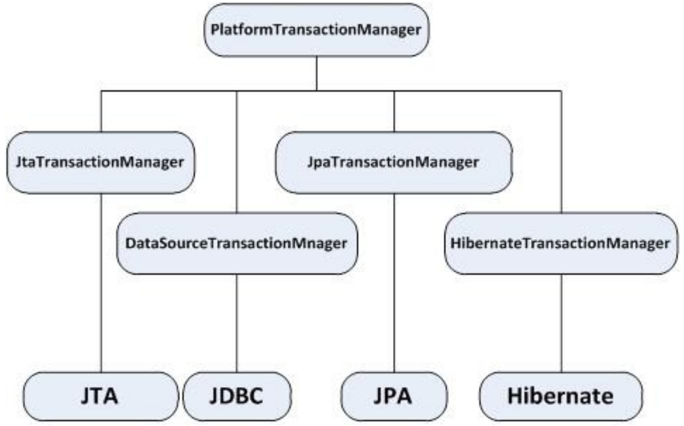
```
<tx:annotation-driven transaction-manager="txManager"/>

<bean id=“dataSource” class=“org.apache.commons.dbcp.BasicDataSource”
                      destroyMethod=“close”>
    …
</bean

<bean id=“jdbcTemplate”
                  class=“org.springframework.jdbc.core.JdbcTemplate”>
    <property name=“dataSource” ref=“dataSource”/>
</bean>

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

PlatformTransactionManager


PlatformTransactionManager
JtaTransactionManager
JpaTransactionManager
DataSourceTransactionMnager
HibernateTransactionManager
JTA
JDBC
JPA
Hibernate

```
public interface PlatformTransactionManager {
   TransactionStatus getTransaction(TransactionDefinition definition)
                                                  throws TransactionException;

void commit(TransactionStatus status) throws TransactionException;

void rollback(TransactionStatus status) throws TransactionException;
}
```

Programmatic transaction definition

```
public interface TransactionDefinition {

    int getPropagationBehavior();
    int getIsolationLevel();
    int getTimeout();
    boolean isReadOnly();
    String getName();
}
```

```
public interface TransactionStatus extends SavepointManager {
    boolean isNewTransaction();
    boolean hasSavepoint();
    void setRollbackOnly();
    boolean isRollbackOnly();
    void flush();
    boolean isCompleted();
}
```

THANK YOU!

MAKSYM_GOVORISCHEV@EPAM.COM

MARCH, 2015