1. Transaction의 개념
1)논리적 단위로 어떤 한 부분의 작업이 완료되었다 하더라도, 다른 부분의 작업이 완료되지 않을 경우 전체 취소되는 것이다.
2)이때, 작업이 완료되는 것을 커밋(commit)이라고 하고, 작업이 취소되는 것을 롤백(rollback)이라고 한다.
3)우리 일상생활에 transaction의 예는 많이 볼 수 있다.
4)영화 예매를 할 경우 카드 결제 작업과 마일리지 적립 작업은 transaction으로 작동해야 한다.
5)또한 은행 ATM기도 마찬가지 이다.
6)transaction은 그래서 성공적으로 처리되거나 또는 하나라도 실패하면 완전히 실패 처리를 해야 하는 경우에 사용된다.
7)Spring이 지원하는 transaction 방법 - 코드 기반 처리, 선언적 transaction, annotation 기반등을 사용한다.
8)인터넷에서 도서를 구매할 경우 다음과 같은 순서가 필요할 것이다.
-결제 수행
-결제 내역 저장
-구매 내역 저장
9)위의 과정 모두 성공적으로 이루어져야 한다.
10)하나라도 실패할 경우 반드시 모든 과정이 취소되어야 한다.
11)예를 들어, 결제 내역 저장까지는 성공했는데, 구매 내역을 저장하는 과정이 실패했다고 하자.
12)이때, 전 과정이 취소되지 않는다면 구매자는 결제만 하고 구매는 하지 않은 것처럼 될 것이다.
13)이처럼 transaction은 여러 과정을 하나의 행위로 묶을 때 사용된다.
14)transaction은 transaction 범위 내에 있는 처리 과정 중 한 가지라도 실패할 경우 전체 과정을 취소시킴으로써 데이터의 무결성을 보장한다.
15)즉, transaction은 모두 반영하거나 모두 반영하지 않는 all or nothing 방식을 취한다.
16)transaction은 보통 4가지 특징인 ACID를 이용한다.
-원자성(Atomicity)
--transaction은 한 개 이상의 동작을 논리적으로 한 개의 작업 단위(unit or work)로 묶는다.
--원자성은 transaction 범위에 있는 모든 동작이 모두 실행되거나 또는 모두 실행이 취소됨을 보장한다.
--모든 동작이 성공적으로 실행되면 transaction은 성공한다.
--만약 하나라도 실패하면 transaction은 실패하고 모든 과정을 롤백한다.

-일관성(Consistency)
--transaction이 종료되면, system은 business에서 기대하는 상태가 된다.
--예를 들어, 서적 구매 transaction이 성공적으로 실행되면 결제 내역, 구매 내역, 잔고 정보가 business에 맞게 저장되고 변경된다.

-고립성(Isolation)
--transaction은 다른 transaction과 독립적으로 실행되어야 하며, 서로 다른 transaction이 동일한 데이터에 동시에 접근할 경우 알맞게 동시 접근을 제어해야 한다.

-지속성(Durability)
--transaction이 완료되면, 그 결과는 지속적으로 유지되어야 한다.
--현재의 application이 변경되거나 없어지더라도 data는 유지된다.
--일반적으로 물리적인 저장소를 통해서 transaction 결과가 저장된다.

2. Spring Transaction을 사용하지 않았을 경우
-transaction 처리를 하지 않았을 경우 rollback이 되지 않는 경우이다.

1)Spring Transaction Project 생성
-Package Explorer > right-click > New >Spring Legacy Project
-Project name : SpringTransactionDemo
-Select [Spring MVC Project]
-Next
-Project Settings - com.javasoft.biz > Finish

2)Maven Install and Update
-In pom.xml > Dependencies tab > click [Add]
-[Enter groupId, artifactId or sha1... 'spring jdbc' 입력
-org.springframework spring-jdbc 선택

-[OK]
-pom.xml > right-click > Run As > Maven clean and Maven install

3)/WEB-INF/config folder 생성
-config > right-click > New > Spring Bean Configuration File
-File name : applicationContext.xml > Finish
-web.xml 에서

```
<init-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
</init-param>
```

-아래로 수정할 것

```
<init-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/config/applicationContext.xml</param-value>
</init-param>
```

4)config/dbinfo.properties

```
db.driver=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@192.168.56.2:1521:ORCL
db.username=scott
db.password=tiger
```

5)pom.xml에 코드 추가

```
<dependency>
   <groupId>com.oracle</groupId>
   <artifactId>ojdbc6</artifactId>
   <version>11.2</version>
</dependency>
```

6)config/applicationContext.xml 코드 추가

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:mvc="http://www.springframework.org/schema/mvc"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/mvc
   http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-4.3.xsd">

   <!-- Enables the Spring MVC @Controller programming model -->
   <mvc:annotation-driven />

   <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static
   resources in the ${webappRoot}/resources directory -->
   <mvc:resources mapping="/resources/**" location="/resources/" />


   <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the
   /WEB-INF/views directory -->
   <bean
```

```
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
109         <property name="prefix" value="/WEB-INF/views/" />
110         <property name="suffix" value=".jsp" />
111      </bean>
112
113      <context:component-scan base-package="com.javasoft" />
114      <context:property-placeholder location="/WEB-INF/config/dbinfo.properties" />
115      <bean id="dataSource"
         class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
116         <property name="driverClass" value="${db.driver}" />
117         <property name="url" value="${db.url}" />
118         <property name="username" value="${db.username}" />
119         <property name="password" value="${db.password}" />
120      </bean>
121
122      <bean name="template" class="org.springframework.jdbc.core.JdbcTemplate">
123         <property name="dataSource" ref="dataSource" />
124      </bean>
125
126   </beans>
127
128 7)src/main/java/com.javasoft.biz.HomeController.java
129
130   package com.javasoft.biz;
131
132   import java.text.DateFormat;
133   import java.util.Date;
134   import java.util.Locale;
135
136   import org.slf4j.Logger;
137   import org.slf4j.LoggerFactory;
138   import org.springframework.stereotype.Controller;
139   import org.springframework.ui.Model;
140   import org.springframework.web.bind.annotation.RequestMapping;
141   import org.springframework.web.bind.annotation.RequestMethod;
142
143   @Controller
144   public class HomeController {
145
146      @RequestMapping(value = "/", method = RequestMethod.GET)
147      public String home(Model model) {
148         model.addAttribute("greeting", "Hello Spring Transaction");
149         return "home";
150      }
151   }
152
153 8)views/home.jsp
154
155   <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
156   <%@ page session="false" %>
157   <html>
158   <head>
159      <title>Home</title>
160   </head>
161   <body>
162      <h1>${greeting}</h1>
163   </body>
164   </html>
165
166 9)project > right-click > Run As > Run on server
```

10)Database Table 생성

```
CREATE TABLE Card
(
    consumerId    VARCHAR2(20),
    amount        NUMBER(2),
    CONSTRAINT card_consumerid_pk PRIMARY KEY(consumerId)
);

CREATE TABLE Ticket
(
    consumerId    VARCHAR2(20),
    countnum       NUMBER(2),
    CONSTRAINT ticket_consumerid_pk PRIMARY KEY(consumerId),
    CONSTRAINT ticket_consumerid_fk FOREIGN KEY(consumerId) REFERENCES
    Card(consumerId),
    CONSTRAINT ticket_countnum_ck CHECK(countnum < 5)
);
```

11)src/main/java/com.javasoft.biz.TicketVO.java

```
package com.javasoft.biz;

public class TicketVO {
   private String consumerId;
   private String amount;
   public String getConsumerId() {
      return consumerId;
   }
   public void setConsumerId(String consumerId) {
      this.consumerId = consumerId;
   }
   public String getAmount() {
      return amount;
   }
   public void setAmount(String amount) {
      this.amount = amount;
   }
}
```

12)src/main/java/com.javasoft.biz.TicketDao.java

```
package com.javasoft.biz;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository("ticketDao")
public class TicketDao {
   @Autowired
   private JdbcTemplate jdbcTemplate;

   public void buyTicket(final TicketVO ticketVO) {
      System.out.println("buyTicket()");
      System.out.println("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
      System.out.println("ticketVO.getAmount() : " + ticketVO.getAmount());

      String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
```

```
            this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());

            sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
            this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
        }
    }
```

13)views/buy_ticket.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
    <!DOCTYPE html>
    <html>
        <head>
            <meta charset="UTF-8">
            <title>Ticket 구매 창</title>
        </head>
        <body>
            <h1>카드 결제</h1>

            <form action="buy_ticket_card" method="post">
                고객 아이디 : <input type="text" name="consumerId" > <br />
                티켓 구매수 : <input type="text" name="amount" > <br />
                <input type="submit" value="구매" > <br />
            </form>

        </body>
    </html>
```

14)views/buy_ticket_result.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
    <!DOCTYPE html>
    <html>
        <head>
            <meta charset="UTF-8">
            <title>Ticket 구매 결과 창</title>
        </head>
        <body>
            <h1>Ticket 구매 결과</h1>
            <ul>
                <li>고객 아이디 : ${ticketInfo.consumerId }</li>
                <li>구매 갯수 : ${ticketInfo.amount }</li>
            </ul>
        </body>
    </html>
```

15)HomeController.java 코드 추가

```
    package com.javasoft.biz;

    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Controller;
    import org.springframework.ui.Model;
    import org.springframework.web.bind.annotation.RequestMapping;
    import org.springframework.web.bind.annotation.RequestMethod;

    @Controller
    public class HomeController {
```

```
286         @Autowired
287         private TicketDao dao;
288
289         @RequestMapping(value = "/", method = RequestMethod.GET)
290         public String home(Model model) {
291            model.addAttribute("greeting", "Hello Spring Transaction");
292            return "home";
293         }
294
295         @RequestMapping("/buy_ticket")
296         public String buy_ticket() {
297            return "buy_ticket";
298
299         }
300
301         @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
302         public String buy_ticket_card(TicketVO ticketVO, Model model) {
303            System.out.println( "buy_ticket_card" );
304            System.out.println( "고객 아이디 : " + ticketVO.getConsumerId() );
305            System.out.println( "구매 갯수 : " + ticketVO.getAmount() );
306
307            dao.buyTicket(ticketVO);
308
309            model.addAttribute("ticketInfo", ticketVO);
310
311            return "buy_ticket_result";
312         }
313      }
314
315   16)project > right-click > Run As > Run on server
316
317   17)http://localhost:8080/biz/buy_ticket
318      -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
319      -하지만 구매 갯수를 5장 이상을 입력하면 오류 발생
320         --Card table에는 구매 갯수가 5 이상의 값 입력이 가능하다.
321         --하지만 Ticket table에는 ORA-02290: check constraint
            (SCOTT.TICKET_COUNTNUM_CK) 위반이 발생했기 때문에 입력에 실패하게 된다.
322      -이렇게 오류가 나오는 것이 정상이다.
323      -그리고 치명적인 오류인 것은 Car 테이블에는 5장 이상 구매한 데이터는 입력이 되지만, Ticket
         table에는 체크 위반 때문에 입력되지 않는다는 것이다.
324      -두 테이블 모두 입력 성공하거나 입력 취소가 되어야 한다.
325
326
327 3. Spring Transaction 처리
328   1)JDBC 기반 Transaction Manager 설정
329      -JDBC나 MyBatis와 같이 JDBC를 이용하는 database 연동을 처리하는 경우
330
331      <bean name="transactionManager"
         class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
332         <property name="dataSource" ref="dataSource" />
333      </bean>
334
335   2)JPA Transaction Manager 설정
336      -JPA를 사용할 경우
337
338      <bean id="entityManagerFactory"
         class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
339         ...
340      </bean>
341
```

```
<bean name="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
   <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

3)Hibernate Transaction Manager 설정
   -Hibernate를 사용하는 경우

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

   <bean id="sessionFactory"
   class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
      <property name="dataSource" ref="dataSource" />
      ...
   </bean>

4)Transaction 전파 속성
   -org.springframework.transaction.TransactionDefinition interface
   -2개 이상의 transaction이 작동할 때, 기존의 transaction에 참여하는 방법을 결정하는 속성.
   -PROPAGATION_REQUIRED : 0
      --method를 수행하는 데 transaction이 필요하다는 것을 의미
      --현재 진행 중인 transaction이 존재하면, 해당 transaction을 사용한다.
      --존재하지 않는다면 새로운 transaction을 생성한다.
      --default, 즉 전체 처리한다.

   -PROPAGATION_MANDATORY : 2
      --method를 수행하는 데 transaction이 필요하다는 것을 의미한다.
      --하지만 위의 REQUIRED와 달리, 진행중인 transaction이 존재하지 않을 경우 exception을
      발생시킴
      --transaction에 꼭 포함되어야 함

   -PROPAGATION_REQUIRES_NEW : 3
      --항상 새로운 transaction을 시작한다.
      --기존 transaction이 존재하면 기존 transaction을 일시 중지하고 새로운 transaction을 시작한다.
      --새로 시작된 transaction이 종료된 뒤에 기존 transaction이 계속된다.
      --각자 transaction 처리(별도의 transaction 처리)

   -PROPAGATION_SUPPORTS : 1
      --method가 transaction을 필요로 하지 않지만, 기존 transaction이 존재할 경우
      transaction을 사용한다는 것을 의미한다.
      --진행중인 transaction이 존재하지 않더라고 method는 정상적으로 동작한다.
      --기존 transaction에 의존

   -PROPAGATION_NOT_SUPPORTED : 4
      --method가 transaction을 필요로 하지 않음을 의미한다.
      --SUPPORTS와 달리 진행 중인 transaction이 존재할 경우 method가 실행되는 동안
      transaction은 일시 중지되며, method 실행이 종료된 뒤에 transaction을 계속 진행한다.
      --transaction에 포함하지 않음 -> transaction이 없는 것과 동일

   -PROPAGATION_NEVER : 5
      --method가 transaction을 필요로 하지 않으며, 만약 진행 중인 transaction이 존재하면
      exception을 발생시킴.
      --transaction에 절대 포함하지 않음.

   -PROPAGATION_NESTED : 6
      --기존 transaction이 존재하면, 기존 transaction에 중첩된 transaction에서 method를 실행한다.
      --기존 transaction이 존재하지 않으면 REQUIRED와 동일하게 동작한다.
```

--이 기능은 JDBC 3.0 driver를 사용할 때에만 적용된다.

REQUIRED <----> REQUIRES_NEW
MANDATORY <---> NEVER
SUPPORTS <---> NOT_SUPPORTED

5)Spring에서 설정 가능한 transaction 격리 level
-ISOLATION_DEFAULT
--기본 설정 사용

-ISOLATION_READ_UNCOMMITTED
--다른 transaction에서 commit하지 않은 data를 읽을 수 있다.

-ISOLATION_READ_COMMITTED
--다른 transaction에 의해 commit된 data를 읽을 수 있다.

-ISOLATION_REPEATABLE_READ
--처음에 읽어 온 data와 두 번째 읽어 온 data가 동일한 값을 갖는다.

-ISOLATION_SERIALIZABLE
--동일한 data에 대해서 동시에 두 개 이상의 transaction이 수행될 수 있다.

4. PlatformTransactionManager를 사용하는 Lab
1)applicationContext.xml에 다음의 코드를 추가한다.

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

2)TicketDao.java 코드 추가

```
@Autowired
private PlatformTransactionManager transactionManager;  <-- 코드 추가

public void buyTicket(final TicketVO ticketVO) {
   System.out.println("buyTicket()");
   System.out.println("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
   System.out.println("ticketVO.getAmount() : " + ticketVO.getAmount());

   TransactionDefinition definition = new
   DefaultTransactionDefinition();                    <--코드 추가
   TransactionStatus status =
   this.transactionManager.getTransaction(definition);          <--코드 추가

   try
   {
     <--코드 추가
      String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
      this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());

      sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
      this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());

      this.transactionManager.commit(status);
           <--코드 추가
   }catch(Exception e)
   {                                                                    <--코드 추가
```

```
            e.printStackTrace();
                  <--코드 추가
447
            this.transactionManager.rollback(status);
                  <--코드 추가
448
         }
               <--코드 추가
449      }
450
451   3)project > right-click > Run As > Run on server
452
453   4)http://localhost:8080/biz/buy_ticket
454      -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
455      -하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
456      -transaction 성공했음.
457
458
459 5. TransactionTemplate 이용한 Transaction 처리하기
460   1)기본적으로 사용한 PlatformTransactionManager interface보다 더 많이 사용되는 객체는
      TransactionTemplate 이다.
461   2)Transaction을 처리하기 위해 PlatformTransactionManager의 메소드를 직접 사용해도 되지만
      try/catch 블록을 써야 하는 번거로움이 발생한다.
462   3)Transaction 안에서 작업 중에 예외가 발생한 경우에는 Transaction을 롤백해주도록 만들어야 하기
      때문이다.
463   4)그래서 PlatformTransactionManager의 메소드를 직접 사용하는 대신 Template/Callback 방식의
      TransactionTemplate을 이용하면 편리하다.
464
465
466   2)applicationContext.xml 코드 추가
467
468      <bean name="transactionTemplate"
         class="org.springframework.transaction.support.TransactionTemplate">
469         <property name="transactionManager" ref="transactionManager" />
470      </bean>
471
472   3)TicketDao.java 코드 수정 및 추가
473
474      @Repository("ticketDao")
475      public class TicketDao {
476         @Autowired
477         private JdbcTemplate jdbcTemplate;
478
479         @Autowired
480         private TransactionTemplate transactionTemplate;
481
482
483         public void buyTicket(final TicketVO ticketVO) {
484            System.out.println("buyTicket()");
485            System.out.println("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
486            System.out.println("ticketVO.getAmount() : " + ticketVO.getAmount());
487
488            this.transactionTemplate.execute(new TransactionCallbackWithoutResult() {
489
490               @Override
491               protected void doInTransactionWithoutResult(TransactionStatus status) {
492                  String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
493                  TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
                     ticketVO.getAmount());
494
```

```
                sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
                TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
                ticketVO.getAmount());
                // 트랜잭션 안에서 동작하는 코드, 트랜잭션 매니저와 연결되어 있는 모든 DAO는 같은
                트랜잭션에 참여한다.
                // 정상적으로 작업을 마치고 리턴되면 트랜잭션은 커밋된다.
                // 만약 이전에 시작한 트랜잭션에 참여했다면 해당 트랜잭션의 작업을 모두 마칠 때까지
                커밋은 보류된다.
                // 리턴되기 이전에 예외가 발생하면 트랜잭션은 롤백된다.
            }

        });
      }
    }
```

4)project > right-click > Run As > Run on server

5)http://localhost:8080/biz/buy_ticket
-구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
-하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
-다만 화면에 에러 메시지 display
-transaction 성공했음.

4. Transaction 전파에 관한 Lab
1)TicketService.java

```
package com.javasoft.biz;

public interface TicketService {
    void execute(TicketVO ticketVO);
}
```

2)TicketServiceImpl.java

```
package com.javasoft.biz;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.TransactionCallbackWithoutResult;
import org.springframework.transaction.support.TransactionTemplate;

@Service("ticketService")
public class TicketServiceImpl implements TicketService {
    private TicketDao ticketDao;
    private TransactionTemplate transactionTemplate2;

    public void setTicketDao(TicketDao ticketDao) {
        this.ticketDao = ticketDao;
    }

    public void setTransactionTemplate2(TransactionTemplate transactionTemplate2) {
        this.transactionTemplate2 = transactionTemplate2;
    }

    @Override
    public void execute(final TicketVO ticketVO) {

        //ticketVO.setAmount("6");
        //ticketDao.buyTicket(ticketVO);
```

```
552
553             transactionTemplate2.execute(new TransactionCallbackWithoutResult() {
554
555                 @Override
556                 protected void doInTransactionWithoutResult(TransactionStatus status) {
557                     ticketVO.setAmount("1");
558                     ticketDao.buyTicket(ticketVO);
559                 }
560             });
561         }
562     }
563
564 3)HomeController.java
565
566     @Autowired
567     private TicketService ticketService;
568     ...
569     ...
570
571     @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
572     public String buy_ticket_card(TicketVO ticketVO, Model model) {
573         System.out.println( "buy_ticket_card" );
574         System.out.println( "고객 아이디 : " + ticketVO.getConsumerId() );
575         System.out.println( "구매 갯수 : " + ticketVO.getAmount() );
576
577         //dao.buyTicket(ticketVO);
578         this.ticketService.execute(ticketVO);
579         model.addAttribute("ticketInfo", ticketVO);
580
581         return "buy_ticket_result";
582     }
583
584 4)TicketDao.java
585
586     @Repository("ticketDao")
587     public class TicketDao {
588         private JdbcTemplate jdbcTemplate;
589         private TransactionTemplate transactionTemplate1;
590
591         public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
592             this.jdbcTemplate = jdbcTemplate;
593         }
594
595         public void setTransactionTemplate1(TransactionTemplate transactionTemplate1) {
596             this.transactionTemplate1 = transactionTemplate1;
597         }
598
599         public void buyTicket(final TicketVO ticketVO) {
600             System.out.println("buyTicket()");
601             System.out.println("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
602             System.out.println("ticketVO.getAmount() : " + ticketVO.getAmount());
603
604             this.transactionTemplate1.execute(new TransactionCallbackWithoutResult() {
605
606                 @Override
607                 protected void doInTransactionWithoutResult(TransactionStatus status) {
608                     String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
609                     TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
                        ticketVO.getAmount());
610
```

```
                sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
                TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
                ticketVO.getAmount());
            }

        });
      }
    }
```

5)applicationContext.xml

```
    ...
    <bean name="transactionTemplate1"
    class="org.springframework.transaction.support.TransactionTemplate">
      <property name="transactionManager" ref="transactionManager" />
      <property name="propagationBehavior" value="0"></property>
    </bean>
    <bean name="transactionTemplate2"
    class="org.springframework.transaction.support.TransactionTemplate">
      <property name="transactionManager" ref="transactionManager" />
      <property name="propagationBehavior" value="0"></property>
    </bean>
    <bean name="ticketDao" class="com.javasoft.biz.TicketDao" >
      <property name="jdbcTemplate" ref="template" />
      <property name="transactionTemplate1" ref="transactionTemplate1" />
    </bean>

    <bean name="ticketService" class="com.javasoft.biz.TicketServiceImpl" >
      <property name="ticketDao" ref="ticketDao" />
      <property name="transactionTemplate2" ref="transactionTemplate2" />
    </bean>
```