전자정부 표준프레임워크
실행환경(데이터처리)
실습

eGovFrame

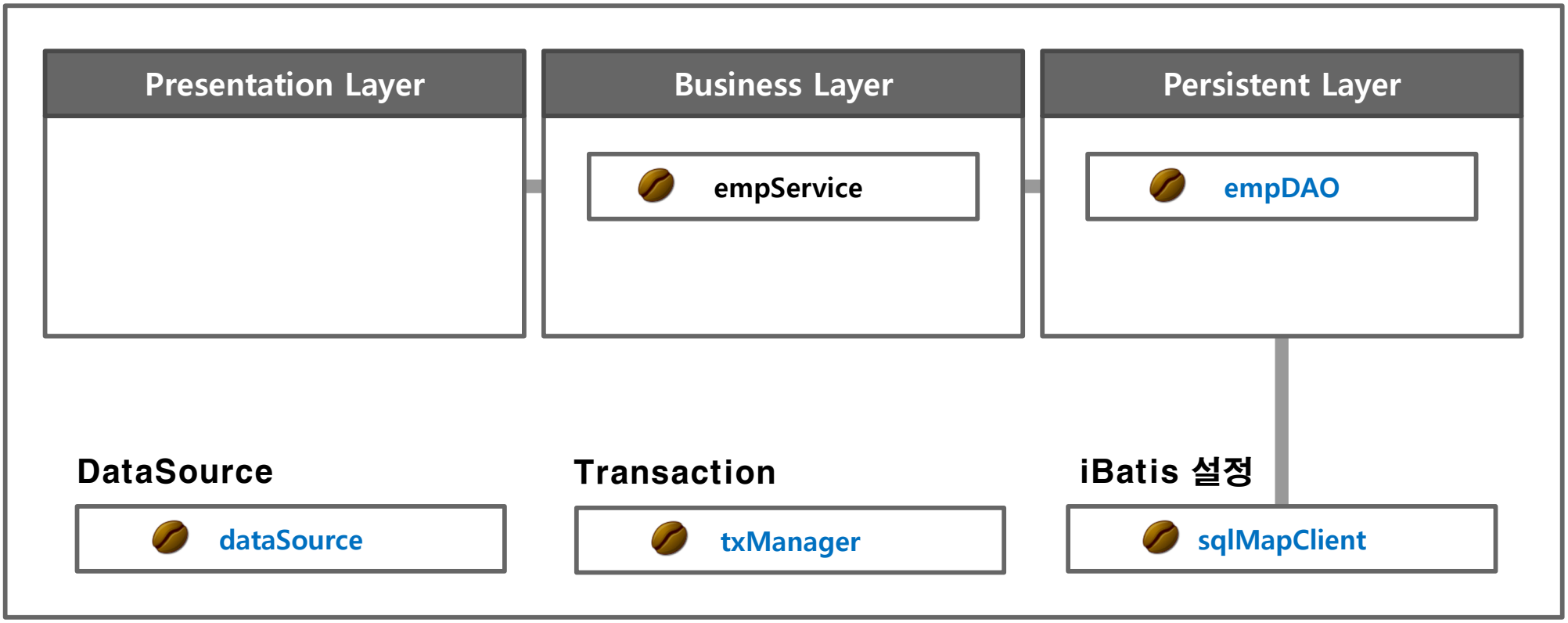
Contents

실습 개요

- 실습을 통해 iBatis와 MyBatis 설정 및 사용법에 대해 알아본다.
- 실습 순서
  - iBatis 예제
    1. 구동 환경 설정 (DataSource 빈 설정, 스프링 연동 빈 설정, iBatis XML 설정)
    2. Service, DAO 클래스 작성
    3. 오브젝트-SQL 매핑 파일 작성
    4. CRUD 테스트
  - MyBatis 예제
    1. 구동 환경 설정 (DataSource 빈 설정, 스프링 연동 빈 설정, MyBatis XML 설정)
    2. Service, DAO 클래스 작성
    3. 오브젝트-SQL 매핑 파일 작성
    4. CRUD 테스트

LAB 204-iBatis 실습

❑ lab204-ibatis 프로젝트의 beans



Step 1. 구동 환경 설정

❑ 1. Hsqldb 초기화 스크립트

- /lab204-ibatis/src/test/resources/META-INF/testdata/sample_schema_hsql.sql 를 확인한다.
- 현 실습 프로젝트에서는 편의상 매 테스트 케이스 재실행 시 관련 table 을 drop/create 하고 있음.

❑ 2. dataSource 설정

- <bean id="dataSource" .../>를 이용한 방법
- <jdbc:embedded-database .../>를 이용한 방법

Step 1. 구동 환경 설정

❑ 2-1. dataSource 설정 (1/2)

- /lab204-ibatis/src/test/resources/META-INF/spring/context-datasource.xml 에 dataSource 빈 설정을 추가한다.

```
<!-- TODO [Step 1-2] dataSource 설정 -->
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource" destroy-
method="close">
  <property name="driverClassName" value="${db.driver}" />
  <property name="url" value="${db.dburl}" />
  <property name="username" value="${db.username}" />
  <property name="password" value="${db.password}" />
  <property name="defaultAutoCommit" value="false" />
  <property name="poolPreparedStatements" value="true" />
</bean>
```

- /lab204-ibatis/src/test/resources/META-INF/spring/context-common.xml 에 context:property-placeholder 설정을 추가한다.

```
<!-- TODO [Step 1-2] PropertyPlaceholderConfigurer 설정 -->
<context:property-placeholder location="classpath:/META-INF/spring/jdbc.properties" />
```

Step 1. 구동 환경 설정

❑ 2-1. dataSource 설정 확인 (2/2)

- /lab204-ibatis/src/test/resources/META-INF/spring/jdbc.properties 를 **확인**한다.

```
#TODO [Step 1-2] dataSource 설정
db.driver=org.hsqldb.jdbcDriver
#db.dburl=jdbc:hsqldb:mem:testdb
db.dburl=jdbc:hsqldb:hsql://localhost/sampledb
db.username=sa
db.password=
```

cf.) /lab204-ibatis/db 상에서 (외부 탐색기에서) runHsqlDB.cmd 를 실행하여 DB Server 를 구동하고 테스트를 진행한다.

Step 1. 구동 환경 설정

❑ 3. transaction 설정

- /lab204-ibatis/src/test/resources/META-INF/spring/context-transaction.xml 를 작성한다.

```
<!-- TODO [Step 1-3] transaction 설정 -->
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

cf.) 여기서는 transaction manager 만을 설정하였고, TestCase 내에서 전역 @Transactional 설정으로 트랜잭션을 일괄 지정하고 있으나, 보통 AOP 형식(tx:aop)의 트랜잭션 대상 지정으로 비즈니스 서비스의 메서드에 일괄 지정하는 경우가 많다.

cf2.) @Transactional Annotation 으로 대상 메서드에 개별로 따로 지정할 수도 있다.

Step 1. 구동 환경 설정

❑ 4. Spring 의 iBATIS 연동 설정

- /lab204-ibatis/src/test/resources/META-INF/spring/context-sqlMap.xml 를 작성한다.

```
<!-- TODO [Step 1-4] Spring 의 iBATIS 연동 설정 -->
<bean id="sqlMapClient"
class=”org.egovframe.rte.psl.orm.ibatis.SqlMapClientFactoryBean">
<property name="configLocation"
value="classpath:/META-INF/sqlmap/sql-map-config.xml" />
<!--
<property name="mappingLocations"
value="classpath:/META-INF/sqlmap/mappings/lab-*.xml" />
 -->
<property name="dataSource" ref="dataSource" />
</bean>
```

- 최신 프레임워크 환경에서는 sql-map-config.xml 내에 개별 sql 맵핑 파일일 일일이 지정하는 것이 아니라 위의 mappingLocations 영역을 주석 해제하여 Spring 의 ResourceLoader 형식으로 패턴 매칭에 의거한 일괄 로딩으로 처리가 가능하다. (단, 테스트 결과 CacheModel 등의 일부 기능에서 문제가 발생하는 경우가 있으므로 사용에 유의할것!!)

❑ 5. iBATIS 의 sql-map-config 설정 파일 작성

- /lab204-ibatis/src/test/resources/META-INF/sqlmap/sql-map-config.xml 를 작성한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//ibatis.apache.org//DTD SQL Map Config
2.0//EN" "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
<!-- TODO [Step 1-5] iBATIS 의 sql-map-config 설정 파일 작성 -->

<settings useStatementNamespaces="false" cacheModelsEnabled="true" />

<!-- Spring 2.5.5 이상, iBATIS 2.3.2 이상에서는 iBATIS 연동을 위한
SqlMapClientFactoryBean
정의 시 mappingLocations 속성으로 Sql 매핑 파일의 일괄 지정이 가능하다.
("sqlMapClient" bean 설정 시 mappingLocations="classpath:/META-
INF/sqlmap/mappings/lab-*.xml" 로 지정하였음)
단, sql-map-config-2.dtd 에서 sqlMap 요소를 하나 이상 지정하도록 되어 있으므로 아래
의 dummy 매핑 파일을 설정한다. -->
<sqlMap resource="META-INF/sqlmap/mappings/lab-emp.xml" />
</sqlMapConfig>
```

❑ 6. ID Generation Service 설정 확인

- /lab204-ibatis/src/test/resources/META-INF/spring/context-idgen.xml 를 확인한다.

```
<!-- [Step 1-6] Id Generation Service 설정 -->
<bean name="primaryTypeSequenceIds"
class="org.egovframe.rte.fdl.idgnr.impl.EgovSequenceIdGnrService"
destroy-method="destroy">
<property name="dataSource" ref="dataSource" />
<property name="query" value="SELECT NEXT VALUE FOR empseq FROM DUAL" />
</bean>
```

- 여기서는 Hsqldb 를 사용하여 Oracle 의 DUAL 테이블 역할을 할 수 있도록 초기화 스크립트 sql 에 create 하였으며, DB Sequence 기반의 Id Generation 을 사용한 예이다. (위에서 select next value for seq_id from xx 는 Hsqldb 의 특화된 sequence 사용 문법임에 유의!)

Step 1. 구동 환경 설정

❑ 7. common설정 확인

- /lab204-ibatis/src/test/resources/META-INF/spring/context-common.xml 을 **확인**한다.

- **PropertyPlaceholderConfigurer 설정 :** 외부 properties 파일을 Container 구동 시 미리 Spring Bean 설정 파일의 속성값으로 대체하여 처리해주는 PropertyPlaceholderConfigurer 설정
- **MessageSource 설정 :** Locale 에 따른 다국어 처리를 쉽게 해주는 messageSource 설정. 여기서는 전자정부 실행환경의 id generation 서비스와 properties 서비스의 메시지 파일과 업무 어플리케이션을 위한 사용자 메시지(/message/message-common - message-common_en_US.properties, message-common_ko_KR.properties 를 확인할 것) 를 지정하였다.
- **전자정부 TraceHandler 설정 관련 :** exception 처리 Handler 와 유사하게 특정한 상황에서 사용자가 Trace Handler 를 지정하여 사용할 수 있도록 전자정부 프레임워크에서 가이드하고 있는 TraceHandler 설정
- **component-scan 설정 :** 스테레오 타입 Annotation 을 인식하여 Spring bean 으로 자동 등록하기 위한 component-scan 설정

❑ 8. aspect 설정 확인 (1/2)

- /lab204-ibatis/src/test/resources/META-INF/spring/context-aspect.xml 를 확인한다.

```
<aop:config>
	<aop:pointcut id="serviceMethod"
	expression="execution(* egovframework.lab..impl.*Impl.*(..))" />

	<aop:aspect ref="exceptionTransfer">
		<aop:after-throwing throwing="exception"
		pointcut-ref="serviceMethod" method="transfer" />
	</aop:aspect>
</aop:config>

<bean id="exceptionTransfer" class="”org.egovframe.rte.fdl.cmmn.aspect.ExceptionTransfer">
	<property name="exceptionHandlerService">
		<list>
		<ref bean="defaultExceptionHandleManager" />
		</list>
	</property>
</bean>

... (계속)
```

Step 1. 구동 환경 설정

❑ 8. aspect 설정 확인 (2/2)

```
... (이어서)

<bean id="defaultExceptionHandleManager"
class=”org.egovframe.rte.fdl.cmmn.exception.manager.DefaultExceptionHandleManager">
      <property name="reqExpMatcher" ref="antPathMater" />
      <property name="patterns">
            <list>
            <value>**service.impl.*</value>
            </list>
      </property>
      <property name="handlers">
            <list>
            <ref bean="egovHandler" />
            </list>
      </property>
</bean>


<bean id="egovHandler"
class="egovframework.lab.dataaccess.common.JdbcLoggingExcepHndlr" />
```

- Spring AOP(xml 설정 방식) 를 사용하여 비지니스 메서드에서 exception 이 발생한 경우 일괄적으로 ExceptionTransfer 의 transfer 메서드 기능(Advice) 를 수행해 주게 됨. --> Exception logging 및 BizException 형태로 wrapping 하여 재처리하는 Exception 공통처리 후 ExceptionHandleManager 에 의해 관리(설정) 되는 Handler (ex. exception 내용을 메일링 한더던지.. 사용자 구현 가능) 가 자동적으로 추가 수행될 수 있음.

Step 2. 자바 클래스 작성

❑ 1. Service Interface 확인

- /lab204-ibatis/src/main/java/egovframework/lab/dataaccess/service/EmpService.java 를 **확인**한다.

❑ 2. VO 확인

- /lab204-ibatis/src/main/java/egovframework/lab/dataaccess/service/EmpVO.java 를 **확인**한다.
- (참고) Getter와 Setter 메서드는 다음과 같이 생성할 수 있다.

  우클릭 > Source > Generate getters and setters > Select All 선택 > OK

Step 2. 자바 클래스 작성

❑ 3. Annotation을 적용한 Impl 작성

– /lab204-ibatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpServiceImpl.java 에 아래 내용을 추가한다.

```
@Service("empService")
public class EmpServiceImpl extends EgovAbstractServiceImpl implements EmpService {

    // TODO [Step 2-3] EmpServiceImpl 작성 추가

    @Resource(name = "primaryTypeSequenceIds")
    EgovIdGnrService egovIdGnrService;

    @Resource(name = "empDAO")
    private EmpDAO empDAO;
```

– Ctrl + Shift + O 를 눌러서 import 되지 않은 클래스를 import 시킨다.

– 위 방법을 통해서도 import 되지 않은 클래스가 남아있으면, 이클립스 상단에 Project > Clean을 수행한다.

Step 2. 자바 클래스 작성

❑ 4. DAO 작성

- /lab204-ibatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpDAO.java 를 작성한다.

```
@Repository("empDAO")
public class EmpDAO extends EgovAbstractDAO {

// TODO [Step 2-4] EmpDAO 작성
    public void insertEmp(EmpVO vo) {
        insert("insertEmp", vo);
    }
    public int updateEmp(EmpVO vo) {
        return update("updateEmp", vo);
    }
    public int deleteEmp(EmpVO vo) {
        return delete("deleteEmp", vo);
    }
    public EmpVO selectEmp(EmpVO vo) {
        return (EmpVO) select ("selectEmp", vo);
        //return (EmpVO) select ("selectEmpUsingCacheModelLRU", vo);
    }
    @SuppressWarnings("unchecked")
    public List<EmpVO> selectEmpList(EmpVO searchVO) {
        return (List<EmpVO>) list("selectEmpList", searchVO);
    }
}
```

Step 2. SQL 매핑 파일 작성

❑ 5. mapping xml 작성 (1/6)

- /lab204-ibatis/src/test/resources/META-INF/sqlmap/mappings/lab-emp.xml 를 작성한다.

```
<sqlMap namespace="Emp">
<!-- TODO [Step 2-5] lab-emp.xml mapping xml 작성 -->

<typeAlias alias="empVO" type="egovframework.lab.dataaccess.service.EmpVO" />

<resultMap id="empResult" class="empVO">
<result property="empNo" column="EMP_NO" />
<result property="empName" column="EMP_NAME" />
<result property="job" column="JOB" />
<result property="mgr" column="MGR" />
<result property="hireDate" column="HIRE_DATE" />
<result property="sal" column="SAL" />
<result property="comm" column="COMM" />
<result property="deptNo" column="DEPT_NO" />
</resultMap>
```

- 다음 슬라이드에서 계속...

Step 2. SQL 매핑 파일 작성

❑ 5. mapping xml 작성 (2/6)

- /lab204-ibatis/src/test/resources/META-INF/sqlmap/mappings/lab-emp.xml 를 작성한다.

```
<insert id="insertEmp" parameterClass="empVO">
<![CDATA[
insert into EMP
   (EMP_NO,
     EMP_NAME,
     JOB,
     MGR,
     HIRE_DATE,
     SAL,
     COMM,
     DEPT_NO)
     values   (#empNo#,
     #empName#,
     #job#,
     #mgr#,
     #hireDate#,
     #sal#,
     #comm#,
     #deptNo#)
]]>
</insert>
```

- 다음 슬라이드에서 계속...

Step 2. SQL 매핑 파일 작성

❑ 5. mapping xml 작성 (3/6)

- /lab204-ibatis/src/test/resources/META-INF/sqlmap/mappings/lab-emp.xml 를 작성한다.

```
<update id="updateEmp" parameterClass="empVO">
<![CDATA[
update EMP
   set EMP_NAME= #empName#,
   JOB = #job#,
   MGR = #mgr#,
   HIRE_DATE = #hireDate#,
   SAL = #sal#,
   COMM = #comm#,
   DEPT_NO = #deptNo#
 where EMP_NO = #empNo#
]]>
</update>
```

- 다음 슬라이드에서 계속...

Step 2. SQL 매핑 파일 작성

❑ 5. mapping xml 작성 (4/6)

- /lab204-ibatis/src/test/resources/META-INF/sqlmap/mappings/lab-emp.xml 를 작성한다.

```
<delete id="deleteEmp" parameterClass="empVO">
<![CDATA[
delete from EMP
 where EMP_NO = #empNo#
]]>
</delete>

<select id="selectEmp" parameterClass="empVO" resultMap="empResult">
<![CDATA[
select EMP_NO,
   EMP_NAME,
   JOB,
   MGR,
   HIRE_DATE,
   SAL,
   COMM,
   DEPT_NO
  from EMP
 where EMP_NO = #empNo#
]]>
</select>
```

- 다음 슬라이드에서 계속...

Step 2. SQL 매핑 파일 작성

❑ 5. mapping xml 작성 (5/6)

- /lab204-ibatis/src/test/resources/META-INF/sqlmap/mappings/lab-emp.xml 를 작성한다.

```
<select id="selectEmpList" parameterClass="empVO" resultMap="empResult">
<![CDATA[
select EMP_NO,
   EMP_NAME,
   JOB,
   MGR,
   HIRE_DATE,
   SAL,
   COMM,
   DEPT_NO
  from EMP
 where 1 = 1
]]>
<isNotNull prepend="and" property="empNo">
EMP_NO = #empNo#
</isNotNull>
<isNotNull prepend="and" property="empName">
EMP_NAME LIKE '%' || #empName# || '%'
</isNotNull>
</select>

</sqlMap>
```

Step 3. 테스트 케이스 확인

❑ 1. EmpServiceTest 확인

– /lab204-ibatis/src/test/java/egovframework/lab/dataaccess/service/EmpServiceTest.java 를 확인한다.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath*:META-INF/spring/context-*" })
@Transactional(transactionManager = "txManager")
@Commit
public class EmpServiceTest {

    // TODO [Step 3-1] BasicCRUDTest 실행

    @Resource(name = "dataSource")
    DataSource dataSource;

    @Resource(name = "empService")
    EmpService empService;

    @Before
    public void onSetUp() throws Exception {
        // 테스트 편의상 매 테스트메서드 수행전 외부의 sql file 로부터 DB 초기화
        // (기존 테이블 삭제/생성)
        JdbcTestUtils.executeSqlScript(new JdbcTemplate(dataSource), new ClassPathResource("META-
INF/testdata/sample_schema_hsql.sql"), true);
    }


    public EmpVO makeVO() throws ParseException {
        EmpVO vo = new EmpVO();

        // empNo 는 Biz. 서비스 내에서 id generation service 에
        // 의해 key 를 따고 설정할 것임.

        ... (계속)
```

Step 3. 테스트 케이스 확인

```
        ... (이어서)
        vo.setEmpName("홍길동");
        vo.setJob("개발자");
        vo.setMgr(new BigDecimal(7902));
        SimpleDateFormat sdf =
            new SimpleDateFormat("yyyy-MM-dd", java.util.Locale.getDefault());
        vo.setHireDate(sdf.parse("2009-07-09"));
        vo.setSal(new BigDecimal(1000));
        vo.setComm(new BigDecimal(0));
        vo.setDeptNo(new BigDecimal(20));

        return vo;
    }

    public void checkResult(EmpVO vo, EmpVO resultVO) {
        assertNotNull(resultVO);
        assertEquals(vo.getEmpNo(), resultVO.getEmpNo());
        assertEquals(vo.getEmpName(), resultVO.getEmpName());
        assertEquals(vo.getJob(), resultVO.getJob());
        assertEquals(vo.getMgr(), resultVO.getMgr());
        assertEquals(vo.getHireDate(), resultVO.getHireDate());
        assertEquals(vo.getSal(), resultVO.getSal());
        assertEquals(vo.getComm(), resultVO.getComm());
        assertEquals(vo.getDeptNo(), resultVO.getDeptNo());
    }

    @Test
    public void testInsertEmp() throws Exception {
        EmpVO vo = makeVO();
        // insert
        BigDecimal empNo = empService.insertEmp(vo);
        vo.setEmpNo(empNo);
        // select
        EmpVO resultVO = empService.selectEmp(vo);
        // check
        checkResult(vo, resultVO);
    }
     ... (계속)
```

Step 3. 테스트 케이스 확인

```
    ... (이어서)
@Test
public void testUpdateEmp() throws Exception {
    EmpVO vo = makeVO();

    // insert
    BigDecimal empNo = empService.insertEmp(vo);
    vo.setEmpNo(empNo);

    // data change
    vo.setEmpName("홍길순");
    vo.setJob("설계자");

    // update
    empService.updateEmp(vo);

    // select
    EmpVO resultVO = empService.selectEmp(vo);

    // check
    checkResult(vo, resultVO);
}

@Test
public void testDeleteEmp() throws Exception {
    EmpVO vo = makeVO();

    // insert
    BigDecimal empNo = empService.insertEmp(vo);
    vo.setEmpNo(empNo);

    // delete
    empService.deleteEmp(vo);

    ... (계속)
```

Step 3. 테스트 케이스 확인

```
        ... (이어서)
        // select
        try {
            @SuppressWarnings("unused")
            EmpVO resultVO = empService.selectEmp(vo);
            fail("EgovBizException 이 발생해야 합니다.");
        } catch (Exception e) {
            assertNotNull(e);
            // 여기서는 비지니스 단에서 명시적으로 exception 처리 하였음.
            // AbstractServiceImpl 을 extends 하고
            // processException("info.nodata.msg"); 과 같이 메서드 콜 형태로 처리
            assertTrue(e instanceof EgovBizException);
            assertEquals("info.nodata.msg", ((EgovBizException) e)
                .getMessageKey());
            assertEquals("해당 데이터가 없습니다.", e.getMessage());
        }
    }

    @Test
    public void testSelectEmpList() throws Exception {
        EmpVO vo = makeVO();
        // insert
        BigDecimal empNo = empService.insertEmp(vo);
        vo.setEmpNo(empNo);

        // 검색조건으로 key 설정
        EmpVO searchVO = new EmpVO();
        searchVO.setEmpNo(vo.getEmpNo());

        // selectList
        List<EmpVO> resultList = empService.selectEmpList(searchVO);

        ... (계속)
```

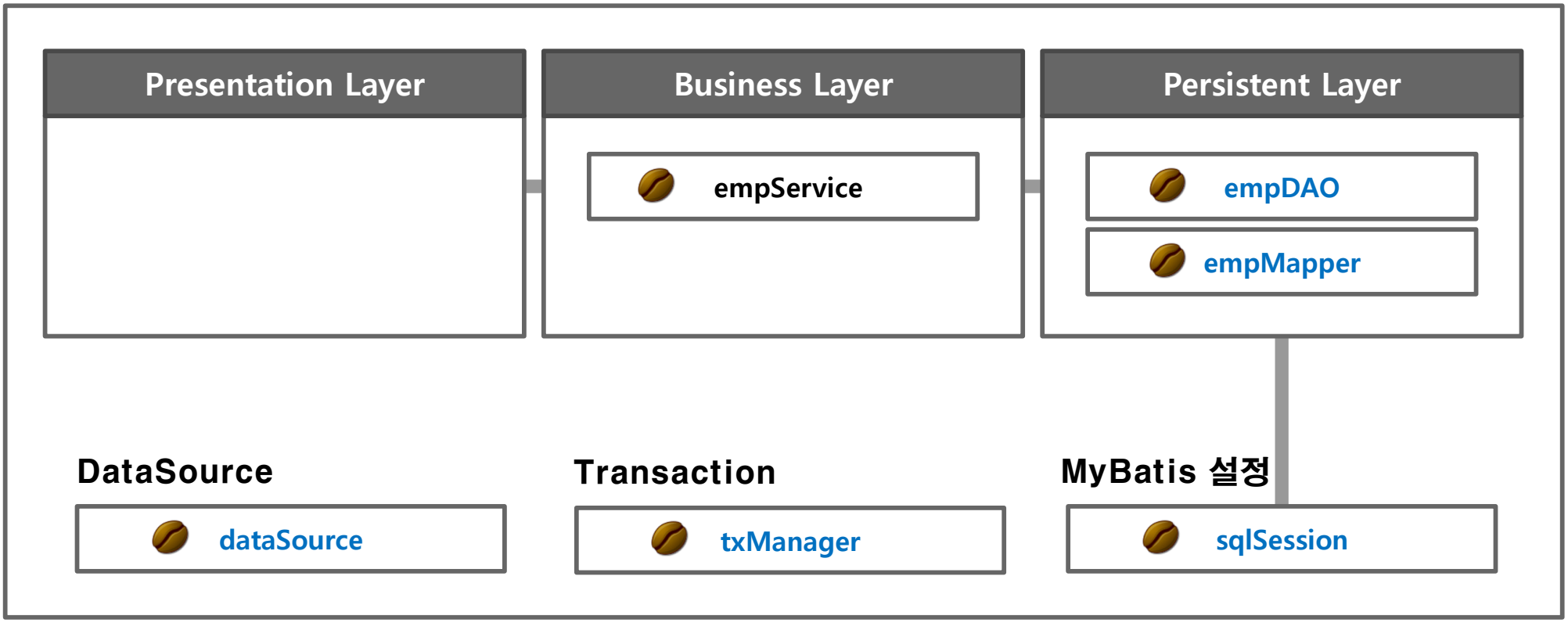
Step 3. 테스트 케이스 확인

```
        ... (이어서)
        // key 조건에 대한 결과는 한건일 것임
        assertNotNull(resultList);
        assertTrue(resultList.size() > 0);
        assertEquals(1, resultList.size());
        checkResult(vo, resultList.get(0));

        // 검색조건으로 name 설정 - '%' || #empName# || '%'
        EmpVO searchVO2 = new EmpVO();
        searchVO2.setEmpName(""); // '%' || '' || '%'
                                  // --> '%%'
        // selectList
        List<EmpVO> resultList2 = empService.selectEmpList(searchVO2);

        // like 조건에 대한 결과는 한건 이상일 것임
        assertNotNull(resultList2);
        assertTrue(resultList2.size() > 0);
    }
}
```

LAB 205-MyBatis 실습

실습개요

❑ lab205-mybatis 프로젝트의 beans



Step 1. 구동 환경 설정

❑ 1. Hsqldb 초기화 스크립트

- /lab205-mybatis/src/test/resources/META-INF/testdata/sample_schema_hsql.sql 를 확인한다.
- 현 실습 프로젝트에서는 편의상 매 테스트 케이스 재실행 시 관련 table 을 drop/create 하고 있음.

❑ 2. dataSource 설정

- <bean id="dataSource" .../>를 이용한 방법
- <jdbc:embedded-database .../>를 이용한 방법

❑ 2. dataSource 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-datasource.xml 에 dataSource 빈 설정을 추가한다.

```
<!-- TODO [Step 1-2] dataSource 설정 -->
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource" destroy-
method="close">
  <property name="driverClassName" value="${db.driver}" />
  <property name="url" value="${db.dburl}" />
  <property name="username" value="${db.username}" />
  <property name="password" value="${db.password}" />
  <property name="defaultAutoCommit" value="false" />
  <property name="poolPreparedStatements" value="true" />
</bean>
```

- 스프링에서 제공하는 EmbeddedDataBase를 사용할 수 있음

```
<!--  TODO [Step 1-2] DataSource 설정 -->
<jdbc:embedded-database id="dataSource" type="HSQL">
  <jdbc:script location= "META-INF/testdata/sample_schema_hsql.sql"/>
</jdbc:embedded-database>
```

❑ 3. transaction 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-transaction.xml 를 작성한다.

```
<!-- TODO [Step 1-3] transaction 설정 -->
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>
<tx:annotation-driven transaction-manager="txManager" />
```

cf.) 여기서는 transaction manager 와 메소드 혹은 클래스 레벨에 @Transactional 을 선언하여 트랜잭션 서비스를 이용한다. @Transactional Anntation 스캔을 위해서는 <tx:annotation-driven />을 선언해야 한다. @Transactional을 이용하면 대상 메소드에 개별적으로 트랜잭션을 지정할 수 있다는 장점이 있으나, 보통 AOP 형식(tx:aop)으로 선언하여 트랜잭션 대상 메서드들에 일괄 지정하는 경우가 많다.

❑ 4. Spring 의 MyBatis 연동 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-mybatis.xml 를 작성한다.

```
<!-- TODO [Step 1-4] MyBaits와 Spring 연동 설정 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
<property name="configLocation" value="classpath:/META-INF/sqlmap/sql-mybatis-config.xml" />
<!-- <property name="mapperLocations" value="classpath:**/lab-*.xml" /> -->
</bean>
```

- 최신 프레임워크 환경에서는 sql-mybatis-config.xml 내에 개별 sql 맵핑 파일을 일일이 지정하는 것이 아니라, 위의 mapperLocations 영역을 주석 해제하여 Spring 의 ResourceLoader 형식으로 패턴 매칭에 의거한 일괄 로딩으로 처리가 가능하다. (단, 테스트 결과 CacheModel 등의 일부 기능에서 문제가 발생하는 경우가 있으므로 사용에 유의할것!!)

❑ 5. MyBatis 의 sql-mybatis-config 설정 파일 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/sql-mybatis-config.xml 를 작성한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
<!-- TODO [Step 1-5] MyBatis Configuration File 작성  -->
<typeAliases>
<typeAlias alias="empVO" type="egovframework.lab.dataaccess.service.EmpVO" />
</typeAliases>

<!-- MyBatis 연동을 위한 SqlSessionFactoryBean 정의 시 mapperLocations 속성으로
한 번에 모든 Mapper XML File을 설정할 수 있다.
(<property name="mapperLocations" value="classpath:**/lab-*.xml" /> 추가)
단, 아래 <mappers> 설정과 mapperLocations 설정 중 한가지만 선택해야 한다.
-->
<mappers>
<mapper resource="META-INF/sqlmap/mappers/lab-dao-class.xml" />
<mapper resource="META-INF/sqlmap/mappers/lab-mapper-interface.xml" />
</mappers>

</configuration>
```

❑ 6. ID Generation Service 설정 확인

- /lab205-mybatis/src/test/resources/META-INF/spring/context-idgen.xml 를 확인한다.

```
<!-- [Step 1-6] Id Generation Service 설정 -->
<bean name="primaryTypeSequenceIds"
class="org.egovframe.rte.fdl.idgnr.impl.EgovSequenceIdGnrService"
destroy-method="destroy">
<property name="dataSource" ref="dataSource" />
<property name="query" value="SELECT NEXT VALUE FOR empseq FROM DUAL" />
</bean>
```

- 여기서는 Hsqldb 를 사용하여 Oracle 의 DUAL 테이블 역할을 할 수 있도록 초기화 스크립트 sql 에 create 하였으며, DB Sequence 기반의 Id Generation 을 사용한 예이다. (위에서 select next value for seq_id from xx 는 Hsqldb 의 특화된 sequence 사용 문법임에 유의!)

Step 1. 구동 환경 설정

❑ 7. common설정 확인

- /lab205-mybatis/src/test/resources/META-INF/spring/context-common.xml 을 **확인**한다.

- **PropertyPlaceholderConfigurer 설정 :** 외부 properties 파일을 Container 구동 시 미리 Spring Bean 설정 파일의 속성값으로 대체하여 처리해주는 PropertyPlaceholderConfigurer 설정
- **MessageSource 설정 :** Locale 에 따른 다국어 처리를 쉽게 해주는 messageSource 설정. 여기서는 전자정부 실행환경의 id generation 서비스와 properties 서비스의 메시지 파일과 업무 어플리케이션을 위한 사용자 메시지(/message/message-common - message-common_en_US.properties, message-common_ko_KR.properties 를 확인할 것) 를 지정하였다.
- **전자정부 TraceHandler 설정 관련 :** exception 처리 Handler 와 유사하게 특정한 상황에서 사용자가 Trace Handler 를 지정하여 사용할 수 있도록 전자정부 프레임워크에서 가이드하고 있는 TraceHandler 설정
- **component-scan 설정 :** 스테레오 타입 Annotation 을 인식하여 Spring bean 으로 자동 등록하기 위한 component-scan 설정

❑ 8. aspect 설정 확인 (1/2)

- /lab205-mybatis/src/test/resources/META-INF/spring/context-aspect.xml 를 확인한다.

```
<aop:config>
	<aop:pointcut id="serviceMethod"
	expression="execution(* egovframework.lab..impl.*Impl.*(..))" />

	<aop:aspect ref="exceptionTransfer">
		<aop:after-throwing throwing="exception"
		pointcut-ref="serviceMethod" method="transfer" />
	</aop:aspect>
</aop:config>

<bean id="exceptionTransfer" class="org.egovframe.rte.fdl.cmmn.aspect.ExceptionTransfer">
	<property name="exceptionHandlerService">
		<list>
		<ref bean="defaultExceptionHandleManager" />
		</list>
	</property>
</bean>

... (계속)
```

Step 1. 구동 환경 설정

❑ 8. aspect 설정 확인 (2/2)

```
... (이어서)

<bean id="defaultExceptionHandleManager"
class=”org.egovframe.rte.fdl.cmmn.exception.manager.DefaultExceptionHandleManager">
      <property name="reqExpMatcher" ref="antPathMater" />
      <property name="patterns">
            <list>
            <value>**service.impl.*</value>
            </list>
      </property>
      <property name="handlers">
            <list>
            <ref bean="egovHandler" />
            </list>
      </property>
</bean>


<bean id="egovHandler"
class="egovframework.lab.dataaccess.common.JdbcLoggingExcepHndlr" />
```

- Spring AOP(xml 설정 방식) 를 사용하여 비지니스 메서드에서 exception 이 발생한 경우 일괄적으로 ExceptionTransfer 의 transfer 메서드 기능(Advice) 를 수행해 주게 됨. --> Exception logging 및 BizException 형태로 wrapping 하여 재처리하는 Exception 공통처리 후 ExceptionHandleManager 에 의해 관리(설정) 되는 Handler (ex. exception 내용을 메일링 한더던지.. 사용자 구현 가능) 가 자동적으로 추가 수행될 수 있음.

Step 2. 자바 클래스 작성

❑ 1. Annotation을 적용한 Impl 작성

– /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpServiceImpl.java 에 아래 내용을 추가한다.

```
@Service("empService")
public class EmpServiceImpl extends EgovAbstractServiceImpl implements EmpService {

    // TODO [Step 2-1] EmpServiceImpl 작성 추가

    @Resource(name = "empDAO")
    private EmpDAO empDAO;
    // TODO [Step 4-2] EmpServiceImpl의 주석 EmpMapper -> EmpDAO 변경 후 다시 테스트

    // TODO [Step 3-1] EmpServiceImpl 변경
    // EmpMapper를 사용하도록 주석 변경
    // @Resource(name = "empMapper")
    // EmpMapper empDAO;
```

– Ctrl + Shift + O 를 눌러서 import 되지 않은 클래스를 import 시킨다.

– 위 방법을 통해서도 import 되지 않은 클래스가 남아있으면, 이클립스 상단에 Project > Clean을 수행한다.

Step 2. 자바 클래스 작성

❑ 2. DAO 작성

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpDAO.java 를 작성한다.

```
@Repository("empDAO")
public class EmpDAO extends EgovAbstractMapper {
// TODO [Step 2-2] EmpDAO 작성 (EgovAbstractMapper 상속한 DAO)

    public void insertEmp(EmpVO vo) {
        insert("Emp.insertEmp", vo);
    }
    public int updateEmp(EmpVO vo) {
        return update("Emp.updateEmp", vo);
    }
    public int deleteEmp(EmpVO vo) {
        return delete("Emp.deleteEmp", vo);
    }
    public EmpVO selectEmp(EmpVO vo) {
        return selectOne("Emp.selectEmp", vo);
    }
    @SuppressWarnings("unchecked")
    public List<EmpVO> selectEmpList(EmpVO searchVO) {
        return selectList("Emp.selectEmpList", searchVO);
    }
}
```

Step 2. SQL 매핑 파일 작성

❑ 3. mapping xml 작성 (1/4)

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="Emp">

<!-- TODO [Step 2-3] lab-dao-class.xml 작성 (EgovAbstractMapper 상속한 DAO) -->
<resultMap id="empResult" type="empVO">
<id property="empNo" column="EMP_NO" />
<result property="empName" column="EMP_NAME" />
<result property="job" column="JOB" />
<result property="mgr" column="MGR" />
<result property="hireDate" column="HIRE_DATE" />
<result property="sal" column="SAL" />
<result property="comm" column="COMM" />
<result property="deptNo" column="DEPT_NO" />
</resultMap>
```

- 다음 슬라이드에서 계속...

Step 2. SQL 매핑 파일 작성

❑ 3. mapping xml 작성 (2/4)

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<insert id="insertEmp" parameterType="empVO">
<![CDATA[
insert into EMP (EMP_NO, EMP_NAME, JOB, MGR, HIRE_DATE, SAL, COMM, DEPT_NO)
values(#{empNo}, #{empName}, #{job}, #{mgr}, #{hireDate}, #{sal}, #{comm}, #{deptNo})
]]>
</insert>

<update id="updateEmp" parameterType="empVO">
<![CDATA[
update EMP
set EMP_NAME = #{empName},
JOB = #{job},
MGR = #{mgr},
HIRE_DATE = #{hireDate},
SAL = #{sal},
COMM = #{comm},
DEPT_NO = #{deptNo}
where EMP_NO = #{empNo}
]]>
</update>
```

- 다음 슬라이드에서 계속...

Step 2. SQL 매핑 파일 작성

❑ 3. mapping xml 작성 (3/4)

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<delete id="deleteEmp" parameterType="empVO">
<![CDATA[
delete from EMP
where EMP_NO = #{empNo}
]]>
</delete>

<select id="selectEmp" parameterType="empVO" resultMap="empResult">
<![CDATA[
select EMP_NO, EMP_NAME, JOB, MGR, HIRE_DATE, SAL, COMM, DEPT_NO
from EMP
where EMP_NO = #{empNo}
]]>
</select>
```

- 다음 슬라이드에서 계속...

Step 2. SQL 매핑 파일 작성

❑ 3. mapping xml 작성 (4/4)

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<select id="selectEmpList" parameterType="empVO" resultMap="empResult">
<![CDATA[
Select EMP_NO, EMP_NAME, JOB, MGR, HIRE_DATE, SAL, COMM, DEPT_NO
From EMP
where 1 = 1
]]>
<if test="empNo != null">
AND EMP_NO = #{empNo}
</if>
<if test="empName != null">
AND EMP_NAME LIKE '%' || #{empName} || '%'
</if>
</select>
</mapper>
```

Step 3. 자바 클래스 작성

❑ 1. Service Impl 추가 변경 작성

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpServiceImpl.java 를 작성한다.
- Mapper Interface 방식으로 테스트 시 사용할 DAO 클래스를 미리 작성한다.

  (주석처리를 하여 EmpDAO empDAO와 충돌되지 않도록 한다.)

```
@Service("empService")
public class EmpServiceImpl extends EgovAbstractServiceImpl implements EmpService {

    ...

    // TODO [Step 3-1] EmpServiceImpl 추가 작성
    // EmpMapper를 사용하도록 주석 변경
    // @Resource(name = "empMapper")
    // EmpMapper empDAO;
```

Step 3. 자바 클래스 작성

❑ 2. Mapper Interface 작성

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpMapper.java 를 작성한다.
- 아래와 같이 MyBatis에서는 Dao 클래스 대신 Interface를 이용해 데이터 처리가 가능하다.

```java
@Mapper("empMapper")
public interface EmpMapper {

    // TODO [Step 3-2] EmpMapper 작성 (Mapper Interface)

    public void insertEmp(EmpVO vo);

    public int updateEmp(EmpVO vo);

    public int deleteEmp(EmpVO vo);

    public EmpVO selectEmp(EmpVO vo);

    public List<EmpVO> selectEmpList(EmpVO searchVO);

}
```

Step 3. 자바 클래스 작성

❑ 3. @Mapper 스캔 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-mybatis.xml 에 다음을 추가한다.

```
<!-- MapperConfigurer setup for @Mapper -->
<!-- TODO [Step 3-3] MyBatis의 Mapper Interface 자동스캔 설정 -->

<bean class=”org.egovframe.rte.psl.dataaccess.mapper.MapperConfigurer">
<property name="basePackage" value="egovframework.lab.dataaccess.service.impl" />
</bean>
```

Step 3. SQL 매핑 파일 작성

❑ 4. mapping xml 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-mapper-interface.xml 를 작성한다.
- /lab-dao-class.xml과 <mapper>의 name 속성값만 다르다.

```
<mapper namespace="egovframework.lab.dataaccess.service.impl.EmpMapper">

<!-- TODO [Step 3-4] lab-mapper-interface.xml 작성 (Mapper Interface)
실습교재 p.45(<resultMap>부터) ~ p.48까지 내용을 동일하게 작성
-->
```

- DAO 클래스의 Statement 호출 방식: 사용자가 직접 지정해준 ID 파라미터 값과 일치하는 Statement를 호출. 동일한 Statement ID가 있으면, <mapper>의 namespace를 지정한다.
  - namespace=A, statement id=insertEmp → A.insertEmp으로 호출
  - namespace=B, statement id=insertEmp → B.insertEmp으로 호출
- Mapper 인터페이스의 Statement 호출 방식: 메소드명과 일치하는 Statement를 자동 호출. 이 때 MyBatis는 호출된 메서드가 포함된 인터페이스의 풀네임을 namespace 값으로 사용하기 때문에, 반드시 namesapce 값을 지정해주어야 한다.
  - namespace=x.y.z.EmpMapper, statement id=insertEmp → 내부적으로 x.y.z.EmpMapper.insertEmp을 호출

Step 4. 테스트 케이스 확인

❑ 1. EmpServiceTest 확인 – EmpDAO 테스트

– /lab205-mybatis/src/test/java/egovframework/lab/dataaccess/service/EmpServiceTest.java 를 확인하고 실행한다.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath*:META-INF/spring/context-*" })
@Transactional(transactionManager = "txManager")
@Commit
public class EmpServiceTest {

// TODO [Step 4-1] EmpServiceTest 실행
@Resource(name = "dataSource")
DataSource dataSource;

@Resource(name = "empService")
EmpService empService;

@Before
public void onSetUp() throws Exception {
// 편의상 각 테스트 메서드 수행 전에
// 외부의 스크립트 파일(sample_schema_hsql.sql)로 DB를 초기화하도록 설정
JdbcTestUtils.executeSqlScript(new JdbcTemplate(dataSource), new ClassPathResource("META-
INF/testdata/sample_schema_hsql.sql"), true);
}
```

Step 4. 테스트 케이스 확인

```
/**
 * 사원정보 생성
 *
 * @throws ParseException
 */
public EmpVO makeVO() throws ParseException {
EmpVO vo = new EmpVO();

// empNo는 Biz. 서비스 내에서 IDGeneration Service 에 의해 key를 생성하고 설정
vo.setEmpName("홍길동");
vo.setJob("개발자");
vo.setMgr(new BigDecimal(7902));
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd",
java.util.Locale.getDefault());
vo.setHireDate(sdf.parse("2009-07-09"));
vo.setSal(new BigDecimal(1000));
vo.setComm(new BigDecimal(0));
vo.setDeptNo(new BigDecimal(20));

return vo;
}

public void checkResult(EmpVO vo, EmpVO resultVO) {
assertNotNull(resultVO);
assertEquals(vo.getEmpNo(), resultVO.getEmpNo());
assertEquals(vo.getEmpName(), resultVO.getEmpName());
assertEquals(vo.getJob(), resultVO.getJob());
assertEquals(vo.getMgr(), resultVO.getMgr());
assertEquals(vo.getHireDate(), resultVO.getHireDate());
assertEquals(vo.getSal(), resultVO.getSal());
assertEquals(vo.getComm(), resultVO.getComm());
assertEquals(vo.getDeptNo(), resultVO.getDeptNo());
}
```

Step 4. 테스트 케이스 확인

```
/** 사원정보 입력 */
@Test
public void testInsertEmp() throws Exception {
EmpVO vo = makeVO();

// insert
BigDecimal empNo = empService.insertEmp(vo);
vo.setEmpNo(empNo);

// select
EmpVO resultVO = empService.selectEmp(vo);

// check
checkResult(vo, resultVO);
}
/** 사원정보 수정 */
@Test
public void testUpdateEmp() throws Exception {
EmpVO vo = makeVO();

// insert
BigDecimal empNo = empService.insertEmp(vo);
vo.setEmpNo(empNo);

// data change
vo.setEmpName("홍길순");
vo.setJob("설계자");

// update
empService.updateEmp(vo);

// select
EmpVO resultVO = empService.selectEmp(vo);

// check
checkResult(vo, resultVO);
}
```

Step 4. 테스트 케이스 확인

```
/** 사원정보 삭제 */
@Test
public void testDeleteEmp() throws Exception {
EmpVO vo = makeVO();

// insert
BigDecimal empNo = empService.insertEmp(vo);
vo.setEmpNo(empNo);

// delete
empService.deleteEmp(vo);

// select
try {
@SuppressWarnings("unused")
EmpVO resultVO = empService.selectEmp(vo);
fail("EgovBizException 이 발생해야 합니다.");
} catch (Exception e) {
assertNotNull(e);
// 여기서는 비지니스 단에서 명시적으로 exception 처리하였음
// AbstractServiceImpl 을 extends 하고
// processException("info.nodata.msg"); 과 같이 메서드 콜 형태로 처리
assertTrue(e instanceof EgovBizException);
assertEquals("info.nodata.msg",
((EgovBizException) e).getMessageKey());
assertEquals("해당 데이터가 없습니다.", e.getMessage());
}
}
```

Step 4. 테스트 케이스 확인

```java
/** 사원정보 목록조회 */
@Test
public void testSelectEmpList() throws Exception {
EmpVO vo = makeVO();

// insert
BigDecimal empNo = empService.insertEmp(vo);
vo.setEmpNo(empNo);

// 검색조건으로 empNo 설정
EmpVO searchVO = new EmpVO();
searchVO.setEmpNo(vo.getEmpNo());

// selectList
List<EmpVO> resultList = empService.selectEmpList(searchVO);

// empNo 조건에 대한 결과는 1건일 것임
assertNotNull(resultList);
assertTrue(resultList.size() > 0);
assertEquals(1, resultList.size());
checkResult(vo, resultList.get(0));

// 검색조건으로 empName 설정 - '%' || #{empName} || '%'
EmpVO searchVO2 = new EmpVO();
searchVO2.setEmpName(""); // '%' || '' || '%' // --> '%%'

// selectList
List<EmpVO> resultList2 = empService.selectEmpList(searchVO2);

// like 조건에 대한 결과는 1건 이상일 것임
assertNotNull(resultList2);
assertTrue(resultList2.size() > 0);
}

}
```

Step 4. 테스트 케이스 확인

❑ 2. EmpServiceTest 확인 – EmpMapper 테스트

– /lab205-mybatis/src/test/java/egovframework/lab/dataaccess/service/impl/EmpServiceImpl.java 를 다음과 같이 수정한 후, EmpServiceTest를 다시 실행해본다.

```
@Service("empService")
public class EmpServiceImpl extends EgovAbstractServiceImpl implements EmpService {

    // TODO [Step 2-1] EmpServiceImpl 추가 작성
    // @Resource(name = "empDAO")
    // public EmpDAO empDAO;


    // TODO [Step 4-2] EmpServiceImpl 변경
    // EmpMapper를 사용하도록 주석 변경
    @Resource(name = "empMapper")
    EmpMapper empDAO;
```

전자정부 표준프레임워크

실행환경(데이터처리) – 별첨

eGovFrame

Contents

❑ Spring Data

- 데이터베이스 관련 많은 하위 프로젝트를 포함하는 오픈 소스 프로젝트
- non-relational databases, map-reduce frameworks, and cloud based data services 등의 새로운 데이터 액세스 기술을 보다 쉽게 사용 할 수 있는 기능 제공

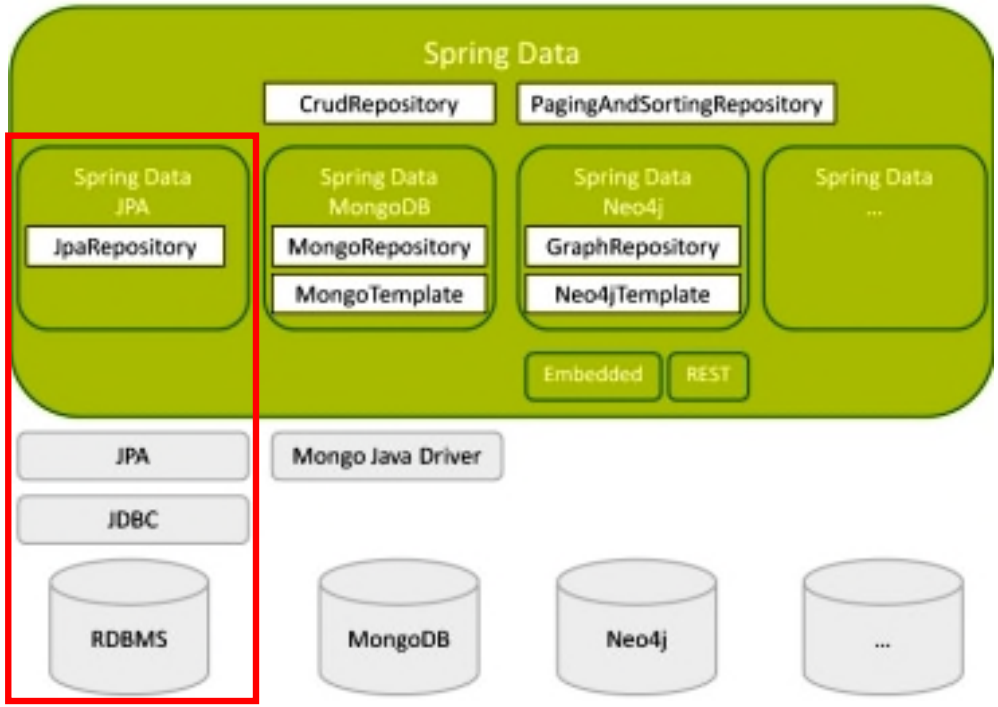
❑ Spring Data Project

Category	Sub project	Description
Relational Databases	JPA	SpringData JPA - JPA 기반 데이터 액세스 계층 생성 개발을 단순화합니다.
	JDBC Extensions	Oracle RAC, Advanced Queuing 및 Advanced 데이터 유형을 지원합니다. JdbcTemplate에서 QueryDSL 사용을 지원합니다.
Big Data	Apache Hadoop	Apache Hadoop 프로젝트는 안정적이고 확장가능한 분산컴퓨팅 및 데이터 스토리지를 위한 오픈소스 프레임워크입니다.
Data-Grid	GemFire	VMware vFabric GemFire는 동적 확장성, 고성능 및 데이터베이스와 유사한 지속성을 제공하는 분산데이터 관리 플랫폼입니다. 복제, 파티셔닝, 데이터 인식 라우팅 및 연속 쿼리와 같은 고급 기술을 결합합니다.
HTTP	REST	Spring Data REST - HTTP 및 SpringData Repository를 사용하여 지속성 모델의 CRUD 작업을 수행합니다.
Key Value Stores	Redis	Redis는 메모리 기반의 “키-값” 구조 데이터 관리 시스템이며, 모든 데이터를 메모리에 저장하고 조회하기에 빠른 Read, Write 속도를 보장하는 비 관계형 데이터베이스입니다.
Document Stores	MongoDB	MongoDB는 확장 가능한 고성능 오픈 소스 문서 지향 데이터베이스입니다.
Graph Databases	Neo4j	Neo4j는 그래프로 구성된 데이터를 데이터를 저장하고 관리하기 위한 그래프 데이터베이스입니다.
Column Stores	HBase	Apache HBase는 Google의 Bigtable을 모델로 한 오픈 소스, 분산, 버전 관리, 열 지향 스토어입니다. HBase 기능은 Spring for Apache Hadoop 프로젝트의 일부입니다.
Common Infrastructure	Commons	다양한 데이터 액세스 프로젝트에서 사용할 수있는 공유 인프라를 제공합니다. 데이터베이스간 지속성에 대한 일반 지원은 여기에 있습니다.

❑ Spring Data 특징

- DB 종류에 관계 없이 데이터 처리가 가능하도록 Persistence Layer를 추상화
- Repository라는 인터페이스를 통해 데이터 처리에 기본적으로 사용되는 CRUD나 pagination, sorting 과 같은 오퍼레이션을 제공
- 개발자는 기본 Repository Interface를 상속받아 정해진 키워드로 오퍼레이션을 쉽게 작성 가능 (Spring Data가 런타임 시에 구현체를 생성)

❑ Spring Data Architecture



❑ CrudRepository

- CRUD 오퍼레이션을 제공하는 인터페이스

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
<S extends T> S save(S entity);----------------전달받은 엔터티 저장
T findOne(ID primaryKey);----------------------전달된 ID로 식별한 엔터티 리턴
Iterable<T> findAll();---------------------------모든 엔터티를 리턴
Long count();-----------------------------------엔터티의 갯수를 리턴
void delete(T entity);---------------------------전달받은 엔터티를 삭제
boolean exists(ID primaryKey);----------------- 전달된 ID에 해당하는 엔터티의 존재여부를 리턴
}
```

❑ PagingAndSortingRepository

- Pagination과 Sorting 오퍼레이션을 제공하는 인터페이스

```
public interface PagingAndSortingRepository<T, ID extends Serializable> extends CrudRepository<T, ID> {
        Iterable<T> findAll(Sort sort);
        Page<T> findAll(Pageable pageable);
}
```

❑ Repository Interface 정의

- Repository, CrudRepository, PagingAndSortingRepository 중 하나를 상속받아 인터페이스 작성
- @RepositoryDefinition를 사용하면 제공되는 Repository를 상속받지 않고도 정의 가능
- 인터페이스 내 오퍼레이션은 정해진 키워드에 따라 작성 = 쿼리 메서드

```
public interface UserRepository extends Repository<User, Long> {
List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
```

- 위의 JPA 표준 API는 정의된 메소드를 다음 Query로 변경

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

❑ Query Method 정의

- 정해진 키워드에 따라 작성된 메서드명으로부터 쿼리를 생성

- 지원하는 키워드와 예제

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false

❑ @Query

- @Query 어노테이션을 사용하여 쿼리를 직접 선언하는 방법
- 메서드와 쿼리문이 결합하여 동작하는 구조
- 메서드를 호출하고 파라미터를 바인딩하는 즉시 선언한 쿼리가 실행
- 파라미터 바인딩은 '?숫자' 혹은 @Param을 통해 가능

```
public interface UserRepository extends JpaRepository<User, Long> {
    @Query("select u from User u where u.emailAddress = ?1")
    User findByEmailAddress(String emailAddress);
}
```

```
public interface UserRepository extends JpaRepository<User, Long> {
    @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")
    User findByLastnameOrFirstname(@Param("lastname") String lastname,
        @Param("firstname") String firstname);
}
```

❑ 서비스 개요

- 객체 모델과 관계형 데이터베이스 간의 매핑 기능인 ORM(Object-Relational Mapping) 기능을 제공함으로써, **SQL이 아닌 객체을 이용한 업무 로직의 작성이 가능하도록 지원**

❑ 주요 기능

- **객체와 관계형 데이터베이스 테이블 간의 매핑**
  - 프레임워크 설정정보에 저장된 **ORM 매핑정보를 이용**하여 객체와 관계형 데이터베이스 테이블간의 매핑 지원
- 객체 로딩
  - 객체와 매핑되는 관계형 데이터베이스의 값을 읽어와 객체의 속성 값으로 설정함
- 객체 저장
  - 저장하고자 하는 객체의 속성 값을 객체와 매핑되는 관계형 데이터베이스에 저장
- 다양한 연관 관계 지원
  - 객체와 객체 간의 1:1, 1:n, n:n 등의 다양한 연관 관계를 지원
  - 객체의 로딩 및 저장 시, 연관 관계를 맺고 있는 객체도 로딩 및 저장 지원
- Caching
  - 객체에 대한 Cache 기능을 지원하여 성능을 향상시킴

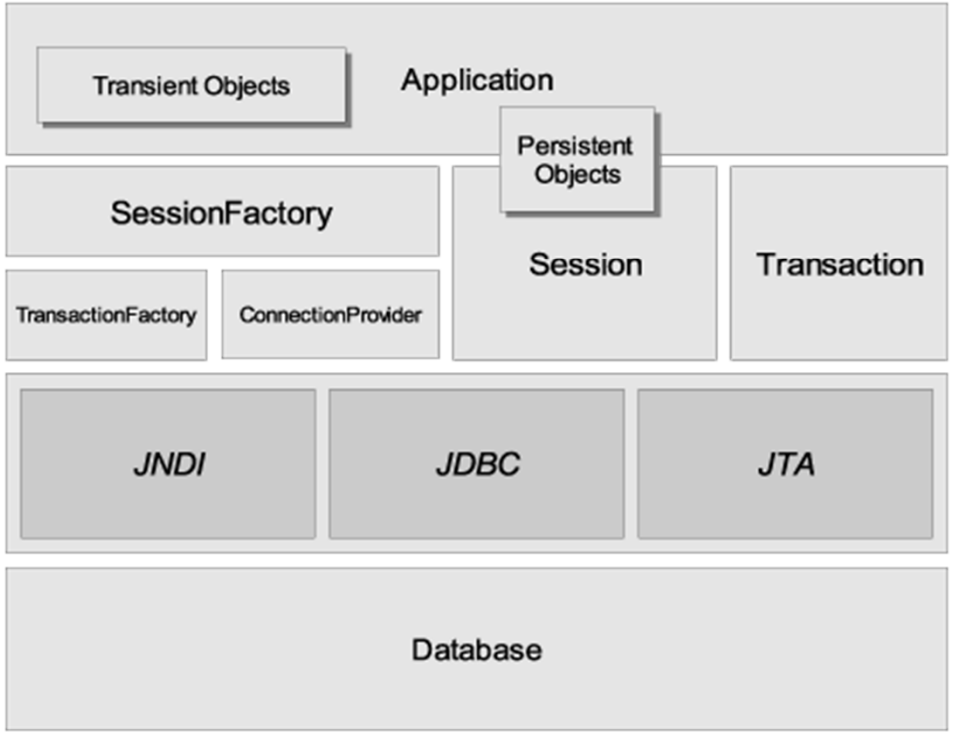
❑ Hibernate는 자바 객체와 관계형 데이터 모델간의 매핑을 위한 도구이며 쿼리 서비스를 지원하는 강력한 고성능의 퍼시스턴스 프레임워크임

- 관계형 데이터 모델에 대한 객체지향 관점을 제공하는 객체/관계 매핑(Object Relational Mapping) 프레임워크
- Gavin King (JBoss, 현재 Red Hat)을 중심으로 한 소프트웨어 개발팀에 의해 개발됨.

분류 및 성숙도 평가*	설명
라이선스	LGPL (Lesser GNU Public License)
기능성 (Functionality)	✓✓✓✓ (중대형 규모의 기업의 기능적인 요구사항을 충족시킴)
커뮤니티 (Community)	✸✸✸ (개발, 오류 보고, 수정 등의 활발한 커뮤니티 활동이 있음)
성숙도 (Maturity)	★★★★ (강력하며 높은 품질의 안정적이며 우수한 성능을 충족함)
적용성 (ER-Rating)	◆◆◆ (프레임워크가 성숙하여 기업 환경에 즉시 반영 가능함)
트렌드 (Trend)	↗ (평가 Criteria 전반적으로 발전하고 있으며, 중요도가 커지고 있음)

* Open Source Catalogue 2007, Optaros (Hibernate 3.2 기준)

❑ Hibernate는 J2EE 표준인 JNDI, JDBC, JTA를 기반으로 객체 관계형 매핑(OR Mapping), 데이터베이스 연결 및 트랜잭션 관리 기능 등을 제공함

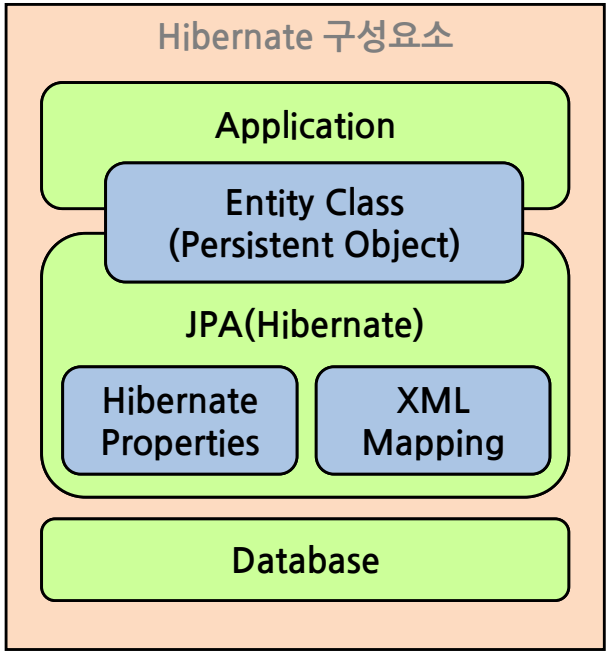


아키텍처 구성요소	설명
SessionFactory	▪ 단일 데이터베이스에 대한 캐시로서, Session에 대한 팩토리 기능을 제공한다.
Session	▪ 어플리케이션과 영속 저장소 사이의 연결를 표현하는 객체로서, JDBC 커넥션을 Wrapping한다. Transaction에 대한 팩토리 기능을 제공한다.
Persistent Objects and Collections	▪ Session과 연관되어 있는 영속(persistent) 상태의 객체로서, 일반적인 JavaBeans/POJO이다. Session이 닫히면, Session과 분리되어 Application 내에서 자유롭게 사용할 수 있게 된다.
Transient and Detached Objects and Collections	▪ Session과 연관되어 있지 않은 영속 클래스들의 인스턴스로서. 어플리케이션에 의해 초기화된 후 영속화 되지 않았거나, 닫혀진Session에 의해 초기화 되었을 수도 있다.
Transaction	▪ 작업의 완전성을 보장하기 위한 어플리케이션에 의해 사용되는 객체이다.
ConnectionProvider	▪ JDBC 커넥션들에 대한 팩토리 기능을 제공한다.
TransactionFactory	▪ Transaction 인스턴스들에 대한 팩토리 기능을 제공한다.

❑ Hibernate 특징

- 객체 모델링(Object Oriented Modeling)과 관계형 데이터 모델링(Relational Data Modeling) 사이의 불일치를 해결해주는 OR Mapping 서비스
- **특정 DBMS에 영향을 받지 않으므로 DBMS가 변경되더라도 데이터 액세스 처리 코드에 대한 변경 없이 설정 정보의 변경만으로도 동작 가능**
- 별도의 XML 파일로 매핑을 관리하지 않고 **Entity Class에 최소한의 Annotation으로 정의함으로써 작업이 용이**함
- SQL 실행 결과로부터 전달하고자 하는 객체로 변경하는 **코드 작성 시간 감소** (필요 시 SQL 사용도 가능)
- 기본적으로 필요 시점에만 DBMS에 접근하는 **Lazy Loading 전략**을 채택하고 Cache활용을 통해 DBMS에 대한 접근 횟수를 줄여나가 어플리케이션의 성능을 향상 시킴
- **Entity Class가 일반 클래스로 정의됨으로써 상속이나 다양성, 캡슐화 같은 것들을 그대로 적용하면서 퍼시스턴스 오브젝트로 사용 가능**
- **자바 표준이므로 많은 벤더들에서 구현체를 지원**하고 개발을 편리하게 할 수 있는 JPA툴(Dali)을 지원함
- SQL을 이용하여 처리하는 방식에 익숙한 개발자가 사용하려면 학습이 필요하고 이에 따른 장벽이 존재함

❑ Hibernate 구성요소



- Entity Class (Persistenct Object)
  - **객체의 변수와 테이블의 컬럼 간 관계 정보가 정의된 클래스**
  - Annotation으로 컬럼명, 테이블 관계 등의 정보를 Entity 클래스에서 직접 정의
  - 어플리케이션 실행 여부와 상관없이 물리적으로 존재하는 데이터들을 다룸
  - **데이터 처리시 Entity Class를 중심으로 어플리케이션의 데이터와 DBMS 연동함**
  - XML기반으로 매핑하는 경우, 별도의 XML 파일이 필요
- Hibernate Properties File
  - **파일명: hibernate.cfg.xml or hibernate.properties**
  - **Hibernate 실행과 관련한 공통설정 파일**
  - 구현체에 대한 선언 및 대상 엔티티 클래스 지정 구현체 별 프로퍼티 지정 등을 할 수 있는 설정파일
- Hibernate(JPA 구현체)
  - Hibernate 모듈은 Hibernate Core, Hibernate Annotations, Hibernate EntityManager 등으로 구성
  - **JPA 구성에 필요한 Entity Manager등 구현 클래스를 포함하고 있음**

❑ Entity 클래스 작성

- 네 개의 Attribute와 각각의 getter • setter 메소드로 구성되어 있는 간단한 Entity 클래스를 생성

Entity 클래스 샘플

```
@Entity
public class Department implements Serializable {

  private static final long serialVersionUID = 1L;

  @Id
  private String deptId;

  private String deptName;

  private Date createDate;

  private BigDecimal empCount;

  public String getDeptId() {
    return deptId;
  }
  public void setDeptId(String deptId) {
    this.deptId = deptId;
  }
  ...
}
```

Department가 Entity 클래스임을 명시, 영속성 처리에 사용 가능한 클래스

테이블의 Primary Key 컬럼과 매핑할 변수에 지정

❑ persistence.xml 작성

- Entity 클래스를 가지고 JPA 수행하기 위한 프로퍼티 파일 작성
- 구현체 제공 클래스정보, 엔티티 클래스 정보, DB 접속 정보, 로깅 정보, 테이블 자동 생성 정보 등을 정의함

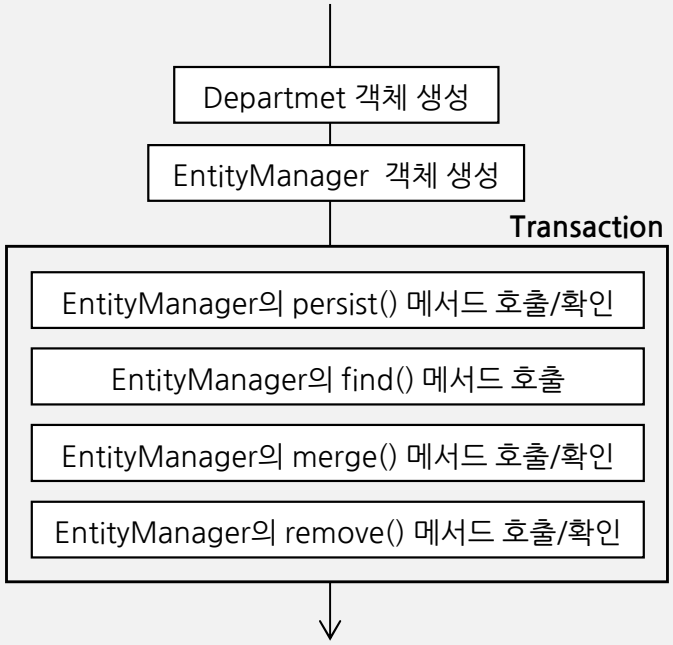
```
<persistence-unit name="PersistUnit" transaction-type="RESOURCE_LOCAL">

   <provider>org.hibernate.ejb.HibernatePersistence</provider>

   <class>egovframework.Department</class>
   <exclude-unlisted-classes />

   <properties>
      <property name="hibernate.connection.driver_class"
         value="org.hsqldb.jdbcDriver" />
      <property name="hibernate.connection.url" value="jdbc:hsqldb:mem:testdb" />
      <property name="hibernate.connection.username" value="sa" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />

      <property name="hibernate.connection.autocommit" value="false" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create" />
   </properties>

</persistence-unit>
```

❑ 테스트 클래스 작성 (1/2)

- Department를 사용하여 입력,수정,조회,삭제 처리를 하는 것을 테스트 케이스로 작성하여 시험

Entity 클래스 테스트 코드

```
@Test
public void testDepartment() throws Exception {

  String modifyName = "Marketing Department";
  String deptId = "DEPT-0001";
  Department department = makeDepartment(deptId);

  // EntityManager 생성
  entityManagerFactory = Persistence.createEntityManagerFactory("PersistUnit");
  entityManager = entityManagerFactory.createEntityManager();

  // 입력
  entityManager .getTransaction().begin();
  entityManager .persist(department);
  entityManager .getTransaction().commit();

  entityManager .getTransaction().begin();
  Department departmentAfterInsert = entityManager .find(Department.class, deptId );

  // 입력 확인
  assertEquals("Department Name Compare!",department.getDeptName(),departmentAfterInsert.getDeptName());
```



❑ 테스트 클래스 작성 (2/2)

- Department를 사용하여 입력,수정,조회,삭제 처리를 하는 것을 테스트 케이스로 작성하여 시험

```
    // 수정
    departmentAfterInsert.setDeptName(modifyName);
    entityManager.merge(departmentAfterInsert);
    entityManager.getTransaction().commit();

    em.getTransaction().begin();
    Department departmentAfterUpdate = em.find(Department.class, deptId );

    // 수정 확인
    assertEquals("Department Modify Name Compare!",modifyName,departmentAfterUpdate.getDeptName());

    // 삭제
    entityManager.remove(departmentAfterUpdate);
    entityManager.getTransaction().commit();

    // 삭제 확인
    Department departmentAfterDelete = em.find(Department.class, deptId );
    assertNull("Department is Deleted!",departmentAfterDelete);

    entityManager.close();
}
```

❑ Entities (1/3)

- ORM 서비스를 구성하는 가장 기초적인 클래스로 어플리케이션에서 다루고자 하는 테이블에 대응하여 구성할 수 있으며 테이블이 포함하는 컬럼에 대응한 속성들을 가지고 있음

```
@Entity
public class User  implements Serializable {
    private static final long serialVersionUID = -8077677670915867738L;

    public User(){
    }
    @Id
    private String userId;

    private String userName;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
}
```

Entity annotation 선언

Serializable 인터페이스 구현

argument 없는 생성자 선언

Primary Key 선언

❑ Entities (2/3)

- @Entity
  - 해당 클래스가 Entity 클래스임을 표시하는 것으로 클래스 선언문 위에 기재
  - 테이블명과 Entity명이 다를 때에는 name에 해당 테이블명을 기재

```
@Entity(name="USER_TB")
public class User {
}
```

- @Id
  - 해당 Attribute가 Key임을 표시하는 것으로 Attribute 위에 기재

```
@Id
private String userId;
```

- @Column
  - 해당 Attribute와 매핑되는 컬럼정보를 입력하기 위한 것으로 Attribue위에 기재
  - 컬럼명과 Attribute명이 일치할 경우는 기재하지 않아도 됨

```
@Column(name = "DEPT_NAME", length = 30)
private String deptName;
```

❑ Entities (3/3)

- @OneToOne, @OneToMany, @ManyToOne, @ManyToMany
  - 테이블간 관계를 구성하기 위한 것으로 정의되는 Attribute위에 기재
  - 각각은 1:1,1:N,N:1,N;N의 관계를 표현함.

```
@ManyToMany
private Set<Role> roles = new HashSet(0);
```

- @Transient
  - 테이블의 컬럼과 매핑되지 않고 쓰이는 Attribute를 정의하고자 할때 Attribute위에 기재

```
@Transient
private String roleName;
```

❑ Entity Status

- New(transient) : 단순히 Entity 객체가 초기화되어 있는 상태
- Managed(persistent) : Entity Manager에 의해 Entity가 관리되는 상태
- Detached : Entity 객체가 더 이상 Persistance Context와 연관이 없는 상태
- Removed : Managed 되어 있는 Entity 객체가 삭제된 상태

❑ Entity Operation (1/3)

- 특정 DB에 데이터를 입력,수정,조회,삭제,배치 입력 등의 작업을 수행하는 오퍼레이션
- 입력
  - EntityManager의 persist()메소드를 호출하여 DB에 단건의 데이터 추가

```
Department department = new Department();
String DepartmentId = "DEPT-0001";

entityManager.persist(department);
```

- 수정
  - EntityManager의 merge() 메소드 호출
  - 특정 객체가 Persistent 상태이고, 동일한 트랜잭션 내에서 해당 객체의 속성 값에 변경이 발생한 경우, merge() 메소드를 직접적으로 호출하지 않아도 트랜잭션 종료 시점에 변경 여부가 체크되어 변경 사항이 DB에 반영됨

```
// 2. update a Department information
department.setDeptName("Purchase Dept");

// 3. 명시적인 메소드 호출
entityManager.merge(department);
```

❑ Entity Operation (2/3)

- 조회
  - EntityManager의 find()메소드를 호출하여 DB에서 원하는 한건의 데이터를 조회할 수 있음
  - find() 메소드 호출시 대상이 되는 Entity의 Id를 입력 인자로 전달해야 함

```
Department result = (Department) entityManager.find(Department.class, departmentId);
```

- 삭제
  - EntityManager의 remove() 메소드 사용
  - 삭제 할 객체가 동일한 경우 remove() 메소드 호출시 대상이 되는 Entity를 입력 인자로 전달하여 삭제함

```
// 1. insert a new Department information
Department department = addDepartment();

// 2. delete a Department information
entityManager.remove(department);
```

  - 삭제 할 객체가 동일한 객체가 아닐 경우 getReference 메소드를 호출하여 Entity의 Id에 해당하는 객체 정보를 추출하여 그정보를 입력인자로 해서 remove를 호출하여 삭제함

```
Department department = new Department();
department.setDeptId = "DEPT_1";

// 2. delete a Department information
entityManager.remove(em.getReference(Department.class, department.getDeptId()));
```

❑ Entity Operation (3/3)

- 배치입력
  - EntityManager의 persist()메소드를 호출하여 DB에 입력하고 loop를 통해 반복적으로 수행
  - OutOfMemoryException 방지를 위해서 일정한 term을 두고 flush(),clear()을 호출하여 메모리에 있는 사항을 삭제

```
public void testMultiSave() throws Exception {

   for (int i = 0; i < 900 ; i++) {

      Department department = new Department();
      String DeptId = "DEPT-000" + i;
      department.setDeptId(DeptId);
      department.setDeptName("Sale" + i);
      department.setDesc("판매부" + i);

      em.persist(department);
      logger.debug("=== DEPT-000"+i+" ===");

      // OutOfMemoryException 피하기 위해서
      if (i != 0 && i % 9 == 0) {
         em.flush();
         em.clear();
      }
   }
}
```

❑ Callback Methods (1/2)

- 엔티티 Operation 직전 직후에 비지니스 로직 체크 등의 로직을 별도 분리하여 처리하도록 지원함
- Callback Methods 종류
  - PrePersist : Persist이전 시점에 수행
  - PostPersist : Persist이후 시점에 수행
  - PreRemove : Remove이전 시점에 수행
  - PostRemove : Remove이후 시점에 수행
  - PreUpdate : Merge이전 시점에 수행
  - PostUpdate : Merge이후 시점에 수행
  - PostLoad : Find 이후 시점에 수행

❑ Callback Methods (2/2)

- Callback Methods 정의 방식
  - 엔티티 클래스에 내부 정의
  - EntityListener를 지정하여 콜백 함수 정의

- 엔티티 클래스에 내부 정의 예제
  - salary가 2000000 이하로 설정되어 Update가 실행될 경우 Exception 이 발생함

```
@Entity
public class User {
  @PrePersist
  @PreUpdate
  protected void validateCreate() throws Exception {
    if (getSalary() < 2000000 )
      throw new Exception("Insufficient Salary !");
  }
}
```

❑ Association Mapping (1/5)

- 두 클래스 사이의 연관관계 유형에 따라 매핑 관계를 선언함
  - One To One Mapping
  - One To Many Mapping
  - Many To Many Mapping

- One To One Mapping 예제
  - Employee 와 TravelProfile가 각각 OneToOne이라는 Annotation을 기재하여 매핑 선언

```
@Entity
public class Employee {
  @OneToOne
  private TravelProfile profile;
}

@Entity
public class TravelProfile {
  @OneToOne
  private Employee employee;
}
```

❑ Association Mapping (2/5)

- One To Many Mapping
  - Department:User = 1:N 의 관계가 있으며 그 관계에 대해서 Department 클래스에서 OneToMany로 표시하고 User 클래스에서 ManyToOne으로 표시하여 관계를 나타냈다.

```
@Entity
public class Department{
  @OneToMany(targetEntity=User.class)
  private Set<User> users = new HashSet(0);
}

@Entity
public class User{
  @ManyToOne
  private Department department;
}
```

❑ Association Mapping (3/5)

- Collection Type
  - Many관계에서 Collection Type은 Set 이외에도 List, Map를 사용할 수 있음
  - Set 타입 : java.util.Set 타입으로 <set>을 이용하여 정의
  - List 타입 : java.util.List 타입으로 <list>를 이용하여 정의
  - Map 타입 : java.util.map 타입으로 <map>을 이용하여 (키,값)을 쌍으로 정의

```
// Set 예제
@OneToMany(targetEntity=User.class)
private Set<User> users = new HashSet(0);

// List 예제
@OneToMany(targetEntity=User.class )
private List<User> users = new ArrayList(0);

// Map 예제
@OneToMany(targetEntity=User.class)
@MapKey(name="userId")
private Map<String,User> users ;
```

❑ Association Mapping (4/5)

- 단방향/양방향 관계 속성
  - 1:N(부모:자식)관계 지정에 있어서 자식쪽에서 부모에 대한 참조 관계를 가지고 있느냐 없느냐에 따라서 참조관계가 있으면 양방향 관계, 없으면 단방향 관계로 정의
  - 단방향/ 양방향 예제

```
//단방향 예제
@Entity
public class Department {
  @OneToMany(targetEntity=User.class)
  private Set<User> users = new HashSet(0);
}

@Entity
public class User {
  @Column(name="DEPT_ID")
  private String deptId;
}
```

```
//양방향 예제
@Entity
public class Department {
  @OneToMany(targetEntity=User.class)
  private Set<User> users = new HashSet(0);
}

@Entity
public class User {
  @ManyToOne
  private Department department;
}
```

❑ Association Mapping (5/5)

- Many To Many Mapping
  - Role:User = M:N 의 관계가 있다면 그 관계에 대해서 Role클래스에서 ManyToMany로 표시하고 User 클래스에서 ManyToMany로 표시하여 관계를 나타내면서 User-Role 관계 테이블을 정의
  - ROLE과 USER를 연결하는 관계 테이블로 AUTHORITY가 사용되었음을 선언

```
@Entity
public class Role {
  @ManyToMany(targetEntity=User.class)
  private Set<User> users = new HashSet(0);
}

@Entity
public class User {
  @ManyToMany
  @JoinTable(name="AUTHORITY",
                joinColumns=@JoinColumn(name="USER_ID"),
                inverseJoinColumns=@JoinColumn(name="ROLE_ID"))
  private Set<Role> roles = new HashSet(0);
}
```

❑ Spring Integration (1/7)

- Spring에서는 JPA 기반에서 DAO 클래스를 쉽게 구현할 수 있도록 하기 위해 JdbcTemplate, HibernateTemplate 등 처럼 JpaTemplate클래스를 제공함
- JPA에서 정의한 Entity Manager의 Method를 직접 이용하는 방식도 제공함
- 기본설정
  - persistence.xml 설정 (persistHSQLMemDB.xml 파일)

```
<persistence-unit name="HSQLMUnit" transaction-type="RESOURCE_LOCAL">
// 구현체는 Hibernate
<provider>org.hibernate.ejb.HibernatePersistence</provider>

// Entity Class List
<class>egovframework.sample.model.bidirection.User</class>
<class>egovframework.sample.model.bidirection.Role</class>
<class>egovframework.sample.model.bidirection.Department</class>
<exclude-unlisted-classes/>

<properties>
  // DBMS별 다른 설정 여기는 HSQL 설정.
  <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
  <property name="hibernate.show_sql" value="true"/>
  <property name="hibernate.format_sql" value="true"/>
  <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
</properties>
</persistence-unit>
```

❑ Spring Integration (2/7)

- 기본설정

  • Application Context 설정 (1/2)

```
// 1.Transation Manager 설정
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
   <property name="entityManagerFactory" ref="entityManagerFactory " />
</bean>

// 2.Entity Manager Factory 설정
<bean id="entityManagerFactory "
   class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
   <property name="persistenceUnitName" value="HSQLMUnit" />
   <property name="persistenceXmlLocation" value="classpath:META-INF/persistHSQLMemDB.xml" />
   <property name="dataSource" ref="dataSource" />
</bean>

// 3.DataSource 설정
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource"
   destroy-method="close">
   <property name="driverClassName" value="net.sf.log4jdbc.DriverSpy" />
   <property name="url" value="jdbc:log4jdbc:hsqldb:mem:testdb" />
   <property name="username" value="sa" />
   <property name="password" value="" />
   <property name="defaultAutoCommit" value="false" />
</bean>
```

persistence.xml 파일의 persistence-unit name 속성값과 파일 위치를 지정

❑ Spring Integration (3/7)

- 기본설정

  • Application Context 설정 (2/2)

```
// 4. JPA Annotation 사용 설정
<context:annotation-config />

// 5. JPA 예외변환 후처리기 설정
<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />

// 6.Annotation 기반의 Transaction 활성화 설정
<tx:annotation-driven />
```

❑ Spring Integration (4/7)

- Plain JPA 이용 (1/2)
  - JPA에서 정의한 Entity Manager의 Entity Method를 호출 작업할 수 있음
  - Entity Manager를 통해 작업함으로써 Spring 환경하에서 Spring에 대한 의존성을 최소화 할 수 있음

```
public class RoleDAO {
  // Application Context 설정의 4 .JPA Annotation 사용 설정에 의해서 가능한 것으로,
JPA의 @PersistenceContext를 사용하면 EntityManager 객체를 바로 사용할 수 있다.
  @PersistenceContext
  private EntityManager em;

   // EntityManager를 통한 입력
  public void createRole(Role role) throws Exception {
    em.persist(role);
  }
   // EntityManager를 통한 조회
  public Role findRole(String roleId) throws Exception {
    return (Role) em.find(Role.class, roleId);
  }
  // EntityManager를 통한 삭제
  public void removeRole(Role role) throws Exception {
    em.remove(em.getReference(Role.class, role.getRoleId()));
  }
  // EntityManager를 통한 수정
  public void updateRole(Role role) throws Exception {
    em.merge(role);
  }
}
```

❑ Spring Integration (5/7)

- Plain JPA 이용 (2/2)
  - Entity 클래스

```
@Entity
public class Role implements Serializable {

  private static final long serialVersionUID = 1042037005623082102L;

  @Id
  @Column(name = "ROLE_ID", length=10)
  private String roleId;

  @Column(name = "ROLE_NAME", length=20)
  private String roleName;

  @Column(name = "DESC" , length=50)
  private String desc;
  ...
}
```

❑ Spring Integration (6/7)

- JpaTemplate 이용 (1/2)
  - Spring에서 정의한 JpaDaoSupport를 상속받아 getJpaTemplate()를 통해서 Entity Method 등을 호출 작업할 수 있음

```
public class UserDAO extends JpaDaoSupport {
  // Application Context 에서 설정한 Entity Manager Factory 명을 지정하여 부모의 EntityManagerFactory를 설정한다.
  @Resource(name="entityManagerFactory")
  public void setEMF(EntityManagerFactory entityManagerFactory) {
    super.setEntityManagerFactory(entityManagerFactory);
  }
   // getTemplate()에 의한 입력
  public void createUser(User user) throws Exception {
    this.getJpaTemplate().persist(user);
  }
   // getTemplate()에 의한 조회
  public User findUser(String userId) throws Exception {
    return (User) this.getJpaTemplate().find(User.class, userId);
  }
   // getTemplate()에 의한 삭제
  public void removeUser(User user) throws Exception {
    this.getJpaTemplate().remove(this.getJpaTemplate().getReference(User.class, user.getUserId()));
  }
}
```

❑ Spring Integration (7/7)

- JpaTemplate 이용 (2/2)
  - Entity 클래스

```
@Entity
public class User implements Serializable {

  private static final long serialVersionUID = -8077677670915867738L;

  @Id
  @Column(name = "USER_ID", length=10)
  private String userId;

  @Column(name = "USER_NAME", length=20)
  private String userName;

  @Column(length=20)
  private String password;

  ...
}
```

❑ Spring Data JPA

- https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

❑ Hibernate 공식 사이트

- www.hibernate.org