HOL : Spring MVC

Task1. Spring MVC Demo

1. Package Explorer > right-click > New > Other > Spring > Spring Legacy Project
2. Select Spring MVC Project
3. Project name : HelloWorldWeb > Next
4. Enter a topLevelPackage : com.example.biz > Finish

5. pom.xml 수정하기

```
<properties>
   <java-version>11</java-version>
   <org.springframework-version>5.3.12</org.springframework-version>
   <org.aspectj-version>1.9.7</org.aspectj-version>
   <org.slf4j-version>1.7.32</org.slf4j-version>
</properties>
   ...
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>javax.servlet-api</artifactId>
   <version>4.0.1</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>javax.servlet.jsp</groupId>
   <artifactId>javax.servlet.jsp-api</artifactId>
   <version>2.3.3</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>org.junit.jupiter</groupId>
   <artifactId>junit-jupiter-api</artifactId>
   <version>5.8.1</version>
   <scope>test</scope>
</dependency>
...
<build>
   <plugins>
      <plugin>
         <artifactId>maven-eclipse-plugin</artifactId>
         <version>2.10</version>
         ...
      </plugin>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.8.1</version>
         <configuration>
            <source>11</source>
            <target>11</target>
            ...
      </plugin>
      <plugin>
         <groupId>org.codehaus.mojo</groupId>
         <artifactId>exec-maven-plugin</artifactId>
         <version>3.0.0</version>
```

6. pom.xml > right-click > Run As > Maven install

   [INFO] BUILD SUCCESS

7. HelloWorldWeb project > right-click > Properties > Project Facets > Select Java > Change Version 11

   -Select Runtimes Tab > Check Apache Tomcat v9.0 > Click Apply and Close

8. HelloWorldWeb Project right-click > Run As > Run on Server > Finish

9. http://localhost:8080/biz/

   Hello world!

   The time on the server is 2019년 6월 11일 (화) 오후 11시 40분 58초.<--원래 한글 깨짐

10. 한글 깨짐을 수정하는 것은 src/main/webapp/WEB-INF/views/home.jsp에서
<%@ page session="false" pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"%>로 수정

11. Context name 변경하기
1)Package Explorer에서 Servers/Tomcat v9.0 Server at localhost-config/server.xml에서 다음과 같이 수정한다.
-path="/biz" --> path="/demo"
<Context docBase="HelloWorldWeb" path="/demo" reloadable="true"
source="org.eclipse.jst.jee.server:HelloWorldWeb"/>

2)수정 후 restart 하면 http://localhost:8080/biz --> http://localhost:8080/demo 로 변경됨

Task2. resources Folder 이용하기
1. Image 경로 알아내기
1)src/main/webapp/resources/에 images Folder를 STS Package Explorer에서 생성한다.
2)Download받은 image를 src/main/webapp/resources/images/에 넣는다.
3)home.jsp에 아래 code를 추가한다.

4)Image가 잘 나온다.

2. Image 경로 변경
1)apple.jpg image 경로를 src/main/webapp/images/로 이동.
2)하지만 이렇게 하면 image가 보이지 않는다.
3)왜냐하면, servlet-context.xml에서 resource의 경로는 <resources mapping="/resources/**" location="/resources/" />이기 때문.
4)즉, 기본적으로 resources folder 아래에서 resource를 찾는다.

3. <resources />추가
1)다시 resources Folder 하위로 images Folder를 복사
2)/src/main/webapp/하위에 images Folder를 생성하고 image를 넣고 home.jsp에 아래의 code를 추가한다.

3)하지만 아래의 image는 보이지 않는다.
4)왜냐하면 새로 추가한 images Folder는 servlet-context.xml에서 설정하지 않았기 때문.
5)Image를 보이게 하기 위해 servlet-context.xml에 아래의 Code를 추가한다.
<resources mapping="/resources/**" location="/resources/" />
<resources mapping="/images/**" location="/images/" />

6)src/main/webapp/images Folder 추가
7)Project right-click > Run As > Run on Server > Restart >
-Image가 제대로 2개가 나온다.

Task3. Controller Class 제작하기
1. 제작순서
1)@Controller를 이용한 class 생성
2)@RequestMapping을 이용한 요청 경로 지정
3)요청 처리 method 구현
4)View 이름 return

5)src/main/java/com.example.biz.UserController class 생성

@Controller
public class UserController {

2. 요청 처리 method 생성

```
package com.example.biz;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
```

```
@Controller
public class UserController {
   @RequestMapping("/view")
   public String view(Model model){
      /*
      model.addAttribute("username", "한지민");
      model.addAttribute("userage", 24);
      model.addAttribute("job", "Developer");
      return "view";
      */
      model.addAttribute("currentDate", new java.util.Date());
      return "view";    // /WEB-INF/views/view  + .jsp
   }

   @RequestMapping("/fruits")
   public String fruits(Model model){
      String [] array = {"Apple", "Mango", "Lemon", "Grape"};

      model.addAttribute("fruits", array);

      return "fruits";    // /WEB-INF/views/fruits  + .jsp
   }
}
```

3. View에 Data 전달

1)src/main/webapp/WEB-INF/views/view.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Insert title here</title>
   </head>
   <body>
      <h1>view.jsp 입니다.</h1>
      현재 날짜와 시간은 ${currentDate} 입니다.
   </body>
</html>
```

2)src/main/webapp/WEB-INF/views/fruits.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h2>fruits.jsp</h2>
   <ul>과일 종류
   <c:forEach items="${fruits}" var="fruit">
      <li>${fruit}</li>
   </c:forEach>
   </ul>
</body>
</html>
```

3)http://localhost:8080/demo/view --> /view.jsp

4)http://localhost:8080/demo/fruits --> /fruits.jsp

4. View에 ModelAndView 객제로 data 전달

1)UserController.java에 아래의 코드 추가

```
@RequestMapping(value = "/demo", method = RequestMethod.GET)
public ModelAndView demo() {
   /*
   ModelAndView mav = new ModelAndView("view2");
   mav.addObject("username", "한지민");
   mav.addObject("currentDate", new java.util.Date());
```

```
        return mav;
        */
        ModelAndView mav = new ModelAndView();
        mav.addObject("userid", "example");
        mav.addObject("passwd", "12345678");
        mav.setViewName("/demo");
        return mav;
    }
```

2)src/main/webapp/WEB-INF/views/demo.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html">
<html>
    <head>
        <meta charset="UTF-8">
        <title>Insert title here</title>
    </head>
    <body>
        아이디 : ${userid} <br />
        패스워드 : ${passwd}
    </body>
</html>
```

3)http://localhost:8080/demo/demo --> /demo.jsp

아이디 : example

패스워드 : 12345678

5. Controller class에 @RequestMapping 적용

1)src/main/java/com.example.biz.StudentController.java 생성

```
package com.example.biz;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/bbs")
public class StudentController {

    @RequestMapping(value="/get", method = RequestMethod.GET)
    public ModelAndView getStudent() {

        ModelAndView mav = new ModelAndView();
        mav.setViewName("/bbs/get");
        mav.addObject("name", "한지민");
        mav.addObject("age", 25);
        return mav;
    }
}
```

2)src/main/webapp/WEB-INF/views/bbs/get.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    학생 이름 : ${name} <br />
    학생 나이 : ${age}
</body>
</html>
```

3)http://localhost:8080/demo/bbs/get

학생 이름 : 한지민

학생 나이 : 25

Task4. 다양한 GET Request 처리하기

1. Package Explorer > right-click > New > Spring Legacy Project
2. Select Spring MVC Project
3. Project name : MVCDemo > Next
4. Enter a topLevelPackage : com.example.biz > Finish

5. pom.xml 수정하기

```
<properties>
   <java-version>11</java-version>
   <org.springframework-version>5.3.12</org.springframework-version>
   <org.aspectj-version>1.9.7</org.aspectj-version>
   <org.slf4j-version>1.7.32</org.slf4j-version>
</properties>
...
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>javax.servlet-api</artifactId>
   <version>4.0.1</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>javax.servlet.jsp</groupId>
   <artifactId>javax.servlet.jsp-api</artifactId>
   <version>2.3.3</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>org.junit.jupiter</groupId>
   <artifactId>junit-jupiter-api</artifactId>
   <version>5.8.1</version>
   <scope>test</scope>
</dependency>
...
   <build>
      <plugins>
         <plugin>
            <artifactId>maven-eclipse-plugin</artifactId>
            <version>2.10</version>
            ...
         </plugin>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
               <source>11</source>
               <target>11</target>
               ...
         </plugin>
         <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>3.0.0</version>
```

6. pom.xml > right-click > Run As > Maven install
   [INFO] BUILD SUCCESS

7. Project > right-click > Properties > Project Facets > Select Java > Change Version 11
   -Select Runtimes Tab > Check Apache Tomcat v9.0 > Click Apply and Close

8. src/main/java/com.example.biz/RequestController.java 생성

```
package com.example.biz;

import org.springframework.stereotype.Controller;

@Controller
public class RequestController {
```

9. HttpServletRequest class 이용하기

1)RequestController.java

```
@RequestMapping(value="/confirm", method=RequestMethod.GET)
public String confirm(HttpServletRequest request, Model model) {
   String userid = request.getParameter("userid");
   String passwd = request.getParameter("passwd");
   String name = request.getParameter("name");
   int age = Integer.parseInt(request.getParameter("age"));
   String gender = request.getParameter("gender");

   model.addAttribute("userid", userid);
   model.addAttribute("passwd", passwd);
   model.addAttribute("name", name);
   model.addAttribute("age", age);
   model.addAttribute("gender", gender);
   return "confirm";
}
```

2)src/main/webapp/WEB-INF/views/confirm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Insert title here</title>
   </head>
   <body>
      아이디 : ${userid} <br />
      패스워드 : ${passwd} <br />
      사용자 이름 : ${name} <br />
      나이 : ${age} <br />
      성별 : ${gender} <br />
   </body>
</html>
```

3)Project right-click > Run As > Run on Server > restart

4)localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234

아이디 : jimin
패스워드 : 1234
사용자 이름 : 한지민
나이 : 25
성별 : 여성

10. @RequestParam Annotation 이용하기

1)RequestController.java

```
@RequestMapping(value="/confirm", method=RequestMethod.GET)
public String confirm(@RequestParam("userid") String userid,
                      @RequestParam("passwd") String passwd,
                      @RequestParam("name") String name,
                      @RequestParam("age") int age,
                      @RequestParam("gender") String gender ,Model model) {

   model.addAttribute("userid", userid);
   model.addAttribute("passwd", passwd);
   model.addAttribute("name", name);
   model.addAttribute("age", age);
   model.addAttribute("gender", gender);
   return "confirm";
}
```

2)src/main/webapp/WEB-INF/views/confirm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Insert title here</title>
   </head>
   <body>
      아이디 : ${userid} <br />
```

```
            패스워드 : ${passwd} <br />
            사용자 이름 : ${name} <br />
            나이 : ${age} <br />
            성별 : ${gender} <br />
        </body>
    </html>
```

3)localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234
아이디 : jimin
패스워드 : 1234
사용자 이름 : 한지민
나이 : 25
성별 : 여성

11. Data Commander 객체 이용하기1
1)src/main/java/com.example.vo.UserVO.java 생성

```
package com.example.vo;

public class UserVO {
    private String userid;
    private String passwd;
    private String name;
    private int age;
    private String gender;

    public UserVO(){}
    public UserVO(String userid, String passwd, String name, int age, String gender){
        this.userid = userid;
        this.passwd = passwd;
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
    public String getUserid() {
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    public String getPasswd() {
        return passwd;
    }
    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    @Override
    public String toString() {
        return "UserVO [userid=" + userid + ", passwd=" + passwd + ", name=" + name + ", age=" + age +
        ", gender="
                + gender + "]";
    }
}
```

2)RequestController.java

```
@RequestMapping(value="/confirm", method=RequestMethod.GET)
public String confirm(@RequestParam("userid") String userid,
      @RequestParam("passwd") String passwd,
      @RequestParam("name") String name,
      @RequestParam("age") int age,
      @RequestParam("gender") String gender, Model model) {

   UserVO userVO = new UserVO();
   userVO.setUserid(userid);
   userVO.setPasswd(passwd);
   userVO.setName(name);
   userVO.setAge(age);
   userVO.setGender(gender);

   model.addAttribute("userVO", userVO);

   return "confirm1";
}
```

3)src/main/webapp/WEB-INF/views/confirm1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="user" value="${userVO}"/>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h1>confirm1.jsp</h1>
   <h2>사용자 정보</h2>
   아이디 : ${user.userid} <br />
   패스워드 : ${user.passwd} <br />
   이름 : ${user.name} <br />
   나이 : ${user.age} <br />
   성별 : ${user.gender}
</body>
</html>
```

4)localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234
confirm1.jsp

사용자 정보

아이디 : jimin
패스워드 : 1234
사용자 이름 : 한지민
나이 : 25
성별 : 여성

12. Data Commander 객체 이용하기2

1)RequestController.java

```
@RequestMapping(value="/confirm", method=RequestMethod.GET)
public String confirm(UserVO userVO) {

   return "confirm2";
}
```

2)src/main/webapp/WEB-INF/views/confirm2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="user" value="${userVO}"/>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
```

```
<body>
   <h1>confirm2.jsp</h1>
   <h2>사용자 정보</h2>
   아이디 : ${user.userid} <br />
   패스워드 : ${user.passwd} <br />
   이름 : ${user.name} <br />
   나이 : ${user.age} <br />
   성별 : ${user.gender}
</body>
</html>
```

3)localhost:8080/biz/confirm?name=한지민&gender=여성&age=25&userid=jimin&passwd=1234
confirm2.jsp

사용자 정보

아이디 : jimin
패스워드 : 1234
사용자 이름 : 한지민
나이 : 25
성별 : 여성

13. @PathVariable 이용하기
1)RequestController.java

```
@RequestMapping(value="/confirm/{userid}/{passwd}/{name}/{age}/{gender}",
method=RequestMethod.GET)
public String confirm(@PathVariable String userid, @PathVariable String passwd,
                      @PathVariable String name, @PathVariable int age,
                      @PathVariable String gender, Model model) {
   model.addAttribute("userInfo", new UserVO(userid, passwd, name, age, gender));
   return "confirm3";
}
```

2)src/main/webapp/WEB-INF/views/confirm3.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="user" value="${userInfo}"/>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h1>confirm3.jsp</h1>
   <h2>사용자 정보</h2>
   아이디 : ${user.userid} <br />
   패스워드 : ${user.passwd} <br />
   이름 : ${user.name} <br />
   나이 : ${user.age} <br />
   성별 : ${user.gender}
</body>
</html>
```

3)localhost:8080/biz/confirm/jimin/1234/한지민/25/여성
confirm3.jsp

사용자 정보

아이디 : jimin
패스워드 : 1234
사용자 이름 : 한지민
나이 : 25
성별 : 여성

Task5. @RequestMapping Parameter 다루기
1. GET 방식과 POST 방식

1)src/main/java/com.example.biz/HomeController.java

```
@RequestMapping(value="/login", method=RequestMethod.POST)
public String login(@RequestParam("userid") String userid,
                    @RequestParam("passwd") String passwd,
                    Model model) {

   model.addAttribute("userid", userid);
   model.addAttribute("passwd", passwd);
   return "login";
}
```

2)src/main/webapp/resources/login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인 폼</title>
</head>
<body>
   <form method="GET" action="/biz/login">
      아이디 : <input type="text" name="userid" /><br />
      패스워드 : <input type="password" name="passwd" /><br />
      <input type="submit" value="로그인하기" />
   </form>
</body>
</html>
```

3)http://localhost:8080/biz/resources/login.html에서 submit 하면 405 error 발생
4)왜냐하면 서로의 method가 불일치하기 때문
5)해결방법
-src/main/java/com.example.biz/HomeController.java 수정
-즉 login method(요청 처리 method)의 이름은 같지만 parameter의 type과 return type이 틀리기 때문에 Method Overloading 됨.

```
@RequestMapping(value="/login", method=RequestMethod.POST)
public String login(@RequestParam("userid") String userid,
                    @RequestParam("passwd") String passwd,
                    Model model) {

   model.addAttribute("userid", userid);
   model.addAttribute("passwd", passwd);
   return "login";
}
@RequestMapping(value="/login", method=RequestMethod.GET)
public ModelAndView login(@RequestParam("userid") String userid,
                    @RequestParam("passwd") String passwd) {

   ModelAndView mav = new ModelAndView();
   mav.addObject("userid", userid);
   mav.addObject("passwd", passwd);
   mav.setViewName("login");
   return mav;
}
```

6)src/main/webapp/WEB-INF/views/login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
   아이디 : ${userid} <br />
   패스워드 : ${passwd}
</body>
</html>
```

7)http://localhost:8080/biz/resources/login.html
아이디 : jimin
패스워드 : 1234

2. @ModelAttribute Annotation 이용하기

1)@ModelAttribute Annotation을 이용하면 Data Commander 객체의 이름을 변경할 수 있다.

2)src/main/webapp/resources/register.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원가입 폼</title>
</head>
<body>
   <form method="POST" action="/biz/register">
      아이디 : <input type="text" name="userid" /><br />
      패스워드 : <input type="password" name="passwd" /><br />
      이름 : <input type="text" name="name" /><br />
      나이 : <input type="number" name="age" /><br />
      성별 : <input type="radio" name="gender" value="남성" />남성   
            <input type="radio" name="gender" value="여성" />여성<br />
      <input type="submit" value="가입하기" />
   </form>
</body>
</html>
```

3)src/main/java/com.example.biz/HomeController.java

```
@RequestMapping(value="/register", method=RequestMethod.POST)
public String register(@ModelAttribute("u") UserVO userVO) {    //userVO가 아니라 u로 변경

   return "register";
}
```

4)src/main/webapp/WEB-INF/views/register.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="user" value="${u}"/>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h1>사용자 정보</h1>
   <ul>
      <li>아이디 : ${user.userid}</li>
      <li>패스워드 : ${user.passwd}</li>
      <li>이름 : ${user.name}</li>
      <li>나이 : ${user.age}</li>
      <li>성별 : ${user.gender}</li>
   </ul>
</body>
</html>
```

5)Spring에서 POST 방식으로 Data를 보낼 때 한글깨짐 현상 발생

6)해결방법

7)web.xml

```
<filter>
   <filter-name>encodingFilter</filter-name>
   <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
   <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>encodingFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

8)http://localhost:8080/biz/resources/register.html -->

9)http://localhost:8080/biz/register

사용자 정보

아이디 : jimin
패스워드 : 1234
사용자 이름 : 한지민
나이 : 25
성별 : 여성

3. redirect: 키워드 이용하기
1)src/main/java/com.example.biz/HomeController.java

```
@RequestMapping("/verify")
public String verify(HttpServletRequest request, Model model) {
   String userid = request.getParameter("userid");
   if(userid.equals("admin")) {     //만일 userid가 admin 이면 /admin으로 리다이렉트
      return "redirect:admin";
   }
   return "redirect:user";          //만일 userid가 admin 이 아니면 /user로 리다이렉트
   //return "redirect:http://www.naver.com";    //절대 경로도 가능
}

@RequestMapping("/admin")
public String verify1(Model model) {
   model.addAttribute("authority", "관리자권한");
   return "admin";
}

@RequestMapping("/user")
public String verify2(Model model) {
   model.addAttribute("authority", "일반사용자");
   return "user";
}
```

2)/src/main/webapp/WEB-INF/views/admin.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h1>관리자 페이지</h1>
   권한 : ${authority}
</body>
</html>
```

3)/src/main/webapp/WEB-INF/views/user.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h1>일반 사용자 페이지</h1>
   권한 : ${authority}
</body>
</html>
```

4)http://localhost:8080/biz/verify?userid=admin --> http://localhost:8080/biz/admin
5)http://localhost:8080/biz/verify?userid=user --> https://www.naver.com

---

Task6. Database와 연동하기
1. Package Explorer > right-click > New > Spring Legacy Project
2. Select Spring MVC Project
3. Project name : MVCDemo1 > Next
4. Enter a topLevelPackage : com.example.biz > Finish
5. pom.xml 수정하기

```
    <properties>
      <java-version>11</java-version>
      <org.springframework-version>5.3.12</org.springframework-version>
      <org.aspectj-version>1.9.7</org.aspectj-version>
      <org.slf4j-version>1.7.32</org.slf4j-version>
    </properties>
      ...
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.3</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.8.1</version>
      <scope>test</scope>
    </dependency>
    ...
    <build>
      <plugins>
        <plugin>
          <artifactId>maven-eclipse-plugin</artifactId>
          <version>2.10</version>
          ...
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.1</version>
          <configuration>
            <source>11</source>
            <target>11</target>
            ...
        </plugin>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>3.0.0</version>
```

6. pom.xml > right-click > Run As > Maven install
   [INFO] BUILD SUCCESS

7. MVCDemo1 Project > right-click > Properties > Project Facets > Select Java > Change Version 11
   Select Runtimes Tab > Check Apache Tomcat v9.0 > Click Apply and Close

8. Create Table in MariaDB

```
CREATE TABLE Member
(
   userid        VARCHAR(20),
   username        VARCHAR(20)   NOT NULL,
   userage        TINYINT   NOT NULL,
   gender      VARCHAR(10)   NOT NULL,
   city          VARCHAR(50),
   CONSTRAINT member_userid_pk  PRIMARY KEY(userid)
);
```

-반드시 [test] Database의 조합을 utf8_general_ci로 맞출 것
-반드시 Member Table의 기본조합이 utf8_general_ci 임을 확인할 것

9. src/main/webapp/static folder 생성
   1)src/main/webapp/static/css folder

   2)src/main/webapp/static/images folder

3)src/main/webapp/static/js folder
-jquery-1.12.4.js

4)src/main/webapp/static/register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>회원 가입</title>
</head>
<body>
   <h1>회원 가입 창</h1>
   <form action="/biz/create" method="post">
      <ul>
         <li>ID : <input type="text" name="userid" /></li>
         <li>이름 : <input type="text" name="username" /></li>
         <li>나이 : <input type="number" name="age" /></li>
         <li>성별 : <input type="radio" name="gender" value="남성"/>남성
                        <input type="radio" name="gender" value="여성"/>여성</li>
         <li>거주지 : <input type="text" name="city" /></li>
         <li><input type="submit" value="가입하기" /></li>
      </ul>
   </form>
</body>
</html>
```

10. src/main/webapp/WEB-INF/spring/appServlet/sevlet-context.xml 수정

```
<resources mapping="/static/**" location="/static/" /> 추가

<context:component-scan base-package="com.example" /> 수정
```

11. src/main/resources/mariadb.properties

```
db.driverClass=org.mariadb.jdbc.Driver
db.url=jdbc:mariadb://localhost:3306/test
db.username=root
db.password=javamariadb
```

12. Spring JDBC 설치

1)JdbcTemplate를 사용하기 위해 pom.xml에 다음 dependency를 추가해야 함.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.3.12</version>
</dependency>
```

2)pom.xml에 붙여 넣고 Maven Install 하기

[INFO] BUILD SUCCESS

13. MariaDB Jdbc Driver library 검색 및 설치

1)Maven Repository 에서 'mariadb'로 검색하여 MariaDB Java Client를 설치한다.

```
<dependency>
      <groupId>org.mariadb.jdbc</groupId>
      <artifactId>mariadb-java-client</artifactId>
      <version>2.7.4</version>
</dependency>
```

2)pom.xml에 붙여 넣고 Maven Install 하기

[INFO] BUILD SUCCESS

14. src/main/webapp/WEB-INF/spring/root-context.xml

```
<?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:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="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">

   <!-- Root Context: defines shared resources visible to all other web components -->
   <!-- <context:property-placeholder location="classpath:mariadb.properties"/> -->

   <!-- or -->
   <bean id="my.propertyConfigurer"
   class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
      <property name="locations">
         <list>
            <value>classpath:dbinfo.properties</value>
         </list>
      </property>
   </bean>

   <bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
      <property name="driverClassName" value="${db.driverClass}"/>
      <property name="url" value="${db.url}"/>
      <property name="username" value="${db.username}" />
      <property name="password" value="${db.password}" />
   </bean>

   <!-- <bean id="dataSource" class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
      <property name="driverClass" value="${db.driverClass}" />
      <property name="url" value="${db.url}" />
      <property name="username" value="${db.username}" />
      <property name="password" value="${db.password}" />
   </bean> -->

   <!-- or -->
   <bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">
      <constructor-arg>
         <bean class="com.zaxxer.hikari.HikariConfig">
            <constructor-arg>
               <props>
                  <prop key="jdbcUrl">${db.url}</prop>
                  <prop key="username">${db.username}</prop>
                  <prop key="password">${db.password}</prop>
               </props>
            </constructor-arg>
            <property name="driverClassName" value="${db.driverClass}"/>
            <property name="minimumIdle" value="5" />
            <property name="maximumPoolSize" value="10" />
            <property name="connectionTestQuery" value="select 1 from sys.dual" />
            <property name="connectionTimeout"  value="300000" />
         </bean>
      </constructor-arg>
   </bean>

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


15. src/test/java/com.example.biz/TestApp class 생성
  1)com.example.biz > right-click > New > JUnit Test Case
  2)Select [New JUnit 4 test]
  3)Name : TestApp
  4)Finish
    package com.example.biz;

    import org.junit.Before;
    import org.junit.Test;
    import org.springframework.context.ApplicationContext;
    import org.springframework.context.support.GenericXmlApplicationContext;
    import org.springframework.jdbc.core.JdbcTemplate;

    public class TestApp {
       private ApplicationContext ctx;

       @Before
       public void init() {
```

```
            this.ctx = new
            GenericXmlApplicationContext("file:src/main/webapp/WEB-INF/spring**/root-context.xml");
        }
        @Test
        public void test() {
            JdbcTemplate jdbcTemplate = this.ctx.getBean("jdbcTemplate", JdbcTemplate.class);
            System.out.println(jdbcTemplate);
        }
    }
```

5)Run as > JUnit Test > Green bar

16. package 생성
    1)src/main/java/com.example.vo
    2)src/main/java/com.example.dao
    3)src/main/java/com.example.service

17. src/com.example.vo.MemberVO class 생성

```
package com.example.vo;

public class MemberVO {
    private String userid;
    private String username;
    private int age;
    private String gender;
    private String city;

    public MemberVO() {}

    public MemberVO(String userid, String username, int age, String gender, String city) {
        this.userid = userid;
        this.username = username;
        this.age = age;
        this.gender = gender;
        this.city = city;
    }

    public String getUserid() {
        return userid;
    }

    public void setUserid(String userid) {
        this.userid = userid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    public String getCity() {
        return city;
```

```
    }

    public void setCity(String city) {
        this.city = city;
    }

    @Override
    public String toString() {
        return "MemberVO [userid=" + userid + ", username=" + username + ", age=" + age + ", gender=" +
        gender
                + ", city=" + city + "]";
    }
}
```

18. com/example.dao

1)MemberDao interface

```
package com.example.dao;

import java.util.List;

import com.example.vo.MemberVO;

public interface MemberDao {
    int create(MemberVO memberVo);
    MemberVO read(String userid);
    List<MemberVO> readAll();
    int update(MemberVO memberVo);
    int delete(String userid);
}
```

2)MemberDaoImpl.java

```
package com.example.dao;

import java.util.List;

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

import com.example.vo.MemberVO;

@Repository("memberDao")
public class MemberDaoImpl implements MemberDao {
    @Autowired
    JdbcTemplate jdbcTemplate;

    @Override
    public int create(MemberVO memberVo) {
        return 0;
    }

    @Override
    public MemberVO read(String userid) {
        return null;
    }

    @Override
    public List<MemberVO> readAll() {
        return null;
    }

    @Override
    public int update(MemberVO memberVo) {
        return 0;
    }

    @Override
    public int delete(String userid) {
        return 0;
    }
}
```

19. com.example.service

1)MemberService interface

```
package com.example.service;

import java.util.List;

import com.example.vo.MemberVO;

public interface MemberService {
   int create(MemberVO memberVo);
   MemberVO read(String userid);
   List<MemberVO> readAll();
   int update(MemberVO memberVo);
   int delete(String userid);
}
```

2)MemberServiceImpl.java

```
package com.example.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.dao.MemberDao;
import com.example.vo.MemberVO;

@Service("memberService")
public class MemberServiceImpl implements MemberService {
   @Autowired
   MemberDao memberDao;

   @Override
   public int create(MemberVO memberVo) {
      return 0;
   }

   @Override
   public MemberVO read(String userid) {
      return null;
   }

   @Override
   public List<MemberVO> readAll() {
      return null;
   }

   @Override
   public int update(MemberVO memberVo) {
      return 0;
   }

   @Override
   public int delete(String userid) {
      return 0;
   }
}
```

20. com.example.biz

1)HomeController.java

```
package com.example.biz;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import com.example.service.MemberService;

/**
 * Handles requests for the application home page.
 */
@Controller
public class HomeController {
```

```
    @Autowired
    MemberService memberService;
}
```

21. Data Insert

1)/src/main/webapp/static/register.html

2)com.example.biz/HomeController.java

```
@Controller
public class HomeController {
    @Autowired
    MemberService memberService;

    @RequestMapping(value = "/create", method = RequestMethod.POST)
    public String home(MemberVO memberVo, Model model) {
        int row = this.memberService.create(memberVo);
        if(row == 1) model.addAttribute("status", "Insert Success");
        else model.addAttribute("status", "Insert Failure");
        return "create";
    }
}
```

3)com.example.service/MemberServiceImpl.java

```
@Service("memberService")
public class MemberServiceImpl implements MemberService {
    @Autowired
    MemberDao memberDao;

    @Override
    public int create(MemberVO memberVo) {
        return this.memberDao.create(memberVo);
    }
}
```

4)com.example.dao.MemberDaoImpl.java

```
@Repository("memberDao")
public class MemberDaoImpl implements MemberDao {
    @Autowired
    JdbcTemplate jdbcTemplate;

    @Override
    public int create(MemberVO memberVo) {
        String sql = "INSERT INTO Member VALUES(?,?,?,?,?)";
        return this.jdbcTemplate.update(sql, memberVo.getUserid(),
                memberVo.getUsername(), memberVo.getAge(),
                memberVo.getGender(), memberVo.getCity());
    }
}
```

5)views/create.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>${status}</h1>
</body>
</html>
```

22. POST 발송시 한글 깨짐 처리하기

1)web.xml

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
```

```
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

23. Test
    1)http://localhost:8080/biz/static/register.html
    2)http://localhost:8080/biz/create
    Insert Success

24. Data Select
    1)HomeController.java

```
    @RequestMapping(value = "/view/{userid}", method = RequestMethod.GET)
    public String view(@PathVariable String userid, Model model) {
        MemberVO memberVo = this.memberService.read(userid);
        model.addAttribute("member", memberVo);
        return "view";
    }
```

    2)MemberServiceImpl.java

```
    @Override
    public MemberVO read(String userid) {
        return this.memberDao.read(userid);
    }
```

    3)MemberDaoImpl.java

```
    @Override
    public MemberVO read(String userid) {
        String sql = "SELECT * FROM Member WHERE userid = ?";
        return this.jdbcTemplate.queryForObject(sql, new Object[] {userid},
                            new MyRowMapper());
    }

    class MyRowMapper implements RowMapper<MemberVO>{
        @Override
        public MemberVO mapRow(ResultSet rs, int rowNum) throws SQLException {
            MemberVO memberVo = new MemberVO(rs.getString("userid"),
                    rs.getString("username"), rs.getInt("userage"),
                    rs.getString("gender"), rs.getString("city"));
            return memberVo;
        }
    }
```

    4)views/view.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
    <c:set var="user" value="${member}" />
    <!DOCTYPE html>
    <html>
    <head>
        <meta charset="UTF-8">
        <title>Insert title here</title>
        <script src="/biz/static/js/jquery-1.12.4.js"></script>
        <script>
            $(function(){
                $("#btnList").bind("click", function(){
                    location.href = "/biz/list";
                });
                $("#btnDelete").bind("click", function(){
                    location.href = "/biz/delete/${user.userid}";
                });
            });
        </script>
    </head>
```

```
<body>
  <h1>${user.username}의정보</h1>
  <form action="/biz/update" method="post">
    <input type="hidden" name="userid" value = "${user.userid}" />
    <ul>
      <li>아이디 : ${user.userid }</li>
      <li>나이 : <input type='number' name="age" value='${user.age}' /></li>
      <li>성별 : <c:if test='${user.gender eq "남성"}'>
            <input type="radio" name="gender" value="남성" checked />남성  
            <input type="radio" name="gender" value="여성" />여성
            </c:if>
            <c:if test='${user.gender eq "여성"}'>
            <input type="radio" name="gender" value="남성" />남성  
          <input type="radio" name="gender" value="여성" checked />여성
      </c:if>
      </li>
      <li>거주지 : <input type="text" name="city" value="${user.city }" /></li>
      <li><input type='submit' value='수정하기' /></li>
      <li><input type='button' value='삭제하기'  id="btnDelete"/></li>
      <li><input type='button' value='목록으로'  id="btnList"/></li>
    </ul>
  </form>
</body>
</html>
```

5)Test

http://localhost:8080/biz/view/jimin

25. Data List

1)HomeController.java

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
public String list(Model model) {
  List<MemberVO> list = this.memberService.readAll();
  model.addAttribute("userlist", list);
  return "list";
}
```

2)MemberServiceImpl.java

```
@Override
public List<MemberVO> readAll() {
  return this.memberDao.readAll();
}
```

3)MemberDaoImpl.java

```
@Override
public List<MemberVO> readAll() {
  String sql = "SELECT * FROM Member ORDER BY userid DESC";
  return this.jdbcTemplate.query(sql, new MyRowMapper());
}
```

4)iews/list.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <h1>Member List</h1>
  <table border='1'>
    <thead>
      <tr>
        <th>아이디</th><th>이름</th><th>나이</th><th>성별</th><th>거주지</th>
      </tr>
    </thead>
    <tbody>
      <c:forEach items="${userlist}" var="user">
```

```
                <tr>
                   <td><a
                   href="/biz/view/${user.userid}">${user.userid}</a></td><td>${user.username}</td>
                   <td>${user.age}</td><td>${user.gender }</td>
                   <td>${user.city }</td>
                </tr>
             </c:forEach>
          </tbody>
       </table>
    </body>
    </html>
```

5)Test
http://localhost:8080/biz/list

26. Data Delete

1)HomeController.java

```
@RequestMapping(value = "/delete/{userid}", method = RequestMethod.GET)
public String delete(@PathVariable String userid) {
   this.memberService.delete(userid);
   return "redirect:/list";
}
```

2)MemberServiceImpl.java

```
@Override
public int delete(String userid) {
   return this.memberDao.delete(userid);
}
```

3)MemberDaoImpl.java

```
@Override
public int delete(String userid) {
   String sql = "DELETE FROM Member WHERE userid = ?";
   return this.jdbcTemplate.update(sql, userid);
}
```

4)Test
http://localhost:8080/biz/delete/chulsu

27. Data Update

1)HomeController.java

```
@RequestMapping(value = "/update", method = RequestMethod.POST)
public String update(@RequestParam("userid") String userid,
      @RequestParam("age") int age,
      @RequestParam("gender") String gender,
      @RequestParam("city") String city) {
   this.memberService.update(
         new MemberVO(userid, "", age, gender, city));
   return "redirect:/list";
}
```

2)MemberServiceImpl.java

```
@Override
public int update(MemberVO memberVo) {
   return this.memberDao.update(memberVo);
}
```

3)MemberDaoImpl.java

```
@Override
public int update(MemberVO memberVo) {
   String sql = "UPDATE Member SET userage = ?, gender = ?, city = ? " +
                "WHERE userid = ?";
   return this.jdbcTemplate.update(sql, memberVo.getAge(),
         memberVo.getGender(), memberVo.getCity(), memberVo.getUserid());
}
```

4)Test

http://localhost:8080/biz/list에서

해당 ID Click

데이터 수정

[수정하기] button click

28. All Codes

1)HomeController.java

```
package com.example.biz;

import java.util.List;

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

import com.example.service.MemberService;
import com.example.vo.MemberVO;

/**
 * Handles requests for the application home page.
 */
@Controller
public class HomeController {
   @Autowired
   MemberService memberService;

   @RequestMapping(value = "/create", method = RequestMethod.POST)
   public String home(MemberVO memberVo, Model model) {
      int row = this.memberService.create(memberVo);
      if(row == 1) model.addAttribute("status", "Insert Success");
      else model.addAttribute("status", "Insert Failure");
      return "create";    //   /WEB-INF/views/create.jsp
   }

   @RequestMapping(value = "/view/{userid}", method = RequestMethod.GET)
   public String view(@PathVariable String userid, Model model) {
      MemberVO memberVo = this.memberService.read(userid);
      model.addAttribute("member", memberVo);
      return "view";
   }

   @RequestMapping(value = "/list", method = RequestMethod.GET)
   public String list(Model model) {
      List<MemberVO> list = this.memberService.readAll();
      model.addAttribute("userlist", list);
      return "list";    // /WEB-INF/views/list.jsp
   }

   @RequestMapping(value = "/delete/{userid}", method = RequestMethod.GET)
   public String delete(@PathVariable String userid) {
      this.memberService.delete(userid);
      return "redirect:/list";
   }

   @RequestMapping(value = "/update", method = RequestMethod.POST)
   public String update(@RequestParam("userid") String userid,
         @RequestParam("age") int age,
         @RequestParam("gender") String gender,
         @RequestParam("city") String city) {
      this.memberService.update(
            new MemberVO(userid, "", age, gender, city));
      return "redirect:/list";
   }
}
```

2)MemberServiceImpl.java

```
package com.example.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.dao.MemberDao;
import com.example.vo.MemberVO;

@Service("memberService")
public class MemberServiceImpl implements MemberService {
   @Autowired
   MemberDao memberDao;

   @Override
   public int create(MemberVO memberVo) {
      return this.memberDao.create(memberVo);
   }

   @Override
   public MemberVO read(String userid) {
      return this.memberDao.read(userid);
   }

   @Override
   public List<MemberVO> readAll() {
      return this.memberDao.readAll();
   }

   @Override
   public int update(MemberVO memberVo) {
      return this.memberDao.update(memberVo);
   }

   @Override
   public int delete(String userid) {
      return this.memberDao.delete(userid);
   }
}
```

3)MemberDaoImpl.java

```
package com.example.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

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

import com.example.vo.MemberVO;

@Repository("memberDao")
public class MemberDaoImpl implements MemberDao {
   @Autowired
   JdbcTemplate jdbcTemplate;

   @Override
   public int create(MemberVO memberVo) {
      String sql = "INSERT INTO Member VALUES(?,?,?,?,?)";
      return this.jdbcTemplate.update(sql, memberVo.getUserid(), memberVo.getUsername(),
      memberVo.getAge(),
            memberVo.getGender(), memberVo.getCity());
   }

   class MyRowMapper implements RowMapper<MemberVO> {
      @Override
      public MemberVO mapRow(ResultSet rs, int rowNum) throws SQLException {
         MemberVO memberVo = new MemberVO(rs.getString("userid"), rs.getString("username"),
         rs.getInt("userage"),
```

```
                    rs.getString("gender"), rs.getString("city"));
            return memberVo;
         }
      }

      @Override
      public MemberVO read(String userid) {
         String sql = "SELECT * FROM Member WHERE userid = ?";
         return this.jdbcTemplate.queryForObject(sql, new Object[] { userid }, new MyRowMapper());
      }

      @Override
      public List<MemberVO> readAll() {
         String sql = "SELECT * FROM Member ORDER BY userid DESC";
         return this.jdbcTemplate.query(sql, new MyRowMapper());
      }

      @Override
      public int update(MemberVO memberVo) {
         String sql = "UPDATE Member SET userage = ?, gender = ?, city = ? " + "WHERE userid = ?";
         return this.jdbcTemplate.update(sql, memberVo.getAge(), memberVo.getGender(),
         memberVo.getCity(),
               memberVo.getUserid());
      }

      @Override
      public int delete(String userid) {
         String sql = "DELETE FROM Member WHERE userid = ?";
         return this.jdbcTemplate.update(sql, userid);
      }
   }



---------------------------------------------------------
Task7. Form Data Validation
1. Package Explorer > right-click > New > Other > Spring > Spring Legacy Project
2. Select Spring MVC Project
3. Project name : FormValidationDemo > Next
4. Enter a topLevelPackage : com.example.biz > Finish
5. pom.xml 수정하기
   <properties>
      <java-version>11</java-version>
      <org.springframework-version>5.3.12</org.springframework-version>
      <org.aspectj-version>1.9.7</org.aspectj-version>
      <org.slf4j-version>1.7.32</org.slf4j-version>
   </properties>
      ...
   <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
   </dependency>
   <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.3</version>
      <scope>provided</scope>
   </dependency>
   <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.8.1</version>
      <scope>test</scope>
   </dependency>
   ...
   <build>
      <plugins>
         <plugin>
            <artifactId>maven-eclipse-plugin</artifactId>
            <version>2.10</version>
            ...
         </plugin>
```

```
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.1</version>
          <configuration>
            <source>11</source>
            <target>11</target>
            ...
        </plugin>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>3.0.0</version>
```

6. pom.xml > right-click > Run As > Maven install
   [INFO] BUILD SUCCESS

7. FormValidationDemo Project > right-click > Properties > Project Facets > Select Java > Change Version 11
   Select Runtimes Tab > Check Apache Tomcat v9.0 > Click Apply and Close

8. UserVO 객체 생성
   1)src/main/java/com.example.vo package 생성
   2)src/main/java/com.example.vo.UserVO class

```
package com.example.vo;

public class UserVO {
    private String name;
    private int age;
    private String userid;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getUserid() {
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    @Override
    public String toString() {
        return "UserVO [name=" + name + ", age=" + age + ", userid=" + userid + "]";
    }
}
```

9. Validator를 이용한 검증
   1)Data Command 객체에서 유효성 검사를 할 수 있다.
   2)UserValidator 객체 생성
   3)src/main/java/com.example.biz.UserValidator class

```
package com.example.biz;

import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

import com.example.vo.UserVO;

public class UserValidator implements Validator {

    @Override
    public boolean supports(Class<?> arg0) {
```

```
        //검증할 객체의 class 타입 정보를 반환
        return UserVO.class.isAssignableFrom(arg0);
    }

    @Override
    public void validate(Object obj, Errors errors) {
        System.out.println("검증시작");
        UserVO userVO = (UserVO)obj;

        String username = userVO.getName();
        if(username == null || username.trim().isEmpty()) {
            System.out.println("이름의 값이 빠졌습니다.");
            errors.rejectValue("name", "No Value");
        }

        int userage = userVO.getAge();
        if(userage == 0) {
            System.out.println("나이의 값이 빠졌습니다.");
            errors.rejectValue("age", "No Value");
        }

        String userid = userVO.getUserid();
        if(userid == null || userid.trim().isEmpty()) {
            System.out.println("아이디의 값이 빠졌습니다.");
            errors.rejectValue("userid", "No Value");
        }
    }
}
```

4)src/main/java/com.example.biz/HomeController.java

```
@RequestMapping(value = "/register", method=RequestMethod.GET)
public String register() {
    return "register";
}

@RequestMapping(value = "/register", method=RequestMethod.POST)
public String register(@ModelAttribute("userVO") UserVO userVO, BindingResult result) {
    String page = "register_ok";
    UserValidator validator = new UserValidator();
    validator.validate(userVO, result);
    if(result.hasErrors()) {
        page = "register";
    }
    return page;
}
```

5)src/main/webapp/WEB-INF/views/register.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원 가입 폼</title>
</head>
<body>
    <form action="/biz/register" method="post">
        Name : <input type="text" name="name" /><br />
        Age : <input type="number" name="age" /><br />
        ID : <input type="text" name="userid" /><br />
        <input type="submit" value="가입하기" />
    </form>
</body>
</html>
```

6)src/main/webapp/WEB-INF/views/register_ok.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="user" value="${userVO}" />
<!DOCTYPE html">
<html>
<head>
<meta charset="UTF-8">
<title>회원 가입 결과 창</title>
```

```
    </head>
    <body>
      <ul>
        <li>이름 : ${user.name}</li>
        <li>나이 : ${user.age}</li>
        <li>아이디 : ${user.userid}</li>
      </ul>
    </body>
    </html>
```

7)Test
http://localhost:8080/biz/register에서
이름, 나이, 아이디를 모두 입력하면 결과창으로 넘어오고
한 개라도 입력하지 않으면 다시 입력창으로 간다.

10. ValidataionUtils class를 이용한 검증
1)ValidatationUtils class는 validate() method를 좀 더 편리하게 사용할 수 있게 해줌.
2)UserValidator.java 수정

```
    /*String username = userVO.getName();
    if(username == null || username.trim().isEmpty()) {
      System.out.println("이름의 값이 빠졌습니다.");
      errors.rejectValue("name", "No Value");
    }*/

    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "No Value");
```

11. @Valid와 @InitBinder 이용하기
1)Spring Framework이 대신 검증해 줌
2)mvnrepository에서 'hibernate validator'로 검색

```
  <dependency>
      <groupId>org.hibernate.validator</groupId>
      <artifactId>hibernate-validator</artifactId>
      <version>6.0.18.Final</version>
  </dependency>
```

3)pom.xml에 넣고 Maven Clean > Maven Install
4)HomeController.java 수정

```
  @RequestMapping(value = "/register", method=RequestMethod.POST)
  public String register(@ModelAttribute("userVO") @Valid UserVO userVO, BindingResult result) {
    String page = "register_ok";
    //UserValidator validator = new UserValidator();
    //validator.validate(userVO, result);
    if(result.hasErrors()) {
      page = "register";
    }

    return page;
  }

  @InitBinder
  protected void initBinder(WebDataBinder binder) {
    binder.setValidator(new UserValidator());
  }
```

12. Test
http://localhost:8080/biz/register에서
이름, 나이, 아이디를 모두 입력하면 결과창으로 넘어오고
한 개라도 입력하지 않으면 다시 입력창으로 간다.

Task8. Convert J2EE to Spring MVC
1. In J2EE Perspective
2. Project Explorer > right-click > New > Dynamic Web Project
3. Project name : SpringWebDemo > Next
-Default output folder : build\classes > Next
-Content directory : WebContent

-Check [Generate web.xml deployment descriptor] > Finish

4. Convert to Maven Project
1)project right-click > Configure > Convert to Maven Project > Finish
2)Project : /SpringWebDemo
3)Group Id : SpringWebDemo
4)Artifact Id : SpringWebDemo
5)version : 0.0.1-SNAPSHOT
6)Packaging : war
7)Finish

5. Add Spring Project Nature
-project right-click > Spring Tools > Add Spring Project Nature

6. 새로 생성된 pom.xmlfile에 필요한 library 추가 > Maven Clean > Maven Install

```
<dependencies>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.12</version>
   </dependency>
   <dependency>
   <groupId>org.junit.jupiter</groupId>
   <artifactId>junit-jupiter-api</artifactId>
   <version>5.8.1</version>
   <scope>test</scope>
</dependency>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.3.12</version>
   </dependency>
</dependencies>
...
<build>
   <plugins>
      <plugin>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.8.1</version>
         <configuration>
            <release>11</release>
         </configuration>
      </plugin>
```

7. Spring mvc library 검색 및 설치
1)http://mvnrepository.com에서 'spring mvc'로 검색
2)pom.xml에 추가

```
   <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-webmvc</artifactId>
         <version>5.3.12</version>
   </dependency>
```

3)Java version setting
-SpringWebDemo project > right-click > Properties > Project Facets > Select Java > Change Version 11
-Select Runtimes Tab > Check Apache Tomcat v9.0 > Click Apply and Close

4)Maven Clean > Maven Install

8. Build path에 config foler 추가
1)project right-click > Build Path > Configure Build Path > Select [Source] tab
2)Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
3)Folder name : config > Finish > OK > Apply and Close
4)Java Resources > config 폴더 확인

9. config folder에 beans.xml file 생성
1)Spring Perspective로 전환

2)config > right-click > New > Other > Spring > Spring Bean Configuration File > beans.xml
3)생성시 beans,context 체크

```
<?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:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="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">


</beans>
```

10. ContextLoaderListener class 설정
1)web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener - ContextLoaderListener]를 선택하면 아래의 code가 자동 삽입

```
<!-- needed for ContextLoaderListener -->
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>location</param-value>
</context-param>

<!-- Bootstraps the root web application context before servlet initialization -->
<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

2)아래 code로 변환

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>classpath:beans.xml</param-value>
</context-param>
```

11. DispatcherServlet Class 추가
1)web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet - DispatcherServlet declaration] 선택하면 아래의 code가 자동 추가된다.

```
<!-- The front controller of this Spring Web application, responsible for handling all application requests -->
<servlet>
   <servlet-name>springDispatcherServlet</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>location</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->
<servlet-mapping>
   <servlet-name>springDispatcherServlet</servlet-name>
   <url-pattern>url</url-pattern>
</servlet-mapping>
```

2)아래의 code로 변환

```
<init-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>classpath:beans*.xml</param-value>
</init-param>

<servlet-mapping>
   <servlet-name>springDispatcherServlet</servlet-name>
   <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

12. mvnrepository에서 'jstl'로 검색 후 설치
1)목록에서 2번째 : 1.2버전

```
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
```

```
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
```

2)pom.xml에 붙여넣고 Maven Clean > Maven Install

13. Hello Controller 작성

1)src/com.example.vo package 생성

2)src/com.example.vo.HelloVO class 생성

```
package com.example.vo;

public class HelloVO {
   private String name;

   public void setName(String name) {
      this.name = name;
   }

   public String sayHello() {
      return "Hello " + name;
   }
}
```

3)src/com.example.controller package 생성

4)com.example.controller.HelloController class 생성

```
package com.example.controller;

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 com.example.vo.HelloVO;

@Controller
public class HelloController {
   @Autowired
   private HelloVO helloBean;

   @RequestMapping("/hello.do")
   public String hello(Model model) {
      String msg = helloBean.sayHello();
      model.addAttribute("greet", msg);
      return "hello.jsp";
   }
}
```

14. beans.xml 수정

```
    <context:component-scan base-package="com.example" />

    <bean id="helloVO" class="com.example.vo.HelloVO">
       <property name="name" value="한지민" />
    </bean>
```

15. WebContent/hello.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Insert title here</title>
   </head>
   <body>
      ${greet}
   </body>
</html>
```

16. project > right-click > Run As > Run on Server > Finish

17. http://localhost:8080/SpringWebDemo/hello.do
Hello 한지민

Task9. Convert J2EE to Spring MVC
1. In J2EE Perspective
2. Project Explorer > right-click > New > Dynamic Web Project

3. Project name : SpringWebDemo > Next
-Default output folder : build\classes > Next
-Content directory : WebContent
-Check [Generate web.xml deployment descriptor] > Finish

4. Convert to Maven Project
-project right-click > Configure > Convert to Maven Project > Finish

5. Add Spring Project Nature
-project right-click > Spring Tools > Add Spring Project Nature

6. 새로 생성된 pom.xmlfile에 필요한 library 추가 > Maven Clean > Maven Install

```
<dependencies>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.12</version>
   </dependency>
   <dependency>
   <groupId>org.junit.jupiter</groupId>
   <artifactId>junit-jupiter-api</artifactId>
   <version>5.8.1</version>
   <scope>test</scope>
</dependency>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.3.12</version>
   </dependency>
   <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
   </dependency>
   <dependency>
      <groupId>com.oracle</groupId>
      <artifactId>ojdbc6</artifactId>
      <version>11.2</version>
   </dependency>
   <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-webmvc</artifactId>
         <version>5.3.12</version>
   </dependency>
</dependencies>
...
...
<build>
   <plugins>
      <plugin>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.8.1</version>
         <configuration>
            <release>11</release>
         </configuration>
      </plugin>
```

2)Java version setting
-SpringWebDemo project > right-click > Properties > Project Facets > Select Java > Change Version 11

-Select Runtimes Tab > Check Apache Tomcat v9.0 > Click Apply and Close

3)Maven Clean > Maven Install

7. Build path에 config foler 추가
1)project right-click > Build Path > Configure Build Path > Select [Source] tab
2)Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
3)Folder name : config > Finish > OK > Apply and Close
4)Java Resources > config 폴더 확인

8. config folder에 beans.xml file 생성
1)Spring Perspective로 전환
2)config Folder > right-click > New > Spring Bean Configuration File
3)File name : beans.xml
4)생성시 beans,context, 체크

```
<?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:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="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">


</beans>
```

9. ContextLoaderListener class 설정
1)web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener - ContextLoaderListener]를 선택하면 아래의 코드가 자동 삽입

```
<!-- needed for ContextLoaderListener -->
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>location</param-value>
</context-param>

<!-- Bootstraps the root web application context before servlet initialization -->
<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

2)아래 코드로 변환

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>classpath:beans.xml</param-value>
</context-param>
```

10. DispatcherServlet Class 추가
1)web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet - DispatcherServlet declaration] 선택하면 아래의 코드가 자동 추가된다.

```
<!-- The front controller of this Spring Web application, responsible for handling all application requests -->
<servlet>
   <servlet-name>springDispatcherServlet</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>location</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->
<servlet-mapping>
   <servlet-name>springDispatcherServlet</servlet-name>
   <url-pattern>url</url-pattern>
</servlet-mapping>
```

2)아래의 코드로 변환

```
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:beans*.xml</param-value>
    </init-param>

    <servlet-mapping>
      <servlet-name>springDispatcherServlet</servlet-name>
      <url-pattern>*.do</url-pattern>
    </servlet-mapping>
```

11. UserVO class 생성
  1)src/com.example.vo package 생성
  2)src/com.example.vo.UserVO class 생성

```
    package com.example.vo;

    public class UserVO {
      private String userId;
      private String name;
      private String gender;
      private String city;
      public UserVO() {}
      public UserVO(String userId, String name, String gender, String city) {
        this.userId = userId;
        this.name = name;
        this.gender = gender;
        this.city = city;
      }
      public String getUserId() {
        return userId;
      }
      public void setUserId(String userId) {
        this.userId = userId;
      }
      public String getName() {
        return name;
      }
      public void setName(String name) {
        this.name = name;
      }
      public String getGender() {
        return gender;
      }
      public void setGender(String gender) {
        this.gender = gender;
      }
      public String getCity() {
        return city;
      }
      public void setCity(String city) {
        this.city = city;
      }
      @Override
      public String toString() {
        return "UserVO [userId=" + userId + ", name=" + name + ", gender=" + gender + ", city=" + city +
        "]";
      }
    }
```

12. UserDao 객체 생성
  1)src/com.example.dao package 생성
  2)src/com.example.dao.UserDao interface

```
    package com.example.dao;

    import java.util.List;

    import com.example.vo.UserVO;

    public interface UserDao {
      void insert(UserVO user);
```

```
    List<UserVO> readAll();

    void update(UserVO user);

    void delete(String id);

    UserVO read(String id);
}
```

-src/com.example.dao.UserDaoImplJDBC.java 생성

```
package com.example.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import javax.sql.DataSource;

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

import com.example.vo.UserVO;

@Repository("userDao")
public class UserDaoImplJDBC implements UserDao {

    private JdbcTemplate jdbcTemplate;

    @Autowired
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    class UserMapper implements RowMapper<UserVO> {
        public UserVO mapRow(ResultSet rs, int rowNum) throws SQLException {
            UserVO user = new UserVO();
            user.setUserId(rs.getString("userId"));
            user.setName(rs.getString("name"));
            user.setGender(rs.getString("gender"));
            user.setCity(rs.getString("city"));
            return user;
        }
    }

    @Override
    public void insert(UserVO user) {
        String SQL = "INSERT INTO users (userid, name, gender,city) VALUES (?, ?, ?, ?)";
        jdbcTemplate.update(SQL, user.getUserId(), user.getName(), user.getGender(), user.getCity());

        System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" + user.getName());
    }

    @Override
    public List<UserVO> readAll() {
        String SQL = "SELECT * FROM users";
        List<UserVO> userList = jdbcTemplate.query(SQL, new UserMapper());
        return userList;
    }

    @Override
    public void update(UserVO user) {
        String SQL = "UPDATE users  SET name = ?, gender = ?, city = ? WHERE userid = ?";
        jdbcTemplate.update(SQL, user.getName(), user.getGender(), user.getCity(), user.getUserId());
        System.out.println("갱신된 Record with ID = " + user.getUserId());
    }

    @Override
    public void delete(String id) {
        String SQL = "DELETE FROM users WHERE userid = ?";
        jdbcTemplate.update(SQL, id);
```

```
            System.out.println("삭제된 Record with ID = " + id);
        }

        @Override
        public UserVO read(String id) {
            String SQL = "SELECT * FROM users WHERE userid = ?";
            try {
                UserVO user = jdbcTemplate.queryForObject(SQL, new Object[] { id }, new UserMapper());
                return user;
            } catch (EmptyResultDataAccessException e) {
                return null;
            }
        }
    }
```

13. UserService 객체 생성

1)src/com.example.service package 생성

2)src/com.example.service.UserService interface

```
package com.example.service;

import java.util.List;

import com.example.vo.UserVO;

public interface UserService {
    void insertUser(UserVO user);

    List<UserVO> getUserList();

    void deleteUser(String id);

    UserVO getUser(String id);

    void updateUser(UserVO user);
}
```

3)src/com.example.service.UserServiceImpl.java

```
package com.example.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.dao.UserDao;
import com.example.vo.UserVO;

@Service("userService")
public class UserServiceImpl implements UserService {
    @Autowired
    UserDao userDao;

    @Override
    public void insertUser(UserVO user) {
        this.userDao.insert(user);
    }

    @Override
    public List<UserVO> getUserList() {
        return this.userDao.readAll();
    }

    @Override
    public void deleteUser(String id) {
        this.userDao.delete(id);
    }

    @Override
    public UserVO getUser(String id) {
        return this.userDao.read(id);
    }
```

```
    @Override
    public void updateUser(UserVO user) {
        this.userDao.update(user);
    }
}
```

14. UserController 객체 생성
    1)src/com.example.controller package 생성
    2)com.example.controller.UserController class 생성

```
package com.example.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import com.example.service.UserService;

@Controller
public class UserController {
    @Autowired
    private UserService userService;

    @RequestMapping("/userInfo.do")
    public String getUserList(@RequestParam("userId") String userId, Model model) {
        UserVO user = userService.getUser(userId);
        //System.out.println(user);
        model.addAttribute("user", user);
        return "userInfo.jsp";
    }
}
```

15. config/dbinfo.properties file 생성

```
db.driverClass=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@localhost:1521:XE
db.username=hr
db.password=hr
```

16. beans.xml 수정

```
<context:component-scan base-package="com.example" />

<context:property-placeholder location="classpath:dbinfo.properties" />
<bean id="dataSource" class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
    <property name="driverClass" value="${db.driverClass}" />
    <property name="url" value="${db.url}" />
    <property name="username" value="${db.username}" />
    <property name="password" value="${db.password}" />
</bean>
```

17. WebContent/index.jsp 생성

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:redirect url="userInfo.do" />
```

18. WebContent/userInfo.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <c:set var="user" value="${user}"/>
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
    </head>
    <body>
        <h1>userInfo.jsp</h1>
```

```
        <h2>사용자 정보</h2>
        아이디 : ${user.userId} <br />
        이름 : ${user.name} <br />
        성별 : ${user.gender} <br />
        도시 : ${user.city} <br />
    </body>
    </html>
```

19. project > right-click > Run As > Run on Server > Finish

20. http://localhost:8080/SpringWebDemo/userinfo.do?userId=scott

---

Task10. File Upload with Spring MVC

1. In J2EE Perspective
2. Project Explorer > right-click > New > Dynamic Web Project
3. Project name : FileUploadDemo > Next > Check [Generate web.xml deployment descriptor] > Finish
4. Convert to Maven Project
   1)project right-click > Configure > Convert to Maven Project > Finish

5. Add Spring Project Nature
   1)project right-click > Spring Tools > Add Spring Project Nature

6. 새로 생선된 pom.xmlfile에 필요한 library 추가 > Maven Clean > Maven Install

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.12</version>
    </dependency>
    <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-webmvc</artifactId>
         <version>5.3.12</version>
    </dependency>
</dependencies>
```

7. ContextLoaderListener class 설정
   1)web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener - ContextLoaderListener]를 선택하면 아래의 코드가 자동 삽입

```
    <!-- needed for ContextLoaderListener -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>location</param-value>
    </context-param>

    <!-- Bootstraps the root web application context before servlet initialization -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
```

   2)아래 코드로 변환

```
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
```

8. DispatcherServlet Class 추가
   1)web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet - DispatcherServlet declaration] 선택하면 아래의 코드가 자동 추가된다.

```
    <!-- The front controller of this Spring Web application, responsible for handling all application requests -->
    <servlet>
        <servlet-name>springDispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>location</param-value>
```

```
            </init-param>
            <load-on-startup>1</load-on-startup>
        </servlet>

        <!-- Map all requests to the DispatcherServlet for handling -->
        <servlet-mapping>
            <servlet-name>springDispatcherServlet</servlet-name>
            <url-pattern>url</url-pattern>
        </servlet-mapping>
```

2)아래의 코드로 변환

```
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:beans.xml</param-value>
        </init-param>

        <servlet-mapping>
            <servlet-name>springDispatcherServlet</servlet-name>
            <url-pattern>/</url-pattern>
        </servlet-mapping>
```

9. FileUpload library 추가

1)Apache에서 제공하는 Common FileUpload library를 사용하여 file upload를 처리하기 위한 library

2)mvnrepository에서 'common fileupload'라고 검색하여 library 추가

```
        <dependency>
            <groupId>commons-fileupload</groupId>
            <artifactId>commons-fileupload</artifactId>
            <version>1.4</version>
        </dependency>
```

3)mvnrepository에서 'commons io'라고 검색하여 library 추가

```
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.6</version>
        </dependency>
```

4)Maven Clean > Maven Install

10. Thumbnail Image Library 추가

1)mvnrepository에서 'imgscalr-lib'라고 검색하여 libary 추가

```
        <dependency>
            <groupId>org.imgscalr</groupId>
            <artifactId>imgscalr-lib</artifactId>
            <version>4.2</version>
        </dependency>
```

2)Maven Clean > Maven Install

11. Build path에 config Foler 추가

1)project right-click > Build Path > Configure Build Path > Select [Source] tab

2)Click [Add Folder] > Select 현재 project > Click [Create New Folder...]

3)Folder name : config > Finish > OK > Apply and Close

4)Java Resources > config 폴더 확인

12. config Folder에 applicationContext.xml file 생성

1)config > right-click > New > Other > Spring > Spring Bean Configuration File

2)Name : applicationContext.xml

3)Namespace Tab에서 context, mvc check 할 것

```
        <?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:context="http://www.springframework.org/schema/context"
            xmlns:mvc="http://www.springframework.org/schema/mvc"
            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">

    <context:component-scan
          base-package="com.example" />
       <mvc:annotation-driven />
    </beans>
```

13. config Folder에 beans.xml file 생성
   1)config > right-click > New > Other > Spring > Spring Bean Configuration File
   2)Name : beans.xml
   3)beans.xml에 Spring mulipartResolver 추가

```
    <bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
       <property name="maxUploadSize" value="10240000" />
       <property name="defaultEncoding" value="utf-8" />
    </bean>
```

14. web.xml에 한글 File Encoding 처리하기
   1)한글 File이 Upload될 때 File 명이 깨지는 것을 해결하기 위해 web.xml에 아래 내용을 추가한다.

```
       <filter>
          <filter-name>encodingFilter</filter-name>
          <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
          <init-param>
             <param-name>encoding</param-name>
             <param-value>UTF-8</param-value>
          </init-param>
          <init-param>
             <param-name>forceEncoding</param-name>
             <param-value>true</param-value>
          </init-param>
       </filter>
       <filter-mapping>
          <filter-name>encodingFilter</filter-name>
          <url-pattern>/*</url-pattern>
       </filter-mapping>
```

15. View 작성
   1)WebContent/form.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
    </head>
    <body>
       <h1>file 업로드 예제</h1>
       <form method="post" action="upload" enctype="multipart/form-data">
          <label>email:</label> <input type="text" name="email"> <br>
          <br> <label>file:</label> <input type="file" name="file1">
          <br>
          <br> <input type="submit" value="upload">
       </form>
    </body>
    </html>
```

16. Service 작성
   1)src/com.example.service package
   2)src/com.example.service.FileUploadService.java

```
    package com.example.service;

    import java.io.FileOutputStream;
    import java.io.IOException;
    import java.util.Calendar;

    import org.springframework.stereotype.Service;
```

```
import org.springframework.web.multipart.MultipartFile;

@Service
public class FileUploadService {
  // 리눅스 기준으로 file 경로를 작성 ( 루트 경로인 /으로 시작한다. )
  // 윈도우라면 workspace의 드라이브를 파악하여 JVM이 알아서 처리해준다.
  // 따라서 workspace가 C드라이브에 있다면 C드라이브에 upload 폴더를 생성해 놓아야 한다.
  private static final String SAVE_PATH = "/upload";
  private static final String PREFIX_URL = "/upload/";

  public String restore(MultipartFile multipartFile) {
    String uri = null;

    try {
      // file 정보
      String originFilename = multipartFile.getOriginalFilename();
      String extName = originFilename.substring(originFilename.lastIndexOf("."), originFilename.length());
      Long size = multipartFile.getSize();

      // 서버에서 저장 할 file 이름
      String saveFileName = genSaveFileName(extName);

      System.out.println("originFilename : " + originFilename);
      System.out.println("extensionName : " + extName);
      System.out.println("size : " + size);
      System.out.println("saveFileName : " + saveFileName);

      writeFile(multipartFile, saveFileName);
      uri = PREFIX_URL + saveFileName;
    }
    catch (IOException e) {
      // 원래라면 RuntimeException 을 상속받은 예외가 처리되어야 하지만
      // 편의상 RuntimeException을 던진다.
      // throw new FileUploadException();
      throw new RuntimeException(e);
    }
    return uri;
  }


  // 현재 시간을 기준으로 file 이름 생성
  private String genSaveFileName(String extName) {
    String fileName = "";

    Calendar calendar = Calendar.getInstance();
    fileName += calendar.get(Calendar.YEAR);
    fileName += calendar.get(Calendar.MONTH);
    fileName += calendar.get(Calendar.DATE);
    fileName += calendar.get(Calendar.HOUR);
    fileName += calendar.get(Calendar.MINUTE);
    fileName += calendar.get(Calendar.SECOND);
    fileName += calendar.get(Calendar.MILLISECOND);
    fileName += extName;

    return fileName;
  }


  // file을 실제로 write 하는 메서드
  private boolean writeFile(MultipartFile multipartFile, String saveFileName)
                  throws IOException{
    boolean result = false;

    byte[] data = multipartFile.getBytes();
    FileOutputStream fos = new FileOutputStream(SAVE_PATH + "/" + saveFileName);
    fos.write(data);
    fos.close();

    return result;
  }
}
```

3)SAVE_PATH는 file을 저장할 위치를 가리킨다.
-일반적으로 server는 Linux 기반이므로 Linux 경로명을 사용하는 것이 좋다.

-즉 file을 root 경로인 / 아래의 upload folder에 저장하겠다는 의미인데, Windows에서는 JVM이 알아서 workspace가 존재하는 drive의 위치를 찾아서 drive를 root 경로로 하여 upload folder에 저장한다.
-예를들어 Eclipse workspace가 C drive에 있다면 C drive의 upload folder에 file이 저장될 것이다.
4)PREFIX_URL은 저장된 file을 JSP에서 불러오기 위한 경로를 의미한다.
5)MultipartFile 객체는 file의 정보를 담고 있다.
6)uri을 반환하는 이유는 view page에서 바로 image file을 보기 위함이다.
-만약 DB에서 image 경로를 저장 해야 한다면, 이와 같이 uri을 반환하면 좋을 것이다.
7)현재 시간을 기준으로 file 이름을 바꾼다.
-이렇게 하는 이유는, 여러 사용자가 올린 file의 이름이 같을 경우 덮어 씌어지는 문제가 발생하기 때문이다.
-따라서 file 이름이 중복될 수 있는 문제를 해결하기 위해 ms단위의 시스템 시간을 이용하여 file 이름을 변경한다.
8)FileOutputStream 객체를 이용하여 file을 저장한다.

17. Controller 작성
1)com.example.controller package
2)com.example.controller.FileUploadController.java

```
package com.example.controller;

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;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;

import com.example.service.FileUploadService;

@Controller
public class FileUploadController {
   @Autowired
   FileUploadService fileUploadService;

   @RequestMapping("/form")
   public String form() {
      return "form.jsp";
   }

   @RequestMapping(value = "/upload", method = RequestMethod.POST)
   public String upload(@RequestParam("email") String email, @RequestParam("file1") MultipartFile file,
   Model model) {
      String uri = fileUploadService.restore(file);
      model.addAttribute("uri", uri);
      return "result.jsp";
   }
}
```

18. WebContent/result.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
   <h1>Upload completed</h1>
   <div class="result-images">
      <img src="${pageContext.request.contextPath }${uri }" style="width: 150px">
   </div>
   <p>
      <a href='/FileUploadDemo/form'> 다시 업로드 하기 </a>
   </p>
</body>
</html>
```

19. C:/(현재 workspace가 C:라면)upload Folder 생성할 것

20. Project > right-click > Run As > Run on Server
http://localhost:8080/FileUploadDemo/form

21. 문제점 및 해결

1)Upload Folder(C:/upload)를 보면 File이 Upload된 것을 확인할 수 있지만, 결과 화면을 보면 Image가 제대로 출력 되지 않을 것이다.
2)Image File을 right-click하여 경로를 보면 아마 다음과 같을 것이다.
3)http://localhost:8080/FileUploadDemo/upload/업로드한 파일
4)File을 저장할 때 [upload]라는 Folder에 저장을 했는데, File을 저장할 때의 Upload는 C Drive 내의 [upload] Folder이고,
5)위 URL에서 [upload]는 Application 상 경로에 있는 upload이므로 WEB-INF 폴더의 하위 folder로서의 upload를 의미한다.
6)즉 실제 File이 저장된 Server 상의 위치(물리 주소)와 Application에서 보여주고자 하는 File 경로(가상 주소)가 일치하지 않은 것이다.
7)따라서 실제 File이 저장되어 있는 위치와 Application 상의 위치를 일치시키는 작업이 필요하다.
8)beans.xml에 물리 주소와 가상 주소를 mapping 해주는 code를 추가하도록 해야한다.

```
<!-- resource mapping -->
<!-- location : 물리적 주소 / mapping : 가상 주소 -->
<mvc:resources location="file:///C:/upload/" mapping="/upload/*"/>
```

9)이제 정상적으로 result.jsp에서 image가 출력될 것이다.

22. Multiple File Upload

1)이번에는 여러 개의 File을 Upload 할 수 있는 Multiple Upload를 알아보자.
2)수정할 부분은 <input> tag와 Controller에서 MultipartFile 객체를 받는 Parameter 부분 두 곳인데, 필요한 부분만 보자.
3)form.jsp

```
<input type="file" name="files" multiple>
```

4)<input> 태그에서는 multiple 속성만 추가하면 된다.
5)"File선택"을 클릭하면 ctrl 키를 눌러서 여러 개의 File을 선택할 수 있다.

6)FileUploadController

```
@RequestMapping( "/upload" )
public String upload(@RequestParam String email,
    @RequestParam(required=false) List<MultipartFile> files, Model model) {

        ...

}
```

7)Controller에서는 여러 개의 File을 받기 때문에 MultipartFile을 List로 받아야 한다.