01. 프로젝트 준비

- 10장을 위한 sp4-chap13폴더를 생성한다.
- 9장의 pom.xml과 폴더를 그대로 sp4-chap13폴더에 복사한다.
- sp4-chap13폴더의 pom.xml에서 <artifactId>태그의 값을 sp4-chap13으로 변경한다.
- 이클립스에서 sp4-chap13메이븐 프로젝트를 임포트한다.

02. 날짜를 이용한 회원 검색 기능

예제 1) src\main\java\spring\MemberDao.java

회원의 가입 일자를 기준으로 검색하는 기능을 구현하면서 몇 가지 스프링 MVC의 특징을 설정할 것이므로, 이를 위해 MemberDao 클래스에 메서드를 추가 한다.

```
package spring;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.Date;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;

public class MemberDao {

	private JdbcTemplate jdbcTemplate;

	public MemberDao(DataSource dataSource) {
		this.jdbcTemplate = new JdbcTemplate(dataSource);
	}

	public Member selectByEmail(String email) {
		List<Member> results = jdbcTemplate.query(
				"select * from MEMBER where EMAIL = ?",
```

```
                        new RowMapper<Member>() {
                            @Override
                            public Member mapRow(ResultSet rs, int
rowNum)
                                    throws SQLException {
                                Member member = new
Member(rs.getString("EMAIL"),

rs.getString("PASSWORD"),
                                        rs.getString("NAME"),

rs.getTimestamp("REGDATE"));
                                member.setId(rs.getLong("ID"));
                                return member;
                            }
                        },
                        email);

        return results.isEmpty() ? null : results.get(0);
    }

    public void insert(final Member member) {
        KeyHolder keyHolder = new GeneratedKeyHolder();
        jdbcTemplate.update(new PreparedStatementCreator() {
            @Override
            public                          PreparedStatement
createPreparedStatement(Connection con)
                    throws SQLException {
                PreparedStatement pstmt = con.prepareStatement(
                        "insert into MEMBER (EMAIL,
PASSWORD, NAME, REGDATE) "+
                        "values (?,?, ?, ?, ?)");
                pstmt.setLong(1, member.getId());
                pstmt.setString(2, member.getEmail());
                pstmt.setString(3, member.getPassword());
                pstmt.setString(4, member.getName());
                pstmt.setTimestamp(5,
                        n          e          w
Timestamp(member.getRegisterDate().getTime()));
                return pstmt;
            }
```

```
                });
        }

        public void update(Member member) {
                jdbcTemplate.update("update MEMBER set NAME = ?, PASSWORD =
? where EMAIL = ?",
                                member.getName(),           member.getPassword(),
member.getEmail());
        }

        public List<Member> selectAll() {
                List<Member> results = jdbcTemplate.query("select * from MEMBER",
                                new RowMapper<Member>() {
                        @Override
                        public Member mapRow(ResultSet rs, int rowNum)
                                        throws SQLException {
                                Member          member          =          new
Member(rs.getString("EMAIL"),
                                                rs.getString("PASSWORD"),
                                                rs.getString("NAME"),
                                                rs.getTimestamp("REGDATE"));
                                member.setId(rs.getLong("ID"));
                                return member;
                        }
                });
                return results;
        }

        public int count() {
                Integer count = jdbcTemplate.queryForObject("select count(*) from
MEMBER", Integer.class);
                return count;
        }

        public List<Member> selectByRegdate(Date from, Date to) {  //①
                List<Member> results = jdbcTemplate.query(
                                "select * from MEMBER where REGDATE between ?
and ? "+
                                "order by REGDATE desc",
                                new RowMapper<Member>() {
                                        @Override
```

```
                                        public Member mapRow(ResultSet rs, int
rowNum)
                                                throws SQLException {
                                            Member member = new
Member(rs.getString("EMAIL"),

rs.getString("PASSWORD"),
                                                    rs.getString("NAME"),

rs.getTimestamp("REGDATE"));
                                            member.setId(rs.getLong("ID"));
                                            return member;
                                        }
                                    },
                                    from, to);
                return results;
        }

        public Member selectById(Long id) {
                List<Member> results = jdbcTemplate.query(
                                    "select * from MEMBER where ID = ?",
                                    new RowMapper<Member>() {
                                        @Override
                                        public Member mapRow(ResultSet rs, int
rowNum)
                                                throws SQLException {
                                            Member member = new
Member(rs.getString("EMAIL"),

rs.getString("PASSWORD"),
                                                    rs.getString("NAME"),

rs.getTimestamp("REGDATE"));
                                            member.setId(rs.getLong("ID"));
                                            return member;
                                        }
                                    },
                                    id);

                return results.isEmpty() ? null : results.get(0);
        }
```

```
}
```

selectByRegdate()메서드는 REGDATE값이 두 파라미터로 전달받은 from과 to 사이에 있는 Member의 목록을 구해준다.
이 메서드를 이용해서 특정기간에 가입한 회원 목록을 보여주는 기능을 구현할 것이다.

03. 커맨드 객체 Data타입 프로퍼티 변환처리

예제 3) src\main\java\controller\ListCommand.java
회원이 가입한 일시를 기준으로 회원을 검색할 수 있도록 하기 위해 시작 기준과 끝 기준을 파라미터로 전달받는다고 하자. 시간을 표현하기 위해 자바의 java.util.Date타입을 이용해서 커맨드 클래스를 구현해 볼 수 있을 것이다.

```
package controller;

import java.util.Date;

import org.springframework.format.annotation.DateTimeFormat;

public class ListCommand {

	@DateTimeFormat(pattern="yyyyMMddHH")	// ①
	private Date from;
	@DateTimeFormat(pattern="yyyyMMddHH")	// ②
	private Date to;

	public Date getFrom() {
		return from;
	}

	public void setFrom(Date from) {
		this.from = from;
	}

	public Date getTo() {
		return to;
	}

	public void setTo(Date to) {
		this.to = to;
```

```
	}
}
```

스프링은 Long이나 int와 같은 기본 데이터타입으로의 변환은 처리해주지만, Data타입으로 변환하려면 추가 설정이 필요하다. ①, ②처럼 설정해 준다.

스프링의 MVC는 커맨드 객체에 @DateTimeFormat 애노테이션이 적용되어 있을 경우 @DateTimeFormat에서 지정한 형식을 이용해서 문자열을 Data타입으로 변환한다.

예제 4) src\main\java\controller\MemberListController.java
ListCommand클래스를 사용하는 컨트롤러 클래스는 별도 설정 없이 ListCommand클래스를 커맨드 객체로 사용하면된다.

```
package controller;

import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

import spring.Member;
import spring.MemberDao;

@Controller
public class MemberListController {

	private MemberDao memberDao;

	public void setMemberDao(MemberDao memberDao) {
		this.memberDao = memberDao;
	}

	@RequestMapping("/member/list")
	public String list(
			@ModelAttribute("cmd") ListCommand listCommand,
			Model model) {
```

```
                if (listCommand.getFrom() != null && listCommand.getTo() != null) {
                        List<Member> members = memberDao.selectByRegdate(
                                        listCommand.getFrom(),
listCommand.getTo());
                        model.addAttribute("members", members);
                }
                return "member/memberList";
        }
}
```

예제 5) src\main\webapp\WEB-INF\view\member\memberList.jsp
폼에 입력한 문자열이 커맨드 객체의 Date타입 프로퍼티로 잘 변환되는지 확인하기 위한 뷰 코드를 작성해 보자.
MemberListController 클래스의 list()메서드는 커맨드 객체로 받은 ListCommand 의 from 프로퍼티와 to 프로퍼티를 이용해서 해당 기간에 가입한 Member목록을 구하고, 뷰에 "members" 속성으로 전달한다.
이에 맞게 ListCommand객체를 위한 폼을 제공하고 member 속성을 이용해서 회원 목록을 출력하도록 뷰 코드를 구현하면 된다.

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
        <title>회원 조회</title>
</head>
<body>
        <form:form commandName="cmd">
        <p>
                <label>from: <form:input path="from" /></label>
                ~
                <label>to:<form:input path="to" /></label>
                <input type="submit" value="조회">
        </p>
        </form:form>

        <c:if test="${! empty members}">
        <table>
```

```
                <tr>
                        <th>아이디</th><th>이메일</th>
                        <th>이름</th><th>가입일</th>
                </tr>
                <c:forEach var="mem" items="${members}">
                <tr>
                        <td>${mem.id}</td>
                        <td><a href="<c:url value="/member/detail/${mem.id}"/>">
                                ${mem.email}</a></td>
                        <td>${mem.name}</td>
                        <td><fmt:formatDate value="${mem.registerDate}"
                                        pattern="yyyy-MM-dd"/></td>
                </tr>
                </c:forEach>
        </table>
        </c:if>
</body>
</html>
```

예제 5-1) 새로운 컨트롤러를 작성했으므로, spring-controller.xml파일에 빈 설정을 추가한다. src\main\resources\spring-controller.xml

```
... 생략
        <bean class="controller.MemberListController">
                <property name="memberDao" ref="memberDao" />
        </bean>
```

실행) http://localhost:8088/sp4-chap13/member/list
날짜입력시 yyyyMMddHH로 입력(예를 들어 2017011100 으로 시까지 입력해야 한다.)

3.1 변환에러 처리

폼에서 from 이나 to에 "20131201"을 입력해보자. 지정한 형식은 "**yyyMMddHH**"이기 때문에 "yyyyMMdd" 부분만 입력하면 지정한 형식과 일치하지 않아 400에러가 발생한다.

예제 6) src\main\java\controller\MemberListController.java
입력한 파라미터 값이 지정 형식에 맞지 않을 때, 400에러 대신 폼에 알맞은 에러 메시지를 보여주고 싶다면 Errors타입 파라미터를 @RequestMapping 적용 메서드에 추가해주면 된다.

```
package controller;
```

```
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

import spring.Member;
import spring.MemberDao;

@Controller
public class MemberListController {

	private MemberDao memberDao;

	public void setMemberDao(MemberDao memberDao) {
		this.memberDao = memberDao;
	}

	@RequestMapping("/member/list")
	public String list(
			@ModelAttribute("cmd") ListCommand listCommand,
			Errors errors, Model model) {
		if (errors.hasErrors()) {
			return "member/memberList";
		}
		if (listCommand.getFrom() != null && listCommand.getTo() != null) {
			List<Member> members = memberDao.selectByRegdate(
					listCommand.getFrom(),
listCommand.getTo());
			model.addAttribute("members", members);
		}
		return "member/memberList";
	}
}
```

예제 7) src\main\resources\message\label.properties

member.register=회원가입

... 생략

typeMismatch.java.util.Date=잘못된 형식

예제 8) src\main\webapp\WEB-INF\view\member\memberList.jsp
<form:errors>태그를 사용해서 에러 메시지를 출력하는 코드를 추가한다.

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
	<title>회원 조회</title>
</head>
<body>
	<form:form commandName="cmd">
	<p>
		<label>from: <form:input path="from" /></label>
		<form:errors path="from" />
		~
		<label>to:<form:input path="to" /></label>
		<form:errors path="to" />
		<input type="submit" value="조회">
	</p>
	</form:form>

	<c:if test="${! empty members}">
	<table>
		<tr>
			<th>아이디</th><th>이메일</th>
			<th>이름</th><th>가입일</th>
		</tr>
		<c:forEach var="mem" items="${members}">
		<tr>
```

```
                        <td>${mem.id}</td>
                        <td><a href="<c:url value="/member/detail/${mem.id}"/>">
                                ${mem.email}</a></td>
                        <td>${mem.name}</td>
                        <td><fmt:formatDate value="${mem.registerDate}"
                                        pattern="yyyy-MM-dd"/></td>
                </tr>
                </c:forEach>
        </table>
        </c:if>
</body>
</html>
```

04. MemberDao클래스 중복 코드 정리

sp4-chap12\src\main\java\spring\MemberDao.java의 코드에 있는 중복을 제거한다.

다음 코드를 보면 RowMapper객체를 생성하는 부분의 코드가 중복된 것을 확인할 수 있다.

```
package spring;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;

public class MemberDao {

        private JdbcTemplate jdbcTemplate;

        public MemberDao(DataSource dataSource) {
                this.jdbcTemplate = new JdbcTemplate(dataSource);
```

```
        }

        public Member selectByEmail(String email) {
                List<Member> results = jdbcTemplate.query(
                                "select * from MEMBER where EMAIL = ?",
                                new RowMapper<Member>() {
                                        @Override
                                        public Member mapRow(ResultSet rs, int
rowNum)
                                                        throws SQLException {
                                                Member member = new
Member(rs.getString("EMAIL"),

rs.getString("PASSWORD"),

rs.getString("NAME"),

rs.getTimestamp("REGDATE"));
                                                member.setId(rs.getLong("ID"));
                                                return member;
                                        }
                                },
                                email);

                return results.isEmpty() ? null : results.get(0);
        }

        public void insert(final Member member) {
                KeyHolder keyHolder = new GeneratedKeyHolder();
                jdbcTemplate.update(new PreparedStatementCreator() {
                        @Override
                        public                                  PreparedStatement
createPreparedStatement(Connection con)
                                        throws SQLException {
                                PreparedStatement               pstmt           =
con.prepareStatement(
                                                "insert into MEMBER (EMAIL,
PASSWORD, NAME, REGDATE) "+
                                                "values (?, ?, ?, ?)",
                                                new String[] {"ID"});
```

```
                                pstmt.setString(1,  member.getEmail());
                                pstmt.setString(2,  member.getPassword());
                                pstmt.setString(3,  member.getName());
                                pstmt.setTimestamp(4,
                                                n              e              w
Timestamp(member.getRegisterDate().getTime()));
                                return pstmt;
                        }
                }, keyHolder);
                Number keyValue = keyHolder.getKey();
                member.setId(keyValue.longValue());
        }

        public void update(Member member) {
                jdbcTemplate.update("update MEMBER set NAME = ?, PASSWORD
= ? where EMAIL = ?",
                                member.getName(),       member.getPassword(),
member.getEmail());
        }

        public List<Member> selectAll() {
                List<Member>  results  =  jdbcTemplate.query("select  *  from
MEMBER",
                                new RowMapper<Member>() {
                                        @Override
                                        public Member mapRow(ResultSet rs, int
rowNum)
                                                        throws SQLException {
                                                Member    member    =    new
Member(rs.getString("EMAIL"),

rs.getString("PASSWORD"),

rs.getString("NAME"),

rs.getTimestamp("REGDATE"));
                                                member.setId(rs.getLong("ID"));
                                                return member;
                                        }
                                });
```

```
                return results;
        }

        public int count() {
                Integer count = jdbcTemplate.queryForObject("select count(*)
from MEMBER", Integer.class);
                return count;
        }

}
```

예제 9) sp4-chap13\src\main\java\spring\MemberDao.java
RowMapper를 생성하는 부분의 코드 중복을 제거하기 위해 임의 객체를 필드에 할당하고 그 필드를 사용하도록 코드를 수정하자.

```
package spring;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.Date;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;

public class MemberDao {

        private JdbcTemplate jdbcTemplate;
        private RowMapper<Member> memRowMapper = new RowMapper<Member>()
{
```

```
		@Override
		public Member mapRow(ResultSet rs, int rowNum)
				throws SQLException {
			Member member = new Member(rs.getString("EMAIL"),
					rs.getString("PASSWORD"),
					rs.getString("NAME"),
					rs.getTimestamp("REGDATE"));
			member.setId(rs.getLong("ID"));
			return member;
		}
	};

	public MemberDao(DataSource dataSource) {
		this.jdbcTemplate = new JdbcTemplate(dataSource);
	}

	public Member selectByEmail(String email) {
		List<Member> results = jdbcTemplate.query(
				"select * from MEMBER where EMAIL = ?",
				memRowMapper,
				email);

		return results.isEmpty() ? null : results.get(0);
	}

	public void insert(final Member member) {
		KeyHolder keyHolder = new GeneratedKeyHolder();
		jdbcTemplate.update(new PreparedStatementCreator() {
			@Override
			public PreparedStatement
createPreparedStatement(Connection con)
					throws SQLException {
				PreparedStatement pstmt = con.prepareStatement(
						"insert into MEMBER (EMAIL,
PASSWORD, NAME, REGDATE) "+
						"values (?, ?, ?, ?)",
						new String[] {"ID"});
				pstmt.setString(1, member.getEmail());
				pstmt.setString(2, member.getPassword());
				pstmt.setString(3, member.getName());
				pstmt.setTimestamp(4,
```

```
                                                    n                    e                    w
Timestamp(member.getRegisterDate().getTime()));
                                    return pstmt;
                            }
                    }, keyHolder);
                    Number keyValue = keyHolder.getKey();
                    member.setId(keyValue.longValue());
            }

            public void update(Member member) {
                    jdbcTemplate.update("update MEMBER set NAME = ?, PASSWORD =
? where EMAIL = ?",
                                    member.getName(),           member.getPassword(),
member.getEmail());
            }

            public List<Member> selectAll() {
                    List<Member> results = jdbcTemplate.query("select * from MEMBER",
                                    memRowMapper);
                    return results;
            }

            public int count() {
                    Integer count = jdbcTemplate.queryForObject("select count(*) from
MEMBER", Integer.class);
                    return count;
            }

            public List<Member> selectByRegdate(Date from, Date to) {
                    List<Member> results = jdbcTemplate.query(
                                    "select * from MEMBER where REGDATE between ?
and ? "+
                                    "order by REGDATE desc",
                                    memRowMapper,
                                    from, to);
                    return results;
            }

            public Member selectById(Long id) {
                    List<Member> results = jdbcTemplate.query(
                                    "select * from MEMBER where ID = ?",
```

```
                                memRowMapper,
                                id);

                return results.isEmpty() ? null : results.get(0);
        }


}
```

05. @PathVariable을 이용한 경로변경처리
ID가 10인 회원의 정보를 조회하기 위한 URL을 구성할 때 다음과 같이 ID값을 요청 경로에 포함시키는 방법을 사용할 수 있다.
http://localhost:8088/sp4-chap13/member/detail/216135209

각 회원의 ID값이 다르므로 회원마다 경로의 마지막 부분이 달라진다.
이렇게 경로의 특정부분의 값이 고정되어 있지 않고 달라질 때 사용할 수 있는 것이 @PathVriable이다.

예제 10) src\main\java\controller\MemberDetailController.java
@PathVariable을 사용하여 가변경로를 처리한다.

```
package controller;

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 spring.Member;
import spring.MemberDao;
import spring.MemberNotFoundException;

@Controller
public class MemberDetailController {

        private MemberDao memberDao;

        public void setMemberDao(MemberDao memberDao) {
                this.memberDao = memberDao;
```

```
	}

	@RequestMapping("/member/detail/{id}")	// 21행
	public String detail(@PathVariable("id") Long memId, Model model) {  //22행
		Member member = memberDao.selectById(memId);
		if (member == null) {
			throw new MemberNotFoundException();
		}
		model.addAttribute("member", member);
		return "member/memberDetail";
	}

}
```

매핑경로에 "{경로변수}"와 같이 중괄호로 둘러싸인 부분을 경로변수라고 한다,
"{경로변수}"에 해당하는 값은 같은 경로 변수 이름을 지정한 @PathVariable파라미터에 전달된다.
21~22행의 경우 "/member/detail/{id}"에서 {id}에 해당하는 부분의 경로 값을 @PathVariable("id")애노테이션이 적용된 memId파라미터에 전달한다.
요청한 경로가 /member/detail/10이면, {id}에 해당하는 "10"이 memId파라미터에 값으로 전달된다.

예제 11) src\main\resources\spring-controller.xml
빈으로 등록한다.

```
<?xml version="1.0" encoding="UTF-8"?>

... 생략
	<bean class="controller.MemberDetailController">
		<property name="memberDao" ref="memberDao" />
	</bean>
</beans>
```

예제 12) src\main\webapp\WEB-INF\view\member\memberDetail.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
```

```
<html>
<head>
	<title>회원 정보</title>
</head>
<body>
	<p>아이디: ${member.id}</p>
	<p>이메일: ${member.email}</p>
	<p>이름: ${member.name}</p>
	<p>가입일: <fmt:formatDate value="${member.registerDate}"
				pattern="yyyy-MM-dd HH:mm"/> </p>
</body>
</html>
```

서버를 시작하고 웹브라우저에
"http://localhost:8088/sp4-xhap13/member/detail/209162536"를 입력해보자.

06. 컨트롤러 익셉션 처리하기

예제 12를 실행시켰을 때 ID가 존재할 때는 정상적으로 출력결과가 화면에 출력되지만 존재하지 않는 ID를 경로에 준다면 MemberNotFoundException이 발생하는 것을 알 수 있다.
경로 변수는 Long타입인데, 실제 요청경로에 숫자가 아닌 문자를 입력했을 때,
즉, http://localhost:8088/sp4-chap13/member/detail/a 주소를 입력하면 400에러가 발생할 것이다,

익셉션 화면이 보이는 것보다는 알맞게 익셉션을 처리해서 사용자에게 더 적합한 안내를 해주는 것이 좋은 방법일 것이다.

예제 13) src\main\java\controller\MemberDetailController.java
MemberNotFoundException의 경우 try~catch 로 잡은 뒤 안내 화면을 보여주는 뷰를 보여주면 된다.
그런데 타입 변환 실패에 따른 익셉션은 @ExceptionHandler 애노테이션을 사용하면 된다.

```
package controller;

import java.io.IOException;

import org.springframework.beans.TypeMismatchException;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import spring.Member;
import spring.MemberDao;
import spring.MemberNotFoundException;

@Controller
public class MemberDetailController {

	private MemberDao memberDao;

	public void setMemberDao(MemberDao memberDao) {
		this.memberDao = memberDao;
	}

	@RequestMapping("/member/detail/{id}")
	public String detail(@PathVariable("id") Long memId, Model model) {
		Member member = memberDao.selectById(memId);
		if (member == null) {
			throw new MemberNotFoundException();
		}
		model.addAttribute("member", member);
		return "member/memberDetail";
	}

	@ExceptionHandler(TypeMismatchException.class)
	public String handleTypeMismatchException(TypeMismatchException ex) {
		// ex 사용해서 로그 남기는 등 작업
		return "member/invalidId";
	}

	@ExceptionHandler(MemberNotFoundException.class)
	public String handleNotFoundException() throws IOException {
		return "member/noMember";
	}
}
```

예제 13-1) src\main\webapp\WEB-INF\view\member\noMember.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>회원 없음</title>
</head>
<body>
	<p>
		존재하지 않는 회원입니다.
	</p>
</body>
</html>
```

예제 13-2) src\main\webapp\WEB-INF\view\member\invalidId.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<!DOCTYPE html>
<html>
<head>
	<title>오류</title>
</head>
<body>
	<p>
		잘못된 요청입니다.
	</p>
</body>
</html>
```