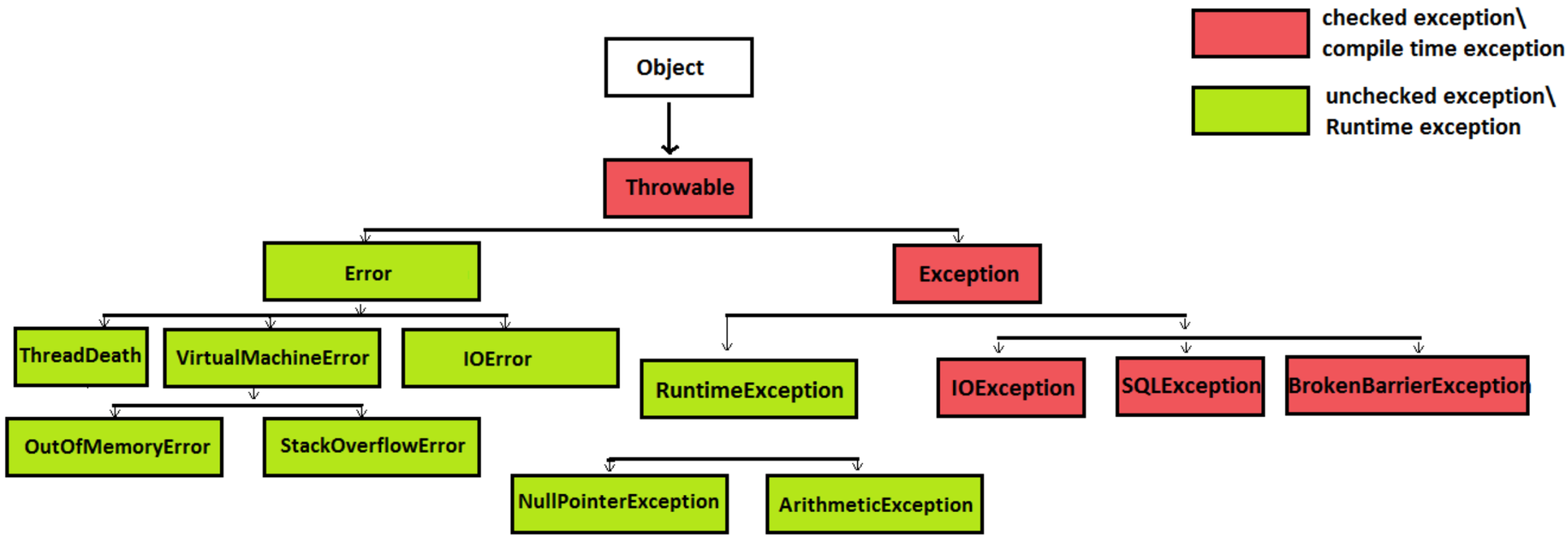
에러(error)와 예외(exception)

프로그램의 예외상황은 크게 에러(error)와 예외(exception)로 나눌 수 있습니다. 에러는 시스템 자체에서 비정상적인 상황이 발생하여 프로그램이 계속 실행될 수 없는 경우를 말하며, 예외는 내부적 처리를 통해 프로그램을 계속 실행가능한 상태로 만들 수 있는 경우를 말합니다.

자바에서는 이런 예외상황을 처리하기 위해 `Throwable` 인터페이스를 만들어두었습니다. 에러를 처리하는 `Error` 클래스와 예외를 처리하는 `Exception` 클래스는 Throwable 인터페이스를 상속받습니다.



에러는 대응할 방법이 없기때문에 Error 클래스는 주로 JVM에서 사용하며, 어플리케이션에서 사용하지는 않습니다. 대표적인 에러로는 스택의 공간이 부족한 경우 발생되는 `StackOverflowError` 가 있습니다.

예외(exception)의 분류

예외에는 예외 처리가 필요한 `Checked Exception` 과 예외 처리가 불필요한 `Unchecked Exception` 이 있습니다.

Checked Exception 은 컴파일 시점에서 예외가 발생할 가능성이 있기 때문에 반드시 예외처리를 해야합니다. 대표적으로 파일을 불러올 때 파일이 없을 경우인 FileNotFoundException 이 있습니다.

Unchecked Exception 은 런타임 시점에서 예외가 발생할 가능성이 있는 경우이며, 예외처리를 하지 않아도 됩니다. Exception 클래스의 하위 클래스인 RuntimeException 클래스를 상속받은 클래스는 Unchecked Exception이 됩니다.

예외(exception)의 처리

자바에서 예외처리는 try-catch 문을 통해 이루어질 수 있습니다.

throw 키워드를 통해 예외를 던질 수 있으며, throw된 예외는 catch문을 통해 받을 수 있습니다. 즉, Checked Exception이 throw 된다면 해당 메소드를 호출한 곳에서는 try-catch문을 통해 던져진 예외를 처리해야 합니다.

만약 메소드에 throws 키워드를 붙인다면 throw된 예외를 처리하지 않고 해당 메소드를 호출한 곳으로 다시 예외를 throw 할 수도 있습니다.

SimpleMappingExceptionResolver

이제 Exception을 통해 예외가 발생하면 에러 페이지를 response 할 수 있도록 코드를 작성해보겠습니다.

지난번 글에서 설명했듯이 스프링 프레임워크에서는 ExceptionHandlerExceptionResolver를 통해 @ExceptionHandler 어노테이션이 붙은 메소드를 호출하여 예외를 처리하고, ResponseStatusExceptionResolver를 통해 상태코드를 설정하며, ExceptionHandlerExceptionResolver에서 처리되지 않은 예외는 DefaultHandlerExceptionResolver가 처리하게 됩니다.

스프링 프레임워크는 컨테이너에 기본적으로 등록되지는 않지만 간단하게 적용할 수 있는 SimpleMappingExceptionResolver 가 있습니다. 이름 그대로 간단하게 예외를 매핑할 수 있는 ExceptionResolver입니다. 서블릿 컨테이너에서 web.xml에 error-page를 등록하여 에러페이지를 매핑한 것과 같이 SimpleMappingExceptionResolver의 프로퍼티를 설정하여 스프링 컨테이너의 예외를 예외 페이지 뷰와 매핑할 수 있습니다.

빈(bean)을 생성해야 하는데 레거시를 사용하고 있으니 xml 파일을 통해 빈(bean)을 생성하겠습니다. dispatcherServlet의 컨텍스트에 SimpleMappingExceptionResolver 빈을 등록하겠습니다.

/src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml

```
...
    <beans:bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
        <beans:property name="exceptionMappings">
```

```
            <beans:props>
                <beans:prop key="TestException">error/testException</beans:prop>
            </beans:props>
        </beans:property>
        <beans:property name="defaultErrorView" value="error/error" />
    </beans:bean>
...
```

SimpleMappingExceptionResolver 를 빈으로 등록하고 exceptionMappings 의 프로퍼티를 설정해주면 됩니다. 해당 프로퍼티는 key가 예외객체 이름이고 value가 뷰 이름인 맵으로 되어있습니다. 예외객체 이름은 패키지명을 입력하지 않아도 자동으로 매핑됩니다.

검증(validation)

클라이언트의 요청이 올바르지 않다면 예외처리를 통해 올바른 요청을 하도록 유도해야합니다. 이러한 과정을 검증(validation) 이라고 합니다. 웹 애플리케이션에서 검증은 중요한 과정입니다.

검증과정을 알아보기위해 임시로 count와 price로 구성된 ProductDTO를 작성해보겠습니다.

/kro/rubisco/dto/ProductDTO.java

```
package kro.rubisco.dto;

import lombok.Data;

@Data
public class ProductDTO {
    private Long count;
```

```
    private Long price;
}
```

검증과 관련된 객체에는 `BindingResult` , `FieldError` , `ObjectError` 가 있습니다.

FieldError 추가

BindingResult는 FieldError를 보관하는 객체입니다. FieldError는 필드의 타입에 맞지 않는 경우 스프링 컨테이너에 의하여 자동으로 생성되어 BindingResult에 저장되는데, 필요한 경우 검증을 통해 FieldError를 직접 생성하여 BindingResult에 넣어줄 수 있습니다.

FieldError의 생성자는 2개가 있습니다.

/kro/rubisco/dto/ProductDTO.java

```
public FieldError(String objectName, String field, String defaultMessage);
```

```
public FieldError(String objectName, String field, @Nullable Object rejectedValue, boolean bindingFailure, @Nullable String[] codes, @Nullable Object[] arguments, @Nullable String defaultMessage)
```

`objectName` 은 오류가 발생한 객체의 이름이며, `field` 는 오류가 발생한 필드의 이름입니다. `rejectedValue` 는 바인딩에 실패한 오류인지 검증에 실패한 오류인지를 나타내고, `codes` 는 메시지 코드를, `arguments` 는 메시지에서 사용할 인자를, `defaultMessage` 는 기본 오류 메시지를 나타냅니다.

우선 1번째 생성자를 통해 `defaultMessage` 를 지정하여 예외처리를 해보겠습니다.

컨트롤러를 작성하기 전에 `BindException` 을 상속받은 `BindExceptionWithViewName` 을 작성하겠습니다. BindException을 던져줄 때 뷰 이름도 같이 던지기 위함입니다.

/kro/rubisco/config/BindExceptionWithViewName.java

```
package kro.rubisco.config;

import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;
```

```
public class BindExceptionWithViewName extends BindException {

    private final String viewName;

    public BindExceptionWithViewName(BindingResult bindingResult, String viewName) {
        super(bindingResult);
        this.viewName = viewName;
    }

    public String getViewName() {
        return viewName;
    }

}
```

이제 컨트롤러를 작성하세요.

/kro/rubisco/controller/TestController.java

```
package kro.rubisco.controller;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import kro.rubisco.config.BindExceptionWithViewName;
```

```java
import kro.rubisco.dto.ProductDTO;

@Controller
@RequestMapping("/test")
public class TestController {

    @GetMapping()
    public void getTestView() {
    }

    @PostMapping()
    public String getTestView(ProductDTO product, BindingResult bindingResult) throws BindException {

        if(product.getCount() == null || product.getCount() < 10 || product.getCount() > 100){
            bindingResult.addError(new FieldError("product", "count", "수량은 10 이상 100 이하 입니다."));
        }

        if(product.getPrice() == null || product.getPrice() < 100 || product.getPrice() > 10_000_000){
            bindingResult.addError(new FieldError("product", "price", "가격은 100원 이상 10,000,000원 이하 입니다."));
        }

        if(bindingResult.hasErrors()) {
            throw new BindExceptionWithViewName(bindingResult, "/test");
        }

        return "redirect:/test";
    }
}
```

`TestController` 를 만들고 `/test` 에 매핑합니다. post 요청의 경우 `ProductDTO` 를 주입받아 처리하도록 합니다. ProductDTO 뒤에 `BindingResult` 를 주입했는데, 해당 객체가 없는 경우 모델에 대하여 binding이 실패하면 컨트롤러를 호출하지 않고 `BindException` 을 던져 컨테이너가 자동으로 예외처리를 해버립니다. 하지만 객체 뒤에 BindingResult 가 있으면 예외를 던지지 않고 우선 컨트롤러에 진입하게 됩니다.

이제 if문을 통해 검증을 하고, 예외상황인 경우 `FieldError` 를 만들어서 `bindingResult` 에 추가해주면 됩니다. 위에 코드에서는 count가 10 미만이거나 100 초과인 경우 count에 대하여 에러를 추가하고, price가 100 미만이거나 10,000,000 초과인 경우 price에 대하여 에러를 추가합니다.

bindingResult에 FieldError가 있는 경우 `BindException` 을 던져줍니다. BindException은 `bindingResult` 를 가지고 있으며, 이를 상속받아 작성한 `BindExceptionWithViewName` 으로 뷰의 이름도 같이 전달해줍시다. 예외가 없다면 컨트롤러 로직을 수행하고 PRG 패턴을 사용하여 get 방식으로 redirect 합니다.

이제 예외컨트롤러를 작성하겠습니다. `@ControllerAdvice` 어노테이션을 통해 전역적인 예외처리를 해주는 `ExceptionHandlingController` 를 작성합니다.

/kro/rubisco/controller/ExceptionHandlingController.java

```java
package kro.rubisco.controller;


import org.springframework.ui.Model;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import kro.rubisco.config.BindExceptionWithViewName;


@ControllerAdvice
public class ExceptionHandlingController {


    @ExceptionHandler(BindExceptionWithViewName.class)
    protected String handleBadRequest(BindExceptionWithViewName e, Model model) {
        model.addAttribute("errors", e);
        return e.getViewName();
    }
}
```

`@ExceptionHandler` 어노테이션을 통해 `BindExceptionWithViewName` 에 `handleBadRequest` 메소드를 매핑시킵니다. 해당 메소드에서는 모델에 Errors 객체를 추가하고, BindExceptionWithViewName을 통해 전달받은 뷰 이름을 반환합니다.

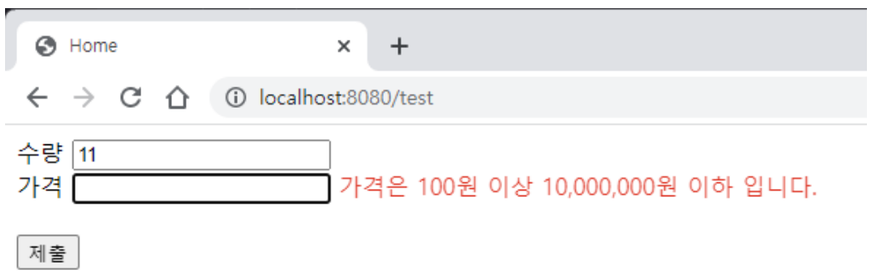
이제 뷰 템플릿을 작성하겠습니다.

/src/main/webapp/WEB-INF/views/test.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"  %>
<html>
<head>
    <title>Home</title>
</head>
<body>
<form  method="POST"  action="/test">
    <c:forTokens var="field" items="count,price" delims=",">
        <label>
            <c:choose>
                <c:when test="${field  eq  'count'}">수량</c:when>
                <c:when test="${field  eq  'price'}">가격</c:when>
            </c:choose>
            <input  type="text"  name="${field}"  value="${errors.getFieldValue(field)}"  />
            <c:if test="${errors.hasFieldErrors(field)}">
                <font  color="red">${errors.getFieldError(attr).defaultMessage}</font>
            </c:if>
        </label>
        <br>
    </c:forTokens>
    <br>
    <input  type="submit"  value="제출"  />
</form>
<c:if test="${errors.hasErrors()}">
<script>
[...document.forms[0].querySelectorAll("[name]")].filter(e=>!e.value)[0].focus();
</script>
```

```
</c:if>
</body>
</html>
```

JSP에서 EL의 경우 해당 변수가 null이라면 출력되지 않게 됩니다. errors 객체에서 getFieldValue 메소드는 해당 프로퍼티에 에러가 있으면 null값이 되어 출력되지 않고 그렇지 않으면 클라이언트에서 전송한 값을 출력합니다. 그러므로 검증을 통과하지 못한 필드는 빈값이 되고 에러 메시지가 출력됩니다.



에러코드를 통한 메시지 출력

이번에는 에러코드를 통해 메시지를 출력해보겠습니다. 방금 작성한 코드처럼 하드코딩으로 defaultMessage를 직접 작성해도 되지만, 메시지를 따로 관리하여 메시지를 코드를 통해 간접적으로 메시지를 출력할 수도 있습니다.

우선 메시지를 관리하기위한 `messageSource` 를 컨테이너에 빈으로 추가합니다. 다국어 처리가 가능한 `ReloadableResourceBundleMessageSource` 를 메시지 소스로 사용하도록 하겠습니다.

/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
  https://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 -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property value="oracle.jdbc.driver.OracleDriver" name="driverClassName"/>
        <property value="[DB 주소]" name="url"/>
        <property value="[DB 아이디]" name="username"/>
        <property value="[DB 암호]" name="password"/>
</bean>

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
       <property name="dataSource" ref="dataSource" />
      <property name="configLocation" value="classpath:/mybatis-config.xml"></property>
      <property name="mapperLocations" value="classpath:mappers/**/*Mapper.xml"></property>
</bean>

<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate" destroy-method="clearCache">
      <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory"></constructor-arg>
</bean>

<bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">

      <!-- Encoding 설정 -->
      <property name="defaultEncoding" value="UTF-8" />

      <!-- Reload Cache 설정 -->
      <property name="cacheSeconds" value="5" />

      <!-- BaseName 설정 -->
      <property name="basenames">
            <list>
                  <value>/WEB-INF/message/error</value>
```

```
            </list>
        </property>

    </bean>

    <context:component-scan base-package="kro.rubisco.service"></context:component-scan>
</beans>
```

basenames을 기준으로 `[basenames]_[언어코드]_[국가코드].properties` 형식으로 메시지 파일을 작성해야합니다. 예를 들어 한국어 메시지의 경우 `error_ko_KR.properties` 파일에, 영어 메시지의 경우 `error_en_US.properties` 파일에 작성하면 됩니다. 기본파일은 basename인 `error.properties` 에 작성합니다.

/src/main/webapp/WEB-INF/message/error.properties

```
range.product.count = 제품의 수량은 {0}개 이상 {1}개 이하로 입력할 수 있습니다.
range.product.price = 제품의 가격은 {0}원 이상 {1}원 이하로 입력할 수 있습니다.
```

{0}, {1}은 DataSource에서 전달되는 인수입니다.

BindExceptionWithViewName이 생성자를 통해 DataSource를 주입받도록 수정하고, 해당 객체를 통해 메시지를 출력할 수 있도록 getMessage 메소드를 추가하겠습니다.

/kro/rubisco/config/BindExceptionWithViewName.java

```
package kro.rubisco.config;

import java.util.Locale;

import org.springframework.context.MessageSource;
import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;

public class BindExceptionWithViewName extends BindException {
```

```
    private final String viewName;
    private final MessageSource messageSource;
    private final Locale locale;

    public BindExceptionWithViewName(
        BindingResult bindingResult,
        String viewName,
        MessageSource messageSource,
        Locale locale
    ) {
        super(bindingResult);
        this.viewName = viewName;
        this.messageSource = messageSource;
        this.locale = locale;
    }

    public String getViewName() {
        return viewName;
    }

    public String getMessage(String field) {
        return messageSource.getMessage(getFieldError(field), locale);
    }
}
```

TestController도 다음과 같이 `FieldError` 의 생성자에 인수를 추가하고, `BindExceptionWithViewName` 에 DataSource와 Locale을 주입해주도록 합시다.

/kro/rubisco/controller/TestController.java

```java
package kro.rubisco.controller;

import java.util.Locale;

import org.springframework.context.MessageSource;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import kro.rubisco.config.BindExceptionWithViewName;
import kro.rubisco.dto.ProductDTO;
import lombok.RequiredArgsConstructor;

@Controller
@RequiredArgsConstructor
@RequestMapping("/test")
public class TestController {

    private final MessageSource messageSource;

    @GetMapping()
    public void getTestView() {}

    @PostMapping()
    public String getTestView(ProductDTO product, BindingResult bindingResult, Locale locale) throws BindException {
```

```
if(product.getCount() == null || product.getCount() < 10 || product.getCount() > 100){
    bindingResult.addError(
        new FieldError(
            "product", "count", product.getCount(), false, new String[]{"range.product.count"},
            new Object[] {10L, 100L}, null
        )
    );
}

if(product.getPrice() == null || product.getPrice() < 100 || product.getPrice() > 10_000_000){
    bindingResult.addError(
        new FieldError(
            "product", "price", product.getPrice(), false, new String[]{"range.product.price"},
            new Object[] {100L, 10_000_000L}, null
        )
    );
}

if(bindingResult.hasErrors()) {
    throw new BindExceptionWithViewName(bindingResult, "/test", messageSource, locale);
}

return "redirect:/test";
    }
}
```

`FieldError` 의 생성자 매개변수가 늘어났습니다. 1번째, 2번째 매개변수는 `객체이름` , `필드이름` 이고, 3번째 매개변수는 클라이언트가 요청한 `필드값` 입니다.

4번째 매개변수는 binding 오류인지 검증 오류인지 판단하기 위한 인수인데, 검증 오류이므로 false를 입력합니다.

5번째 매개변수는 `에러코드` 를 String 배열로 입력합니다. DataSource에 입력된 key값과 매칭하여 일치하는 코드들 중에 첫번째 코드를 에러코드로 사용하게 됩니다.

6번째 매개변수는 DataSource로 전달될 인수 를 객체 배열로 생성하여 입력합니다. 인수가 필요없다면 null값을 주어도 됩니다. 범위 오류를 출력하는 경우와 같이 문구가 반복되고 값만 변한다면 동일한 코드에 인수만 전달하여 범용적으로 사용할 수 있습니다.

7번째 매개변수는 오류메시지의 기본값 으로, 일치하는 코드가 없다면 해당 값을 오류 메시지로 사용합니다. 위에 코드에서는 null값을 주었습니다.

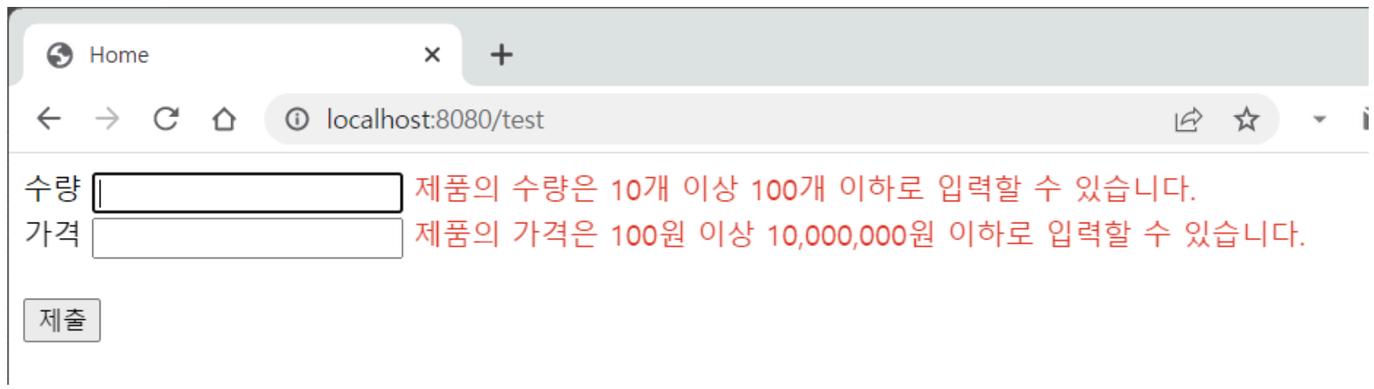
이제 뷰 템플릿을 수정합니다.

/src/main/webapp/WEB-INF/views/test.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"  %>
<html>
<head>
    <title>Home</title>
</head>
<body>
<form  method="POST"  action="/test">
    <c:forTokens var="field" items="count,price" delims=",">
        <label>
            <c:choose>
                <c:when test="${field  eq  'count'}">수량</c:when>
                <c:when test="${field  eq  'price'}">가격</c:when>
            </c:choose>
            <c:choose>
                <c:when test="${!errors.hasFieldErrors(field)}">
                    <c:set var="value" value="${errors.getFieldValue(field)}"  />
                </c:when>
                <c:otherwise><c:remove var="value"  /></c:otherwise>
            </c:choose>
            <input  type="text"  name="${field}"  value="${value}"  />
```

```
                <c:if test="${errors.hasFieldErrors(field)}">
                    <font color="red">${errors.getMessage(field)}</font>
                </c:if>
            </label>
            <br>
        </c:forTokens>
        <br>
        <input type="submit" value="제출" />
</form>
<c:if test="${errors.hasErrors()}">
<script>
[...document.forms[0].querySelectorAll("[name]")].filter(e=>!e.value)[0].focus();
</script>
</c:if>
</body>
</html>
```

메시지를 출력하는 부분의 코드를 errors 객체의 getMessage 메소드를 통해 출력하도록 변경했습니다.



rejectValue 메소드 사용

FieldError를 직접 생성할 수도 있지만 BindingResult에서는 FieldError를 간단하게 생성할 수 있는 `rejectValue` 메소드가 있습니다. 아래와 같이 TestController를 수정해보세요.

/kro/rubisco/controller/TestController.java

```java
package kro.rubisco.controller;

import java.util.Locale;

import org.springframework.context.MessageSource;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import kro.rubisco.config.BindExceptionWithViewName;
import kro.rubisco.dto.ProductDTO;
import lombok.RequiredArgsConstructor;

@Controller
@RequiredArgsConstructor
@RequestMapping("/test")
public class TestController {

    private final MessageSource messageSource;

    @GetMapping()
    public void getTestView() {}

    @PostMapping()
    public String getTestView(
        @ModelAttribute("product") ProductDTO product,
```

```
        BindingResult bindingResult,
        Locale locale
    ) throws BindException {

        if(product.getCount() == null || product.getCount() < 10 || product.getCount() > 100){
            bindingResult.rejectValue("count", "range", new Object[] {10L, 100L}, null);
        }

        if(product.getPrice() == null || product.getPrice() < 100 || product.getPrice() > 10_000_000){
            bindingResult.rejectValue("price", "range", new Object[] {100L, 10_000_000L}, null);
        }

        if(bindingResult.hasErrors()) {
            throw new BindExceptionWithViewName(bindingResult, "/test", messageSource, locale);
        }

        return "redirect:/test";
    }
}
```

FieldError를 직접 생성하는 것이 아니라 rejectValue 메소드를 호출하여 FieldError를 생성했습니다. `BindingResult` 는 검증할 객체 바로 다음에 주입되는 성질을 이용한 것입니다.

첫번째 변수는 `필드명` , 두번째 변수는 `에러코드` 입니다.

에러코드는 직접 FieldError를 생성할 때랑 차이가 있는데 MessageSource에 입력되는 키값은 `{에러코드}.{오브젝트명}.{필드명}` 형식으로 작성하고, 매개변수에는 에러코드 부분만 입력하면 됩니다. 예를 들어 MessageSource에 입력된 메시지 키값이 `range.product.cost` 라면 `range` 만 입력합니다.

오브젝트명은 검증할 객체의 타입으로 자동 설정되는데, 오브젝트명을 변경하려면 `@ModelAttribute` 어노테이션을 붙여 키값을 설정해주어야 합니다. 위에 예시 코드에서는 검증 객체의 이름이 `productDTO` 이므로, `@ModelAttribute("product")` 어노테이션을 붙여 객체이름을 `product` 로 변경해 주었습니다. 물론 MessageSource에 키값을 `range.productDTO.cost` 라고 변경해도 됩니다.

이렇게 에러코드를 입력해두면 내부 로직에서는 `MessageCodesResolver` 객체를 통해 `"{에러코드}.{오브젝트명}.{필드명}", "{에러코드}.{필드명}", "{에러코드}.{필드타입}", "{에러코드}"` 라는 4개의 에러코드를 생성하며, 여기에서 MessageSource에 존재하는 코드 중 첫번째 코드를 에러코드로 사용합니다. 예를 들어 위에 예시코드의 경우 `{"range.product.cost", "range.cost", "range.java.lang.Long", "range"}` 순서로 에러코드가 생성되고, MessageSource에 존재하는 `range.product.cost` 가 에러코드로 적용됩니다.

세번째 변수는 DataSource로 전달될 `인수의 배열` 이고, 네번째 변수는 `오류메시지 기본값` 으로 null을 입력했습니다.

ObjectError 추가

FieldError 객체가 필드값의 오류를 나타낸다면 ObjectError는 객체의 오류를 나타냅니다. 예를 들어 총액이 100,000,000원 이하가 되도록 범위를 정하려면 FieldError 만으로는 오류를 나타낼 수 없고 ObjectError를 통해 객체 자체에 오류가 있음을 나타내야 합니다.

/kro/rubisco/controller/TestController.java

```java
package kro.rubisco.controller;

import java.util.Locale;

import org.springframework.context.MessageSource;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import kro.rubisco.config.BindExceptionWithViewName;
import kro.rubisco.dto.ProductDTO;
import lombok.RequiredArgsConstructor;

@Controller
```

```java
@RequiredArgsConstructor
@RequestMapping("/test")
public class TestController {

    private final MessageSource messageSource;

    @GetMapping()
    public void getTestView() {}

    @PostMapping()
    public String getTestView(
        @ModelAttribute("product") ProductDTO product,
        BindingResult bindingResult,
        Locale locale
    ) throws BindException {

if(product.getCount() == null || product.getCount() < 10 || product.getCount() > 100){
    bindingResult.rejectValue("count", "range", new Object[] {10L, 100L}, null);
}

if(product.getPrice() == null || product.getPrice() < 100 || product.getPrice() > 10_000_000){
    bindingResult.rejectValue("price", "range", new Object[] {100L, 10_000_000L}, null);
}

if(product.getCount() != null && product.getPrice() != null) {

    Long total = product.getCount() * product.getPrice();

    if(total > 100_000_000) {
        bindingResult.addError(
            new ObjectError("product", new String[] {"totalPriceMax"}, new Object[] {100_000_000L, total}, null)
        );
```

```
            }
        }

        if(bindingResult.hasErrors()) {
            throw new BindExceptionWithViewName(bindingResult, "/test", messageSource, locale);
        }

        return "redirect:/test";
    }
}
```

`ObjectError` 의 생성자는 4개의 매개변수가 필요합니다. 첫번째 매개변수는 `객체이름` , 두번째 매개변수는 `에러코드` , 세번째 매개변수는 DataSource로 전달될 `인수의 배열` , 네번째 매개변수는 `오류메시지 기본값` 입니다.

`BindExceptionWithViewName` 클래스에 글로벌 오류에 대한 메시지를 받을 `getGlobalMessage` 메소드를 추가합니다.

/kro/rubisco/config/BindExceptionWithViewName.java

```
package kro.rubisco.config;

import java.util.Locale;

import org.springframework.context.MessageSource;
import org.springframework.validation.BindException;
import org.springframework.validation.BindingResult;

public class BindExceptionWithViewName extends BindException {

    private final String viewName;
```

```
    private final MessageSource messageSource;
    private final Locale locale;

    public BindExceptionWithViewName(
        BindingResult bindingResult,
        String viewName,
        MessageSource messageSource,
        Locale locale
    ) {
        super(bindingResult);
        this.viewName = viewName;
        this.messageSource = messageSource;
        this.locale = locale;
    }

    public String getViewName() {
        return viewName;
    }

    public String getMessage(String field) {
        return messageSource.getMessage(getFieldError(field), locale);
    }

    public String getGlobalMessage() {
        return messageSource.getMessage(getGlobalError(), locale);
    }
}
```

메시지도 다음과 같이 추가해줍니다.

/src/main/webapp/WEB-INF/message/error.properties

```
range.product.count = 제품의 수량은 {0}개 이상 {1}개 이하로 입력할 수 있습니다.
range.product.price = 제품의 가격은 {0}원 이상 {1}원 이하로 입력할 수 있습니다.
totalPriceMax = 최대 예산은 {0}원 입니다. (현재 총액: {1}원)
```

뷰 템플릿을 다음과 같이 수정합니다.

/src/main/webapp/WEB-INF/views/test.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"  %>
<html>
<head>
    <title>Home</title>
</head>
<body>
<form  method="POST"  action="/test">
    <c:forTokens var="field" items="count,price" delims=",">
        <label>
            <c:choose>
                <c:when test="${field  eq  'count'}">수량</c:when>
                <c:when test="${field  eq  'price'}">가격</c:when>
            </c:choose>
            <c:choose>
                <c:when test="${!errors.hasFieldErrors(field)}">
                    <c:set var="value" value="${errors.getFieldValue(field)}"  />
                </c:when>
                <c:otherwise><c:remove var="value"  /></c:otherwise>
            </c:choose>
            <input  type="text"  name="${field}"  value="${value}"  />
```

```
                <c:if test="${errors.hasFieldErrors(field)}">
                    <font color="red">${errors.getMessage(field)}</font>
                </c:if>
            </label>
            <br>
        </c:forTokens>
        <c:if test="${!error.hasFieldErrors() and errors.hasGlobalErrors()}">
            <p><font color="red">${errors.globalMessage}</font></p>
        </c:if>
        <br>
        <input type="submit" value="제출" />
</form>
<c:if test="${errors.hasErrors()}">
<script>
[...document.forms[0].querySelectorAll("[name]")].filter(e=>!e.value)[0].focus();
</script>
</c:if>
</body>
</html>
```

FieldError와 마찬가지로 BindingResult의 `reject` 메소드를 통해서 간편하게 생성할 수 있습니다.

/kro/rubisco/controller/TestController.java

```
package kro.rubisco.controller;

import java.util.Locale;

import org.springframework.context.MessageSource;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindException;
```

```java
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import kro.rubisco.config.BindExceptionWithViewName;
import kro.rubisco.dto.ProductDTO;
import lombok.RequiredArgsConstructor;

@Controller
@RequiredArgsConstructor
@RequestMapping("/test")
public class TestController {

    private final MessageSource messageSource;

    @GetMapping()
    public void getTestView() {}

    @PostMapping()
    public String getTestView(
        @ModelAttribute("product") ProductDTO product,
        BindingResult bindingResult,
        Locale locale
    ) throws BindException {

        if(product.getCount() == null || product.getCount() < 10 || product.getCount() > 100){
            bindingResult.rejectValue("count", "range", new Object[] {10L, 100L}, null);
        }

        if(product.getPrice() == null || product.getPrice() < 100 || product.getPrice() > 10_000_000){
```

```
        bindingResult.rejectValue("price", "range", new Object[] {100L, 10_000_000L}, null);
    }

    if(product.getCount() != null && product.getPrice() != null) {

        Long total = product.getCount() * product.getPrice();

        if(total > 100_000_000) {
            bindingResult.reject("totalPriceMax", new Object[] {100_000_000L, total}, null);
        }
    }

    if(bindingResult.hasErrors()) {
        throw new BindExceptionWithViewName(bindingResult, "/test", messageSource, locale);
    }

        return "redirect:/test";
    }
}
```

reject 메소드의 1번째 매개변수는 에러코드 , 두번째 매개변수는 DataSource로 전달될 인수의 배열 , 세번째 매개변수는 오류메시지 기본값 입니다.