Spring-Hibernate Training Program
Session 5

BoTree Consulting

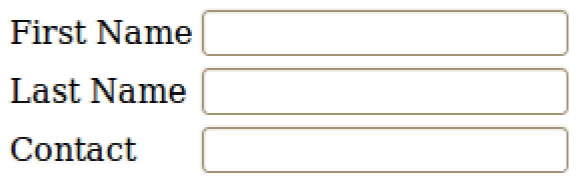
Data Binding and Validation

- **Validation** – Validating beans
- **Data Binding** – Binding user input to bean

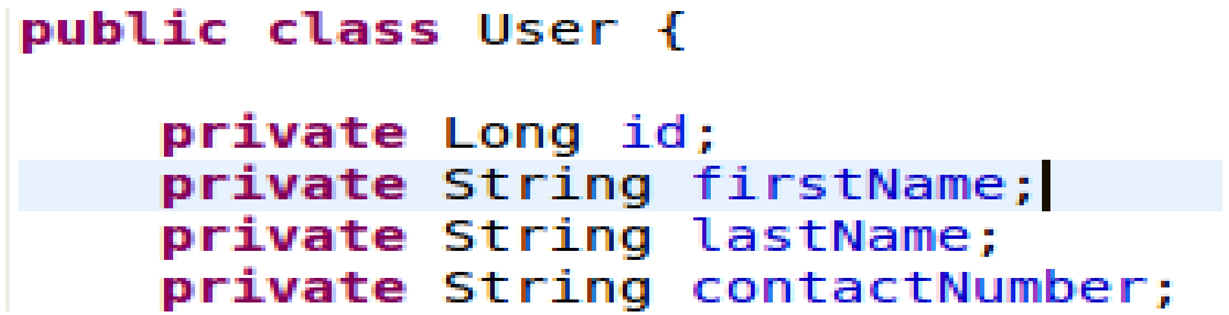
Data Binding

- Dynamically bind user input to domain model
- *Strings* to arbitrary *Objects*
- *org.springframework.validation.DataBinder*
- *org.springframework.web.bind.ServletRequestDataBinder*
- Customize binding process
  - PropertyEditors
  - Type Conversion

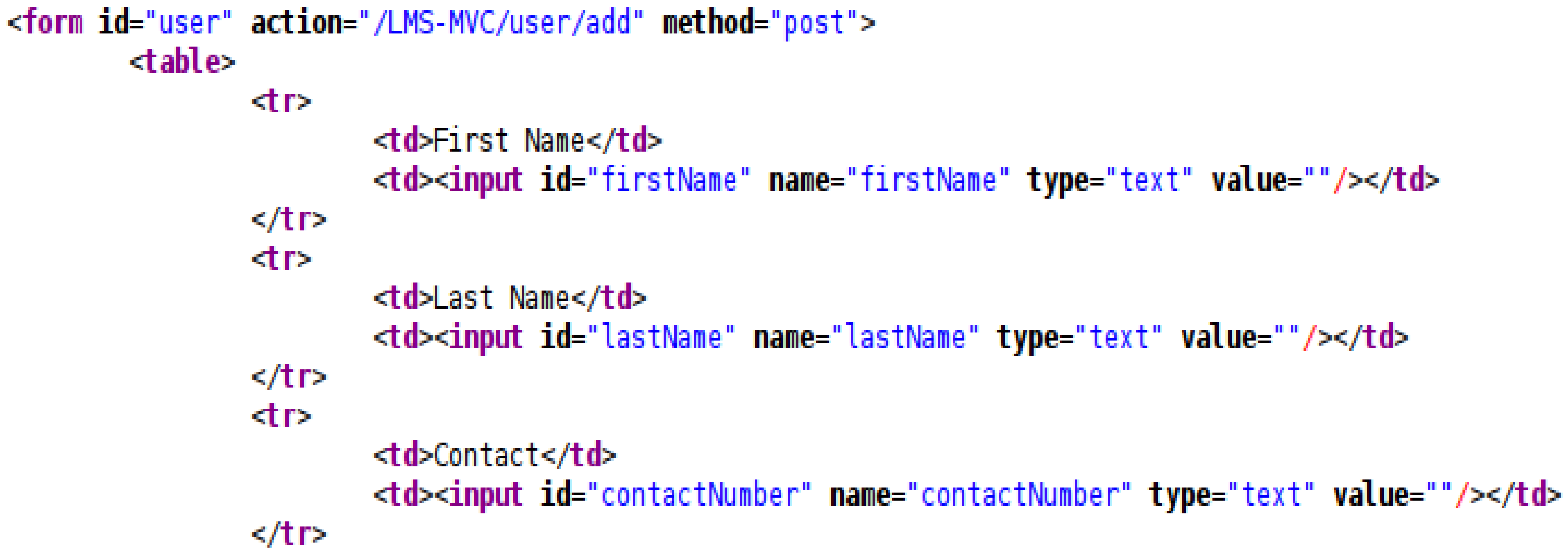
Data Binding



First Name	
Last Name	
Contact	



```java
public class User {

        private Long id;
        private String firstName;
        private String lastName;
        private String contactNumber;
```



```html
<form id="user" action="/LMS-MVC/user/add" method="post">
        <table>
                <tr>
                        <td>First Name</td>
                        <td><input id="firstName" name="firstName" type="text" value=""/></td>
                </tr>
                <tr>
                        <td>Last Name</td>
                        <td><input id="lastName" name="lastName" type="text" value=""/></td>
                </tr>
                <tr>
                        <td>Contact</td>
                        <td><input id="contactNumber" name="contactNumber" type="text" value=""/></td>
                </tr>
```

Data Binding – Built in PropertyEditors

- ByteArrayPropertyEditor
- CharacterEditor
- CustomBooleanEditor
- CustomCollectionEditor - Set, SortedSet and List
- CustomDateEditor*
- CustomNumberEditor - Number subclass like Integer, Long, Float, Double
- More....

Data Binding Customization

Step 1 : Create custom property editor

- PropertyEditorSupport

Step 2 : Configure

- CustomEditorConfigurer
- PropertyEditorRegistrar
- @InitBinder
- WebBindingInitializer

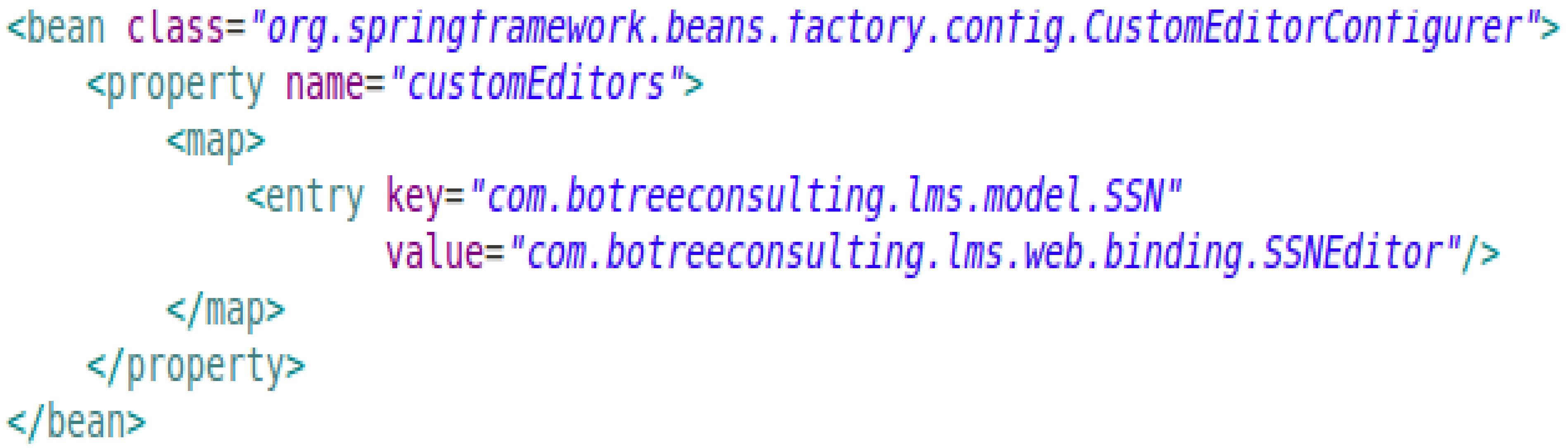
Step 3 : Use

Step 1 : Custom Property Editor

```java
public class SSNEditor extends PropertyEditorSupport{

    public void setAsText(String inputText) {
        setValue(new SSN(inputText.toUpperCase()));
    }
}
```

Step 2 : Configure

Option 1 : CustomEditorConfigurer



```xml
<bean class="org.springframework.beans.factory.config.CustomEditorConfigurer">
    <property name="customEditors">
        <map>
            <entry key="com.botreeconsulting.lms.model.SSN"
                   value="com.botreeconsulting.lms.web.binding.SSNEditor"/>
        </map>
    </property>
</bean>
```

Step 2 : Configure

Option 2 : Using PropertyEditorRegistrars

```java
public final class CustomPropertyEditorRegistrar implements PropertyEditorRegistrar {

    public void registerCustomEditors(PropertyEditorRegistry registry) {

        // it is expected that new PropertyEditor instances are created
        registry.registerCustomEditor(SSNType.class, new SSNTypeEditor());

        // you could register as many custom property editors as are required here...
    }
}
```

```xml
<bean class="org.springframework.beans.factory.config.CustomEditorConfigurer">
    <property name="propertyEditorRegistrars">
        <list>
            <ref bean="customPropertyEditorRegistrar"/>
        </list>
    </property>
</bean>

<bean id="customPropertyEditorRegistrar"
      class="com.botreeconsulting.lms.web.binding.CustomPropertyEditorRegistrar"/>
```

Step 2 : Configure

Option 3 : Using @InitBinder

```java
@InitBinder
public void initBinder(WebDataBinder binder) {
    binder.registerCustomEditor(SSN.class, new SSNEditor());
}
```

Step 2 : Configure

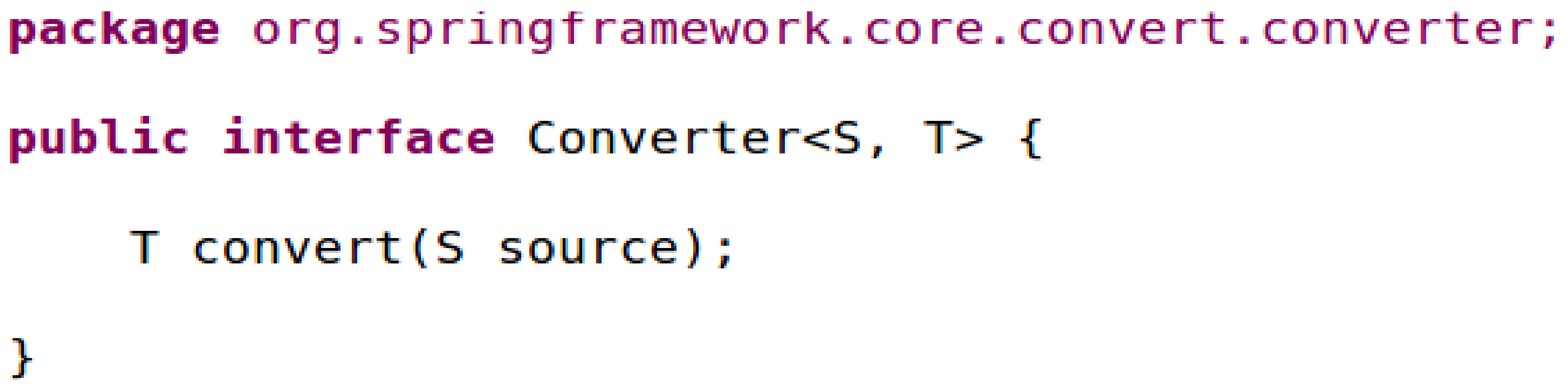
Option 4 : Using WebBindingInitializer

```
public class GlobalBindingInitializer implements WebBindingInitializer {

    public void initBinder(WebDataBinder binder, WebRequest request) {

        binder.registerCustomEditor(SSN.class, new SSNEditor());

    }

}
```

```
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
    <property name="webBindingInitializer">
        <bean class="com.botreeconsulting.lms.web.binding.GlobalBindingInitializer"/>
    </property>
</bean>
```

Type Conversion

- Alternative to PropertyEditor
- Available org.springframework.core.convert
- Implement



```java
package org.springframework.core.convert.converter;

public interface Converter<S, T> {

    T convert(S source);

}
```

Validation

- Spring's Validator
- JSR – 303 metadata

Validation – Using Validator

- Not coupled with web tier
- Implement org.springframework.validation.Validator
  - *public boolean supports(Class clazz)*
  - *public void validate(Object target, Errors error)*
- Configure a Validator in Spring MVC (3 ways)
  - *@Autowire*
  - *@InitBinder*
  - *<mvc:annotation-driven validator="globalValidator"/>*

Validation – Using Validator

```java
public class UserLoginValidator implements Validator {

    private static final int MINIMUM_PASSWORD_LENGTH = 6;

    public boolean supports(Class clazz) {
       return UserLogin.class.isAssignableFrom(clazz);
    }

    public void validate(Object target, Errors errors) {

       ValidationUtils.rejectIfEmptyOrWhitespace(errors, "userName", "field.required");

       ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password", "field.required");

       UserLogin login = (UserLogin) target;

       if (login.getPassword() != null
             && login.getPassword().trim().length() < MINIMUM_PASSWORD_LENGTH) {

          errors.rejectValue("password", "field.min.length",
                new Object[]{Integer.valueOf(MINIMUM_PASSWORD_LENGTH)},
                "The password must be at least [" + MINIMUM_PASSWORD_LENGTH + "] characters in length.");
       }
    }
 }
```

Validation – JSR 303

- Standardizing validation constrains
- Declarative rules
- Common pre-built constraints
- Annotate command object with JSR-303 annotations
- Detect
  - *<mvc:annotation-driven/>*

End of
Spring – Hibernate Training Program
Session 5