Validators and Editors

In the Spring framework

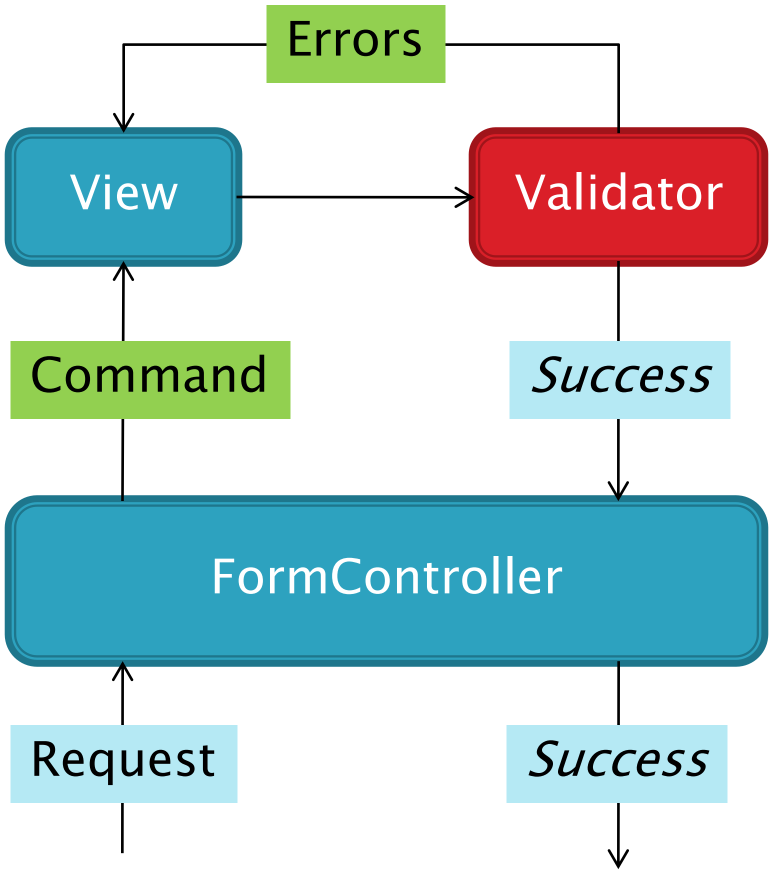
EHSDI
eBuzima

Why validators and editors?

- Controllers can get quite complex when they are responsible for...
  - Creating new domain objects
  - Editing existing domain objects
  - Validating values for create and edit
  - Converting string values to actual objects
- If you have a separate controllers/views for creating and editing, input processing code could be duplicated

Validators

- The sole purpose of a Validator is to validate the command object of a form
- Any errors it finds, it sends back to the view



Example

- We have a `Person` class and a form for editing a person's details
- We need to ensure that the name and age values are valid

```
public class Person {
   ...

   public String getName() {...}
   public int getAge() {...}
}
```

- So we create `PersonValidator`...

Example

```
public class PersonValidator implements Validator {

  public boolean supports(Class clazz) {
    return Person.class.equals(clazz);
  }


  public void validate(Object obj, Errors e) {
    ValidationUtils.rejectIfEmpty(e, "name", "name.empty");
    Person p = (Person) obj;
    if (p.getAge() < 0)
      e.rejectValue("age", "age.negative");
    else if (p.getAge() > 110)
      e.rejectValue("age", "age.too.old");
  }

}
```

This validator only validates `Person` objects

Reject `name` values that are empty

Rejects `age` values not between `0` and `110`

Message codes for language files

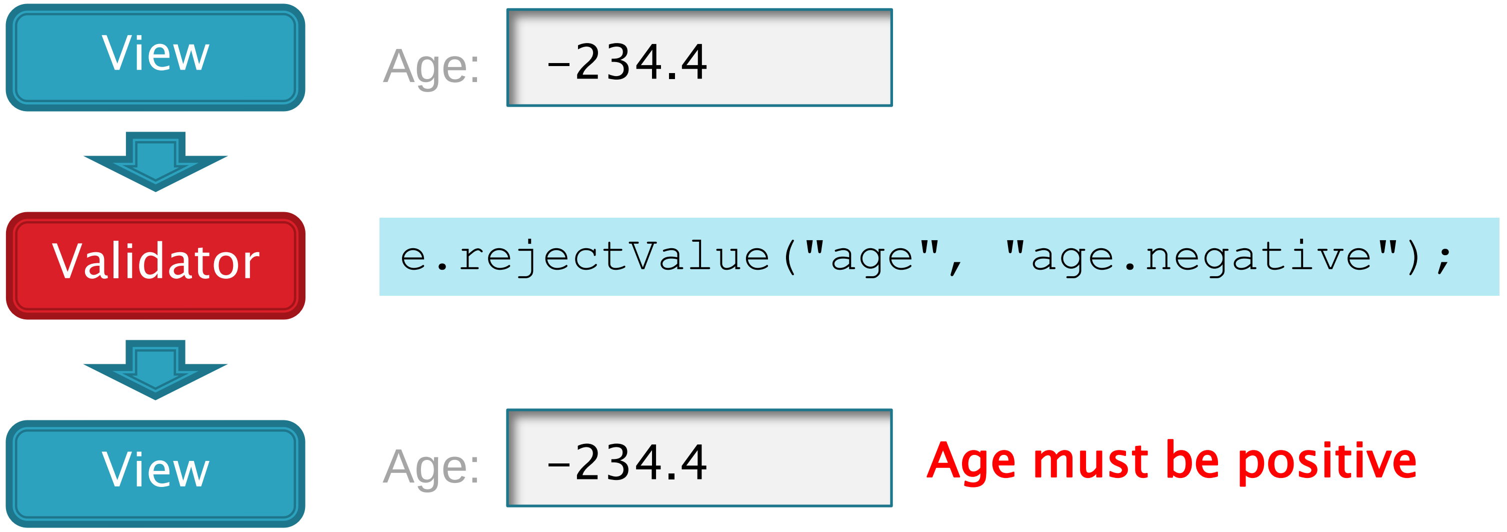
ValidationUtils

- This is a helper class to simplify your validators. Contains the following methods:
  - `rejectIfEmpty` – rejects a specific field if it is null or empty
  - `rejectIfEmptyOrWhitespace` – rejects a specific field if it is null, empty or whitespace
  - `invokeValidator` – invokes a specific validator on an object, e.g.
    - `Project` class has a `manager` property of class `Person`. From the `Project` validator we can invoke a `Person` validator on the `manager` rather than duplicate its functionality.

EHSDI
eBuzima

Validation errors

- Validators make it easy to associate validation errors with fields in the form, e.g.

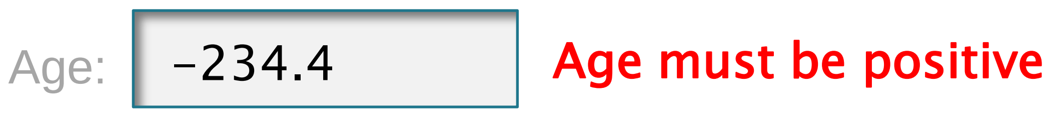


```
e.rejectValue("age", "age.negative");
```

Validation errors

- The errors tag will display the errors associated with that field, e.g.

```
<form:form><table>
...
<tr>
  <td>Age:</td>
  <td><form:input path="age" /></td>
  <td><form:errors path="age" /></td>
</tr>
...
</table></form:form>
```



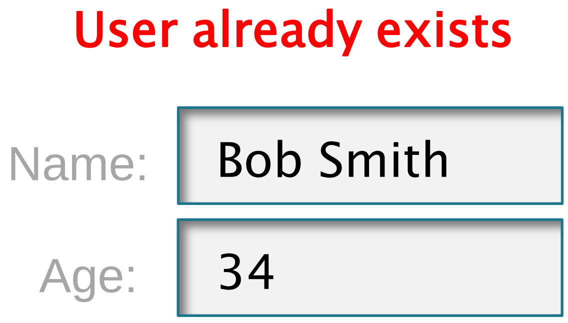
Age: -234.4 Age must be positive

Validation errors

- Some errors are not associated with a specific field (global errors)
- These can be displayed by omitting the `path` attribute, e.g.

```
e.reject("user.exists");
```

```
<form:form>
  <form:errors />
  <table>
  ...
  </table>
</form:form>
```



User already exists

Name:	Bob Smith
Age:	34

Controllers and validators

- Controllers beans are configured with validator beans in XML, e.g.

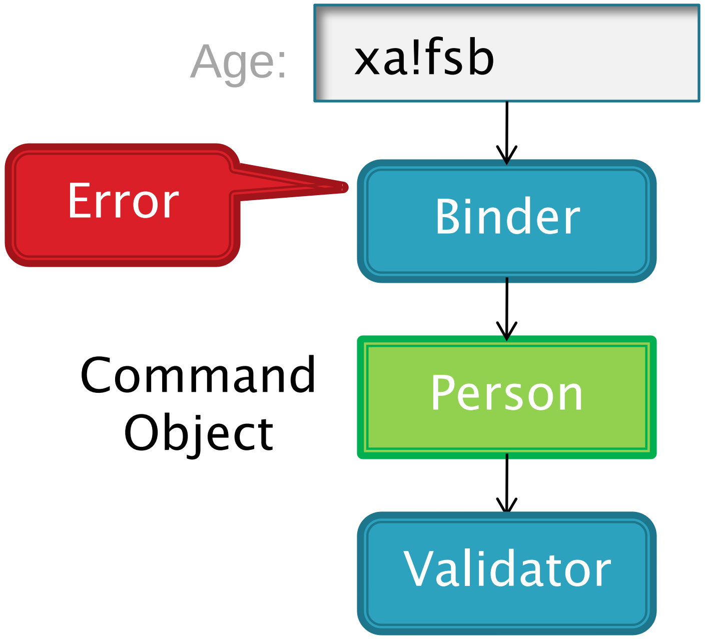
```
<bean id="personValidator" class="ehsdi.PersonValidator" />

<bean id="personForm" class="ehsdi.PersonFormController">
  <property name="commandName"><value>person</value></property>
  <property name="formView"><value>/personForm</value></property>
  <property name="successView"><value>person.list</value></property>

  <property name="validator">
    <ref bean="personValidator" />
  </property>

</bean>
```

Controller will now automatically validate the person when form is submitted

Binding errors

- What if submitted values cannot even be bound to the command object?
- Spring will create its own error message, which will be displayed by `<form:errors>`, e.g.

Failed to convert property value of type [java.lang.String] to required...



Binding errors

- This can be customized by defining `typeMismatch.`**`xxx`** in our message source, where **`xxx`** is the name of the property

```
typeMismatch.age=Age must be a number
```

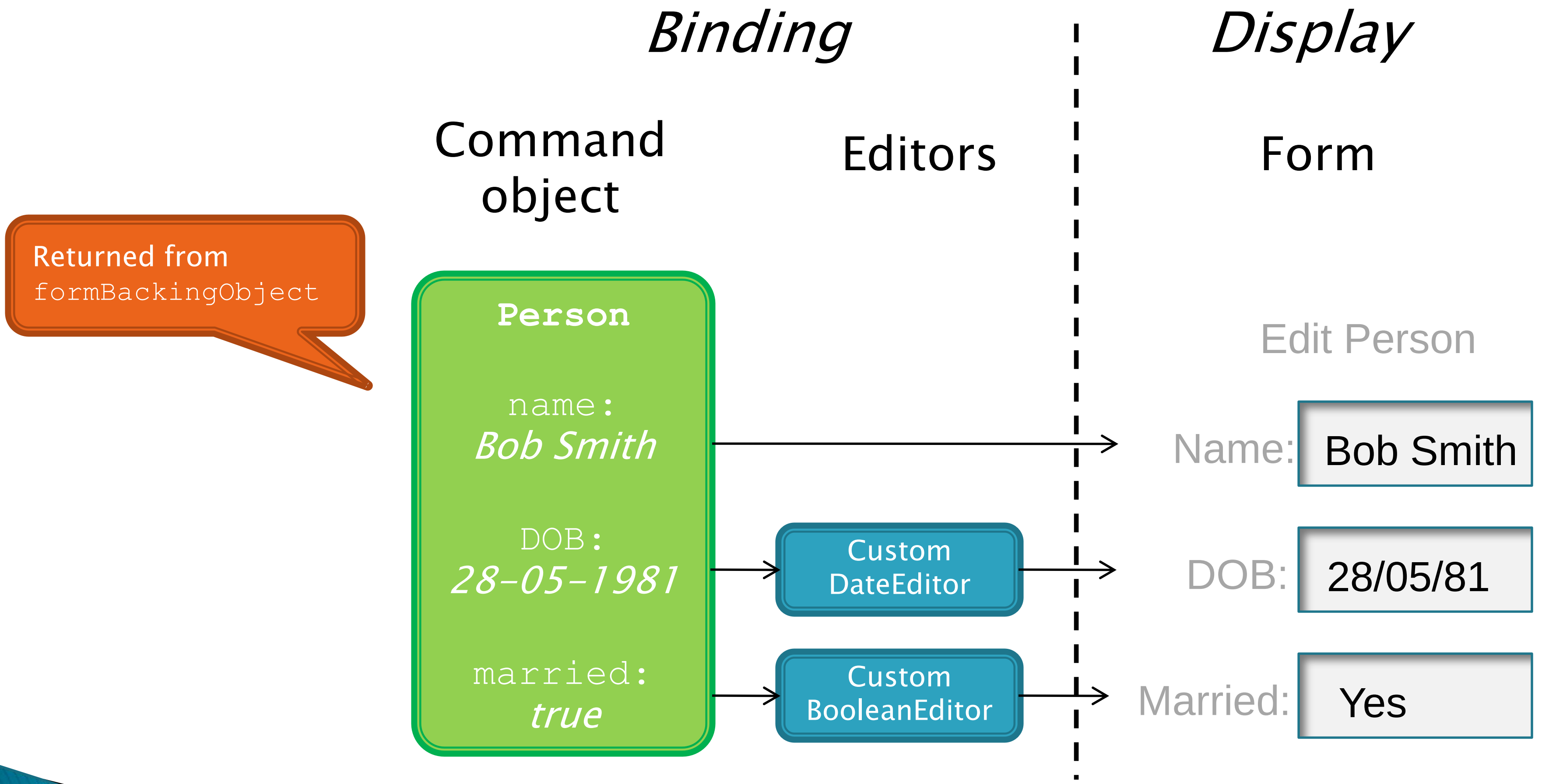
- We can also define a message to be used for all type mismatch errors with that type (class or primitive type), e.g.

```
typeMismatch.int={0} must be a number
typeMismatch.java.util.Date={0} must be a date
```

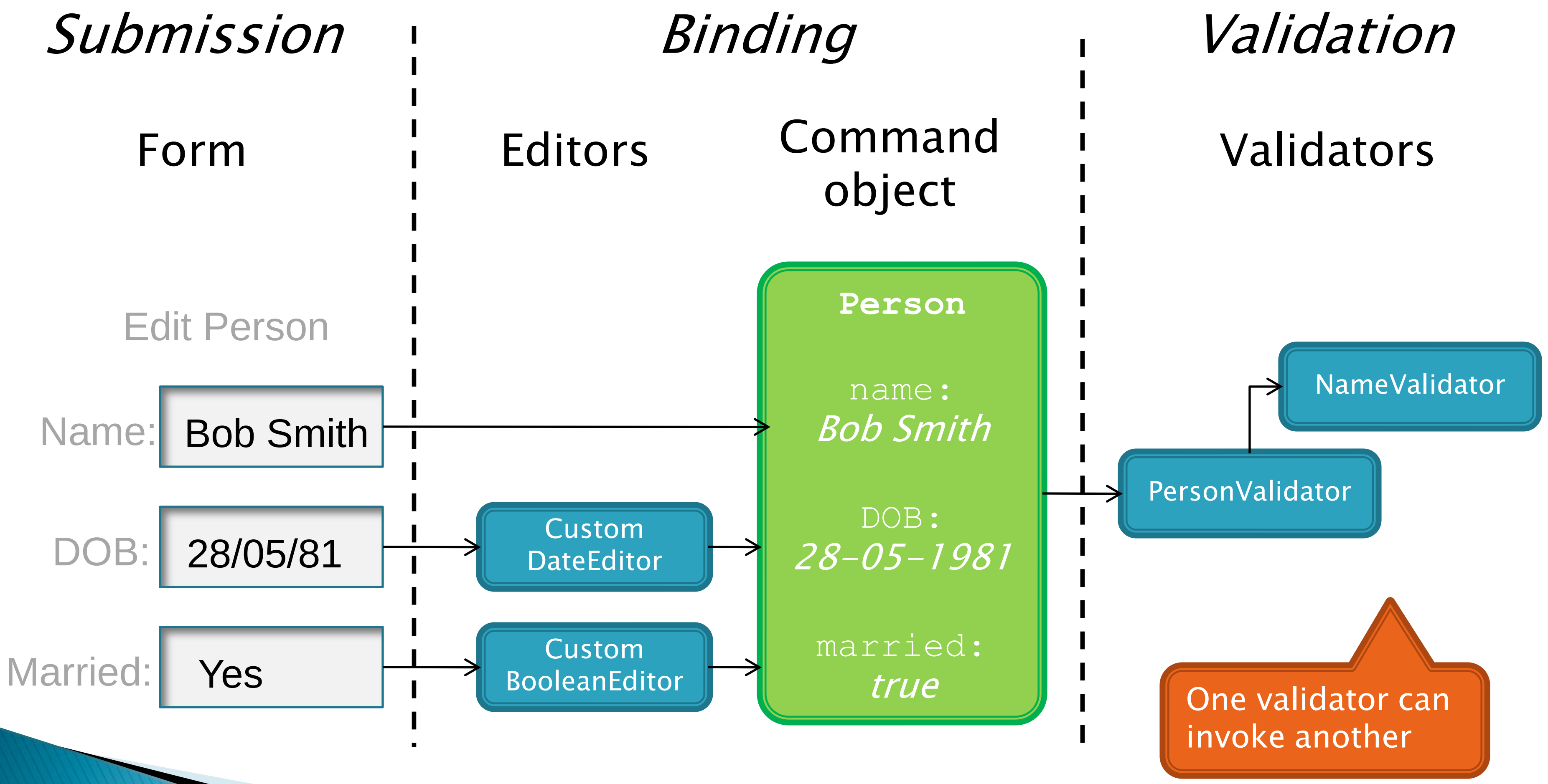
Editors

- An editor handles the conversion between a value/object and a string
- Spring uses lots of editors, e.g.
  - Converting request parameters to command object properties during binding
  - Binding string values in XML files to object properties
- Spring defines several editors which all implement `java.beans.PropertyEditor`

Editor and validator workflow

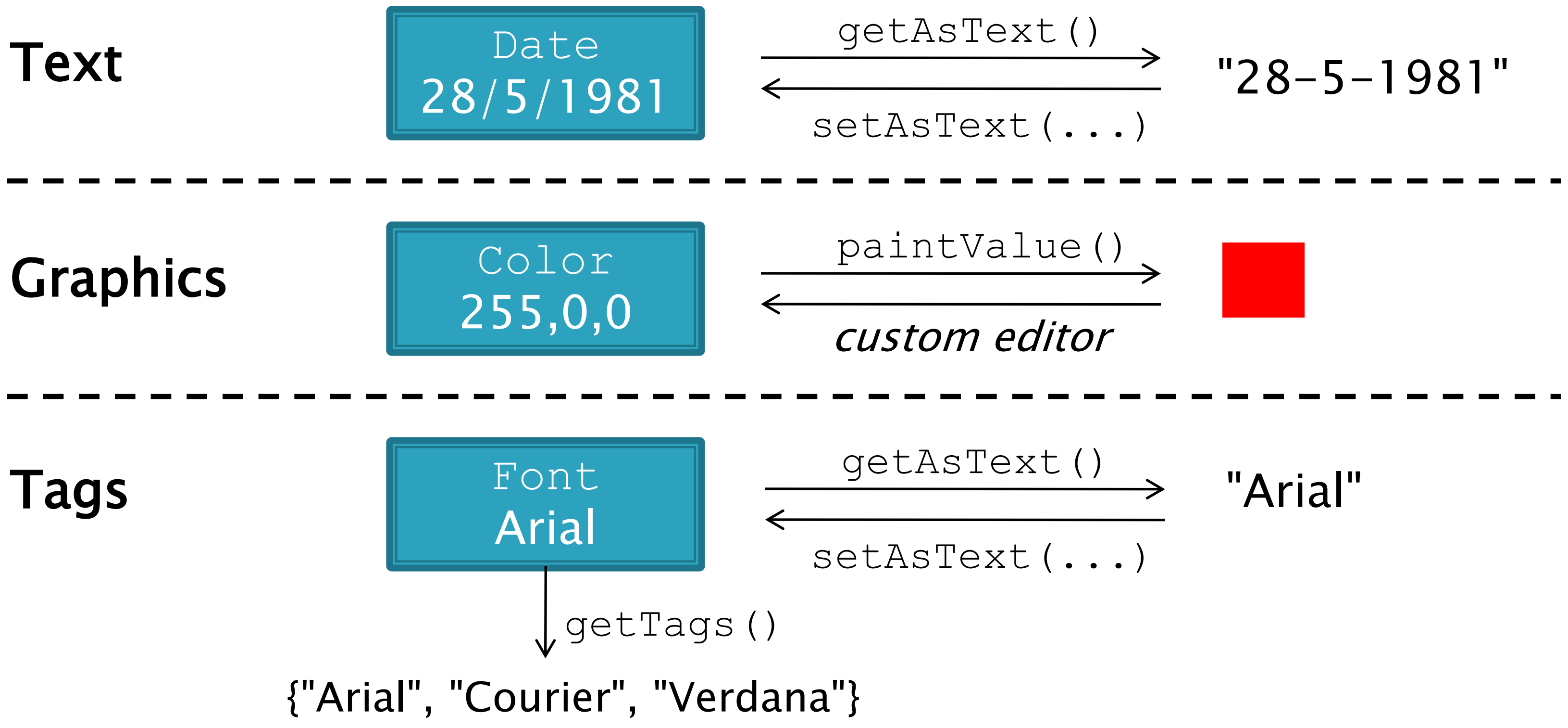


Editor and validator workflow



java.beans.PropertyEditor

- Used to allow GUI applications to edit beans
- Every editor must support one of three modes:



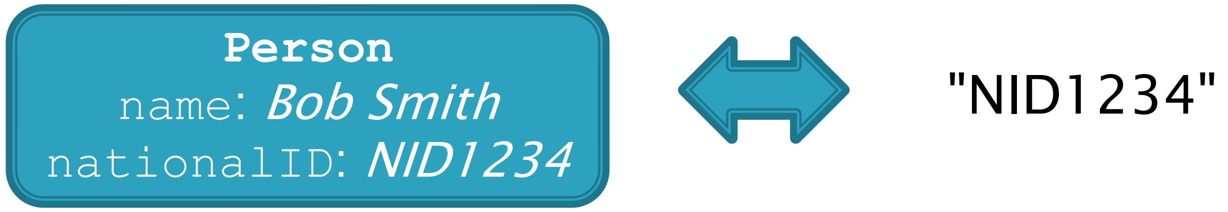
Spring's editors

- Spring provides several predefined editors, such as...
  - `ClassEditor` – converts between Java classes and strings (i.e. the class name)
  - `CustomDateEditor` – converts between `Date` objects and strings using a format string
  - `LocaleEditor` – converts between `Locale` objects and strings using the "*en_GB*", *"fr_RW"* format

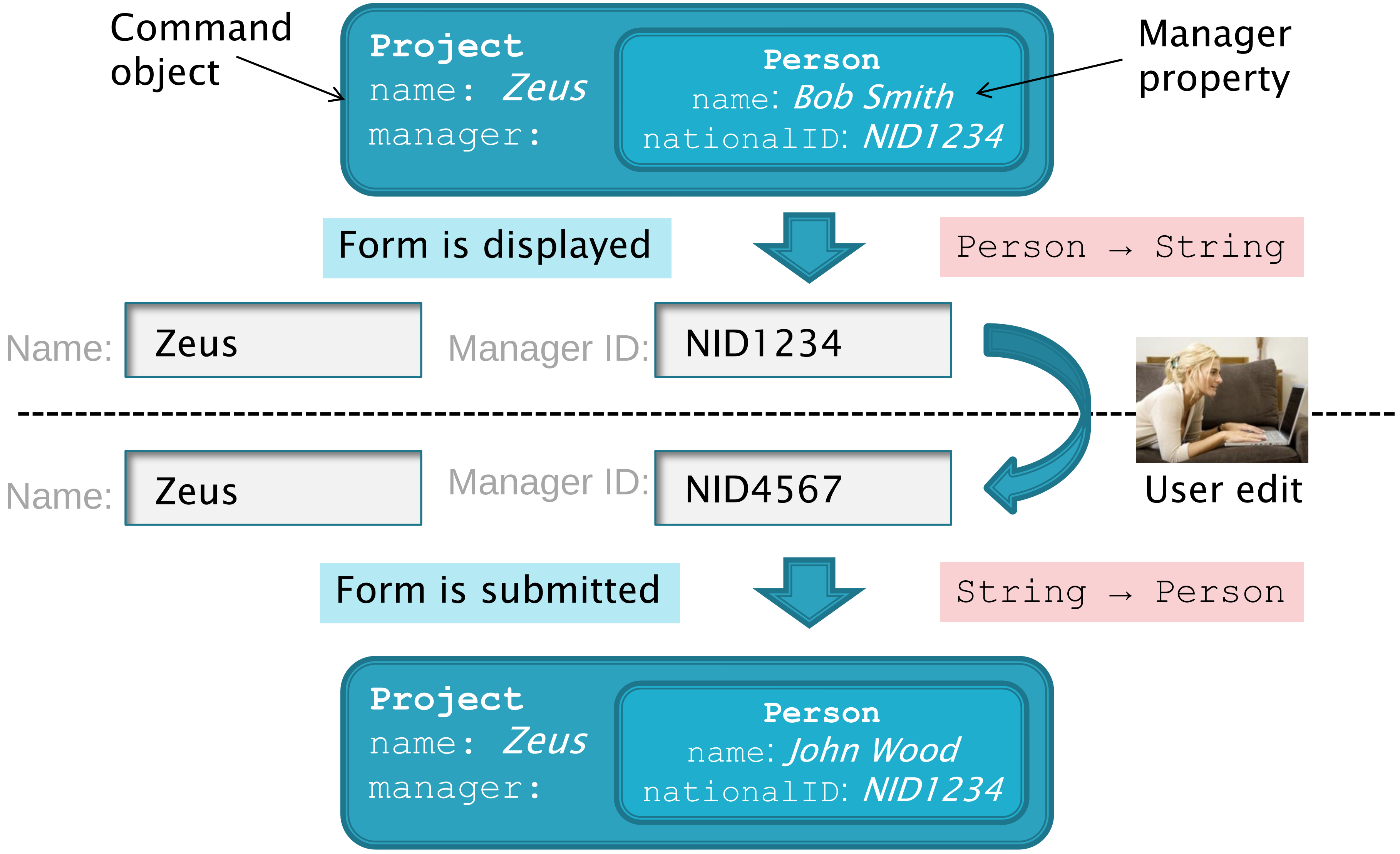
- But sometimes it is necessary to create our own editor...

Custom editor example

- Suppose our `Person` class has a `nationalID` property
- A field on a certain form is for specifying a `Person`, which should be inputted as that person's National ID value
- When form is redisplayed, the `Person` object should be displayed as their National ID



Custom editor example



Custom editor example

```
public class PersonEditor extends PropertyEditorSupport {

  public void setAsText(String text) {
    Person p = PersonService.getByNationalID(text);

    setValue(p);
  }


  public String getAsText() {
    Person p = (Person)getValue();

    return p.getNationalID();
  }

}
```

Sets object value, given string containing ID

Gets string value of ID given the object

EHSDI
eBuzima

Registering custom editors

- So that Spring knows to use our editor for a specific field type, we override `initBinder` and register our custom editor, e.g.

```
public class ChoosePersonController extends SimpleFormController {
  ...

  protected void initBinder(
    HttpServletRequest request, ServletRequestDataBinder binder
  ) throws Exception {
    super.initBinder(request, binder);

    binder.registerCustomEditor(Person.class, new PersonEditor());
  }
}
```

References

- Websites
  - http://static.springsource.org/spring/docs/2.0.x/reference/validation.html

EHSDI
eBuzima