Spring Framework Workshop
Practical Part

Task 1 – Validation Exercise

In this assignment we will create some validation criteria for the registration process of our application.

For our tutorial we will presume the fact that we want the provided fields not to be empty and our email to be valid.

In order to do this, we will have to follow the following steps:

1. Go to the 'user' class from the 'model' package
2. Add the 'NotBlank' tag followed by parentheses and an appropriate error message to the fields that should not be empty

```
@NotBlank(message = "firstName is mandatory")
private String firstName;
```

3. Add the 'Email' tag followed again by an appropriate error message to the field that stocks the email

```
@Email(message = "a valid email is mandatory")
```

4. Next, we must move to the 'controller' package and the 'UserRegestrationController' class
5. In the 'registerUserAccount' method we have to surround the 'userService.save' command in a 'try-catch' structure
6. We want to catch an 'ConstraintViolationException' error
7. Next, we will create a redirect string which we will use later
8. Create a 'for-each' structure where for each 'ConstraintViolation' we will concat it to the redirect string created earlier, follow the picture bellow for a more precise explanation:

```
try{
    userService.save(registrationDto);
} catch (ConstraintViolationException e){

    String redirect = "redirect:/registration?";
    for (ConstraintViolation<?> error:
            e.getConstraintViolations()) {
        redirect = redirect.concat(error.getPropertyPath() + "=true&");
    }
```

9. Return the 'redirect' string, this will make the browser to redirect the user to the same page only with the error parameter as 'GET' variables
10. Next, we will want to go the registration html template located under resources
11. Under each label for the registration form create a paragraph

12. In the paragraph tag we will use thymeleaf (an explanation on how thymeleaf works exceeds the scope of this workshop, please follow the structure bellow) to check if an error was triggered for that specific field

```
<p th:if="${param.email}">Invalid email</p>
```

Task 2 – Resource Interface Exercise

In order to understand the basics of the Spring's Resource interface, we will try to get a text file from the resources, get its text as String and paste it on the Employee List page.

1. Go to EmployerController.
2. Locate loadResourceWithClassPathResource() method.
3. Make sure to return the resource (a text file in the resources file called "resource.txt") using ClassPathResource
4. Locate findPaginated() method.
5. Add the attribute "happyResources" to the model and make use of inputStreamToString(), loadResourceWithClassPathResource(), and getInputStream() to get the text on the page.

In the end, you should have something like this:

```
        /*
        * TODO
        */
        model.addAttribute("happyResource", inputStreamToString(loadResourceWithClassPathResource().getInputStream()));

        model.addAttribute("listEmployees", listEmployees);
        return "index";
    }

    public Resource loadResourceWithClassPathResource() {
        /*
        * TODO
        */
        return new ClassPathResource("resource.txt");
    }
```

Task 3 – Event Exercise

In this assignment we will create an event, publish that event and a listener for that custom event.

For our tutorial we will presume the fact that we want to log to console every time someone deletes an employee.

In order to do this, we will have to follow the following steps:

1. Create a new java class 'CustomEvent'
2. This new class should extend the ApplicationEvent class
3. Create a new field in the class in which we will store the id of the deleted employee
4. Create a constructor for the class with the following fields: the source of the event which will be of type 'Object' and the id of the deleted employee

5. Create a getter for the employee id; You should end up with something that looks like this:

```
public class CustomEvent extends ApplicationEvent {
    private String id;

    public CustomEvent(Object source, String id) {
        super(source);
        this.id = id;
    }

    public String getId() { return id; }
}
```

6. Next we will move to the 'EmployeeController' class where we will create a new private field of type 'ApplicationEventPublisher'
7. Create a constructor for the controller which takes in an ApplicationEventPublisher and assigns it to the field created above
8. Add the 'Autowired' tag to the constructor, you should have something like this:

```
private ApplicationEventPublisher applicationEventPublisher;
@Autowired
public EmployeeController(ApplicationEventPublisher applicationEventPublisher){
    this.applicationEventPublisher = applicationEventPublisher;
}
```

9. Next we will go to the 'deleteEmployee' method where we will publish the event
10. Create a new object of type customEvent where we will parse it this source and the id of the deleted employee
11. Invoke the 'publishEvent' method of the applicationEventPublisher object created earlier where we will give it the customEvent just created, this is the result:

```
@GetMapping("/deleteEmployee/{id}")
public String deleteEmployee(@PathVariable (value = "id") long id) {

    CustomEvent customEvent = new CustomEvent( source: this, id);
    applicationEventPublisher.publishEvent(customEvent);

    this.employeeService.deleteEmployeeById(id);
    return "redirect:/";
}
```

12. Next we will create a listener for the event just published
13. Create a new method which takes in an object of class 'CustomEvent'
14. Add the 'EventListener' tag to the method above

15. Print out an appropriate message to log the fact that an employee was just deleted

```java
@EventListener
public void printDeletedEmployees(CustomEvent customEvent){
    System.out.println("Warning the following employee was deleted: " + customEvent.getId());
}
```

Task 4 – Type Conversion Exercise

This assignment is based on the 'Event' task so if you have not completed it please go and do that before this one.

In this assignment we will create a custom converter which will transform an object of type 'Employee' to a string.

For our tutorial we will presume the fact that instead of printing the id of an employee when it is deleted, we want to print his first name, last name and lastly his email. In order to accomplish this, we will create a custom converter which will take in an object of type Employee and convert it to our desired form as a string.

In order to achieve this, we will have to follow the following steps:

1. Create a new java class called 'EmployeeToStringConverter'
2. This new class will implement 'Converter<Employee, String> from the library 'org.springframework.core.convert.converter.Converter'
3. We will want to override the convert method
4. This will return, in our case, a string since we are converting an employee to string and will have as a parameter an 'Employee'
5. Here we will get the first name, last name and email of an employee and concat it all into a string
6. Return the string and you should reach the following result:

```java
public class EmployeeToStringConverter implements Converter<Employee, String> {
    @Override
    public String convert(Employee employee) {
        String stringEmployee = employee.getFirstName() + " " + employee.getLastName() + " " + employee.getEmail();
        return stringEmployee;
    }
}
```

7. In order to use our new converter, we will want to make Spring boot aware of it and register it to the FormatterRegistry
8. Create a new class called 'WebConfig' which implements 'WebMvcConfigurer' and give it the 'Configuration' tag
9. Override the addFormatters method which will take in as a parameter a 'FormatterRegistry' object

10. Use the ‘addConverter’ method of the registry object and give it a new ‘EmployeeToStringConverter’. You should end up with something like this:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addFormatters(FormatterRegistry registry){
        registry.addConverter(new EmployeeToStringConverter());
    }
}
```

11. The new converter is done now we just have to use it, go to the ‘EmployeeController’ class
12. Give the class a new object of type ‘ConversionService’ this should be private and given the ‘Autowired’ tag

```
@Autowired
private ConversionService conversionService;
```

13. Go to the ‘printDeletedEmployees’ method (or your version of the method) and use the ‘getEmployeeById’ method of the ‘employeeService’ object in order to get the actual Employee
14. Create a new string in which we will store the converted employee
15. Use the ‘convert’ method of the ‘conversionService’ object and give it the employee instance as the first parameter and ‘String.class’ as the second.
16. Instead of printing the id of the employee use the newly created string

```
@EventListener
public void printDeletedEmployees(CustomEvent customEvent){
    Employee employee = employeeService.getEmployeeById(customEvent.getId());
    String employeeString = conversionService.convert(employee, String.class);
    System.out.println("Warning the following employee was deleted: " + employeeString);
}
```