

CS544
Enterprise Architecture Final
Exam 2 May 2017

Name _____

Student ID __108779_____

NOTE: This material is private and confidential. It is the property of MUM and is not to be disseminated.

1. [15 points] **CIRCLE** which of the following are TRUE/FALSE concerning EAI :

T ☒ F Spring Enterprise Integration is essentially another name for Spring Remoting
[JMS,AMQP,HTTP etc.]

EXPLAIN:

Spring features integration classes for remoting support using various technologies. The remoting support eases the development of remote-enabled services, implemented by your usual (Spring) POJOs. Spring Enterprise Integration high-level abstraction over spring remoting.

T ☒ F Spring Integration is based on lightweight messaging which is based on JMS.

EXPLAIN:

Spring integration is yes, it is a lightweight messaging but is based on some declarative adapters like JMS, AMPQ and others not only JMS.

☒ T F A Content Router basically determines routing based on IF-THEN-ELSE logic.

EXPLAIN:

What Content router does is that it determines if the message header and payload satisfies a condition and based on the satisfaction of the condition it will pass it to respective channel. The If Else condition might not be always one it will have continuity like IF-THEN-ELSE-IF-THEN-ELSE

☒ T F A Message Channel is a core component of Spring Integration

EXPLAIN:

Spring integration has three main/core components Message, Message channel and Message Endpoint

☒ T F A basic capability of ESB is transport conversion.

EXPLAIN:

ESB:- Enterprise service bus helps in converting message from one type to another while transporting it the other outlet or destination.

2. [10 points] Consider the following AOP Aspect:

Create an Advice that executes before an *annotated* application joinpoint that has a pointcut that would identify service methods that have a `OrderItem` object as a parameter.

Note you need to create an additional pointcut AND the Advice.

The Advice method should calculate and print out the cost of the inventory using quantity and price fields from `OrderItem`.

Also show a complete implementation of an example join point.

Two Pointcuts are provided for your use.

```
@Aspect
```

```
@Component
```

```
AccountingAspect.java
```

```
@Pointcut("execution(* edu.mum.service..*(..))")
public void accountingMethod() {}
```

```
@Pointcut("@annotation(edu.mum.validation.Accounting)")
public void accounting() {}
```

Answer

```
@Pointcut("args(orderItem)")
```

```
public void testOrderItem(OrderItem orderItem) {}
```

```
@Before("accountingMethod() && accounting() && testOrderItem(orderItem)")
```

```
public void adviceMethod(OrderItem orderItem) {
```

```
System.out.println();
```

```
System.out.println("Cost of inventory is: "+orderItem.getQuantity*orderItem.getPrice);
```

```
System.out.println("Order Number: " + orderItem.getOrderNumber());
```

```
}
```

```
edu.mum.service.impl.OrderServiceImpl
```

```
@Accounting
```

```
public Order update(Order order) {
```

```
    return orderDao.update(order);
```

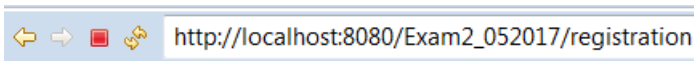
```
}
```

3. [20 points] Spring has 3 SPEL expressions that can be used to provide annotation-based access control on Service methods. We commonly used `@PreAuthorize` as the annotation.
- a. Explain the use & give examples of the 3 expressions using the `@PreAuthorize`.
 - b. Explain how Spring facilitates RBAC [include the use of 1 of the 3 expressions] *by example*
 - c. Explain how Spring facilitates ABAC [include the use of 1 of the 3 expressions] *with example*
- a. `@PreAuthorize("hasRole('ROLE_ADMIN')")`: grants authorization to the specified method based on the users authority level, checks if `@PreAuthorize("hasAuthority('READ')")`: grants authorization for the user based on the users permission level. The permissions can be assigned to a group or not. `@PreAuthorize("hasPermission('#comment', 'update')")`: the authorization level is based on custom attributes that can be a constraint in the application.
- b. Role Based Access control: permissions are given to roles instead of users. Users have roles. It is better for a group of users. `hasAuthority('READ')`
- c. Attribute Based Access Control: Permissions are given based on user attributes, object attributes and environmental condition with requirements specified in access control policies.

4. [15 Points] This is a student Registration form. There is validation required before the student can be entered into the system successfully. If the student information is entered correctly then a JSP page success.jsp is displayed. Below you can see the error messages resulting from wrong input. Fill in ALL the content of the supplied resources. USE BEST PRACTICES...

NOTE: In the interest of time, you only need to annotate ONE of the phone fields [all 3 look about the same]. For “3 digits”, test for values between 100 & 999.

INITIAL SCREEN:



Registration Form

Student Id :

FirstName :

LastName :

Birthday :

Phone :

Gender :

NO INPUT & “SAVE CHANGES”

Registration Form

Student Id : Student ID must be grater that 14

FirstName : First Name field must have a value
Size of the First Name must be between 4 and 50

LastName : Last Name field must have a value

Birthday : Birthday is a required field

Phone : Area Code is Invalid.It must 3 digits. Prefix is Invalid. It must 3 digits. Number is Invalid. It must be 4 digits.

Gender :

WITH VALID INPUT & “SAVE CHANGES”

Registration Form

Student Id :	<input type="text" value="22"/>		
FirstName :	<input type="text" value="Will"/>		
LastName :	<input type="text" value="Pickett"/>		
Birthday :	<input type="text" value="12/12/1212"/>		
Phone :	<input type="text" value="222"/>	<input type="text" value="333"/>	<input type="text" value="4444"/>
Gender :	<input type="text" value="Female"/>	<input type="button" value="v"/>	
<input type="button" value="Save Changes"/>			

SUCCESSFUL “SAVE CHANGES”

Student Save successfully

Student Id: 22
Student FirstName: Will
Student LastName: Pickett
Student Phone: 222-333-4444

StudentController.java

```
@RequestMapping(value = "", Method = RequestMethod.Get)
public String showForm(@ModelAttribute("newStudent") Student newStudent){

    return "addStudent";

}

@RequestMapping(value = "", method = RequestMethod.POST)
public String processForm (@Valid @ModelAttribute("newStudent") Student
studentToBeAdded, BindingResult result){

    if(result.hasErrors)
        return "addStudent";

    studentService.saveFull(studentToBeAdded);
    return "redirect:/success/{studentToBeAdded.getId}";

}

@RequestMapping("/success/{id}")
public String success(@PathVariable("id") Model model) {
    model.addAttribute("newStudent", studentService.findOne(id));
    return "student";
}
```

Student.java

```
public class Student {  
  
    @Min(value = 14)  
    private int StudentId;  
  
    @NotEmpty  
    @Size(min=4, max=50, message = "{Size.validation}")  
    private String firstName = null;  
  
    @NotEmpty  
    private String lastName = null;  
  
    private String gender = null;  
  
    @NotNull  
    private Date birthday;  
  
    @Valid  
    private Phone phone;  
}
```

Phone.java

```
public class Phone {  
  
    @Range(100-999)  
    private Integer area;  
  
    private Integer prefix;  
  
    private Integer number;  
}
```

Errormessages.properties

NotNull= {0} is a required field

NotEmpty= {0} field must have a value

Min= {0} must be greater than {1}

Range = {0} is invalid. It must 3 digits.

Size.validation = Size of the {0} must be between {2} and {1}

StudentId = Student ID

firstName = First Name

lastName = Last Name

birthday = Birthday

phone.area = Area Code

5. [20 points] The validation for the Student Registration in the previous problem [#4] deals with a Web UI scenario. Consider 2 other alternative scenarios for Student Registration:
A REST service and a Batch service.

BATCH simply requires validating: lastName & phone

REST requires: ALL fields to be validated: The fields in problem 4 PLUS gender field.

Assume the gender field requires a value.

Fields to be validated & when to validate them:

	BATCH	WEB UI	REST
studentId		X	X
firstName		X	X
lastName	X	X	X
gender			X
birthday		X	X
phone	X	X	X

Implement Validation groups for the 3 options. Give examples of annotating the fields for each group. For example annotate firstName, lastName, gender.

WEB UI & REST can use the Spring MVC validation functionality for handling groups because they are HTTP based. On the other hand, BATCH group validation uses the Hibernate Validator. It is on a different layer of the N-tier architecture.

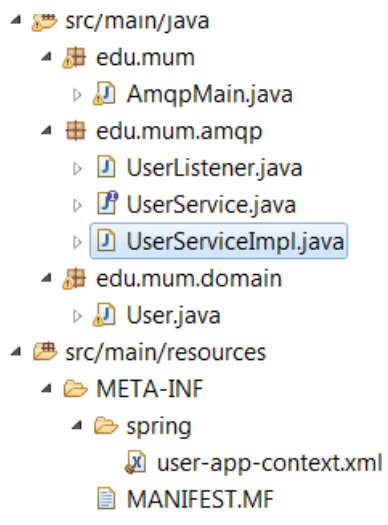
Explain a robust validation strategy related to an N-tier architecture. Describe the value of validation at different layers in the N-tier.

6. [15 points] Complete the following AMQP messaging application.
The project structure and UserService interface are provided.
To be completed:

- user-app-context.xml
- UserServiceImpl.java
- UserListener.java

It is a direct Exchange. The exchange name is *user*. There is a single queue named *userQueue*.
The binding between the exchange & queue is through *user.key*.

Project Structure:



UserService.java

```
public interface UserService {
    public void publish(RabbitTemplate rabbitTemplate);
}
```

user-app-context.xml

Fill in the parts indicated by underline _____

```
<rabbit:connection-factory id="connectionFactory" host="localhost" username="joe" password="joe"/>
<rabbit:admin connection-factory="connectionFactory" />
```

```
<!-- ***** EXCHANGE ***** -->
    <rabbit:queue name="_____userQueue_____" durable="true"/>
```

```
<rabbit:direct-exchange name="_____user_____" durable="true">
    <rabbit:bindings>
        <rabbit:binding __queue__="__userQueue__" key____="__user.key____" />
    </rabbit:bindings>
</rabbit:direct-exchange>
```

```
<!-- ***** PRODUCER -->
associated with it...] -->
    <rabbit:template id="userTemplate" connection-factory="connectionFactory"

        reply-timeout="2000" routing-key = "_____user.key_____"

        exchange="_____user_____" />
```

```
<!-- ***** CONSUMER ***** -->
    <rabbit:listener-container connection-factory="connectionFactory">

        <rabbit:listener ref="queueListener" method="_onMessage_" queue-names="__userQueue_____"
    </rabbit:listener-container>
```

```
<!-- ***** LISTENER -->

<bean id="__ userListener ____" class="__edu.mum.ampq.UserListener_____" />
```

UserServiceImpl.java

```
public class UserServiceImpl implements UserService {  
    public void publish(RabbitTemplate rabbitTemplate) {  
  
        // Send 2 User messages  
        User user = new User("Bill", "Mee", "bMee@usa.com");  
  
        rabbitTemplate.convertAndSend(user);  
  
        user = new User("Paul", "Tree", "pTree@mav.com");  
        rabbitTemplate.convertAndSend(user);  
  
    }  
}
```

UserListener.java

```
public class UserListener {  
  
    public void onMessage(User user) {  
  
        System.out.println("AMQP User on DIRECT Queue: " + user.getFirstName);  
  
    }  
}
```