CS544
Enterprise Architecture Final
Exam 2 May 2016

Name____

Student ID ____

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

1. [10 points] Determine which of the following are TRUE/FALSE concerning Data Binding :

   T **F** Spring security only supports the authentication model HTTP Basic defined by RFC 1945 which is the most popular authentication mechanism in the web.

   EXPLAIN:

   Spring Security supports not only HTTP BASIC but also HTTP Digest, HTTP X.509, Java Open Source Single Sign On (JOSSO) Form-based, Java Authentication and Authorization (JAAS)

   T F Spring ACL is used to give permissions to access methods (i.e. specify who can execute which method)

   EXPLAIN:

   Spring provides URL based and method Level authorization. Some of the method level authorization (permissions to access methods) @PreAuthorize("hasRole('ROLE_ADMIN')"), @PreAuthorize("hasPermission(#comment,'update')")

   T **F** Digest authentication uses Base64 encoding to transmit encrypted usernam/password

   EXPLAIN:

   It is Basic authentication that uses Base64 encoding to transmit encrypted username/password. Digest authentication uses a private key to encrypt username/password

   T **F** Authorization refers to unique identifying information from each system user, generally in the form of a username and password.

   EXPLAIN:

   Authentication is the one that refers to unique identifying information from each system user, generally in the form of a username and password

   T F The ACLs or access control lists are specifically for assuring domain object security

   EXPLAIN:

   Since every secure domain object has an associated definition[ACL] that defines domain object specific authorization [CRUD]. We can say, Access Control List (ACLs) are specifically for assuring domain object security.

2. [15 points] AOP is a Spring Core Technology. It is used in numerous places within the Spring Framework, itself. Explain the fundamentals of Spring's AOP implementation; how it works, how it relates to AspectJ. Give examples of its usage within Spring.

   To help in your explanation of how it works consider the following use case:
   A client application needs to access a server application over the network. For monitoring purposes, it is necessary to log all calls to all save [save(Object object)] methods at the service tier.
   For example:

```
@Service
@Transactional
Class FooServiceImpl {

        @Logging
        public void save (Foo foo) {
           fooDao.save (foo);
        }

        Public List<Foo> findAll() {
           return fooDao.findAll();
        }

        Public Foo findOne(Long id) {
           return fooDao.findOne(id);
        }
}
```

   Using AOP terminology, describe what would need to be implemented. Be as specific as you can with respect to syntax.

   Spring IOC use AOP internally for transaction management, for remote access, for security and cache management. We may also use AOP if we have a requirement to encapsulate some functionality that needs to be isolated, we can use AOP when we have a core functionality that is tangled or scattered over multiple objects and can't refactor using OOP which is also not the main business function, so here we cross cut this functionality and encapsulate it in Aspect.
   This is how it relates to AspectJ
   AspectJ(Static) and Spring AOP(Dynamic) both are cross cutting concerns.
   Spring AOP the cross cutting applies on the run time, uses Proxy based implementation, applies on methods only and Spring managed beans.
   AspectJ ;- the cross cutting applies at compile time, modification performed at bytecode level makes more performance than Spring AOP, can be applied in fields and any java Code
   The implementation of the Aspect is called Advice. Advise will be applied on the JOINPOINT. the POINTCUT matches(expresses) the advice and the JOINTPOINT.

   Implementation:
   In the above example,
   1. The FooServiceImpl class we annotate with @Logging(this is the JOIN POINT) on the method that is going to be checked by our Aspect

2. @Logging is custom annotation interface that we created like
   public @interface Logging{}

3. We need to create aspect class by using @Aspect, on the Advice(on the implementation of the aspect functionality and advice has some keywords like @Before, @After , @Around ...) we express the POINTCUT.

   example : @Before("@Annotation(@logging)")

So our aspect Implementation will be like

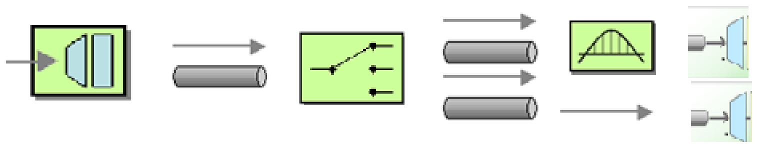
```
        @Aspect
        @Component
        Public class AspectImpl{

                @Before("@annotations(Logging)")
                Public void log(){
                        //TO DO
                }
}
```

3. [20 points] Enterprise Integration Patterns [EIP] are a fundamental definition of how to do integration in a company of any significant size. Spring Integration implements those patterns.

   Explain the fundamental aspects of Spring Integration. Why is it necessary & valuable? Describe the 3 main components. Drawing on the demo from class [Routing an order through the "enterprise"], give details on some of the EIP components.

   Be specific. Give examples. Diagrams are good but be sure to explain them.
   Here is a diagram that you should use to describe [some] components and an ESB type flow:

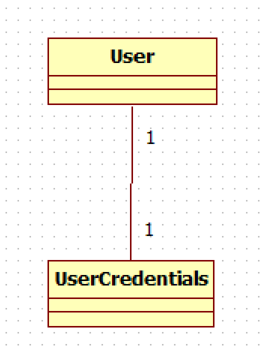


Spring Integration provides a simple model for implementing complex enterprise integration solutions. and facilitate asynchronous, message-driven behavior within a Spring-based application. It also promotes intuitive, incremental adoption for existing Spring users.
Spring Integration is valuable because it follows a certain principle
such as: -
Components should be loosely coupled for modularity and testability.
Enforce separation of concerns between business logic and integration logic
Extension points should be abstract & promote reuse and portability.
The main component of Spring Integrations are message, message Channel and Message Endpoint.
Message: It is a generic wrapper for any Java object combined with metadata used by the framework while handling that object. It consists of a payload and header(s).
Message Channel: A message channel is the component through which messages are moved so it can be thought as a pipe between message producer and consumer. [PTP or Pub/Sub].
Message Endpoint: A message endpoint isolates application code from the infrastructure. In other words, it is an abstraction layer between the application code and the messaging framework.
The demo or the drawing is describing about how a payload comes from one system and passes through a channel which used to connect two different system or adapters. after it passes from the channel it gets to a content router that examines the coming payload to determine the message channel

where to go after determining the right path, it might select the bridge one which used to connect two channels or channel adapters

4. [15 Points] Annotate for validation both the User and UserCredentials from the Auction System. The action that triggers validation is an invocation of - userService.save(user);

   It is NOT necessary to invoke validation, just annotate the domain Models AND externalize the messages in errorMessages.properties [space left at end of question.]



Here are the generated error messages when validation fails:

```
First Name field must have a value
Size of the Last Name must be between 5 and 9
Email must have valid syntax
Ranking must be between 4 and 6

Password must have at least 6 characters
Size of the Login User Name must be between 6 and 16
```

Here is the relevant part of the User Domain Class:

```
@Entity
@Table(name = "USERS")
public class User implements Serializable  {
        @Id @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name = "USER_ID")
        private Long id = null;
        @Version
        private int version = 0;

        @NotEmpty(message="{NotEmpty.validation.message}")
        @Column(name = "FIRSTNAME", nullable = false)
        private String firstName;
        @NotEmpty(message="{NotEmpty.validation.message}")
        @Size(min=5, max=9, message="{Size.validation.message}")
        @Column(name = "LASTNAME", nullable = false)
        private String lastName;

        @Email(message="{Email.validation.message}")
        @Column(name = "EMAIL", nullable = false)
        private String email;
        @Range(min=4, max=6, message = "{Range.rank.validation.message}")
        @Column(name = "RANK", nullable = false)
        private int ranking = 0;
```

```
	@Column(name = "IS_ADMIN", nullable = false)
	private boolean admin = false;

	@OneToOne(fetch=FetchType.EAGER,  cascade = CascadeType.ALL)
	@JoinColumn(name="userId")
	private UserCredentials userCredentials;
```

Here is the UserCredentials:

```
@Entity(name = "Authentication")
public class UserCredentials {
	@Id
	@Column(name = "USER", nullable = false, unique = true, length = 127)
	@Size(min=6, max=16, message="{Size.validation.message}")
	String userName;
	@Size(min=6, message="{Size.password.validation.message}")
	@Column(name = "PASSWORD", nullable = false, length = 32)
	String password;
	@Column( nullable = false, length = 32)
	String verifyPassword;
	Boolean enabled;
	@OneToOne(mappedBy="userCredentials", cascade = {CascadeType.PERSIST, CascadeType.MERGE})
	private User user;
```

ErrorMessage.properties

```
NotEmpty.validation.message= {0} field must have a value
Size.validation.message= Size of the {0} must be between {2} and {1}
Size.password.validation.message = Size of the {0} must have at least 6 characters
Range.rank.validation.message= Rank is incorrect. It must be between 4 & 6
Email.validation.message= Email format is incorrect

firstName= First Name
lastName= Last Name
ranking = Ranking
password = Password
```

5. [15 points]

Messaging is basic to scalable enterprise architectures. We covered two messaging technologies, JMS & AMQP. Explain the fundamentals of messaging.

Be sure to cover: the 2 types of messages, the messaging architecture, and the differences between the two technologies, JMS & AMQP and how they are implemented.

Be specific. Give examples. Diagrams are good but be sure to explain them.

Messaging has a great role for enterprise applications it makes loosely coupled asynchronous and reliable communication between applications. It also used to increase performance by improving response times and by doing some tasks asynchronously, reducing complexity by decoupling and isolating applications, by scaling distribute tasks across machines based on load
The java Message Service (JMS) is a java oriented Middleware API for sending message between two or more clients. JMS is part of java platform, Enterprise edition and it is defined by a specification developed under java Community process as JSR 914 which also mean that it is highly depend on java enterprise edition to create, send, receive and read messages.

Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message oriented middleware. The defining features of amqp are message orientation, queuing, routing, reliability and security.

Difference
Unlike JMS, which only depend on JAVA platform, AMQP is a wire-level protocol. It is a description of the format of the data that sent across the network as a stream of octets. Consequently any tool that can create and interpret messages that conform to this data format can interoperate with any other compliant tool irrespective of implementation language.
JMS is API, it does not use any protocol; A JMS provider like Active could be any underlying protocol to realize the JMS API

6. [20 points] The Spring framework is the “example” architecture that we used in this course. It emphasizes good design, best practices and use of design patterns.

   Explain the value of the framework. Things you might consider:

   N-Tier; Separation of Concerns; Different types of N-tier [Monolith, etc.]; Distributed capabilities; the characteristics & value of any good framework.
   How does Spring “facilitate” a good Enterprise architecture?

   Be specific. Give examples. Diagrams are good but be sure to explain them.

Spring framework is based on several Java standards like JSR 303, JSR 330, JSR 107… and design patterns like Proxy, Factory, MVC, Strategy… which helps us

- enforce good practices and rules.
- increase reliability and reduce programming time
- adhere to the DRY principle
- use boilerplate code from the framework itself
- simplify the development
- separate concerns

It is an N-tier, lightweight application model that has:

- a presentation layer - a layer that separates presentation from the rest of the business logic
- service layer - a layer that has business specific components, that has some business logic, and gives access to data from persistence to any presentation or external service through delegation
- persistence layer - a data layer that hides CRUD functionality from the rest of our business.
- a domain model that can be used throughout our application

There are 3 variations to the N-tier architecture.
1 - Monolith, where all the layers are designed as one project
2 - Functional layer, where each layer is one project by itself
3 - Component, where a project has all the layers (especially the service and persistence) and serves functionalities of one component.