**CORE**

**What is difference between JDK,JRE and JVM?**

JVM is an acronym for Java Virtual Machine, it is an abstract machine which provides the runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (so JVM is plateform dependent).

JRE stands for Java Runtime Environment. It is the implementation of JVM and physically exists.

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

**How many types of memory areas are allocated by JVM?**

Class(Method) Area

Heap - Objects and instance variables are stored in heap

Stack - local variables are stored in stack. Each thread has its own stack

Program Counter Register - address of the next instruction to be executed in stored in PC Register

Native Method Stack

**What gives Java its 'write once and run anywhere' nature?**

The bytecode. Java is compiled to be a byte code which is the intermediate language between source code and machine code. This byte code is not platorm specific and hence can be fed to any platform.

**What is classloader?**

The classloader is a subsystem of JVM that is used to load classes and interfaces.There are many types of classloaders e.g. Bootstrap classloader, Extension classloader, System classloader, Plugin classloader etc.

**If I do not provide any arguments on the command line, then the String array of Main method will be empty or null?**

Its empty, not null

**What is the default value of the local variables?**

The local variables are not initialized to any default value, neither primitives nor object references.

**Few facts about constructor**

It does not explicitly return any value, but implicitly returns the object instance.

Cannot be final or is not inherited

**Few facts about static variables**

Created in class area at the time of class loading, static blocks are executed before the main method. We can execute a program with main method using a static block

**What is marker interface?**

An interface that have no data member and method is known as a marker interface.For example Serializable,Cloneable etc.

**Is Empty .java file name a valid source file name?**

Yes, save your java file by .java only, compile it by javac .java and run by java yourclassname

**Can you use this() and super() both in a constructor?**

No. Because super() or this() must be the first statement.

**Can we overload main method?**

Yes, but the main method with public static void main(String[] args) can be only one

**Can abstract class or Interface be final**

No. Because they always have to be implemented, they cant be final

**Can abstract or interafce method be static?**

No. because an abstract method will not have implementation and hence there is no point in it being static. And also all interface methods are abstract by default

**Will finally block get executed if an exception is caught and there is a return statement in the catch block?**

Yes finally block is executed irrespective of the return statement. But it will not be executed if it occurs after a system.exit(0)

**Can a main class be private or protected?**

No. Compiler throws an error. Main class can only be public or default access modifier

**When can an object reference be cast to an interface reference?**

An object reference can be cast to an interface reference when the object implements the referenced interface.

**How many objects will be created in the following code?**

a. String s1="Welcome"

String s2="Welcome"

String s3="Welcome"

b. String s1 = new String("Welcome")

Only one object "Welcome" will be created in the string literal(constant). b-> 2 objects, one in string literal and other in heap

**Can we call run() method of a thread?**

Yes. But it will be called as a normal method. Only way to execute a thread is to start it first

**How to create a thread**

1. MyThread extends Thread and implement run method

2. MyThread implements Runnable Interface and override run method

3. MyThread t = new Mythread("eg")

4. t.start()

**Different constructors of thread**

Thread(String name)

Thread(Runnable r, String name)

Thread(Runnable)

Thread(String)

**Different states of a thread**

1. New: Once it is instantiated and before start() is called -> New State

2. Runnable: Once start() method is called, thread enters runnable state. will wait in a runnable queue

3. Running: Once thread is picked by the scheduler for execution. It can either go to runnable state or wait/sleep/blocked state from here

4. Wait/Sleep/Blocked: when wait() or sleep() is called or when waiting on a resource

4. Dead - after it complete execution, thread goes to dead state, when start() is invoked on this RunTimeException occurs

**What is Monitor?**

A synchronized block of code on which a thread can obtain a lock. Only one thread can obtain lock to monitor at a time. Once that thread releases the lock, second thread waiting on that object's monitor can obtain the lock

**What is a semaphore?**

Semaphore is a shared object which is used for communication between two threads

**Thread dump and how to take it**

Thread dump will show all the active threads in JVM. We can analyse thread dump to find if there are any dead locks. Jstack tool used to obtain thread dump

**Thread pool**

Thread pool is a pool of worker threads or a colletion of Runnable objects. It has a queue where all tasks to be executed are waiting for execution

Executor executor = Executors.newFixedThreadPool(100)

executor.execute(New MyRunnable(1)) -> this will add runnable object to the thread pool

**Runnable, Callable and future?**

Runnable - I/F that implements run method, does not return anything

Callable - I/F that implements run method, can return a generic object

Future<MyObj> future = executor.submit(New MyCallable(10))

we can use future.get() to get the Myobj instance

**How to achieve asynchronous programming?**

Use Callable and future with executor service interface. once we do a submit, threads are executed asynchronously. does not wait for completion before proceeding to next thread

**What is deadlock**

Thread A waiting for resource A and holding resource B. Thread B holding reasource A and waiting for resource B. This is called dead lock - will keep the threads hanging.

How to prevent -> use synchronized blocks and locks

**What is atomic operations?**

An operation which has to happpen at a stretch and cannot be stopped in middle. Eg. read - write operations other than long or double. long or double can be atomic only if they are declared volatile

**What is volatile keyword**

Volatile keyword before a variable will make it thread safe. All threads read the variable from memory, cache cannot be created for a volatile variable

**How to solve producer - consumer problem**

Use wait() and notify() or BlockingQueue

**How to share data b/w 2 threads**

use shared object (with synchronized keyword) or use a blocking queue

**What is BlockingQueue**

concurrent.BlockingQueue -> implementation is such that it waits for the queue to be non-empty before removing an item from the queue and waits for the queue to have free space before loading items. It solves the producer-consumer issue

**How to avoid concurrency issues or how to make an object immutable?**

Avoid concurrency issues by using immutable classes. Class should be final, all fields private final, no setter method

Avoid concurrency issues on collections by making a defensive copy and using that, eg. passing a collections.unmodifiableList(list) to another function so that the other function cant change the list

**Difference between process and thread**

Process - heavy weight, a program or an application. inter process communication is costly

Thread - a small task in the process, light weight, inter thread communication is easy

**Difference between user thread and deomon thread**

user thread - jvm shuts down only after all user threads are executed

deomon thread - jvm shuts down even if its running. it runs in the background. .setDeamon(true) will create a deamon thread

**Difference between synchronized method and synchronized block**

synchronized method - gets implicit lock on that object

synchronized block - can get lock on any object. This is advantageous.

**Difference between wait, sleep and yield**

sleep -> thread goes to sleep state, does not release lock on monitor, can go to runnable state based on priority after the sleep time

wait -> thread goes to suspended state, releases lock. can go to runnable state only when a notify() is called on that object or notifyAll is called by another object

yield -> yields scheduler time to another thread, releases lock

**Difference between notify and notifyAll**

notify -> wakes up an arbitrary thread waiting on that object

notifyAll -> wakes up all the threads waiting on that object

**Difference between submit and execute()**

Both methods are used to submit tasks to thread pool

submit -> implemented in Executor interface, returns future object

execute -> implemented in Executor Service interface, does not return anything

**Difference between ArrayList and Vector**

ArrayList and Vector - for both, underlying implementation is Arrays

ArrayList -> Not synchronized and hence not thread safe, ie the add and remove methods do not have synchronized keyword

Vector -> Synchronized and thread safe.

When increasing the size, vector just doubles the current size whereas ArrayList increases the size by 50%

**Difference between ArrayList and LinkedList**

ArrayList -> Implements List, Uses index for saving and retrieving. Random retrieving and adding is fast using index, but deletion takes time since all other items has to be moved after deletion.

LinkedList -> Implements List and Queue Uses iterator for saving and retrieving, retrieving from middle of the list is slower than arraylist but deletion is very faster than arraylist

**Difference between List and Set**

List -> Retrieve in the same order it is saved. Can contain duplicates

Set -> Does not guarantee same order retrievel. cannot contain duplicates. If we want ordered set, go for treeset.

**Implementations of List and Set**

List -> Vector, ArrayList, LinkedList

Set -> TreeSet, HashSet (all objects are stored as keys in a hashmap)

**Can set contain null values?**

HashSet can contain one null value, but in treeset null value can be stored only if atleast one element is already present in the treeset. Since set checks for duplicates while inserting, if the first element is null, then it leads to null pointer exception

**Difference between HashMap and HashTable**

HashMap -> Not synchronized/Hash table -> synchronized.

HashMap -> failsafe/ Hashtable ->fail fast

HashMap -> can contain one null key/ Hashtable -> Cannot contain a null key

**Difference between fail-safe and fail-fast iterators?**

fail-fast -> Iterators for all Collections in util.package are fail-fast iterators. If we modify an element after starting the iterator, by any other means other than using the iterator mthod, Concurrent Modification Exception is thrown

fail-safe -> Iterators for all Collections in java.concurrent package are fail-safe. So they never throw a ConcurrentModificationException

**Different types of list from Collection**

Collections.synchronizedList(list)

Collections.sort(list,String.String.CASE\_INSENSITIVE\_ORDER) -> to sort list irrespective of case

Collections.sort(list,comp) -> to sort list based on a comparator

Collections.unmodifieablelist(list) -> To create an unmodifiable list

**What is shallow cloning and deep cloning?**

Shallow cloning - Just the top layer of the object is copied, no new references are created for dependent objects. Object.clone() defaults to shallow cloning

Deep cloning - replicates entrire object structure. creates new reference for dependent objects..

**why cloning?**

We can clone an object and do modifications and operations on that object without modifying the original object

**How to create an immutable class**

class should be final, all variables private final and can be set only thru' constructor, no setter for any variable. If there are any mutable objects that are referenced, dont provide any method to modify that mutable object or dont share that mutable object's reference

**Which classes are immutable in java?**

All wrapper classes like String,Interger,Boolean,Double,Char are immutable in java

**How to remove elements from a list without getting concurrent modification exception**

Use Iterator to iterate through the list, and use iterator.remove() method to remove any element from the list

**SPRING**

**Benefits of using spring**

Minimizes the amount of code in your application.Loose coupling.IOC containers support eager instantiation and lazy loading of services.Spring has layered architecture. Open source

**Different modules in Spring framework**

Core, Web, MVC, AOP, ORM, JDBC (Spring DAO), Context

**What is Spring configuration file?**

Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and related to each other.

**What is Dependency Injection?**

Injecting the object dependencies through configuration files. Class A depedentant of class B, to over come the dependency of B on A, inject B in to A. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. Spring IOC Container is responsible for hooking up the objects.

This concept says that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (the IOC container) is then responsible for hooking it all up.

**Different types of IoC (dependency injection)**

<ul class="list">

Constructor-based dependency injection: Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.

Setter-based dependency injection: Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

</ul>

**Which DI would you suggest Constructor-based or setter-based DI?**

Since you can mix both, Constructor- and Setter-based DI, it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies. Note that the use of a <i>@Required</i> annotation on a setter can be used to make setters required dependencies.

**What are Spring beans?**

A bean is an object that is instantiated, assembled, and managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML <bean/> definitions.

**What does a bean definition contain?**

How to create a bean, Bean's lifecycle details,bean scopes etc

**How do you provide configuration metadata to the Spring Container?**

XML based configuration file.,Annotation-based configuration,Java-based configuration

**How do add a bean in spring application?**

Check the following example:

<bean id="helloWorld" class="com.tutorialspoint.HelloWorld">

<property name="message" value="Hello World!"/>

</bean>

**Explain Bean lifecycle in Spring framework?**

Instantiate - First the spring container finds the bean's definition from the XML file and instantiates the bean.

Populate properties - Using the dependency injection, spring populates all of the properties as specified in the bean definition.

Set Bean Name - If the bean implements BeanNameAware interface, spring passes the bean's id to setBeanName() method.

Set Bean factory - If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.

Pre Initialization - Also called postprocess of bean. If there are any bean BeanPostProcessors associated with the bean, Spring calls postProcesserBeforeInitialization() method.

Initialize beans - If the bean implements IntializingBean,its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.

Post Initialization - If there are any BeanPostProcessors associated with the bean, their postProcessAfterInitialization() methods will be called.

Ready to use - Now the bean is ready to use by the application.

Destroy - If the bean implements DisposableBean , it will call the destroy() method .

**What are inner beans in Spring?**

A <bean/> element inside the <property/> or <constructor-arg/> elements defines a so-called inner bean. An inner bean definition does not require a defined id or name the container ignores these values. It also ignores the scope flag. Inner beans are always anonymous and they are always scoped as prototypes.

**How do you define a bean scope?**

while defining a bean in xml file, we can mention the bean scope. bean id="bean id" scope="prototype" or by using @Scope("prototype") annotation. Default scope is singleton

**What bean scopes does Spring support? Explain them.**

The Spring Framework supports following five scopes, three of which are available only if you use a web-aware ApplicationContext.

singleton: This scopes the bean definition to a single instance per Spring IoC container.

prototype: This scopes a single bean definition to have any number of object instances.

request: This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

session: This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

global-session: This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

**What is default scope of bean in Spring framework?**

Singleton

**Are Singleton beans thread safe in Spring Framework?**

No, singleton beans are not thread-safe in Spring framework.

**What is Auto wiring?**

The Spring container is able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory.

1 <beans>

2 <bean id="bar" class="com.act.Foo" Autowire=”autowire type”/>

3 </beans>

**What are different types of Autowiring?**

no, byName,byType,constructor,autodetect

What are the important beans lifecycle methods?

There are two important bean lifecycle methods. The first one is setup which is called when the bean is loaded in to the container. The second method is the teardown method which is called when the bean is unloaded from the container.

<beans>

<bean id="bar" class="com.act.Foo" init-method=”fooSetup” destroy=”fooTeardown”/>

</beans>

**How to integrate your Struts application with Spring?**

Configure Spring to manage your Actions as beans, using the ContextLoaderPlugin, and set their dependencies in a Spring context file.

**What is DelegatingVariableResolver?**

Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard Java Server Faces managed beans mechanism which lets you use JSF and Spring together. This variable resolver is called as DelegatingVariableResolver

**How to integrate Java Server Faces (JSF) with Spring?**

JSF and Spring do share some of the same features, most noticeably in the area of IOC services. By declaring JSF managed-beans in the faces-config.xml configuration file, you allow the FacesServlet to instantiate that bean at startup. Your JSF pages have access to these beans and all of their properties.We can integrate JSF and Spring in two ways:

DelegatingVariableResolver: Spring comes with a JSF variable resolver that lets you use JSF and Spring together.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"

"http://www.springframework.org/dtd/spring-beans.dtd">

<faces-config>

<application>

<variable-resolver>

org.springframework.web.jsf.DelegatingVariableResolver

</variable-resolver>

</application>

</faces-config>

The DelegatingVariableResolver will first delegate value lookups to the default resolver of the underlying JSF implementation, and then to Spring's 'business context' WebApplicationContext. This allows one to easily inject dependencies into one's JSF-managed beans.

FacesContextUtils:custom VariableResolver works well when mapping one's properties to beans in faces-config.xml, but at times one may need to grab a bean explicitly. The FacesContextUtils class makes this easy. It is similar to WebApplicationContextUtils, except that it takes a FacesContext parameter rather than a ServletContext parameter.

ApplicationContext ctx = FacesContextUtils.getWebApplicationContext(FacesContext.getCurrentInstance())

Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard JavaServer Faces managed beans mechanism. When asked to resolve a variable name, the following algorithm is performed:

Does a bean with the specified name already exist in some scope (request, session, application)? If so, return it

Is there a standard JavaServer Faces managed bean definition for this variable name? If so, invoke it in the usual way, and return the bean that was created.

Is there configuration information for this variable name in the Spring WebApplicationContext for this application? If so, use it to create and configure an instance, and return that instance to the caller.

If there is no managed bean or Spring definition for this variable name, return null instead.

BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

As a result of this algorithm, you can transparently use either JavaServer Faces or Spring facilities to create beans on demand.

**What is Significance of JSF- Spring integration ?**

Spring - JSF integration is useful when an event handler wishes to explicitly invoke the bean factory to create beans on demand, such as a bean that encapsulates the business logic to be performed when a submit button is pressed.

<body>

**What are the exceptions thrown by the Spring DAO classes ?**

Spring DAO classes throw exceptions which are subclasses of DataAccessException(org.springframework.dao.DataAccessException).Spring provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception.

**What is SQLExceptionTranslator ?**

SQLExceptionTranslator, is an interface to be implemented by classes that can translate between SQLExceptions and Spring's own data-access-strategy-agnostic org.springframework.dao.DataAccessException.

**What is Spring's JdbcTemplate ?**

With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database.

JdbcTemplate template = new JdbcTemplate(myDataSource)

**What is PreparedStatementCreator ?**

Is one of the most common used interfaces for writing data to database.

Has one method – createPreparedStatement(Connection)

Responsible for creating a PreparedStatement.

Does not need to handle SQLExceptions.

What is RowCallbackHandler ?

The RowCallbackHandler interface extracts values from each row of a ResultSet.

Has one method – processRow(ResultSet) Called for each row in ResultSet.

**What is AOP?**

Aspect-oriented programming, or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management. The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules.

**How the AOP used in Spring?**

AOP is used in the Spring Framework: To provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management, which builds on the Spring Framework's transaction abstraction.To allow users to implement custom aspects, complementing their use of OOP with AOP.

**What do you mean by Aspect ?**

A modularization of a concern that cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).

**What do you mean by JointPoint?**

A point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution.

**What do you mean by Advice?**

Action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors "around" the join point.

**What are the types of Advice?**

Before advice: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).

After returning advice: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.

After throwing advice: Advice to be executed if a method exits by throwing an exception.

After (finally) advice: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).

Around advice: Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception

**How to integrate Spring and Hibernate using HibernateTemplate?**

1. Define datasource bean

<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="org.hsqldb.jdbcDriver"/>

<property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/>

<property name="username" value="sa"/>

<property name="password" value=""/>

</bean>

2. Define SessionFactory bean

<bean id="sqlSessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

<property name="dataSource" ref="myDataSource" />

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect"> org.hibernate.dialect.HSQLDialect</prop>

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.hbm2ddl.auto">create</prop>

</props>

</property>

</bean>

3. Define HibernateTemplate bean

<bean id = "hibernateTemplate" class = "org.springframework.orm.hibernate3.HibernateTemplate"/>

<constructor-arg index="0" ref="sqlSessionFactory" />

</bean>

Inject hibernatetemplate in DAO class and use it

@Inject

HibernateTemplate hibernateTemplate

hibernateTemplate.save(obj)

hibernateTemplate.update(obj) etc.,

**How to integrate Spring and Hibernate using HibernateDaoSupport?**

Spring and Hibernate can integrate using Spring’s SessionFactory called LocalSessionFactory. The integration process is of 3 steps.

Configure the Hibernate SessionFactory

Extend your DAO Implementation from HibernateDaoSupport

Wire in Transaction Support with AOP

What are ORM’s Spring supports ?

Hibernate,iBatis,JPA (Java Persistence API)

What are the ways to access Hibernate using Spring ?

Inversion of Control with a HibernateTemplate and Callback

**Explain Spring MVC**

1. Dispatcher Servlet is configured in web.xml with url pattern \*.\*. So all requests go to dispatcher servlet. It acts as a front controller

2. With the help of SimpleUrlHandlerMapping configured in applicationcontext.xml, request is sent to the appropriate controller

3. Controller processes the request and sends response through ModelAndViewObject.

4. With the help of ViewResolver, view is resolved and MV object is sent to the view

5. View displays the response to user

**Dispatcher Servlet**

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>2</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>\*.\*</url-pattern>

</servlet-mapping>

**SimpleUrlHandlerMapping**

<bean id='simpleUrlMapping' class='org.springframework.web.servlet.handler.SimpleUrlHandlerMapping'>

<property name='mappings'>

<props>

<prop key='/showAllMails.jsp'>showController</prop>

<prop key='/composeMail.jsp'>composeController</prop>

<prop key='/deleteMail.jsp'>deleteController</prop>

</props>

</property>

</bean>

**ViewResolver**

<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix">

<value>/WEB-INF/jsp/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

**Types of Spring Controllers**

AbstractController : Simple controller, extend and override handleRequestInternal method. Used when someone wants to write their own custom controller

SimpleFormController Used for forms with submit button. FormView (Class that represents form inputs), and successView (output view)

CancellableFormController extension of simpleformcontroller. used to take action if cancel button is pressed on forms (implements onCancel method)

AbstractWizardFormController when there are multiple pages with multiple submit buttons, use WizardFormController controller

ParameterizableViewController forward directly to a jsp page without actual controller (no java controller)

MultiActionController One controller for multiple pages (ie. multiple submit)

AbstractCommandController,

ServletForwardingController,

ServletWrappingController,

UrlFilenameViewController,

**Controller - Bean Configuration example**

<bean id="loginValidator" class="net.roseindia.web.LoginValidator"/>

<bean id="loginController" class="net.roseindia.web.LoginFormController">

<property name="sessionForm"><value>false</value></property>

<property name="commandName"><value>login</value></property>

<property name="commandClass"><value>net.roseindia.web.Login</value></property>

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

<property name="formView"><value>login</value></property>

<property name="successView"><value>success</value></property>

</bean>

**Controller - Code Implementation example**

public class UserFormController extends SimpleFormController {

@Override

protected ModelAndView onSubmit(request, response, command, errors) throws ServletException {

User user = (User) command

String name = user.getName()

String prestatement = "Hello"

ModelAndView modelAndView = new ModelAndView(getSuccessView())

modelAndView.addObject("user", user)

modelAndView.addObject("prestatement", prestatement)

return modelAndView

} (or)

override showForm(request,response)

override doSubmitAction(command)

implements Controller -> and override handleRequestInternal(request,response)

**HIBERNATE**

**Benefits of using hibernate**

Productivity. Maintainability, portability (can easily switch to any other sql driver)

**Explain Hibernate flow**

1. Write a pojo representing the table 2. Write a hbm (hibernate mapping file) specifying the mapping information between table and pojo. 3. Write a configuration file - it has details like driver class, url, username, password, sql dilect, showsql, default schema, mapping file location

<div class="question">hbm file example >/div>

<hibernate-mapping>

<class name="com.test.User" table="user">

<property column="USER\_NAME" length="255"

name="userName" not-null="true" type="java.lang.String"/>

<property column="USER\_PASSWORD" length="255"

name="userPassword" not-null="true" type="java.lang.String"/>

</class>

</hibernate-mapping>

**Basic hibernate code**

SessionFactory sessionFactory = Configuration.configure.buildSessionFactory(); Session session = sessionFactory.openSession(); Transaction transaction = session.beginTransaction(); session.save(obj); transaction.commit();

**What is ORM?**

ORM Stands for Object Relational Mapping. It represents automated persistence of java objects in database. There are different levels of ORM Pure relational (stored procedure.) Light objects mapping (JDBC) Medium object mapping Full object Mapping (composition,inheritance, polymorphism, persistence by reachability)

**Core Interfaces of hibernate**

Configuration,

Session Factory ,

Session,

Transaction,

Query and Criteria interfaces

**What is SessionFactory**

There is only one sesssionfactory in an application. It is multi-threaded,We can get sessions from session factory

**What is Session**

Session is a short lived, single-threaded object obtained from SF. It represents the conversation bw the appl. and persistance store. It is used to save objects and retrieve objects from the persistance store

**Difference between load() and get()**

Only use the load() method if you are sure that the object exists. If you are not sure that the object exists, then use one of the get() methods.

load() method will throw an exception if the unique id is not found in the database. get() method will return null if the unique id is not found in the database.

load() just returns a proxy by default and database won’t be hit until the proxy is first invoked. get() will hit the database immediately.

**Difference between and merge and update**

Use update() if you are sure that the session does not contain an already persistent instance with the same identifier, and merge() if you want to merge your modifications at any time without consideration of the state of the session.

**What is generator**

Used for generating the id column, there are different classes for generator such as sequence, increment, native etc.

<id column="USER\_ID" name="id" type="java.lang.Long">

<generator class="sequence">

<param name="table">SEQUENCE\_NAME</param>

<generator>

</id>

**How to access an instance variable directly and not through a setter method ?**

By mapping the property with access="field" in Hibernate metadata. This forces hibernate to bypass the setter method and access the instance variable directly while initializing a newly loaded object.

**How can a whole class be mapped as immutable?**

Mark the class as mutable="false" (Default is true),. This specifies that instances of the class are (not) mutable. Immutable classes, may not be updated or deleted by the application.

**Types of Hibernate instance states ?**

Transient -Newly created objects which are not yet associated with any session. Can make it persistant by calling save or update method

Persistent -The instance that is associated with a session

Detached -The instance was associated with a session which has been closed – currently not associated. Can make it persistant by calling save or update method

**Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.

cascade="all|none|save-update|delete|all-delete-orphan"

inverse - mark this collection as the "inverse" end of a bidirectional association.

inverse="true|false"

Essentially "inverse" indicates which end of a relationship should be ignored

**What are derived properties?**

The properties that are not mapped to a column, but calculated at runtime by evaluation of an expression are called derived properties. The expression can be defined using the formula attribute of the element.

**What is HQL**

Hibernate query Language (HQL), is an object-oriented extension to SQL. flexible mechanism to query, store, update, and retrieve objects from a database

**Named – SQL query**

Named SQL queries are defined in the mapping xml document and called wherever required.<sql-query name = "empdetails">

<return alias="emp" class="com.test.Employee"/>

SELECT statement

</sql-query>

List people = session.getNamedQuery("empdetails").setString("TomBrady", name).setMaxResults(50).list();

**How to invoke stored procedure**

<sql-query name="selectAllEmployees\_SP" callable="true">

<return alias="emp" class="employee">

<return-property name="empid" column="EMP\_ID"/>

<return-property name="name" column="EMP\_NAME"/>

<return-property name="address" column="EMP\_ADDRESS"/>

{ ? = call selectAllEmployees() }

</return>

</sql-query>

**Criteria API example**

List employees = session.createCriteria(Employee.class)

.add(Restrictions.like("name", "a%") )

.add(Restrictions.like("address", "Boston"))

.addOrder(Order.asc("name") )

.list();

**What is Hibernate Template**

HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.Sessions are automatically closed.

**What is SQL Dialects?**

Using Hibernate SQL Dialects , we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

**What is Hibernate proxy?**

Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a getter method of the proxy is invoked.

**What is the use of dynamic-insert and dynamic-update attributes in a class mapping?**

dynamic-update (defaults to false): Specifies that UPDATE SQL should be generated at runtime and contain only those columns whose values have changed

dynamic-insert (defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

**What do you mean by fetching strategy ?**

There are different fetching strategies to fetch associated instances and collections

Join Fetching - sql outer join in the same select statement

select fetching - a different select statement. the 2nd select is called only when the assoc. is hit

batch fetching - batches. can specify batch size in collection declarations

lazy fetching - by default lazy-fetching is true for collections. assoc elements are fetched only when they are invoked

proxy fetching - associated single-valued instances are fetched using this strategy. proxy is returned first and db is hit only when the instance is invoked

**What is automatic dirty checking?**

Without any explicit calls to save or update, hibernate automatically determines that a collection or object is modified inside a transaction and updates the DB. As long as the object is in persistant state (i.e., associated with a session) hibernate monitors the changes and executes SQL in write-behind fashion. This is called automatic dirty checking.

**What is transactional write-behind?**

Hibernate uses a sophisticated algorithm to determine an efficient ordering that avoids database foreign key constraint violations. This feature is called transactional write-behind.

saved or deleted.

**What are callback interfaces**

Callback interfaces are used to receive notification when some object events occur. Like when an object is loaded, saved or deleted.

**What is lock method**

Session lock method associates an object with a session, but does not sync it with database. Where as update method will sync it with database and associate the object with the session

**Pagination using Hibernate**

Criteria criteria = Session.createCriteria(Student.class);

criteria.add(Restrictions.like("name","a%");

criteria.addOrder(Order.asc("name"));

criteria.setMaxResults(maxResults);

criteria.setFirstResult(maxResults \* (pageNum-1));

List<Student> studentList = criteria.list();

Same can be done using Query Interface also

Other pagination solutions -> Use ScrollableResult result = Query.scroll(); or use SQL DB Pagination using row no.

**What is First level Cache in Hibernate?**

Also called as session level cache. It is default and mandatory cache. When an object is updated inside a session, hibernate does not update the DB and retains it in the session cache until possible. Only when the session is closed, the object is persisted to DB.

When similar select statements are made, hibernate retrieves data for 2nd select from the cache rather than from the DB.

Example: Employee emp1 = session.get(1);

Employee emp2 = session.get(1); -> Obtained from session cache

session.getTransaction().commit();

session.close();

Employee emp3 = session.get(1); -> Hibernate looks for this object in first level cache, it'll not be found, so it looks in 2nd level cache, if its not found then hibernate hits DB

**What is Second level Cache in Hibernate?**

Second level Cache is maintained at SessionFactory level and can improve performance by saving few database round trip. second level cache is available to whole application rather than any particular session.

From the above example it is evident that 2nd level cache, objects are available across sessions, clusters and applications.

**How to configure a Second level Cache in Hibernate?**

In Hibernate.cfg.xml file configure cache

<property name = "cache.use-secondlevel-cache">true </property>

<property name = "cache.provider-class">org.hibernate....EHCache</property>

Add @Cacheable annotation to entity, @cache(Usage = CacheConcurrencyStrategy.READ\_ONLY)

**Different concurrent strategies in 2nd level cache**

Read Only - Only select

Read Write - update and fetch - updates cache when data is updated, does not read dirty data

Non Strict Read write - reads dirty data ie. data updated in some other place

Transactional - cache is transactional level

**What is query cache in Hibernate ?**

QueryCache actually stores result of sql query for future calls. Query cache can be used along with second level cache for improved performance. Hibernate support various open source caching solution to implement Query cache e.g. EhCache.

Query.setCacheable(true) option available with Query criteria

**How to clear cache ?**

session.evict(Employee.class) -> Will clear first level cache

sessionFactory.evict(Employee.class) -> Will clear second level cache

Class Student{

String name;

Address address; --> One to One Mapping

Department department; --> Many to One Mapping

Set<Phone> Phonenos = new HashSet<Phone>(); --> One to Many

Set<Elective> electivesList = new HashSet<Elective>(); --> Many to Many

}

**One-to-One Mapping**

<class name = "com.user.Student" table = "STUDENT" >

<property name="studentName" type="string" not-null="true" length="100" column="STUDENT\_NAME" />

<many-to-one name="studentAddress" class="com.user.Address" column="STUDENT\_ADDRESS" not-null="true" cascade="all" unique="true" />

Student s1 = new Student("Shriya",new Address("123","abc","456"));

session.save(s1); --> will generate two insert statements, one to insert address record and the other to insert student record

**Many-to-One relationship**

<property name="studentName" type="string" length="100" not-null="true" column="STUDENT\_NAME" />

<many-to-one name="studentAddress" class="com.vaannila.student.Address" column="STUDENT\_ADDRESS" cascade="all" not-null="true" />

Student s1 = new Student("Shriya",new Department(1, "Aeronautics","ABlock"));

Student s2 = new Student("Liya",new Department(1, "Aeronautics","ABlock"));

session.save(s1); --> will generate three insert statements, one to insert department record and the other to insert 2 student record

**One-to-Many relationship**

<property name="studentName" type="string" not-null="true" length="100" column="STUDENT\_NAME" />

<set name="studentPhoneNumbers" table="STUDENT\_PHONE" cascade="all">

<key column="STUDENT\_ID" />

<many-to-many column="PHONE\_ID" unique="true" class="com.vaannila.student.Phone" />

</set>

**Many-to-Many (unidirectional) relationship**

same as one-to-many without unique = true

**All types of relationships**

One-to-One |

Many-to-One |\_\_\_\_\_\_\_Unidirectional, Unidir using join table, Bidir using foreign key, Bidir using jointable

One-to-Many |

Many-to-Many |

**What is Component Mapping**

Component - contained object

<class name="eg.Person" table="person">

<id name="Key" column="pid" type="string">

<generator class="uuid"/>

</id>

<component name="Name" class="eg.Name"> <!-- class attribute optional -->

<property name="initial"/>

<property name="first"/>

<property name="last"/>

</component>

</class>

**List Inheritance strategies in hibernate**

Table per class hierarchy

Table per subclass

Table per concrete class

**CORE**

**Can an abstract class contain static method?**

Yes. It can contain static method

**Can an abstract class have a constructor?**

Yes.

**Why static variables cannot be accessed by non-static methods or vice versa?**

Static methods and variables doesn’t need any object to start. So when a non-static variable is used that will not be initialized during its usage and hence will lead to exceptions

How to access private methods of a class?

Using Reflections

**What is the difference between static block and init block?**

Static block will be called once when jvm loads the class. Init block will be called everytime an object is instantiated. Static block will be executed first followed by the init block

**What is transient keyword?**

When some of the properties of a class are marked transient, they will not be included in Serialization

**How to change heap size in java?**

Java –Xms64m –Xms1280 program

Difference between instance of and isIstance(Object obj)?

First one – reserved keyword

2nd - method