Prelims with Solution
Application Development Using Advance Java
(M.Sc.IT-1)

1 (a). Objective type questions:

1. DI stands for - **Dependancy Injection**
2. What is the default spring bean scope. **Singleton**
3. IOC stands for : **Inversion of Control**
4. What is Autowiring?

Autowiring feature of spring framework enables you to inject the object dependency implicitly.

1 (b). Attempt any one out of two:

1. Explain Pointcut.

- PointCut is a set of one or more JoinPoint where an advice should be executed.
- You can specify PointCuts using expressions or patterns.
- In Spring, PointCut helps to use specific JoinPoints to apply the advice.

examples −

- @PointCut("execution(* com.springtutorial.*.*(..))")
- @PointCut("execution(* com.springtutorial.Student.getName(..))")

2. Explain Factory Method.

- Spring framework provides facility to inject bean using factory method.
- we can use two attributes of bean element:

1. **factory-method:** represents the factory method that will be invoked to inject the bean.
2. **factory-bean:** represents the reference of the bean by which factory method will be invoked. It is used if factory method is non-static.

- A method that returns instance of a class is called factory method.

```
public class A {
public static A getA()  {//factory method
   return new A();
}  }
```

1 (c). **Attempt any one out of two:**

1. **Explain Collection.**

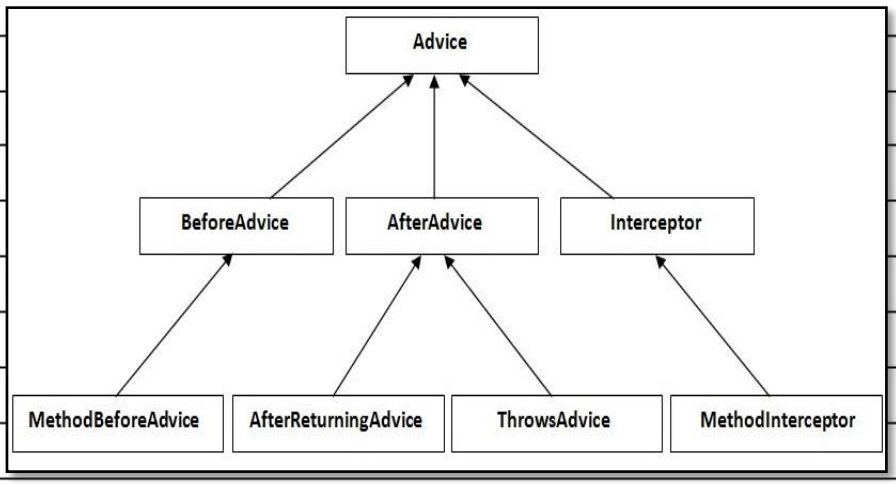
- Hibernate provides map collection elements of Persistent class in Hibernate.
- Declare the type of collection in Persistent class from one of the following types:
  - java.util.List
  - java.util.Set
  - java.util.SortedSet
  - java.util.Map
  - java.util.SortedMap
  - java.util.Collection
- There are many subelements of <class> elements to map the collection.
- To map the collection elements are <list>, <bag>, <set> and <map>.
- Example of mapping file for mapping collection:

```
<class name=" Question" table="q100">
   <id name="id">
   <generator class="increment"></generator>
   </id>
   <property name="qname"></property>
   <list name="answers" table="ans100">
   <key column="qid"></key>
   <index column="type"></index>
   <element column="answer" type="string"></element>
   </list>
</class>
```

- There are three subelements used in the list:
- <key> element is used to define the foreign key in this table based on the Question class identifier.
- <index> element is used to identify the type.
- <element> is used to define the element of the collection.
- The collection elements can be categorized in two forms: indexed and non-indexed
- The List and Map collection are indexed whereas set and bag collections are non-indexed.
- Indexed collection means List and Map requires an additional element <index>.

2. Explain Advices.

- This is the actual action to be taken either before or after the method execution.
- This is the actual piece of code that is invoked during program execution by Spring AOP framework.
- There are different types of advices:
  - ✓ **Before Advice**: it executes before a join point.
  - ✓ **After Returning Advice:** it executes after a joint point completes normally.
  - ✓ **After Throwing Advice:** it executes if method exits by throwing an exception.
  - ✓ **After (finally) Advice:** it executes after a join point regardless of join point exit whether normally or exceptional return.
  - ✓ **Around Advice:** It executes before and after a join point.
- All are interfaces in AOP.
- Let's understand the advice hierarchy by the diagram given below:



1 (d). Attempt any one out of two:

1. Explain AOP Terminology.

AOP concepts and terminologies are as follows:

1. Join point

- Join point is any point in your program such as method execution, exception handling, field access etc. Spring supports only method execution join point.

2. Advice

- Advice represents an action taken by an aspect at a particular join point. There are different types of advices:

- **Before Advice**: it executes before a join point.
- **After Returning Advice:** it executes after a joint point completes normally.
- **After Throwing Advice:** it executes if method exits by throwing an exception.
- **After (finally) Advice:** it executes after a join point regardless of join point exit whether normally or exceptional return.
- **Around Advice:** It executes before and after a join point.

3. **Pointcut**
   - It is an expression language of AOP that matches join points.
4. **Introduction**
   - It means introduction of additional method and fields for a type. It allows you to introduce new interface to any advised object.
5. **Target Object**
   - It is the object i.e. being advised by one or more aspects. It is also known as proxied object in spring because Spring AOP is implemented using runtime proxies.
6. **Aspect**
   - It is a class that contains advices, joinpoints etc.
7. **Interceptor**
   - It is an aspect that contains only one advice.
8. **AOP Proxy**
   - It is used to implement aspect contracts, created by AOP framework. It will be a JDK dynamic proxy or CGLIB proxy in spring framework.
9. **Weaving**
   - It is the process of linking aspect with other application types or objects to create an advised object. Weaving can be done at compile time, load time or runtime. Spring AOP performs weaving at runtime.

2. Write a Program for a question having a multiple answers using constructor injection with Map. Display Question id, Question, answers. Write a code for java & xml file.

Question.java

This class contains three properties, two constructors and displayInfo() method to display the information.

```
import java.util.*;
public class Question {
private int id;
private String name;
private Map<String,String> answers;
public Question() {}
public Question(int id, String name, Map<String, String> answers)
{  this.id = id;
   this.name = name;
   this.answers = answers;
}
public void displayInfo( )
{  System.out.println("question id:"+id);
   System.out.println("question name:"+name);
   System.out.println("Answers....");
   Set<Entry<String, String>> set=answers.entrySet();
   Iterator<Entry<String, String>> itr=set.iterator();
   while(itr.hasNext()){
      Entry<String,String> entry=itr.next();
      System.out.println("Answer:"+entry.getKey()+" Posted By:"+entry.getValue());
 } } }
```

applicationContext.xml

The **entry** attribute of **map** is used to define the key and value information.

```
<bean id="q" class=" Question">
<constructor-arg value="11"></constructor-arg>
<constructor-arg value="What is Java?"></constructor-arg>
<constructor-arg>
<map>
<entry key="Java is a Programming Language"  value="Ajay Kumar"></entry>
<entry key="Java is a Platform" value="John Smith"></entry>
<entry key="Java is an Island" value="Raj Kumar"></entry>
</map>
</constructor-arg>
</bean>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the displayInfo() method.

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
public class Test {
public static void main(String[] args) {
    Resource r=new ClassPathResource("applicationContext.xml");
    BeanFactory factory=new XmlBeanFactory(r);
    Question q=(Question)factory.getBean("q");
    q.displayInfo();
} }
```

2 (a).	**Objective type questions:**
1.	**RowMapper** interface allows to map a row of the relations with the instance of user-defined class.
2.	Controller class renders the objects passed by the controller's handler method.
	[True/False] ---- **True**
3.	SPEL stands for **Spring Expression Language**
4.	**getJdbcTemplate()** is method to retrieve the JDBC template.
2 (b).	**Attempt any one out of two:**
1.	**Explain Http Invoker.**
•	Spring provides its own implementation of remote service known as HttpInvoker.
•	It can be used for http request than RMI and works well across the firewall.
•	By the help of HttpInvokerServiceExporter and HttpInvokerProxyFactoryBean classes, we can implement the remote services provided by Spring's Http Invokers.
•	Http Invoker and Other Remote techniques
	Http Invoker Vs RMI
	○ RMI uses JRMP protocol whereas Http Invokes uses HTTP protocol.
	○ Since enterprise applications mostly use http protocol, it is the better to use HTTP Invoker.
	○ RMI also has some security issues than HTTP Invoker. HTTP Invoker works well across firewalls.
	Http Invoker Vs Hessian and Burlap
	○ HTTP Invoker is the part of Spring framework but Hessian and burlap are proprietary.
	○ All works well across firewall. Hessian and Burlap are portable to integrate with other languages such as .Net and PHP but HTTP Invoker cannot be.
2.	**Explain Named Parameter.**
•	Spring provides another way to insert data by named parameter.
•	In such way, we use names instead of ?(question mark).
•	So it is better to remember the data for the column.

- Simple example of named parameter query

```
insert into employee values (:id,:name,:salary)
```

- Method of NamedParameterJdbcTemplate class : In this example, we are going to call only the execute method of class.

Syntax of the method is as follows:

```
pubic T execute(String sql,Map map,PreparedStatementCallback psc)
```

2 (c). Attempt any one out of two:

1. Explain operators in SpEL.

- We can use many operators in SpEL such as arithmetic, relational, logical etc. There are given a lot of examples of using different operators in SpEL.
- **Examples of using operators in SPEL**

```
import org.springframework.expression.*;
import org.springframework.expression.spel.standard.*;
public class Test {
public static void main(String[] args) {
ExpressionParser parser = new SpelExpressionParser();
 //arithmetic operator
System.out.println(parser.parseExpression("'Welcome SPEL'+'!'").getValue());
System.out.println(parser.parseExpression("10 * 10/2").getValue());
System.out.println(parser.parseExpression("'Today is: '+
                                        new java.util.Date()").getValue());
  //logical operator
System.out.println(parser.parseExpression("true and true").getValue());
  //Relational operator
System.out.println(parser.parseExpression("'sonoo'.length()==5").getValue());
} }
```

2. **Explain Spring with JMS.**

- To integrate spring with JMS, you need to create two applications.

1. JMS Receiver Application
2. JMS Sender Application

- To create JMS application using spring, we are using Active MQ Server of Apache to create the Queue.
- Let's see the simple steps to integration spring application with JMS:

Required Jar Files : spring core, spring misc, spring aop, spring j2ee and spring persistence core jar files.

- Add activemqall5.9.jar file located inside the activemq directory.
- Create a queue in ActiveMQ Server
- Download the Active MQ Server
- Double Click on the activemq.bat file located inside apache-activemq-5.9.1-bin\apache-activemq-5.9.1\bin\win64 or win32 directory.
- Now activemq server console will open.
- Access the admin console of activemq server by http://localhost:8161/admin/ url.
- Now, click on the Queues link, write myqueue in the textfield and click on the create button.

2 (d). **Attempt any one out of two:**

1. **Write a Program to print cube of given number using Spring and RMI.**

Calculation.java

It is the simple interface containing one method cube.

```
public interface Calculation
{    int cube(int number);  }
```

CalculationImpl.java

This class provides the implementation of Calculation interface.

```
public class CalculationImpl implements Calculation{
    public int cube(int number) {
        return number*number*number;  }  }
```

applicationContext.xml

In this xml file we are defining the bean for CalculationImpl class and **RmiServiceExporter** class. We need to provide values for the following properties of RmiServiceExporter class.

- ✓ service
- ✓ serviceInterface
- ✓ serviceName
- ✓ replaceExistingBinding
- ✓ registryPort

```
<bean id="cBean" class="CalculationImpl"></bean>
<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
   <property name="service" ref="cBean"></property>
   <property name="serviceInterface" value="Calculation"></property>
  <property name="serviceName" value="CalculationService"></property>
   <property name="replaceExistingBinding" value="true"></property>
   <property name="registryPort" value="1099"></property>
</bean>
```

client-beans.xml

In this xml file, we are defining bean for **RmiProxyFactoryBean**. You need to define two properties of this class.

- ✓ serviceUrl
- ✓ serviceInterface

```
<bean id="cBean"class="org.springframework.remoting.rmi.RmiProxyFactoryBean">
<property name="serviceUrl" value="rmi://localhost:1099/CalculationService">
</property>
<property name="serviceInterface" value="Calculation"></property>
</bean>
```

Host.java

It is simply getting the instance of ApplicationContext. But you need to run this class first to run the example.

```
import org.springframework.context.*;
import org.springframework.context.support.*;
public class Host{
public static void main(String[] args){
ApplicationContext context = new ClassPathXmlApplicationContext("applicati
onContext.xml");
System.out.println("Waiting for requests");
} }
```

Client.java

This class gets the instance of Calculation and calls the method.

```
import org.springframework.context.*;
import org.springframework.context.support.*;
 public class Client {
public static void main(String[] args)  {
ApplicationContext context = new ClassPathXmlApplicationContext("client-
beans.xml");
Calculation calculation = (Calculation)context.getBean("calculationBean");
System.out.println(calculation.cube(7));
} }
```

2. Write a Program to display all records of Employee Table (id, name, salary) using ResultSetExtractor.

Employee.java

```
public class Employee {
private int id;
private String name;
private float salary;
//no-arg and parameterized constructors  and getters and setters
public String toString(){      return id+" "+name+" "+salary;  }  }
```

EmployeeDao.java

```
import java.sql.*;
import java.util.*;
import org.springframework.dao.*;
import org.springframework.jdbc.core.*;
public class EmployeeDao {
private JdbcTemplate template;
public void setTemplate(JdbcTemplate template) {
this.template = template;  }
public List<Employee> getAllEmployees(){
 return template.query("select * from employee",new ResultSetExtractor<List<Employ
ee>>(){  public List<Employee> extractData(ResultSet rs) throws SQLException,
            DataAccessException
 {     List<Employee> list=new ArrayList<Employee>();
        while(rs.next())
{     Employee e=new Employee();
        e.setId(rs.getInt(1));
        e.setName(rs.getString(2));
        e.setSalary(rs.getInt(3));
        list.add(e);          }
        return list;     }  });
} }
```

applicationContext.xml

```
<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource
<property name="driverClassName" value="com.mysql.jdbc.Driver" />
<property name="url" value="jdbc:mysql://localhost:3306/TestDB" />
<property name="username" value="root" />
<property name="password" value="root" /> </bean>
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="ds"></property> </bean>
<bean id="edao" class="EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property> </bean>
```

Test.java

```
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Test {
public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext
                                                ("applicationContext.xml");
   EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
   List<Employee> list=dao.getAllEmployees();
   for(Employee e:list)
System.out.println(e);     }   }
```

3 (a).	**Answer the following.**
1.	**OXM is an acronym for :** Object XML Mapping
2.	**JAXB stands for : Java Architecture for XML Binding**
3.	**What is XStream?**
	Xstream is a library to serialize objects to xml and vice-versa without requirement of any mapping file.
4.	**Which is the root node/element of hbm.xml file?**
	<hibernate-mapping> </hibernate-mapping>
3 (b).	**Answer in detail. (ANY ONE)**
1.	**Explain Hibernate Annotation.**
•	The hibernate application can be created with annotation.
•	There are many annotations that can be used to create hibernate application such as @Entity, @Id, @Table etc.
•	Hibernate Annotations are based on the JPA 2 specification and supports all the features.
•	All the JPA annotations are defined in **the javax.persistence package**.
•	The core advantage of using hibernate annotation is that you don't need to create mapping (hbm) file.
2.	**Explain <hibernate-mapping> tag.**
•	The mapping document is an XML document having <hibernate-mapping> as the root element, which contains all the <class> elements.
•	The <class> elements are used to define specific mappings from a Java classes to the database tables.
•	The <meta> element is optional element and can be used to create the class description.
•	The <id> element maps the unique ID attribute in class to the primary key of the database table.
•	The <generator> element within the id element is used to generate the primary key values automatically.
•	The <property> element is used to map a Java class property to a column in the database table.

3 (c). Answer in brief. (ANY ONE)

1. Explain Hibernate Generator Classes.

- The **<generator> class** is a sub-element of id.
- It is used to generate the unique identifier for the objects of persistent class.
- There are many generator classes defined in the Hibernate Framework.
- All the generator classes implements the **org.hibernate.id.IdentifierGenerator interface.**
- Hibernate framework provides many built-in generator classes:

For example:

```
<id ...>
  <generator class="type"></generator>
</id>
```

assigned	the default generator strategy if there is no <generator> element .
increment	uses for the auto increment field of DB. It generates **short, int or long type** identifier.
sequence	uses sequence of the database. (i.e s1, s2...)
hilo	It uses high and low algorithm to generate the id of type **short, int and long**.
identity	It is used in Sybase, MySQL, MS SQL Server, DB2 etc. to support the id column. (short, int or long).
native	It uses identity, sequence or hilo depending on the database vendor.
seqhilo	combination of Sequence and hilo
uuid	generate 128 bit Universal Unique ID
guid	genertated by DB of type String specially used with MySql
select	It uses the primary key returned by the database trigger.
foreign	It uses the id of another associated object.
sequence-identity	It uses a special sequence generation strategy. It is supported in Oracle 10g drivers only.

2. Explain Hibernate Dialects.

- For connecting any hibernate application with the database, you must specify the SQL dialects.
- Dialects classes defined for RDBMS in the org.hibernate.dialect package.
- Few of them are as follows:

RDBMS	Dialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle9i	org.hibernate.dialect.Oracle9iDialect
Oracle10g	org.hibernate.dialect.Oracle10gDialect
MySQL	org.hibernate.dialect.MySQLDialect
DB2	org.hibernate.dialect.DB2Dialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect

3 (d). Write a note on following Questions. (ANY ONE)

1. Explain Hibernate with Log4j 1.

- Logging enables the programmer to write the log details into a file permanently.
- Log4j and Logback frameworks can be used in hibernate framework to support logging.
- There are two ways to perform logging using log4j:
  - By log4j.xml file (or)
  - By log4j.properties file

Levels of Logging

- Following are the common logging levels.

Levels	Description
OFF	This level is used to turn off logging.
WARNING	This is a message level that indicates a problem.
SEVERE	This is a message level that indicates a failure.
INFO	This level is used for informational messages.
CONFIG	This level is used for static configuration messages.

Steps to perform Hibernate Logging by Log4j using xml file

- There are two ways to perform logging using log4j using xml file:

1. Load the log4j jar files with hibernate
2. Create the log4j.xml file inside the src folder (parallel with hibernate.cfg.xml file)

Example

Create log4j.xml file

- Now you need to create log4j.xml file.

log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"  debug="false">
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
 <layout class="org.apache.log4j.PatternLayout">
  <param name="ConversionPattern" value="[%d{dd/MM/yy hh:mm:ss:sss z}] %5p %
c{2}: %m%n" />  </layout>  </appender>
   <appender name="ASYNC" class="org.apache.log4j.AsyncAppender">
      <appender-ref ref="CONSOLE" />
      <appender-ref ref="FILE" />  </appender>
<appender name="FILE" class="org.apache.log4j.RollingFileAppender">
   <param name="File" value="C:/test.log" />
   <param name="MaxBackupIndex" value="100" />
 <layout class="org.apache.log4j.PatternLayout">
  <param name="ConversionPattern" value="[%d{dd/MM/yy hh:mm:ss:sss z}] %5p %
c{2}: %m%n" />  </layout>  </appender>
   <category name="org.hibernate">
      <priority value="DEBUG" />    </category>
   <category name="java.sql">
      <priority value="debug" />     </category>
   <root>
      <priority value="INFO" />
      <appender-ref ref="FILE" />     </root>  </log4j:configuration>
```

2. Explain Hibernate with Log4j 2.

- As we know, Log4j and Logback frameworks are used to support logging in hibernate, there are two ways to perform logging using log4j:
  - By log4j.xml file (or)
  - By log4j.properties file
- Here, we are going to enable logging using log4j through properties file.

Steps to perform Hibernate Logging by Log4j using properties file

- There are two ways to perform logging using log4j using properties file:
- Load the log4j jar files with hibernate
- Create the log4j.properties file inside the src folder (parallel with hibernate.cfg.xml file)

Example : Create log4j.properties file

- Now you need to create log4j.properties file. In this example, all the log details will be written in the C:\\test.log file.

log4j.properties

```
# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:\\test.log
log4j.appender.file.MaxFileSize=1MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
 %m%n
 # Direct log messages to stdout
        log4j.appender.stdout=org.apache.log4j.ConsoleAppender
        log4j.appender.stdout.Target=System.out
        log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
        log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1
}: %L - %m%n
        # Root logger option
        log4j.rootLogger=INFO, file, stdout
        # Log everything. Good for troubleshooting
```

```
log4j.logger.org.hibernate=INFO
# Log all JDBC parameters
log4j.logger.org.hibernate.type=ALL
```

4 (a). Answer the following.

1. Table per Object Inheritance model is available in Hibernate. [True/False]

False

2. hbm2ddl.auto property of hibernate configuration file is used to enable automatic table creation option.

3. In hibernate; database table configuration is stored in hibernate.cfg.xml file.

4. Bidirectional association allows us to fetch details of dependent object from both side.

4 (b). Answer in brief. (ANY ONE)

1. Explain one to many by Set.

Question.java

```
import java.util.Set;
public class Question {
private int id;
private String qname;
private Set<String> answers;
//getters and setters
}
```

question.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
 <hibernate-mapping>
<class name=" Question" table="q100">
  <id name="id">
```

```
    <generator class="increment"></generator>
   </id>
   <property name="qname"></property>
   <set name="answers" table="ans100">
    <key column="qid"></key>
    <element column="answer" type="string"></element>
   </set>   </class>  </hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
      <property name="hbm2ddl.auto">update</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
 <property name="connection.url">jdbc:mysql://localhost:3306/Test</property>
    <property name="connection.username">root</property>
     <property name="connection.password"></property>
  <property name="connection.driver_class"> com.mysql.jdbc.Driver</property>
   <mapping resource="question.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

StoreData.java

```
import java.util.*;
import org.hibernate.*;
import org.hibernate.boot.*;
import org.hibernate.boot.registry.*
public class StoreData {
 public static void main(String[] args)
```

```
{     Session session = new    Configuration().configure("question.hbm.xml").
buildSessionFactory().openSession();
          Transaction tx=session.beginTransaction();
    HashSet<String> list1=new  HashSet <String>();
    list1.add("Java is a programming language");
    list1.add("Java is a platform");

    Question question1=new Question();
    question1.setQname("What is Java?");
    question1.setAnswers(list1);

    session.persist(question1);
    t.commit();
    session.close();
    System.out.println("success");
 }
}
```

2. Explain Lazy Collection.

- When working with an ORM, data fetching/loading can be classified into two types: **eager and lazy**.
- Hibernate provide both types of collection.
- Lazy collection loads the child objects on demand, it is used to improve performance.
- Since Hibernate 3.0, lazy collection is enabled by default.
- To use lazy collection, you may optionally use lazy="true" attribute in your collection.
- It is by default true, so you don't need to do this. If you set it to false, all the child objects will be loaded initially which will decrease performance in case of big data.

4 (c). **Answer in detail. (ANY ONE)**

1. **Explain Table Per Concrete using Annotation.**

File: Employee.java

```
import javax.persistence.*;
@Entity
@Table(name = "employee102")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Employee
{  @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name = "id")
        private int id;
        @Column(name = "name")
        private String name;
        //setters and getters
}
```

File: Regular_Employee.java

```
import javax.persistence.*;
@Entity
@Table(name="regularemployee102")
@AttributeOverrides({
   @AttributeOverride(name="id", column=@Column(name="id")),
   @AttributeOverride(name="name", column=@Column(name="name"))
})
public class Regular_Employee extends Employee
{  @Column(name="salary")
        private float salary;
        @Column(name="bonus")
        private int bonus;
        //setters and getters
}
```

File: Contract_Employee.java

```
import javax.persistence.*;
@Entity
@Table(name="contractemployee102")
@AttributeOverrides({
   @AttributeOverride(name="id", column=@Column(name="id")),
   @AttributeOverride(name="name", column=@Column(name="name"))
})
public class Contract_Employee extends Employee{
     @Column(name="pay_per_hour")
   private float pay_per_hour;
   @Column(name="contract_duration")
   private String contract_duration;
     public float getPay_per_hour() {
       return pay_per_hour;      }
   public void setPay_per_hour(float payPerHour) {
       pay_per_hour = payPerHour;      }
   public String getContract_duration() {
       return contract_duration;      }
   public void setContract_duration(String contractDuration) {
       contract_duration = contractDuration;    }
}
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
      <property name="hbm2ddl.auto">update</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
 <property name="connection.url">jdbc:mysql://localhost:3306/Employee</property>
     <property name="connection.username">root</property>
     <property name="connection.password"></property>
    <property name="connection.driver_class"> com.mysql.jdbc.Driver</property>
```

```
  <mapping class="Employee"/>
   <mapping class="Regular_Employee"/>
      <mapping class="Contract_Employee"/>
   </session-factory>
</hibernate-configuration>
```

StoreData.java

```
import org.hibernate.*;
import org.hibernate.cfg.AnnotationConfiguration;
public class StoreData
{    public static void main(String[] args)
     {    Session session = new
          AnnotationConfiguration().configure().buildSessionFactory().openSession();
           Transaction tx=session.beginTransaction();
         Employee e1=new Employee();
         e1.setName("Dhaval Shah");
        Regular_Employee e2=new Regular_Employee();
        e2.setName("Hardik Makawana");
        e2.setSalary(50000);
        e2.setBonus(5);
       Contract_Employee e3=new Contract_Employee();
        e3.setName("Dhruv Parekh");
        e3.setPay_per_hour(1000);
        e3.setContract_period("15 hours");
        session.persist(e1);
        session.persist(e2);
        session.persist(e3);
        tx.commit();
        session.close();
        System.out.println("success");
          }}
```

2. Explain Table Per Subclass using XML.

Example :

File: Employee.java

```
public class Employee
{  private int id;
private String name;
  //getters and setters
}
```

File: Regular_Employee.java

```
public class Regular_Employee extends Employee
{
    private float salary;
    private int bonus;
      //getters and setters
}
```

File: Contract_Employee.java

```
public class Contract_Employee extends Employee
{
   private float pay_per_hour;
   private String contract_duration;
  //getters and setters
}
```

File: employee.hbm.xml

```
 <hibernate-mapping>
  <class name=" Employee" table="emp123">
  <id name="id">
  <generator class="increment"></generator>
  </id>
    <property name="name"></property>
```

```
<joined-subclass name="Regular_Employee" table="regemp123">
<key column="eid"></key>
<property name="salary"></property>
<property name="bonus"></property>
</joined-subclass>

<joined-subclass name="Contract_Employee" table="contemp123">
<key column="eid"></key>
<property name="pay_per_hour"></property>
<property name="contract_duration"></property>
</joined-subclass>
  </class>    </hibernate-mapping>
```

File: hibernate.cgf.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
     <property name="hbm2ddl.auto">update</property>
     <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
   <property name="connection.url">jdbc:mysql://localhost:3306/Test</property>
    <property name="connection.username">root</property>
  <property name="connection.password"></property>
<property name="connection.driver_class"> com.mysql.jdbc.Driver </property>
   <mapping resource="employee.hbm.xml"/>
   </session-factory>  </hibernate-configuration>
```

File: StoreData.java

```
import org.hibernate.*;
import org.hibernate.cfg.*;
 public class StoreData {
   public static void main(String[] args) {
    Configuration cfg=new Configuration();
    cfg.configure("hibernate.cfg.xml");
    SessionFactory factory=cfg.buildSessionFactory();
     Session session=factory.openSession();
     Transaction tx=session.beginTransaction();
     Employee e1=new Employee();
     e1.setName("Gaurav Chawla");
      Regular_Employee e2=new Regular_Employee();
       e2.setName("Vivek Kumar");
       e2.setSalary(50000);
       e2.setBonus(5);
     Contract_Employee e3=new Contract_Employee();
       e3.setName("Arjun Kumar");
       e3.setPay_per_hour(1000);
       e3.setContract_duration("15 hours");
       session.persist(e1);
       session.persist(e2);
       session.persist(e3);
       tx.commit();
       session.close();
       System.out.println("success");
   } }
```

4 (d). Write a note on following Questions. (ANY ONE)

1. Explain Mapping List with example.

Question.java

```
import java.util.List;
public class Question {
private int id;
private String qname;
private List<String> answers;
//getters and setters
}
```

question.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
 <hibernate-mapping>
 <class name="Question" table="q100">
  <id name="id">
    <generator class="increment"></generator>
  </id>
  <property name="qname"></property>
  <list name="answers" table="ans100">
    <key column="qid"></key>
    <index column="type"></index>
    <element column="answer" type="string"></element>
  </list>
 </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
      <property name="hbm2ddl.auto">update</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
 <property name="connection.url">jdbc:mysql://localhost:3306/Test</property>
    <property name="connection.username">root</property>
     <property name="connection.password"></property>
  <property name="connection.driver_class"> com.mysql.jdbc.Driver</property>
   <mapping resource="question.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

StoreData.java

```
import java.util.ArrayList;
import org.hibernate.*;
import org.hibernate.boot.*;
import org.hibernate.boot.registry.*
public class StoreData {
 public static void main(String[] args)
{     Session session = new   Configuration().configure("question.hbm.xml").
           buildSessionFactory().openSession();
           Transaction tx=session.beginTransaction();
          ArrayList<String> list1=new ArrayList<String>();
           list1.add("Java is a programming language");
            list1.add("Java is a platform");

            ArrayList<String> list2=new ArrayList<String>();
            list2.add("Servlet is an Interface");
           list2.add("Servlet is an API");
```

```
Question question1=new Question();
   question1.setQname("What is Java?");
   question1.setAnswers(list1);

   Question question2=new Question();
   question2.setQname("What is Servlet?");
   question2.setAnswers(list2);

   session.persist(question1);
   session.persist(question2);

   t.commit();
   session.close();
   System.out.println("success");
 }
}
```

2. Explain Mapping Bag with example.

Question.java

```
import java.util.List;
public class Question {
private int id;
private String qname;
private List<String> answers;
//getters and setters
}
```

question.hbm.xml

```
 <hibernate-mapping>
<class name="Question" table="q100">
  <id name="id">
    <generator class="increment"></generator>
```

```
  </id>
  <property name="qname"></property>
  <bag name="answers" table="ans100">
    <key column="qid"></key>
    <element column="answer" type="string"></element>
  </bag>
 </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
      <property name="hbm2ddl.auto">update</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
 <property name="connection.url">jdbc:mysql://localhost:3306/Test</property>
    <property name="connection.username">root</property>
     <property name="connection.password"></property>
  <property name="connection.driver_class"> com.mysql.jdbc.Driver</property>
   <mapping resource="question.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

StoreData.java

```
import java.util.*;
import org.hibernate.*;
import org.hibernate.boot.*;
import org.hibernate.boot.registry.*
public class StoreData {
 public static void main(String[] args)
{     Session session = new   Configuration().configure("question.hbm.xml").
buildSessionFactory().openSession();
          Transaction tx=session.beginTransaction();
```

```
ArrayList<String> list1=new ArrayList<String>();
list1.add("Java is a programming language");
list1.add("Java is a platform");
Question question1=new Question();
question1.setQname("What is Java?");
question1.setAnswers(list1);
session.persist(question1);
t.commit();
session.close();
System.out.println("success");     }    }
```

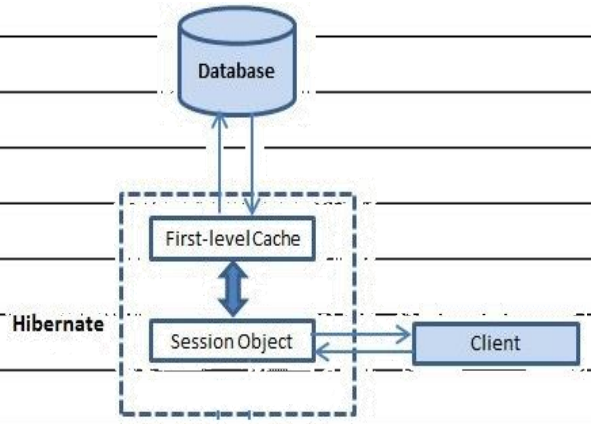
5 (a). Answer the following.

1. HQL stands for : **Hibernate Query Language**
2. The object of Criteria can be obtained by calling the **createCriteria()** method of Session interface.
3. What is Component Mapping?
   In component mapping, we will map the dependent object as a component.
4. **session factory** object holds second level cache data.

5 (b). Answer in brief. (ANY ONE)

1. Explain first level cache.

- Session object holds the first level cache data. It is enabled by default.
- The first level cache data will not be available to entire application.
- An application can use many session object.



2. Explain Association Mapping.

- The mapping of associations between entity classes and the relationships between tables is the soul of ORM.
- Following are the four ways in which the relationship between the objects can be expressed.
- An association mapping can be unidirectional as well as bidirectional.

Sr.No.	Mapping type & Description
1	Many-to-One : Mapping many-to-one relationship using Hibernate
2	One-to-One : Mapping one-to-one relationship using Hibernate
3	One-to-Many : Mapping one-to-many relationship using Hibernate
4	Many-to-Many : Mapping many-to-many relationship using Hibernate

5 (c). Answer in detail. (ANY ONE)

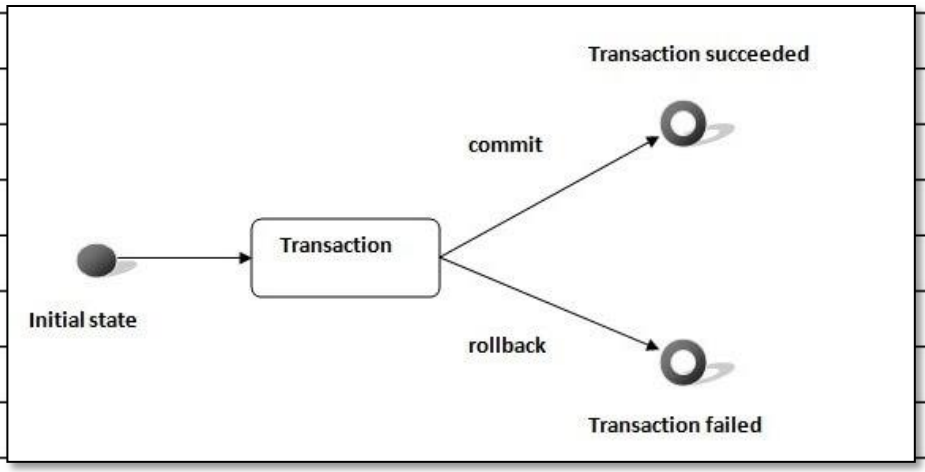
1. Explain Named Query.

- The hibernate named query is way to use any query by some meaningful name. It is like using alias names. The Hibernate framework provides the concept of named queries so that application programmer need not to scatter queries to all the java code.
- There are two ways to define the named query in hibernate:
  a. by annotation
  b. by mapping file.
- @NameQueries annotation is used to define the multiple named queries.
- @NameQuery annotation is used to define the single named query.
- Let's see the example of using the named queries:

```
@NamedQueries(
   {
      @NamedQuery(
      name = "findEmployeeByName",
      query = "from Employee e where e.name = :name"   )
} )
```

2. Explain Transaction Management in Hibernate.

- A transaction simply represents a unit of work. In such case, if one step fails, the whole transaction fails (which is termed as atomicity).
- A transaction can be described by **ACID properties (Atomicity, Consistency, Isolation and Durability).**



- In hibernate framework, we have **Transaction interface** that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC).
- A transaction is associated with Session and instantiated by calling session.beginTransaction().
- The methods of Transaction interface are as follows:

1. **void begin()** starts a new transaction.
2. **void commit()** ends the unit of work
3. **void rollback()** forces this transaction to rollback.
4. **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. **boolean isAlive()** checks if the transaction is still alive.
6. **boolean wasCommited()** checks if the transaction is committed successfully.
7. **boolean wasRolledBack()** checks if the transaction is rolled back successfully.

5 (d).	**Write a note on following Questions. (ANY ONE)**
1.	**Explain with example One-to-One Mapping using Primary Key.**
	Employee.java
	public class Employee {
	private int employeeId;
	private String name,email;
	private Address address;
	//Getters and Setters
	}
	Address.java
	public class Address {
	private int addressId;
	private String addressLine1,city,state,country;
	private int pincode;
	private Employee employee;
	}
	hibernate.cfg.xml
	<hibernate-configuration>
	<session-factory>
	<property name="hbm2ddl.auto">update</property>
	<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
	<property name="connection.url">jdbc:mysql://localhost:3306/Test</property>
	<property name="connection.username">root</property>
	<property name="connection.password"></property>
	<property name="connection.driver_class"> com.mysql.jdbc.Driver</property>
	<mapping resource="employee.hbm.xml"/>
	<mapping resource="address.hbm.xml"/>
	</session-factory>
	</hibernate-configuration>

address.hbm.xml

```
<hibernate-mapping>
 <class name="Address" table="address2111">
        <id name="addressId">
        <generator class="increment">
                </generator>
        </id>
   <property name="addressLine1"></property>
        <property name="city"></property>
        <property name="state"></property>
        <property name="country"></property>
         <property name="pincode"></property>
        <one-to-one name="employee"></one-to-one>
        </class>
    </hibernate-mapping>
```

employee.hbm.xml

```
<hibernate-mapping>
        <class name="Employee" table="emp2111">
        <id name="employeeId">
        <generator class="increment"></generator>
        </id>
        <property name="name"></property>
        <property name="email"></property>
        <one-to-one name="address" cascade="all"></one-to-one>
        </class>
</hibernate-mapping>
```

StoreData.java

```
import org.hibernate.*;
import org.hibernate.boot.*;
import org.hibernate.boot.registry.*;
 public class Store {
public static void main(String[] args) {
        Session session = new   Configuration().configure("question.hbm.xml").
        buildSessionFactory().openSession();
        Transaction t=session.beginTransaction();
         Employee e1=new Employee();
        e1.setName("Ravi Malik");
        e1.setEmail("ravi@gmail.com");
        Address address1=new Address();
        address1.setAddressLine1("G-21,Lohia nagar");
        address1.setCity("Ghaziabad");
        address1.setState("UP");
        address1.setCountry("India");
        address1.setPincode(201301);
        e1.setAddress(address1);
        address1.setEmployee(e1);
        session.persist(e1);
        t.commit();
        session.close();
        System.out.println("success");
}
}
```

2. Create a project illustrating one-to-one using foreign key.

Employee.java

```
public class Employee {
private int employeeId;
private String name,email;
private Address address;
//Getters and Setters
}
```

Address.java

```
public class Address {
private int addressId;
private String addressLine1,city,state,country;
private int pincode;
private Employee employee;
}
```

hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
<property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
 <property name="connection.url">jdbc:mysql://localhost:3306/Test</property>
    <property name="connection.username">root</property>
    <property name="connection.password"></property>
  <property name="connection.driver_class"> com.mysql.jdbc.Driver</property>
   <mapping resource="employee.hbm.xml"/>
   <mapping resource="address.hbm.xml"/>
   </session-factory>
  </hibernate-configuration>
```

address.hbm.xml

```
<hibernate-mapping>
 <class name="Address" table="address2111">
        <id name="addressId">
        <generator class="foreign">
        <param name="property">employee</param>
        </generator>
        </id>
   <property name="addressLine1"></property>
        <property name="city"></property>
        <property name="state"></property>
        <property name="country"></property>
         <property name="pincode"></property>
        <one-to-one name="employee"></one-to-one>
        </class>
    </hibernate-mapping>
```

employee.hbm.xml

```
<hibernate-mapping>
        <class name="Employee" table="emp2111">
        <id name="employeeId">
        <generator class="increment"></generator>
        </id>
        <property name="name"></property>
        <property name="email"></property>
        <one-to-one name="address" cascade="all"></one-to-one>
        </class>
</hibernate-mapping>
```

StoreData.java

```
import org.hibernate.*;
import org.hibernate.boot.*;
import org.hibernate.boot.registry.*;
 public class Store {
public static void main(String[] args) {
        Session session = new   Configuration().configure("question.hbm.xml").
        buildSessionFactory().openSession();
        Transaction t=session.beginTransaction();
         Employee e1=new Employee();
        e1.setName("Ravi Malik");
        e1.setEmail("ravi@gmail.com");
        Address address1=new Address();
        address1.setAddressLine1("G-21,Lohia nagar");
        address1.setCity("Ghaziabad");
        address1.setState("UP");
        address1.setCountry("India");
        address1.setPincode(201301);
        e1.setAddress(address1);
        address1.setEmployee(e1);
        session.persist(e1);
        t.commit();
        session.close();
        System.out.println("success");
}
}
```