# Advanced Java - Java Notes

Advanced Java (Pokhara University)

Date: .....
Page: .....

# RMI Architecture :-

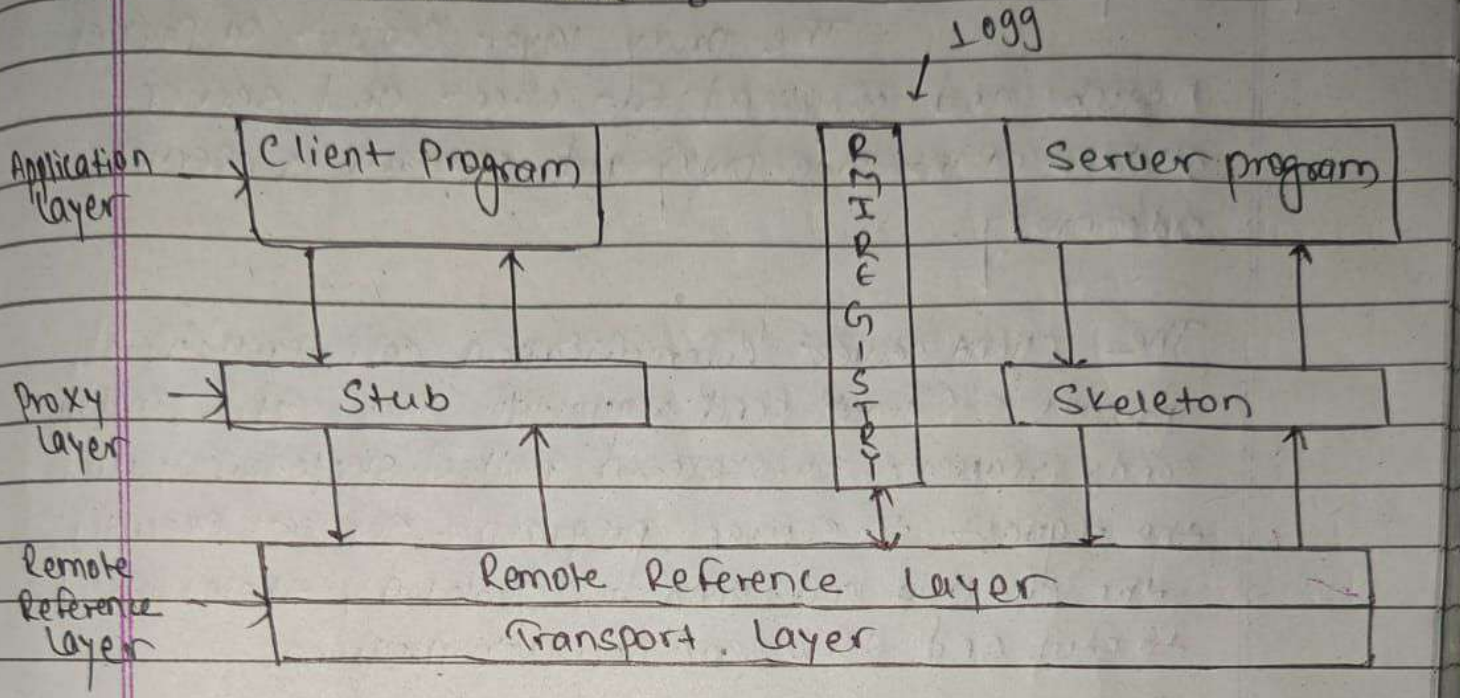



fig :- Architecture of RMI

The RMI architecture consist of ~~Ap~~ 3 layers, Application layer, Proxy layer and Remote reference layer ( Transport layer is the part of Remote reference layer ).

1) Application layer :-

This layer is the actual systems, i.e. client and servers, which are involved in communication. The java program on the client side communicates with the java program on the server side.

Date: .....
Page: .....

# Steps to create RMI client server distributed architecture :-

Step1 :- Create a Interface by executing Remote Interface.

Eg :-

```
import java.rmi.Remote;
public interface Hello extends remote {
    public String greeting() throws RemoteException;
}
```

Step2 :- Implement the interface you created by extending unicast remote object.

Eg :-

```
public class HelloImpl extends UnicastRemoteObject
                        implements Hello
{
    @Override
    public String greeting() throws RemoteException
    {
        return HelloImpl() throws RemoteException
    {
        super();
    }
}
```

Date:
Page:

# RMI ( Remote Method Invocation ) :-

The RMI is an API that provides a mechanism to create distributed applications in Java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

1) Stub :-

The stub is an object, acts as a gateway for client side. All the outgoing requests are routed through it.

2) Skeleton :-

The skeleton is an object, acts as a gateway for the server side objects. All the incoming requests are routed through it.

Date:
Page:

The following are the paths followed by a JSP.

1) Compilation.
2) Initialization.
3) Execution.
4) Cleanup.

The compilation process involves three steps:-

1. Parsing the JSP.
2. Turning the JSP into a servlet
3. Compiling the servlet.

2) JSP Initialization :-

When a container loads a JSP it invokes the JSPInit() method before servicing any requests. If we need to perform JSP-specific initialization, override the JSPInit() method

```
Public void JSPInit() {
    // Initialization code
}
```

3) JSP Execution :-

This phase of the JSP lifecycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes _jspService method in JSP.

Step4:- Create a client program :-

```
public class SI_Client
{
  public static void main (String[] args)
{
  try {
    Registry r = LocateRegistry.getRegistry (1099);
SimpleInterest SI = (SimpleInterest) r.lookup("SimpleInterest");

Scanner sc = new Scanner (System.in);
System.out.print ("Enter Principle :");
double principle = sc.nextDouble ();

System.out.print(" Enter rate :");
double principle = sc.nextDouble ();

System.out.print (" Enter time :");
double principle = sc.nextDouble ();

scanner.close();

double interest = SI.calculateInterest (P, r, t);
System.out.println (" SI = " + interest);
}
  catch (Exception e) { }
}
}
}
```

2) Proxy layer :-

The proxy layer consist of proxies (stub and skeleton) for client and server. Stub is client side proxy and skeleton is server side proxy.

The client server communication goes through the proxies. Client send request to stub, stub in turn sends request to skeleton, then skeleton passes the request to server program. Server executes the method and send to skeleton, skeleton send to stub and stub send to client program.

3) Remote Reference layer :-

The Remote Reference layer connects the proxy layer to the RMI mechanism. This layer is responsible for communicating and transfering objects between client and server. In this layer, the actual implementation of communication protocol is handled.

4) Transport layer :-

The transport layer is responsible for setting up communication between the two machines. This layer uses standard TCP/IP protocol for connection. This layer performs the actual transportation of data. This layer doesnot exist seperately but is part of Remote Reference layer.

Date:
Page:

Step3:- Public Create server program:-

```
public class SIserver {
  public static void main (String ... args)
{
  try
    {
     Registry r = locateRegistry.CreateRegistry (1099);
     r.rebind = (" SimpleInterest ", new SimpleInterestImpl());
      System.out.println ("Server ready");
    }
     catch (Exception e)
{
    }
  }
}
}
```

3) JSP response Implicit object :-

In JSP, response is an implicit object of type HttpServletResponse. It can be used to add or manipulate response.

4) JSP session implicit object :-

In JSP, session is an implicit object of type HttpSession. It can be used to set, get or remove attribute or to get session information.

5) JSP page implicit object :-

In JSP, page is an implicit object of type Object class. This object is assigned to the reference of autogenerated servlet class.

6) JSP application context implicit object :-

In JSP, application is an implicit object of type ServletContext

7) JSP pagecontext implicit object :-

In JSP, pagecontext is an implicit object of type PageContext class. It can be used to get, set or remove attribute from one of the following scopes :-

→ page → request
→ session → application

Date:
Page:

Q) JSP Cleanup :-

The destruction phase of the JSP lifecycle represents when a JSP is being removed from used by a Container. ~~The JSP de~~

The JSPDestroy() method is the JSP equivalent of the destroy method for servlets.

Date:
Page:

# Define JSP Implicit objects in detail with example.

→ JSP Implicit objects are the Java objects that the JSP container makes available to developers in each page and developer can call them directly without being explicitly declared.

There are 9 JSP Implicit objects :-

1. out
2. request
3. response
4. session
5. page
6. application context
7. page context
8. exception
9. config

1 JSP out implicit object :-

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter.

Eg :-

```
<html>
<body>
    <% out.print("Statement"); %>
</body>
</html>
```

2) JSP request Implicit object :-

The JSP request is an implicit object of type HttpServletRequest. Each time a client requests a page the JSP engine creates a new object to represent that request.

Date: .....
Page: .....

→ All static content (HTML, CSS, JavaScript) is converted into 'out.println()' statements.

→ All JSP elements (<%. %>, <%.= %>, <%.@ %.>, etc) are translated into Java code.

2) Compiling the Servlet :-

→ The translated Java servlet source file (demo_jsp.java) is compiled into a servlet class (demo_jsp.class).

→ This class file is loaded into the web container.

3) Class Loading :-

The compiled .class file is loaded into memory by the servlet container.

4) Instantiation :-

An object of the compiled servlet is created.

5) Initialization :-

The jspInit() method is invoked once when the servlet is initialized.

7) Request processing :-

The _jspService() method is called for every client request. It processes dynamic content and generates HTML outputs.

Date: .....
Page: .....

8) JSP exception Implicit Object :-

In JSP, exception is an implicit object of type java.lang.Throwable class. It can be used to print exception. But it can only be used in error pages.

9) JSP Config Implicit Object :-

In JSP, config is an implicit object of type ServletConfig. It can be used to get initialization parameter for a particular JSP page.

Date:
Page:

8) Destruction :-

When the JSP is no longer needed, the _JSPDestroy() method is called to clean up the resources.

Date:
Page:

Q. Write a client / server RMI program to calculate Simple Interest.

→

Step1: Create a Interface by extending Remote Interface.

```
import java.rmi.Remote;

public interface SimpleInterest extends Remote {
public double calculateInterest (double p, double r,
        double t) throws RemoteException;
}
```

Step2: Implement the Interface you created by extending UnicastRemoteObject

```
public class SimpleInterestImpl extends UnicastRemoteObject
    implements SimpleInterest {
public SimpleInterestImpl () throws RemoteException {
  super(0);
}
@Override
public double calculateInterest (double p, double r,
      double t) throws RemoteException {

  return (p*r*t)/100;
}
}
```

Date:
Page:

# Define JSP. Explain the lifecycle of JSP in detail.

→ JSP ( Java server page ) is a server-side technology that creates dynamic web applications. It allows developers to embed java code directly into HTML or XML pages and makes web development more efficient.

A JSP lifecycle is defined as the process from its creation till the destruction.

* Different phases of JSP life cycle :-

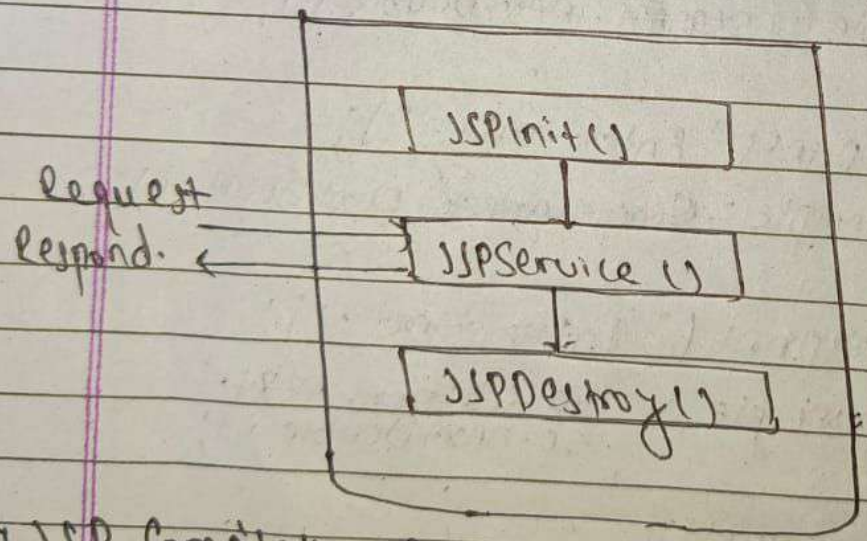



fig :- JSP life cycle

1) JSP Compilation :-

When the browser asks for a JSP, JSP engine first checks whether it need to compile the page or not. If the JSP is last compiled or the recent modification is done in JSP, then the JSP engine compiles the page.

Step3:- Create Server program

```
public class HelloServer {
public static void main (String... args) {
  try {

    Registry r = LocateRegistry.CreateRegistry (1099);
    r.rebind ("greeting", new HelloImpl ());
    System.out.println ("ready");
  }
  Catch (Exception e) { }
}
```

Step4:- Create a client program

```
public class Helloclient {
public static void main (String... args) {
  try {

  Hello h = (Hello) Naming.lookup ("localhost/greeting");
  Registry r = LocateRegistry.getRegistry ("ip/host", port);
  Hello h = (Hello) r.lookup ("greeting");
  String reply = h.greeting ();
  System.out.println (reply);
    }
    Catch (Exception e) { }
  }
}
```

# JSP Compilation Process :-

The JSP Compilation process involves multiple steps that converts a JSP file into a servlet and execute it in a web container.

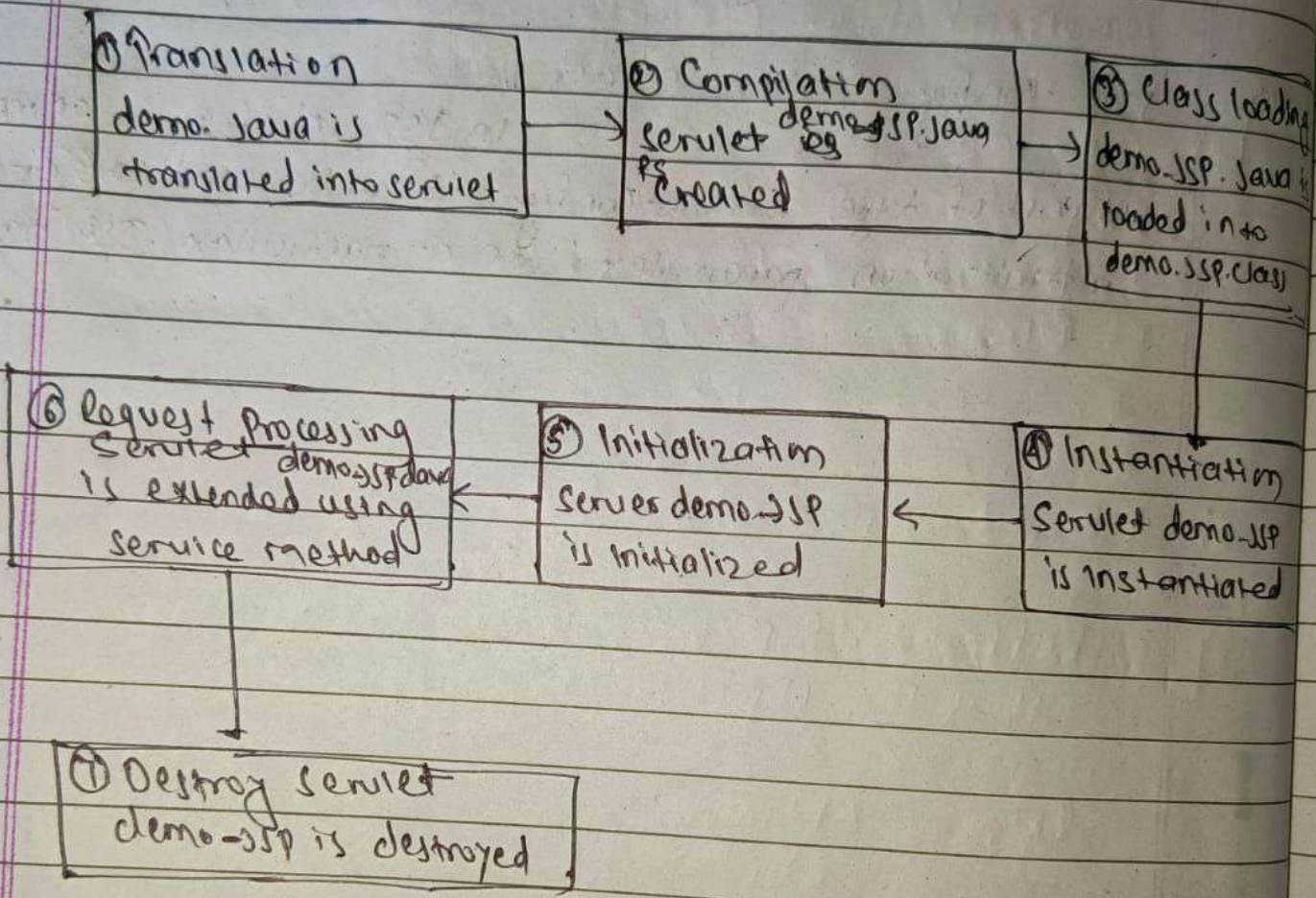



fig :- JSP Compilation process.

Following steps explain the JSP compilation process :-

1) Turning JSP into a servlet :-

→ The JSP page (eg: demo.jsp) is translated into a Java servlet source file (eg demo_jsp.java).

# Define JSP. Explain different types of syntax in details with example.

→

(1) Scriplet ( <%- ---- %> ).
(2) Expression ( <%= exp %> ).
(3) Declaration ( <%! ---- %>.
(4) Directive ( <%@ Import page = "java.sql.*" %> ).

(1) JSP element declaration :-

The declaration tag allows us to define variables and methods. The variables and methods are accessible from scriplets and expression on the same page.

```
<%!
    Delalation
%>.
```

(2) JSP Expression :-

We can access variable and defined in declaration with expression. Expression are embedded directly into HTML. The web browser will display the value of the expression in place of the tag.

Syntax :- <%= Expression %>.

<%

String jdbcUrl = "jdbc:mysql://localhost:3306/db_name";
String dbUser = "root";
String dbPassword = "";

String name = request.getParameter("txt_name");
String email = request.getParameter("txt_gmail");
String feedback = request.getParameter("message");

Connection conn = null;
PreparedStatement pstmt = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
    conn = DriverManager.getConnection("jdbc:mysql://localhost:
            3306/db_name", "root", "");

String sql = "Insert into tbl_name( name, email, feedback)
        values (?,?,?)";

pstmt = conn.prepareStatement(sql);
pstmt.setString(1, name);
pstmt.setString(2, email);
pstmt.setString(3, message);

int rows = pstmt.executeUpdate();
} catch (Exception e) {

}
    con.close();

}
finally {
}

3) JSP scriptlet :-

It allows writing java code inside a JSP page.

Syntax :- <%
java code
%>.

4) JSP Directive :-

JSP Directives instruct the compiler how to process a JSP program.

Syntax :-

<%@ page import = "java.util.*, java.sql.*" %>.

<%

String jdbcUrl = "jdbc:mysql://localhost:3306/db_name";
String dbUser = "root";
String dbPassword = "";

String name = request.getParameter("txt_name");
String email = request.getParameter("txt_gmail");
String feedback = request.getParameter("message");

Connection conn = null;
PreparedStatement pstmt = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
    conn = DriverManager.getConnection("jdbc:mysql://localhost:
            3306/db_name", "root", "");

String sql = "Insert into tbl_name( name, email, feedback)
        values (?,?,?)";

pstmt = conn.prepareStatement(sql);
pstmt.setString(1, name);
pstmt.setString(2, email);
pstmt.setString(3, message);

int rows = pstmt.executeUpdate();
} catch (Exception e) {

}
    con.close();

}
finally {
}

Date:
Page:

# Write down simple JSP page to login and check the credential of valid set session otherwise print error message to the webpage.

→

```
<!DOCTYPE html>
<html>
<head>
<title> login page </title>
</head>
<body>
<form method = "Post" action = "authenticate.JSP">
<label> Name : </label>
<Input type = "text" name = "txt_name" placeholder
: 'Username" /><br/>
<label> Password : </label>
<Input type : "Password" name = "txt_password"
placeholder = " password " /> </br>
<button type = "submit" Login </button> <br/>

<a href : " Signup.JSP" > Don't have account, click here
</a>.

</form>
</body>
</html>
```

Date: .....
Page: .....

# Create feedback form and save to database using JSP.

→

```
<!DOCTYPE html>
<html>
<head>
<title> Feedback Form </title>
</head>
<body>
<h1> Feedback Form </h1>

<form method = "post" >
<label> Name : </label>
<input type = "text" name = "txt_name" required placeholder = "Enter your name" /> <br/>.
<label> Email : </label>
<input type = "text" name = "txt_gmail" required
placeholder = "Enter your email " /> <br/>.
<label> Feedback : </label>
<input type = "text" name = "feedback" required
placeholder = "Enter feedback " /> <br/>.

<button type = "submit" > Submit </button>.
</form>
</body>
</html>.
```

Date:
Page:

Authenticate.jsp

```
<%
    String name = request.getParameter("txt_name");
    String pass = request.getParameter("txt_password");

if(name.equals("Nasir") && pass.equals("123"))
{
    session.setAttribute("User", name);
    response.sendRedirect("Dashboard.jsp");
}
else {
Out.println("Error");
}
%>

Out.println("<a href:"login.jsp"> Error </a> ");
}
%>.
```

CHAPTER:- 4

Date:
Page:

# ~~Define Java Bean~~

# Define EJB. Explain the types of EJB in details.

→ EJB (Enterprise JavaBean) is the specification provided by Sun Microsystem to develop secured, robust and scalable distributed applications.

To Run EJB application we need an application server (EJB Container) such as JBoss, Glassfish etc.

Types of EJB :-

There are three types of EJB :-

1) Session Bean (SB) :-

Session Bean contains business logic that can be invoked by local, remote or webservice client. It's 2 type are:

a) Stateless SB :- It Doesnot maintain state of the client betn multiple method calls.

b) Stateful SB :-

Maintains state of client across multiple request.

c) Singleton SB :-

It is shared betn client and support concurrent access.

Date: .....
Page: .....

Message.JSP

```
<!DOCTYPE html>
<html>
  <head>
    <title> Message Send </title>
  </head>
  <body>
    <h3> Compose Message </h3>
    <form method = "post" action = "send.JSP ">
    <label > Email : </label> ~~place~~
    <input type = "text" name = "email" /> <br />
    <lable > Name : </label>
    <input type = "text" name = "name" /> <br />
    <label> Message : </label>
     <text area name = "message">
      </text area> <br />.
      e'
    <button type = "submit "> Send </button>.
     </form>
     </body>
    </html>.
```

Date:
Page:

[send.jsp]

```
<!DOCTYPE html>
<html>
  <head>
    <title> sending </title>
    </head>
<body>
 <JSP: UseBean Id="msg" scope="request" class=
          "messageBean" />.

<JSP: setProperty property="*" name="msg" />.
<JSP: forward

 </body>
 </html>.
```

Date:
Page:

# Define JavaBean. Write simple JavaBean and access that bean using useBean JSP tag in webpage.

→ Java Beans are introduced in 1996 by ~~Sunn~~ Sunmicrosystem and defined as, "A JavaBean is a reusable platform independent component that can be manipulated visually in a building tool.

# Use Bean Concept :-

```
public class MessageBean implements Serializable {
    private String email;
    private String message;
    public MessageBean() {
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getEmail() {
        return email;
    }
    public void setMessage(String msg) {
        this.message = msg;
    }
    public String getMessage() {
        return message;
    }
}
```

# What are the rules to create JavaBean? Write down simple JavaBean having at least 3 properties.

→ Rules to create JavaBean :-

→ A java bean should be public
→ A javabean should have no argument default constructor
→ It should implement Serializable interface.

```
public class Person implements Serializable {
    private String name;
    private int age;

public Person() { }

Public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}

public int age() { getage() {
    return age;
}
public void setname(int age) {
    this.age = age;
}

}.
```

Date: .....
Page: .....

2) Message Driven Bean :-
Like Session Bean, it contains the business logic but it is invoked by passing message.

3) Entity Bean :-
It encapsulates the state that can be persisted in the database. It is depricated. Now, it is replaced with JPA ( Java persistent API ).

When to use EJB :-

1) Application needs Remote Access.
2) Application needs to be scalable.
3) Application needs encapsulated business logic.

Advantages :-

→ EJB Components can be reused in diff. applications, making it easier to build new software without starting from scratch.

→ EJB provides built-in system services like transaction management and security, so developers can focus on business logic instead of handling complex-level tasks.

Disadvantages :-

→ Requires application server.
→ Complex to understand and develop EJB applications.

Date:
Page:

1) EJB server provider :-

→ Provides an application framework in which to run EJB container.

→ implements and provide access to a JNDI-compatible naming service

2) EJB Container provider :-

→ It supplies runtime classes that provide the required services to the EJB Bean instances.

3) EJB provider :-

→ Responsible for coding the business logic into server side components.

→ Should also have knowledge of the EJB specification.

4) EJB deployer :-

→ Have an in-depth understanding of the characteristics of the particular run-time server and environment such as database type, I/O etc.

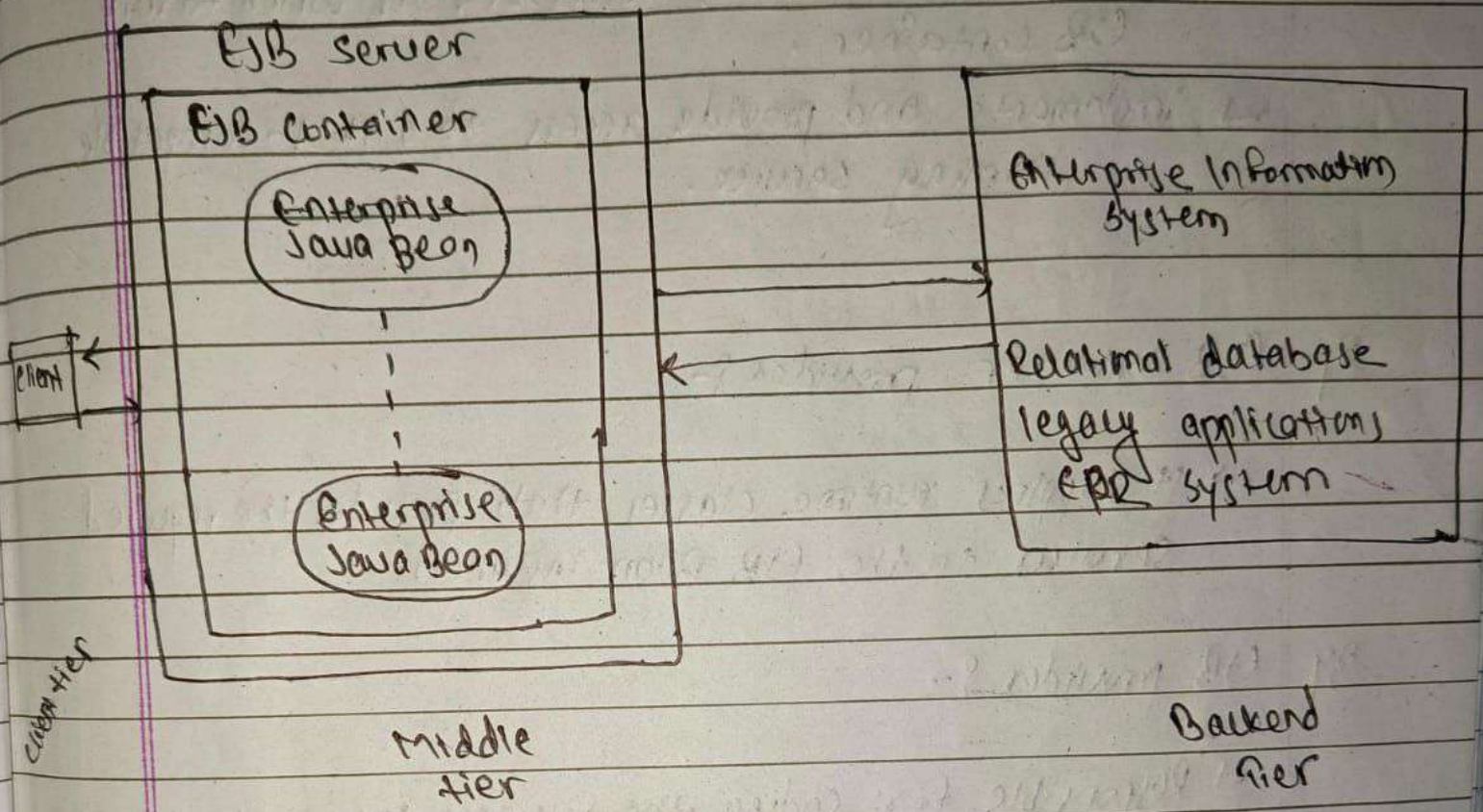
5) Application assembler :-

→ Writes the client applications using pre-built EJB components, Configuration Configuring and customizing them for the assembled application.

Date: .....
Page: .....

# Define EJB. Explain with Architectural diagram.

→




There are different components used in enterprise JavaBean architecture. Generally EJB is used for largescale distributed application which uses :-

→ EJB server :-
→ EJB Container
→ Enterprise Java Bean
→ EJB Client.
→ Other auxillary services like JNDI, JTS etc.
→ EJB server provider
→ EJB Container provider
→ EJB provider
→ EJB deployer
→ Application assembler.

Date: .....
Page: .....

6) EJB Server :-
→ Provides the execution ~~him~~ environment for EJB Container.

→ Provides system services for EJB Container.

7) EJB Container :-
→ acts as the interface bet^n Enterprise Java Bean and low-level, platform specific functionality that support the Bean - the EJB Server.

8) EJB ~~Bean~~ Clients :-
→ Clients interacts with Enterprise Bean instances through proxy objects. Commonly, the communication is based on IIOP protocol and proxy objects are regular CORBA stubs.

Date:
Page:

The ~~brief~~ brief description of each of the class objects involved in Hibernate application architecture is as follow :-

1) Configuration object :-

The Configuration object is the first Hibernate object we create in any Hibernate application. It is ~~only~~ created only once during application initialization.

The Configuration object provides two key components :-

1) Database Connection.
2) Class Mapping setup.

2) SessionFactory Object :-

Session factory object configures Hibernate for the application using the supplied configuration file and allows for a session object to be instantiated.

It is usually created during application start up and kept for later use. We need one session factory object per database using a seperate configuration file. So, if we are using multiple databases, we would have to create multiple sessionfactory objects.

CHAPTER

Date:
Page:

# Define Hibernate. Explain with Architectural Diagram.

→ Hibernate is essential object relational mapping tool for Java programming language. It offers mapping framework for a domain model to one relational database

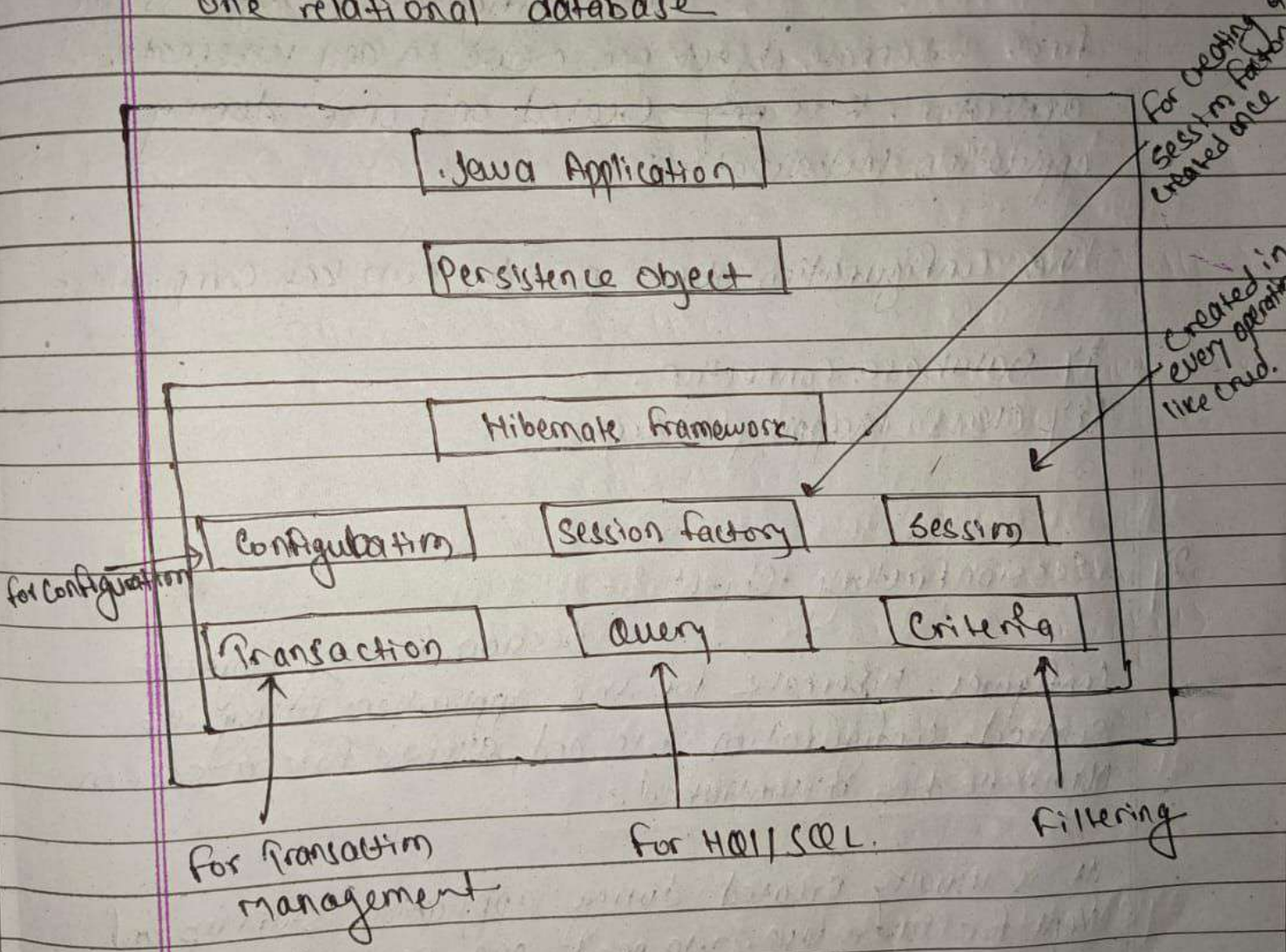



fig :- Hibernate Architecture

Q.11 Explain the Advantages of using Hibernate. Write down Hibernate Annotations with example.

→ Advantages of using Hibernate :-

→ Hibernate enables us to communicate with any database by making tiny alterations in code.

→ MySQL, Db2 or Oracle, Hibernate is DB independent

→ Low risk of data loss and it ~~requires~~ requires less power

Disadvantage of Hibernate :-

→ If power goes off, we could lose all our data.

→ Restarting can be extremely slow.

# Hibernate Annotations :-

1) @Entity Annotation :-

Used for declaring class as an entity (table) in the database

2) @Table Annotation :-

By default, Hibernate uses the class name as the table name. To set a

Date:
Page:

3) Session Object :-

A session is used to get a physical connection with a database. The session object is lightweight and designed to be instantiated each time an interaction is needed with the database.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

4) Transaction object :-

A Transaction represents a unit of of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

5) Query Object :-

Query objects use SQL or Hibernate Query language (HQL) string to retrieve data from the database & create objects.

6) Criteria Object :-

Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Custom table name, we use
@Table (name = "table_name").

Eg :-

```
@Entity
@Table (name = "users_data")
public class user {
    ----
    ----
}
```

3) @Column annotation :-
It is used to define Column names and constraints.

Eg :-

```
@Column (name = "first_name", length = 50,
         nullable = false)
private String name;
```

4) @Id and @GeneratedValue :-

@Id → declares primary key.
@GeneratedValue :-
Defines how the primary key is generated.

Eg :-

```
@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
private int id;
```

Date: .....
Page: .....

# Define the use of application.properties files in Spring Boot Application? Explain with example.

→ MySQL Database Configuration &
eg:- ~~spring.datasource~~

spring.datasource.driver-class-name = com.mysql.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/Crudapp
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = create
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.~~MySQL5Di~~ MySQL.Dialect

The application.properties file is used for external configuration in Spring Boot applications. It allows developers to easily configure various aspects of the Spring Boot application, such as:

1) Database connection
2) Security setting.
3) Several port and context path.
4) Custom application properties

Spring Boot automatically loads the application.properties file located in the src/main/resources directory & uses the properties defined there to configure the application.

5) @Transient :-

Tells the hibernate, not to add this particular column.

Eg :-

```
@Transient
private String tempPassword;
```

6) @Lob annotation :-

used to store large text or binary data.

7) @OrderBy annotation :-

This annotation tell hibernate to orderBy as we do in SQL.

Eg :- if we need to order by student firstname in ascending order

```
@OrderBy("firstname asc")
```

# Springboot annotation :-

* Core Spring framework Annotations :-

1) @Required :- It applies to the bean setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception BeanInitializationException.

2) @ Autowired :-

This annotation is applied on fields, setter methods and constructors. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

3) @ Configuration :-

The annotation is used on classes which define beans. @Configuration is an analog for xml configuration file - it is configuration using java class.

4) @ Qualifier :-

This annotation is used along with @Autowired annotation. When we need more control of the dependency injection process, @Qualifiers can be used.

Date:
Page:

## Why use application properties?

1) Centralized Configuration :-
Centralizes all the configuration values in one place, making it easier to manage settings for different environment.

2) Environment specific configuration : We can specify different configurations for diff environments.

Date:
Page:

(6) (5) @Bean :-

This annotation is used at the method level. @Bean annotation works with @Configuration to create Spring beans.

(6) (6) @Controller :-

The @Controller annotation is used to indicate the class is spring controller.

This annotation is used on Java classes that play the role of controller in our application. This annotation allows autodetection of component classes in the classpath and auto-registering bean definition for them.

Date:
Page:

## Why use application properties?

1) Centralized Configuration :-
Centralizes all the configuration values in one place, making it easier to manage settings for different environment.

2) Environment specific configuration : We can specify different configurations for diff environments.

CH-1

Date:
Page:

# Define Localization & Internalization with example.

→

Internationalization is the process of designing an application so that it can be adopted to various language, currency, data and time & region with engineering changes. Sometimes the term ~~internation~~ internationalization is abbreviated as (I18N) because there are 18 letters bet^n the first I and last N.

An Internationalized program have following characteristics &-

1) With the addition of localized data, the same executable can run worldwide

2) Support for new language doesn't require recompilation.

3) Culturally dependent data such as date, time and currencies, appears in format that confirms to the end users region & language and so on.

4) It can be localized quickly.

# Localization &-

Localization is the process of adopting s/w for specific reason or language by adding locale specific components and translation text. The term localization is often abbrebated as L10N because there are 10 letters bet^n the L & N.