

JAVA

Spring, Hibernate

BY

VARMA

ameerpetmaterials.blogspot.in

NARESH TECHNOLOGIES

Konark XEROX

Ameerpet Hyderabad
Ph...9949090558

10/9/09
Thursday

140 SPRING WITH HYBERNATE

Rs: 140/-

- Gavin King

Pre-Requirements for Any Project:

I. IDE [Integrated Development Environment]

1.1. eclipse, MyEclipse, [RAD, RSA, WSAD, WID] [IBM], IntelliJ.

1.2. NetBeans

Note: eclipse does not support J2EE Compatability. In eclipse if we required the J2EE Compatability we need to Add the plug-in.

How do you implement the eclipse plug-in?

Eclipse plug-in we can implemented using eclipse only.

How to implement the plug-in project?

New → other → plug-in → project → Next → Next

Project name of the plug-in: → Next → Next

Select →

→ To run the plug-in project.

* right click on the Project Run As → Eclipse Application

* It will open Another eclipse

* We can see a New Icon. If we click on that it will display a pop-up with some message.

→ If we required this eclipse plug-in project in some system. Copy the eclipse plug-in project and Paste in it plug-in folder.

Path: MyEclipse 6.0/eclipse/plugins

→ In the New System, If we start the eclipse the Icon will come automatically.

→ In the above way eclipse team has implemented some of the plug-ins in eclipse. If we required the J2EE compatibility download the plug-ins and keep it in

→ If we are working with MyEclipse, it's having some inbuilt plug-ins for the J2EE technology (or) Compatibility

Note: If we required the Tomcat Compatibility [starting the Server, stopping and deploying the application etc.,] in eclipse. Keep the Tomcat compatibility plug-ins in eclipse.

Workspace:

- It is a group of projects and all those project settings will be there in workspace folder.
- While opening if there is an existing workspace it will open that If there is no existing one then it will create a new workspace
- If once the workspace is created it creates a metadata folder It contains the project settings information.

How to Create a Project?

2. File → New → other.

- 2.1. Core Java Application → Java Project
- 2.2. Servlet and JSP's → Web Project
- 2.3. EJB → EJB Project
- 2.4. Web Services → Web Service Project
- 2.5. To Create ear → Enterprise Application Project

Creating a Java Project:

- * Select the Java Project
- * Give the Project Name: **Projectoo**
- * **Finish**

To Create the classes,

* Right click on the project → New → other.

- 2.1.1. To Create a class → class
- 2.1.2. To Create a Interface → Interface
- 2.1.3. To Create a Package → Package

Note: To identify the required class or interface etc. in the Window apply the filter [Type the required Text]

Purpose of Document Comment:

- While calling a method, if we required some description related to the method we need to implement the document comment.
- It's starts with `/**`
`*`
`*/`
- If it is General comment `/*` while calling the method we don't get any description.
- A sample Java file should contain Copy Right package and author and each method with some description.

```
/*
 * Copy Right
 */
package com.Companyname.projectionname.modulename;
Submodulenamem.actions.NareshAction;
• servlet.NareshServlet;
• exception.NareshException;
• util.NareshUtil;
• helper.NareshHelper;

/*
 * @author Administrator
 */
```

Note: While creating the class Select the main, Select the Generate Comments. Then it comes automatically.

Note: While creating the class, If we want to extend another class (or) If we want to implement any interface Then select Browse (or) Add. Then the default implementation will come automatically.

Note: While creating the project, If we want to change the src (or) If we want to Add the jar files
changing src : →

While giving the project name, click on Project Layer Configuration Default

Adding the jar file: →

After giving the project name, click on Next
→ Library → click on Add External JARs

How to Add the jar files to the project:

1. Create a Lib folder
2. Copy and Paste the jar file in Lib folder
3. right click on the jar file

Build path → Add to Buildpath

→ To remove the jar file inside Referenced Libraries right click on the jarfile

Build path → Remove from Buildpath

How to Add the JRE Compatability:

Later right click on the project → Build path → Configure Build path → Add Library → JRE System Library → Finish.

12/19/09 Saturday How To change The JVM Version:

Right click on the Project → Properties → Java Compiler →

Enable Project Specific Settings → compiler Compliance level : 1.4

SHORTCUTS :

1. While Typing Any class name give Two or Three characters
Ctrl + Space Bar

2. To Delete a Line : Ctrl + D

3. To Organize imports : Ctrl + Shift + O

[It will remove unnecessary imports and it will import the required one].

4. To Apply SingleLine Comment and To Remove the Singleline Comment : Ctrl + /

5. To Apply Block Comment : Ctrl + Shift + / [Forward slash]

To Remove the Block Comment : Ctrl + Shift + \ [Backward slash]

6. To open Any type of file which is there in our project i.e. `src\test\project` : Ctrl + Shift + R

7. To open a Java file which is there in a Jar file
Ctrl + Shift + T

8. To go to a particular Method Ctrl + O

To go to a particular Method in Notepad : Ctrl + G

9. To go to a Particular Line : Ctrl + L

10. To go to a Particular Line in Notepad : Ctrl + G

11. While Analyzing the code some cases we follow Bottom

to Top Approach , Then that time while Selecting a Variable . If you want Highlight all the Variable with some color : Alt + Shift + C - F - L

12. If you want to go to a Method, If we are Calling the method from X class and If the method is there in Y class.

→ Keep the cursor on the Method and F3

or

→ Ctrl+ and Select the Method

13. If we do some method Navigation and After that If we want to Navigate Forward [→] and Backward [←] directions.

Alt + →

Alt + ←

14. If we want to identify the complete hierarchy of a method means what is the Top Level Call for the method and How many places it is called

Select the Method → Right click → open Call Hierarchy
[Ctrl+Alt+H]

15. If we want to identify the immediate call of a method

(or) The references of a method

Select the Method → Right click → References → WorkSpace
[Ctrl+Shift+G]

16. If we want to change the method name:

Go to package Explorer → Right click on the Method.

→ Refactor [Alt+Shift+T] → Rename
(or)

Select the Method → Alt+Shift+R

17. If we want to identify something in a file

Ctrl+F

18. If we want to Search in the Entire project

Ctrl+H [Select File Search]

19. For Minimize and Maximize : $Ctrl + M$
20. To close all the Open files : $Ctrl + Shift + W$
21. To Save all the files at a time : $Ctrl + Shift + S$
22. If we open N number of files, If we want to Navigate between the files : $Ctrl + F6$
23. To Navigate Between the views:
 $Ctrl + F7$ [views Are Console, Search, Package Explorer...]
"View" Means it is a simple window related to some results --- etc.
24. To See all the views : Window \rightarrow ShowView \rightarrow other
25. We have different Perspective, A perspective is a group of Views.
In a perspective the look and feel will be different based on the perspective.
26. To go to a perspectives : $Ctrl + F8$
27. To See all the perspectives :
Window \rightarrow Open Perspective \rightarrow other
28. To Reset the perspective in the Right Top corner :
Right click on the perspective \rightarrow Click on Reset
29. The files we can see in different Explorers Called Package Explorer, Project Explorer, Navigator, ... etc.
In one Explorer we can see All the files and In Another Explorer we Cannot see All the files. Because the Reason is in the Explorer the "Default Filter will be Applied"

30. To change the Filter in the Explorer

Menu: → Select Filters

14/9/09
Monday

31. How to Format the Code: Ctrl+Shift+F

32. What is the basic rule, we need to follow before we Commit the code into the Version Control tool.

* Organize the imports: Ctrl+Shift+O

* Format the code: Ctrl+Shift+F

* Right click on the project → Team → Commit

33. How to Debug a Java Program:

F6 : Statement By Statement [Step over]

F5 : It will go inside a Method [Step ~~into~~ ^{into}]

F7 : It will come out of the method [Step return]

F8 : It will jump from one Break point to Another Break point [Resume]

34. While Debuging, how to see the Variable Value, once we reach the statement

Select the Variable → Right click → Inspect [Ctrl+Shift+I]

35. If i want to see the Variable Value Continuously, once we reach the statement

Select the Variable → Right click → Watch

We can see the Variable in Expression Window.

Note: To remove the Breakpoint (or) Expression (or) To Disable the Breakpoint

Go to the Breakpoints (or) Expressions Window and Select the required options.

36. How to change the Variable Value at run time, once we reach the statement.

Go to Variables Window → change the value.

Note: If we are not debugging the program, If we want to go inside a method

→ F3 (or) Ctrl+Select the method

How to Create a Web Project:

1. Select the Web project: [File → New → Web project]

Give the Project Name: NareshProject → Finish

2. Configure the Server

Window → preferences → Filter: Tomcat → Select Enable →
Select the Home Directory → OK

Note: If we are getting any problem with the JDK while starting the server, Then change the JDK to the Server JDK

Window → preferences → Filter: Tomcat → SDK → Add

3. Deploying the Application in the Server

* Start the Server

* Rightclick on the Server → Manage Deployment → click on Add → Select the project → Finish → OK

Note: Shortcuts

1. System.out.println(); : Ctrl+Space

2. public static void main(String args[]); : Ctrl+Space

3. If we know the Method Name, Then How To identify from how many places it is called.

Select the Method → Rightclick → Goto (or) click on

15/9/09
Tuesday

How to Define a Variable in DOS Command Prompt.

c:\> Set Variable-name = Variable-value ;

c:\> Echo % Variable-name %

% Variable-name%

c:\> Set int = 8 ;

c:\> set a = 8 ;

c:\> echo %a% ; To Take The Value %a%

→ All the Home directories it should not End with Semicolon(;) .

It takes the Home Directory and it will Append some other path also, It will give Semicolon(;) it will be terminated But it must not terminated.

→ To start the Tomcat start.bat file information uses Catalina.bat to identify the location it takes internally % CATALINA_HOME% \ bin \ catalina.bat

How To Create a Servlet:

→ After creating the Web project,

Right click on the Src → New → Servlet

Package : edu.servlet

Name : NareshServlet01

Then click on **Next** → **Finish**

→ Then it generates Servlet with default methods and web.xml with the default mapping.

→ If we want to change the web.xml mapping, Then

Go to web.xml **design** → rightclick → Select the required tag

How To Create EJB And Web Service Project:

Go to New → Select EJB (or) Web Service Project →

Give the project Name: **NareshEJBProject**, Then **Finish**

How To Create Enterprise Application Project:

→ It is just the Combination (Any) of WAR and JAR. While creating the EAR we can create WEB and EJB projects also. But we never create like this in the project!

Go to New → Enterprise Application Project →

Give the Project Name : NareshEAR

→ Select Web and EJB projects → Then click on Finish.

→ While creating the EAR we can "Select the Existing projects" Also.

1. Create the Webproject
2. Create the EJB Project
3. Create the EAR

Create the EAR →

Give the project Name : NareshEAR

Select Add Existing Web and EJB module Projects → Next →

Select the projects → Finish.

How we will be doing in the projects:

We checkout the projects from the Version Control tool. Then while creating the EAR,

In the Select the Projects [checkedout projects] → Finish

16/9/84
Wednesday

TOAD

Goto help → About

→ It is a "Front End tool" for the DataBase Using TOAD we can Simplify the Execution of Oracle Queries and Creating the Tables, etc.

* Double click on the TOAD, Then

Database : studentdev1 / studentdev2 / studentdev3

[SERVER (SERVICE - NAME)]

User / Schema : student1 [Scott (User Name)]

Password : studentdev1 (Same as your Service - Name)

* Click on Connect

Then it will open SQL Editor.

1. How to Execute a Query :

Select * from student

Write the Query → Select the Query → F7

2. If we are having more than one insert Query and If we want to Execute both at a time.

insert into student (3, 'Narash@it') ;

Select the Query → click on Execute as a Script

3. To see all the tables or views or procedures or functions etc.,

→ Go to Schema Browser

4. To see the Table related information,

Go to Schema Browser → Tables → Select the Table

* To See the Columns → Columns ↕

* To See the Data → data ↕

5. If we know the column name then How to identify the Table

Go to Tools → Object Search →

Give the column Name : Sham

→ Select the Column Names → Search

6. If we are having the Table then How to Draw the ER-Diagram

Go to Tools → ER-Diagram → Drag and Drop the Table

7. To Create the Table (or) Views --- etc,

Go to Create → Select the required option [Table (or) view]

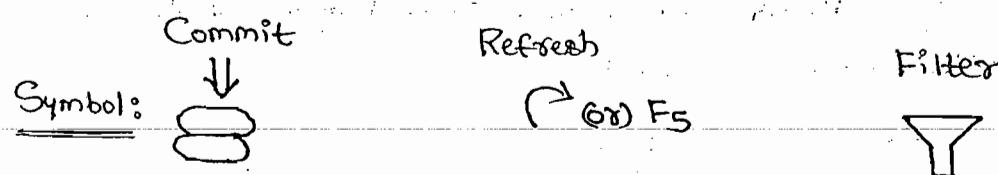
8. To Alter the Table or To Drop the Table or To Do some changes on a Table,

Select the Table → Right click → Do the required option

9. To Know Some information about the data

Tables → Data → Right click on the Data → select the required option.

10. How to Commit and Refresh and Filter the Data



How many Environments will be there in a Project:

3 Environments, They Are

1. DEV → Development

2. UAT → [User Acceptance Test] → Testing

3. PROD → Production

→ In the Development we will be having 3 different DataBase Setups

It's same as for UAT also.

→ But in the Production Environment we will be having one DataBase setup and some projects there will be more than one in production also but it will be Configured with Clustered Environment.

1. To Create a Database

Programs → Oracle-~~oratomego~~ → Configurations and Migration tools → Database Configuration Assistant.

- * Like in the same way we create same Databases. Each Database will be with one Service and to check that

Goto run: **Services.msc** → OK

- * To see the DataBase or To Identify the DataBase

Look into Oracle Services and The Suffix will be the DataBase Name (or) the service will be the DataBase name.

OracleServiceStudent1 → Student1 [DB-Name: student1]

- * Like this there will be 3 services in each Environment

2. After Creating the DataBase Configure the Listener

Oracle-~~oratomego~~ → Configurations and Migration tools → → Net Configuration Assistance

so that we Configure the Listener.

3. Configure the TNSNames Editor

File New Connection → click on TNSNames Editor → right click → Add Service and Configure the service name

Note: Initially we don't get any issue if we don't configure Listener and TNSName, If we are getting any configuration issues then Configure Listener and TNS Name.

Creating a Schema:

Create User student1 identified by student1;

grant Create Session, DBA to student1;

<u>DB/Service Name</u>	<u>Schema/User</u>	<u>Password</u>
Student1/213...	→ Student1	→ Student1
Dev1/2/3 ...	→ Student1	→ Student1
UAT1/2/3 ...	→ Student1	→ Password might be different

student1: Stu is related to DataBase and Dev1 is related to Environment. So, we have one DataBase with 6 Different Environments [Dev1/2/3 and UAT1/2/3].

Note:

Step1: Create schema(user)

Create User student1 identified by studentdev1;

Step2: Give the permissions.

Grant Create Session, DBA to Student1;

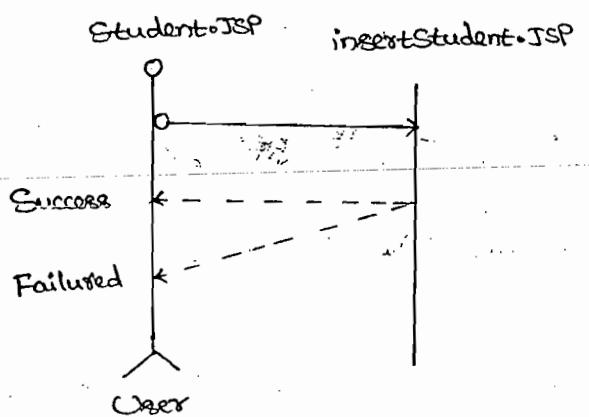
Step3: Connect to the Studentdev1 Schema

Connect Student1 / Studentdev1;

17/9/09
Thursday

JSP To JSP Communication:

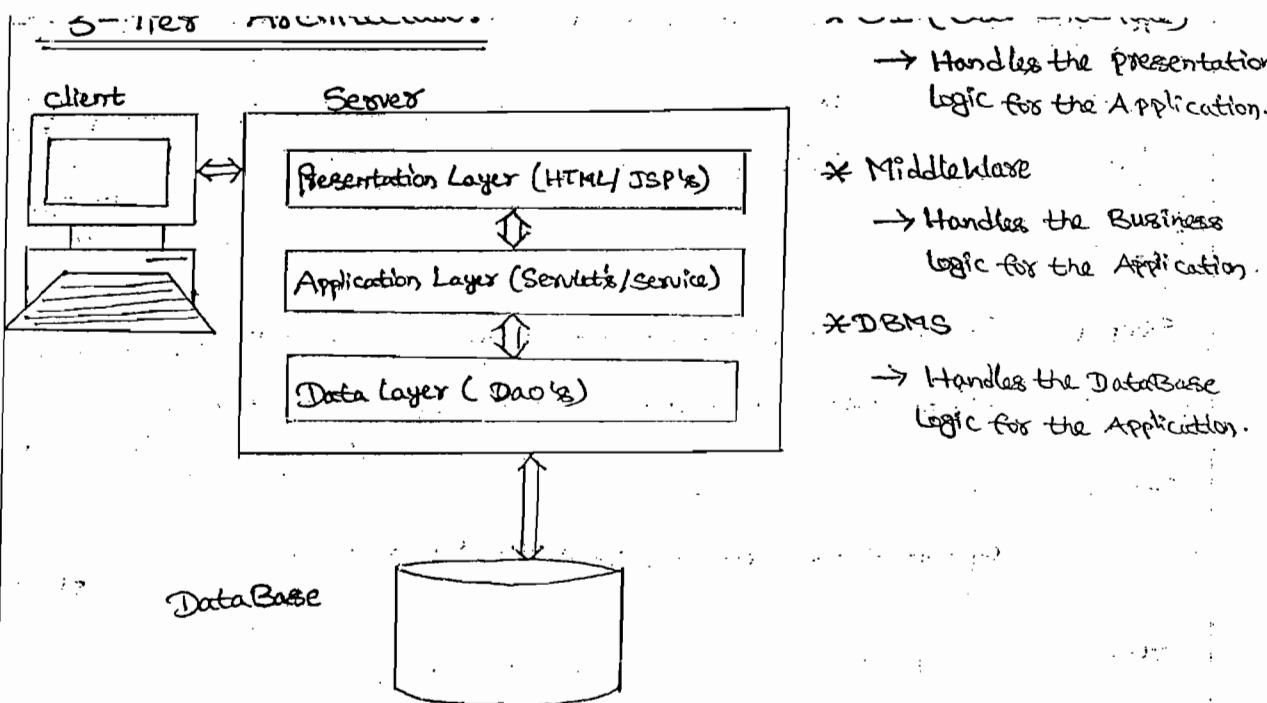
Project Name: JSP-JSPAPP



```
<% JDBC
if(Success)
    rd.forward("success-JSP");
else
    rd.forward("failed-JSP");
%>
```

→ We never implement Single line of JDBC Logic also Inside JSP's.

→ JSP's are for only "Presentation purpose" and the presentation logic also implemented using "JSP tags".



→ For the Presentation purpose we can use Velocity instead of JSP's.
Because Performance wise Velocity is Big faster than JSP's.

→ Servlet Contains Controlling logic sometimes it Contains Business logic and sometimes the Business logic will be there Service class (or) sometimes the Business logic will be there in Helper classes.

→ Services classes Contains the Transaction logic

→ Instead of Services classes we might be writing Manager classes.

→ Service (or) Manager classes Contains the Transaction logic

→ DAO classes Contains JDBC logic (or) ORM [Object Relational Model] logic

JSP To Servlet Communication:

→ We never write the JDBC logic inside Servlet.

→ We never pass the DataBase details in the form of Init Parameter because,

→ Handles the presentation logic for the Application.

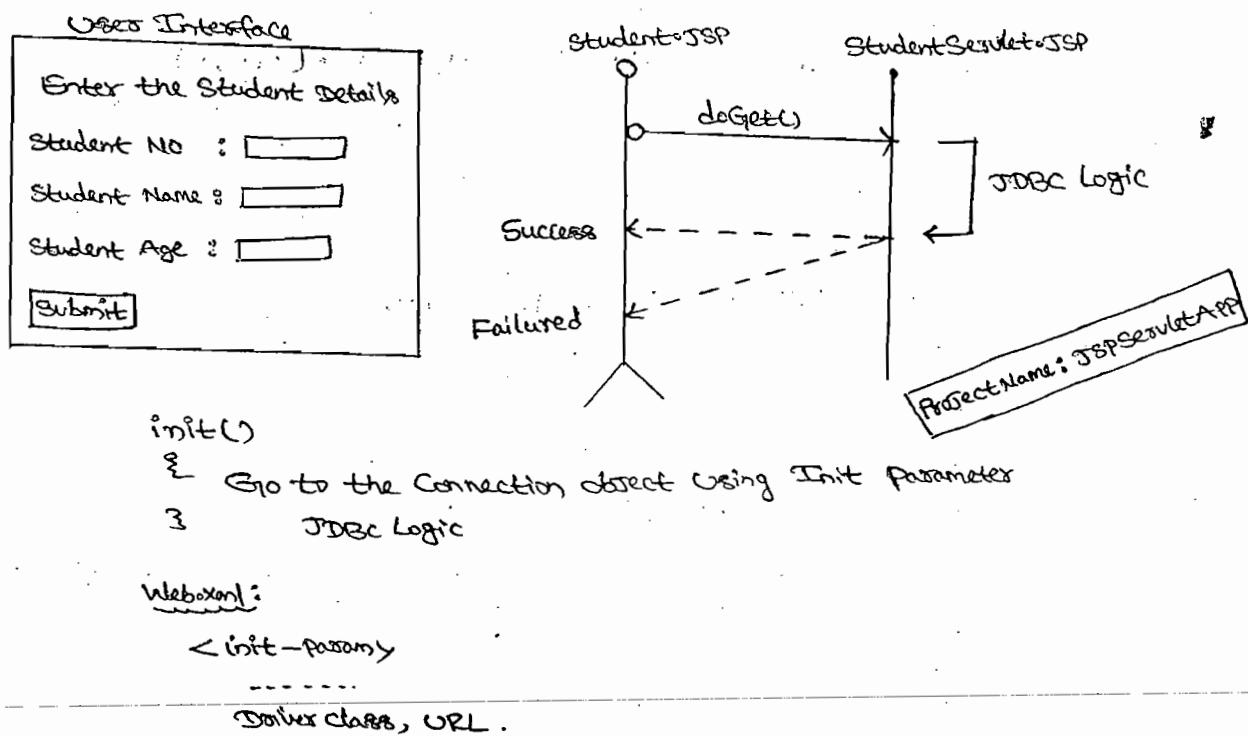
* MiddleWare

→ Handles the Business logic for the Application.

* DBMS

→ Handles the DataBase logic for the Application.

1. If there are "n" number of Servlets we need to pass "n" number of times.
2. We are getting the Connection object inside Init method. Then how to close the Connection object.
3. If we close the Connection object, then how to get it once again Because the logic is there inside Init method.



How to Read a File [XML file (or) Properties file] :

```

Properties properties = new Properties();
properties.load(new FileInputStream("f:/..../edu/config/naresh.properties"));

```

→ In the above code we are hard coding the file location but we should not hard code the file location in the project always we need to add a file using Class Loader concept

```

Properties properties = new Properties();
properties.load(DBUtil.class.getClassLoader().getResourceAsStream(
    ("edu/config/dbDetails.properties")));

```

DBUtil class :

→ It creates the class object and we are calling `getClassLoader()`
Then it takes till "F:/.....classes" directory → Then we are
Appending `edu/config/...` Then it takes correct file location.

To Get the Connection object we write,

```
Class.forName(driverClass);
```

```
Connection con = DriverManager.getConnection(..., ...);
```

→ The Two Lines of code is required in all the DAO classes,
so don't write it in all the DAO's move to a separate
class called DBUtil / Resourcehelper / xx

→ Whenever we want to close the connection we required,

```
if (con != null)
```

`con.close();` → Exception logic etc... and this is required
in all the DAO classes, so move to separate class DBUtil / ... etc.

→ DB Connection close logic is required for all the projects, so
Apache people are providing the logic in a common Jar file called
`commons-dbutils.jar`. So, we don't require any Database
connection close logic in the DBUtil ... classes ::.

Syntax: `DbUtils.closeQuietly(connection);`

DBUtil :

→ It is a simple Java class it takes care of Reading the Database
details from Properties file or XML file and Connection logic

dbDetails.properties :

`driverClass = oracle.jdbc.driver.OracleDriver`

`url = jdbc:oracle:thin:@localhost:1521:STUDEV1`

`userName = student1`

`password = STUDEV1`

DBUtils.java :

```
public class DBUtil {
    private static Properties properties = new Properties();
    private static String driverClass;
    private static String url;
    private static String userName;
    private static String password;

    static {
        try {
            properties.load(DBUtil.class.getClassLoader().getResourceAsStream(
                "edu/config/dbDetails.properties"));
            driverClass = properties.getProperty("driverClass");
            url = properties.getProperty("url");
            userName = properties.getProperty("userName");
            password = properties.getProperty("password");
        } catch (Exception e) {
            System.out.println("Exception is :" + e);
        }
    }

    public static Connection getStoConnection() throws Exception {
        Class.forName(driverClass);
        Connection connection = DriverManager.getConnection(url, userName, password);
        return connection;
    }
}
```

How to Identify the Environment :

1. Set the Environment Variable in Server.
2. String environment = System.getenv("env");
3. Based on the environment Read the related information from XML file

DAO class :

- DAO is a Design pattern and the objects will be used to interact with the Database (insert, loading data.etc-),
- "DAO" is Any Java class which contains JDBC Logic.

```

public class StudentDAO {
    public boolean insertStudent(String studentNo, String studentName,
                                String studentAge) {
        boolean status = false;
        Connection connection = null;
        try {
            connection = DBUtil.getStuConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(
                "Insert into STUDENT Values (?, ?, ?)");
            preparedStatement.setInt(1, Integer.parseInt(studentNo));
            preparedStatement.setString(2, studentName);
            preparedStatement.setInt(3, Integer.parseInt(studentAge));
            status = preparedStatement.executeUpdate() > 0 ? true : false;
        } catch (Exception e) {
        }
        finally {
            DBUtil.closeQuietly(connection);
        }
        return status;
    }
}

```

→ We should not pass the primitive Data from Servlet to DAO
 We need to pass in Object Representation.

```

public void doGet(HttpServletRequest request, HttpServletResponse response) {
    String studentNo = request.getParameter("studentNo");
    String studentName = request.getParameter("studentName");
    String studentAge = request.getParameter("studentAge");
    StudentDAO studentDao = new StudentDAO();
    boolean status = studentDao.insertStudent(studentNo, studentName,
                                              studentAge);
}

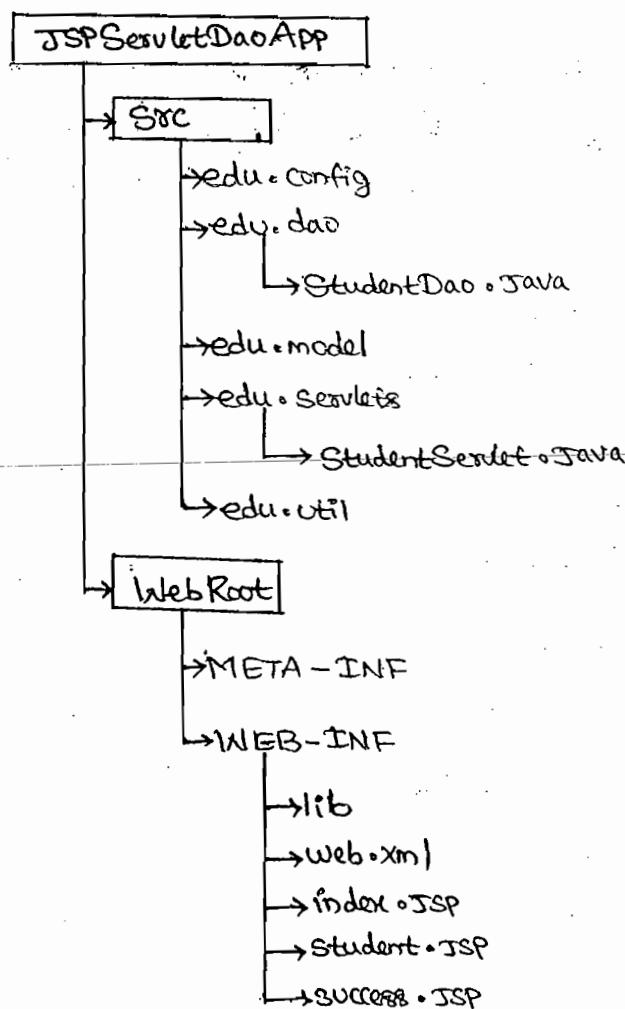
```

→ Here for 3 local variables the Memory allocated and in the DAO class also for 3 local variables the memory will be allocated.

→ Make the 3 variables in the object representation and pass the object reference to DAO, then it allocates memory for 3 variables only.

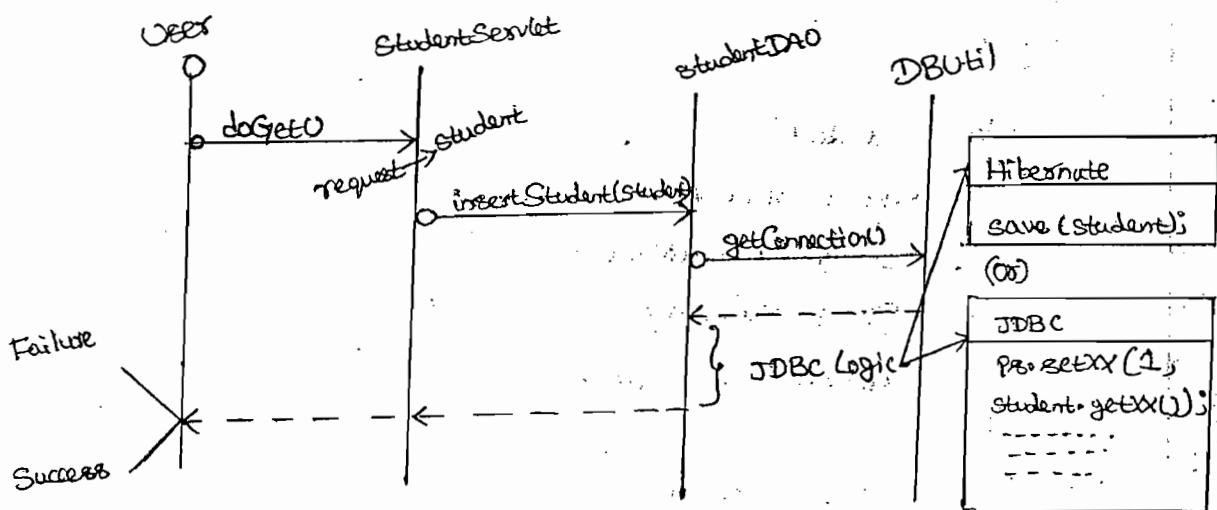
```
class Student  
{  
    String studentNo;  
    String studentName;  
    String studentAge;  
    getters and setters  
}
```

→ The Final Code should be.



21/11/17
Monday

Sequence - 4



Transaction:

→ Do All or Do Nothing [Insert All the records or Don't insert Any record]

→ AutoCommit Value will be true by default which means After Executing a `insertQuery()` it will Execute `commit()` also

1. `insert into student ...`

`ps.executeUpdate();`

 → Insert the Record → Commit ;

2. `insert into student ...`

`ps.executeUpdate();`

 → Insert the Record → Commit ;

→ Don't Apply the Commit Two times

 Apply the Commit only once.

`con.setAutoCommit(false);`

 1. `insert Query`

 2. `insert Query`

`con.commit();`

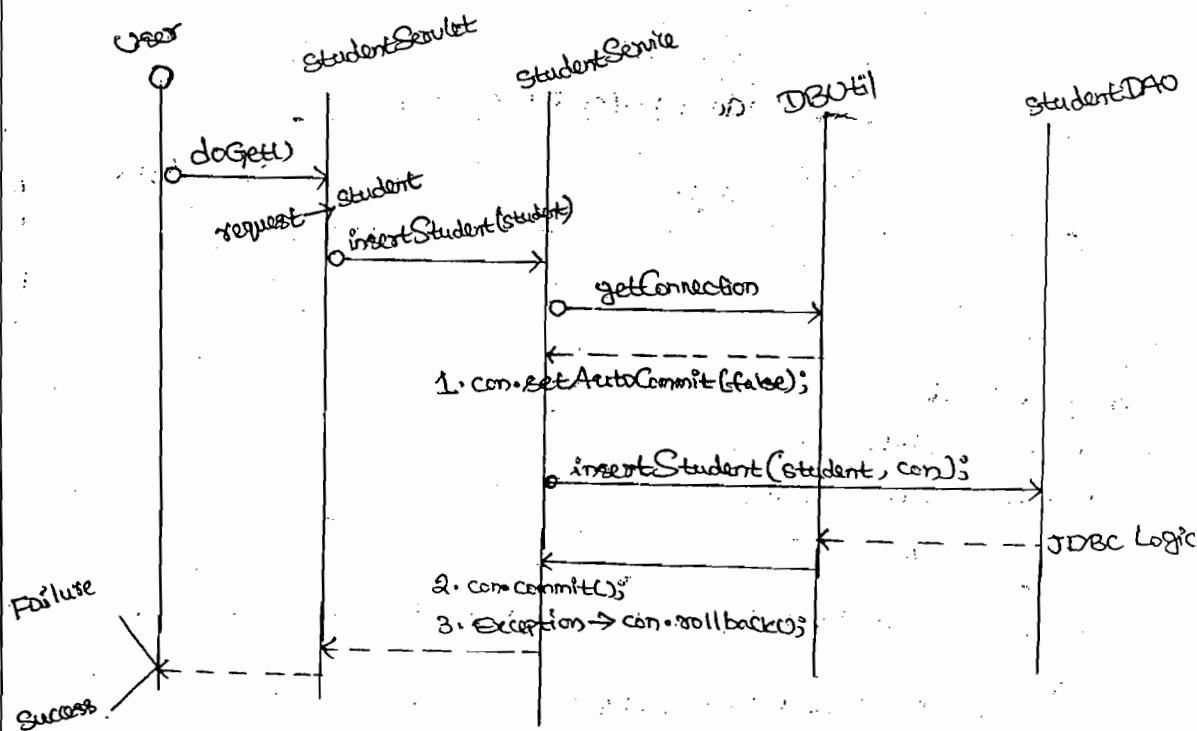
→ If Any exception comes → `con.rollback();`

→ The above 3 Lines of code should be there in Service/Manager Layer.

Service class:

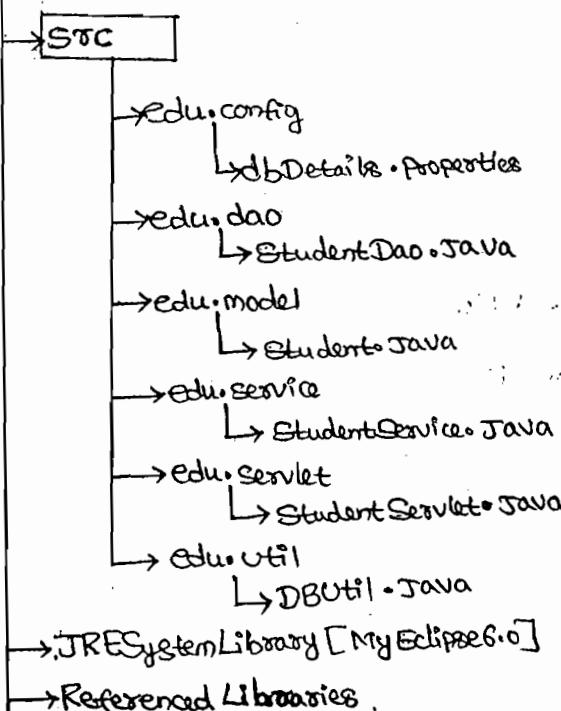
→ It is Any Java class, If it Contains Transaction logic we can call those classes as Service class.

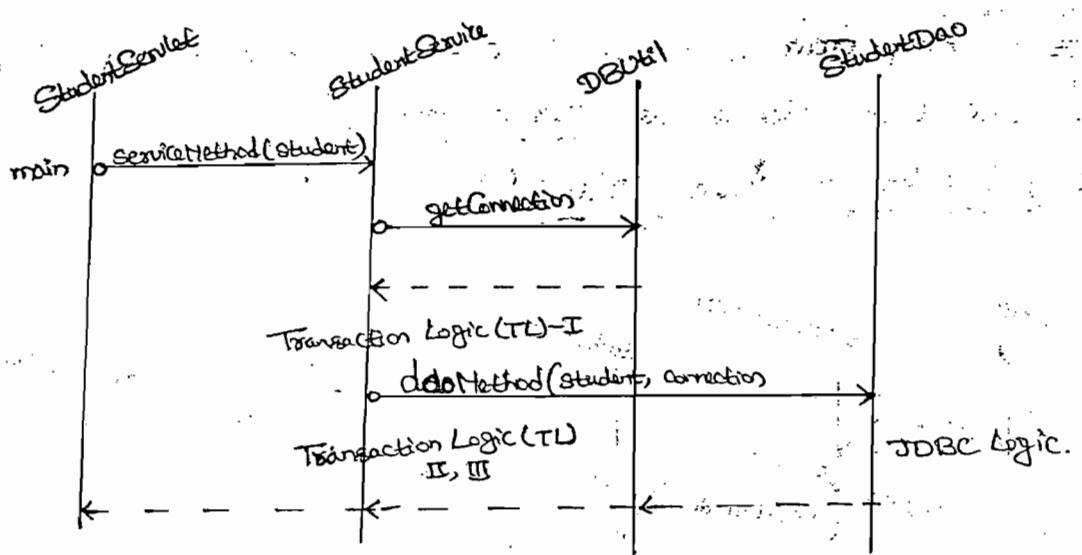
JSP → Servlet → Service → DAO :



22/9/09
Tuesday: Passing The Connection Object to DAO As a Comma(,) , (07) As a Argument's

CoreJavaServiceDao1





Dao class:

```

package edu.dao;
import java.sql.Connection;
import edu.model.Student;
public class StudentDao {
    public void daoMethod (Student student, Connection connection) {
        SOP (" StudentDao . daoMethod . START ");
        // JDBC Logic
        SOP (" JDBC Logic . " + connection );
        // P8 . setXXX ( 1, student . getXXX () );
        SOP (" . StudentDao . daoMethod . END ");
    }
}

```

Student class:

```

package edu.model;
public class Student {
    private String studentNo;
    private String studentName;
    getter and setters.
}

```

StudentService class :

```
public class StudentService
{
    public void serviceMethod ( Student student ) throws SQLException
    {
        SOP ( " StudentService . ServiceMethod . START " );
        Connection connection = null ;
        try
        {
            connection = DBUtil . getStuConnection () ;
            connection . setAutoCommit ( false );
            StudentDao studentDao = new StudentDao ();
            studentDao . daoMethod ( student , connection );
            connection . commit ();
        }
        catch ( Exception e )
        {
            connection . rollback ();
        }
        finally
        {
            DBUtil . closeQuietly ( connection );
        }
        SOP ( " StudentService . ServiceMethod . END " );
    }
}
```

StudentServlet class :

```
public class StudentServlet
{
    // service ( req , res )
    PSVM ( String args [] ) throws SQLException
    {
        SOP ( " StudentServlet . main . START " );
        Student student = new Student ();
        student . setStudentNo ( " 1 " );
        student . setStudentName ( " Narash @ It " );
        StudentService . StudentService = new StudentService ();
        StudentService . serviceMethod ( student );
        SOP ( " StudentServlet . Main . End " );
    }
}
```

Note: We are passing the Connection from service to Dao as a Method argument. It is a wrong Approach because if we are having "N" number of Dao methods we need to pass "N" number of times.

2. 2nd Approach:

- Set the Connection object to the Dao Constructor so that we no need to pass to the Dao Methods.
- Only the Service and Dao class is going to be change.

```
public class StudentService  
{  
    public void serviceMethod (Student student) throws SQLException  
    {  
        System.out.println ("StudentService - serviceMethod - START");  
        Connection connection = null;  
        try  
        {  
            connection = DBUtil.getStuConnection();  
            connection.setAutoCommit(false);  
            StudentDao studentDao = new StudentDao (connection);  
            studentDao.daoMethod (student);  
            connection.commit();  
        }  
        catch (Exception e)  
        {  
            connection.rollback();  
        }  
        finally  
        {  
            DBUtil.closeQuietly (connection);  
        }  
        System.out.println ("StudentService - serviceMethod - END");  
    }  
}
```

StudentDao Class:

```
public class StudentDao  
{  
    private Connection connection;  
    public StudentDao (Connection connection)  
    {  
        this.connection = connection;  
    }  
    public void daoMethod (Student student)  
    {  
        // Logic is the same as old program  
    }  
}
```

Note: If we use two DaoMethods using the Constructor approach in both the Methods same connection object will be used, If we want to use two different connection objects Then better approach is the better one.

→ Once the object is constructed Using constructor approach we cannot change the state . so that we cannot use different connections in different DaoMethods using Setter approach we can change the state of the object . so before calling the DaoMethod set the corresponding connection object and call the method .

StudentService class :

```
public class StudentService
{
    public void serviceMethod(Student student) throws SQLException
    {
        System.out.println("StudentService.serviceMethod.START");
        Connection connection = null;
        try
        {
            connection = DBUtil.getDbConnection();
            connection.setAutoCommit(false);
            StudentDao studentDao = new StudentDao();
            studentDao.setConnection(connection);
            studentDao.daoMethod(student);
            connection.commit();
        }
        catch (Exception e)
        {
            connection.rollback();
        }
        finally
        {
            DBUtil.closeQuietly(connection);
        }
        System.out.println("StudentService.serviceMethod.END");
    }
}
```

StudentDao Class:

```
public class StudentDao
{
    private Connection connection;

    public void setConnection( Connection connection )
    {
        this.connection = connection;
    }

    public void daoMethod( Student student )
    {
        // Logic is same as previous program
    }
}
```

Note: If we are having "N" number of Dao classes "N" number of times we need to write `setConnection()` method and Instead of writing "N" number of times Create a Abstract class and Extend the Abstract class in all the Dao classes.

3. 4th Approach:

→ This will be used in most of the Projects with Factory classes.

23/9/09
Wednesday StudentService class:

```
public class StudentService
{
    public void serviceMethod( Student student ) throws SQLException
    {
        System.out.println("StudentService.serviceMethod - START");

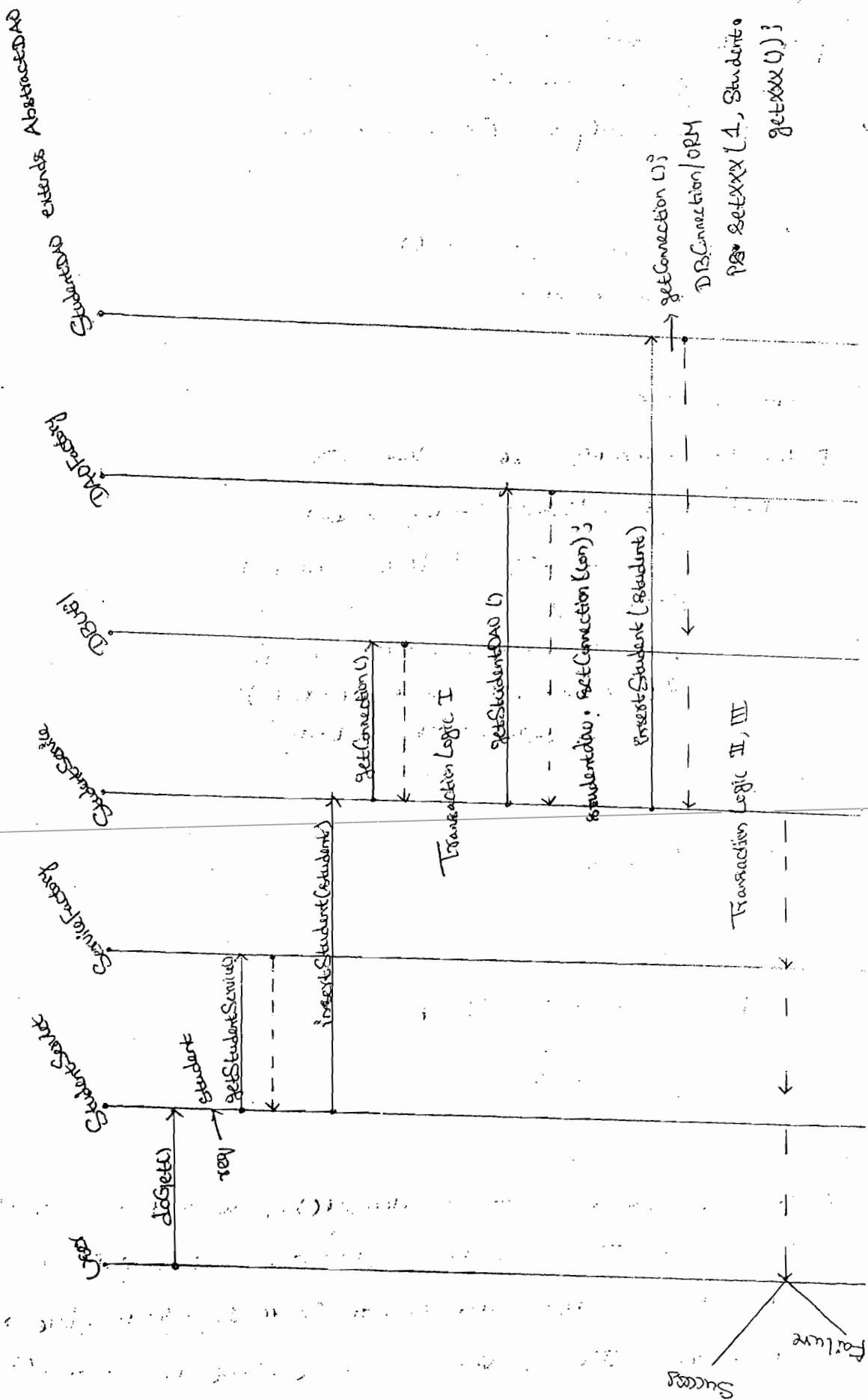
        Connection connection = null;

        try
        {
            connection = DBUtil.getStuConnection();
            connection.setAutoCommit(false);

            StudentDao studentDao = new StudentDao();
            studentDao.daoMethod( student );

            studentDao.setConnection( connection );
            connection.commit();
        }
        catch( Exception e )
        {
            connection.rollback();
        }
        finally
        {
            DBUtil.closeQuietly( connection );
        }

        System.out.println("StudentService.serviceMethod - END");
    }
}
```



```
public class AbstractDao
{
    private Connection connection;
    public void setConnection ( Connection connection )
    {
        this.connection = connection;
    }
    public Connection getConnection()
    {
        return connection;
    }
}
```

StudentDao class:

```
public class StudentDao extends AbstractDao
{
    public void daoMethod ( Student student )
    {
        SOP ( " StudentDao. daoMethod. START " );
        // JDBC Logic
        SOP ( " JDBC Logic. " + getConnection () );
        // ps.setXXX ( 1, student.getXXX () );
        SOP ( " StudentDao. daoMethod. END " );
    }
}
```

Singleton Design Pattern:

```
class Dao
{
    // No Instance Variables ....
    public void daoMethod1 ()
    {
    }
    public void daoMethod2 ()
    {
    }
}
```

→ Whenever we want to daoMethod1(), we need to Create the Dao object. Then only we can call the method, If we want to call from 10 places .we need to Create 10 Objects. We Note have any Instance Variables why should we Create the object 10 times , Let's Make all the methods as a static.

```
public static void main(String args[])
{
    return DaoTwo;
}
```

DaoOne class:

```
package edu.dao;
public class DaoOne
{
    // No Instance Variables
    public void daoMethod1()
    {
        System.out.println("DaoOne. daoMethod1()");
    }

    public void daoMethod2()
    {
        System.out.println("DaoOne. daoMethod2()");
    }
}
```

DaoTwo class:

```
package edu.dao;
public class DaoTwo
{
    // No Instance Variables
    public void daoMethod1()
    {
        System.out.println("DaoTwo. daoMethod1()");
    }

    public void daoMethod2()
    {
        System.out.println("DaoTwo. daoMethod2()");
    }
}
```

MainTest class:

```
package main.edu;
public class MainTest
{
    public static void main(String args[])
    throws Exception
    {
        mainMethod1();
        mainMethod2();
    }
}
```

→ For the objects Memory will be allocated in Heap Area.

$$4 + 4 + \text{defaultSize (8)} = 16$$

What is size for Reference Variable:

Reference → 4 bytes and where it will be allocated

singleton1 = Stack

singleton2 = Method Area

singleton3 = Heap

Local → Stack

Stack / class → Method Area

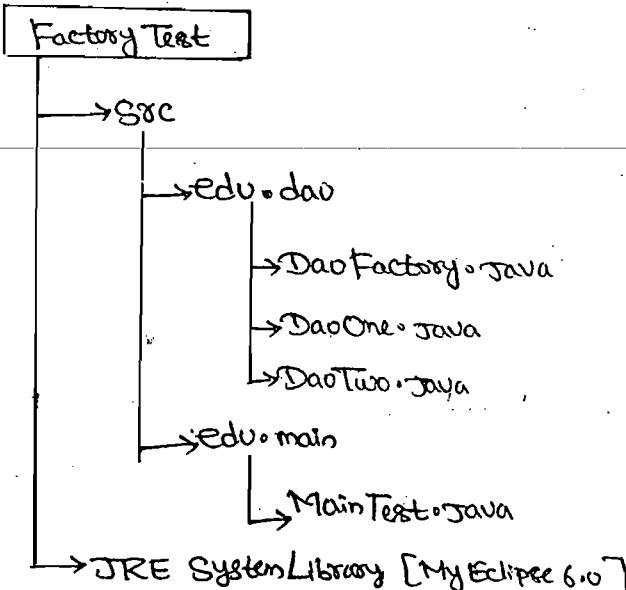
Instance → Heap

Reference → 4 bytes

Object → defaultSize (8) + xxxx

24/9/09
Thursday

Directory Structure:



DaoFactory class:

```
Package edu.dao;
public class DaoFactory
{
    Private static DaoOne daoOne = new DaoOne();
    private static DaoTwo daoTwo = new DaoTwo();
    public static DaoOne getDaoOne()
    {
        ...
    }
}
```

```

public static void mainMethod1()
{
    3 DaoOne daoOne = DaoFactory.getDaoOne();
    daoOne.daoMethod1();
    daoOne.daoMethod2();
    System.out.println("..."+daoOne);
}

```

3

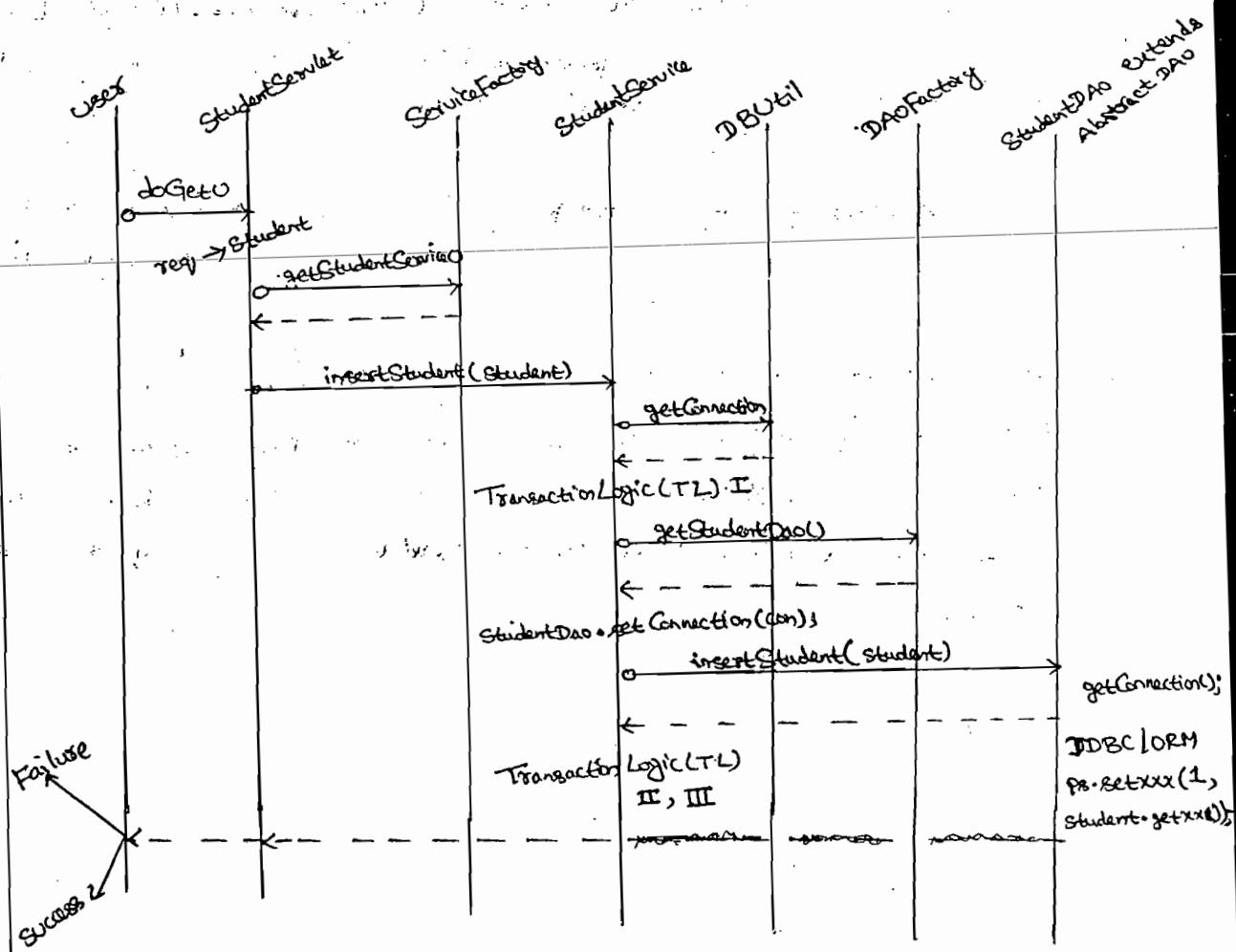
```

public static void mainMethod2()
{
    3 DaoOne daoOne = DaoFactory.getDaoOne();
    daoOne.daoMethod1();
    daoOne.daoMethod2();
    System.out.println("..."+daoOne);
}

```

3

JSP → Servlet → Service → DAO → Factory :



* Set the Environment Variable (In My Computer)

Variable Name : env
Value : Dev1

<DBConfig>

```
<DEV1 user = "student" password = "STUDEV1"  
url = "jdbc:oracle:thin:@localhost:1521:STUDEV1"  
driver = "oracle.jdbc.driver.OracleDriver">  
</DEV1>  
</DBConfig>
```

Step 1 :

```
private static SAXBuilder builder = new SAXBuilder();
```

Step 2 :

```
Document document = builder.build ( MainTest.class.getClassLoader()  
getResourceAsStream ("edu/config/DBConfig.xml"));
```

Step 3 : Element element = document.getRootElement();

Step 4 :

```
SOP ("-driver" + element.getChild ("DEV1").getAttributeValue  
("driver"));
```

Exception Handling :

→ Whenever we get Any Exception in the Dao classes we need to display the Error Pages, To display the Error page we require Request, Response objects, Those will not available in Dao, So Rethrow the Userdefined Exception Back to the Servlet and Display the Error Pages.

→ If we are having 10 Dao classes 10 times, we need to Apply
 Singleton Design pattern for all the Dao classes instead of Applying
 10 times Singleton. Let's Apply Factory pattern which takes
 care of giving the Dao objects.

Factory Design pattern:

→ Factory is a Design pattern and it is responsible to Create a
 object and give the object.

Steps to implement Factory:

1. Create the Object Using static reference

```
private static DaoOne daoOne = new DaoOne();
```

2. Create a static method and return object

```
public static DaoOne getDaoOne()
```

```
{  
    return daoOne;  
}
```

```
3
```

Some Extra Points:

Singleton Design pattern:

class Singleton

```
{
```

```
Singleton singleton1 = new Singleton();
```

```
static Singleton singleton2 = new Singleton();
```

```
private int i, j;
```

```
private static int k;
```

```
void Method1()
```

```
{  
    int x;
```

```
    Singleton singleton3 = new Singleton();
```

```
3
```

→ When the class is loaded, it allocates

4 bytes + default size (8) → Method Area

→ Once the object is created.

```
Singleton singleton = new Singleton();
```

→ If There are "N" No. of Methods "N" No. of times we need to Apply static instead of Applying "N" No. of times. Let's Create class as a Singleton, Then all the methods we no need to declare it as a static. We can leave it as a Normal methods.

Steps to Apply Singleton Design pattern:

1. Create a Static reference and Create a Object

```
private static Dao dao = new Dao();
```

2. Create a private Constructor.

```
private Dao()  
{ }
```

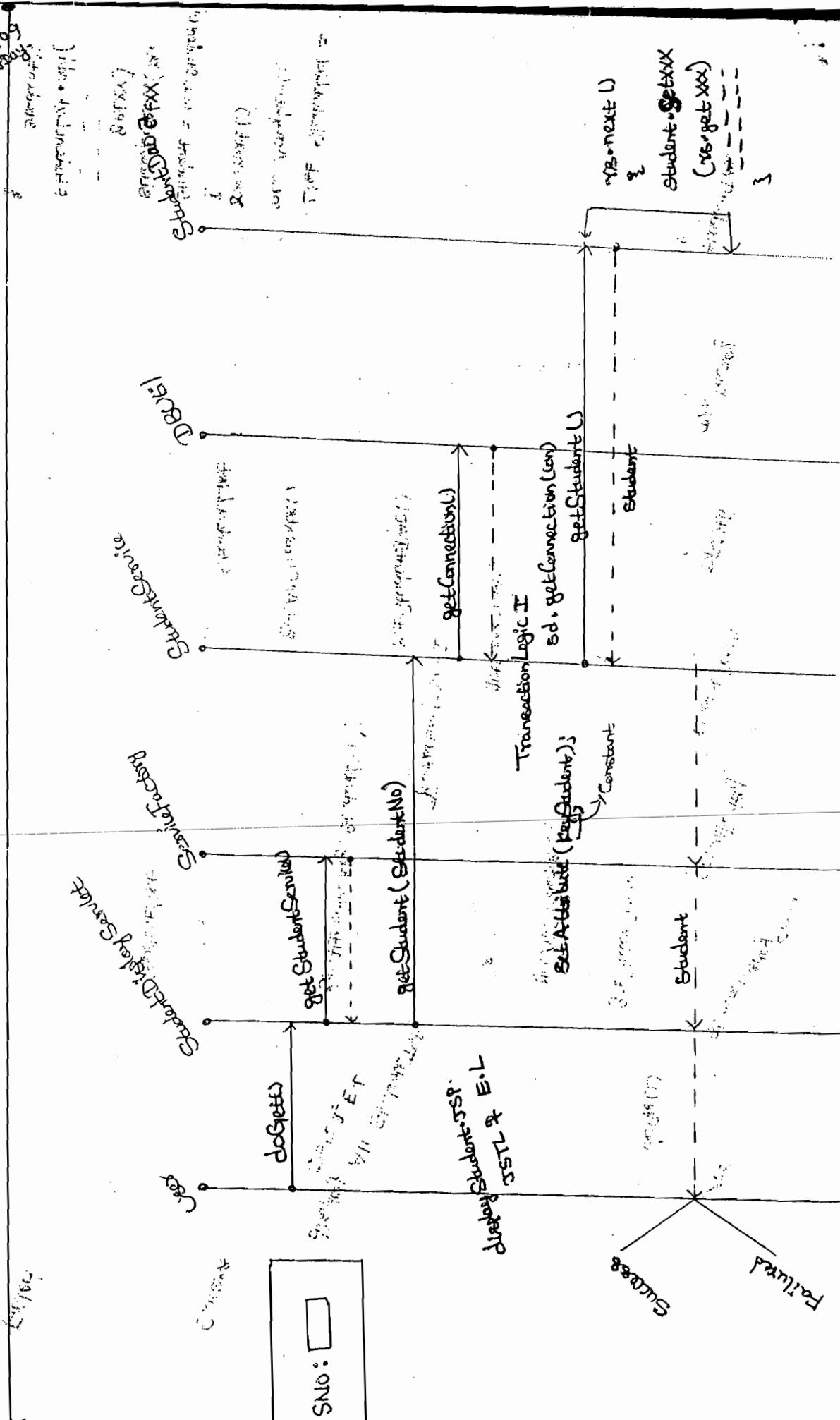
3. Create a static Method and return the object

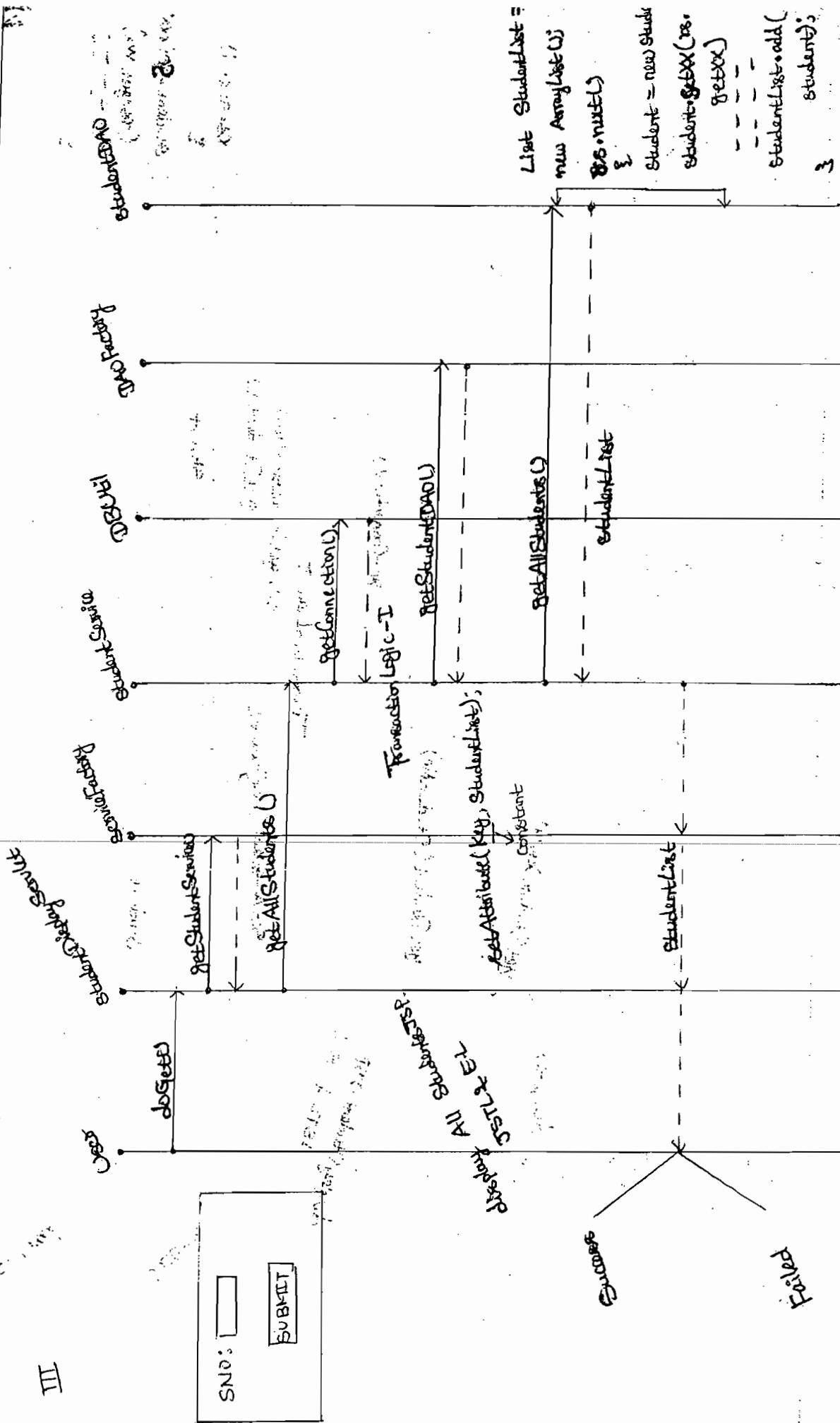
```
public static Dao getInstance()  
{  
    return dao;  
}
```

→ Singleton is a Core Design pattern only one object should be created for JVM.

```
class Dao  
{  
    private static Dao dao = new Dao();  
  
    private Dao()  
    { }  
  
    // No Instance Variables  
  
    public void daoMethod1()  
    { }  
  
    public void daoMethod2()  
    { }  
  
    private static Dao getInstance()  
    {  
        return dao;  
    }  
}
```

26/9/09
Saturday





private static List statesList = null;

List getStatesList()

{

if (statesList != null)

return statesList;

else

{

Hit the DB

get the rs

rs → states

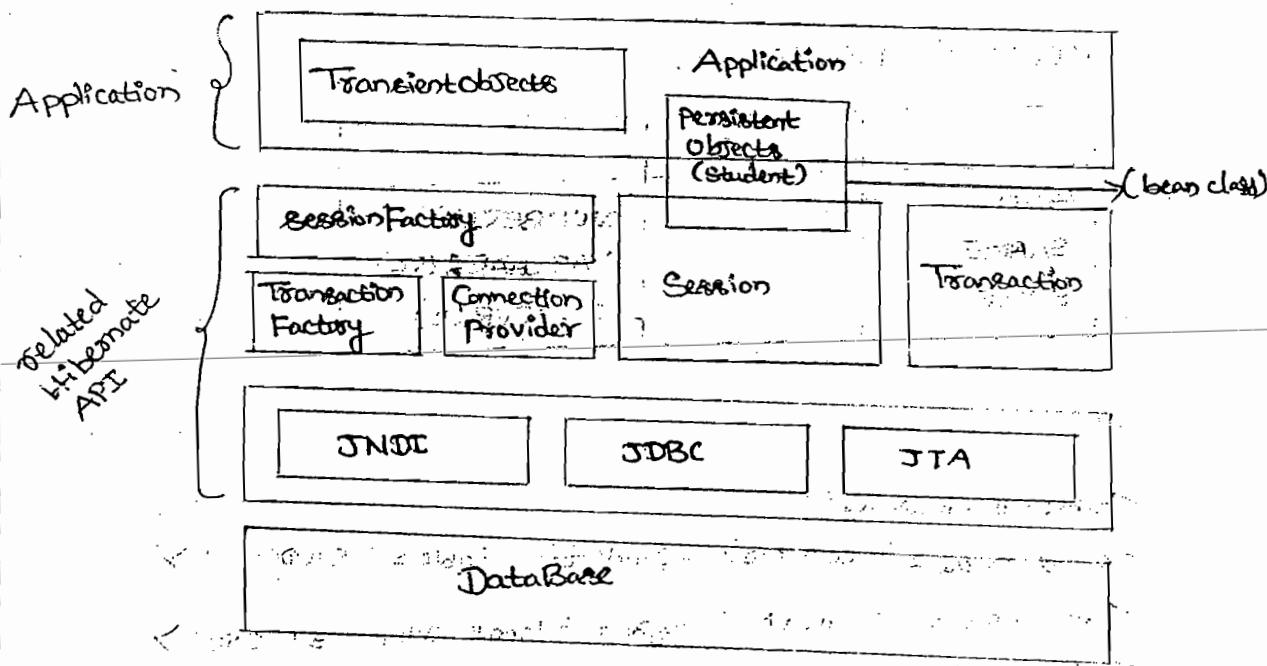
states → stateList

}

3

29/09/09
Tuesday

HIBERNATE ARCHITECTURE



What are the required programs for the Hibernate Sample Application?

JDBC

1. Main program / DAO
2. DBConfig.xml (DB-Details)
3. Student
4. Not Applicable (mapping file)

Hibernate

1. Main program / DAO
2. Hibernate-Cfg.xml (DB-Details)
3. Student
4. student.hbm.xml

1. Spring - Struts / MVC
2. ORM - EJB, XML, webservices
3. Servlet, JSPs
4. Core Java
5. .Net

Object Relational Model (ORM) :

1. Java object :

```
public class Student
{
    private Long studentNo;
    private String studentName;
    private Long studentAge;
}
```

getters & setters

2. RDBMS : STUDENT Table

Name	Null?	Type
SNO		NUMBER(8)
SNAME		VARCHAR(25)
SAGE		NUMBER(8)

3. mapping-File

```
<hibernate-mapping>
<class name = "com.model.Student" table = "STUDENT">
    <id name = "studentNo" type = "long" column = "SNO">
        <generated class = "increment"/>
    <property name = "studentName">
        <column name = "SNAME"/>
    </property>
    <property name = "studentAge">
        <column name = "SAGE"/>
    </property>
</class>
</hibernate-mapping>
```

4. Store the Object in DataBase:

```
Student student = new Student()
student.setStudentNo(1);
student.setStudentName("Naresh");
student.setStudentAge(12);
session.save(student);
```

5. DataBase - Table Data:

SNO	SNAME	SAGE
1	Naresh	12

JDBC:

```
ps.setInt(1, student.getStudentNo());
ps.setString(2, ---);
ps.executeUpdate();
```

1. <forward name = "success" path = "naresh.jsp?param1=value & param2=value" />

&, >, <-----

2. We can call Java code from a Oracle procedure

3. If we execute stored procedure, some times it returns Database cursor, then we need to iterate the cursor.

4. We can invoke the servlet from core Java code using

java.net package class {

[instead of invoking it]

Advantages of Hibernate:

①. Hibernate is an ORM Tool

(i). SAVE

JDBC

```
ps.setXXX(student.getXX());
ps.setXXX(student.getXX());
```

Hibernate

```
session.save(student);
```

(ii) Read

```
List studentList = new ArrayList();  
List studentList = null;  
Student student = null;  
while (rs.next())  
{  
    student = new Student();  
    student.setXX(rs.getXX(1));  
    student.setXX(rs.getXX(2));  
    studentList.add(student);  
}  
return studentList; [multiple records  
Retrive from the DB]
```

(iii).

```
Student student = null;  
if (rs.next())  
{  
    student = new Student();  
    student.setXX(rs.getXX(1));  
    student.setXX(rs.getXX(2));  
}  
return student;
```

[Retrive the single record from
the DataBase]

- * → Using Hibernate, we can store & Retrive an object, (o)
- list of object and from the database, It is not possible
using JDBC.
2. Hibernate is used to persist the objects in the DB.
 3. JDBC is used to store the primitives in the DataBase.
 4. It provides Connection pool.
 5. It provides Two levels of Caching. It provides Transaction manager

② Hibernate Provides Connection pool mechanism:

→ Hibernate provides connection pool mechanism to prepare the pool of connection in the cfg file we need to give an entry.

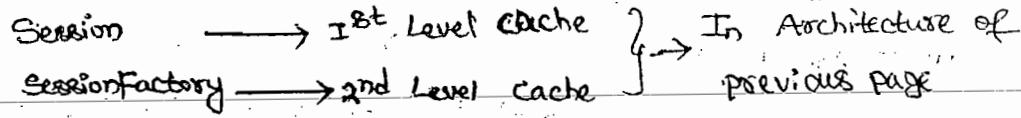
property name = "hibernate.connection.pool-size">10</property>

→ But we never use hibernate provided connection pool in the projects. We use the server provided connection pool.

→ In JDBC, It is not providing any direct connection pool mechanism. If we want we need to contact apache commons Jars (commons-pool.jar) (or) Server.

→ In Hibernate also we can get the connection pool from the Third Party Jars, But we never use it in the projects, most of the cases we get it from server.

③ Hibernate provides Two Levels of Caching [1st Level, 2nd Level]



→ 1st Level Cache will be on top of Session and 2nd Level cache will be on top of SessionFactory.

Cache :

→ If the data is not changed for a particular table [Constant Tables (or) Lookup Tables]

Eg: STATES, ZIPCODE, etc,

→ If we required the data "n" no. of times instead of the database "n" no. of it only once and cache it using static variables.

→ In JDBC + JBoss Seam

```

private static final long REFRESH_INTERVAL = (1000 * 60 * 60); // 1 hour
private static List studentList = new ArrayList();
private static Long timeStamp = null;
if ((timeStamp == null) || (System.currentTimeMillis() - timeStamp
    * longValue() > REFRESH_INTERVAL))
{
    if (timeStamp == null)
        studentList.clear();
    else
    {
        if (studentList.size() > 0)
            return studentList;
        else
            timeStamp = System.currentTimeMillis();
        hit the DB and add the records to list
        studentList.add(...);
    }
}

```

→ With the above logic, we hit the Database each one hour in JDBC projects.

→ In hibernate, we no need to implement any logic, it is just xml-configuration.

30/9/09
Wednesday

(4) It provides the Transaction Management:

JDBC [ServiceLayer]

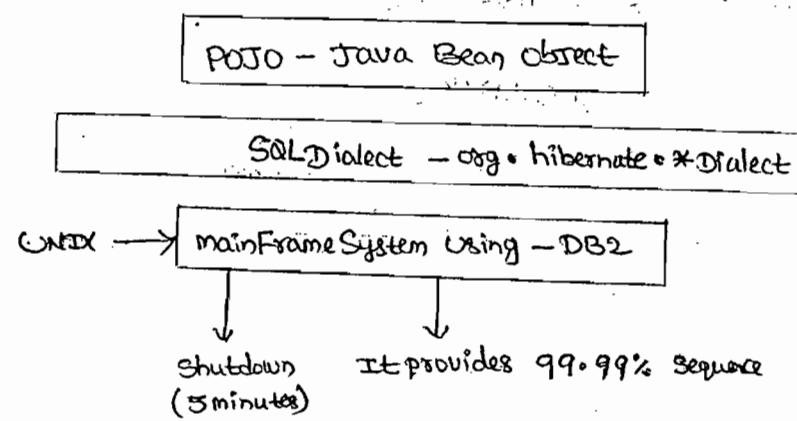
Hibernate

- | | |
|-------------------------------------|---|
| 1. connection.setAutoCommit(false); | 1. Transaction tx = session.beginTransaction(); |
| 2. connection.commit(); | 2. tx.commit(); |
| 3. connection.rollback(); | 3. tx.rollback(); |

Note:

connection.commit();
 It is not must in JDBC. Because by default it will be true but in projects we applying the transaction management.

→ In Hibernate, the default Value will be false, while doing any "CRUD()" [Create, Read, Update & Delete] operations are apply the transaction management and it is must.



⑤ If we are shifting one DataBase to another DataBase
(MySQL to Oracle)

→ In JDBC, we need to change the DAO Layer logic because the datatypes will be different.

→ In Hibernate, we no need to touch the DAO Layer just we need to change the Dialect class in the Configuration file.

DIALECT Class :

→ Dialect is a simple Abstract Java class. Each DataBase there is a Dialect class & all the Dialect classes will extends Dialect.
Dialect classes Contains,

- * What is the Java datatypes
- * What is the Database datatype

→ If we are having our own Database, then we need to implement our own Dialect class, if there is no Dialect class is provided by hibernate.

→ Dialect class represents a Dialect of SQL implemented by a Particular RDBMS. Subclasses implement Hibernate compatibility with different Systems.

- ~~constructor~~ constructor that register() a set of type mappings and default Hibernate Properties
- Subclasses should be immutable

```

import java.sql.Types;          ↗ SQLPackage
(I). public class Types
{
    public final static int INTEGER = 4;  ↗ Variable
}

```

```

(II).
import java.sql.Types;
public class OracleDialect extends Dialect
{
    public OracleDialect()
    {
        super();
        registerColumnType(Types.INTEGER, "number(10,0)");
    }
}

```

```

(III).
public class MySQLDialect extends Dialect
{
    public MySQLDialect()
    {
        super();
        registerColumnType(Types.INTEGER, "integer");
    }
}

```

```

(IV).
public class NareshDialect extends Dialect
{
    public NareshDialect()
    {
        super();
        registerColumnType(Types.INTEGER, "nareshInteger");
    }
}

```

Note: 90% of the Cases we don't shift from one DataBase to Another DataBase, we shift from One Environment to Another Environment, we don't write Dialect class also.

→ You don't need to change the SQL Scripts, if you change Database

⑥ you don't need to write most of the SQL Scripts for persisting, Deleting and objecting the resultsets.

JDBC

Hibernate

1. INSERT INTO STUDENT VALUES (...); 1. session.save(student);

2. update student set ...; 2. session.update(student);

3. Delete from student; 3. session.delete(student);

Note: If the Query is having some condition, we write the Query using "HQL" (or) we build the Query using "Criteria".

↓ ①

Hibernate Query Language

↓ ②

① & ② Imp topics

⑦ It is very easy to make a cleaner separation of Data Access Layer

→ In JDBC, we have to write very huge code and in Hibernate it is very less code, readability is more. So it is very cleaner.

⑧ The Core drawback of JDBC is that it doesn't follow you to store objects directly to the Database you must convert the object to a relational format. For instance, if you want to persist a new instance of the Student class to the Database you must first convert the student object to SQL statement that can be executed on the underlying Database.

→ Similarly, When rows are returned from the Database, you must convert each result row into an instance of student.

→ In Hibernate, Directly, we deal with objects, we don't deal with

(9) Hibernate is not tightly tied with any underlying database.

whereas JDBC is tightly tied with the underlying database.

(10) Hibernate is independent, your code will work for all ORACLE, MYSQL, SQLSERVER etc.

(*) In case of JDBC Query must be database specific

→ As Hibernate is set of objects, you don't need to learn SQL language, you can treat TABLE as a object. Only Java knowledge is need.

(*) In case of JDBC you need to learn SQL

Note: If we are working with hibernate, we don't require the Database SQL Syntax knowledge. But we require the SQL knowledge and Hibernate SQL Syntax knowledge.

(11) Don't need Query tuning in case of Hibernate. If you use Criteria Queries in Hibernate then Hibernate automatically tuned your Query and return Best result with performance.

(*) In case of JDBC you need to "tune" your queries

The Query is not proper order

(12) Development fast in case of Hibernate because you don't need to write Queries.

(13) In the XML file, you can see all the Relations between tables in case of Hibernate.

(*) Easy Readability.

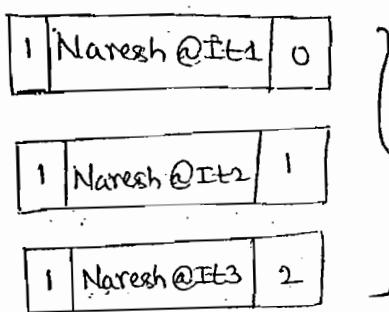
One-to-one
One-to-many
Many-to-one
Many-to-many
Inheritance Concepts.

(14) You can Load your objects on Startup using Lazy = false

In case of Hibernate - JDBC don't have such support.

21/10/20
FRI

15) Hibernate supports Automatic Versioning of rows but JDBC Not.



Automatic change the Version

Eg: Hibernate Record Version

2/10/09
Friday

Controller it Never Goto The XML File :

- The Controller it never go to the XML file once the Container is up some Java class will read the complete XML file and it will keep the XML information in a Java object.
- If it is Struts, The Complete struts-config.xml information will be available in a Java object called "Module Config", In Servlet Application the implementation might be,

```
< servlet >
  < servlet-name > StudentServlet < /servlet-name >
  < servlet-class > edu. sevlets. StudentServlet < /servlet-class >
< /servlet >
< servlet-mapping >
  < servlet-name > StudentServlet < /servlet-name >
  < url-pattern > / StudentServlet < /url-pattern >
< /servlet-mapping >
```

// Read the XML file and Take the data and keep it in Map.

```
Map servletMappingMap = new HashMap();
servletMappingMap . put ( " StudentServlet " , StudentServlet );
Map servletMap = new HashMap();
servletMap . put ( " StudentServlet " , edu. sevlets. StudentServlet );
```

`http://localhost:8080/StudentServlet / StudentServlet`

```
public class Container
{
    String servletName = servletMappingMap.get(StudentServlet);
    String servletClass = servletMap.get(servletName);
    Servlet servlet = (Servlet) Class.forName(servletClass).newInstance();
    servlet.service();
}

class A // GenericServlet
{
    public void X()
    {
        ...
    }
}

class B extends A // StudentServlet
{
    public void X()
    {
        ...
    }
}

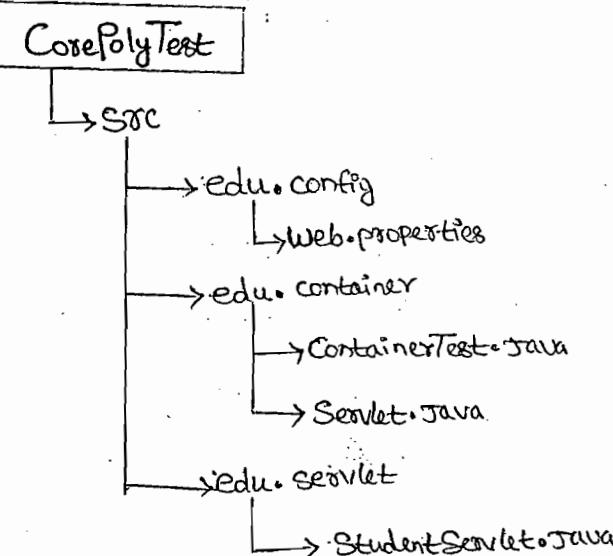
A a = new B();
a.X();

// Servlet servlet = StudentServlet Object
// servlet.service();
```

Note:

A a = new B();
Understanding wise it is completely wrong.

Directory Structure:



Web.properties:

```
className = edu.servlet.StudentServlet
```

ContainerTest.java:

```
package edu.container;
import java.io.IOException;
public class ContainerTest {
    private static Properties properties = new Properties();
    private static String servletClass = "";
    static {
        try {
            properties.load(ContainerTest.class.getClassLoader().getResourceAsStream("edu/config/web.properties"));
        } catch (IOException e) {
        }
        servletClass = properties.getProperty("className");
    }
    public static void main(String args[]) throws InstanceException {
        // A a = new B();
    }
}
```

Servlet Servlet = (Servlet) Class.forName("ServletClass").
newInstance();

Servlet.service();

// Connection - SUN → A

// JDBCODBC CONNECTION, ORACLE CONNECTION

- SUN, ORACLE → B

// Connection con = DriverManager.getConnection();

- Developer → A a = new B();

}

3

StudentServlet.java :

```
package edu.servlet;  
import edu.container.servlet;  
// B  
public class StudentServlet implements Servlet  
{  
    public void service()  
    {  
        System.out.println("StudentServlet "+ "service()");  
    }  
}
```

3

3
W

Note: Most of the classes at the Architectural Level are never implemented A, B and A a = new B(); By the same person...

It will be done by different people [Create B object using
Class.forName()]

Note: 1: Session session = sf.openSession();
session.save();

Session → A

SessionImpl → B

Session session = sf.openSession(); → A a = new BU;
session.save(); → a.setX();

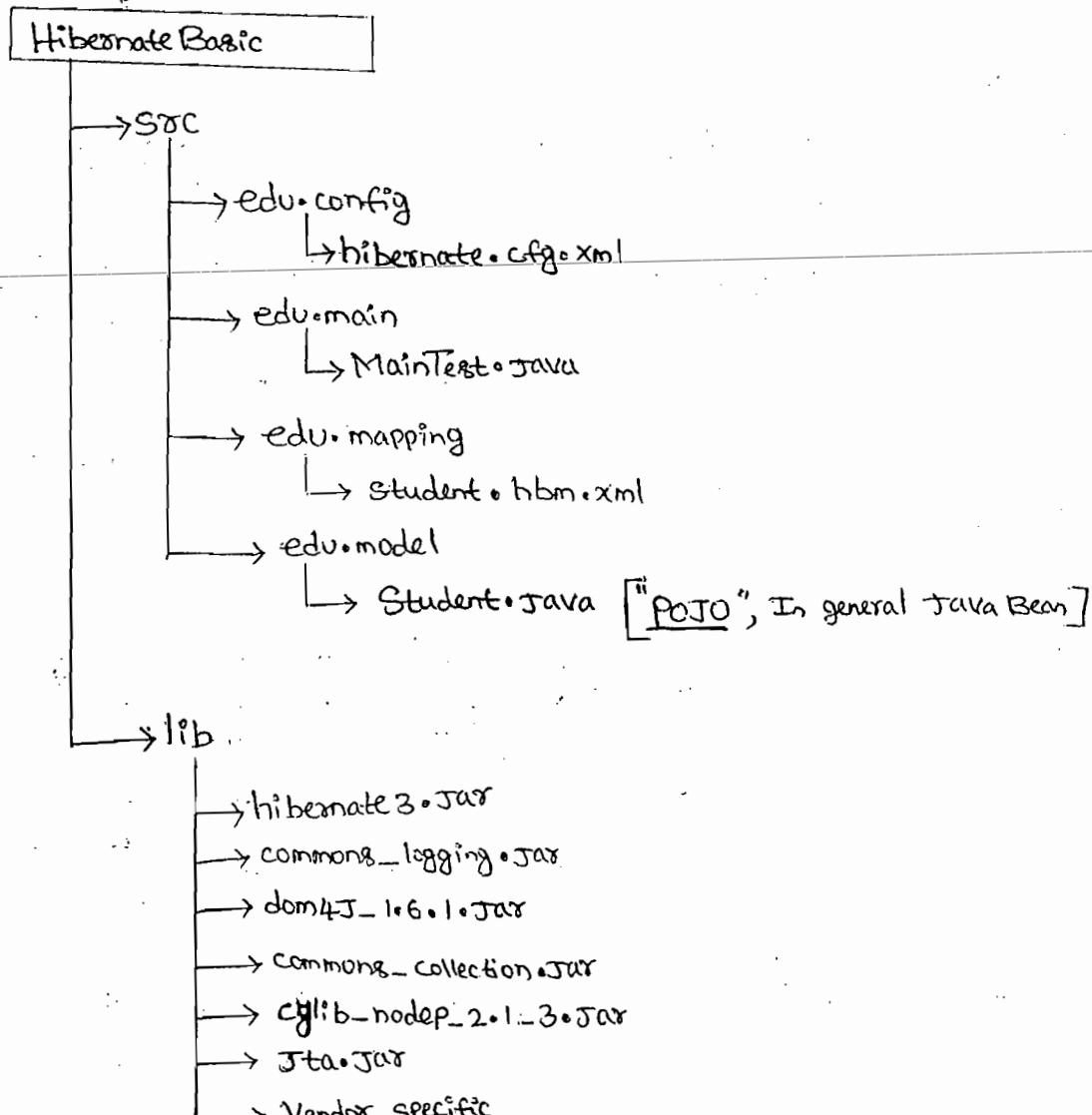
Note: 2. SessionFactory sf = configuration.buildSessionFactory();

SessionFactory → A

SessionFactoryImpl → B

SessionFactory sf = configuration.buildSessionFactory();
sf.openSession(); → a.setX(); → A a = new BU;

Hibernate First Application:



Hibernate • Cfg.xml

Hibernate Configuration file contains,

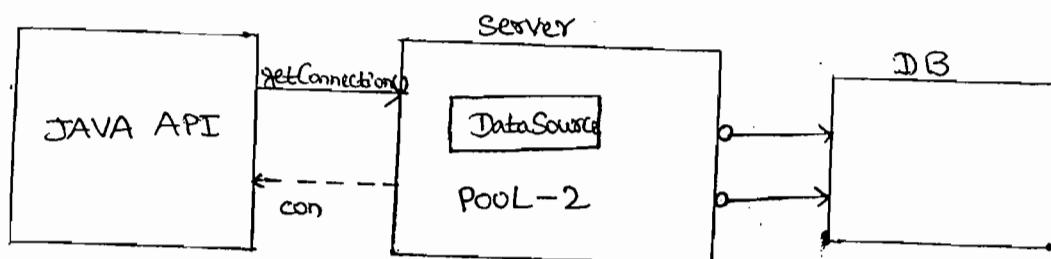
1. DataBase Connection Details
2. Connection pool size
3. Show SQL
4. hbm2ddl
5. All the HBM file Entries
6. Dialect class etc.

DataBase Connection Details

→ We are providing the DataBase connection details URL, UserName ---- etc. But it never be given in the project,
In projects we specify the INDI and DataSource details.

Note:

→ In JDBC projects to get the connection object we use DataSource concept means in the Server we need to configure the DataBase details while configuring we select TYPE4 Driver and which will be mapped in a JNDI name means we are using 3 and 4 combination [3 means Configuration the DataSource in the Server] while configuring this we provide the Connection pool.



Note: - The Connection pool size is 2 Once the container is up it will be ready with 2 connections from Server to DataBase, whenever we required the connection from the Application we contact

The server it means the connection and After our JDBC Logic we close the connection. Then the connection will be closed from Application to the Server, Server to DataBase connections will be closed whenever the Server is shutdown.

→ The Advantage is performance wise it will be fast.

3/10/09
Saturday Connection Pool size :

→ Based on the specified connection pool size once the Session Factory object is created it will be ready with the specified number of session objects [connection objects].

Dialect :

→ Dialect contains Java Datatype and DataBase Datatype.

Show-Sql :

→ If it is true, we can see the Hibernate generated Query in Console.
If it is false, It does not show the generated Query in the Console.

hbm2ddl :

→ If the value is create, it will drop the old table and it will recreate the table.
→ If it is update and there is no table. Then it creates the table and if there is a table, it will do the SQL operation, the update value should be small.

Note : We don't provide these value in the project Because the Table creation is done by the DataBase people.

Mapping Tag :

→ Each table should be mapped to a POJO and The mapping tag we write in the HBM file and All the HBM file entries should be given in the configuration file using Mapping tag.

hbm file [hibernate-mapping file] :

- hbm file contains the mapping details means what is the Pojo Name and what is the Table Name, what is property and what is the column name and All other mappings [one-to-one, one-to-many, subclass, ... etc].
- Each hbm file should contain identifier column (or) property using Id tag (or) The corresponding tag [Composite Id].
- Id tag should contain primarykey column (or) Non-primary key column and In projects most of the classes we give Primary key column and some classes we give Non-primary key column also.
- To update the record we call update method on session while passing the pojo object.

1. Session.update(student);

HBM

```
<id name = "studentName" column = "SNAME">  
<generator class = "assigned"/>
```

```
</id>
```

generated Query

Update STUDENT SET SNO=? WHERE SNAME=?

→ Here it is taking SNAME in the where condition Because the object is identified using studentName [Id tag is having studentName] Property and it is Non-primary key.

2. Session.delete(student);

HBM

```
<id name = "studentNo" column = "SNO">  
<generator class = "assigned"/>  
</id>
```

generated Query

Select * from STUDENT where SNO = ?

→ Here, it is taking SNO in the where condition Because the object is identified using studentNo property and it is primary key [Id tag is having studentNo].

Note: If there is Query with more than one condition - Then we write userdefined Query using HQL...etc.

Property tag :

→ To map the other columns we can use property tag.

<property name = "studentName" column = "SNAME" />
(OR)

<property name = "studentName" />

→ If we don't give the column name the property name is treated as the column.

→ If Both are same also (Property & column), In project we give both.

How To Read the Configuration file:

→ Using the Configuration object we can read the configuration file and we call the configure method with passing a cfg file
(Or) without passing a cfg file

→ An instance of Configuration allows the Application to specify properties [Dialect class, ShowSql, URL,----] and an Entire set of mappings of an applications Java types of an SQL DB when creating a SessionFactory. Usually an Application will create a single Configuration, Build a single instance of SessionFactory and then Instructs the SessionFactory in threads servicing client requests.

- The Configuration is meant only as an initialization time object
- SessionFactories are immutable and do not retain any association back to the configuration.
- The mappings are compiled from various XML Mapping files.
- Once the SessionFactory direct is created, we cannot change the state like String because it's immutable.

Configuration configuration = new Configuration();

configuration.configure();

- While calling the configure() method, If we don't pass the Configuration file then it looks for hibernate.cfg.xml and it should be there in the source directory.

Internal code:

```
public Configuration configure()
{
    configure("hibernate.cfg.xml");
    return this;
}
```

- In project's, Cfg file will be there in Config package, so we need to use the parameterized Configure method.

Configuration configuration = new Configuration().configure("edu/config/hibernate.cfg.xml");

- Configure method internally implements ClassLoader.getResourceAsStream to identify the Cfg file location then they will read the Cfg file and they will create a Stream object and After they will apply XML parsers to parse the XML and to get the values and After they create an Environment object and it contains the complete information about the Cfg file

Internal code:

```
public Configuration configure( String resource )
{
    InputStream stream = getConfigurationInputStream( resource );
    return doConfigure( stream, resource );
}

protected Configuration doConfigure( InputStream stream, String resourceName )
{
    org.dom4j.Document doc;
    return doConfigure( doc );
}

protected Configuration doConfigure( org.dom4j.Document doc )
{
    Element sfNode = doc.getRootElement().element("session-factory");
    String name = sfNode.getAttributeValue("name");
    Properties.setProperty( Environment.SESSION_FACTORY_NAME, name );
}
```

Environment:

```
public final class Environment
```

```
    public static final String DRIVER = "hibernate.connection.  
driver-class";
```

```
    public static final String SHOW_SQL = "hibernate.showSQL";
```

→ Provides Access to Configuration info passed in properties
objects [cfg. file details].

→ Instead of giving the details in the .cfg file directory we
set those details to Configuration object . But we never use
this Approach.

```
Configuration configuration = new Configuration();
configuration.setProperty( driverClass, value );
```

SessionFactory:

- The SessionFactory you use the Hibernate SessionFactory to Create Session objects that manage connection data, caching, and mappings.
- Your Application is responsible for mapping the SessionFactory. You should only have one SessionFactory unless you are using Hibernate to connect to two or more DataBase instances with different settings, in which case you should still have one SessionFactory for each DB instance.
- You obtain a Session from the SessionFactory object using one of the four openSession() methods. The no argument openSession() method opens a Session.
- A Threadsafe (immutable) Cache of compiled mapping for a single DataBase.
It's a Factory to provide Session objects...

SessionFactory sessionfactory = configuration.buildSessionFactory();

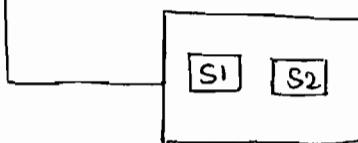
51
Hour

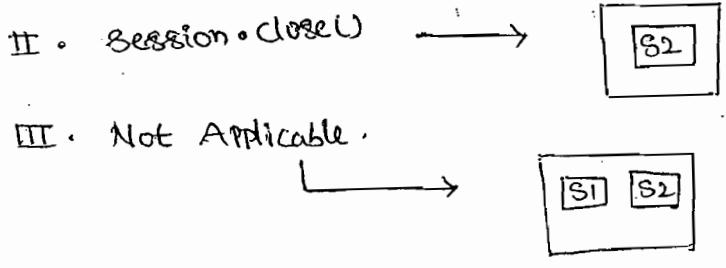
Session:

- A Single-threaded, short-lived object representing a conversation between the Application and the persistent store.
- Wraps a JDBC connection factory for transaction. For each client request create a Session object and destroy after the task (session.close()).

I. Session

Session = sessionfactory.openSession();





Transaction :

- A single-threaded, short-lived object used by the Application to specify ATOMIC Units of work.
- Abstracts Application from underlying JDBC, JTA (or) CORBA transaction. A Session might span several transactions in some cases.

Transaction transaction = session.beginTransaction();

transaction.commit();
 (or)
 transaction.rollback();

5/16/10 9
 Monday

hibernate.cfg.xml :

```

<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver-class">
      oracle.jdbc.driver.OracleDriver </property>
    <property name="hibernate.connection.url">
      jdbc:oracle:thin:@localhost:1521:SERVER </property>
    <property name="hibernate.connection.username">scott </property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect </property>
    <property name="show-sql">true </property>
    <property name="hibernate.hbm2ddl.auto">update </property>
  </session-factory>
</hibernate-configuration>
```

```
<mapping resource = "edu/mapping/student.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Student.java:

```
package edu.model;
public class Student {
    private Long studentNo;
    private String studentName;
    getters & setters
}
```

Student.hbm.xml:

```
<hibernate-mapping>
<class name = "edu.model.Student" table = "STUDENT">
<id name = "studentNo" column = "SNO">
<generator class = "assigned"/>
</id>
<property name = "studentName" column = "SNAME"/>
</class>
</hibernate-mapping>
```

MainTest.java:

```
public class MainTest {
    public static void main (String args[]) {
        Student student = new Student();
        student.setStudentNo (new Long(1));
        student.setName ("Naresh @ It");
    }
}

Configuration configuration = new Configuration();
configuration.configure ("edu/config/hibernate.cfg.xml");
```

// Step 2

SessionFactory sessionFactory = configuration.buildSessionFactory();

// Step 3

Session session = sessionFactory.openSession();

// Step 4

Transaction transaction = session.beginTransaction();

// Step 5.

session.save(student);

transaction.commit();

System.out.println("Success");

}

}

Drawbacks (or) DisAdvantages of Above Example:

1. We are ~~already~~ reading the Configuration details in the main program it suppose to be in a util class .. Because we need to send the Configuration only once in the project.

2. We are creating the Student object and initializing the Hard coded values ; it suppose to be in Servlet and we need to initialize the request values to the Student object.

Main:

Student student = new Student();

student.setStudentNo(new Long(1)); — Wrong

Servlet/Action:

Student student = new Student();

student.setStudentNo(request.getParameter("studentNo"));]

Correct ←

3. Transaction Logic we are writing in main program . It suppose to be there in Service classes .

Transaction transaction = session.beginTransaction();

4. Persistence Logic we are waiting in Main program, it suppose to be there in DAO classes.

Session. save (student);

5. We are not handling the Exception, we suppose to handle the exception and we need to rethrow the exception back to the servlet.

public class SessionUtil

{

/** ThreadLocal SessionMap */

public static final ThreadLocal Map = new ThreadLocal();

private static final Log LOG = LogFactory.

getLog (SessionUtil.class);

private static final SessionFactory SESSION_FACTORY;

/** Make Default Construct private */

private SessionUtil()

{ }

/** Loads Hibernate config */

static

{
try
{

LOG.debug ("SessionUtil.static - Loading config");

SESSION_FACTORY = new Configuration().

configure ("edu/config/hibernate.config.xml").

buildSessionFactory();

LOG.debug ("SessionUtil.static - End");

}

catch (HibernateException ex)

{

LOG.error ("Exception Building SessionFactory");

throw new ExceptionInInitializerError ("Exception Building Session Factory :" + ex.getMessage());

{ }

public static Session getSession() throws HibernateException

{
 /* Open a New Session */

 Session session = SESSION_FACTORY.openSession();
 return session;

}

/** Gets Hibernate Session for current thread */

public static Session currentSession() throws HibernateException

{
 Session session = (Session) MAP.get();

 /* Open a New Session, If this Thread has none yet */

 if (session == null)

{

 Session = SESSION_FACTORY.openSession();

 MAP.set(session);

}

 return session;

}

/** Closes the Hibernate Session, Users must call this
method After calling currentSession() */

public static void closeSession() throws HibernateException

{
 Session session = (Session) MAP.get();

 MAP.set(null);

 if (session != null)

{
 session.close();

}

}

public static void closeSession(Session session) throws HibernateException

{
 if (session != null)

{
 session.close();

}

}

}

- It is used to read the Configuration once and to Create Configuration and SessionFactory Objects once.
- To Create the session object and to close the session object.

`getSession()`

User1 → m Requests

User2 → n Requests — Total → m+n Session Objects.

`currentSession()`

User1 → m Request

User2 → n Request — Total → 2 Session Objects.

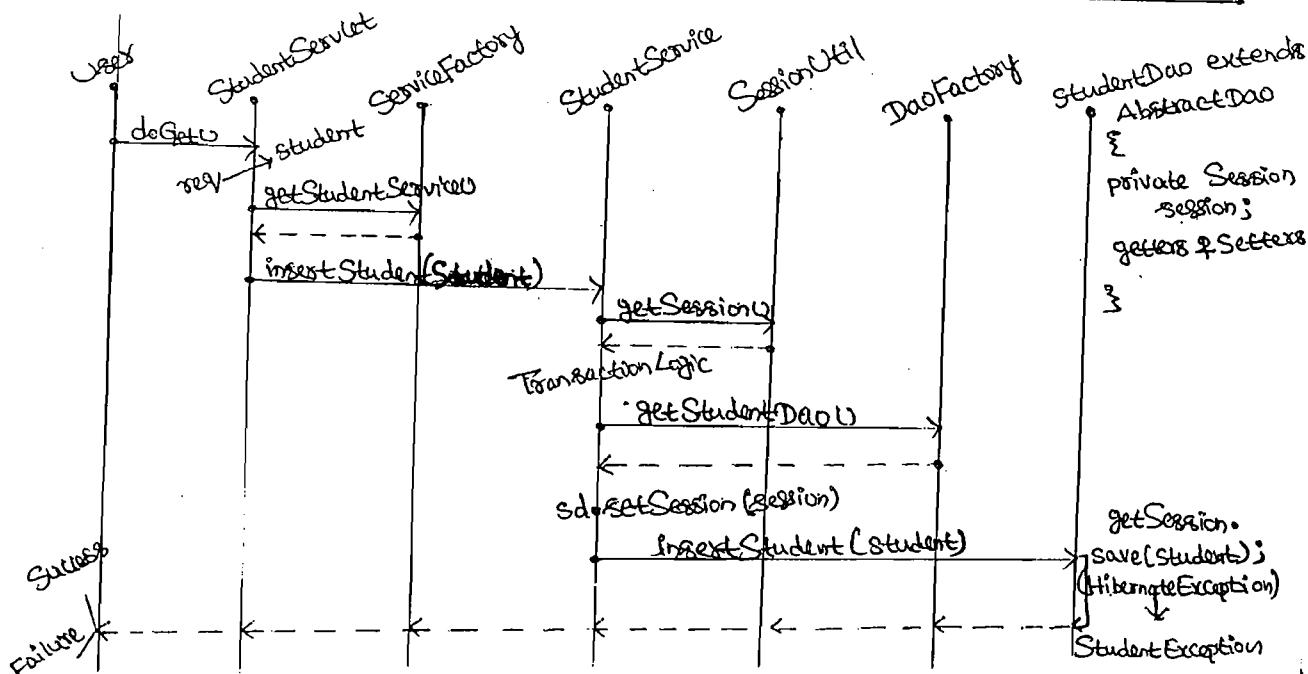
ThreadLocal:

- It is used to Cache the Session object for each Thread, If there "N" threads there will be "N" number of ThreadLocal objects.
- For all Threads if we want to Cache the Session object we need to take HashMap.

6/10/09
Tuesday

35
Wk

Layered Architecture Using Servlet & Hibernate Communication:



read

Generator class:

Ctrl+Shift+D:

* Hibernate → M→ Generators

<generator class = " <anyGeneratorClass>" />

→ All the generator classes are implementing IdentifierGenerator interface and it has to override generate() method.

(1) Assigned:

→ assigned Lets the Application to assign an identifier to the object before save() is called. This is the default strategy if no <generator> element is specified.

Syntax:

```
<id>
  <generator class = " assigned" />
</id>
```

Application Code:

```
Student student = new Student();
student.setStudentNo(new Long(1));
student.setStudentName("Naresh@It");
session.save(student); → True
```

```
Student student = new Student();
student.setStudentName("Naresh@It");
session.save(student); → False
```

→ Here it will throw IdentifierGenerationException. Because, before calling save() method we need to set the "studentNo"

Internal code for Assigned:

```
public class Assigned implements IdentifierGenerator, Configurable
{
    private String entityName;
    // Executes generate 2nd
    public Serializable generate(SessionImplementor session, Object obj)
    {
        final Serializable id = session.getEntityPersister(entityName,
obj).getIdentifier(obj, session.getEntityMode());
    }
}
```

// TODO: Cache The persister, This shows up in your kit

```
if (id == null)
{
    throw new IdentifierGenerationException("ids for this
class must be manually assigned before calling save():
entityName");
}
return id;
}
```

// Executes Configure 1st

```
public void configure(Type type, Properties params, Dialect d)
{
    throws MappingException
    entityName = params.getproperty(ENTITY_NAME);
    if (entityName == null)
    {
        throw new MappingException("no entityName");
    }
}
```

→ From Application we are calling save method, Internally it will call configure and generate methods.

→ org.hibernate.id.Assigned is the generator class for assigned

(2). Sequence:

→ Uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, Mckoi (68)
a generator in Interbase - The returned identifier is of type long, short (or) int.

→ 'sequence' generates a number and generated number will be stored in SNo column.

→ MySQL does not support sequence

Syntax

```
<generator class = "sequence">           SequenceName
                                            ↓
<param name = "sequence"> STUDENT_SEQ </param>
</generator>
```

- CREATE SEQUENCE STUDENT_SEQ START WITH 1 INCREMENT BY 1.
- If we don't give the sequence name. Then it takes HIBERNATE-SEQUENCE as the default one.

[<generator class = "Sequence" />]

- SequenceGenerator is the generator class for Sequence.

Application code :

```
Student student = new Student();
student.setStudentName("Naresh@IT");
session.save(student);
```

Internal code for SequenceGenerator :

```
public interface PersistentIdentifierGenerator extends IdentifierGenerator
{
}
```

```
public class SequenceGenerator, implements PersistentIdentifierGenerator,
Configurable
```

```
{ public static final String SEQUENCE = "sequence"; }
```

```
---- configure(----)
```

```
{
```

```
// If there is no sequence
```

```
sequenceName = PropertiesHelper.getString(SEQUENCE, params,
"hibernate_sequence");
```

```
// If there is sequence, Then it takes from hbm file
```

```
{
```

```
---- generate(----)
```

```
{
```

```
try
```

```
{ PreparedStatement st = Execute the Sequence;
```

```
ResultSet rs = st.executeQuery();
```

```
rs.next();
```

```
Serializable result = IdentifierGeneratorFactory.
get(rs, identifierType);
```

```
{
```

```
.... in 'em column.
```

8/16/07 (3). UserdefinedGenerator class;
Thursday

→ We can write our Generator class to generate the number.
Take Any class and implement IdentifierGenerator and override
generate() method.

→ Give the UserdefinedGenerator class in the hibernate file.

Package eduoids;

public class StudentNoGenerator implements IdentifierGenerator

{
 public Serializable generate(SessionImplementor session,
 Object object) throws HibernateException

 String studentNo = "STU";

 Connection connection = null;

 connection = DBUtil.getConnection();

 StringBuffer studentQuery = new StringBuffer("SELECT
 TO_CHAR(STUDENT_SEQ.NEXTVAL, 'FM0000000000')
 FROM DUAL");

 try

{

 Statement statement = connection.createStatement();

 ResultSet resultSet = statement.executeQuery
 (studentQuery.toString()));

 if (resultSet != null && resultSet.next())

{

 studentNo = studentNo + "" + resultSet.
 getString(1);

}

}

 catch (SQLException e)

{}

 return studentNo;

}

Notes:

'FM000000000' → Format the Number

If sequence returns 1 Then 9 zero's 1

If sequence returns 12 Then 8 zero's 12

Syntax:

```
<generator class="edu.id.StudentNoGenerator"/>
```

Application code:

```
-- do -- [Refer the Sequence Example code]
```

Note: Change studentNo datatype to String in Student object and
change sno column datatype to Varchar in STUDENT table

Identity:

- Identity supports identity columns in DB2, MySQL, MS SQL Server,
Sybase and HypersonicSQL • The returned identifier is of type
long, short or int.
- Oracle doesn't support identity columns.
- Here DataBase supports identity columns and it returns a
number and it will be stored in student table
- Check this example with MySQL

How To Connecting To MySQL:

1. Double click on the MySQL Icon and give password as "root"
2. Create DataBase [DatabaseName as] studen1
3. use studen1 [Then it will be connected]

In cfg file we need to provide this information:

```
com.mysql.jdbc.Driver  
jdbc:mysql://localhost/studen1  
root  
root
```

Syntax:

```
<generator class="identity"/>
```

Application Code:

```
-- do -- [Refer the Sequence Example code]
```

→ IdentityGenerator is the generator class for Identity

- hilo uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a table and column (By default hibernate-unique-key and next_hi respectively) as a source of hi values.
- The hi/lo algorithm generates identifiers that are unique only for a particular database.

Syntax :

```
<generator class = "org.hibernate.id.TableHiLoGenerator">
  <param name = "table"> HI-VALUE </param>
  <param name = "column"> NEXT-VALUE </param>
</generator>
```

Application code :

-- do -- [Refer the sequence example code]

Note : Change Student object data type to Long

→ TableHiLoGenerator is the generator class for hilo algorithm.

Internal code :

```
public class TableHiLoGenerator extends TableGenerator
{
  --- generate(---)
  {
    if (maxLo < 2)
    {
      // keep the behaviour consistent even for boundary usages
      int val = ((Integer)super.generate(session, obj))
        .intValue();
      return IdentifierGeneratorFactory.createNumber(val,
        returnClass);
    }
    if (lo > maxLo)
    {
      int hival = ((Integer)super.generate(session, obj))
        .intValue();
```

```
lo = (hival == 0) ? 1 : 0;
```

```
hi = hival < (maxLo + 1) ?
```

```
}
```

```
return IdentifierGeneratorFactory.createNumber(hi + lo + ,
```

```
returnClass);
```

```
}
```

```
}
```

(4). native :

→ native picks Identity, Sequence or hilo depending upon the Capabilities of the underlying database.

Syntax :

```
<generator class = "native" />
```

Application code :

-- do -- [Refer the Sequence Example code]

→ If we are connecting to Oracle, it will select Sequence and generator class is SequenceGenerator.

→ If we are connecting to MySQL, it will select Identity and generator class is IdentityGenerator.

Q. When should we have to use native ?

→ Oracle doesn't support Identity and MySQL doesn't support Sequence.

→ If we are using Sequence for Oracle and if we are changing the DataBase to MySQL. Then it doesn't work and for Identity is also same.

→ Whereas in above scenario, if we use native it will select automatically Identity (or) Sequence based on the DataBase.

9/10/07
Friday

→ increment generates identifiers of type long, short, or int that are unique only when no other process is inserting data into the same table. Do not use in a cluster.

Syntax:

```
<generator class = "increment"/>
```

Application code:

```
-- do -- [Refer the sequence example code]
```

Note: Take studentNo datatype as Long

Internal code:

```
public class IncrementGenerator implements IdentifierGenerator
{
    generate() {
        buf.append("select").append(column).append("from");
        PreparedStatement st = ***;
        ResultSet rs = st.executeQuery();
        if (rs.next())
            next = rs.getLong(1) + 1;
    }
}
```

(G) SeqHilo:

→ seqhilo uses a hi/lo algorithm to efficiently generate identifiers of type long, short (or) int, given a named DataBase sequence.

Syntax:

```
<generator class = "seqhilo">
```

```
<param name = "sequence" > STUDENTIDSEQ </param>
```

```
<param name = "max_lo" > 100 </param>
```

```
</generator>
```

Application code:

-- do -- [Refer The sequence Example code]

Note: studentNo data type should be Long.

→ SequenceHiLoGenerator is the generator class for seqhilo

→ Now, The HiLo Algorithm uses sequence generator number and it will apply algorithm logic then the final number will be stored in SNO column.

(7). UUID:

→ Uuid uses a 128-bit UUID algorithm to generate identifiers of type string, unique within a network [The IP address is used].

→ The UUID is encoded as a string of hexadecimal digits of length 32.

→ It uses the below values to generate the number

getIP() getJVM() getHiTime()) getLoTime()) getCount()

Syntax:

<generator class = "UuidHex"/>

Note: studentNo datatype must be String and SNO column must be varchar.

→ UuidHexGenerator is the generator class for Uuid.

Note: To generate different password using this Uuid.

Application code:

-- do -- [Refer The sequence Example code]

(8). GUID:

→ guid uses a database-generated GUID String on MS SQL Server and MySQL.

(9). Select:

→ Select retrieves a primary key assigned by a database trigger by selecting the row by some unique key and retrieving the

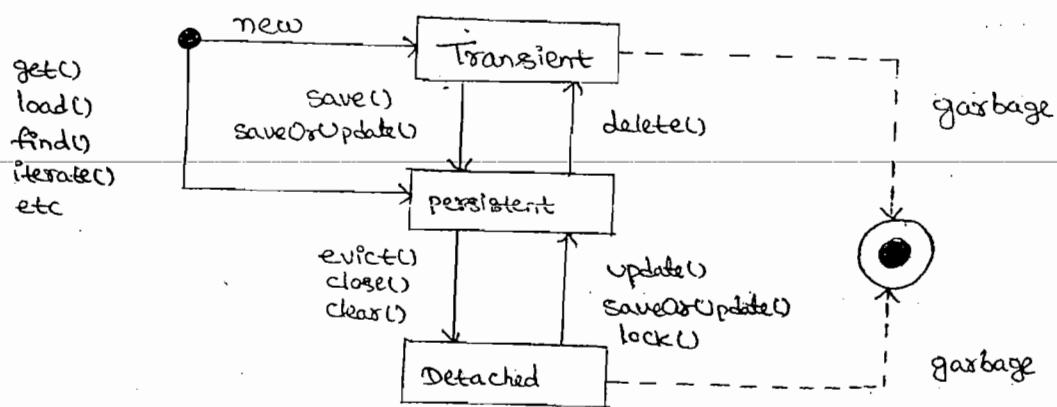
→ foreign uses the identifier of another associated object.
Usually used in conjunction with a one-to-one primary key association.

Note: SNO might be inserted using database trigger, if we want to Select the trigger generated number. Then use Select

Object State [POJO Object State (Student)]

1. Transient Objects
2. Persist Objects
3. Detached Objects

Object Lifecycle



* Affects all instances in a session

1. No Record in the DataBase → object is not Associated with a session is called as Transient state.
2. Associate session → Persistent state
3. There is Record in the DataBase → object is not Associated with a session is called as Detached state.

10/10/09

Saturday Transient Objects:

- Objects instantiated using the new operator aren't immediately persistent. Their state is transient, which means they are not associated with any database table row, and so their state is lost as soon as they are dereferenced.

Persist Objects:

- A persistent instance is any instance with a database identity. Persistent instances are associated with the persistence manager. Persistent instances are always associated with a session and are transactional.

Detached Objects:

- Instances lose their association with the persistence manager when you close the session. We refer to these objects as detached, indicating that their state is no longer guaranteed to be synchronized with database state;
- They are no longer under the management of Hibernate.

Eg:

Table Name: STUDENT

SNO	SNAME
—	—

→ No Records

Application Code:

```
Student student = new Student();
student.setStudentName("N@IT");
Session session = sessionFactory.openSession();
session.beginTransaction();
session.save(student);
transaction.commit();
session.close();
```

After Executing the program

SNO	SNAME
1	N@IT

Transient : Student object is not there in DataBase and it is not associated with session.

Persistent : Student object is associated with session.

Detached : Student object is there in DataBase and it is not associated with session.

Note : Persistence object will be synchronized automatically with Database while committing the transaction.

1. load():

→ To retrieve a DataBase record in the form of Java object we can use load() method.

Syntax :

```
Student student = (Student) session.load(  
    Student.class, new Long(1));  
    [POJO.class, Identifier column value(SNO)(IDtag)]
```

1. If we are trying to retrieve a record and if it is not there in the DataBase then it will throw ObjectNotFound Exception.

2. The object will not be initialized with the values while calling session.load() method, The object will be initialized while calling student.getXXX() method.

→ load() method is Lazy initialization.

2. get():

→ It is same as load() method.

Syntax :

```
Student student = (Student) session.get(  
    Student.class, new Long(1));  
    [POJO.class, Identifier column value(SNO)(IDtag)]
```

1. While loading a record from the DataBase, If the record is not there in the DataBase, Then it will return null. It does not return any Exception.

2. While calling `session.get()` method object [student] will be initialized.

→ The Above two Syntax we can use while loading an object using load() method.

→ If we Create Student object.

```
Student student = new Student();
```

```
session.load(student, new Long(1));
```

→ If we don't create the student object.

```
Student student = null;
```

```
student = (Student) session.load(Student.class, new Long(1));
```

CoreJava Fundamental Example:

```
package edu.main;
public class MainTest {
    public static void print(Object obj) {
        public static Object get(boolean status) {
            if (status)
                return new String("N@It");
            else
                return new Integer("123");
        }
        public static void main(String args[]) {
            //I
            print(new String("N@It"));
            print(new Integer("123"));
            //II
            String str = (String) get(true);
            //III
            Object obj = new String();
            String str1 = (String) obj;
            //IV
            Object obj1 = new Object();
            String str2 = (String) obj1;
            //V
            Object obj3 = new String("N@It");
            // System.out.println(" --- " + obj3.length());
        }
    }
}
```

10. delete:

→ To delete a record in the DataBase.

Application code:

```
Student student = new Student();
student.setStudentNo(new Long(1));
session.delete(student);
```

Generated Query:

```
delete from STUDENT where SNO=?
```

→ Hibernate will take the id column in the where condition

Note: Before calling the delete() method we need to assign the identifier value to the Student object, other values are not required [studentName, studentAge etc]

(4) update:

→ To update a database record, we call the update() method

Application code:

```
Student student = new Student();
student.setStudentNo(new Long(1));
student.setStudentName("updated@it");
session.update(student);
```

Generated Query:

```
update STUDENT set SNAME=? where SNO=?
```

→ Hibernate will take the id column in the where condition.

Note: Before calling the update() method we need to assign id tag value (studentNo) with the other values, whatever we want to update (name, age etc...). But studentNo is must. If we don't assign the other values it will be taken as null.

Note:

1. While calling the update() method Detached Object will become Persistence.
2. While calling the delete() method persistent object will become Transient.

Monday 2/10/09 (5) `clear()`: The records will be removed from the session not from the DataBase.

Syntax: `Session.clear();`

1. Student `student = null;`

```
student = (Student) session.get(Student.class, new Long(1)); → p  
student.setStudentName("updated");  
session.clear(); → D
```

`transaction.commit(); // Not updated to the DB. Because if we call clear object becomes Detached, so it will not be synchronized with the DB while committing the transaction`

2. Student `student = null;`

```
student = (Student) session.get(Student.class, new Long(1)); → p  
student.setStudentName("updated");  
transaction.commit(); // Updated to the DB. Because object is in persistence state, so it will synchronize with DataBase with committing the transaction
```

(6) `close()`:

→ End the Session by disconnecting from the JDBC Connection and clearing up.

→ It is not strictly necessary to `close()` the session but you must at least `disconnect()` it.

Syntax: `Session.close();`

How To Get The Connection Object in Hibernate?

Syntax:

```
Connection connection = session.Connection();
```

→ It is connected to the Hibernate to JDBC.

→ This is Deprecated.

(7) `contains()`:

→ Check if this instance is associated with this session.

```
Student student = null;
```

```
Student student = ...
```

```

student2.setStudentNo(new Long(1));
student1 = (Student) session.get(Student.class, new Long(1));
if (session.contains(student1))
{
    System.out.println("1. True");
}
else
{
    System.out.println("1. False");
}
if (session.contains(student2))
{
    System.out.println("2. True");
}
else
{
    System.out.println("2. False");
}

```

→ If the object is there in session returns true otherwise it returns false

(B). commit();

1. Flush the Associated Session and End the Unit of work
2. This method will commit the underlying transaction if and only if the transaction was initiated by this object.
3. Just the Connection is close. If we want we can reconnect.

Syntax: transaction.commit();

```

Student student = null;
student = (Student) session.get(Student.class, new Long(1));
Session.disconnect();
student.setStudentName("updated");
transaction.commit();

```

→ The Record is updated to DataBase because this method will commit the transaction [Refer The previous 2 points]

(9) .evict():

- evict() method will remove an object from the session not from the DataBase. whereas clear() method will remove all the objects from the session.
- Remove this instance from the Session Cache, changes to the instance will not be synchronized with the DataBase.
- The record is not updated to DataBase Cache, Student is in Detached state after calling evict method.
Syntax: session.evict(Student);

Student student = null;

Student = (Student) session.get(Student.class, new Long(1));
session.evict(student);

student.setStudentName("Updated");
transaction.commit();

(10). flush():

- Force the Session to flush must be called at the end of a unit of work, before committing the transaction and closing the session [transaction.commit() calls this method]
- Flushing is the process of synchronizing the underlying persistent store with persistable state held in memory.
- Whenever the flush method is called the Session objects will be synchronized with the Database.
- When commit() method is called the Session objects will be synchronized with the DataBase and commit() is called.

1. Student student = null;

student = (Student) session.get(Student.class, new Long(1));
student.setStudentName("Updated");

System.out.println("Break point.");

transaction.commit();

↳ Student session

```
student = (Student) session.get(Student.class, new Long(1));
student.setStudentName("Updated");
session.flush(); → update
System.out.println("Break point.");
transaction.commit(); → commit
```

3. Student student = null;

```
student = (Student) session.get(Student.class, new Long(1));
student.setStudentName("Updated");
session.flush(); → update
System.out.println("Break point.");
student.setStudentName("N@IE");
transaction.commit(); → update/commit
```

B110702
Tuesday

(1) .getEntityName();

→ It Returns the Entity Name for a Persistent Entity (Student POJO)

Student student = null;

```
student = (Student) session.get(Student.class, new Long(1));
System.out.println("..."+session.getEntityName(student));
```

→ It will give the Total package.

(2) .getIdentifier();

→ Return the identifier of an ~~object~~ Entity instance cached by the Session, or throw an exception if the instance is transient or associated with a different Session.

```
Student student = (Student) session.get(Student.class,
new Long(1));
```

```
System.out.println("..."+session.getIdentifier(student));
```

→ Here studentNo value is returned because it is an identifier for the Student object (studentNo is there in ID tag)

(3). getTransaction();

Long());

→ Get the transaction instance associated with this session.

1. Session. beginTransaction();

— Transaction transaction = session.beginTransaction();

2. Session. getTransaction(). commit();

↓
Verb

↓
Noun

— transaction. commit();

3. Session. getTransaction(). rollback();

— transaction.rollback();

13/10/09
Tuesday (4). isConnected(): It checks whether session is connected or not.

(5). isOpen(): Checks whether session is still open or not.

(6). merge():

Student student1 = new Student();

Student student2 = new Student();

student2.setStudentId(new Long(1));

student1 = (Student) session.get(Student.class, new Long(1));

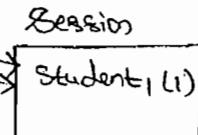
session.update(student2); // It does not work

1. student1 = (Student) session.get();

After step1

2. session.update();

↓
(1 → same identifier)



Because identifier1 is already exist.

→ We will get Exception as NonUniqueObjectException:

A different object, with the same identifier value was already associated with the session: [edu.model.Student # 1]

→ In how many ways can we merge objects?

→ Where as in this case use merge() method

session.merge(student2); // It is correct.

→ Copy the state of the given object onto the persistent object with the same identifier. If there is no persistent instance currently associated with the session, it will be loaded.

→ Return the persistent instance:

If the given instance is unsaved, save a copy of and return it as a newly persistent instance.

→ The given instance does not become associated with the session.

(7) - persist():

→ Make a transient instance persistent. It does not return the identifier value and it works same as save() whereas save return identifier value.

```
Student student = new Student();
student.setName("R@It");
session.persist(student);
```

(8) - reconnect():

→ Reconnect to the given JDBC connection. This is used by applications which require long transactions and use Application-supplied connections.

Syntax:

session.reconnect(connection);

JDBC Connection How do you create in Hibernate Session?

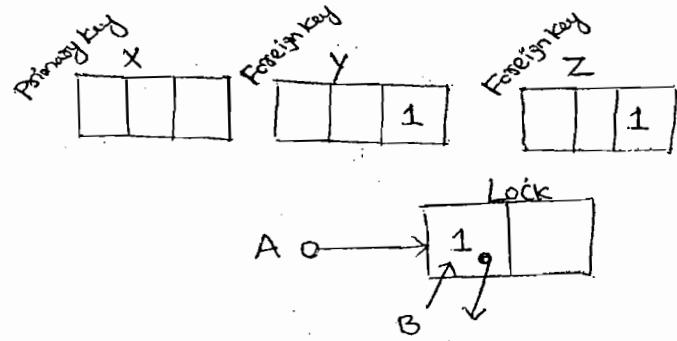
Session session = SESSION_FACTORY.openSession(connection);

Note: If we are having connection object and how can we associate connection with Session

(19) · lock();

- Obtain the specified lock level upon the given object - This may be used to perform a version check (LockMode·READ), To upgrade to a pessimistic lock (LockMode·UPGRADE), or To simply reassociate a transient instance with a session. (LockMode·NONE)

```
Student student = new Student();
student.setStudentNo(new Long(1));
session.lock(student, LockMode.UPGRADE);
System.out.println("Breakpoint"); // Try To update the DataBase Record)
```



- Whenever we update the records in the DataBase first we check is there any record in the lock table or not. If there is no record and insert the record and start updating in X,Y,Z tables.
- If there is a record, we display a message to the user [some other user is updating]. After updating all the records delete the record from the lock table.

(20) · refresh();

- DataBase data is synchronized with a session.
- Re-read the state of the given instance from the underlying Database.
- It is inadvisable to use this to implement long-running sessions that span many business tasks.

Student student=null;

14/10/17
Wednesday

```
sop(" StudentName "+ student.getStudentName());  
sop(" Break point : update DataBase Record ");  
Session.refresh(student);  
sop(" studentName "+ student.getStudentName());
```

(2) .replicate();

- The Detached object will be ReAttach to the Session means it will become persistent state.
- Persist the state of the given Detached instance, reusing the current identifier value.

```
Student student = null;  
student = (Student) session.get(Student.class, new Long(1));  
session.evict(student);  
// session.replicate(student, ReplicationMode.LATEST_VERSION);  
student.setStudentName(" update ");  
transaction.commit();
```

LATEST-VERSION :

- When a row already exists, choose the latest version.

Imp



How To Compare Two Versions Of Same Project:

- * Select Both the Projects then Right click
- * Select Compare With
- * Then Select Each other.

Imp



How To Get The Previous Local changes :

- * Right click on the file
- * Then Select Compare With
- * Then Select Local History

H

1

14/11/09
Wednesday

HQL / Criteria / Native SQL

- Till Now we are getting only one Record in the form of Java object using session.get() and session.load() etc.
- If we want to retrieve more than one record in the form of List object we can use HQL, Criteria and NativeSQL.
- In JDBC, After Executing the Query it returns ResultSet and we need to prepare the List object which contains Each row in the form of one Student object.

In JDBC :

```
1. con  
2. st / ps  
3. rs  
4. List studentList = new ArrayList();  
   Student student = null;  
   while (rs.next())  
   {  
     student = new Student();  
     student.setXXX (rs.getXXX());  
     -----  
     -----  
     studentList.add (student);  
   }  
   return studentList;
```

- In Hibernate we can get List object directly which contain each row in the form of Student object and All the above JDBC Logic is done by Hibernate internally.

How To Execute The Queries :

1. HQL :

```
String stuQuery = "From Student";  
Query query = session.createQuery(stuQuery);
```

2. Criteria

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

[Internally it generates SELECT * FROM STUDENT]

List studentList = criteria.list();

3. Native SQL:

String stuQuery = "SELECT * FROM STUDENT";

SQLQuery sqlQuery = session.createSQLQuery(stuQuery);

List studentList = sqlQuery.list();

(OR)

Query query = session.createSQLQuery(stuQuery);

List studentList = query.list();

HQL

① (1). HQL is just like SQL and The Main difference is Instead of Table Name we give the POJO Name and Instead of Column name we give the Property name, POJO and property are CaseSensitive and Keywords are not CaseSensitive.

② (2)- If we are writing a SQL Query for a DataBase it

- might work (or) might not work in other Databases.

If we are writing a HQL Query it will work in all the Databases which are supported by Hibernate.

③ (3)- Basic insert Query Execution is not possible using HQL

INSERT INTO STUDENT VALUES (-----);

If we are Selecting all the data from one table and inserting to another table those type of insert queries we can execute using HQL but most of the cases we don't use these also in the projects.

INSERT INTO STUDENTCOPY SELECT * FROM STUDENT

- (4) Non Select Queries are must be in Transactional State But it is not related to HQL it is the part of Basic DataBase Concepts.
- (5) We Cannot perform RDBMS operations Because HQL is just like SQL and it is not like PL/HQL It is not a valid point.
- (6) Using HQL we can execute Aggregate functions, It is not a valid point because If it is possible with SQL most of the things which will be possible with HQL.

1. FROM

SELECT * FROM STUDENT [SQL]

from Student [HQL]

SPECIFIED COLUMNS:

SELECT SNO, SNAME FROM STUDENT

Select student.studentNo, student.studentName

from Student student

2. WHERE

SELECT * FROM STUDENT WHERE SNO >? [SQL]

from Student where studentNo >? studentNo /? VarName [HQL]

3. ORDER BY

SELECT * FROM STUDENT WHERE SNAME =? ORDER BY SNO ASC; [SQL]

from Student student

where student.studentName =? name order by student.studentNo asc;

[HQL]

4. AND

SELECT * FROM STUDENT WHERE (SNO >? AND SNAME >=?) [SQL]

from Student student

where (studentNo >? and studentAge >=?)

[HQL]

Note: Select All the Records from the DataBase Using HQL

5. SELECT * FROM STUDENT;

```
Session session = SessionUtil.getSession();  
String strQuery = "From Student";  
Query query = session.createQuery(strQuery);  
List studentList = query.list();
```

Note: studentList contains N number of Student Objects,
Each student object contains one Record (1 row) data.

6. Session session = SessionUtil.getSession();

```
Transaction transaction = session.beginTransaction();
```

```
String stuQuery = " delete from Student  
where studentNo = ? no ");
```

```
Query query = session.createQuery(stuQuery);  
query.setLong("no", new Long(1));  
query.executeUpdate();  
transaction.commit();
```

Structs - config.xml : [Important Tags] in Structs

1. < global-exception key, Path, Scope, type />
2. < Action scope, validate />
3. < forward name = "naresh" path = "/naresh.do (or) /naresh.jsp" />

```
<forward path = "/naresh.do ? paramName = paramValue" />
```

Eg:
<forward path = "/naresh.do ? origin = register" />

Note: If we need to pass Two Parameters and Values we can use them

```
<forward path = "/naresh.do ? origin = register & param1 =  
param2 = paramValue" />
```

```
<forward name = "success" path = "/naresh.vm" />
```

```
<forward ..... redirect = "true" />
```

4. <!-- Controller Settings -->

```
<controller>  
<set-property property = "nocache" value = "true" />  
<set-property property = "processorClass"  
value = "edu.processor.NareshProcessor" />  
</controller>
```

5. <!-- Message Resources -->

```
<message-resources parameter = "edu.util.ApplicationResources" />  
<plug-in className = "org.apache.struts.validators.ValidatorPlugIn">  
<set-property property = "pathnames"  
value = "/WEB-INF/validator-roles.xml",  
value = "/WEB-INF/validation.xml" />  
</plug-in>  
<plug-in className = "org.apache.struts.tiles.Tilesplugin" />  
<set-property property = "definitions-config"  
value = "/WEB-INF/tiles-defs.xml" />  
<set-property property = "definitions-debug" value = "2" />  
<set-property property = "definitions-parser-details" value = "2" />  
<set-property property = "definitions-parser-validate" value = "true" />  
</plug-in>
```

use these

20/10/09
Tuesday

Select ... from ... DOME YOUR STUDENT

→ When we need to call `list()` method (or) `iterate()` method on query object.

[`query.iterate()` / `query.list()`]

`query.iterate()` → [If we require the result immediately (in DAO Layer ...)]

`query.list()` → [If we require the result later (in presentation layer (or) servlet)]

→ When we need type cast to Student object (or) object array.

→ If we are selecting all the columns data then type cast to Student and If we are selecting some of the columns (or) If we are applying aggregate function then type cast to Object array (or) the require data type.

(I).

String stuQuery = " from Student";

Query query = session.createQuery(stuQuery);

Iterator iterator = query.iterate();

while (iterator.hasNext())

{

 Student student = (Student) iterator.next(); [X]

 // Object object[] = (Object[]) iterator.next(); [X]

}

→

(II)

→ Here we are selecting all the columns data so type cast to Student and don't type cast to Object array

(II).

String stuQuery = " SELECT studentNo, studentName
 from Student";

Query query = session.createQuery(stuQuery);

```

Iterator iterator = query.iterator();
while (iterator.hasNext())
{
    // Student student = (Student) iterator.next(); [x]
    Object object[] = (Object[]) iterator.next(); [y]
}

```

→ Here we are giving the column names so type cast to Object[] and don't type cast to Object Student.

Note: Here we get N number of records and each record will be available in one Object[], If there are N number of records there will be N number of Object[]'s.

(III).

```

String stuQuery = "SELECT studentId from Student";
Query query = session.createQuery(stuQuery);
Iterator iterator = query.iterator();
while (iterator.hasNext())
{
    // Student student = (Student) iterator.next(); [x]
    // Object object[] = (Object[]) iterator.next(); [x]
    Object object = (Object) iterator.next(); [y]
    Long studentId = (Long) iterator.next(); [y]
}

```

→ Here we are giving one column name so type cast to Object (or) the required data type and don't type cast to Object[] (or) Student object.

(IV).

```

String stuQuery = "SELECT max(studentId) from Student";
Query query = session.createQuery(stuQuery);
Iterator iterator = query.iterator();
if (iterator.hasNext())
{
    // Student student = (Student) iterator.next(); [x]
}

```

Object object = (Object) iterator.next(); [y]

Long studentNo = (Long) iterator.next(); [y]

}

→ Here we are giving one aggregate function column name so type cast to Object (or) the required data type and don't type cast to Object[] (or) Student object. We are using if because there will be only one Record.

(V) .

String stuQuery = "SELECT max(studentNo), min(studentNo)
from Student";

Query query = session.createQuery(stuQuery);

Iterator iterator = query.iterator();

if (iterator.hasNext())
{

// Student student = (Student) iterator.next(); [x]

Object object[] = (Object[]) iterator.next(); [y]

SOP(object[0] + " " + object[1] + " " + object[2]);

for (int i=0; i < object.length; i++)

SOP(" " + object[i]);

// Object object = (Object) iterator.next(); [x]

// Long studentNo = (Long) iterator.next(); [x]

}

→ Here we are giving two Aggregate functions and don't type cast to Object (or) Student object. We are using if because there will be only one record and inside we can directly use object[0], object[1], (or) for loop.

Working with Positional Parameter:

String hql = " from Student student where student.studentNo > ?

and student.studentName like ? ";

Query query = session.createQuery(hql);

query.setLong(0, new Long(1));

query.setString(1, "studentName%");

→ Here we are giving ? and while we set values to the query the position should start with 0 (zero). In JDBC Prepared Statement it should be with 1.

→ we should not use positional parameter in the project because in future if we don't keep studentNo condition in ~~filter~~ the query. so many places the changes are required ... like 2 should become 1 n should become n-1 etc...

→ we use Named Parameter in the project.

Named parameters:

→ Named Parameter will be any name which will be given in the query instead of ?

Syntax: _____ : anyName [There should not be any space between : and anyName]

→ AnyName which starts with : is named parameter.

Eg:

String hql = " from Student student where student.studentNo > :No and student.studentName like :Name ";

Query query = session.createQuery(hql);

query.setString ("tName", "Kareesh %");

List studentList = query.list();

→ Before we execute the query we need to set the values to the query using Named parameter.

Advantage:

→ Advantage is in future if we are not using studentNo condition in the query there will be changes in two places....

Note:

We can use the positional parameter and the Named parameter combination in the query whenever the combination is given first all the positional parameter should be given and all the Named parameters should be given. But first we should not give Namedparameter and then Positional parameter.

1. String hql = "from Student student where

student.studentNo > ? and

student.studentName like ?Name";

query.setLong (0, new Long(1));

→ [y]

2. String hql = " from Student student where

student.studentNo > :NO and

student.studentName like ? ";

query.setLong ("NO", new Long(1));

→ [x]

Note:

* In the project we never use the Combination

* In project always we use Named parameter

Note: 1. To Execute the Update (or) Delete Queries we call ExecuteUpdate() method.

2. To Execute the Select Queries we call List() (or) Iterate() method.

21/10/09
Wednesday
Ascending And Descending order

ASC:

String stuQuery = " FROM Student student ORDER BY student.studentNo ASC";

Query query = session.createQuery(stuQuery);

Iterator iterator = query.iterator();

DESC:

String stuQuery = " FROM Student student ORDER BY student.studentNo DESC";

Query query = session.createQuery(stuQuery);

Iterator iterator = query.iterator();

Note:

SELECT SNO, SNAME FROM STUDENT;

↳ Projection

Note:

→ If there are 100 records in the DataBase if we require the result from 20th record to 30th record like Pagination Concept.

→ We need to set, what is the first record and what is the max record.

Query query = session.createQuery(" from Student");

query.setFirstResult(20);

query.setMaxResults(10); ↳ n+1 - n

now To implement the Pagination Concept in Struts we can use display tag, To implement in JSP we can use the Existing tld Library & which will be given by third party.

How To Retrieve The UniqueResult:

- To retrieve the unique results we need to call `uniqueResult()` method,
- It returns a single instance that matches the query,
(or) null if the query returns no results

Syntax:

```
String hql = "from Student student where  
student.studentNo = 1;"
```

```
Query query = session.createQuery(hql);
```

```
Student student = (Student) query.uniqueResult();
```

```
if (student != null)
```

```
{
```

```
sop("... StudentNo ... " + student.getStudentNo());
```

```
sop("... StudentName ... " + student.getStudentName());
```

```
}
```

```
else
```

```
{
```

```
sop(" No Record ");
```

```
}
```

1. `INSERT INTO Student VALUES (1, 'N@IT');` [X]

Normal insert Queries are not possible using HQL

2. `INSERT INTO StudentTest SELECT * FROM Student` [Y]

While inserting the data, if we are Selecting the data from another table those type of insert Queries are possible using HQL.

3. We don't implement this case also in the project

String studentQuery = "INSERT INTO StudentTest studentNo,
studentName SELECT student.studentNo,
student.studentName FROM Student student";

int count = session.createQuery(studentQuery).executeUpdate();

→ For this we require 2 POJO (Student, StudentTest) and
2 HBM files [Student.hbm.xml, StudentTest.hbm.xml]

→ StudentTest and StudentTest.hbm.xml are same as Student
and Student.hbm.xml [only table will be different in the
hbm file]

Criteria :

Query: SELECT * FROM STUDENT;

I. Criteria criteria = session.createCriteria(Student.class);
SELECT * FROM STUDENT;

II. Criterion criterion = Expression.eq("studentNo", new Long(1));
SNO = 1.

III. criteria.add(criterion);
SELECT * FROM STUDENT WHERE SNO = 1

IV.
List studentList = criteria.list();
List studentList = session.createCriteria(Student.class)
• add(Expression.eq("studentNo",
new Long(1))).list();

→ In HQL, we write SQL Syntax in Java style. But in
Criteria it is completely OOP style, we don't write the
query we build the query using methods.

→ In Criteria we can execute only Select queries and we cannot execute Non-Select queries.

Note: In project, If there is a select query most of the cases we prefer Criteria. If there is no Select query [update, delete etc.] we use HQL.

22/10/09

Thursday Criterion:

Ctrl+Spacebar

- To prepare the condition, we use Criterion object and we Add the Criterion object to the Criteria before Executing the queries.
- All the conditions is prepared using Expression class and All the Expression classes implements Criteria.

Expression:

- To prepare the conditions, we call the methods on Expression class and all those methods are static, Expression class Extends Restrictions and most of the methods are there in Restrictions, so that we can call these methods from Expression.

Expression class Methods:

MethodName	Expression	Condition
1. eq	Simple Expression	=
2. ne	"	<> (b8) !=
3. like	"	like (Case Sensitive)
4. gt	"	>
5. lt	"	<
6. le	"	<=
7. ge	"	>=
8. ieq	IdentifierEq Expression	=
9. ilike	Ilike Expression	like (Case In-Sensitive)
10. between	Between Expression	between xx and yy
11. in	In Expression	in(xx)

Method Name	Expression	Condition
12. isNull	Null Expression	is null
13. isNotNull	NotNull Expression	is not null
14. and	Logical Expression	and
15. or	"	or
16. isEmpty	Empty Expression	checks empty or not
17. isNotEmpty	NotEmpty Expression	checks non-empty or not
18. sizeEq	Size Expression	=
19. sizeNe	Size Expression	<>

(Q) Query : Select SNO, SNAME from STUDENT where SNO = 1 ;

I. Student student = (Student) session.load(Student.class, new Long(1));
(or)

Student student = (Student) session.get(Student.class, new Long(1));
(or)

I. Criteria criteria = session.createCriteria(Student.class);
Select SNO, SNAME from STUDENT

II. Criterion criterion = Expression.eq(1L);
SNO = 1

III. criteria.add(criterion);

Select SNO, SNAME from STUDENT where SNO = 1

IV. List studentList = criteria.list();

→ If studentNo is primarykey then list contains the object.

→ In this case we no need pass the property, Because it takes ID column property automatically
(or)

I. Criteria criteria = session.createCriteria(Student.class);
Select SNO, SNAME from STUDENT

II. Criterion criterion = Expression.eq("studentNo", 1L);
SNO = 1

III. criteria.add(criterion);

IV. List studentList = criteria.list();

→ If studentNo is primary key then list contains one object

(2). Query : select SNO, SNAME from STUDENT where SNO < ?

List studentList = session.createCriteria(Student.class)

- add(Expression.property("studentNo"), new Long(2))
- list();

(7). ↗

(3). Query : select SNO, SNAME from STUDENT SNAME like ?

List studentList = session.createCriteria(Student.class)

- add(Expression.like("studentName", "N@IT"))
- list();

(4). Query : Select SNO, SNAME from STUDENT where SNAME like ?

→ We required the results with case insensitive

List studentList = session.createCriteria(Student.class)

- add(Expression.ilike("studentName", "N@IT"))
- list();

(8). ↘

I.

→ Then Hibernate generates query as :

Select SNO, SNAME from STUDENT where lower(SNAME)
like ?

(5). Query : select SNO, SNAME from STUDENT where SNAME is null

List studentList = session.createCriteria(Student.class)

- add(Expression.isNull("studentName"))
- list();

II

(6). Query : select SNO, SNAME from STUDENT where SNAME like '%Araresh%'

List studentList = session.createCriteria(Student.class)

- add(Expression.byRestriction("%Araresh%"))
- SNAME like '%Araresh%'")
- list();

Hibernate Generated Query:

```
Select this_.SNO as SNO_0_, this_.SNAME as SNAME_0_ from STUDENT this_ where this_.SNAME like 'Naresh%';  
? alias ? <-> this_
```

Note: If we want to write the query using SQL syntax, we can do it using "Restriction" method.

(7) Query: Select SNO, SNAME from STUDENT where SNO > ? and SNAME like ?

```
List studentList = session.createCriteria( Student.class )
```

- add(Expression.gt("studentNo", new Long(1)))
- add(Expression.like("studentName", "Naresh%"))
- list();

(8) Query: Select SNO, SNAME from STUDENT where (SNO > ? or SNAME like ?)

```
I. List studentList = session.createCriteria( Student.class )
```

- add(Expression
- or(Expression.gt("studentNo", new Long(1)), Expression.like("studentName", "Naresh%"))
- list();

(or)

```
II. Criteria criteria = session.createCriteria( Student.class );
```

```
SELECT * FROM STUDENT;
```

II.

2.1 // Criteria I, Criteria II

```
Criterion criterionOne = Expression.gt( "studentNo",  
new Long(1));  
SNO > ?
```

```
2.2 Criterion criterionTwo = Expression.like( "studentName",  
"Naresh%" );  
SNAME like ?
```

2.3 // Criteria or Condition

Logical Expression . orExp = Expression.or(criterionOne , criterionTwo)

SNO > ? OR SNAME like ?

III. criteria • add (orExp);

select SNO, SNAME from STUDENT

where (SNO > ? OR SNAME like ?)

IV. List studentList = criteria.list();

23/10/09
Friday

(Q). Query: select SNO, SNAME from STUDENT where SNO=?
and (SNAME like ? OR SNAME like ?)

I. List studentList = session.createCriteria(Student.class);

• add (Expression.eq("studentNo",
new Long(1)))

• add (Expression.or(Expression.
like("studentName", "%Naresh"),
Expression.like("studentName",
"Naresh%")))

• list();

(or)

I. Criteria criteria = session.createCriteria(Student.class);

SELECT * FROM STUDENT;

II

2.1. Criterion criterion1 = Expression.eq("studentNo", new Long(1))

SNO = ?

2.2 Criterion criterion2 = Expression.like("studentName",
"%Naresh");

SNAME like ?

2.3 Criterion criteria = Expression.like("studentName", "Unnesh");
SNAME like ?

3.1 Criterion criteria = Expression.or(criteria2, criteria3);
SNAME like ? or SNAME like ?

3.2 criteria.add(criteria1);
Select * from STUDENT where SNO = ?

3.3 criteria.add(criteria4);
Select * from STUDENT where SNO = ? and
(SNAME like ? or SNAME like ?)

4. List studentList = criteria.list();

Pagination:

→ If there are 10 records, If we required from 5th record to 7th record.

Criteria criteria = session.createCriteria(Student.class);
criteria.setFirstResult(4);
criteria.setMaxResults(3);
studentList = criteria.list();

Note: Here first result is 4 then it starts from 4+1=5
and it will come till 4+3 (maxResult) = 7

How To Get The Unique Results:

Student student = (Student) session.createCriteria(Student.class)
• add(Expression.eq("studentNo", new Long(1)))
• uniqueResult();

Apply Order By :

Query : select * from STUDENT order by SNO asc

```
List studentList = session.createCriteria(Student.class)
    .addOrder(Order.desc("studentNo")).list();
(OR)
```

I. Criteria criteria = session.createCriteria(Student.class);

Select * from STUDENT

II. Order order = Order.asc("studentNo");

SNO asc

(OR)

// Order order = Order.desc("studentNo");

// SNO desc

III. criteria.addOrder(order);

Select * from STUDENT order by SNO asc

IV. List studentList = criteria.list();

Projections:

How Apply the projection:

Query: SELECT SNO, NAME ^{projection} FROM STUDENT

→ If we are Selecting the specified columns from the table
we call the concept as Projections.

→ To select the specified columns we need to create a projection object and projection objects should be added to projectionList and the list should be added to criteria before we execute the query.

→ If there is only one column we are Selecting, we can prepare projection object and we can add projection directly to criteria.

Syntax:

- To create the projectionList object, call the static method on the projections class.

```
ProjectionList projectionList = Projections.projectionList();
```

Criterion	— Projection
Expression	— Projections
SimpleExpression	-----

Projection classes:

- While calling projections class methods finally it returns projection object.
- projection is an interface and there are different implemented classes for projection interface.

Method Name	Projection Class	Condition
rowCount	RowCountProjection	count(*)
count	CountProjection	count
distinct	Distinct	distinct
max	AggregateProjection	max
min	AggregateProjection	min
avg	AvgProjection	avg
sum	AggregateProjection	sum
Eq/GroupProjection	SQLProjection	group
id	IdentifierProjection	id column
Property	PropertyProjection	Properties (studentNo, studentName)

- All the above projection classes are implementing Projection interface.

Syntax:

1. Query : select SNO from STUDENT

I. Criteria criteria = session.createCriteria(Student.class);
SELECT * FROM STUDENT

II. Projection projection = Projections.property("studentNo");
SNO

III. criteria.setProjection(projection);
Select SNO from STUDENT

IV. List studentList = criteria.list();

Note: While iterating type cast to object or the corresponding object (Here it is Long), Because we are specifying the columns and there is only one column in the query.

→ If there are more than one column while iterating
type cast to Object array.

3.

Note: If we want to apply aggregate functions (or) If we want to select the specified column in the Criteria we need to apply projections.. projection is just like Expression

24/10/09
Saturday

2. Query: SELECT SNO, SNAME FROM STUDENT

List studentList = session.createCriteria(Student.class)

- setProjection(Projections.projectionList())
- add(Projections.property("studentNo"))
- add(Projections.property("studentName"))
- list();

(or)

4.

I. Criteria criteria = session.createCriteria(Student.class);

SELECT * FROM STUDENT

J
J
J

- II. ProjectionList projectionList = Projections.projectionList();
- 2.1. Projection projection1 = Projections.property("studentNo");
SNO
 - 2.2. Projection projection2 = Projections.property("studentName");
SNAME
 - 2.3. ProjectionList.add(projection1);
projectionList.add(projection2);
SNO, SNAME
- III. criteria.setProjection(projectionList);
- SELECT SNO, SNAME FROM STUDENT;

- IV. List studentList = criteria.list();

→ Here we are selecting two columns. So while iterating we need to take object array.

3. Query: select count(*) from STUDENT

List studentList = session.createCriteria(Student.class).
• setProjection(Projections.rowCount()).list();
(08)

- I. Criteria criteria = session.createCriteria(Student.class);
Select * from STUDENT
- II. Projection projection = Projections.rowCount();
count(*)
- III. criteria.setProjection(projection);
select count(*) from STUDENT

- IV. List studentList = criteria.list();

4. Query: select max(SNO), min(SNO), avg(SNO) from STUDENT

I. Criteria criteria = session.createCriteria(Student.class);

II.

2.1. projectionList.add(Projections.max("studentNo"));
max(SNO)

2.1. projectionList.add(Projections.min("studentNo"));
min(SNO)

2.3. projectionList.add(Projections.avg("studentNo"));
avg(SNO)

III. criteria.setProjection(projectionList);
Select max(SNO), min(SNO), avg(SNO) from STUDENT

IV. List studentList = criteria.list();

5. Query : select count(distinct SNAME) from STUDENT

I. Criteria criteria = session.createCriteria(Student.class);

II. Projection projection = Projections.countDistinct("studentName");

III. criteria.setProjection(projection);

IV. List studentList = criteria.list();

Note: If we want to apply only distinct. then call
distinct() method.

6. Query : select SNO, SNAME from STUDENT group by SNAME

SNO	SNAME
1.	N@IT1
2.	N@IT2
3.	N@IT1
4.	N@IT2

I. Select SNAME from STUDENT group by SNAME

Then it returns 2 records.

II. Select SNO, SNAME from STUDENT group by SNAME

Then it has to return 2 SNAME's with 4 SNO's, so it
gives Error.

Whenever we apply group by, if we select any other column
we need to apply aggregate function as,

Select count(SNO), SNAME from STUDENT group by SNAME

7. Query : Select avg(SNO), SNAME from STUDENT group by SNAME

I. Criteria criteria = session.createCriteria(Student.class);

II. ProjectionList projectionList = Projections.projectionList();

2-1. Projection projection1 = Projections.avg("studentNo");

2-2. Projection projection2 = Projections.groupProperty("studentName");

2-3. projectionList.add(projection1);

projectionList.add(projection2);

III. criteria.setProjection(projectionList);

IV. List studentList = criteria.list();

NativeSQL

→ We can Execute the SQL Queries directly in Hibernate using NativeSQL.

→ To execute a query, we need to prepare a SQL query object
Then we need to call a list() method.

Syntax:

SQLQuery query = session.createSQLQuery(query);

List studentList = query.list();

Note: Here, query will be in SQL syntax means TABLE
Name and COLUMN name.

1. Query : select * from STUDENT

I. String sql = select { student.* } from STUDENT student
 { alias }

SQLQuery query = session.createSQLQuery(sql);
(OR)

Query query = session.createSQLQuery(sql);

II. query.addEntity("student", Student.class);

Note : Before execute the query, table should be mapped with the POJO.

→ Here, we are selecting all the columns - so we have to use addEntity() If it single column's - Then it should be mapped with addScalar().

III. List studentList = query.list();

2. Query : select avg(SNO) from STUDENT

String sql = "select avg(student.SNO) as avgSno from STUDENT student";

I. SQLQuery query = session.createSQLQuery(sql);

II. query.addScalar("avgSno", Hibernate.DOUBLE);

III. List studentList = query.list();

NamedQuery

→ If the query is having Union UnionAll Condition and Query logic is complex or If the query is having the conditions and if we are changing those conditions we can move the query to hbm file and we apply NamedQuery concept.

26/10/09
Monday

→ To retrieve the query , we pass the Query name to getNamedQuery() method and we create query object , Here also we write property name in the hbm file for the query .

Syntax :

1. Query : Select SNO, SNAME from Student where SNO > ?

HBM File :

```
<query name = " HQLStudent1 " >  
<! [ CDATA [ select student . studentNo, student . studentName  
from Student student where student . studentNo >  
? : studentNo ] ] >  
</query>
```

Java Main file :

```
Query query = session . getNamedQuery ( " HQLStudent1 " );  
query . setLong ( " studentNo " , new Long ( 1 ) );  
List studentList = query . list ();
```

Note : While iterating , we need to take object array . Because in the query we are selecting specified columns . If we are selecting all the columns while iterating , we can take student object .

2. Query : select * from Student student ;

HBM File :

```
<sql-query name = " HQLStudent2 " >  
<return alias = " student " class = " edu . model . Student " />  
<! [ CDATA [ select * from Student student ] ] >  
</sql-query>
```

Main Java file :

```
Query query = session . getNamedQuery ( " HQLStudent2 " );  
List studentList = query . list ();
```

Note : The results should be mapped with the POJO using return tag , while iterating we can take student object and if we are not able to map then we need to take Object .

3. Query: select SNO, SNAME from STUDENT

HBM File:

```
<sql-query name = "HQLStudent3">
<return alias = "student" class = "edu.model.Student">
<return-property name = "studentNo" column = "SNO"/>
<return-property name = "studentName"/>
<return-column name = "SNAME"/>
</return-property>
</return>

<! [ CDATA [ select student.SNO, student.SNAME from
          Student student ] ] >
</sql-query>
```

5.

Main Java File:

```
Query query = session.getNamedQuery ("HQLStudent3");
```

```
List studentList = query.list();
```

Note: Here we are giving the column names in the query. So the results of the column should be mapped with properties, we are mapping each column with the property because we are specifying the columns in the query.

→ Each property should be mapped with return-property tag and if we want we can give the column name separately using return-column tag.

27/10/
Tues

4. Query: select avg(student.NO) from Student

HBM File:

```
<sql-query name = "HQLStudent4">
<! [ CDATA [ select avg (student.SNO) as avgStudent
          from Student student ] ] >
</sql-query>
```

Main Java File:

```
Query query = session.getNamedQuery("HQLStudent4");
```

```
List studentList = query.list();
```

Note: Here we get one row of result, so use if condition
and type cast to object or the required data type (BigDecimal).

5. Query: select avg (SNO) from Student

HBM File:

```
<sql-query name="HQLStudents5">
```

```
<return-scalar column="avgStudent" type="double"/>
```

```
<![CDATA [select avg (student.SNO) as avgStudent  
from Student student ]]>
```

```
</sql-query>
```

Main Java file:

```
Query query = session.getNamedQuery("HQLStudents5");
```

```
List studentList = query.list();
```

Note: For the average, it gives the data type as BigDecimal
and we are changing it to double using return-scalar.
Using return-scalar we can define the results data type.

Relationships

STUDENTDEV4 • STUDENT

SNO	NUMBER(19)	IDX-1
SNAME	VARCHAR2(25)	

STUDENTDEV1 • STUDENTXTRA

SNO	NUMBER(19)	IDX-1
SAGE	VARCHAR2(10)	

One - To - One :

→ for the above Table design we require Two patos

```
public class Student
```

```
{ private Long studentNo = null;
```

```
private String studentName = null;
```

public class StudentXtra

{ private Long studentNo = null;

private String studentAge = null;

getter & setter;

}

Ctrl +

Ctrl + H

Ctrl + F6

Ctrl + O : open a file

→ We require Two HBM Files

1. student.hbm.xml :

studentNo — SNO → ID → Sequence

studentName — SNAME → Property

2. studentXtra.hbm.xml :

studentNo — SNO → ID → Sequence

studentAge — SAGE → Property

cfg.xml :

<mapping resource = "edu/mappings/student.hbm.xml"/>

<mapping resource = "edu/mappings/studentXtra.hbm.xml"/>

Main.java :

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III. Student student = new Student();

StudentXtra studentXtra = new StudentXtra();

3.1. student.setStudentName("Naresh@IT");

3.2. studentXtra.setStudentAge("7");

IV.

4.1. session.save(student);

4.2. session.save(studentXtra);

Note: The above example is wrong. In the DB Level there is a relation between the tables. But at the Hibernate side we are not specifying any relationship in the POJO's and in the HBM files and we should not call save() method two times and we should not persist the object independently.

→ For the Above Table design ---- We require Two POJO's.

```
public class StudentXtra  
{  
    private Long studentNo = null;  
    private String studentAge = null;  
    getters & setters;  
}
```

```
public class Student  
{  
    private Long studentNo = null;  
    private String studentName = null;  
    private StudentXtra studentXtra = null;  
    getters & setters;  
}
```

→ We require TWO HBM files .

studentXtra.hbm.xml :

studentNo — SNO → ID → Sequence

studentAge — SAGE → property

student.hbm.xml :

<id name = "studentNo" type = "java.lang.Long" column = "SNO" >

<generator class = "foreign" >

<param name = "property" > studentXtra </param>

</generator >

</id >

studentName — SNAME → property

<one-to-one name = " studentXtra" class = "edu.model.StudentXtra" >

cfg.xml :

<mapping resource = "edu/mappings/student.hbm.xml" />

<mapping resource = "edu/model/StudentXtra.hbm.xml" />

Main.java :

```
I. Session session = SessionUtil.getSession();  
II. Transaction transaction = session.beginTransaction();  
III.  
3.1. StudentXtra studentXtra = new StudentXtra();  
studentXtra.setStudentAge("14");  
  
3.2.1  
Student student = new Student();  
student.setStudentName("Akash It");  
  
3.2.2  
student.setStudentXtra(studentXtra);
```

IV. session.save(student);

28/10/09
Wednesday

Note: Here studentXtra is independent object. So, StudentXtra is a normal POJO and studentXtra.hbm is normal hbm.

2. Here student is dependent object. So, Student POJO contains studentXtra reference and student.hbm contains Foreign Generator class and it contains one-to-one tag.

**
Note:

1. In Student POJO studentXtra is the reference Variable for StudentXtra class. So, specify the same reference Variable for the Generator class and specify the same reference Variable in one-to-one tag.

2. student is a dependent object - so, we set studentXtra details StudentXtra, Student details to Student, studentXtra object to student and finally need to persist student object at run time. Hibernate generates two insert queries for both the tables

Persisting studentXtra in the DataBase:

public class Student

{ private Long studentNo = null;

private String studentName = null;

getters & setters

3

public class StudentXtra

{ private Long studentNo = null;

private String studentAge = null;

private ~~String~~ ~~student~~

private Student student = null;

getters & setters

3

→ We require Two HBM Files

student.hbm.xml :

studentNO — SNO → ID → sequence.

studentName — SNAME → property

studentXtra.hbm.xml :

<id name = "studentNo" type = "java.lang.Long" column = "SNO">

<generator class = "foreign">

<param name = "property"> studentXtra </param>

</generator>

</id>

studentAge — SAGE → property

<one-to-one name = "student" class = "edu.model.Student"/>

cfg.xml :

<mapping resource = "edu/mappings/student.hbm.xml" />

<mapping resource = "edu/mappings/studentXtra.hbm.xml" />

Main.java :

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III.

3.1. Student student = new Student();

student.setStudentName("Nameesh It");

3.2.1.

StudentXtra studentXtra = new StudentXtra();

studentXtra.setStudentAge("14");

3.2.2.

studentXtra.setStudent(student);

IV. session.save(studentXtra);

One - To - Many : [Primary Key - To - Foreign Key]

SCOTT • UNCOURSE

CNO	NUMBER(19) <small>idx-1</small>
CNAME	VARCHAR2(25)

SCOTT • STUDENT

SNO	NUMBER(19) <small>idx-1</small>
SNAME	VARCHAR2(25)
XNO	NUMBER(19)
IDX	NUMBER(10)

one-to-many [Primary key -to- Foreign key]

COURSE → STUDENT

→ We are specifying the relation from course to student and student is independent and course is dependent then finally we need to persist course object in the DataBase so, course.hbm should contain one-many relation tags etc

public class Student

{

private Long studentNo;

private String studentName;

getters & setters;

}

29/10/20
Tuesday

```
public class Course
{
    private Long courseNo;
    private String courseName;
    private Student[] studentArray;
    getters & setters;
}
```

student.hbm.xml:

```
studentNo — SNO → ID → sequence
studentName — SNAME → property
```

course.hbm.xml:

```
courseNo — CNO → ID → sequence
courseName — CNAME → property
```

```
<array name = "studentArray">
    <key column = "CNO"/>
    <list-index column = "idx"/>
    <one-to-many class = "edu.model.Student"/>
</array>
```

29/10/09
Thursday

Main.java:

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III.

```
3.1. Student studentOne = new Student();
      Student studentTwo = new Student();
      studentOne.setStudentName("Naresh IT1");
      studentTwo.setStudentName("Naresh IT2");
```

```
Student[] studentArray = new Student[] { studentOne,
                                             studentTwo };
```

```
3.2.1. Course course = new Course();
       course.setCourseName("MCA");
```

3.2.2. course.setStudentArray(studentArray);

Note: < key column = "CNO" />

CNO is student table foreign key column. The primary key of Course table [CNO] will be stored automatically in Student table foreign key column [CNO] by the Hibernate.

UNCOURSE Table Data:

CNO	CNAME
1	MCA
2	MBA

STUDENT Table Data:

SNO	SNAME	CNO	IDX
1	NareshIT1	1	0
2	NareshIT2	1	1
3	NareshIT1	2	0
4	NareshIT2	2	1
5	NareshIT3	2	2

Note:

→ CNO 1 is having relationship with two records and CNO 2 is having relationship with three records and the Each group of records index will start from zero(0) and the index column should be specified using index tag.

< index column = "idx" />

(or)

< list-index column = "idx" />

→ Index (or) The corresponding tag is must for the <array> tag.

<array> tag name should match with the POJO property name.

[< array name = "studentArray" /> \leftrightarrow private Student[] studentArray]

→ Here we are taking the relation from Course to Student means Primary key to Foreign key. So the relationship is One - To - Many.

- We are taking from Course. So, Course is dependent object.
 Then first create independent object and make it as a array. Then create dependent object then set array [] student to course and finally we persist course object in the Database.
- Here Course dependent. So, the course.hbm.xml file should contain <one-to-many> tag.
- One-to-Many means we are taking from one record to many records and the many records is available with some collection object. so, one-to-many always it comes inside a Collection tag.

Working with List:

public class Student

```
  {
    private Long studentNo;
    private String studentName;
    getters & setters
  }
```

public class Course

```
  {
    private Long courseNo;
    private String courseName;
    private List studentList;
    getters & setters
  }
```

Student.hbm.xml :

studentNo — SNO → ID → sequence
 studentName — SNAME → property

course.hbm.xml :

courseNo — CNO → ID → sequence
 courseName — CNAME → property

```
<list name = "studentList" cascade = "all">
    <key column = "CNO"/>
    <index column = "idx"/>
    <one-to-many class = "edu.model.Student"/>
</list>
```

Main.java :

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III.

3.1. Student studentOne = new Student();

Student studentTwo = new Student();

studentOne.setStudentName("Naresh IT1");

studentTwo.setStudentName("Naresh IT2");

List studentList = new ArrayList();

studentList.add(studentOne);

studentList.add(studentTwo);

3.2.1.

Course course = new Course();

course.setCourseName("MCA");

3.2.2.

course.setStudentList(studentList);

IV. session.save(course);



Working with Bag :

→ All the programs are same as List Example and change is in HBM file.

```
<bag name = "studentList" cascade = "all">
```

```
    <key column = "CNO"/>
```

```
    <one-to-many class = "edu.model.Student"/>
```

```
</bag>
```

→ Bag is just like List. But it doesn't contain index i.e means indexed bag is nothing but list...

Working With Set:

public class Student

{ private Long studentNo;

private String studentName;

getters & setters

}

public class Course

{ private Long courseNo;

private String courseName;

private Set studentSet;

getters & setters

}

student.hbm.xml :

studentNo — SNO → ID → sequence

studentName — SNAME → property

course.hbm.xml :

courseNo — CNO → ID → sequence

courseName — CNAME → property

< set name = " studentSet " cascade = " all " >

< key column = " CNO " />

< one-to-many class = " edu.model.Student " />

</set>

Main.java :

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III. 3.1. Student studentOne = new Student();

```

studentOne.setStudentName("Naresh It1");
studentTwo.setStudentName("Naresh It2");
Set studentSet = new HashSet();
studentSet.add(studentOne);
studentSet.add(studentTwo);

```

3.2.1.

```

Course course = new Course();
course.setCourseName("MCA");

```

3.2.2.

```

course.setStudentSet(studentSet);

```

4. session.save(course);

Many-To-One [F.K - To - P.k] [Student -To - Course] :

Course Table Data:

CNO	CNAME
1	MCA

Student Table Data:

SNO	SNAME	CNO
1	N@It1	1
2	N@It2	1

→ There is primary key, Foreign key relation for the above tables

and one course is mapped with two students and how to store a new student in the Database for the existing course.

→ We need to Student in the Database - So, Start the relation from Student to Course and at the table level the relationship is Foreign key -To- Primary key. So, we have to apply many-to-one relationship.

→ Many-To-One means There are many records in the Database [Student] and we need to add the Manyth record in the Database [Means one record] (or) we might be inserting the first record also in the Database.

→ There are Two possibilities, while working with Many-To-one

1. We can Add a New Student for the Existing Course
where as in this case we are inserting one record in
the Student table.

2. We can Add a new Student with a New Course where
as in this case we are inserting one record in Two
Tables [Student Table One record, Course Table One Record]

→ After doing the first scenario, the Database Tables will
be like,

CNO	CNAME
1	MCA

SNO	SNAME	CNO
1	N@IT1	1
2	N@IT2	1
3	N@IT3	1

→ After doing the second possibility, The Database tables
will be like,

CNO	CNAME
1	MCA
2	MBA

SNO	SNAME	CNO
1	N@IT1	1
2	N@IT2	1
3	N@IT3	1
4	N@IT4	2

→ In the above tables, we have inserted one, one record
[Student table one & Course table one with course2]

We think the relation is one-to-one. But still it is
Many-to-one relation only, because the Foreign key is
not unique so that we can add a new student for course2
many times many students will be there.

30/10/09
Friday

Working With Many - To - One:

1. Creating a New Student With a Existing Course:

public class Course

 { private Long courseNo;

 private String courseName;

 getters & setters

}

public class Student

 { private Long studentNo;

 private String studentName;

 private Course course;

 getters & setters

}

Course.hbm.xml :

courseNo — CNO → ID → Sequence

courseName — CNAME → property

Student.hbm.xml :

studentNo — SNO → ID → sequence

studentName — SNAME → property

< many-to-one name = "course" column = "CNO"

class = "edu.model.Course" />

Main.java :

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III. 3.1. Course course = (Course) session.load (Course.class, new Long(1));

3.2.1. Student student = new Student();

student.setStudentName ("N@It3");

3.2.2. student.setCourse (course);

IV. session.save (student);

2. Creating a New Course With a New Student:

```
public class Course {  
    private Long courseNo;  
    private String courseName;  
    getters & setters  
}
```

```
public class Student {  
    private Long studentNo;  
    private String studentName;  
    private Course course;  
    getters & setters  
}
```

course.hbm.xml:

courseNo — CNO → ID → sequence
courseName — CNAME → property

Student.hbm.xml:

studentNo — SNO → ID → sequence
studentName — SNAME → property

<many-to-one name = "course" column = "CNO"
class = "edu.model.Course" cascade = "all" />

Main.java:

- I. Session session = SessionUtil.getSession();
- II. Transaction transaction = session.beginTransaction();
- III.
 - 3.1. Course course = new Course();
course.setCourseName("MCA");
 - 3.2.1. Student student = new Student();
student.setStudentName("N@It3");
 - 3.2.2. student.setCourse(course);
- IV. session.save(student);

Note : If we give the Cascade operation, first the course object is inserted. Then the student object is inserted. If we don't give the Cascade, it cannot insert the parent object first then the child object. To insert the parent object first we give the Cascade operation. → check example with Cascade & Without the Cascade.

How To get the Return Type :

ctrl + o

PSVM()

Goto JAVA2S.COM

{

Class bClass = B.class;

Method[] method = bClass.getMethod();

(6) getAnnotations();

Method method2 = method[0];

SOP (method2.getReturnType());

}

3/1/10/09
Saturday

Working With One-To-one Bi-Direction:

public class Student

```
{ private Long studentNo = null;
  private String studentName = null;
  private StudentXtra studentXtra = null;
  getters & setters }
```

3:

public class StudentXtra

```
{ private Long studentNo = null;
  private String studentAge = null;
  private Student student = null;
  getters & setters }
```

3:

Student.hbm.xml:

studentNo — SNO → ID → sequence

studentName — SNAME → property

```
< one-to-one name = "studentXtra" class = "edu.model.StudentXtra" cascade = "all" />
```

StudentXtra.hbm.xml:

studentNo — SNO → ID → sequence

studentAge — SAGE → property

```
< one-to-one name = "student" class = "edu.model.Student" cascade = "all" />
```

Main.java:

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III.

3.1. Student student = new Student();

student.setStudentName("ManeshIT");

3.2.1. StudentXtra studentXtra = new StudentXtra();

studentXtra.setStudentAge("14");

3.2.2. student.setStudentXtra(studentXtra);

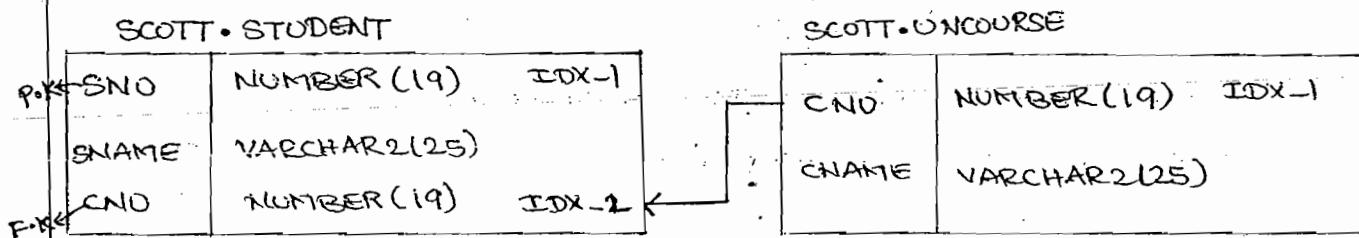
session. save (studentXtra);

Note :

→ Bi-Directional means in both the ways the relationship will be there Means Student contains studentXtra and studentXtra contains student, student.hbm contains one-to-one and studentXtra.hbm contains one-to-one, while persisting the object we associate in the both the ways (or) one way and we can store any object.

Working with Many-To-One [F.o.K - To - P.K] :

→ Foreign key is unique, so we write Many-to-one tag in the hbm file and unique = true. But at the DataBase level, the relationship is one-to-one with Foreign key - To - primary key. If Foreign Key is not unique then it will be many-to-one.



public class Course

{ private Long courseNo;

private String courseName;

getters & setters

3

public class Student

{ private Long studentNo;

private String studentName;

private Course course;

getters & setters

course.hbm.xml:

courseNo — CNO → ID → Sequence.

courseName — CNAME → property

student.hbm.xml:

studentNo — SNO → ID → Sequence.

studentName — SNAME → property

< many-to-one name = "course" column = "CNO"

class = "edu.models.Course" cascade = "all" unique = "true" />

Main.java 1st Way:

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III.

3.1. Course course = new Course();

course.setCourseName("MCA");

3.2.1.

Student student = new Student();

student.setStudentName("N@IT");

3.2.2.

student.setCourse(course);

IV. session.save(student);

Main.java 2nd Way:

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III.

3.1. Course course = (Course) session.load(Course.class, new Long(1));

3.2.1. Student student = new Student();

student.setStudentName("N@IT");

3.2.2.

student.setCourse(course);

IV. session.save(student);

Note: In the 2nd way we get exception - Because CNO is Unique - we

public class Course

 private Long courseNo;
 private String courseName;
 private Set studentSet;
 getters & setters

3

public class Student

 private Long studentNo;
 private String studentName;
 private Course course;
 getters & setters

3

course.hbm.xml:

courseNo — CNO → ID → Sequence.
courseName — CNAME → Property
< set name = "studentSet" cascade = "all" inverse = "true"
 lazy = "true" />

<key column = "CNO" />

<one-to-many class = "edu.model.Student" />

</set>

student.hbm.xml:

studentNo — SNO → ID → Sequence.
studentName — SNAME → Property

<many-to-one name = "course" column = "CNO"

class = "edu.model.Course" cascade = "save-update" />

Main.java 1st:

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III. 3.1. Course course = new Course();

course.setCourseName("MCA");

3.2.1. Student studentOne = new Student();
Student studentTwo = new Student();
studentOne.setStudentName ("N@It I");
studentTwo.setStudentName ("N@It II");

3.2.2. studentSet.add (studentOne);
studentSet.add (studentTwo);

3.2.3. studentOne.setCourse (course);
~~student~~ studentTwo.setCourse (course);
course.setStudentSet (studentSet);

IV. Session.Save (course); or

Note: Store Student object ,while storing the Student object first load the course object from the DataBase and do the association and then store the object (or) New course with new Student.

New Course With New Student (ManyToOne) :

Course course = new Course();
Student student = new Student();
student.setStudentName ("N@It I");
studentSet.add (student);
course.setCourseName ("MCA");
student.setCourse (course);
course.setStudentSet (studentSet);
Session.Save (student);

2/11/09
Monday

SCOTT • STUDENT

SNO	NUMBER(19)	IDX-1
SNAME	VARCHAR2(25)	

SCOTT • STUDENT_INSCOURSE

SNO	NUMBER(19)	
CNO	NUMBER(19)	

SCOTT • INSCOURSE

CNO	NUMBER(19)	IDX-1
CNAME	VARCHAR2(25)	

STUDENT Table Data :

SNO	SNAME
1	Naresh IT1
2	Naresh IT2

INSCOURSE Table Data :

SNO	CNO
1	1
1	2
2	1
2	2

STUDENT_INSCOURSE Table Data :

CNO	CNAME
1	JAVA
2	.Net

ManyToMany With Bag : 

public class Course

private Long courseNo;
private String courseName;
getters & setters

3

public class Student

private Long studentNo;
private String studentName;
private List coursesList;
getters & setters

3

course.hbm.xml:

- courseNo — CNO → ID → sequence.
- courseName — CNAME → property

student.hbm.xml:

- studentID — SNO → ID → sequence.
- studentName — SNAME → property

<bag name = "courseList" table = "STUDENT_INSCOURSE"
cascade = "all" >

"
<key column = "SNO"/> — Student Table column
<many-to-many column = "CNO" class = "edu.model.Course"/>
</bag>

Note: SNO and CNO will be inserted in STUDENT_INSCOURSE table which contains the complete relation.

Main.java Ist:

I. Session session = SessionUtil.getSession();

II. Transaction transaction = session.beginTransaction();

III.
3.1. Course courseOne = new Course();

Course courseTwo = new Course();

courseOne.setCourseName("Java");

courseTwo.setCourseName("Net");

3.2.1.

courseList.add(courseOne);

courseList.add(courseTwo);

3.2.2.

Student studentOne = new Student();

Student studentTwo = new Student();

studentOne.setStudentName("Naresh IT I");

studentTwo.setStudentName("Naresh IT II");

3.2.3.

studentOne.setCourseList(courseList);

studentTwo.setCourseList(courseList);

IV.

Session.close();

SNO	CNO
1	1
1	2
2	1
2	2

CNO	SNO
1	1
1	2
2	1
2	2

Student 1 → 2 Courses

Student 2 → 2 Courses

Course 1 → 2 Students

Course 2 → 2 Students

→ We are implementing ManyToMany in the First way, we can implement it in the Second way also.

1st Way:

- Take Student1 and Collection of courses and store Student1 in the DataBase.
- Take Student2 and Collection of courses and store Student2 in the DataBase.
- Here Course is independent and student is dependent. So student hbm file contains the complement relationship (ManyToMany Tag).

2nd Way:

- Take Course1 and Collection of students and store Course1 in the DataBase.
- Take course2 and Collection of students and store course2 in the DataBase.
- Here Student is independent and Course is dependent. So, Course hbm file contains the complement relationship (ManyToMany Tag)
- If both the ways the relationship is there it will be Bi-direction.
- We can do ManyToMany using List, Set, Bag & Map.

④ Working With Map:

< map name = "courseMap" table = " STUDENT_INSCOURSE"
cascade = "all" >

< Key column = "SNO" />

< index column = "IDX" type = " string" length = "10" />

< many-to-many column = "CNO" class = "edu.model.Course" />

</map>

Note:

→ In IDX column Map object key will be stored

Map → Add Course objects to Map object and set the Map object to Student.

```
courseMap.put ("Java", courseOne);
```

```
courseMap.put (".Net", courseTwo);
```

Map Table :

SNO	CNO	IDX
1	1	Java
1	2	.Net
2	1	Java
2	2	.Net

Inverse:

→ If BiDirection is there inverse should be true then it will generate proper insert query and while changing the above inverse true to false check in the console.

lazy:

→ If lazy = true all the objects is not loaded initially and if the lazy = false all the objects will be loaded initially.

31/11/09
Tuesday

HIBERNATE - INHERITANCE

SCOTT - MBASTUDENT		
SNO	NUMBER(19)	IDX-1
SNAME	VARCHAR2(12)	
MBASTREAM	VARCHAR2(12)	

SCOTT - MCASTUDENT		
SNO	NUMBER(19)	IDX-1
SNAME	VARCHAR2(12)	
MCASTREAM	VARCHAR2(12)	

→ For the above tables we required two POJO's MBASStudent, MCAStudent
public class MBASStudent

{
 private Long studentNo;
 private String studentName;
 private String mbaStream;

}

public class MCAStudent

{
 private Long studentNo;
 private String studentName;
 private String mcaStream;

}

→ The above object design is wrong Because we are repeating
two properties in both the POJO's and Let's Create a Separate
Object for that and Apply inheritance relationship (Is-A Relation)

Is-A Relation :

public class Student

{
 private Long studentNo;
 private String studentName;
 getters & setters

}

public class MBASStudent extends Student

{
 private String mbaStream;
 getters & setters

}

public class MCAStudent extends Student;

{
 private String mcaStream;
 getters & setters

}

→ For the above POJO's design there might be 3 ways of DataBase Tables design and The above Table design is one of the way

MBASTUDENT Table Data:

SNO	SNAME	MBASTREAM
1	MBAName	Marketing

MCASTUDENT Table Data:

SNO	SNAME	MCASTREAM
1	MCAName	Computers

→ To persist the data in the above DataBase Tables design we apply TablePerConcreteClass at the Hibernate side.

→ TablePerConcreteClass means the tables are independent tables & At the Hibernate side we write independent hbm files with the configuration means we write MBAStudent.hbm and MCAStudent.hbm.

Note: While Mapping an object we can Map the Super class object details also means we are specifying MCASTream & It's Super class StudentId and StudentName in the same hbm file

TablePerClass :

SCOTT-STUDENT	
SNO	NUMBER(19) IDX-1
STUDENT_DISC	VARCHAR2(10)
STUDENTNAME	VARCHAR2(12)
MBASTREAM	VARCHAR2(12)
MCASTREAM	VARCHAR2(12)

DataBase Data:

SNO	STUDENT_DISC	STUDENTNAME	MBASTREAM	MCASTREAM
1	MCAStudent	MCAName		Computers
2	MBASStudent	MBAName	Marketing	

- For the above case we need a mapping mechanism between the hypermedia side.
- Both the objects data we are storing in a single table and The record will be identified using Discriminator column value (STUDENT-DISC) we call it as a "TablePerClass".
- In the HBM file we Map the Super class object with the Table and The sub class object without any table because all the data should be stored in a same table.
- The Super class we map inside class tag and Sub class we need to map inside sub class tag.

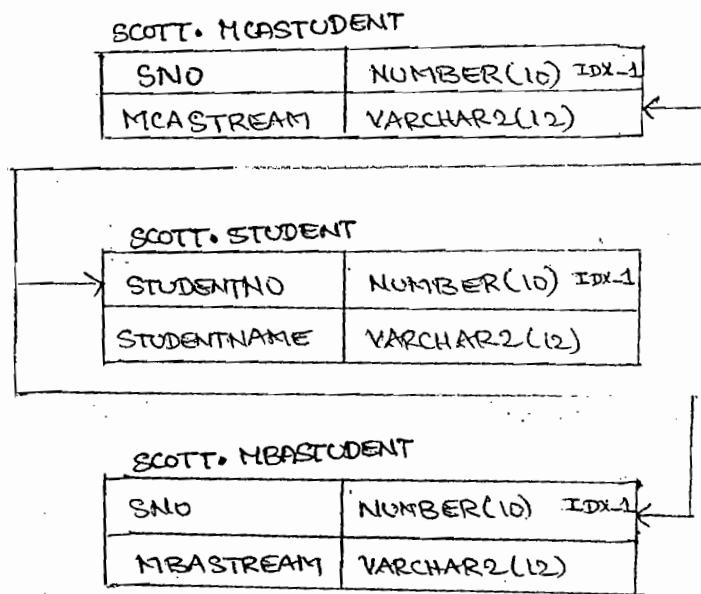

```

<class name="edu.model.Student" table="Student">
    <discriminator-value="Student">
        id -> studentId -> SNO
    <discriminator column="student-disc" type="string">
        property -> SNAME -> studentName
    <subclass name="edu.model.MBAStudent">
        <discriminator-value="MBAStudent">
            <property name="mbaStream"/>
        </subclass>
        <subclass name="edu.model.MCAStudent">
            <discriminator-value="MCAStudent">
                <property name="mcaStream"/>
            </subclass>
        </class>
      
```

Note: In discriminator tag, we specify the table discriminator column. For the objects we specify discriminator-value. While persisting the objects, the discriminator-value will be stored in the discriminator column.

Note: Discriminator column must be a Filter id tag.

TablePerSubClass:



→ For the above DataBase table design we need to apply TablePerSubClass at the hibernate side.

Table Data:

STUDENT TABLE	
STUDENTNO	STUDENTNAME
1	MCA NAME
2	MBANAME

MCASTUDENT TABLE	
SNO	MCASTREAM
1	Computer

MBASTUDENT TABLE

SNO	MBASTREAM
2	Marketing

→ At the DataBase tables design if it is having with the relationship then we need to apply TablePerSubClass at the hibernate side.

→ The Sub classes should be given in JoinedSubClass tag and the sub class table should have relationship with the Super class table and the primary key of the parent table should map with the child table primary key and the Foreign key and we give the corresponding primary key column inside

→ In inheritance examples we can set values of the object using constructor. If we want we can set using setter methods also.

Components:

SCOTT STUDENT		SCOTT EMPLOYEE	
SNO	NUMBER(19) IDX-1	ENO	NUMBER(19) IDX-1
SNAME	VARCHAR2(20)	ENAME	VARCHAR2(20)
DOORNO	VARCHAR2(20)	DOORNO	VARCHAR2(20)
STREET	VARCHAR2(20)	STREET	VARCHAR2(20)

→ For the above Tables we required two POJO's

```
public class Student
{
    private Long studentId = null;
    private String studentName = null;
    private String doorNo = null;
    private String street = null;
}
```

```
public class Employee
{
    private Long employeeId = null;
    private String employeeName = null;
    private String doorNo = null;
    private String street = null;
}
```

→ In the above COP design we are repeating address details in two POJO's and let's create a separate object and apply has-a relationship.

```
public class Address
{
    private String doorNo = null;
    private String street = null;
}
```

using

```
public class Student  
{  
    private Long studentNo = null;  
    private String studentName = null;  
    private Address address = null;  
}
```

```
public class Employee  
{  
    private Long employeeNo = null;  
    private String employeeName = null;  
    private Address address = null;  
}
```

→ Component means while mapping the student object. we do the normal mapping for studentNo, studentName, and for the address we have to use Component because it's a separate component means separate object.

→ If we persist the Student object. All the data will (student & address) will be stored in the same table.

```
private Address address;
```

For this we use Component tag in the hbm file.

```
<component name = "address" class = "edu.model.Address">
```

— Normal Mapping —

```
</component>
```

Note: Component can contain another Component ..

4/11/09
Wednesday CompositeID:

composite-ID:

STUDENTNOONE	NUMBER(19)
STUDENTNOTWO	NUMBER(19)
STUDENTNAME	VARCHAR2(25)

→ If a table contains the primary key based on more than one column we call it as a Composite-ID.

→ How the mapping should be there for the above table

```
<class name = "edu.model.Student" table = "STUDENT">
```

```
<id name = "studentNoOne" type = "long" column = "studentNoOne">
```

```
<generator class = "assigned"/>
```

```
</id>
```

```
<property name = "studentNoTwo">
```

```
<column name = "studentNoTwo" length = "25"/>
```

```
</property>
```

```
<property name = "studentName">
```

```
<column name = "studentName" length = "25"/>
```

```
</property>
```

```
</class>
```

→ The above mapping is wrong because the primary key is based on two columns. But we are specifying only one column in the id tag.

Table Data:

STUDENTNOONE	STUDENTNOTWO	STUDENTNAME
1	2	Naresh@It
1	1	Naresh@It

→ Load the object from the Database and we are expecting the below query should be executed.

```
SELECT * FROM STUDENT WHERE STUDENTNOONE = 1  
AND STUDENTNOTWO = 1
```

Java code:

```
Student studentLoad = new Student();
studentLoad.setStudentNoOne(new Long(1));
studentLoad.setStudentNoTwo(new Long(1));
studentLoad = (Student) session.load(Student.class, studentLoad);
→ But hibernate generates the below query
```

```
SELECT * FROM STUDENT WHERE STUDENTNOONE = 1
```

- We are expecting two columns in the where condition. But here it is generating one column in the where condition because in the id tag we specified one column. So we might be wrong data.
- To avoid the problem... Map both the columns in the HBM file using composite-ID tag.
- The below mapping is the correct one -

```
<class name="edu.model.Student" table="STUDENT">
<composite-id>
  <key-property name="studentNoOne"
    type="java.lang.Long" column="studentNoOne"/>
  <key-property name="studentNoTwo"
    type="java.lang.Long" column="studentNoTwo"/>
</composite-id>
<property name="studentName">
  <column name="studentname" length="25"/>
</property>
</class>
```

Java code :-

```
Student studentLoad = new Student();
studentLoad.setStudentNoOne( new Long(1));
studentLoad.setStudentNoTwo( new Long(1));
studentLoad = ( Student) session.load( Student.class, studentLoad);
```

→ Here hibernate generates the correct query means with 2 conditions then we get the correct data.

```
SELECT * FROM STUDENT WHERE STUDENTNOONE = 1
AND STUDENTNOTWO = 1
```

Two ways To implement Composite-ID :-

1. Define the primary key column related properties and the Non primary key column related properties in a single POJO.
2. Define the primary key column properties in a POJO & define Non primary key column properties in another POJO and make the "has - A - relationship" between both the pojo's.

→ The primary key column related POJO object must have to be implemented Serializable interface.

1st Way :-

Student.hbm.xml :

```
<class name = "edu.model.Student" table = "STUDENT">
<composite-id>
<key-property name = "studentNoOne" type = "java.lang.Long"
column = "studentNoOne"/>
<key-property name = "studentNoTwo" type = "java.lang.Long"
column = "studentNoTwo"/>
</composite-id>
=====
</class>
```

composite-id : It is used to define composite keys.

key-property : It is used to define composite column and to map the column with the property.

Save or Inserting The Object:

```
Student student = new Student();
student.setStudentNoOne(new Long(1));
student.setStudentNoTwo(new Long(1));
student.setStudentName("Unlabeled It");
session.save(student);
```

Load The Object:

```
Student studentLoad = new Student();
studentLoad.setStudentNoOne(new Long(1));
studentLoad.setStudentNoTwo(new Long(1));
studentLoad = (Student) session.load(Student.class, studentLoad)
```

Note: In the where condition Two columns should be taken while retrieving we are setting both the values to an object and we are passing the object to the load method instead of new Long(1).

2nd Way:

POJO's:

```
public class StudentId implements Serializable
{
    private Long studentNoOne;
    private Long studentNoTwo;
    getters & setters
}
```

```
public class Student implements Serializable
{
    private StudentId studentId = null;
    private String studentName = null;
```

```

<class name = "edu.model.Student" table = "STUDENT">
  <composite-id name = "StudentId">
    class = "edu.model.StudentId"
  </composite-id>
  <key-property name = "studentNoOne" type = "java.lang.Long">
    column = "studentNoOne"
  </key-property>
  <key-property name = "studentNoTwo" type = "java.lang.Long">
    column = "studentNoTwo"
  </key-property>
</class>

```

- To avoid the duplication inside StudentId POJO we need to override equals and hashCode methods and we need to generate the hashCode and we need to compile the objects.
- To implement the equals and hashCode logic we use hashCodeBuilder and equalsBuilder and those classes are available in apache-commons jar file.

Save The object :

```

StudentId studentId = new StudentId();
studentId.setStudentNoOne(new Long(1));
studentId.setStudentNoTwo(new Long(2));
Student student = new Student(studentId);
// Student student = new Student();
// student.setStudentId(studentId);
student.setStudentName("Naresh It");
session.save(student);

```

Load The Object:

```
StudentId studentId = new StudentId();
```

```
Student studentLoad = new Student();
```

```
studentLoad.setStudentNoOne(new Long(1));
```

```
studentLoad.setStudentNoTwo(new Long(1));
```

```
studentLoad = (Student) session.load(Student.class, studentId);
```

Note: Here we are passing studentId object to the load method instead of student object Because it contains composite key data.

5/11/09
Thursday
⊗
⊗
⊗

CACHE

www.JaveBeat.com

→ Hibernate supports 2 Levels of Caching.

1. FirstLevel Cache → Session
2. SecondLevel Cache → SessionFactory

FirstLevel Cache [Session]:

→ For the 1st Level Cache, we don't require any configuration.
It is the default cache provided by the hibernate and it will be applied on top of Session object.

→ At the Session level there will be only one DataBase hits.

→ If there are "M" no. of Session objects and If we are loading "N" no. of times there will be "M" no. of DataBase hits.

10 Sessions Load 100 Times → 1:0 DataBase hits.

Example:

```
Session session = null;
```

```
(new SessionFactoryBuilder().build().openSession().load(Student.class, new Long(1));
```

```
SOP("Update the student data in database..");
student = (Student) session.load(Student.class, new Long(1));
SOP("Student Name." + student.getStudentName());
```

Steps To Understanding:

→ There is one Session object we are loading it two times so, there will be only one DataBase hit.

1. Check the Console there will be one Select query means there is one DataBase hit.
2. Keep a Break point at Update SysOut and Update the DataBase Value then Continue with the flow and we get the old value for the second time loading

Example:

```
student = (Student) sessionOne.load(Student.class, new Long(1));
SOP("Student Name." + student.getStudentName());
SOP("Update the student Data in DataBase.");
student = (Student) sessionOne.load(Student.class, new Long(1));
SOP("Student Name." + student.getStudentName());
student = (Student) sessionTwo.load(Student.class, new Long(1));
SOP("Student Name." + student.getStudentName());
```

Steps To Understand:

→ There are two Session objects and 3 Times we are Loading. So, There will be two DataBase hit.

1. Check the Console
2. Update the DataBase value and in the second Session we get the update value.

Note: If we Enable the Second Level Cache in the Second Session also we get the old value only. Because Second Level Cache will be Apply at SessionFactory Level means at the SessionFactory Level there will be only one DataBase hit.



→ If there are 10 SessionFactories and if there are M no. of Sessions and if we are loading N no. of times then there will be 10 DataBase hits.

→ We Create only one SessionFactory object in the project.

2nd Level Cache [SessionFactory]:

→ Second Level Cache will be apply on Top of SessionFactory.

There are 4 Types of 2nd Level Cache:

1. EHCache (org.hibernate.cache.EhCacheProvider)

2. OSCache (org.hibernate.cache.OSCacheProvider)

3. SwarmCache (org.hibernate.cache.SwarmCacheProvider)

4. JBossCache (org.hibernate.cache.TreeCacheProvider)

EHCACHE:

→ EHCache is a fast, LightWeight, and Easy-to-use in-process Cache. It supports read-only and read/write Caching, and Memory-and disk-based Caching. However, it does not support Clustering.

OSCACHE:

→ OSCache is another open-source, Caching solution. It is part of a larger package, which also provides Caching functionalities for JSP pages (or) arbitrary objects.

→ It is powerful and flexible package, which, like EHCache, supports read-only and read/write Caching, and memory-and disk-based Caching.

→ It also provides basic support for clustering via either JavaGroups (or) JMS.

SwarmCache:

→ SwarmCache is a simple cluster-based Caching solution based on JavaGroups.

→ It supports read-only (or) nonstrict read/write Caching.

- This type of cache is appropriate for applications typically have many more read operations than write operations.

JBOSS TreeCache:

- JBOSS TreeCache is a powerful replicated (Synchronous or Asynchronous) and Transactional cache. Use this solution if you really need a true transaction - Capable Caching architecture.

Properties:

Read-only:

- This strategy is useful for data that is read frequently but never updated. This is by far the simplest and best-performing Cache Strategy.

Read/Write:

- Read/Write Caches may be appropriate if your data needs to be updated. They carry more overhead than read-only caches.
- In Non-JTA environments, Cache transaction should be completed when Session.close() (or) Session.disconnect() is called.

Nonstrict read/write:

- This strategy does not guarantee that two transactions won't simultaneously modify the same data.
- Therefore, it may be most appropriate for data that is read often but only occasionally modified.

Transactional:

- This is a fully Transactional Cache that may be used in a JTA Environment.

Cache	Read-only	Nonstrict Read/write	Read/Write	Transactional
EHCache	Yes	Yes	Yes	No
OSSCache	Yes	Yes	Yes	No
SwarmCache	Yes	Yes	No	No
JBOSSTreeCache	Yes	No	No	Yes

ehcache.xml:

Attributes:

maxInMemory:

→ Sets the maximum no. of objects that will be created in memory

eternal:

→ Sets whether elements are eternal. If eternal, Timeouts are ignored and the element is never expired.

timeToIdleSeconds:

→ Sets the time to idle for an element before it expires.
→ Is only used if the element is not eternal.
→ Idle time is now - last accessed.

timeToLiveSeconds:

→ Sets the time to live for an element before it expires.
→ Is only used if the element is not eternal.
→ TTL is now - creation time.

overflowToDisk:

→ Sets whether elements can overflow to disk when the in-memory cache has reached the maxInMemory limit.

Steps To Configure 2nd Level Cache:

1. Give the Provider class details in the Configuration file.

<Property name = "hibernate.cache.use_query_cache"> true </property>

<Property name = "hibernate.cache.use_second_level_cache"> true </property>

```
<Property name = "hibernate.cache.provider-class"> org.hibernate.cache.EhCacheProvider </property>
```

2. Configure the `EhCache.xml` and here we need to provide the POJO Details.

```
<cache name = "edu.model.Student">  
maxElementsInMemory = "300"  
eternal = "true"  
overflowToDisk = "false" />
```

Note: If we don't Configure Specific POO, Then it takes the Default Settings.

3. On whatever Table we are going to apply the Second Level Cache in the Corresponding HBM file give cache usage tag.

```
<cache usage = "read-only" />
```

Note: Step1 and Step2 Configuration will be once in the Project and Step3 will be repeated Multiple times.

Note: We Apply the Cache on Constant tables (or) the Lookup table.

Eg: ZIPCODE, STATES and COUNTRY

Steps To Configure OSCache:

1. Configure the Provider class and oscache.properties entries in the Configuration file. Here instead of EhCache.xml we take a Properties file.

```
<Property name = "hibernate.cache.use-query-cache"> true </Property>
```

```
<Property name = "hibernate.cache.use-second-level-cache"> true </Property>
```

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

<property name="com.opensymphony.oscache.configurationResourceName">
    edu/config/oscache.properties </properties>
```

2. oscache.properties

[region].refresh.period = 4000

3. Give the Entry in the HBM file.

Note: While Testing, test with cache usage entry and without cache usage entry. If Cache usage entry is there in the second session we get the old data. If it is not there we get the updated data.

Working with ConnectionProvider:

→ In Projects, if we want to establish connection with different schema we never write different configuration files.

If we are writing the program will work but it is wrong as per the design.

→ There are two ways to implement the concept.

1. Using `openSession()` method
2. Using `ConnectionProvider` concept

Using `openSession()` method:

1. Don't provide any DB Configuration details in the `cfg` file
[URL, Username, Password and Driver class etc]

2. Pass the JDBC provided Connection object to `OpenSession()` method

Connection con = DBUtil.getConnection();

Session session = sessionFactory.openSession(con);

Using ConnectionProvider Concept:

1. Take any java class and implement the ConnectionProvider interface and inside getconnection() method get the JDBC provided Connection object.
2. Don't provide the DBDetails in the cfg file [URL, username, password, etc.] and Provide the ConnectionProvider class entry in the cfg file.

```
<Property name="hibernate.connection.Provider-class">  
edu.connection.StudentConnectionProvider</property>
```

VERSIONING AND TIMESTAMP:

- Whenever the record is inserted (or) whenever the record is updated. If we want to maintain version change (or) timestamp we need to Map the corresponding column using version tag and timestamp tag.
- Whenever the record is inserted it takes the version number as 0 and current system time (or) current time. Whenever the record is updated automatically hibernate will Add +1 to the version number and current system time.
- If we want to implement the same concept using JDBC we need to apply some logic and here it is done just using an XML entry.

```
<timestamp name = "timestamp" column = "STIMESTAMP"/>
<version name = "version" column = "version" type = "integer"/>
```

7/11/09 Saturday How To Execute The Stored Procedures And The Functions:

There are Two ways to Execute it,

1. Get the Connection object from the Session then implement the Callable Statement code.

```
Connection con = session.Connection();
```

Callable Statement code

2. Give the Stored Proc Entry in the HBM file

2.1.

```
<sql-query name = "StudentProcedure" Callable = "true">
```

```
<return alias = "student" class = "edu.modul.Student">
```

```
<return-property name = "studentNo" column = "SNO"/>
```

```
<return-property name = "studentName" column = "SNAME"/>
```

```
<return-property name = "studentAge" column = "SAGE"/>
```

```
? Call S-PROCEDURE (?, :studentNo); ?
```

```
</sql-query>
```

sql-query:

→ Store Proc call will be there in this tag:

name (StudentProcedure):

→ We can give any name, The same name will be used from the hibernate Java file code.

Callable = "true" Because Stored Proc should be Executed using Callable Statements.

return:

→ Map the Cursor object with the Student object.

StudentNo : Named parameter the value will be passed from the Java program.

cursor : It's just like ResultSet.

Main.java :

```
List studentList = session.getNamedQuery("studentProcedure")
    .setParameter("studentNo", new Long(1)).list();
```

Q. How To Execute Stored Proc in TOAD?

Select the stored Proc in SQLEditor and click on Execute as a script (It should not be Execute) → F9

Q. How To Execute The Function?

Function will be Executed in the same way of Stored Procedure. But the difference is in the HBM file we declare the return type.

FORMULA :

Query : Select SNO, SNAME, CONCAT(SNO, SNAME) from STUDENT

Table:

SNO	SNAME

→ For the above table we required a POJO with Two Properties. But our requirement is whenever we are trying to retrieve the data, we need to CONCAT the First and the Second column and we require that result also in the POJO.

→ So, the POJO should be contain 3 properties. But there is no column to Map the Third Property.

→ To Map the Third Property instead of column name we use Formula attribute then we map with the Third Property.

```

public class Student
{
    private Long studentNo;
    private String studentName;
    private String studentMiddleName;
    getters & setters
}

<id name = "studentNo" type = "long" column = "SNO">
    <generator class = "increment"/>
</id>

<property name = "studentName">
    <column name = "SNAME" length = "15"/>
</property>

<property name = "studentMiddleName">
    formula = "CONCAT(SNO, SNAME)"/>

```

Note: For the Third Property we are taking the data using Oracle Existing function. So, we have to use formula for that

Main.java:

— do — Starting Example

9/11/09
Monday Hibernate Data Types :

Working with Different Data Types:

Binary Large Object (BLOB): It is a SQL datatype that represents Binary data to be stored into a DataBase.

BLOB helps us to store images into a DataBase.

Character Large Object (CLOB): It is a SQL datatype that represents Larger volumes of text data.

CLOB helps to store text files into a DataBase.

```

public class Student implements Serializable
{
    private Long studentNo;
    private String studentName;
    private Date studentDateOne;
    private Date studentDateTwo;
    private Float studentFee;
    private Blob studentImage;

    getters & setters;
}

```

HBM File :

```

class edu.model.Student -- STUDENT
id = studentNo type = "long" -- SNO      sql-type = "int"
generator class = "increment"
Property studentName -- SNAME
Property studentDateOne -- SDATEONE      type = "timestamp"
Property studentDateTwo -- SDATETWO      type = "Date"
Property studentFee -- Sfee      sql-type = "NUMERIC(12,2)"
Property studentImage -- SIMAGE      sql-type = "BLOB"

```

Note : To store the BLOB object in the Java POJO class, First we need Stream it. Then we convert it has a BLOB object. To do that we will write another two methods in the POJO class.

```

public void setStudentImageStream (InputStream sourceStream) throws
IOException
{
    setStudentImage ( Hibernate.createBlob (sourceStream));
}

public InputStream getStudentImage Stream () throws SQLException
{
    if (getStudentImage () == null)
        return null;
    return getStudentImage () .getBinaryStream();
}

```

Note: We are writing two extra methods in the POJO class. Those methods will actually apply the streaming logic from the Java program. We call those extra methods from the Java, which will call the actual Database Mapped column property method.

→ To convert the Stream object to the Blob object we are calling createBlob() method on hibernate class.

Save The Object:

```
Student student = new Student();
student.setStudentName ("Naresh IT" + i);
student.setStudentDateOne (new Date());
student.setStudentDateTwo (new Date());
student.setStudentFee (new Float (2500.50));
student.setStudentImageStream (DataInputStream.class.getClassLoader().
getInputStream ("edu/file/student.txt"));
session.save (student);
```

Load The Object:

```
Student student = (Student) session.load (Student.class, new Long()
SOP (".. StudentNo.." + student.getStudentNo()));
File outputFile = new File ("f:/studentDB.txt");
OutputStream outputStream = null;
InputStream inputStream = null;
try {
    outputStream = new BufferedOutputStream (new FileOutputStream
        (outputFile, true));
    int read = -1;
    inputStream = student.getStudentImageStream();
    while ((read = inputStream.read ()) != -1)
```

```

        catch (Exception e)
        {
        }
    finally
    {
        outputStream.close();
        inputStream.close();
    }
}

```

Note: To declare the Java Data Type, we will use type attribute
 and To declare the sql-data-type, we will use sql-type attribute

Filter Concept in Hibernate:

→ On the data we can apply the Filter then some of the data will be is not retrieve based on the conditions means it is Filtering the data.

Steps To Implement The Filter:

1. Write the Filter definition in the HBM file.

```

<filter-def name = "studentFilter">
    <filter-param name = "studentNameParam"
                  type = "String"/>
</filter-def>

```

2. Write the Filter condition inside the class tag.

```

<filter name = "studentFilter"
       condition = "? : studentNameParam = SNAME" />

```

3. Create the Filter object before Executing the query.

//Filter

```

Filter filter = session.enableFilter("studentFilter");
filter.setParameter("studentNameParam", "M@ITI");
String studentQuery = "from Student student";

```

```
Query resultQuery = session.createQuery("studentQuery");  
List studentList = resultQuery.list();
```

→ Here it suppose to generate the query as

```
SELECT * FROM STUDENT
```

→ But it will generate the query as

```
SELECT * FROM STUDENT WHERE ? = SNAME
```

Because in the query it takes the filter condition also.

Schema Export :

→ Using the HBM file we can generate the sql script.

→ To generate the sql script the SchemaExport class will be used.

I.

```
Configuration configuration = new Configuration().configure("edu/config/  
hibernate.cfg.xml");
```

II.

```
SchemaExport schemaExport = new SchemaExport(configuration);
```

III.

```
schemaExport.setOutputFile("F:/student.sql");
```

IV.

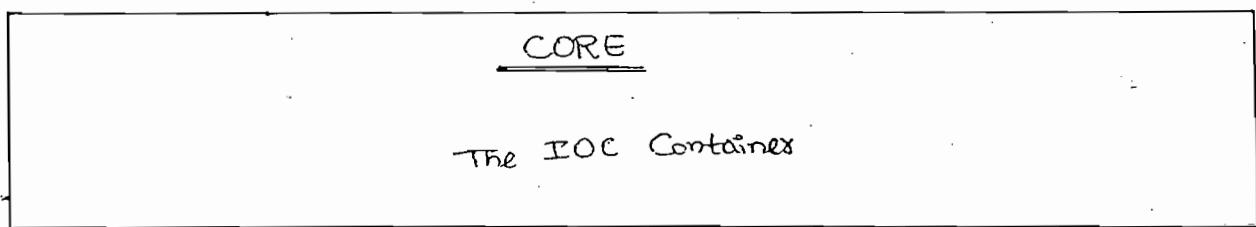
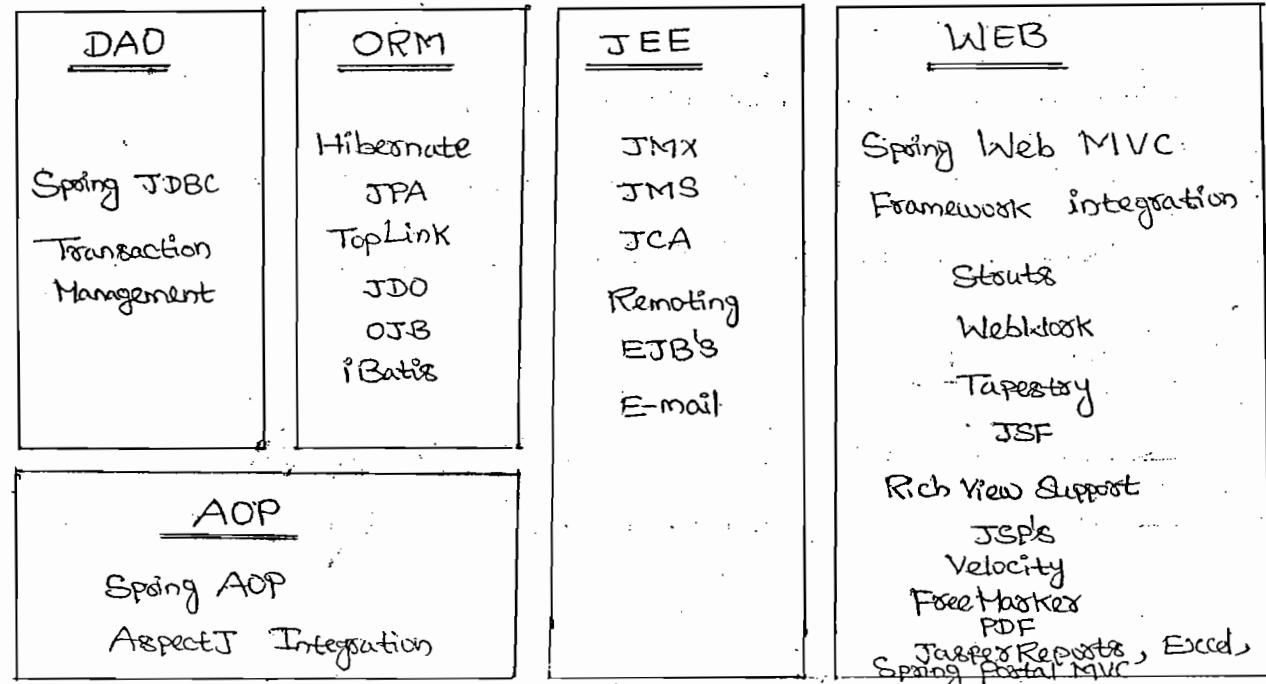
```
schemaExport.create(true, true);
```

12/11/09
Thursday

CL124141M

— Mr. VARMA

Spring Architecture Overview:



CORE Advantages:

- * Spring will play a role in all the areas of a Web Application
Eg: UI, Business Layer (Service), DAO Layer --- etc
- * We can Simplify the complete Web project coding.

I. How Spring Simplify DAO Layer code:

Hibernate:

```
public class StudentDAOImpl extends AbstractDAOImpl implements StudentDAO
{
    public void insertStudent( Student student) throws StudentException
}
```

```

Session session = getSession();
try {
    session.save(student);
}
catch(HibernateException e) {
    throw new StudentException("Exception In Dao");
}
}

```

Spring Integrate with Hibernate(ORM):

```

public class StudentDaoImpl extends XXXDaoSupport implements
                                StudentDao
{
    public void insertStudent(Student student) {
        getSession().save(student);
    }
}

```

- * Using Hibernate we can simplify the persistence logic means the JDBC logic. But we cannot simplify the exception logic.
- * If we integrate Spring with Hibernate we "don't require any exception handling logic".
- * If we want to "rethrow userdefined exception" then declare the exception name in the Spring Configuration file for a particular method, If any exception comes inside the Dao Method then Spring will rethrow the declared userdefined exception in the Spring Configuration file.
- * In Hibernate projects we get the Session object in the Service Layer. Because we apply the transaction in Service Layer and the same Session object should be used in the DAO Layer. So To pass the Session object from Service to DAO we are taking AbstractDAO class.
- * These type of Abstract classes which are not required if we are using Spring. Because Spring people are providing so many

Suffix will be DaoSupport.

Note: To know the Support Classes, Go to

Ctrl + Shift + T : *Spring • *DaoSupport

ctrl+shift+T

If we are using Spring - JDBC \leftrightarrow JdbcDaoSupport /
SimpleJdbcDaoSupport (1+5)

Spring - Hibernate \leftrightarrow HibernateDaoSupport

Spring - JPA \leftrightarrow JpaDaoSupport

Spring - TopLink \leftrightarrow TopLinkDaoSupport

Spring - iBatis \leftrightarrow SqlMapClientDaoSupport

II. How Spring Simplifies Service Layer Code:

Hibernate:

```
public class StudentServiceImpl implements StudentService
{
    public void insertStudent( Student student) throws StudentException
    {
        Session session = SessionUtil.getSession();
        Transaction transaction = null;
        StudentDao studentDao = new StudentDaoImpl();
        (or)
        StudentDao studentDao = DaoFactory.getStudentDao();
        try
        {
            transaction = session.beginTransaction();
            studentDao.setSession(session);
            student = studentDao.insertStudent(student);
            transaction.commit();
        }
        catch( StudentException e)
        {
            transaction.rollback();
            throw new XXXException("Exception In Service");
        }
    }
}
```

finally
 {
 SessionUtil.closeSession(session);
 }
}

Spring code:

```
public class StudentServiceImpl implements StudentService  
{  
    public void insertStudent(Student student)  
    {  
        StudentDao studentDao;  
        [ public setStudentDao(StudentDao studentDao)  
            {  
                this.studentDao = studentDao;  
            }  
            [OR]  
        public StudentServiceImpl(StudentDao studentDao)  
            {  
                this.studentDao = studentDao;  
            }  
        ]  
        public void insertStudent(Student student)  
        {  
            studentDao.insertStudent(student);  
        }  
    }  
}
```

* We don't require util classes

Eg: HibernateUtil / SessionUtil / DBUtil / DBHelper ... etc.

* Instead of creating all the DAO classes a "Singleton" we are applying "Factory Design Pattern (DAOFactory)".

We don't require "DAO Object creation logic, Singleton Logic and DAOFactory Logic".

There are two ways to provide DAO object to Service

1. Constructor
2. Setter.

* We don't require Exception Handling logic in Service Layer.

* We don't require the try .. catch ..

- * We no need to pass the ~~session variable from service to view~~
- * We don't require Resource closing logic [session.close()]

Hibernate/Struts:

public class StudentAction extends Action

```
{ public ActionForward execute( ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest request,
                               HttpServletResponse response)
```

try

```
{ StudentService studentService = new StudentServiceImpl();
```

```
studentService = ServiceFactory.
```

```
getStudentService();
```

```
studentService.insertStudent( student );
```

}

```
catch ( StudentException e )
```

```
{ return mapping.findForward( "Success" );
```

Spring:

public class StudentAction extends Action

```
{ private StudentService studentService;
```

```
public void setStudentService( StudentService studentService )
```

```
{ this.studentService = studentService; }
```

}

```
public ActionForward execute( ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
```

try

```
{ studentService.insertStudent( student );
```

}

```

    catch (StudentException e)
    {
        return mapping.findForward("success");
    }
}

```

13/11/09
Friday Struts And Hibernate:

- * We required Factory classes to get the Service objects.

Spring:

- * We don't require a Factory class to get the Service object.

Note: We discussed All the Above Advantages related to Service And Dao Layer flow.

What Are The Other Benefits (or) Advantages Using Spring:

1. In JDBC Projects we write some logic to get the Connection Statement, PreparedStatement, ResultSet, etc. Here developer is writing the JDBC logic for Connection, ResultSet etc., Then developer should take care of Exception Handling and Resource Closing.
- * In Spring JDBC to get Connection, ResultSet objects we Contact the Framework. Then Framework will provide Connection and ResultSet to us and the Framework will Create Connection and ResultSet objects internally using core Java JDBC only. Here Connection and ResultSet are under Framework control the Framework will take care of Exception and Resource closing Logic.

```

Connection con = DBUtil.getConnection();
Statement stmt = con.createStatement(...);
ResultSet rs = stmt.executeXXX();

```

2. To implement the "Singleton logic" we require below code

```

class Singleton
{
    private static Singleton singleton = new Singleton();
}

```

```
public static Singleton getSingleton()
{
    return singleton;
}
```

- * In normal Projects, we write some logic for the Singleton and if we don't require Singleton then don't remove some logic in the Java file.
- * If it is Spring we do not need to implement any Singleton logic in the Java file - Using an XML Entry whenever we require we can change Singleton to NonSingleton and we don't require the logic Singleton to NonSingleton. The XML File Entry will be "Singleton = true/false" and the Java file will be without Singleton logic.
- * The Singleton Attribute will be different in 2.5 Version and The "Attribute as Scope"

3. How To get Session Object in Hibernate:

```
Configuration configuration = new Configuration().configure("edu/  
config/hibernate.cfg.xml");
```

```
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

```
Session session = sessionFactory.openSession();
```

- * According to the Hibernate Specification we require the above 3 steps to get the Session object. To work with persistence logic "N" no. of times we work with Session object. But according to the specification we require the other two also.

- * The Spring people are talking Let's give me the Control I will create the objects take the Session object from the Spring means we require only one step to get the Session.

```
Session session = sessionFactory.openSession();
```

Dependency Problem

14/11/09
Saturday

- * We have the dependency problem in the API Level and we have the dependency problem in the Application Code [Project Code].
- * In the Above code dependency means Session is dependent on SessionFactory and SessionFactory is dependent on Configuration.

How The Dependency Problem will be there in the Project Code:

Struts / Hibernate:

```
class Dao
{
    public void daoMethod()
    {
        System.out.println("DAO - " + daoMethod());
    }
}

class Service
{
    private Dao dao = new Dao();
    public void serviceMethod()
    {
        dao.daoMethod();
        System.out.println("Service - " + serviceMethod());
    }
}

class MainTest / Servlet / Action / Controller
{
    public void serviceMethod(String[] args)
    {
        Service service = new Service();
        service.serviceMethod();
    }
}
```

- * In the Above Example Servlet / Action / Controller classes are dependent on Service classes & Service classes are dependent on Dao classes.

Note: The Spring people are talking about why the developer should worry about the dependent objects. Let's give the Controller to the Framework and just Configure XML then Spring

* We need to write a XML file in which we will set the Service object to Service and setting the Dao Object to Service and setting the Service Object to Action/Controller. We can call "Setting" as Injection in Spring.

In How Many Ways We Can Inject Dao To Service and Service To Action / Controller

* We can inject in 2 ways,

1. Constructor
2. Setter

Class NareshWorld

```
private String nareshMessage;  
public NareshWorld (String nareshMessage)  
{  
    this.nareshMessage = nareshMessage;  
}  
public void setNareshMessage (String nareshMessage)  
{  
    this.nareshMessage = nareshMessage;  
}
```

```
public NareshWorld ()  
{  
}
```

Class MainTest

```
public (String args[])
```

// 1st Way

```
NareshWorld nareshWorld = new NareshWorld ("NareshWorld");
```

// 2nd Way

```
NareshWorld nareshWorld = new NareshWorld ();
```

```
nareshWorld.setNareshMessage ("NareshWorld");
```

3

3

- 98
- source section
- * In the Above Example, we are Injecting a Value to NareshWorld Property in 2 ways from the Java Program.
 - * If it is Spring we are going to inject a value to nareshMessage using a Configuration file instead of Main Program and we read the Configuration file from the Main program.
 - * For the Java Bean, there are some rules like getters & Setters etc
 - * In Springs, we can call "Any Java program" as "SpringBean". Here NareshWorld is SpringBean and MainTest is SpringBean, NareshWorld is not having getter Method. But still we are calling it has a SpringBean Because Any Java class can become SpringBean.

How To Inject a Value Using Spring :

```

class NareshWorld
{
    private String nareshMessage;
    public NareshWorld ( String nareshMessage )
    {
        this.nareshMessage = nareshMessage;
    }
    [OR]
    public void setNareshMessage ( String nareshMessage )
    {
        this.nareshMessage = nareshMessage;
    }
    public NareshWorld ()
}

```

Spring-config.xml (xxx.xml)

<bean id = "nareshWorld" [id=reference] class = "NareshWorld" >

// Constructor Approach

<constructor-arg ... <Value ... NareshWorld/>

[OR]

// Setter Approach

- <value name = nareshMessage ... <Value ... NareshWorld/>

```
public class MainTest
{
    PSVM( String args[])
}
```

// In Code Java → To get the object

```
NareshWorld nareshWorld = new NareshWorld("NareshWorld");
```

// In Spring → To get the object

// Read the config file After reading the config file Spring will
create the object

// We just take the object.

```
NareshWorld nareshWorld = xxx.getBean("nareshWorld");
```

Note: Spring internally creates the object using Class.forName()
and whenever we required the object just call getBean
Then it will provide the object to us.

```
}
```

What Exactly Spring is Doing Internally While Reading The

Config file :

```
<bean id = "nareshWorld" [id = reference] class = "NareshWorld">
```

// Constructor Approach

```
<constructor-arg ... <value ... NareshWorld/>
```

```
</bean>
```

* While creating the object using Class.forName() - It will
pass NareshWorld to NareshWorld constructor.

```
NareshWorld ("NareshWorld");
```

(OR)

```
[ new NareshWorld("NareshWorld") - JAVA CODE ]
```

```
<bean id = "nareshWorld" [id = reference] class = "NareshWorld">
```

// Setter Approach

```
<property name = nareshMessage ><value ... NareshWorld/>
```

```
</bean>
```

* Note: It will Create the Object using Class.forName(). Then it will call .setNameMessage using Reflection API [It takes the Property. It will Append Set as a prefix and takes the first character of the property as Caps. Then it will be "setNameMessage" while passing "NameMessage".

```
ref.setNameMessage ("NameMessage");
```

* Note: If we are writing a Property in the Spring Configuration file. Then there must be a Setter Method in the Corresponding class (or Its Super class).

16/11/09
Monday

Spring :

* In the Application level code, we Apply Injection on Service & DAO classes [Project Level code].

```
class Dao
{
    public void daoMethod()
    {
        System.out.println ("DAO. daoMethod()");
    }
}

class Service
{
    private Dao dao;
    public Service (Dao dao) // Constructor
    {
        this.dao = dao;
    }
    [OR]
    public void setDao (Dao dao) // Setter
    {
        this.dao = dao;
    }
    public void serviceMethod()
    {
        dao.daoMethod();
        System.out.println ("Service. serviceMethod()");
    }
}

class MainTest
{
    public static void main (String args[])
    {
        Service service = new Service ();
    }
}
```

beans

```

<bean id = "dao" class = "edu.dao.Dao"/>
<bean id = "service" class = "edu.service.Service">
  <construct-arg> ... <ref bean = "dao"/>
  <property name = "dao"> ... <ref bean = "dao"/>
</bean>
</beans>

```

In the Main program, The previous code should be replaced with

```
Service service = ... .getBean("service");
```

Configuration

|
SessionFactory

|
Session — API(Container)

Dao

|

Service

|
Servlet/Action/Controller(Spring)

— Application Code

Advantages:

1. We don't require the lookup's in Spring. Spring internally handles flows.

```
Context context = new InitialContext();
```

```
context.lookup("..."); // Don't required in Spring.
```

2. We can Simplify the Complete JMS Code & Web Services code
Once we integrate with Spring.

* If we integrate Spring & RMI, we no need to create
Any Stubs & Skeletons.

The final Conclusion: We can integrate Spring with lots
of technologies which are available in the Java Market
and The Main Advantage is "Code is going to be Simplified".

~~Implement~~ Spring Core Containers.

17/11/09
Tuesday

Implement Spring Core Container:

Refactoring HelloWorld Example:

```
public class HelloWorld
{
    public void main(String[] args)
    {
        System.out.println("HelloWorld!");
    }
}
```

- * According to the "Spring design" the above Example is "Wrong".
- * HelloWorld class has to Print the message, Here the implementation and the usage both are there in the same class.
- * Let's make it as a separate classes and Print the message.
- * Instead of HelloWorld Program, we are taking MessageProviderTest and HelloWorldMessageProvider. Here HelloWorldMessageProvider will provide the message and MessageProviderTest is using the message and printing message.

```
public class HelloWorldMessageProvider
{
    public String getMessage()
    {
        return "HelloWorld!";
    }
}

public class MessageProviderTest
{
    public void main(String[] args)
    {
        HelloWorldMessageProvider messageProvider = new
            HelloWorldMessageProvider();
    }
}
```

```
String message = messageProvider.getMessage();
System.out.println(message);
```

* A new person is going to provide HaiWorldMessage

```
public class HaiWorldMessageProvider
{
    public String getMessage()
    {
        return "HaiWorld!";
    }
}
```

* Now, The MessageProvider wants to shift from HelloWorld to HaiWorld then what logic he has to change in his main program. The below logic should be changed in the Main Program.

```
public class MessageProviderTest
{
    public static void main (String [] args)
    {
        HaiWorldMessageProvider messageProvider = new
            HaiWorldMessageProvider ();
    }
}
```

```
String message = messageProvider.getMessage ();
System.out.println (message);
```

* If we are shifting from one provider to another provider we need to change the logic in the main() method. The code is "not flexible".

(*) * Let's implement a "Flexible code" in the main() method and the code should not be affected, If you are shifting from one provider to another provider.

* In the main program, we might be changing from Hello to Hai and Hai to Hello Provider classes means The "Runtime behavior is getting changed". So, there we need to "apply Polymorphism".

```
Class A
{
    public void x()
    {
        System.out.println("A.x()");
    }
}
```

Class B extends A

```

{
    public void x()
    {
        System.out.println("B.x()");
    }
}
```

Class MainTest

```

{
    public static void main(String[] args)
    {
        A a = new B();
        a.x();
    }
}
```

* Technically the above code is Correct. But Understanding wise the above code is Completely Wrong [Main() method Logic]

* We know the implemented object B. But still we are assigning B object to A. 90% of the Cases it is wrong.
Because most of the Cases Assign the Sub Class Object to Super class reference variable (or) interface.

**

~~Note:~~

- * If we know the implementation details at Runtime, Then Assign the object to Super class reference (or) interface.
- * Provide the Runtime details to the Main() program using a Properties file (or) XML file.

PolyTest.java:

```
Package edutest
class A
{
    public void x()
}
```

class B extends A

```
{  
    public void x()  
    {  
        System.out.println("B.x()");  
    }  
}
```

class C extends A

```
{  
    public void x()  
    {  
        System.out.println("C.x()");  
    }  
}
```

public class PolyTest

```
{  
    private static Properties properties = new Properties();
```

static

```
{  
    try  
    {
```

```
        properties.load(PolyTest.class.getResourceAsStream("class.properties").
```

```
        getProperties());  
    }  
}
```

```
catch (Exception e)  
{  
}
```

PSVM (String[] args) throws Exception

```
{  
    String className = properties.getProperty("className");
```

```
    A a = (A) Class.forName(className).newInstance();
```

```
a.x();
```

```
}
```

class properties;

className = edu.test.C

* In the design level,

1. A
2. B & C
3. A a = object

All the 3 steps it never be implemented by a Single Person.

How It is implemented in Servlets?

```
// B extends A  
// B - HelloWorldServlet & A - GenericServlet/Servlet  
class HelloWorldServlet extends GenericServlet  
  
    {  
        public void service( req, res )  
        {  
    }  
}
```

Web.xml [class.properties] :

Servlet - Class → HelloWorldServlet

Containers

```
// A a = (A) Class.forName(B).newInstance(); [ A a = new B(); ]
```

```
Servlet servlet = (Servlet) Class.forName(HelloWorldServlet).newInstance();  
servlet.service(req, res);
```

Note :

* Here Container is Applying Polymorphism logic Because they
don't the Runtime details of the class at Compilation Means
They know the HelloWorldServlet at Runtime. So, Servlet API people are applying Polymorphism and the same logic
is apply in "JDBC, Streams and Any one of the MVC".

Steps To implement Polymorphism [For MessageProvide] :

1. Here Main program doesn't know what provider class it has to use means it will be decided at Runtime.

1.1. Here there should be some contract between the provider classes means Both the Provider classes must have to implement the interface.

2. Provide the implementation details in properties file (or) xml file.

5. Read the Properties file at Kutime and Create the Object using Class.forName() and we don't know about the implementation details. So, Assign the object reference to the Contract reference.

18/11/09

Wednesday

MessageProvider Example:

//A

```
Package edu.test;
public interface MessageProvider
{
    public String getMessage();
}
```

//B

```
public class HelloWorldMessageProvider implements MessageProvider
{
    public String getMessage()
    {
        return "HelloWorld!";
    }
}
```

//C

```
public class HelloMessageProvider implements MessageProvider
{
    public String getMessage()
    {
        return "HelloWorld!";
    }
}
```

```
public class MessageProviderTest
```

```
{
    private static Properties properties = new Properties();
```

```
static
```

```
{ try
```

```
    properties.load ( MessageProviderTest.class.
```

```
        getClassLoader().getResourceAsStream ("edu/
            test/message.properties"));
```

```
    catch (Exception e)
```

```
    { }
```

Object

the

so to

PSVM (String[] args) throws Exception

{ String providerName = properties.getProperty ("provider");

// A a = new B();

// A a = Create B Object Using Class.forName

MessageProvider messageProvider = (MessageProvider)

Class.forName (providerName).newInstance ();

String message = messageProvider.getMessage ();

SOP (message);

}

message.properties :

provider = edu.test.HelloWorldMessageProvider

(*) Note: Here the "Beauty of Interface" is, If we are shifting from one provider to another provider. We no need to touch in the Main program.

→ Test the above Example, while changing the class in the Properties file [Hello → Hai]

Why we take Dao interfaces and Service interfaces?

- * In the JDBC Applications, we take Dao interface and the Implemented classes, there it just provides only Abstraction.
- * But if we are shifting from one class to another class we need to change the logic in the Main program.
- * But in Spring Applications, using interfaces we get the Abstraction and while shifting the classes also no need to change the Main program logic.

JDBC :-

Dao.java:

```

package edu.dao;

public interface Dao {
    public void method();
}

public class DaoOneImpl implements Dao {
    public void method();
}

public class DaoTwoImpl implements Dao {
    public void method();
}

```

MainTest.java:

```

package edu.main;

public class MainTest {
    public static void main (String [] args) {
        Dao daoOne = new DaoOneImpl();
    }
}

```

Note: Using interface Here we are getting only Abstractions.

* If we are doing the Above Example , using properties file and creating the object using Class.forName(). Then while shifting focus DaoOne to DaoTwo , we no need to change the logic.

* Here we are getting " Two features from interface"

1. Abstraction

2. We are not changing the main logic

Dao.java:

```
package edu.dao;  
public interface Dao  
{  
    public void method1();  
}  
  
public class DaoOneImpl implements Dao  
{  
    public void method1()  
    {  
    }  
}  
  
public class DaoTwoImpl implements Dao  
{  
    public void method1()  
    {  
    }  
}
```

MainTest.java:

```
package edu.main;  
public class MainTest  
{  
    private static Properties properties = new Properties();  
    static  
    {  
        try  
        {  
            properties.load ( MainTest.class.getClassLoader().  
                getResourceAsStream ( "dao.properties" ));  
        }  
        catch ( Exception e )  
        {  
        }  
    }  
  
    public ( String [] args ) throws Exception  
    {  
        String daoClassName = properties.getProperty ( "daoClassName" );  
        Dao daoOne = ( Dao ) Class.forName ( daoClassName ).  
            newInstance ();  
        // Call the Methods  
    }  
}
```

dao.properties:

daoClassName = edu.dao.DaoOneImpl

Final Conclusion

(x)

- * If we don't know about the implementation details.

Let's take the interface (or) class for the Contract and then do the previous Polymorphism steps.

[A a = don't know (B)]

What is the Advantage of Object class in Java?

Class WhyObject

{

 public static (don't know Return Type) method (don't know Arg(x))

{

 return don't know Arg(x);

{

 public static void main (String [] args)

{

 Integer studentId = (Integer) method(); (new Integer(1));

 String studentName = (String) method ("N@IT");

{

}

- * don't know take it as Object

- * don't know Arg and don't know Return Type then take reference and return type as "Object".

Notes [Refer] :

1. SunMicroSystem they don't know what objects are going to be added to ArrayList using add() method by the Developers. So they have taken the argument for the add() method is Object.

Syntax: public boolean add (Java.lang.Object);

Q. SunMicroSystem they don't know what objects they need return while calling ArrayList get() method because it has returned the object what added by the developer and developer might be adding different object so it has to return different object.

Syntax: public java.lang.Object get(int);

So, return type of get() method should be Object

WhyObjectClass.java:

```
package edu.main;
public class WhyObjectClass {
    public static Object methods(Object object) {
        return object;
    }
    public static void main(String[] args) {
        Integer studentNo = (Integer) methods(new Integer(1));
        String studentName = (String) methods("N@IT");
    }
}
```

* If a method return type is Object which means 90% of the cases it never returns Object object most of the cases it is going to return different objects.

Eg: public java.lang.Object newInstance();

Q. Why The Type Cast?

```
Object obj = new String("N@IT");
```

* Call the length method on String object [Identify the length]
obj.length(); - wrong, Because if we are calling a method on a reference, the method must be there in the corresponding reference type class.

- * So, Type Cast to the original type we can use method.

```
String str = (String) obj;  
str.length();
```

- (*) * Here we are type casting String object to String type.

It is not possible in Java.

- * To type cast object object to String type.

```
Object obj = new Object();
```

String str = (String) obj; Wrong - No compile time

Error But Runtime Error [Sub class we can't type cast to Super class]

Spring Container

- * All the previous Examples, Developer is creating objects using Class.forName() and he is managing the object.

- (*) * If we are working with Spring, Spring Container creates the object [Internally using Class.forName()] and Spring container manages the objects.

- * Spring Container will solve all the dependency problems

[Action is dependent on Service]

Service is dependent on Dad

Session is dependent on SessionFactory

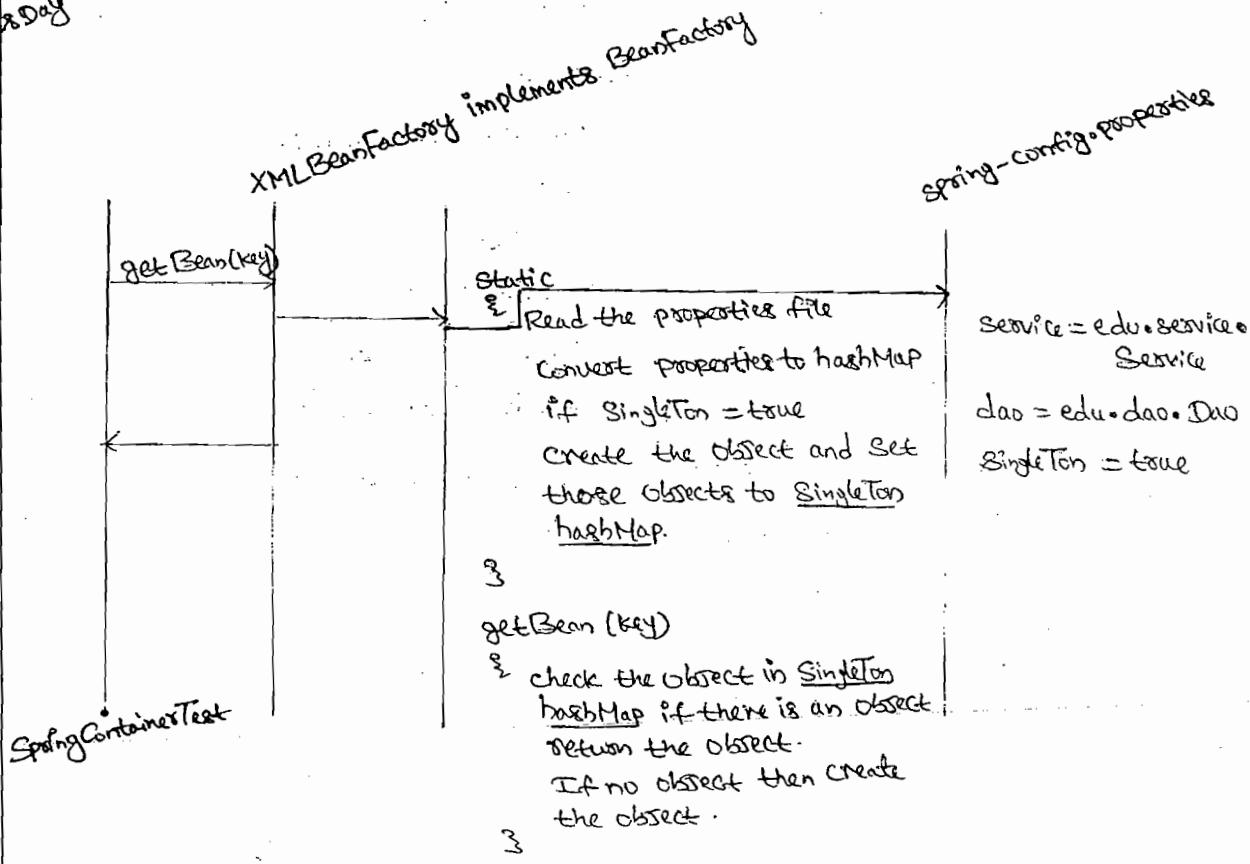
SessionFactory is dependent on Configuration]

- * Spring Container means its Any class which implements BeanFactory interface; If we create the implemented class object which means the Container is ready [while creating the object read the xml file].

all the

- * We can call XMLBeanFactory as Spring Container because it's implementing BeanFactory interface, we can call BeanFactory also as a Container. [If we are calling BeanFactory means indirectly we are talking about BeanFactory implemented class only]

19/11/09
Thursday



Create Spring Core Container and Which Should Handle the Object

Creation, Singleton And Object Life Cycle And Setter Injection

[Hard Coding The Logic] :

- * Here Spring Core Container Means BeanFactory implemented class.

Spring-config.properties:

`service = edu.service.Service`

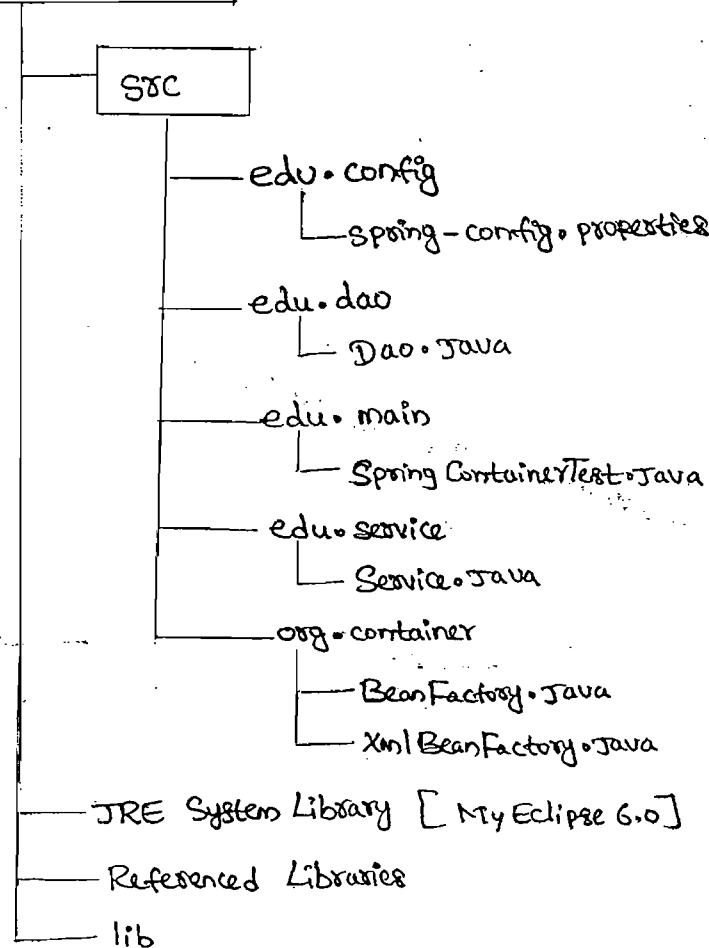
`dao = edu.dao.Dao`

`singleton = true`



★

Spring Container



Dao.java:

```
package edu.dao;
public class Dao {
    public void daoMethod() {
        System.out.println("Dao : daoMethod()");
    }
}
```

Service.java:

```
package edu.service;
import edu.dao.Dao;
public class Service {
    private Dao dao;
    public void setDao(Dao dao) {
        this.dao = dao;
    }
}
```

```
public void serviceMethod()
{
    System.out.println("Service.serviceMethod(). Start");
    dao.daoMethod();
    System.out.println("Service.serviceMethod(). End");
}
```

BeanFactory.java:

```
package org.container;
public interface BeanFactory
{
    public Object getBean(String key) throws Exception;
}
```

XmlBeanFactory.java:

```
package org.container;
public class XmlBeanFactory implements BeanFactory
{
    private static Map springConfigMap = new HashMap();
    private static Map singletonMap = new HashMap();

    static
    {
        Properties properties = new Properties();
        try
        {
            properties.load(XmlBeanFactory.class.getClassLoader()
                .getResourceAsStream("edu/config/spring-config.properties"));
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        springConfigMap.putAll(properties);
    }
}
```

```
String singleton = (String) springConfigMap.get("singleton");
if (StringUtil.equalsIgnoreCase(singleton, "true"))
{
    String className = (String) springConfigMap.get("dao");
}
```

```
Object obj = Class.forName(className).newInstance();
singletonMap.put("dao", obj);
```

```

    className = (String) springConfigMap.get("service");
    obj = Class.forName(className).newInstance();
    Service service = (Service) obj;
    service.setDao(dao);
    singletonMap.put("service", obj);
}

catch (Exception e)
{
}

public Object getBean(String key) throws Exception
{
    Object obj = singletonMap.get(key);
    if (obj == null)
    {
        return obj;
    }
    String className = (String) springConfigMap.get("dao");
    Dao dao = (Dao) Class.forName(className).
        newInstance();
    className = (String) springConfigMap.get(key);
    Service service = (Service) Class.forName(className).
        newInstance();
    service.setDao(dao);
    return service;
}

```

SpringContainerTest.java:

```

public class SpringContainerTest
{
    public void main(String[] args) throws Exception
    {
        BeanFactory beanFactory = new XmlBeanFactory();
        System.out.println("SpringContainerTest.main() start");
        Service service = (Service) beanFactory.getBean("service");
    }
}

```

```
service.serviceMethod();  
System.out.println("SpringContainerTest.main() End");  
Service service2 = (Service) beanFactory.getBean("service");  
System.out.println(" " + service1);  
System.out.println(" " + service2);
```

3

3

Problems:

1. We are not taking interface for Service and Dao.
2. We are creating the objects dynamically But hard coded the logic to push dao object into the Service.
To push Dynamically Apply Reflection
3. We are Applying Singleton logic on both Service and Dao at a time and it should be individual classes.

Advantages:

1. Here Object creation is done by Container and No logic for Singleton.
2. We don't required Dependent object creation logic by the Developer.
Container is going to push Dao object to Service means it will inject Dao to Service.

 **Note:** If we are working with Spring, XmlBeanFactory is provided by Spring just we need to write the Configuration.

20/11/09. WIRE FRAMEWORK
Friday

- * Core package is the most fundamental part of the Framework and provides the IOC and Dependency Injection features.
- * The basic concept here is the BeanFactory which provides a sophisticated implementation of the factory pattern which Removes the need for programmatic singletons and Allows you to decouple the Configuration and Specification of dependencies from your actual program logic.

[IOC]

INVERSION OF CONTROL / DEPENDENCY INJECTION

- * A kind of Inversion of Control (IOC).
- * " Hollywood Principle "
— Don't call me, I'll call you.
- * "Container" resolves (injects) dependencies of components by setting implementation object (push).
- * As opposed to component instantiation (or) Service Locator pattern where component locates implementation (pull).

Service And Dao Example [Developer Container & Spring Container] :

Normal JDBC [pulling] :

```
class Dao
{
    public void daoMethod()
}

// Your Calling
class Service
{
    private Dao dao = new Dao();
    public void daoMethod()
    {
        dao.daoMethod();
    }
}
```

// Container is Calling (Spring-Container) — "pushing"

```
class Service
{
    private Dao dao = null;
    public void setDao(Dao dao)
    {
        this.dao = dao;
    }
    public void daoMethod()
    {
        dao.daoMethod();
    }
}
```

```
<bean id="dao" class="edu.dao.Dao"/>
<bean id="service" class="edu.service.Service"
      scope="singleton">
```

```
<property name="dao">
    <ref bean="dao"/>
</property>
</bean>
```

⑤ What The Spring-Container is doing?

1. Let's create Object using Class.forName();

2. Inject dao to Service

```
.service.setDao(dao);
```

[It's done by container]

[Here Container is calling. Means It'll Call you (I means Container)]

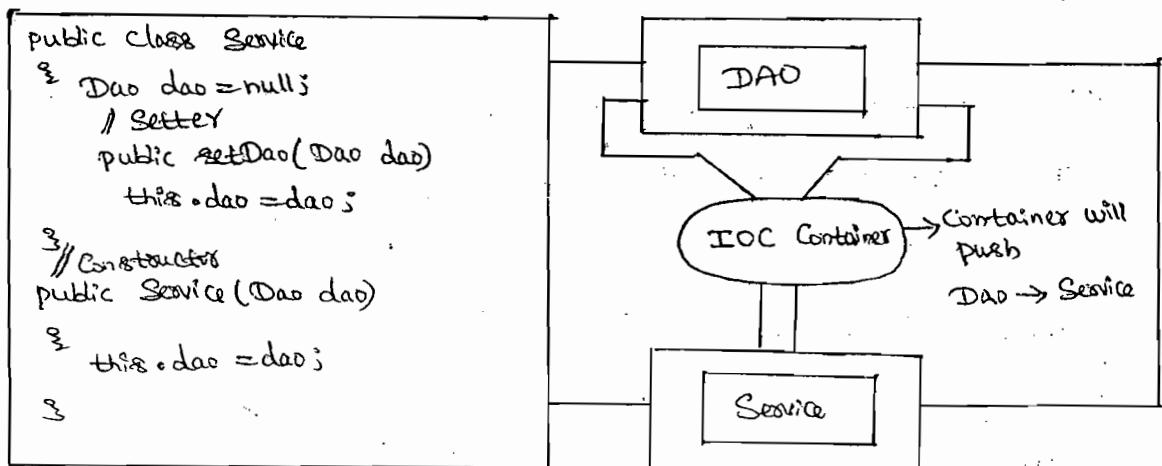
Benefits of DEPENDENCY INJECTION:

* Flexible — Avoid Adding lookup code in Business logic.

* Testable — No need to depend on External resources (or) Container for testing.

- * Maintainable - Allows reuse in different Application Environments by changing Configuration files instead of code.
- Promotes a Consistent Approach across all Applications and teams.

Inversion of Control / Dependency Injection:



Dependency Injection Styles:

- * Two Supported by Spring

1. Setter Based
2. Container Based

TWO DEPENDENCY INJECTION VARIANTS:

1. Constructor Dependency Injection.

→ Dependencies are provided through the Constructor of the Component.

2. Setter Dependency Injection

→ Dependencies are provided through the JAVABean Style Setter methods of the Component.

→ More popular than Constructor Dependency Injection

CONSTRUCTOR DEPENDENCY INJECTION :

```
public class ConstructorInjection  
{  
    private Dependency dep; // > Service  
    public ConstructorInjection(Dependency dep)  
    {  
        this.dep = dep;  
    }  
}
```

SETTER DEPENDENCY INJECTION :

```
public class SetterInjection  
{  
    private Dependency dep; // > Dao  
    public void setDependency(Dependency dep)  
    {  
        this.dep = dep;  
    }  
}
```

Spring Container :

1. BeanFactory
2. Application Context
3. Web Application Context

BeanFactory :

- * Lightweight container that loads bean definitions and manages your beans.
- * Knows how to serve and manage a Singleton (or) prototype defined bean.
- * Responsible for managing bean lifecycle.
- * Injects dependencies into defined beans when served
- * Avoids the usage of Singletons and factories.

BeanFactory factory = XmlBeanFactory(new FileSystemResource("c:/beans.xml"))

BeanFactory factory = new XmlBeanFactory(new
ClassPathResource("beans.xml"));

Note: "ClassPathResource()" is more "Preferable"

ApplicationContext:

* ApplicationContext provides all the features that BeanFactory provides additionally it.

1. Application Contexts provide a means for resolving text messages, including support for internationalization (I18N) of those messages.
2. Application Contexts provide a generic way to load file resources, such as images.
3. Application Contexts can publish events to beans that are registered as listeners.

SEVERAL WAYS TO CONFIGURE A CONTEXT:

1. ClassPathXMLApplicationContext — Loads beans Configuration from classpath.

ApplicationContext context = new

ClassPathXMLApplicationContext("beans.xml");

2. FileSystemXmlApplicationContext — Load from the file system (relative/ absolute)

ApplicationContext context = new

FileSystemXmlApplicationContext("beans.xml");

3. XMLWebApplicationContext — Configuration for a Web Application Under Servlet Context.

Notes :

- * BeanFactory is Lightweight And Application Context will be Heavyweight
- * For Mobile Applications & Applet Applications use BeanFactory
- * For Any Web Applications we can use ApplicationContext
- * For MVC Based Web Applications, we can use Web Application Context
- * In BeanFactory it doesn't Create the objects while reading the Configuration file. It will Create the objects while calling getBean() method.
- * In ApplicationContext it will Create All the objects while reading the Configuration file.

Note: If we are reading the Spring-Configuration file from the Main program it is "wrong".

Let's Create a "Helper Class" or "Util class" and read the Configuration file using a "static block" or "creates a static reference object".

```
private static ApplicationContext context = new  
ClassPathXmlApplicationContext("bean.xml");  
(or)
```

```
ApplicationContext context = null;
```

```
static
```

```
{
```

```
context = new ClassPathXmlApplicationContext("bean.xml");  
//Handle the Exception.
```

```
}
```

21/11/09
Saturday

Implement a Spring example - injecting a value to Spring Object.

1. Create a Java project
2. Create a Lib folder and Copy the 2 Jars [commons-logging-1.0.4.jar
spring.jar]
3. Create a XML file and Write a Bean [for Spring class]
[spring-config.xml]

* Open the XML file and Copy past the doctype

<!DOCTYPE >, then it gives <beans> tag
(or)

* Go to the Design,

right click on the DOCTYPE → Add After → Beans

right click on the beans → Add Child → bean

<beans> // root tag

<bean id = "studentName" class = "java.lang.String" >
</bean> // child tag

</beans>

* Open the String class and check what way it is expecting
the Value [Constructor (or) Setter]

```
public final class String
{
    public String (String value)
}
```

30

- * The Above class is expecting the value using Constructor
only. It is not expecting using Setter - So, we can
Apply only Constructor injection.
- * If it is Constructor injection, we use Constructor
<constructor-arg> tag.

Object:

* Note: While injecting, we need to more concentrate on the argument type. Here the argument type is String, so we can use <Value> tag

```
<beans>
  <bean>
    <constructor-arg>
      <value> N@It </value>
    </constructor-arg>
  </bean>
</beans>
```

Write The Java Program And Read The XML file:

spring-config.xml :

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEANS 2.0//EN"
  "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

<beans>

```
  <bean id = "studentName" class = "java.lang.String">
```

<constructor-arg>

```
      <value> N@It </value>
```

</constructor-arg>

</bean>

</beans>

Main Program:

```
package edu.test;
```

```
public class FileSystemResourceTest
```

```
{
```

```
  public static void main (String [] args)
```

```
  {
```

```
    BeanFactory factory = new XmlBeanFactory (new
```

```
    FileSystemResource ("Complete File Location F:/.../spring-config.xml"))
```

String studentName = (String) factory.getBean("studentName");
System.out.println("Student Name = " + studentName);

23/11/16
Monday

Note: In FileSystemResource we are providing the complete path and we should not give Complete path in the projects. It should be in "Dynamic".

* So, we use "ClassPathResource" and give the Package Structure.

```
BeanFactory factory = new XmlBeanFactory(new  
ClassPathResource("edu/config/spring-config.xml"));
```

* Implement the same Example Using ApplicationContext

```
ApplicationContext context = new FileSystemXmlApplicationContext  
("Complete Path F:/.../spring-config.xml");
```

* File System Xml ApplicationContext works same as FileSystemResource

In ApplicationContext instead of FileSystemResource we use
FileSystemXmlApplicationContext

ClassPath ApplicationContext:

```
ApplicationContext context = new ClassPathXmlApplicationContext  
("edu/config/spring-config.xml");
```

* ClassPath Xml ApplicationContext works same as ClassPathResource

In ApplicationContext instead of ClassPathResource we use
ClassPathXmlApplicationContext.

Student Name: Aman

* Let's implement All the Examples with ApplicationContext and ClassPathXmlApplicationContext.

23/11/09
Monday

How To Get The Session Object Using Spring Core Container:

* To Create the Session object, we required

1. Cfg file, HBM file
2. Configuration object and SessionFactory.

* Developer is going to write some logic to read the configuration files and To Create Configuration and Sessionfactory objects.

* Let's ask the developer Inject All the required inputs to create the Sessionfactory to a Predefined class. Here we are going to create the predefined class.

Create LocalSessionFactoryBean class:

* First Create the Connection object.

→ To create the Connection object Let's Ask the Developer Inject URL, Username and password and we implement the Connection logic.

* Using a simple Java program, If we want to implement the DataSource Concept our class must have to implement Sun provided DataSource interface [Here interface is for Contract]

* Any people in the Java world will be implementing DataSource interface.

Note: So that, The Developer can shift from one Provider to Another provider.

1. Apache:

```
public class BasicDataSource implements DataSource  
{ }
```

2. Spring:

```
public class DriverManagerDataSource extends  
AbstractDataSource  
{ }
```

```
public abstract class AbstractDataSource implements  
DataSource  
{ }
```

3. Our class:

```
public class StudentDataSource implements DataSource  
{ }
```

* Let's Ask the Developer, Insert All the required
cfg file details to LocalSessionFactoryBean

```
public class LocalSessionFactoryBean implements InitializingBean,  
DisposableBean  
{ }
```

```
private Resource[] mappingLocations;
```

```
private Properties hibernateProperties;
```

```
private DataSource dataSource;
```

 mappingLocations — All the hbm files

hibernateProperties — dialectClass, show_SQL,
hbm2ddl, ... etc.

dataSource — URL, username, pass, & driverClass.

* We need to write the logic, To Create SessionFactory
and where should we have to write the logic.

- * While creating the LocalSessionFactoryBean object by the Container, it should call some logic automatically. If it has to call some logic the class should implement InitializingBean and has to override afterPropertiesSet() method.

Note:

InitializingBean:

- * If a class is implementing InitializingBean whenever the Container creates object for the class it will call afterPropertiesSet() method.
- * Here the Developer is Injecting all the required parameters to LocalSessionFactoryBean has a API people we need to implement InitializingBean and using the Developer parameters we need to create the SessionFactoryBean object logic inside the afterPropertiesSet() method.
- * So, whenever the Developer requires Session object he has to Configure LocalSessionFactoryBean in the Spring configuration. while Injecting the required parameters and from the main program he has to get SessionFactoryBean object and he has to call getSession() method.

```
LocalSessionFactoryBean factory = (LocalSessionFactoryBean)
context.getBean("factory");
```

```
Session session = factory.getSession();
```

[We don't require "SessionUtil class" Because SessionFactoryBean is single (In config.xml by default singleton = "true") we don't require Configuration object create and SessionFactory object create just we need to work LocalSessionFactoryBean because the logic is thrown in afterPropertiesSet() ...]

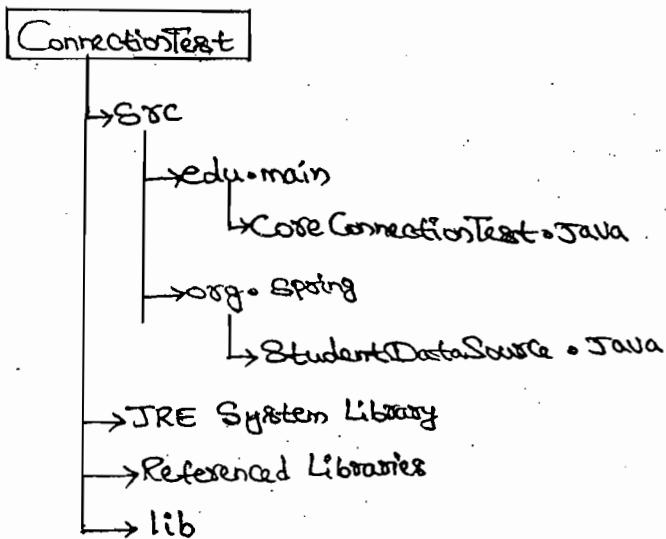
25/11/09
Wednesday

Java Code:

```
package edu.main;
import java.sql.Connection;
import java.sql.DriverManager;

public class ConnectionTest {
    // Java code
    public void main (String [] args) throws Exception {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection connection = DriverManager.getConnection ("jdbc:
        oracle:thin:@localhost:1521:SERVER", "scott", "tiger");
        System.out.println ("Connection: " + connection);
    }
}
```

- * The Common logic, let's separate to a Separate class and provide the flexibility to the user.
- * To set the values in 2 ways or To inject the values in 2 ways.
 1. Constructors
 2. Setter
- * The class should be common - so, let's implement Sun provided DataSource interface.



* The Common Logic, API might be providing below class.

StudentDataSource.java :

```
public class StudentDataSource implements DataSource
{
    private String driverClassName;
    private String url;
    private String username;
    private String password;

    public void setDriverClassName (String driverClassName)
    {
        this.driverClassName = driverClassName;
    }

    public void setUrl (String url)
    {
        this.url = url;
    }

    public void setUsername (String username)
    {
        this.username = username;
    }

    public void setPassword (String password)
    {
        this.password = password;
    }

    public StudentDataSource()
    {
    }

    public StudentDataSource (String driverClassName, String url,
                           String username, String password)
    {
        this.driverClassName = driverClassName;
        this.url = url;
        this.username = username;
        this.password = password;
    }

    public Connection getConnection (String username, String password)
    throws SQLException
    {
        return null;
    }

    public PrintWriter getLogWriter() throws SQLException
    {
        return null;
    }
}
```

```
    public void setLogWriter( PrintWriter argo) throws SQLException  
    { }  
    public void setLoginTimeout( int argo) throws SQLException  
    { }  
}
```

* In the Main program , Create the object for StudentDataSource and call the getConnection() method of the class is dependent on certain inputs before calling the getConnection() provide the values .

CoreConnectionTest.java :

```
public class CoreConnectionTest  
{  
    // Java code  
    public( String [] args ) throws Exception  
    {  
        StudentDataSource studentDataSource = new  
            StudentDataSource();  
        studentDataSource.setDriverClass( "oracle.jdbc.driver.  
            OracleDriver" );  
  
        studentDataSource.setUrl( "jdbc: oracle: thin: @localhost  
            :1521: SERVER" );  
  
        studentDataSource.setUsername( "scott" );  
        studentDataSource.setPassword( "tiger" );  
  
        // Before Calling getConnection() , set the Required parameters  
        Connection connection = studentDataSource.getConnection();  
        System.out.println( "Connection : " + connection );  
    }  
}
```

Advantage:

- * We are not writing the logic for Class·forName
- * The getConnection() method is there in API.

DisAdvantages:

- * We are Creating the API object
- * We are Hard Coding the required inputs in the java file.

Let's Do it Using Spring:

- * Using the SpringBean [StudentDataSource] Configure the Input in XML.
- * Then Spring takes care of "Object Create and Injection".

spring-config.xml :

<beans>

<!-- 1. Create The DataSource Object -->

<bean id = "dataSource" class = "org.springframework.jdbc.datasource.DriverManagerDataSource">

<!-- Constructor OR Setter Injection -->

<!-- Constructor Injection -->

<constructor-arg index = "0">

<value> oracle.jdbc.driver.OracleDriver </value>

</constructor-arg>

<constructor-arg index = "1">

<value> jdbc:oracle:thin:@localhost:1521:SERVER </value>

</constructor-arg>

<constructor-arg index = "2">

<value> scott </value>

</constructor-arg>

<constructor-arg index = "3">

<value> tiger </value>

```
<!-- Setter Injection -->  
<property name = "driverClassName">  
<value> oracle.jdbc.driver.OracleDriver </value>  
</property>  
<property name = "url">  
<value> jdbc:oracle:thin:@localhost:1521:SERVER </value>  
</property>  
<property name = "username">  
<value> scott </value>  
</property>  
<property name = "password">  
<value> tiger </value>  
</property>  
</bean>  
</beans>
```

Main Program:

ConnectionTest.java:

```
public class ConnectionTest  
{  
    public static ApplicationContext context = new  
        ClassPathXmlApplicationContext("edu/config/spring-config.xml");  
    public ( String[] args) throws Exception  
    {  
        DataSource dataSource = (DataSource) context.  
            getBean("dataSource");  
        Connection connection = dataSource.getConnection();  
        System.out.println("Connection: " + connection);  
    }  
}
```

Q. What is the use of TODO Statement?

Syntax: //TODO

* If there is any pending dependency task in the code, we give TODO and the comment.

//TODO Need to implement xxxLogic.

* Means There is some pending task. So, Code Review then don't consider this.

* While implementing in the Editor click on the icon and implement the logic.

Q. What The Spring is Doing Internally:

* Create the DataSource object using Class.forName() and while create the object it will pass the value to constructor
(OR) After creating the object it will call the Setter method and it will set the values.

new StudentDataSource (driverClass, ...); [Using Class.forName()]
(OR)

ref. setDriverClass (driverClass);

ref. setUrl (url);

Note: Internally it will do same as the 2nd program

Points To Remember while working with Spring:

Q.

Java Coding:

- * Create the objects
- * Implement the logic
- * Call the methods.

Spring Coding:

- * Configure the XML
- * Create the methods there in the API class.

26/11/09
Thursday

Note: We never implement StudentDataSource type & similar.
These type of classes will be provided by the API people.

Eg: 1. DriverManagerDataSource
2. BasicDataSource.

Configure the XML file:

<bean id = "dataSource" class = "org.springframework.jdbc.datasource.DriverManagerDataSource">

<!-- Check for Constructor & Setter Method in DataSource
It allows Constructor and Setter Approach -->

Inject the Values

</bean>

Note: In the previous Example Just change className in the XML file.

<bean id = "dataSource"
class = "org.springframework.jdbc.datasource.DriverManagerDataSource" />

We can use Constructor And Setter

</bean> (OR)

<bean id = "dataSource"

class = "org.apache.commons.dbcp.BasicDataSource" />

Don't use Constructor Approach for Apache

Because, There is no Argument Constructor

</bean>

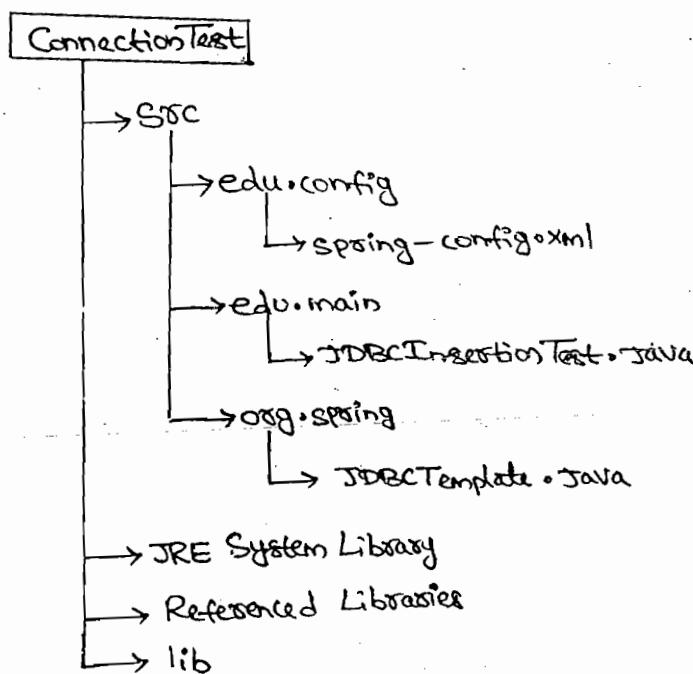
f Classes

i. People

Implement A JDBC Example Using Spring Core:

- * To simplify the JDBC Logic. Let's take a Simple Java class [JdbcTemplate OR SpringBean] and implement the JDBC common Logic in that.
- * To implement the JDBC Logic, we required connection object and connection is available with DataSource.
- * So Let's Inject Database to JdbcTemplate [In 2 ways]

Directory Structure:



spring-config.xml:

<beans>

```
<!-- Configure DataSource using DriverManagerDataSource -->
<bean id = "jdbcTemplate" class = "org.springframework.jdbc.core.JdbcTemplate">
    <property name = "dataSource">
        <ref bean = "dataSource"/>
    </property>
</bean>
```

JDBCTemplate.java

```
public class JDBCTemplate
{
    private DataSource dataSource;

    public JDBCTemplate()
    {
        this.dataSource = null;
    }

    public void setDataSource(DataSource dataSource)
    {
        this.dataSource = dataSource;
    }

    public JDBCTemplate(DataSource dataSource)
    {
        this.dataSource = dataSource;
    }

    public int update(String query, Object object[])
    {
        Connection connection = null;
        PreparedStatement ps = null;
        int count = 0;

        try
        {
            connection = dataSource.getConnection();
            ps = connection.prepareStatement(query);
            ps.setString(1, (String) object[0]);
            ps.setString(2, (String) object[1]);
            count = ps.executeUpdate();
        }
        catch(SQLException e)
        {
            System.out.println("Exception." + e);
            // Convert SQLException to RuntimeException
        }
        finally
        {
            DbUtils.closeQuietly(connection, ps, null);
        }
        return count;
    }
}
```

JDBCInsertionTest.java:

```
package edu.main;

public class JDBCInsertionTest
{
    private static ApplicationContext context = new
        ClassPathXmlApplicationContext("edu/config/spring-config.xml");

    public static void main(String[] args) throws Exception
    {
        JdbcTemplate jdbcTemplate = (JdbcTemplate)
            context.getBean("jdbcTemplate");

        int count = jdbcTemplate.update("UPDATE STUDENT
            SET SNAME=? WHERE SNO=?",
            new Object[]{ "N@IT-Update", "1" });

        System.out.println("No. of Records Updated : " + count);
    }
}
```

Note: We no need to implement "JdbcTemplate" class which will be provided by the "Spring". Our job will be "Configure the Template in the XML file" and Get the Template object and Call the Methods to execute the query. But internally Template is based on "polymorphism logic" and "Callbacks logic".

```
public abstract class JdbcAccessor
{
    public void setDataSource(DataSource dataSource)
    {
        ---  

        ---  

    }
}
```

```
public class JdbcTemplate extends JdbcAccessor
{
    public JdbcTemplate(DataSource dataSource)
    {
        ---  

    }
}
```

* Instead of our Template, Configure Spring Template in the XML

```
<bean id="JdbcTemplate"
      class="org.springframework.jdbc.core.JdbcTemplate">
</bean>
```

Main:

```
JdbcTemplate jdbcTemplate = (JdbcTemplate)
    context.getBean("JdbcTemplate");
```

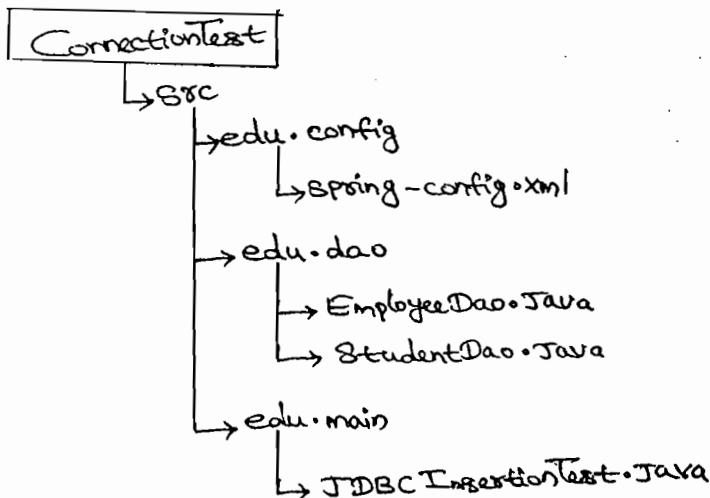
```
int Count = jdbcTemplate.update("UPDATE STUDENT SET
    SNAME=? WHERE SNO=?",
    new Object[] { "N@IT", "14" });
```

We have to use JdbcTemplate object in Dao classes.

* So, Let's Inject Template to Dao.

* In the Main program, Get the Dao and call the Method

27/11/09
Friday * Let's implement a program and which should update employee record and should update student record.



plate

spring-config.xml:

```
<beans>

    <!-- 1. Configure DataSource -->
    <!-- 2. Configure Template -->
    <bean id = "JdbcTemplate"
          class = "org.springframework.jdbc.core.JdbcTemplate">
        <property name = "dataSource">
            <ref bean = "dataSource"/>
        </property>
    </bean>

    <bean id = "studentDao"
          class = "edu.dao.StudentDao">
        <property name = "JdbcTemplate">
            <ref bean = "JdbcTemplate"/>
        </property>
    </bean>

    <bean id = "employeeDao"
          class = "edu.dao.EmployeeDao">
        <property name = "JdbcTemplate">
            <ref bean = "JdbcTemplate"/>
        </property>
    </bean>

</beans>
```

EmployeeDao.java:

```
public class EmployeeDao
{
    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate (JdbcTemplate jdbcTemplate)
    {
        this.jdbcTemplate = jdbcTemplate;
    }
}
```

```
public int updateEmployee (String employeeName, String employeeNo)
{
    return jdbcTemplate.update ("UPDATE EMPLOYEE
        SET ENAME=? WHEREENO=?",
        new Object[] {employeeName, employeeNo});
}
```

StudentDao.java:

```
public class StudentDao
{
    private JdbcTemplate jdbcTemplate;
    public void setJdbcTemplate (JdbcTemplate jdbcTemplate)
    {
        this.jdbcTemplate = jdbcTemplate;
    }
    public int updateStudent (String studentName, String studentNo)
    {
        return jdbcTemplate.update ("UPDATE STUDENT
            SET SNAME=? WHERE SNO=?",
            new Object[] {studentName, studentNo});
    }
}
```

* There are Two Dao classes. So, Two times we are writing setJdbcTemplate() method. To Inject the Template object to the Dao classes.

(*) * Let's simplify the code using Abstract class and extend the Abstract class to All the Dao's.

AbstractDao.java:

```
public class AbstractDao
{
    private JdbcTemplate jdbcTemplate;
    getters & setters
}
```

String

EmployeeDao.java:

```
public class EmployeeDao extends AbstractDao  
{  
    public int updateEmployee( String employee, String employeeName )  
    {  
        return getJdbcTemplate().update( "UPDATE EMPLOYEE SET  
            ENAME = ? WHERE ENO = ? ", new Object[] { employeeName,  
            employeeNo } );  
    }  
}
```

StudentDao.java :

```
public class StudentDao extends AbstractDao  
{  
    public int updateStudent( String studentNo, String studentName )  
    {  
        return getJdbcTemplate().update( "UPDATE STUDENT SET  
            SNAME = ? WHERE SNO = ? ", new Object[] { studentName,  
            studentNo } );  
    }  
}
```

Note: AbstractDao is required for all the projects. Why the developer should implement the abstract class. If the framework provides abstract class then it could be better.

* Spring provides the below classes [Abstract]

		All ORMS
JDBC	— jdbcDaoSupport	*
Hibernate	— hibernateDaoSupport	hibernate * * Support *
JPA	— JpaDaoSupport	* Jpa * * Support *
JDO	— jdoDaoSupport	* jdo * * Support *
IBatis	— SqlMapClientDaoSupport	* ibatis * * Support *
Toplink	— TopLinkDaoSupport	* toplink * * Support *

* All these classes should be extended to Dao classes in the projects [It is based on the ORM we are using].

EmployeeDao.java:

```
public class EmployeeDao extends JdbcDaoSupport  
{  
    public int updateEmployee (String employeeNo, String employeeName)  
    {  
        return getJdbcTemplate().update ("UPDATE EMPLOYEE  
        SET ENAME=? WHERE ENO=? ", new Object[]  
        { employeeName, employeeNo });  
    }  
}
```

StudentDao.java:

```
public class StudentDao extends JdbcDaoSupport  
{  
    public int updateStudent (String studentNo, String  
        studentName)  
    {  
        return getJdbcTemplate().update ("UPDATE STUDENT  
        SET SNAME=? WHERE SNO=? ", new Object[]  
        { studentName, studentNo });  
    }  
}
```

spring-config.xml:

```
<beans>  
    ---  
    ---  
    <bean id = "jdbcTemplate"  
        class = "org.springframework.jdbc.core.JdbcTemplate">  
        <property name = "dataSource">  
            <ref bean = "dataSource"/>  
        </property>  
    </bean>  
</beans>
```

Q. How Spring creates Template? :

1. Initially it creates DataSource object

2. Inject DataSource to Template

Class.forName("JdbcTemplate").newInstance();
• setDataSource(dataSource);

3. Inject Template to Dao

Q. Can't you Remove Injecting DataSource To Template? And
Inject DataSource To Dao. But we required Template object
in the Dao Class?

Yes, we can inject DataSource to Dao, Because Dao
Super class [JDBCDAOsupport] is having setDataSource()
method and it is going to create JDBCTemplate object.

So, we can Template object in our Dao classes while
calling getJdbcTemplate(). [getJdbcTemplate() method will be
there in JDBCDAOsupport]

Internal code:

```
public abstract class JDBCDAOsupport ...  
{  
    public final void setDataSource(Database dataSource)  
    {  
        this.jdbcTemplate = createJdbcTemplate(dataSource);  
    }  
    ...  
    protected JDBCTemplate createJdbcTemplate(Database dataSource)  
    {  
        return new JDBCTemplate(dataSource);  
    }  
}
```

* Here Spring will create template object while Injecting

Final Example

spring-config.xml:

```

<!-- Configure DataSource -->
<bean id = "studentDao" class = "edu.dao.StudentDao">
    <property name = "dataSource">
        <ref bean = "dataSource"/>
    </property>
</bean>

<bean id = "employeeDao" class = "edu.dao.EmployeeDao">
    <property name = "dataSource">
        <ref bean = "dataSource"/>
    </property>
</bean>

```

StudentDao.java:

Note: Copy the same as previous code.

EmployeeDao.java:

Note: Copy the same as previous code

Main.java:

PSVM (String [] args) throws Exception

`StudentDao studentDao = (StudentDao) context.getBean("studentDao");`

`int studentCount = studentDao.updateStudent("1",
 "N@IT update");`

`EmployeeDao employeeDao = (EmployeeDao) context.getBean("employeeDao");`

`int employeeCount = employeeDao.updateEmployee("1",
 "N@IT update");`

`sop ("No. of Records update:" + studentCount);`

`sop ("No. of Records update:" + employeeCount);`

How To Build The SessionFactory "Without Writing Configuration File"

(a) Build The SessionFactory If We Are "Having Cfg file Details":

Steps To Implement:

1. Creating The Configuration object and Set All the properties to Configuration object

```
<property name = "hibernate.connection.driver-class">  
    oracle.jdbc.driver.OracleDriver </property>  
<property name = "hibernate.dialect"> org.hibernate.dialect.  
    OracleDialect </property>  
    - - - - -  
    - - - - -  
    Like, w/o  
    username  
    password
```

```
Configuration configuration = new Configuration();  
configuration.setProperty("hibernate.connection.driver-class",  
    "oracle.jdbc.driver.OracleDriver");  
configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.  
    OracleDialect");  
    - - - - -
```

Key — property name

Value — property value.

2. Set All the hbm file details in the form of Stream object to configuration object and then call buildSessionFactory().

```
configuration.addInputStream( • class • getClassLoader() •  
    getResourceAsStream("edu/mappings/student.hbm.xml"));
```

SessionFactory sessionFactory = configuration.

buildSessionFactory();

How To Create Session without providing DB details
[url, username, password & hbm2ddl] in the cfg file:

```
<hibernate-configuration>
  <session-factory>
    <property name="show-sql">true</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <!-- Don't provide hbm2ddl -->
    <!-- Mapping files -->
    <mapping resource="edu/mapping/student.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Main:

PSVM (String [] args) throws Exception

```
{  
    Configuration configuration = new Configuration().configure  
        ("edu/config/hibernate.cfg.xml");
```

```
SessionFactory sessionFactory = configuration.  
    buildSessionFactory();
```

// Get The Connection Object

```
Class.forName ("oracle.jdbc.driver.OracleDriver");  
Connection connection = DriverManager.getConnection ("jdbc:  
    :oracle:thin:@localhost:1521: SERVER", "scott",  
    "tiger");
```

```
Session session = sessionFactory.openSession (connection);  
String s = "Session." + session;
```

}

We can observe Two Concepts in previous Examples:

1. We are not giving DataBase Details in the cfg file where as in that case Explicitly we need to pass connection object openSession().
2. Without cfg file we can build sessionFactory. In these case Explicitly we need to set cfg file details to configuration.

Let's Simplify Session object creation logic using Spring:

1. Let's create a simple JavaBean class and implement the common logic and Here take the class name as LocalSessionFactoryBean.
2. Let's Ask the developer Inject the cfg file details in 3 parts.
 - 2.1. Ask the developer, provide the DataBase details in the form of DataSource object. So, we can get the connection from DataSource and we can pass to openSession() method.
 - 2.2. Let's Ask the Developer, provide All the hibernate details in the form of List and As a API person create (or) convert Stream Objects and pass to the configuration object [Need to Iterate the List].
 - 2.3. Let's Ask the Remaining Details in the form of Properties object [dialect class, show-sql] and

Note: Based on the previous sessions [2.1, 2.2 & 2.3]

LocalSessionFactoryBean should contain 3 Setter methods.

One should take properties object, The other should take List<Object> Array and DataSource.

- * One method should convert Stream object
- * SessionFactory should be created only once. So, Let's implement InitializingBean interface and implement the logic to create the SessionFactory inside afterPropertiesSet method.
- * SessionFactory should closed once. So, implements DisposableBean and implement the logic to close the SessionFactory inside destroy() method.

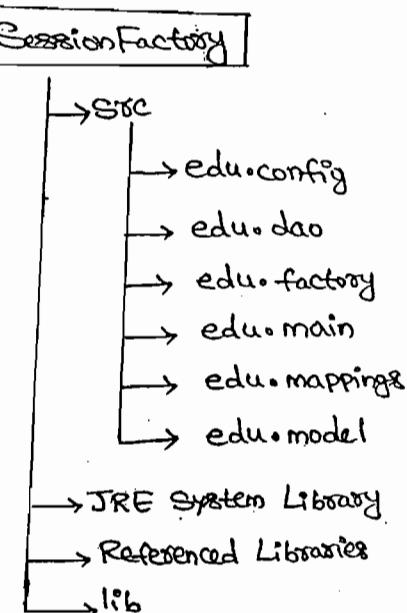
11/12/09

Tuesday

Working With Dao With Session Object:

- * We required Session objects in all the Dao classes and we can get the session object from LocalSessionFactoryBean. So Inject LocalSessionFactoryBean to the Dao classes.
→ We required the Session object in the Dao classes. So, Inject LocalSessionFactoryBean to the Dao classes using setter Approach. So write setSessionFactory() method in the Dao classes.

Directory Structure:



the
Spring-config.xml :

```
<beans>

    <!-- 1. Create The DataSource Object -->
    <!-- Do The DataSource Configuration using DriverManagerDataSource-->
    <!-- 2. -->

    <bean id = "factory" class = "org.springframework.orm.hibernate3.
                                LocalSessionFactoryBean">

        <property name = "mappingResources">
            <list>
                <value>edu/mapping/student.hbm.xml</value>
                <value>edu/mapping/employee.hbm.xml</value>
            </list>
        </property>

        <property name = "hibernateProperties">
            <props>
                <prop key = "hibernate.dialect">org.hibernate.dialect.
                                                Oracle9Dialect</prop>
                <prop key = "hibernate.show-sql">true</prop>
            </props>
        </property>
    
```

```
<!--property name = "dataSource"-->  
<ref bean = "dataSource"/>  
</property>  
</bean>  
<bean id = "studentDao" class = "edu.dao.StudentDaoImpl">  
<property name = "sessionFactory">  
    <ref bean = "factory"/>  
</property>  
</bean>  
<bean id = "employeeDao" class = "edu.dao.EmployeeDaoImpl">  
<property name = "sessionFactory">  
    <ref bean = "factory"/>  
</property>  
</bean>  
</beans>
```

EmployeeDao.java:

```
public interface EmployeeDao {  
    public void insertEmployee (Employee employee);  
}
```

EmployeeDaoImpl.java:

```
public class EmployeeDaoImpl implements EmployeeDao {  
    private SessionFactory sessionFactory;  
    public void setSessionFactory (SessionFactory sessionFactory) {  
        this.sessionFactory = sessionFactory;  
    }  
    public void insertEmployee (Employee employee) {  
        Session session = sessionFactory.openSession();  
        Transaction transaction = session.beginTransaction();
```

```
try
{
    session.save(employee);
    transaction.commit();
}
catch(HibernateException e)
{
    transaction.rollback();
}
```

StudentDao.java:

```
public interface StudentDao
{
    public void insertStudent(Student student);
}
```

StudentDaoImpl.java:

```
public class StudentDaoImpl implements StudentDao
{
    private SessionFactory sessionFactory;
    public void setSessionFactory(SessionFactory sessionFactory)
    {
        this.sessionFactory = sessionFactory;
    }
    public void insertStudent(Student student)
    {
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();
        try
        {
            session.save(student);
            transaction.commit();
        }
        catch(Exception e)
        {
            transaction.rollback();
        }
    }
}
```

2/12/09
Wednesday

Note: In the Above Example, we are Injecting SessionFactory

(1/6)

to Dao class and we are writing setSessionFactory in Both
the Dao classes.

* Instead of Writing Two times Apply Abstraction & Extend
Abstract class to all the Dao classes.

AbstractDao.java :

```
public interface AbstractDao
{
    public void setSessionFactory(SessionFactory sessionFactory);
    public SessionFactory getSessionFactory();
}
```

AbstractDaoImpl.java :

```
public class AbstractDaoImpl implements AbstractDao
{
    private SessionFactory sessionFactory;
    public SessionFactory getSessionFactory()
    {
        return sessionFactory;
    }
    public void setSessionFactory(SessionFactory sessionFactory)
    {
        this.sessionFactory = sessionFactory;
    }
}
```

EmployeeDao.java :

```
public interface EmployeeDao extends AbstractDao
{
    public void insertEmployee(Employee employee);
}
```

in Factor

Both

EmployeeDaoImpl.java :

public class EmployeeDaoImpl extends AbstractDaoImpl implements EmployeeDao

{

 public void insertEmployee (Employee employee)

 {
 Session session = getSessionFactory () . openSession () ;

 Transaction transaction = session . beginTransaction () ;

 try

 session . save (employee) ;

 transaction . commit () ;

 }

 catch (HibernateException e)

 {
 transaction . rollback () ;

 }

}

StudentDao.java :

public interface StudentDao extends AbstractDao

 {
 public void insertStudent (Student student) ;

 }

StudentDaoImpl.java :

public class StudentDaoImpl extends AbstractDaoImpl implements StudentDao

 {
 public void insertStudent (Student student)

 }

 Session session = getSessionFactory () . openSession () ;

 Transaction transaction = session . beginTransaction () ;

 try

 session . save (student) ;

 transaction . commit () ;

 }

 catch (HibernateException e)

 {

 transaction . rollback () ;

 }

3

3

Note: In Spring Application we don't require exception Handling in the "Dao classes". But in the Example, we are doing. Because, we are not taking help from Spring for Exception Handling in this Example.

* If we are working with Hibernate code Transaction Management is must and if we take help from Spring Template class [HibernateTemplate] we don't require Transaction Management.

How To Configure Template:

```
public abstract class HibernateAccessor ...  
{  
    public void setSessionFactory(SessionFactory sessionFactory)  
    {  
        this.sessionFactory = sessionFactory;  
    }  
  
    public class HibernateTemplate extends HibernateAccessor ...  
    {  
        public HibernateTemplate(SessionFactory sessionFactory)  
        {  
            setSessionFactory(sessionFactory);  
            afterPropertiesSet();  
        }  
    }  
}
```

// The Below method is there in HibernateAccessor ...

```
public void afterPropertiesSet()  
{  
    if (getSessionFactory() == null)  
    {  
        throw new IllegalArgumentException("Property 'sessionFact  
        is required");  
    }  
}
```

* Till now we Injected SessionFactory to Dao. Now, Inject Template to Dao and work with Template.

Spring-config.xml:

```
<bean id = "hibernateTemplate"
      class = "org.springframework.orm.hibernate3.HibernateTemplate">
    <property name = "sessionFactory">
      <ref bean = "factory"/>
    </property>
</bean>
<bean id = "employeeDao"
      class = "edu.dao.EmployeeDaoImpl">
    <property name = "hibernateTemplate">
      <ref bean = "hibernateTemplate"/>
    </property>
</bean>
```

Note: In the AbstractDaoImpl class we have written setSessionFactory, it should be changed to setHibernateTemplate and need to work with template in the Dao classes.

(*) * AbstractDaoImpl is required for all the projects. Then why the Developer should implement this class, Let's Extend .. "HibernateDaoSupport". which contains the common methods like "AbstractDaoImpl".

Note: We are injecting SessionFactory to Template and Template to Dao. Because we required Template in Dao classes.

Let's Inject SessionFactory Directly to Dao and HibernateDaoSupport class will prepare Template to us ...

```
public abstract class HibernateDaoSupport extends DaoSupport  
{  
    public final void setSessionFactory(SessionFactory sessionFactory)  
    {  
        this.hibernateTemplate = createHibernateTemplate(sessionFactory);  
    }  
}
```

- * "Template" is powerful than "Session". Because, Transaction is not required and we can store list of objects Directly in the Database... etc. [Lots of Advantages Callbacks etc].
- * "Template" is same as "Session" with Extra Advantages...

spring-config.xml:

```
<bean id="StudentDao" class="edu.dao.StudentDaoImpl">  
    <property name="sessionFactory">  
        <ref bean="factory"/>  
    </property>  
</bean>
```

EmployeeDao.java:

```
public interface EmployeeDao  
{  
    public void insertEmployee(Employee employee);  
}
```

EmployeeDaoImpl.java:

```
public class EmployeeDaoImpl extends HibernateDaoSupport  
    implements EmployeeDao  
{  
    public void insertEmployee(Employee employee)  
    {  
        getHibernateTemplate().save(employee);  
    }  
}
```

SessionFactoryTest.java:

```
public class SessionFactoryTest  
{  
    public void main(String[] args)  
    {  
        Student student = new Student();  
        student.setStudentName("N@IT");  
  
        Employee employee = new Employee();  
        employee.setEmployeeName("N@IT");  
  
        StudentDao studentDao = DaoFactory.getStudentDao();  
        studentDao.insertStudent(student);  
  
        EmployeeDao employeeDao = DaoFactory.getEmployeeDao();  
        employeeDao.insertEmployee(employee);  
  
        System.out.println("SUCCESS.");  
    }  
}
```

Employee.java:

```
public class Employee implements Serializable  
{  
    private Long employeeId = null;  
    private String employeeName = null;  
    getters & setters;  
}
```

Student.java:

```
public class Student implements Serializable  
{  
    private Long studentId = null;  
    private String studentName = null;  
    getters & setters;  
}
```

DaoFactory.java:

```
public class DaoFactory  
{  
    private static ApplicationContext context = new
```

```

private static String STUDENT_DAO = "studentDao";
private static String EMPLOYEE_DAO = "employeeDao";
public static StudentDao getStudentDao()
{
    return (StudentDao)
        context.getBean(DaoFactory, STUDENT_DAO);
}
public static EmployeeDao getEmployeeDao()
{
    return (EmployeeDao)
        context.getBean(DaoFactory, EMPLOYEE_DAO);
}

```

3/12
Thursday

Note: To get the Dao object we are writing Factory class and If we inject Dao also to the Main program [Servlet / Action .. etc] we don't require DaoFactory.

DAO PACKAGE

Spring - JDBC:

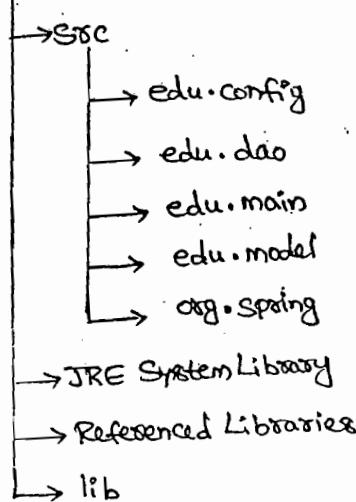
- * The DAO package provides a JDBC - Abstraction Layer that removes the need to tedious JDBC Coding and parsing of Database-Vendor specific error codes.
- * The JDBC package provides a way to do "programmatic" as well as "declarative transaction management"; Not only for classes implementing special interfaces, But for all your Pojos [plain old Java objects]
- * Don't require logic to create object for con, st, ps, rs and cs.
- * Don't require Exception Handling, Resource close [con/st/ps/cs /rs.close()] and programmatic Transaction Logic [declarative Transaction Means XML configuration]

3/2/09
Tuesday

* Implement a JDBC API Example, which should take care of

1. Creating Connection, Statement and ResultSet object
2. Resource closing [connection close ... etc.]
3. Converting SQLException to RuntimeException

SpringCoreJDBCTest



spring-config.xml :

```
<beans>  
    <!-- 1. Configure The DataSource object -->  
    <bean id = "jdbcTemplate" class = "org.springframework.jdbc.JdbcTemplate">  
        <property name = "dataSource">  
            <ref bean = "dataSource"/>  
        </property>  
    </bean>  
    <bean id = "studentDao" class = "edu.dao.StudentDao">  
        <property name = "jdbcTemplate">  
            <ref bean = "jdbcTemplate"/>  
        </property>  
    </bean>  
</beans>
```

```
package edu.dao;
public class StudentDao {
    private JdbcTemplate jdbcTemplate;
    public void setJdbcTemplate (JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
    public Student getStudent (String studentNo) throws SQLException {
        String query = "SELECT * FROM STUDENT
                        WHERE SNO = " + studentNo;
        ResultSet resultSet = jdbcTemplate.execute(query);
        Student student = null;
        if (resultSet.next()) {
            student = new Student();
            student.setStudentNo(resultSet.getString("SNO"));
            student.setStudentName(resultSet.getString("SNAME"));
        }
        return student;
    }
}
```

JdbcTemplate.java:

```
package org.springframework;
public class JdbcTemplate {
    private DataSource dataSource;
    public void setDataSource (DataSource dataSource) {
        this.dataSource = dataSource;
    }
    public ResultSet execute (String query) {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
    }
}
```

```

    try {
        connection = dataSource.getConnection();
        statement = connection.createStatement();
        resultSet = statement.executeQuery(query);
    } catch (SQLException e) {
        // Convert SQLException to RuntimeException.
    } finally {
        DbUtils.closeQuietly(null, null, null);
    }
    return resultSet;
}

```

Student.java :

```

package edu.model;
public class Student {
    private String studentId;
    private String studentName;
    getters and setters;
}

```

SpringJDBCCoreTest.java :

```

package edu.main;
class SpringJDBCCoreTest {
    private static ApplicationContext context = new
        ClassPathXmlApplicationContext("edu/config/spring-config.xml");
}

```

PSVM (String args[]) throws Exception

```

    StudentDao studentDao = (StudentDao)

```

```

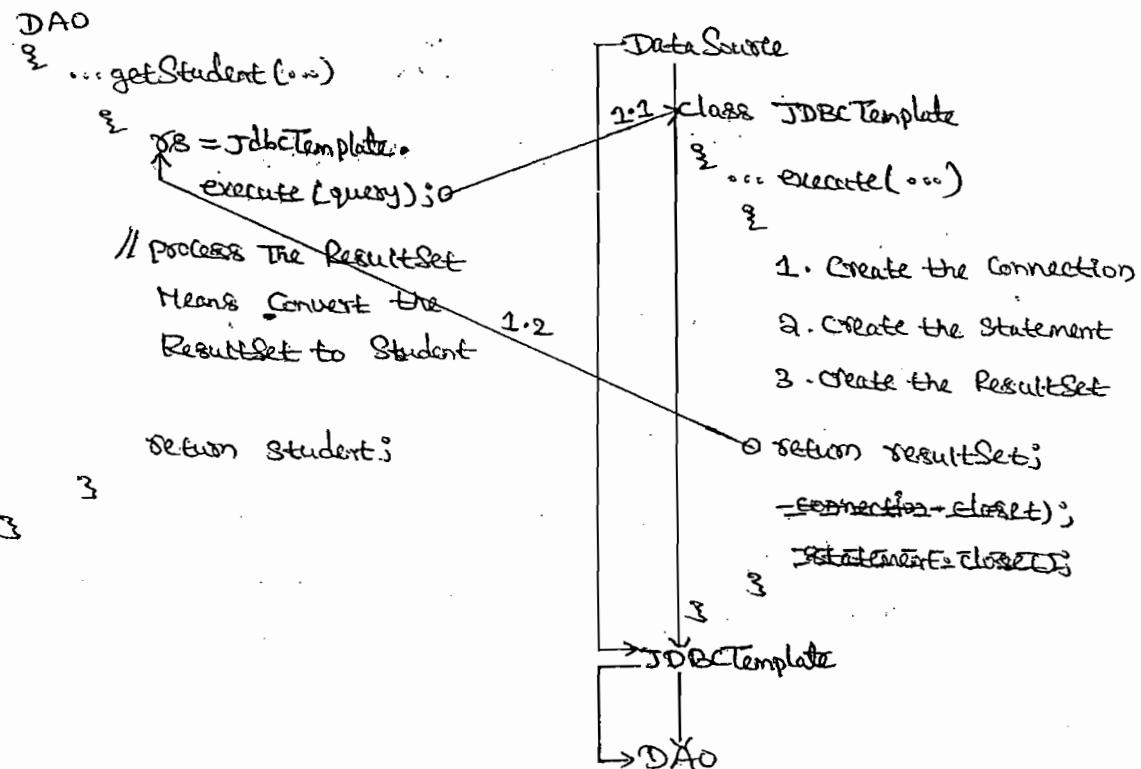
        context.getBean("studentDao");

```

```

    Student student = studentDao.getStudent("1");

```



41121
Frid,

Advantages In The Above Example:

- * No need to Create Connection, statement, ... etc.
- * Don't required Exception Handling

Dis Advantages:

- * API people should not close connection and statement - Because the ResultSet we are processing after finishing execute() method If they close it, we can't process the ResultSet.
- * At the Application Level, we can't close connection and statement Because we don't have those references. If we want, we can close the ResultSet but we should not close it. Because the closing logic is done by framework.

Note: In the above Example; It is one way communication means we are calling execute() method on API template class and it is returning ResultSet.

* To solve the issue, we need to implement "Two-Way communication" using "polymorphism concept". Here we need to call the API execute() method and it has to call the Developer call back method while passing ResultSet.

4/12/09
Friday

How The Container Will Call The Servlet?

StudentServlet studentServlet = Class.forName(StudentServlet)
• newInstance();
[Wrong]

Servlet servlet = Class.forName(StudentServlet) • newInstance();
[Correct]

Prepare HttpServletReq / Res
servlet.service(req, res);

Here request is pass to the Developer from Container in the same in Spring it has to pass ResultSet.

Steps To implement Spring - JDBC Program :

1. Take Any Java class and implement Contract/CallBack
[ResultSetExtractor]
2. In servlet we provide servlet name using web.xml
Here we need to provide the implemented class to JdbcTemplate.execute() method.
3. At the API side [In execute() method] they do the same servlet container process.

Note: For some of the Application we no need to implement the callBack, Spring API takes the callBack's.

Eg: jdbcTemplate.execute(Update query, parameters);

Developer

DAO :

```

    ... class StudentResultSetExtractor implement ResultSetExtractor
    {
        ... extractData(ResultSet rs)
        {
            do Iteration ...
        }
    }

    class Dao
    {
        getStudent(...)
        {
            // Don't iterate ResultSet
            JdbcTemplate.execute(query, StudentResultSetExtractor ref) B
        }
    }

```

API :

```

class JdbcTemplate
{
    ... execute(query, ResultSetExtractor rseRef) A a
    {
        // ref → It will be pointing to StudentResultSetExtractor,
        // EmployeeResultSetExtractor ... etc.

        // Based on what the Developer is passing
        // A a = pointing to B object

        1. get the connection using Datasource
        2. prepare statement using Query
        3. prepare ResultSet

        rseRef.extractData(rs); // a.x();
    }
}

```

Developer / Application

```
class SRSE implements RSE
{
```

```
    extractData(RS)
    {
        Iterate ...
    }
}
```

```
class DAO
```

```
{ daoMethod(...)
```

```
    jdbcTemplate.execute(query, o
        SRSE object);
```

```
}
```

1.1 : Developer is calling API execute() method while passing
Callback object.

2.1 : From API execute() method they will call Developer
callback method [extractData()]

2.2 : Controller will go back to API execute() method .

1.2 : Controller will given back to Developer after closing
all the connections.

Note : The Above Diagram is "Two-way communication" call
"Developer to API" and "API to Developer".

* To implement the Above program Contract is must
[Interface] or [callback]. Because they don't know
about the implementation. So they are Applying
polymorphism logic

API

```
class SRSE object
{
    execute(query, RSE)
    {
        connection ...
        statement ...
        resultSet ...
    }
}
```

```
2.2 → TSE : extractData(rs);
```

```
finally
{
    close ...();
}
```

```
1.2
```

5/12/09
Saturday

Internal Process for JdbcTemplate • query (query, new Object[] { studentNo },
Object array [Values],
ResultExtractor):

```
public class StudentDao extends JdbcDaoSupport
{
    public Student getStudent( String studentNo) throws SQLException
    {
        String query = "SELECT * FROM STUDENT WHERE SNO=?";
        StudentResultSetExtractor studentRSE = new
            StudentResultSetExtractor ();
        return ( Student) getJdbcTemplate () .query (query,
            new Object[] { studentNo }, studentRSE);
    }

    class StudentResultSetExtractor implements ResultSetExtractor
    {
        public Object extractData ( ResultSet resultSet ) throws SQLException
        {
            Student student = null;
            if ( resultSet .next ())
            {
                student = new Student ();
                student .setStudentNo (resultSet .getString ("SNO"));
                student .setStudentName (resultSet .getString ("SNAME"));
            }
            return student;
        }
    }
}
```

Note: Here we are not creating ResultSet object . it will be created by the Spring and it will be called internally from query ... methods while passing ResultSet to extractData()

* These Different interfaces will do a specific job

1. PreparedStatementSetter:

It is used to set the values to the PreparedStatement
[Arg PreparedStatementSetter [class]]

2. PreparedStatementCreator :

It is used to create PreparedStatement object

[SimplePreparedStatementCreator [class]]

3. PreparedStatementCallback :

It is used to call,

1. Set the values to PreparedStatementSetter ... Impl class
2. Execute the PreparedStatement
3. Call the extractData() while passing ResultSet

[Here Developer is calling extractData() will be called]

4. public Object execute (PreparedStatementCreator psc ,

PreparedStatementCallback action) :

1. Get the Connection object
2. Call PreparedStatementCreator ... Impl class and Create the PreparedStatement.
3. Call PreparedStatementCallback

Developer :

Developer Flow :

1.1.1

```
getJdbcTemplate().query(query, new Object[] { studentNo3,  
                                              studentRSE});
```

API :

1.1.2

```
public Object query ( String sql, Object [] args,  
                      ResultSetExtractor rse)
```

1.2.1

```
query(sql, new ArgPreparedStatementSetter(args), rse);
```

1.2.2

```
public Object query ( String sql, PreparedStatementSetter pss,  
                      ResultSetExtractor rse)
```

query (new SimplePreparedStatementCreator(CSvl, PSS, RSE));

1.3.2

```
public Object query ( PreparedStatementCreator psc,
final PreparedStatementSetter pss,
final ResultSetExtractor rse )
```

1.4.1

```
execute ( psc, new PreparedStatementCallback () { } );
```

1.4.2

```
public Object execute ( PreparedStatementCreator psc,
PreparedStatementCallback action )
```

Note:

1.1.1 Pass The Values — call method One();

1.1.2 Argument [Hold The Value] — methodOne (int i);

Sample Code And Flow:

Parameter X(2);	Argument X (int i) Holding the Value
Passing the Value	

1. Class StudentResultSetExtractor implements ResultSetExtractor

{ B. IV.2

```
public Object extractData ( ResultSet resultSet )
```

{ while (resultSet.next())

{ RS → student

}

2. class ArgPreparedStatementSetter

{ private final Object [] args;

```
public ArgPreparedStatementSetter ( Object [] args )
```

{ this.args = args; }

{

B. III.2

```
public void setValues ( PreparedStatement ps )
```

{ ps.setString (....); }

{}

3. private static class SimplePreparedStatementCreator

{
 private final String sql;

 public SimplePreparedStatementCreator (String sql)

 {
 this.sql = sql;

}

B.I.2

 public PreparedStatement createPreparedStatement (Connection con)

 {
 return con.prepareStatement (this.sql);

}

}

4. PSS → PreparedStatementSetter

RSE → ResultSetExtractor

(new PreparedStatementCallback ()

{
 public Object doInPreparedStatement (PreparedStatement ps)

 {
 ResultSet rs = null;

 try

 {
 PSS.setValues (ps);

 rs = ps.executeQuery ();

 RSE.extractData (rsToUse);

 }

 finally

 {

 TableUtils.closeResultSet (rs);

 PSS.cleanupParameters ();

 }

}
})

5. public Object execute (PreparedStatementCreator psc,

 PreparedStatementCallback action)

Flow :

F.I.1

1. Student student = (Student) getJdbcTemplate().query(query,
new Object[]{studentNo3}, studentRSE);

F.II.1

2. F.II.1

```
public Object query(String sql, Object[] args, ResultSetExtractor rse)
{
    return query(sql, new ArgPreparedStatementSetter(args), rse);
}
```

3. F.II.2

3. F.III.1

```
public Object query(String sql, PreparedStatementSetter pss,
                    ResultSetExtractor rse)
```

4.

```
return query(new SimplePreparedStatementCreator(sql), pss, rse);
```

3. F.III.2

4.

F.IV.1

```
public Object query(PreparedStatementCreator psc,
                     final PreparedStatementSetter pss,
                     final ResultSetExtractor rse)
```

5.

```
return execute(psc, new PreparedStatementCallback()
```

6.

B.II.2

```
public Object doInPreparedStatement(PreparedStatement ps)
```

7.

```
ResultSet rs = null;
```

try

8.

```
pss.setValues(ps); B.III.1
```

```
rs = ps.executeQuery();
```

```
rse.extractData(rsToUse); B.IV.1
```

***** Here it will call DeveloperImpl class method (SPSE)

8/12
Hm

finally
 $\{\}$
 $\text{jdbcUtil.closeResultSet(rs);}$

$\text{psc.cleanupParameters();}$

$\}$

$\}$

F. IV.2

$\}$

5.

F. V.1

public Object execute(PreparedStatementCreator psc,
 PreparedStatementCallback action)

Connection con = dataSource.getConnection(); Internally it uses
[util class]

psc.createPreparedStatement(conToUse); B.I.1

Object result = action.doInPreparedStatement(pscToUse); B.II.1

Closing con... etc.

Final Return ... F.V.2

8/12/09
Monday

Note : In SimplePreparedStatementCreator while creating the preparedStatement

It is using sql. But we are not passing any query while calling
createPreparedStatement(). Because while creating PreparedStatementCreator
object it is initializing the query with the object using
constructor and the same logic is applied to ArgPreparedStatement
Setter() etc.

... execute()

$\{\}$ psc.createPreparedStatement(conToUse);

$\}$

public PreparedStatement createPreparedStatement(Connection con)

$\{\}$ // Already sql is initialized and here it is using

return con.prepareStatement(this.sql);

* Here using constructor, it is initializing the query

```
query(new SimplePreparedStatementCreator(Sql, PSS, RSE));
```

ORM PACKAGE

- * The ORM package provides integration layers for popular object-relational mapping API's, Including JPA, JDO, Hibernate and iBatis.
- * Using the ORM package , you can use all those object-relational mappers in combination with all the other features Spring offers , such as the Simple Declarative Transaction Management features mentioned previously.
- * ORM technologies will reduce the code to persist an object in the DataBase and to retrieve the Data from the Database .
- * But we need to write the logic for ExceptionHandling , Transaction Management , ... etc.
- * If we integrate ORM with Spring , we can reduce Exception Handling ... etc logic
- * Using Spring we can Integrate with the Existing ORMs which are available in the market . But Spring Framework is not offering its own ORM .

MVC PACKAGE

1. In Struts Application we should not pass the form object to the DAO Layer . Because Studentform is API dependent class .
- * In Struts 2.0 it is not API Dependent class .

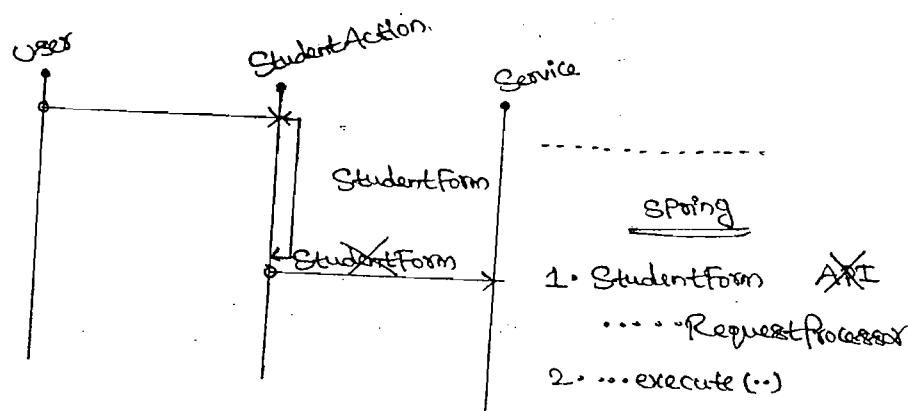
* In Spring Application, we call form object as command object and it is not API dependent. So, we can pass those objects to DAO Layer.

2. In Struts 1.2, we can do pre-process using Request Processor, and Post-process is not possible.

* In struts 2.0 it is possible.

* In Spring, we call Pre-Process as PreHandling, we can do PreHandling and PostHandling.

* SpringMVC is more powerful than other MVC's.



* Initially Spring has 7 modules, Now Spring Web and Spring MVC both are with Spring Web Module.

* Spring is having its own MVC and using Spring, we can integrate other MVC's. Because Struts 1.2 does not provide Transaction Management...etc. and Injection Concept.

So, if we integrate with Spring, Struts can do MVC job and Spring will do Injection Concept, Transaction management. etc

* SpringMVC package provides a Model-View-Controller (MVC) implementation for Web Applications.

* Spring's MVC framework is not just any old implementation. It provides a clean separation between domain model code and Web forms, and allows you to use all the other int

- * If we are working RMI, we need to Create Stub, Skeletons etc. If we are working with Spring and RMI we don't require Stub & Skeletons.
- * We can Integrate Spring with EJB. So, we can reduce lookup code and we can integrate with MessageDrivenBeans and all. Then all the connection ... etc code will be reduced.

SPRING AOP

- * In Spring it reduced Exception Handling and Transaction logic ... etc. Internally using AOP concepts.

SPRING CORE EXAMPLES

Working with Collections:

- * Overdefined Collection configuration is not used most of the cases in the project.
- * Existing classes method arguments might be collection reference means we need to inject collection object to an Existing class. So those cases we work with Collections concept in the projects.

LocalSessionFactoryBean:

```
public void setMappingResources(Spring[] mappingResources)
{ }
```

```
public class TransactionProxyFactoryBean
```

```
public void setTransactionAttributes(Properties transactionAttributes)
{ }
```

Eg:

```
package edu.model;
public class Student
{
    private List studentList;
    private String[] stringArray;
    public void setStudentList (List studentList)
    {
        this.studentList = studentList;
    }
    public void setStringArray (String[] stringArray)
    {
        this.stringArray = stringArray;
    }
}
```

setStudentList — Removes set and S should be small

List / String[] (Arg Type) — use List tag

```
<bean id = "student" class = "edu.model.Student">
```

```
    <property name = "studentList">
```

```
        <list>
```

```
            <value> N@It1 </value>
```

```
            <value> N@It2 </value>
```

```
        </list>
```

```
    </property>
```

```
</bean>
```

Spring Container Internal Process:

1. Create the Student Object

```
Class.forName (Student);
```

2. Create List Object and add the Objects.

```
List studentList = new ArrayList();
```

```
studentList.add ("N@It1");
```

```
studentList.add ("N@It2");
```

3. Call `setStudentList()` method while passing `studentList` object as argument.

9/19/10
Wednesday

```
studentRef.setStudentList(studentList);
```

Now, The `Student` object is ready with `studentList` object.

Note: The internal process logic is same to all other collection Examples.

Set (or) Working with Set:

- * If the setter method argument type is `Set`, To Inject the value use `<set>` tag and the argument type is array also we can inject the value use `<list>` tag (or) `<set>` tag.

```
public class Student {
    private Set studentSet;
    private String[] stringArray;
    public void setStudentSet( Set studentSet ) {
        this.studentList = studentList;
    }
    public void setStringArray( String[] stringArray ) {
        this.stringArray = stringArray;
    }
}
```

```
<bean id="student" class="edu.model.Student">
    <property name="studentSet">
        <set>
            <value> N@IT</value>
            <value> N@IT2 </value>
        </set>
    </property>
</bean>
```

lect

9/12/09 Properties:
Wednesday

- * If we want to Inject properties object then we use <prop> tag and the properties key, value pairs we define using

<prop> tag

public class Student

{
 private Properties properties;

 public void setProperties (Properties properties)

 {
 this.properties = properties;
 }

}

<Property name = "properties">

<prop>

<prop key = "studentNo">1</prop>

<prop key = "studentName">N@IT</prop>

</prop>

</property>

Working with Map:

- * If we want to Inject a Map object , then we use a <map> tag and the key, value we specify using <entry> tag

public class Student

{
 private Map studentMap;

 public void setStudentMap (Map studentMap)

 {
 this.studentMap = studentMap;
 }

}

<bean id = "studentNameStr" class = "java.lang.String">

<constructor-arg>

<value> N@IT </value>

</constructor-arg>

</bean>

```

<bean id = "student" class = "edu.model.Student">
  <property name = "StudentMap">
    <map>
      <entry key = "studentNo">
        <value>1</value>
      </entry>
      <entry key = "studentName">
        <ref bean = "studentNameStr"/>
      </entry>
    </map>
  </property>
</bean>

```

BeanLifeCycle :

- * Any Java class we can call it as a SpringBean . If the class is implementing the Lifecycle methods related interface, those methods will be called automatically, while creating the object by the Spring container . Some of the Lifecycle method interfaces.

Some of the Lifecycle Method Interfaces:

1. BeanNameAware	—	setBeanName
2. BeanFactoryAware	—	setBeanFactory
3. ApplicationContextAware	—	setApplicationContext
4. InitializingBean	—	afterPropertiesSet 
	[init	configure in XML file]
5. DisposableBean	—	destroy 

Note: In projects , most of the cases we don't implement these interfaces for the Java classes . But these are important to understand internal concepts .

- * If we are working with BeanFactory then setApplicationContext
Lifecycle method will not be called.

Note: The interfaces are just like Servlet interfaces and the methods are just like service method. Here most of the lifecycle methods will be called while creating object by the container.

- * Only destroy() method is called later.
- * We can configure init() method and destroy() methods in the spring-config.xml file.

init-method = "init" destroy-method = "destroy"

10/12/09
Thursday

Inheritance Concepts:

- * In Core Java, we cannot declare abstract keyword as dynamic.
In Spring, we can declare abstraction as dynamic.

public class Student {} // Non-Abstract

public abstract class Student {} // Abstract

- * Here if we want to change Abstract to Non-Abstract and Non-Abstract to Abstract, The changes needs to be done in the Java code.

If it is Spring, we can change Abstract to Non-Abstract using xml declaration.

```
<bean id = "Student" class = "edu.model.Student"  
abstract = "true" /> // Abstract
```

```
<bean id = "Student" class = "edu.model.Student" /> // Non-Abstract  
(OR)
```

```
<bean id = "Student" class = "edu.model.Student"  
abstract = "false" />
```

- * Default value for abstract attribute is false. We can get one bean definition injected values to another bean definition.
[We call the concept inheritance]

- * If we want to get the Datasource object for Two Different

In this case, we override multiple times in one line,
the definitions.

- * Instead of repeating 2 times, If we Apply Inheritance concept we can reduce the code

```
<bean id = "scottDataSource" class = "DriverManagerDataSource">  
<value>driverClassName : oracle.jdbc.driver.OracleDriver </value>  
<value> url : jdbc:oracle:@localhost:1521:SERVER </value>  
<value>username : scott </value>  
<value>password : tiger </value>  
</bean>  
  
<bean id = "systemDataSource" class = "DriverManagerDataSource">  
<value>driverClassName : oracle.jdbc.driver.OracleDriver </value>  
<value> url : jdbc:oracle:thin:@localhost:1521:SERVER </value>  
</bean>  
  
<bean id = "systemDataSource" class = "DriverManagerDataSource">  
<value>driverClassName : oracle.jdbc.driver.OracleDriver </value>  
<value> url : jdbc:oracle:thin:@localhost:1521:SERVER </value>  
<value>username : system </value>  
<value>password : manager </value>  
</bean>
```

- * In the above Example, driverClassName and URL is repeated

```
<bean id = "dataSource" class = "DriverManagerDataSource"  
abstract = "true">  
  
<value>driverClassName : oracle.jdbc.driver.OracleDriver </value>  
<value> url : jdbc:oracle:thin:@localhost:1521:SERVER </value>  
</bean>  
  
<bean id = "systemDataSource" class = "DriverManagerDataSource"  
parent = "dataSource">  
username : system  
password : manager  
</bean>
```

```
<bean id="scottDataSource" class="DriverManagerDataSource">
    parent="dataSource"
    username="scott"
    password="tiger"
```

</bean>

- * Using parent attribute we can get the other bean definition values.

(A)

Autowiring

- * There are 4 ways in Autowiring.

1. Constructor
2. Setter [ByName]
3. ByType
4. AutoDetect

- * Autowiring Means Injection will be done Automatically,
- * Autowiring is not suggestible in the projects because we get the ambiguity problem.
- * Ambiguity Means if we don't required the Injection also it might be doing Injection.
- * There are some projects which will be using Autowiring. If there are very less dependencies, we can Apply Autowiring in the projects.

Constructor:

```
public class Service {
    private Dao dao;
    public Service(Dao dao) {
        this.dao = dao;
        System.out.println("Service.Service(Dao)");
    }
}
```

- * The above class is expecting Dao object using Constructor Injection. So the Spring Configuration file should be.

```

<bean id = "service" class = "edu.service.Service">
    <constructor-arg>
        <ref bean = "dao"/>
    </constructor-arg>
</bean>

```

- * Autowire constructor Means Instead of we injecting Dao object using <constructor> tag, If we define autowire = "constructor" Then it will inject all the dependent constructor injection objects automatically

- * So, The above code should be replaced as

```

<bean id = "service" class = "edu.service.Service"
      autowire = "constructor">

```

Setter [ByName] :

```

public class Service
{
    private Dao dao;
    public void setDao(Dao dao)
    {
        this.dao = dao;
        System.out.println("Service.setDao(Dao)");
    }
}

```

- * The Above class is dependent on Dao object using Setter Injection
 So, The Spring configuration file will be

```

<bean id = "service" class = "edu.service.Service">
    <property name = "dao">
        <ref bean = "dao"/>
    </property>
</bean>

```

- * Instead of we Injecting the object, If we apply autowire = "byName" then the dependent objects automatically will be injected using Setter Injection.

* Here, The Above code is replaced as.

```
<bean id = "service" class = "edu.service.Service"
      autowire = "byName" >
```

* Here, It will check the Setter Method for the defined property in the class. But it doesn't check for other setter methods.

[Property is dao, so it will check only setDao(Dao). It doesn't check for other setter methods like setDaoOne(Dao) ... etc]

ByType :

```
public class Service
{
    private Dao dao;
    public void setDao(Dao dao)
    {
        this.dao = dao;
        super("Service.setDao(Dao)");
    }
    public void setDaoOne(Dao dao)
    {
        this.dao = dao;
        super("Service.setDaoOne(Dao)");
    }
}
```

* The Above Service class is expecting Dao object using Setter Injection and If we Apply ByName only setDao() method will be called ByType it will Apply All the setter() methods with ByType and If we Apply the property type and If we Apply ByType both the setter() methods will be called

* Both the setter Methods are taking the same argument type. But In general it will be with different arguments.

Here, It checks for Any setter methods with Argument type.

```
<bean id = "service" class = "edu.service.Service"
      autowire = "byType" >
```

AutoDetect Using Constructor:

* AutoDetect means Constructor (By Type) will be Applied.

If there is no Default Constructor then it will Apply Constructor Injection.

```

public class Service
{
    private Dao dao;
    public Service(Dao dao)
    {
        this.dao = dao;
    }
    public void setDaoOne(Dao dao)
    {
        this.dao = dao;
    }
}

```

* The Above Service class is not having Default constructor.
So, It will Apply constructor.

<bean id="Service" class="edu.service.Services"
autowire="autodetect" >

```

public class Service
{
    private Dao dao;
    public Service()
    {
        this.dao = dao;
    }
    public void setDaoOne(Dao dao)
    {
        this.dao = dao;
    }
}

```

(B)
12
of

* The Service class is having Default constructor. So, It will Apply by Type.

<bean id="Service" class="edu.service.Services"
autowire="autodetect" >

Default Autowiring :

* If you want to Apply Autowire to All the Beans, we can define to <beans> tag. Then it will Apply the Default Autowire to All the Beans. If we are not specifying Explicitly Autowire

<beans default-autowire = "constructor">

<bean id = "serviceOne" class = "edu.service.ServiceOne"
autowire = "byName"/>

[Here It will Apply byName [setter]]

<bean id = "serviceTwo" class = "edu.service.ServiceTwo"/>

[Here we are not defining means constructor] Autowire So, It takes the default

Note: If we want to shutdown the Spring Container Call the close() method on AbstractApplicationContext then it will Destroy all the Beans. Then we can see destroy() method logic in the Console. Now if we call the close() method It will call the lifecycle destroy() method.

AbstractApplicationContext context = new

ClassPathXmlApplicationContext ("edu/config/spring-config.xml")

context.close();

I18N [Internationalization] Concepts :

* Internationalization means Based on the language we select It gives the corresponding messages [LabelNames, ButtonNames..etc]

Steps To Configure I18N :

1. Write N No. of properties files for N No. of languages
 Syntax: <property fileName = <en>.properties

Eg: tables_en.properties

(B)
12/12/09

saturday

Note. To Locality in → tools → Internet Options → Languages
→ Add Then Select the Language

Properties file contains key = Value, Value is the language specific labels (or) button names.

2. Inject the properties files to ResourceBundleMessageSource using setBasenames() or setBasename(). It is one file use basename()
If there are multiple files use Basenames().

```
class ResourceBundleMessageSource {  
    public void setBasename( String basename ) {  
        setBasenames( new String[] { basename } );  
    }  
    public void setBasenames( String[] basenames ) {  
    }  
}
```

```
<bean id = "messageSource" class = "... ResourceBundleMessageSource" >  
<property name = "basenames" >  
    <list>  
        <value> letters </value>  
    </list>  
</property>  
  
<!-- <property name = "basename" >  
    <value> letters </value>  
</property> -->
```

3.

- 3.1. Create The Locale object.

```
Locale czech = new Locale( "cz", "cz" );  
(OR)
```

```
Locale english = Locale.English;
```

```

public final class Locale
{
    static public final Locale ENGLISH = createSingleton("en_4",
                                                       "en", "4");
    private static Locale createSingleton( String key, String language,
                                         String country)
    {
        Locale locale = new Locale(language, country, false);
        cache.put(key, locale);
        return locale;
    }
}

```

Note: We can create the Locale object using new keyword (or) using constant. If we are creating the object using constant it will call createSingleton() method and creates the object using new and it will be cached.

3.2. For the Language Specific message call getMessage() method on context object

```

context.getMessage("key", "studentName");

```

Values To The Message
new Object[]{"N@It1", "N@It2"},
Locale
english);

Note: Locale → language + Locale.

(c). Static factory:

```

public class Student
{
    public static String getStudentName()
    {
        return "N@It";
    }
}

```

* In corejava, If we required getStudentName() method value

How can we do it using Spring?

```
<bean id="staticFactory" class="edu.model.Student"
      factory-method="getStudentName"/>
```

```
String studentName = (String) context.getBean("staticFactory");
```

Note: If we are not giving factory-method then it returns student object and If we are giving factory-method then it returns getStudentName() → returned object.

(ii). Non-StaticFactory:

```
public class Student
{
    public String getStudentName()
    {
        return "N@IT";
    }
}
```

core Java: if we required getStudentName() method then

```
Student student = new Student();
```

```
String studentName = student.getStudentName();
```

How can we do it using Spring?

```
<bean id="student" class="edu.model.Student" /> //object
```

```
<bean id="nonStaticFactory" factory-bean="student"
      factory-method="getStudentName"/>
```

```
String studentName = (String) context.getBean("nonStaticFactory");
```

Note: If we are not giving factory-method and factory-bean then it returns student object and If we are giving factory-method and factory-bean then it returns getStudentName() → returned object.

* Here getStudentName() is Non-Static method. So we are returning the corresponding Object using factory-bean.

factory-bean : To refer the Object (Bean)

factory-method : To refer the Method.

(E) Method Injection:

```
public class EmployeeDaoImpl  
{  
    public class StudentDaoImpl  
    {  
        public class Service  
        {  
            public EmployeeDaoImpl getDao()  
            {  
                return employeeDaoImpl Obj;  
            }  
        }  
    }  
}
```

* Service class getDao() method is returning employeeDao object and It is a static approach. But our requirement is sometimes it has to return studentDao and sometimes it has to return employeeDao. Means we required the method return object in Dynamic Approach.

* Let's declare the method as Abstract and configure what object it has to return in the Spring configuration file.

```
public abstract class Service  
{  
    public abstract Dao getDao();  
}
```

Spring-config.xml:

```
<bean id = "dao" class = "edu.dao.EmployeeDaoImpl" />  
<bean id = "service" class = "edu.service.Service">  
    <lookup-method name = "getDao" bean = "dao" />  
</bean>
```

```
<bean id = "dao" class = "edu.dao.StudentDaoImpl" />  
<bean id = "service" class = "edu.service.Service">  
    <lookup-method name = "getDao" bean = "dao" />  
</bean>
```

* What object the getDao() has to return, we are injecting using lookup-method, Here we are injecting the method return type object

(E) Property Namespace

```
public class Dao  
{  
    public class daoMethod()  
    {  
        SQL("DAO.daoMethod()");  
    }  
}  
  
public class Service  
{  
    public Dao dao;  
    public void setDao(Dao dao)  
    {  
        this.dao = dao;  
    }  
}
```

* Connection
* close
* Transaction
* exception

- * Service is dependent on Dao object - So, if we want to inject Dao to Service. The Spring Configuration file should be

```
<bean id = "dao" class = "edu.dao.Dao"/>  
<bean id = "service" class = "edu.service.Service">  
    <property name = "dao">  
        <ref bean = "dao"/>  
    </property>  
</bean>
```

- In Spring 2.5
- * Instead of writing Property name :: ref :: etc. Using schemaBased Approach the code should be

```
<bean id = "dao" class = "edu.dao.Dao"/>  
<bean id = "service" class = "edu.service.Service"  
    p:dao-ref = "dao"/>
```

P : Property
dao : Property Name
ref : Reference
dao : Another Bean (or) To be Injected Bean

14/12/09
Monday

Spring Dao

Execute The Below Query Using Spring JDBC:

UPDATE STUDENT SET SNAME=? WHERE SNO=?

```
getJdbcTemplate().update("UPDATE STUDENT SET SNAME=?  
WHERE SNO=?", new Object[] {"N@ITUpdate", 1});
```

* JdbcTemplate class execute() method will catch SQLException and convert to DataAccessException and DataAccessException is a RuntimeException. So we no need to handle it.

* In projects using AOP concepts, we can Rethrow SQLException to userdefined exception

```
... execute(...)  
{  
    try  
    {  
        ...  
        catch (SQLException ex)  
        {  
            throw getExceptionTranslator().translate("...", sql, ex);  
        }  
    }  
  
    public abstract DataAccessException translate(String s, String s1,  
                                                SQLException sqlexception);  
  
    public abstract DataAccessException extends NestedRuntimeException  
    {  
    }
```

PreparedStatementCreator :

Steps To implement PreparedStatementCreator :

1. Take Any class and implement PreparedStatementCreator

Private class StudentPreparedStatementCreator implements

```
    PreparedStatementCreator  
    {  
        public PreparedStatement createPreparedStatement (Connection connection)  
        throws SQLException  
        {  
            PreparedStatement preparedStatement = connection.  
                prepareStatement("studentQuery.toString());  
            preparedStatement.setString(1, student.getStudentName());  
            .....  
        }  
    }
```

- * Pass the implemented object to JdbcTemplate. Implemented method is called by Framework.

```
int noOfRecordUpdate = getJdbcTemplate().update("new  
studentPreparedStatementCreator");
```

Internal Process:

```
public class JdbcTemplate  
{  
    ... execute(PreparedStatementCreator psc)  
    {  
        // psc points to StudentPreparedStatementCreator object.  
        psc.createPreparedStatement(conn);  
        // Here Developer implemented method will be called.  
    }  
}
```

- * PreparedStatementCreator is an interface and it is used to create the PreparedStatement and hence we need to implement this interface and has to pass the implemented object to JdbcTemplate.update() method. Then our implemented method createPreparedStatement will be called by template class execute() method while passing connection object.

(*) Note: This interface never be implemented in the project and it is for Spring Framework internally used. The internal class for this interface is SimplePreparedStatementCreator.

PreparedStatementSetter:

- * It is used to set the values to the preparedStatement and here we need to override setValues() method while taking preparedStatement Argument. Here the setValues() method is called by the framework while passing preparedStatement.

(*) Note: It never be used in the project and It is for Spring internally use implemented class for this interface is

ArgPreparedStatementSetter implements PreparedStatementSetter

- * Here Spring Framework is responsible to create the preparedStatement and we are responsible to set the values.

Steps To implemented :

1. Take Any class and implement `PreparedStatementSetter`

private class `StudentPreparedStatementSetter` implements

`PreparedStatementSetter`

 public void `setValues(PreparedStatement ps)`

 ps.setString(1, student.getStudentName());

2. Pass the implemented object to `update()` method. Then the implemented method is call by Framework.

int noOfRecordUpdate = getJdbcTemplate().update(new

`StudentPreparedStatementSetter`);

Internal Process

public class `JdbcTemplate`

 ... update(... final `PreparedStatementSetter` pss)

 // Pass points to `StudentPreparedStatementSetter` object.

Create the Connection object and Create the PreparedStatement object using `SimplePreparedStatementCreator`.

pss.setValues(ps);

// Here Developed implemented methods will be called.

* To process the ResultSet, we have `ResultSetExtractor`, `RowCallbackHandler`, `RowMapper` And `ParameterizedRowMapper`.

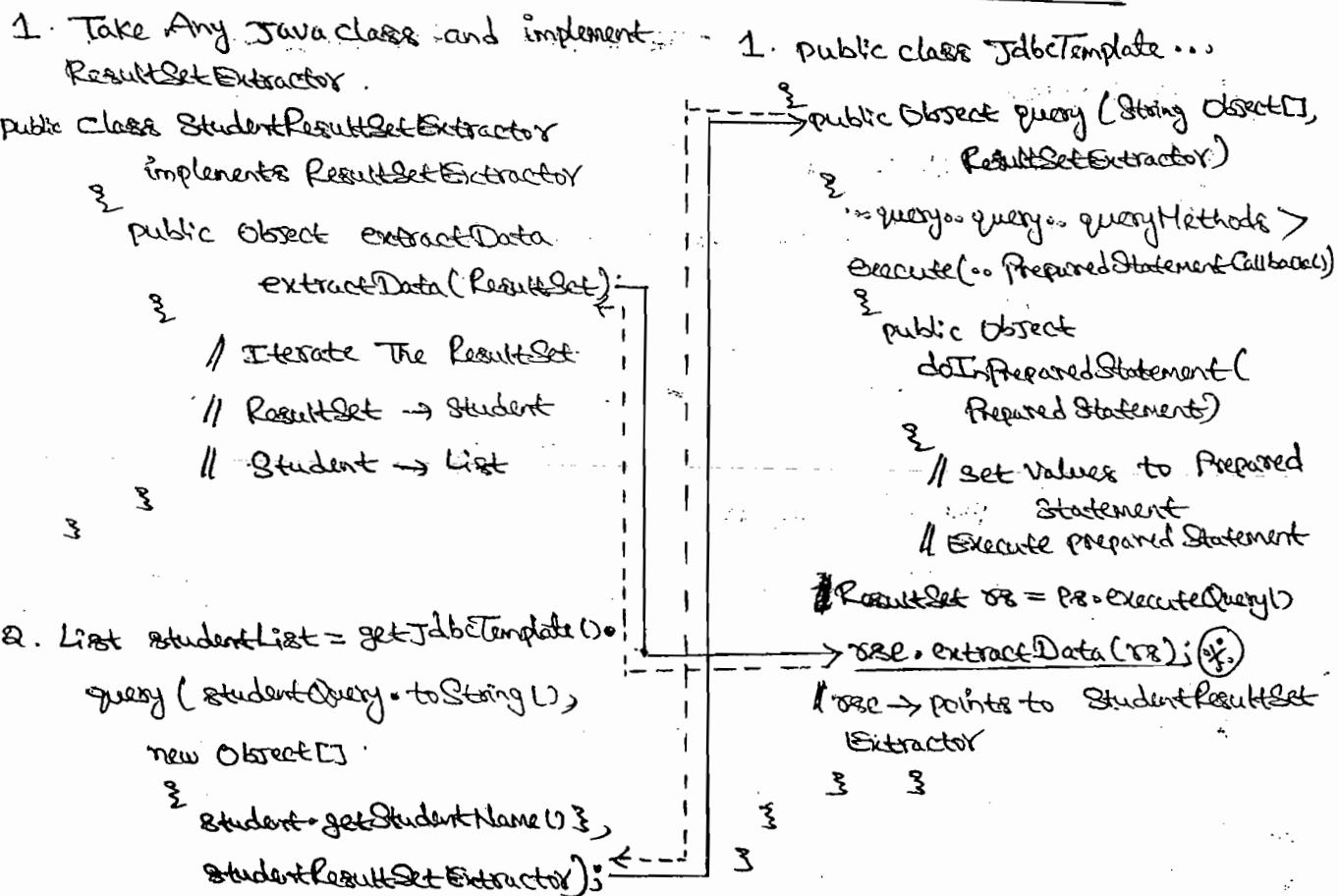
Note: In "RowMapper" and "ParameterizedRowMapper" will be used in the projects and some projects might be implementing ResultSetExtractor. But always prefer "RowMapper & ParameterizedRowMapper".

5/12/07
Tuesday Steps To Implement :

1. Take Any Java class and implement ResultSetExtractor and override extractData()
2. Pass the ResultSetExtractor object, Query and the values to JdbcTemplate query() method.
3. ResultSetExtractor is an interface and here we implement it while overriding extractData() method and it takes the ResultSet argument.

Application

Framework



Process Steps

1. We need to Iterate the Resultset (while...)
2. We need to Add Values to student object (Resultset → Student)
3. We need to Add Student object to List object and then finally we get the List of Student objects.

Note: StudentResultSetExtractor • extractData(ResultSet) method is called by Framework while passing ResultSet.

RowCallbackHandler:

Application

1. Take Any Java class and implement RowCallbackHandler

```
class StudentRowCallbackHandler  
implements RowCallbackHandler  
{  
    public void processRow(ResultSet)  
    {  
        //Iterate the ResultSet is  
        //Not Applicable  
        //ResultSet → Student  
        //Student → List  
    }  
}
```

2. List studentList = getJdbcTemplate()
• query(StudentQuery.toString(),
new Object[] {
 student.getName(),
 studentRowCallbackHandler});

Framework

1. Public class JdbcTemplate...

```
public Object query(String,  
Object[], RowCallbackHandler rch)  
{  
    ...query(RowCallbackHandler  
    ResultSetExtractor(rch))...  
    ...query(... every ... queryMethods ...)...  
    execute(... PreparedStatementCallback)  
}  
public Object  
doInPreparedStatement(  
    PreparedStatement)  
{  
    //Set Values to prepared  
    //Statement.  
    //Execute prepared Statement  
    ResultSet resultSet = ps.  
    executeQuery()  
    ...  
    use: extractData(resultSet);  
    use → points to RowCallbackHandler  
    ResultSetExtractor  
}  
}
```

```
class RowCallbackHandler implements ResultSetExtractor  
{  
    public Object extractData(  
        ResultSet resultSet)  
    {  
        while (resultSet.next())  
        {  
            this.rch.processRow(resultSet);  
        }  
    }  
}  
//rch → points to Student PreparedStatement  
//Callback Handler
```

10
Wednesday

1. We no need to iterate the Resultset (while...) [Not Applicable]

Because Iteration logic is there in RowCallbackHandlerResultSetExtractor
• extractData()

2. We need to Add results to Student object (Resultset → Student)

3. We need to Add Student object to List object and then finally
we get the List of Student objects.

(Ex) Note: RowCallbackHandlerResultSetExtractor.extractData(Resultset) method
is calling StudentRowCallbackHandler processRow() while passing Resultset
and it will be in iteration.

* RowCallbackHandler is an interface and the implemented class has to
override processRow() method and it takes Resultset and it is used
to process the Resultset.

RowMapper:

1. Take Any Java class and implement RowMapper

class StudentRowMapper implements RowMapper

{ public void mapRow(ResultSet, int)

{ //Iterate the Resultset → "Not Applicable"

// ResultSet → Student

// Student → "Not Applicable"

3 3

2. List studentList = getJdbcTemplate().query (StudentQuery.toString(),
new Object[] { student.getStudentName(),
studentRowMapper });

Internal Process:

public class JdbcTemplate ...

{ public Object query (String Object[], RowMapper rowMapper)

{ ...query (RowMapperResultSetExtractor (RowMapper))

...query...query ... queryMethods... >

```

execute(... PreparedStatementCallback())
{
    public Object doInPreparedStatement(PreparedStatement)
    {
        // SetValues to Prepared Statement ...
        // Execute Prepared Statement
        // ResultSet rs = ps.executeQuery()
        // rs.extractData(rs);
        // rs - Points to RowMapperResultSetExtractor
    }
}

```

class RowMapperResultSetExtractor

```
public Object extractData(ResultSet rs)
```

```
{
    List results = ...;
    int rowNum = 0;
    while (rs.next())
    {
        - Iteration

```

(%) results.add(this.rowMapper.mapRow(rs, rowNum++));

// It is Adding to List. So, we don't require Iteration
and List.

// rowMapper → pointer to StudentRowMapper

```

    return results;
}
```

Steps To Process:

1. We no need to iterate the ResultSet (while...) [Not Applicable]
Because Iteration Logic is there in RowMapperResultSetExtractor.extractData()

2. We need to Add Results to Student Object (ResultSet → Student)

3. We no need to Add Student Object to List Object

Note: RowMapperResultSetExtractor.extractData(ResultSet) method is calling
StudentRowMapper mapRow while passing ResultSet and it will be in Iteration

override mapRow() method while taking ResultSet and RowMapper & It is also used to process the ResultSet

- * RowCallbackHandler & RowMapper Methods will be called by ResultSetExtractor implemented class [RowCallbackHandlerResultSetExtractor & RowMapperResultSetExtractor]

(2) QueryForMethods:

(1) QueryForInt():

Query: String sqlQuery = "SELECT SNO FROM STUDENT WHERE SNAME = ? NO@IT";

- * In the above query, we are selecting DB Table Name: STUDENT only one column & the Resultset Datatype is Integer.

- * So To execute the query we can use queryForInt();

SNO	SNAME
1	NOIT1
2	NOIT2

- * The Expected results should not be more than one and should not be zero also

```
int studentNo = getJdbcTemplate().queryForInt(sqlQuery);
```

(2) QueryForObject():

Query: String sqlQuery = "SELECT SNAME FROM STUDENT WHERE SNO = ?"

- * In the above query, we are selecting only one column and Resultset datatype is String. So to execute the query we can use queryForObject();

- * The Expected results should be exactly one

- * QueryForObject() can be used for any type of results datatype Because we pass the datatype to queryForObject().

```
String studentName = getJdbcTemplate().queryForObject(  
    StudentQuery, String.class)
```

- * QueryForObject() method can be used for the previous example also.

Query : String sqlQuery = "SELECT SNO FROM STUDENT WHERE SNAME = ?; N@tE";

* In the above Query, we are selecting only one column and Results Data type is integer. So to execute the query we can use queryForObject().

String studentName = getJdbcTemplate().queryForObject("studentQuery",
 toString, String.class)

How To Set (or) Pass The Values To The Query:

String studentQuery = "STUDENT SNAME FROM STUDENT
WHERE SNO = ?"

String studentName = getJdbcTemplate().queryForObject("studentQuery",
 new Object[]{1}, String.class)

* queryForObject 2nd Argument takes the values in the form of
Object Array.

(3) QueryForList :

Query : String query = "SELECT SNO, SNAME FROM STUDENT";

List<studentList> = getJdbcTemplate().queryForList(query);

Iterator iterator = studentList.iterator();

Map row = null;

while (iterator.hasNext())

{ row = (Map) iterator.next();

SOP(" StudentNo " + row.get("SNO"));

SOP(" StudentName " + row.get("SNAME"));

}

Note : In General List contains student Objects [We write Logic to
convert > Student and Add Student > List], Here List contains
Map object and the Map contains one row... [Map key will be

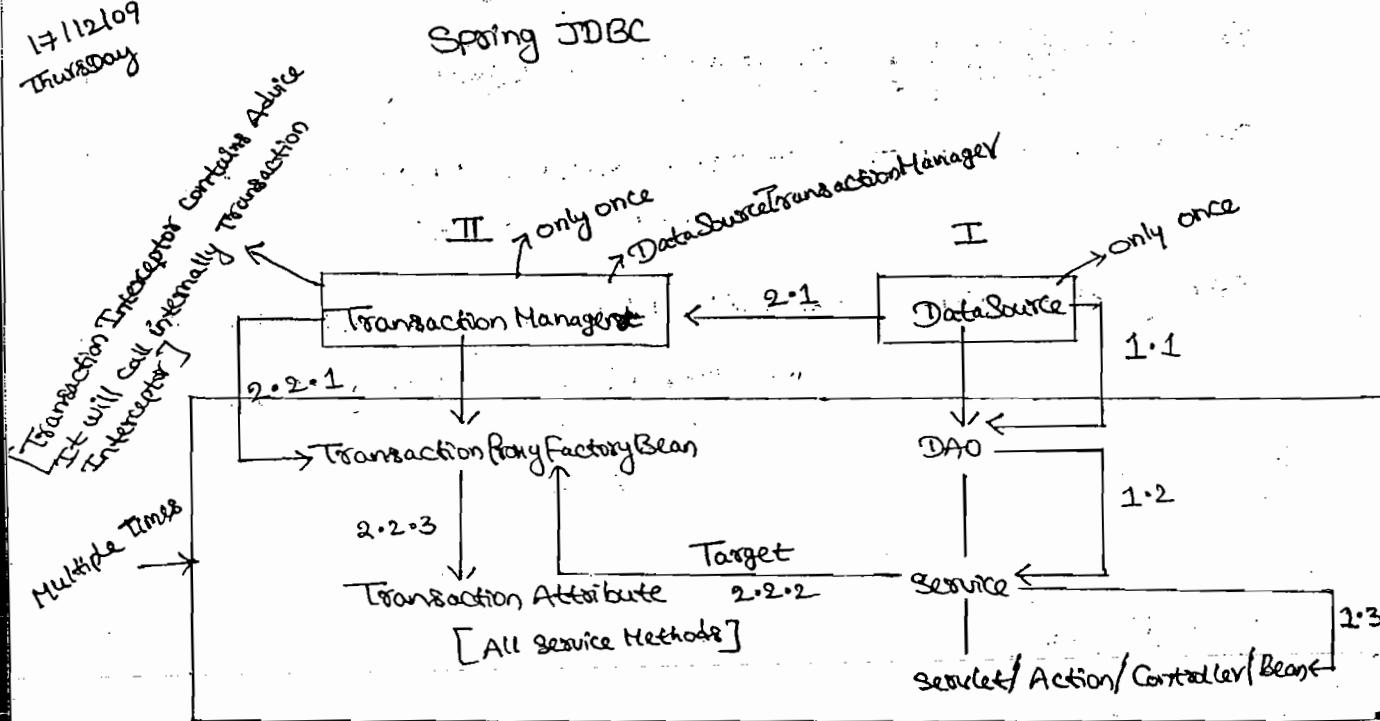
Query: String query = SELECT SNO, SNAME FROM STUDENT
WHERE SNO = ?

Map row = getJdbcTemplate().queryForMap(query, new Object[]{1})

* Map contains a single row. [Map key will be the column]. Here the expected results exactly should 1.

* Using queryForMap() we get a row in the form of Map object.

17/12/09
Thursday



Steps To Apply Transaction Management in Spring JDBC:

I. Inject All the Dependent Objects till the Action class / Controller (Spring) Bean (JSP)

1.1. Configure (or) Create the DataSource object and Inject to the DAO class.

1.2. Configure DAO Inject to Service class

1.3. Inject the Servlet object to Action / Controller / Bean

II. Apply Transaction Management

2.1. Create the Transaction Manager object while Injecting Database / SessionFactory / EntityManager ... etc.

2.2. Configure the proxy object

Inject the below 3 Values to TransactionProxyFactory Beans

- 2.2.1. Inject Transaction Manager
- 2.2.2. Inject Service as a Target
- 2.2.3. Inject the Service class Methods, Transaction Attribute types and Exception names as Transaction Attributes [setTransactionAttributes()]

Notes: DataSource will be created only once and TransactionManager is created only once based on the Service & Dao classes. We configure DAO, Service & TransactionProxyFactory Bean multiple times.

18/12/09
Friday

Spring AOP

* Spring Internally Applies Transaction logic using AOP concepts.

```
public class TransactionProxyFactoryBean {  
    private final TransactionInterceptor transactionInterceptor = new  
        TransactionInterceptor();
```

```
public class TransactionInterceptor extends TransactionAspectSupport {  
    public Object invoke (...) {  
        createTransactionIfNecessary(...);  
        try {  
            ...  
            catch (Throwable ex) {  
                completeTransactionAfterThrowing(...);  
            }  
            commitTransactionAfterReturning(...);  
        }  
    }
```

```
public abstract class TransactionAspectSupport {  
    ...  
    • commit (...);  
    ...  
    • completeTransactionAfterThrowing (...);  
    ...  
    • rollback (...);  
}
```

Advice (CommonLogic...) Interceptor

```
{ try
    {
        con.setAutoCommit(false); // TransactionManager class
        con.commit(); // Transaction Interceptor class
    }
    catch (Exception e)
    {
        con.rollback(); // Will be there inside Transaction Interceptor class
    }
}
```

Service :

```
public int updateStudent ( Student student )
{
    return dao.getDao();
}
```

Advice & Service → Proxy

Take Service Directly → Wrong

Take Service from Proxy → Then proxy return Service & this Service will be
with Advice Logic. [Transaction Logic]

* It is going to Interrupt our Service, So we call it as Interceptor

Advice : (Interceptor)	Methode
1. Before → MethodBeforeAdvice	before()
2. After → AfterReturningAdvice	afterReturning()
3. Exception → ThrowsAdvice	afterThrowing()
4. Around → MethodInterceptor	invoke()

19/12/09
Saturday

* Spring Simplifies Transaction logic using AOP Concepts

* Aspect Oriented programming, i.e. going to Cut the Object into pieces
pieces and At Runtime it is going to Add All those pieces.
Thinking the Oop in a different way is called AOP.

* We cut the object in 5 ways.

1. Method before
2. Method after
3. throws [Exception]
4. Method around [before, after, throws]

Steps To Implement:

1. Implement the Advice classes and write the common logic [Transaction, logging, security ...etc].

* We call the common logic as Cross-cutting Concerns. Advice is going to intercept the business method we call it as Interceptor.

2. Implement the Business methods.

3. Don't take the Business object directly.

* We need to take the proxy Business object Means Give the Business object & Advice to the proxy and Take the Business object from the proxy then at Runtime the Advice will be applied on Business object.

Note: Spring Transaction Management Internally uses Around Advice [TransactionInterceptor implements MethodInterceptor Means AroundAdvice]

Note: For the Transaction Management, we don't implement AOP concepts we configure AOP concepts and Spring internally implements AOP concepts.

* For Logging, Security & others we implement AOP concepts in the project.

SingleColumnRowMapper:

* To process the ResultSet, we have to take Any Java class and we need to implement RowMapper interface, SingleColumnRowMapper is a Helper classes and which is used to process SingleColumnResults.

* SingleColumnRowMapper implements RowMapper interface.

```
StringBuffer StudentQuery = new StringBuffer(" SELECT SNAME  
FROM STUDENT ");
```

* To Execute the Query, we can choose the below "2 Approaches". But The "2nd one" is the "preferable one", we "never implement" using the "1st way".

~~11~~
SingleColumnRowMapper singleColumnRowMapper = new SingleColumnRowMapper
(String.class);

List studentList = (List) getJdbcTemplate().query(studentQuery.toString(),
singleColumnRowMapper);
(OR)

StringBuffer studentQuery = new StringBuffer("SELECT SNAME FROM
STUDENT");

List studentList = (List) getJdbcTemplate().queryForObject(studentQuery.
toString(), String.class);

Internal Process:

queryForObject internally will call queryForObject(sql,
getSingleColumnRowMapper(requiredType))

~~12~~ Note: queryForObject() method internally uses SingleColumnRowMapper,
ColumnMapRowMapper Helper classes, we never implement those
classes in our Application code.

and why?

ColumnMapRowMapper:

- * It is a Helper class and it is used to process more than one column results.
- * Spring internally uses this while calling queryForList() method
- * But we never implement this in our Application code.

StringBuffer studentQuery = new StringBuffer("SELECT SNO,
SNAME FROM STUDENT");

ColumnMapRowMapper columnMapRowMapper = new
ColumnMapRowMapper();

List list = (List) getJdbcTemplate().query(studentQuery.
toString(), columnMapRowMapper);
(OR)

List list = (List) getTableTemplate().queryForList(studentQuery.
toString())

queryForList() Internally will call query method, while passing
columnMapRowMapper object;

query(sql, getColumnMapRowMapper())

Notes 2nd one is the preferable one

SNO	SNAME
1	NOITI → Map contains Two Objects
2	NOITI → Map

It contains
TWO MAP
objects

21/12/09
Monday

Working with Stored Procedure:

1. Take Any Java class and Extends StoredProcedur
2. Declare In & Out Parameter types, In parameter values are call execute() method (or) Super class execute() method.
 - 2.1. Inside the constructor
 - 2.1.1. Call the Super class constructor while passing template of the procedure name.
 - 2.1.2. Declare In Parameter types.
 - 2.1.3. Declare OutParameter types.
 - 2.1.4. Compile the procedure while calling compile method

Set The Inparameter Type — SqlParameter

declareParameter(new SqlParameter("SNO", Types.INTEGER));

Set The Outparameter Type — SqlOutParameter

declareParameter(new SqlOutParameter("SNAME", Types.VARCHAR));

* Call declareParameter() then it will call Super class declareParameter() and it will set the type.

Q2. Write a program which calls super class execute() method & Before call super class execute() method set the procedure values.

```
Map inputParams = new HashMap();
inputParams.put("SNO", studentNo);
super.execute(inputParams);
```

* In and out params values should be in Map object

3. Create object for Stored Procedure implemented class and to the constructor pass the template object and call execute() method which actually calls the Super class execute() method.

```
Map resultMap = new StudentStoredProcedure(getJdbcTemplate());
execute(studentNo);
```

Working with Batch Process:

* If we are inserting or updating one lack (10000) records in the Database for each record we should not execute Prepared Statement and it gives performance issues if create the Prepared Statement in a Batch Process.

* There are 3 ways to Implement the BatchUpdate

1. To Implement the BatchUpdate()

```
getJdbcTemplate().batchUpdate(new String [] {"UPDATE STUDENT
SET SNAME = 'N@IT-U' WHERE SNO = "1", ..."});
```

2. Using batchPreparedStatementSetter Steps

A. To Implement batchPreparedStatementSetter interface and while created implemented class object, pass the list to the constructor

* Override the getBatchSize() method and return the BatchSize using list.size()

* override the setValues() method set Student object to prepared statement.

Don't iterate the List inside the `setValues()` method Because `setValues()` method will be called by framework based on the Batchsize and each iteration (or) each time take the Student object from the List using batchNo ..

`setValues(PreparedStatement, int i)`

`Student student = (Student) studentList.get(i);`
`// Should not iterate List [Wrong]`

3. Using SimpleJdbcTemplate Object we can pass List of Object[] Values at a time.

Note: Initially List contains Student objects and convert student objects to Object[]

`getSimpleJdbcTemplate().batchUpdate(query, batchValues);`

`batchValues` should contain Object Array

Working with Cursor

* cursor is just like Resultset and procedure might be returning the records in the form of cursor.

`cursor;`

`CREATE OR REPLACE PACKAGE S-PACKAGE AS`

`TYPE S-CURSOR IS REF CURSOR;`

`END;`

`Procedure:`

`CREATE OR REPLACE PROCEDURE S-PROCEDURE`

`(S-DETAILS OUT S-PACKAGE, S-CURSOR, S-NAME
IN VARCHAR) AS`

`BEGIN`

`OPEN S-DETAILS FOR`

`'SELECT SNO, SNAME FROM STUDENT WHERE
SNAME = S-NAME';`

`END S-PROCEDURE;`

22/11/2020
Tuesday

1. Take Any Java class and Implement CallableStatementCreator

1.2. override createCallableStatement() method, create the CallableStatement object, declare In & Out parameters

2. we need to iterate the results [cursor returns results].
So, implement RowMapper interface.

3. Execute the Stored Procedure:

Set the RowMapper implemented object has a OutParameter to List or ArrayList object. Then pass ArrayList and CallableStatementCreator implemented object to call() method.

List parameters = new ArrayList();
parameters.add(new SqlOutParameter("StudentDetails",
OracleTypes.CURSOR, new StudentRowMapper()));

Map resultMap = getJdbcTemplate().call(new
StudentCallableStatementCreator(studentName), parameters);

Note: resultMap Map contains Key as Student Details and the results as in the form List... [Value will be list].
List contains Student objects.

Working with SqlUpdate:

UPDATE STUDENT SET SNAME=? WHERE SNO=?

* We can execute the above query using

JdbcTemplate.update(query, values) (or) using SQLUpdate ...

Steps To Implement:

1. Extend SqlUpdate to Any Java class

2. Inside the constructor

2.1. Call the Super class constructor while passing Datasource object

2.2. Set the parameter types

```
declareparameter(new SqlParameter("SNAME",Type.VARCHAR));
```

Q.3. Call compile() method.

ement

To execute the query create SqlUpdate implemented object, then call update() method, while passing values.

```
new StudentSqlUpdate(getDataSource()).update(new Object[]  
{ new Long(1), "NOTE" });
```

Working with MappingSql:

Note: only point out is given by the Sir.

26/12/09
Saturday

Spring MVC

* Spring MVC follows MVC2 Architecture [MVC and The other Design patterns]

Front Controllers

* Front Controller is a Design Pattern, If any request is coming, It will be Delegated to View and Model through Front Controller.

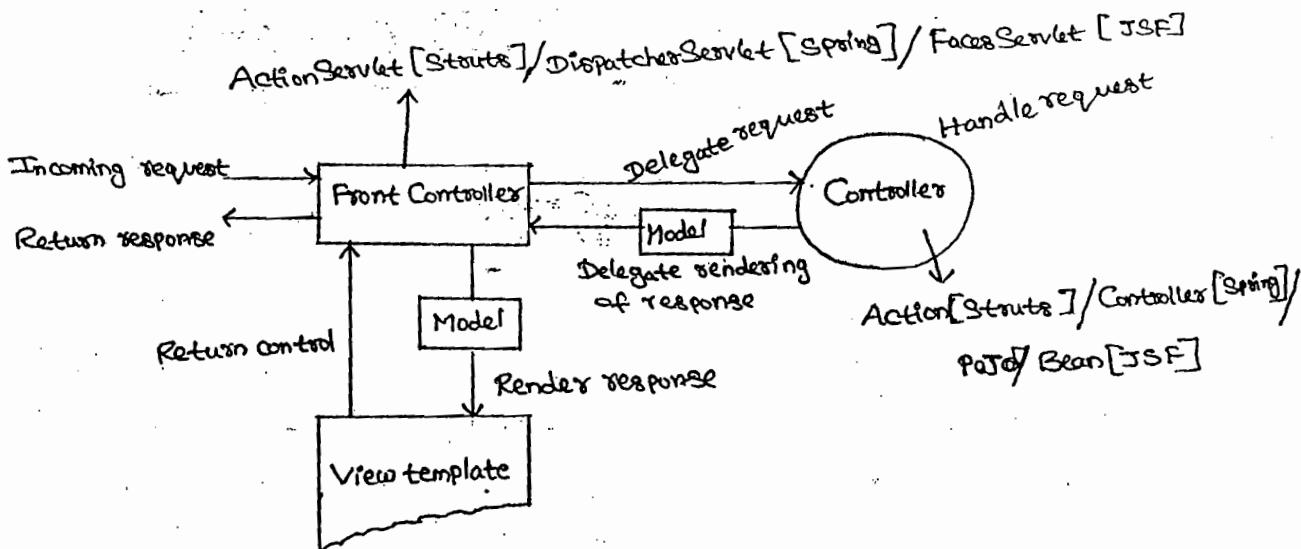


Fig: Servlet Engine

Advantages:

1. Controller classes can implement Controller interfaces (or) Can Extend Abstract classes.

According to the Design Action (or) Controller class should depend on interfaces first, But Struts1.2 is not following that.

2. Form object should not Extend Any API class and In Spring, we call form object as Command objects to those classes we no need to Extend Any API class [Struts1.2 is not following that]

3. Spring Controller classes provides some common functionality, which is required for the project.

3.1. Accept only GET Request (or) POST Request ...etc.

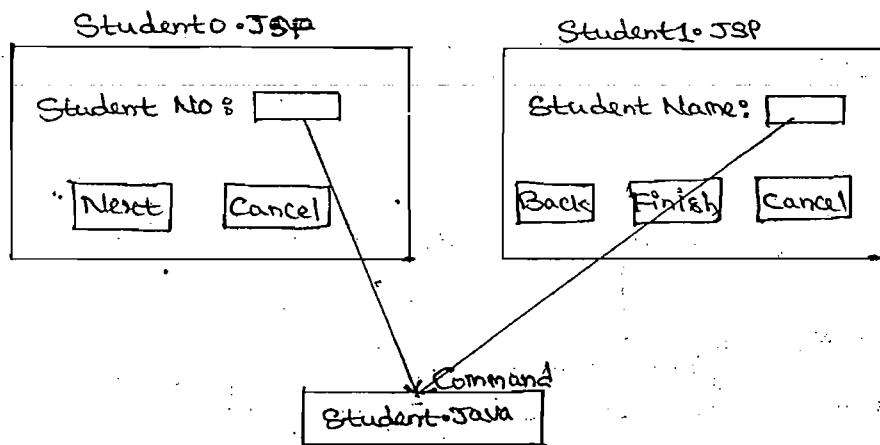
3.2. Set the Browser Cache.

3.3. Execute the Controller Method in a Synchronized block.

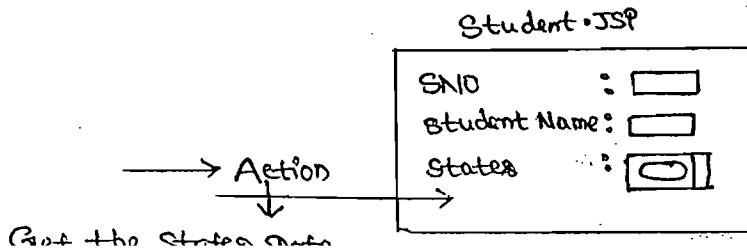
3.4. Execute the Controller Method If there a pre-existing Session.

All the Above features is not provided by Struts1.2

4. It provides "Wizard Style"



If we want to pre populate the data into the JSP, in Struts Applications instead of forwarding the request directly to the JSP first we forward the request to Action class. Then forward the request to JSP



5. In Spring Applications, we can implement it using formBackingObject concepts.
6. We can do Pre & Post process for the Controller, In Struts 1.2 we can do Only Preprocess and PostProcess is not possible.
7. In Struts 1.2, the complete configuration will be there in struts-config.xml (or) <name.xml>, In Spring Applications, The complete configuration should be configured <ServletName - Servlet.xml> (or) <name.xml>
8. In Struts Applications, Controller will be ActionServlet and Spring Applications DispatcherServlet is the Controller.
9. In Struts Applications, Action classes will be Identified using ActionMapping tag and In Spring Applications, Controller classes will be Identified using HandlerMapping classes.
10. In Struts Applications, It is difficult to switch from one presentation Layer (JSP) to Different presentation Layer.
11. In Spring Applications, It will be Easy to switch from presentation to Different presentation layer. Because in Struts, the View will be identified using ActionForward object and In Spring, It is using ViewResolver classes.

HandlerMappings:

1. BeanNameHandlerMapping (X) [A]
2. SimpleUrlHandlerMapping (X) [A]
3. ControllerClassNameHandlerMapping [N/A]
4. CommonsPathMapHandlerMapping [N/A]

ViewResolvers:

1. BeanNameViewResolver [A]
2. InternalResourceViewResolver (X) [A]
3. XmlViewResolver
4. VelocityViewResolver
5. ResourceBundleViewResolver
6. XsltViewResolver

Note: All these Resolvers are used in the projects

Understanding DispatcherServlet init() method:

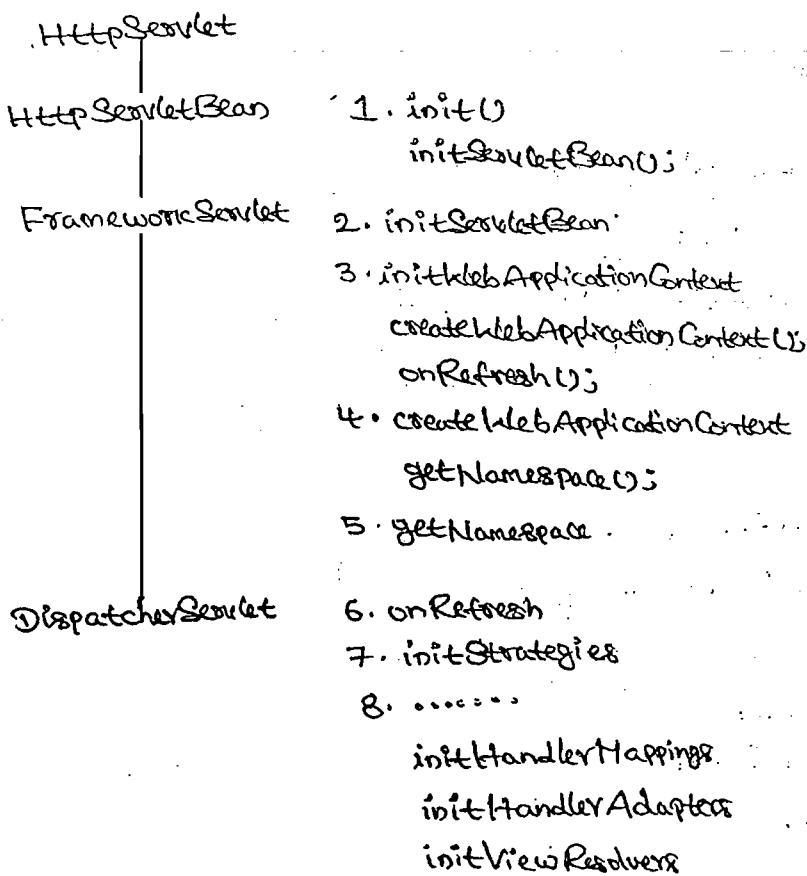
How The DispatcherServlet reads spring-config.xml/DispatcherServlet-servlet.xml

- * In Any Web Application init() Method will parse XML [reading ... etc]

```
< servlet >
< servlet-name > DispatcherServlet < /servlet-name >
< servlet-class > org.springframework.web.servlet.DispatcherServlet < /servlet-
class >
< init-param >
    < param-name > contextConfigLocation < /param-name >
    < param-value > /WEB-INF/DispatcherServlet.xml < /param-value >
< /init-param >
< load-on-startup > 1 < /load-on-startup >
< /servlet >
```

- * If contextConfigLocation is configured, then spring-config.xml file should be DispatcherServlet.xml, and If contextConfigLocation is not configured then xml file name should be DispatcherServlet-servlet.xml

Workflow:



1. `createWebApplicationContext()` method creates `WebApplicationContext`
2. `getNamespace()` is used to identify the XML file
3. `onRefresh()` is used to configure the default strategies (as) default settings [`DefaultHandlerMapping`, `HandlerAdapter` & `ViewResolver`]

HttpServletBean:

```
public abstract class HttpServletBean extends HttpServlet
{
    public final void init()
    {
        initServletBean();
    }
}
```

FrameworkServlet:

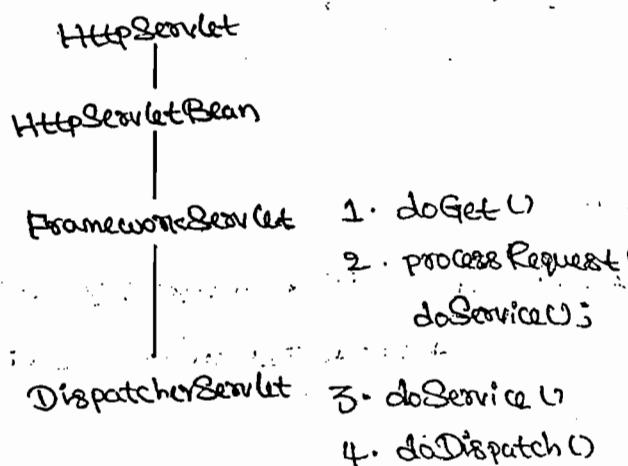
```
public class FrameworkServlet extends HttpServletBean
{
    public static final String DEFAULT_NAMESPACE_SUFFIX = "-Servlet";
    ... initServletBean()
    {
        this.webApplicationContext = initWebApplicationContext();
        initFrameworkServlet();
    }
    ... WebApplicationContext initWebApplicationContext()
    {
        ... WebApplicationContext wac = createWebApplicationContext(...);
        onRefresh(wac);
        return wac;
    }
    ... WebApplicationContext createWebApplicationContext(...)
    {
        wac.setNamespace(getNamespace());
        // getNamespace() = your spring configuration file
        return wac;
    }
    public String getNamespace()
    {
        return (this.namespace != null) ? this.namespace : getServletName()
            + DEFAULT_NAMESPACE_SUFFIX;
    }
}
```

DispatcherServlet :

```
public class DispatcherServlet extends FrameworkServlet  
{  
    private static final String DEFAULT_STRATEGIES_PATH = "...";  
  
    protected void onRefresh ( ApplicationContext context )  
    {  
        initStrategies ( context );  
    }  
  
    ... initStrategies (...);  
    ...  
    initHandlerMappings ( context );  
    initHandlerAdapters ( context );  
    ...  
    ... initHandlerMappings (...);  
    {  
        this . handlerMappings = null ;  
        this . handlerMappings = new ArrayList ();  
        // We keep HandlerMappings in stored order  
        Collections . sort ( this . handlerMappings, ... );  
    }  
    ... initHandlerAdapters (...);  
    {  
        this . handlerAdapters = null ;  
    }  
}
```

Note : If we don't configure Any handlerMapping; then it takes Default

27/12/09
sunday
DispatcherServlet — service() Method Work-Flow [Request Flow] :



```

.... doDispatch(...)

{
    HandlerExecutionChain mappedHandler = null;

    ModelAndView mv = null;
    mappedHandler = getHandler(...);
    mappedHandler = getHandler();
    HandlerInterceptor[] interceptors = mappedHandler.getInterceptors();
    for (int i=0; i<interceptors.length; i++)
    {
        HandlerInterceptor interceptor = interceptors[i];
        interceptor.preHandle(...); I
    }

    HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());
    // ha → Points to SimpleControllerHandlerAdapter
    // [process]
    mv = ha.handle(request, response, mappedHandler.getHandler()); II

    // mappedHandler.getHandler() returns StudentController
    // Apply postHandle methods of registered interceptors
    for (int i=interceptors.length-1; i>=0; i--)
    {
        HandlerInterceptor interceptor = interceptors[i];
        interceptor.postHandle(request, response, mappedHandler.getHandler(), mv); III
    }

    // Did the handler return a view to render?
    if (mv == null || mv.wasCleared())
    {
        render(mv, processedRequest, response);
    }
    // Trigger after-completion for successful outcome
    triggerAfterCompletion(...);
}

```

```

... doDispatch(...)

{
    HandlerExecutionChain mappedHandler = getHandler(...);

}

* HandlerExecutionChain Return Controller Object using HandlerMapping

HandlerExecutionChain getHandler():

{
    Iterator it = this.handlerMappings.iterator();
    // It uses HandlerMapping Object to identify the Controller Object
    return hm.getHandler(request);
}

```

Note: HandlerExecutionChain = HandlerInterceptor + HandlerMapping

Process/Flow:

[mappedHandler Points → HandlerExecutionChain]

HandlerInterceptor[] interceptors = mappedHandler.getInterceptors();

1. interceptor · preHandle()

HandlerAdapter ha = getHandlerAdapter[Controller]
(mappedHandler.getHandler());

doDispatch() →

2. ha · handle(request, response, mappedHandler
getHandler())

3. interceptor · postHandle()

Note:

public class SimpleControllerHandlerAdapter

{ public ModelAndView handle(HttpServletRequest request, HttpServletResponse response, Object handler)

{ return ((Controller) handler).handleRequest(request, response);

[StudentController.handleRequest(request, response)]

}

}

* DispatcherServlet will call HandlerAdapter · handle() method and
handle() method will call our controller handleRequest() method;

* preHandle(), handle() and postHandle() methods is called by
DispatcherServlet.

Spring

— By Mr. Varma

28/12/2018
Monday

DispatcherServlet - Workflow [ViewResolver] :

DispatcherServlet → render() → resolveViewName()

... doDispatch(...)

{ render(mv, processRequest, response);

}

... render(ModelAndView mv, request, response)

{

View view = resolveViewName(mv.getViewName(), ., ., .);

view.render(mv.getModelInternal(), request, response);

}

... View resolveViewName(String viewName, Map model, ., .),

{

ViewResolver viewResolver = ... ; points to InternalResourceViewResolver

View view = viewResolver.resolveViewName(viewName, locale);

return view;

View points to → InternalResourceView

3

* Notes ViewResolver classes are used to identify the View class.

1. InternalResourceViewResolver
2. ResourceBundleViewResolver
3. VelocityViewResolver
4. BeanNameViewResolver
5. XmlViewResolver
6. XsltViewResolver
7. FreeMarkerViewResolver
8. JasperReportsViewResolver

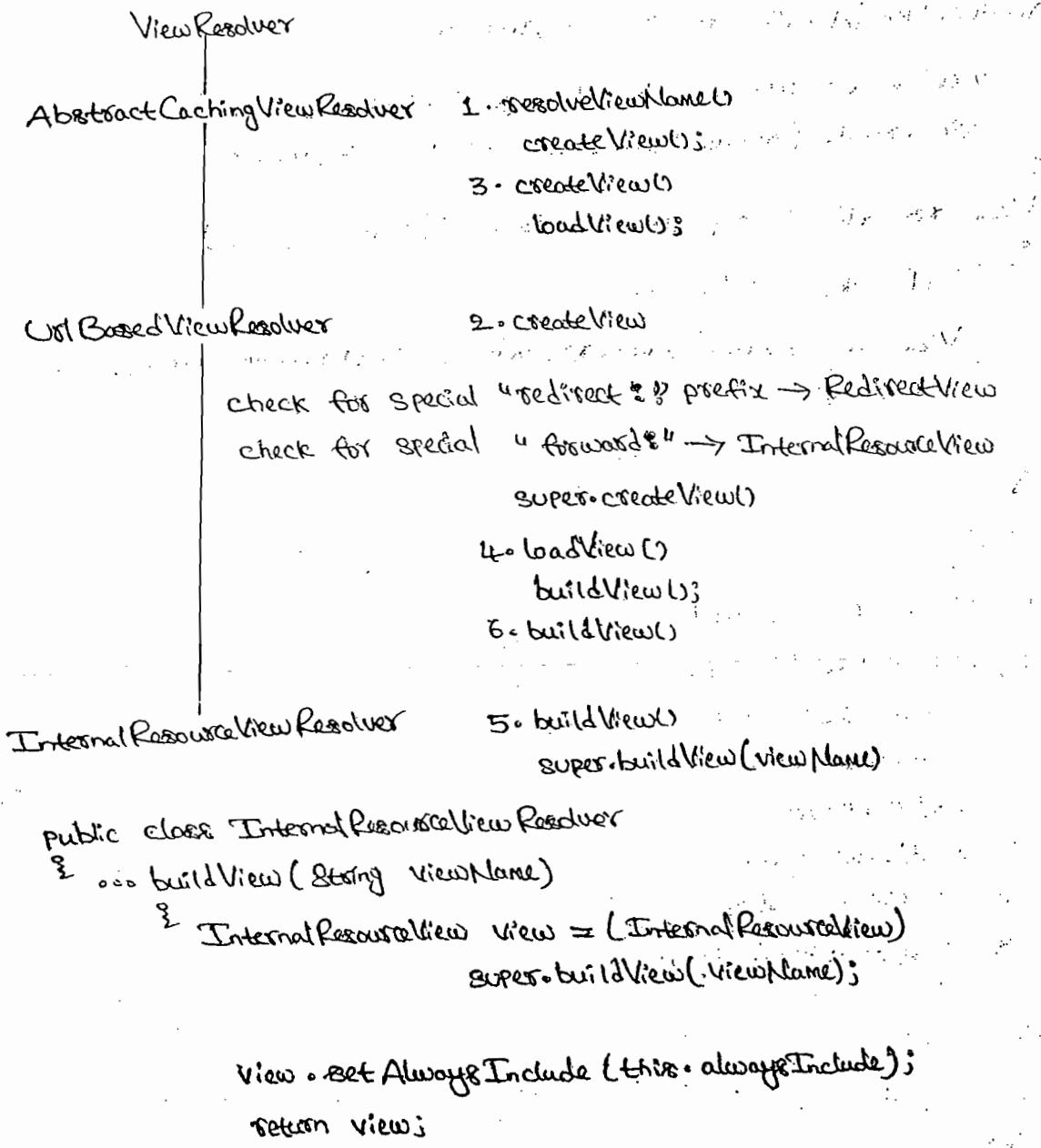
* View classes are used to forward the request to the corresponding View [JSP, Velocity...etc], Means View classes actually contains rd.forward() logic.

1. InternalResourceView
2. JstlView
3. VelocityView
4. FreeMarkerView

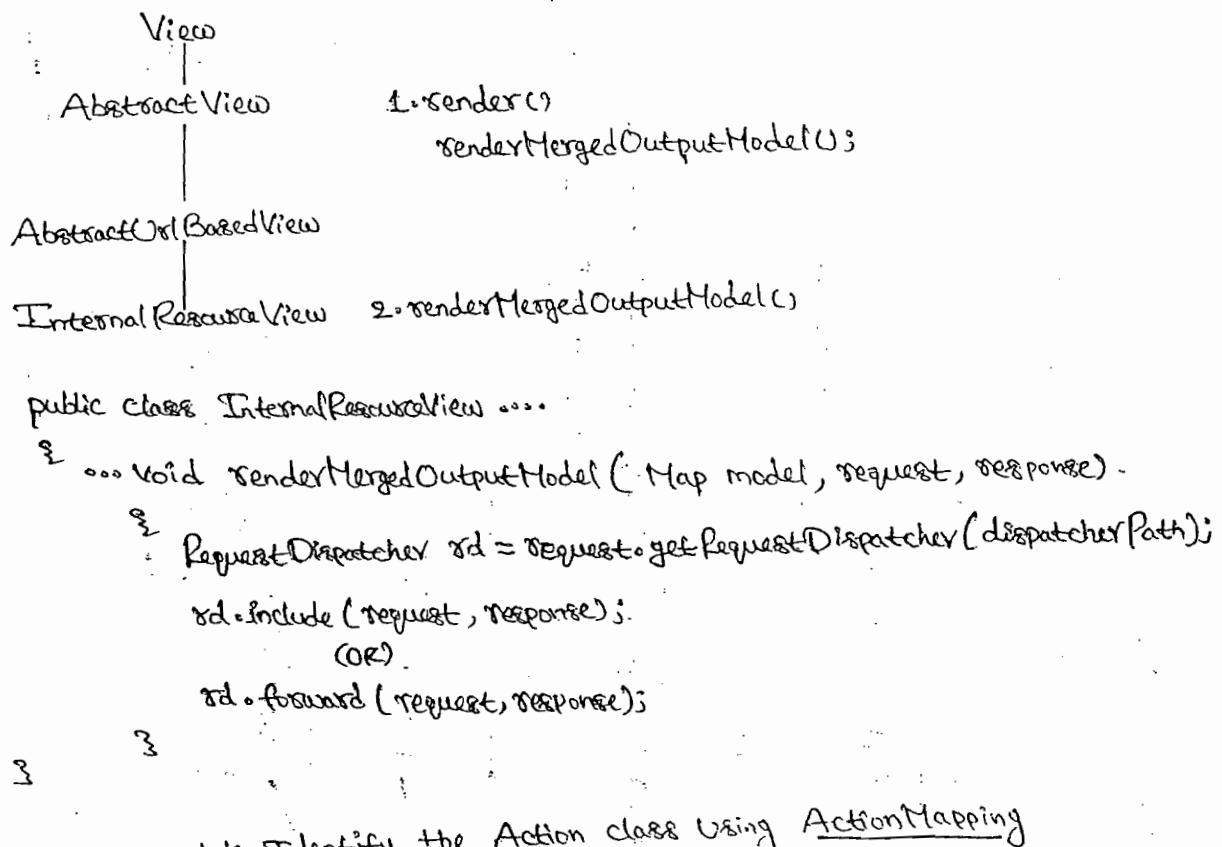
Note: There are 2 render() methods, one is in DispatcherServlet and the other is in View classes which actually forwards the request.

* There are 2 resolveViewName() methods, one is in DispatcherServlet and the other is in ViewResolver class and It is used to Identify the View.

Internal Flow for ViewResolver [resolveViewName() Method].



Internal Flow for View class render() method :-



* In Struts, we Identify the Action class Using ActionMapping

In Spring, it is using HandlerMapping.

execute/handleRequest { Action/Controller } :

* In Struts Applications, execute() method is called from RequestProcessor
• `processActionPerform(...)`

In Spring Applications, handleRequest() method is called from
`HandlerAdapter.handle(SimpleControllerHandlerAdapter)`

* In Struts, execute() method returns ActionForward which contains
Logical Name to identify the View

In Spring, handleRequest()-method returns ModelAndView Object
which contains Logical Name to identify the view and Model Object

[Contains Data for the presentation Layer → Business Data]

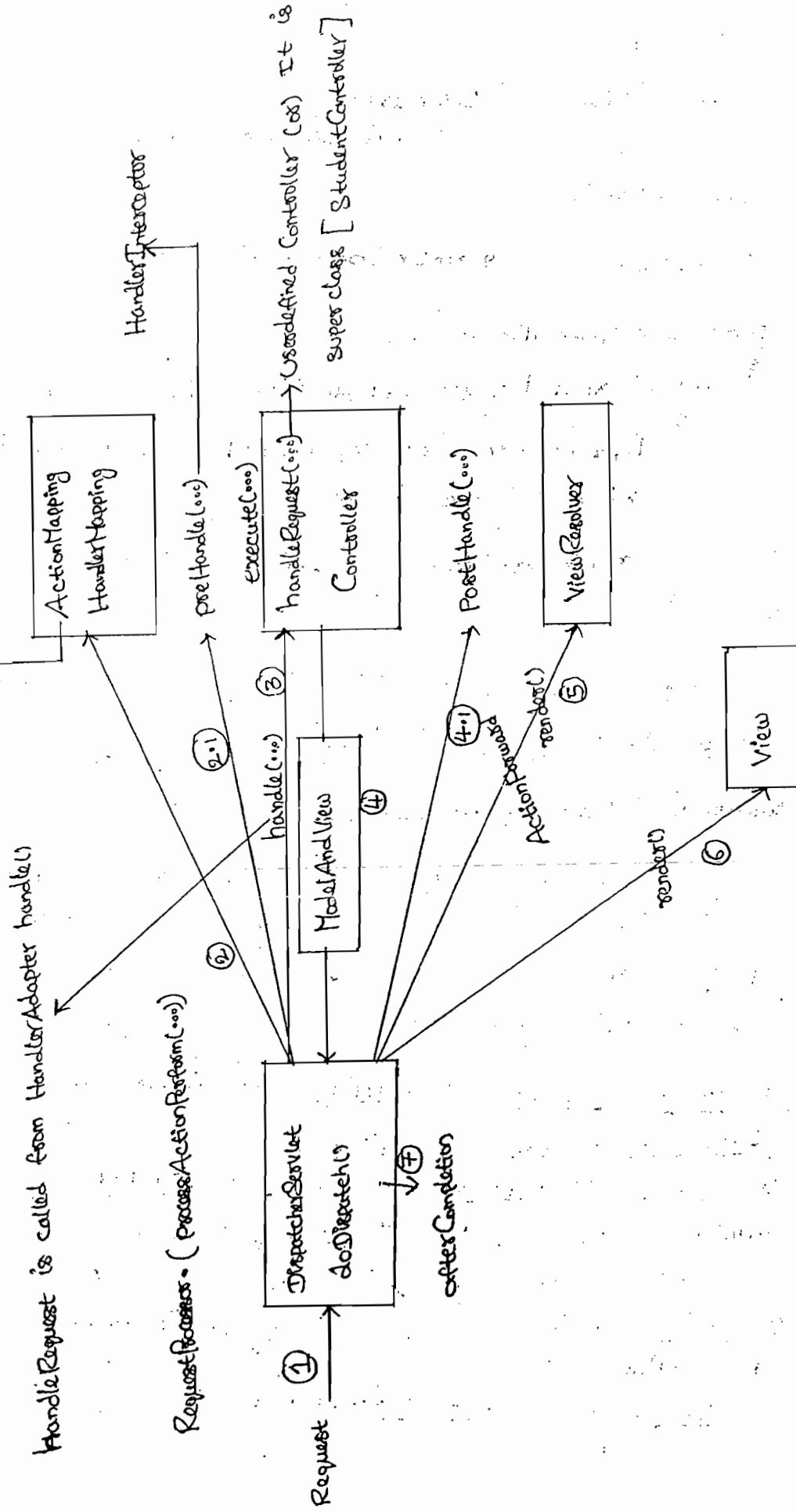
* In Struts, we can create only one flow for "N" No. of requests [using
RequestProcessor]

In Spring's, we can create "N" No. of flows for "N" No. of requests [using
interceptors]

Fig: The Lifecycle of a Request in a Spring MVC

HandlerExecutionChain.getInterceptor() returns chain of HandlerInterceptor objects.

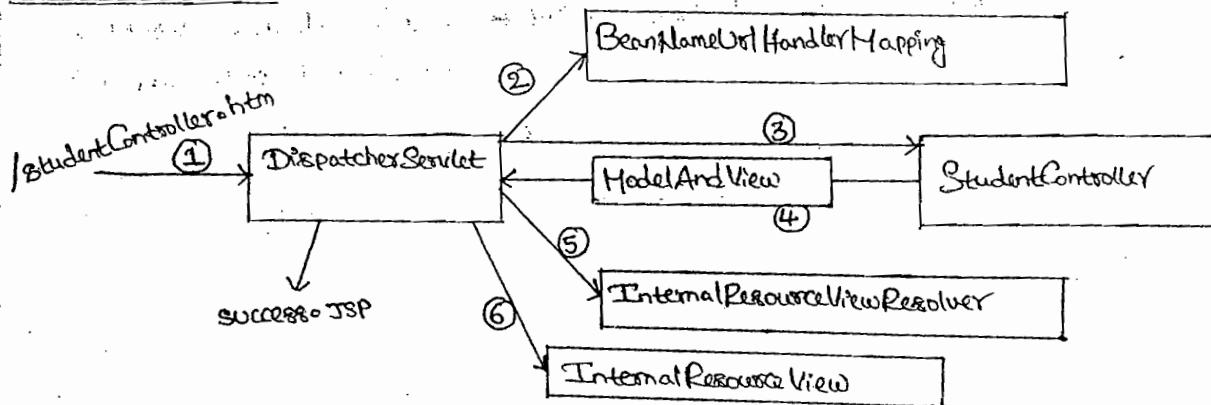
HandlerExecutionChain.getHandler() methods return Controller object internally using HandlerMapping



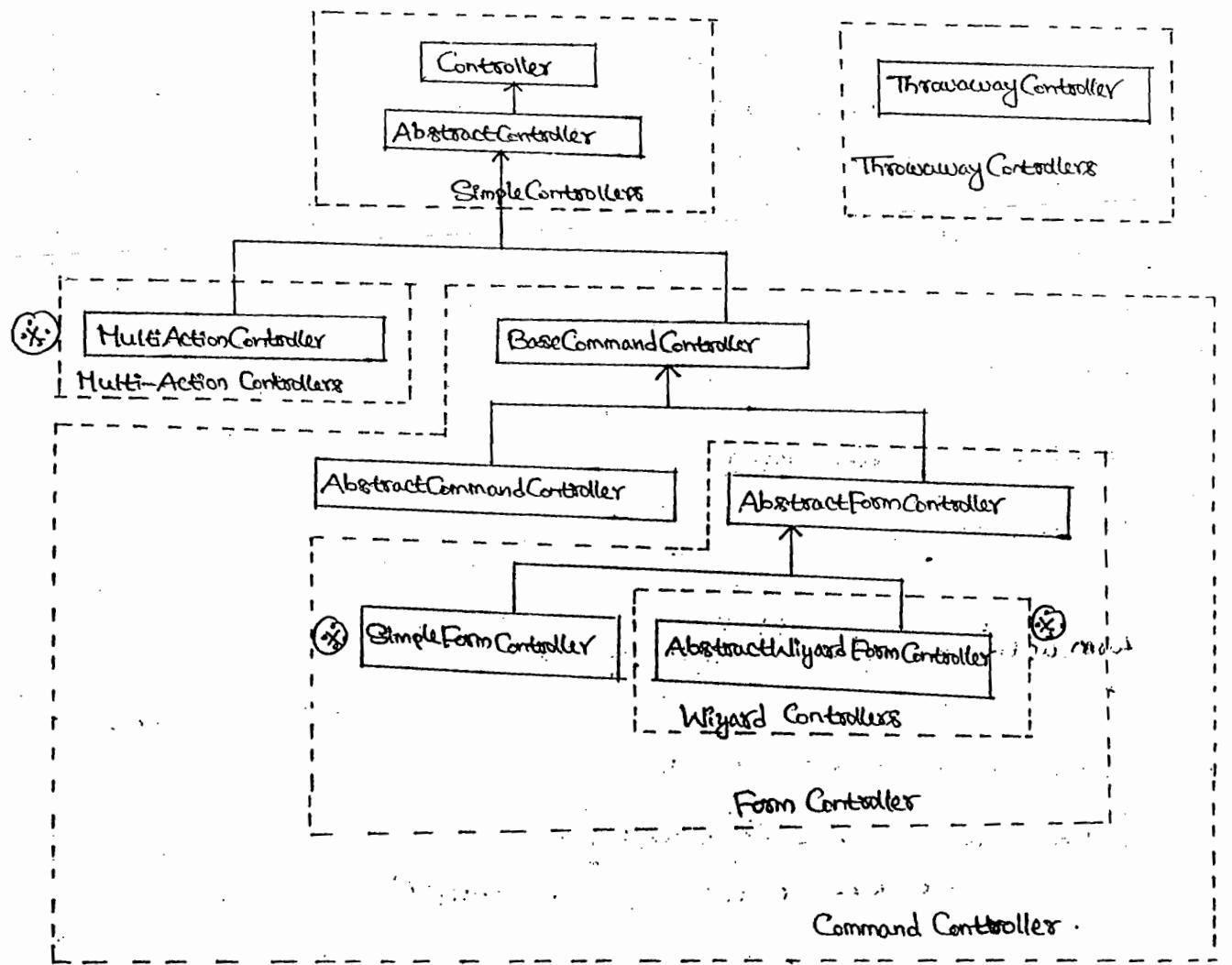
- * In Struts Applications, we call Business Layer from Action classes.
- In Spring Applications, we call Service Layer Means Business Layer from Controller class.

Fig: Processing a request for "/studentController.htm"

HighEnd Diagram:



Different Controller Classes:



29/12/09 Tuesday Controller interface:

- * It is the Base interface for all the controller classes.
- * It has handleRequest() method in SpringMVC any controller workflow should start from handleRequest() method.

public interface Controller

```
{ ModelAndView handleRequest ( HttpServletRequest request ,  
                                HttpServletResponse response ); }
```

- * Any implementation of the Controller interface should be reusable, Thread-safe class, Capable of handling multiple HTTP requests throughout the lifecycle of an Application.
- * Interface should preferably not be implemented by custom controllers directly, since Abstract controller also provided by this package already provide a lot of functionality for typical use cases in web Applications.
- * A few Examples of those controllers :

1. AbstractCommandController
2. SimpleFormController
3. AbstractController

HandlerMapping:

- * This classes are used to identify Controller classes.
- * It has getHandler() method, which returns HandlerExecutionChain object.

BeanNameWithHandlerMapping:

Student.jsp :

- * From the JSP, If the Form Action is like

M-I
<form action = "studentController.htm" /> Then it will check
"/studentController.htm" in the spring-config.xml

DispatcherServlet-servlet.xml file :

M-I
<bean name = "/studentController.htm"
 class = "edu.controller.StudentController" />

- * The StudentController handleRequest() method will be Executed.

Steps To Configure BeanNameUrlHandlerMapping:

```
<bean id = "beanNameUrlHandlerMapping"  
      class = "org.springframework.web.servlet.handler.  
              BeanNameUrlHandlerMapping"/>
```

- * id can be anything
- * If we are not giving the above Entry in the spring-config.xml, Then it takes the default handler Mapping as BeanNameUrlHandlerMapping.

Note: id does not take special character. So we are using name attribute.

ControllerClassNameHandlerMapping:

StudentOneController → Remove the suffix Controller and the other should be small ie. studentone.htm, then it will send to JSP.

* Here the Controller class should be identified using the below process.

* If the Controller name is StudentOneController, then from the JSP the action should be —

Remove the Controller suffix and the others should be small. So it should be "studentone.htm" [Applicable]

* If the Controller name is StudentTwoController, then from the JSP the action should be —

Remove the Controller suffix and the others should be small. So it should be "studenttwo.htm"

Note: If we give "t" as Caps "T" It doesn't work, "studentTwo.htm" [Not Applicable]

* The Convention is to take the short name of the class; Remove the Controller suffix if it exists and return the remaining text, lowercased as the Mapping, with a leading /

Eg: StudentController → /student*

Steps To Configure:

```
<bean name = "controllerClassNameHandlerMapping"
      class = "org.springframework.web.servlet.mvc.support.
      ControllerClassNameHandlerMapping"/> // Name can be Anything
```

- * Just we need to configure this Bean in Spring-config.xml and we need to follow the above rule.

Eg:

1. studentOne.jsp :

```
<form action = "studentone.htm"/> // Then it will call StudentOneController
                                         handleRequest() method
```

2. DispatcherServlet-servlet.xml :

```
<bean name = "studentXXX"
      class = "edu.controller.StudentOneController"/>
```

- * This Bean we are configuring for StudentOneController object and It is not used to identify the controller. So, the name should be Anything [Here the controller is identified using ControllerClassName HandlerMapping based on the above rule].

SimpleUrlHandlerMapping :

- * Here it will check the JSP action in the Spring configuration [DispatcherServlet-servlet.xml] file as the key of properties object and that properties object should be configured using the SimpleUrlHandlerMapping. Instead of properties we can configure it using Map object.

JSP :

M-I

```
<form action = " studentController.htm" />
```

Spring-config.xml :

```
<bean id = "urlMapping"
      class = "org.springframework.web.servlet.handler.
      SimpleUrlHandlerMapping"/>
```

```
<!-- <property name = "mappings" />
```

<props>

M-II

```
<prop key = "/studentController.htm">studentController</prop>
```

```

<property name="urlMap">
  <map>
    <entry key="!/studentController.htm">
      <!-- M-II -->
      <value> StudentController </value>
    </entry>
  </map>
</property>
</bean>
  <!-- M-II -->
<bean id="studentController" class="edu.controller.StudentController" />

```

3/11/2009
Thursday ViewResolver:

- * It is an interface and it has a method resolveViewName() which should contain the logic to identify the view.
- * All the viewResolver classes has to implement ViewResolver interface

InternalResourceViewResolver:

DispatcherServlet-servlet.xml:

```

<!-- Bean -->
<bean name="viewResolver">
  <!-- M-II -->
  <!-- class = "org.springframework.web.servlet.View.InternalResourceViewResolver" -->
  <!-- Property -->
  <property name="viewClass">
    value = "org.springframework.web.servlet.view.JstlView"
  </property> -->
  <!-- Property -->
  <property name="prefix" value = "/WEB-INF/views/"> </property>
  <!-- Property -->
  <property name="suffix" value = ".jsp"> </property>
</bean>

```

- * It is good practice to put JSP files that just serve as views under WEB-INF, To hide them from Direct Access [Eg: via a manually entered URL] only Controllers will be able to access them.

viewClass: It is used to identify View class. If we don't configure any, then it takes InternalResourceView.

prefix: Location of the file [URL]

④ Note: Most of the cases we use InternalResourceViewResolver in the projects.

ResourceBundleViewResolver :

- * Here it checks view classes, ... etc in a properties file. It is used if we are working in I18N concepts.
- * The bundle is typically defined in a properties file, located in the classpath. The default Bundle Basename is "views".
- * This ViewResolver supports localized view definitions, using the default support of `java.util.PropertyResourceBundle`.
Eg: The basename "myview" will be resolved as classpath resources "myview-en.properties", "myview.properties" for a given Locale "en"

Steps To Configure:

1. If StudentController.handleRequest() returns ModelAndView ("success")

2. spring-config.xml [DispatcherServlet-servlet.xml]

```
<bean id = "resourceBundleViewResolver"  
      class = "org.springframework.web.servlet.view.ResourceBundleView  
             Resolver">  
  
<property name = "basename" value = "myview" > </property>  
    (OR)  
<property name = "basenames" >  
    <list>  
        <value> myview </value>  
    </list>  
</property>  
</bean>
```

3. JSP names and View class should be configured in myview-en.properties

M-I.class

success.class = org.springframework.web.servlet.view.JstlView

M-I.class

success.url = /WEB-INF/views/success.jsp

Xml ViewResolver :

- * In Xml ViewResolver, It will check ViewClass in a xml file.
- * Xml ViewResolver, the file will typically be located in the WEB-INF directory.
- * default is " /WEB-INF/views.xml "
- * This ViewResolver does not support internationalization, consider ResourceBundleViewResolver. If you need to Apply different view resources per locale.

Steps To Configure:

1. If StudentController.handleRequest() returns ModelAndView("success");
2. In Spring-config.xml

```
<bean id="xmlViewResolver" M-I  
    class="org.springframework.web.servlet.view.XmlViewResolver">  
    <property name="location"  
        value="/WEB-INF/my-views.xml" />  
</bean>
```

2. So, Here the views should be configured in my-views.xml.

```
<beans> M-I  
    <bean name="success"  
        class="org.springframework.web.servlet.view.JstlView">  
        <property name="url" value="/WEB-INF/success.jsp" />  
</bean>  
</beans>
```

Internal Code:

```
public class XmlViewResolver ...  
    {  
        public final static String DEFAULT_LOCATION = "/WEB-INF/views.xml";  
        public void setLocation(Resource location)  
        {  
            this.location = location;  
        }  
    }
```

1/1/2010 View:

Friday We can Implement Our Own View Classes

Steps To implement:

1. Implement View interface and override getContentType() & render() methods
Forward the request to Corresponding View Using RequestDispatcher.

```
public class StudentView implements View
{
    public String getContentType()
    {
        return "text/html";
    }

    public void render ( Map map , HttpServletRequest request
                        HttpServletResponse response ) throws Exception
    {
        RequestDispatcher rd = request.getRequestDispatcher (" /WEB-INF/
views /success.jsp");
        rd.forward ( request , response );
    }
}
```

2. Configure View class using ResourceBundleViewResolver.

views.properties

success.class=org.view.StudentView

Handler Adapter:

```
public class DispatcherServlet ...
{
    protected void doDispatch ( HttpServletRequest , HttpServletResponse )
    {
        HandlerAdapter ha = getHandlerAdapter ( mappedHandler . getHandler () );
        mv = ha . handle ( processedRequest , response , mappedHandler
                           [ StudentController ] );
        // ha points to SimpleControllerHandlerAdapter
    }
}
```

* In Struts Applications, we Cannot implement Action type of framework classes.

* In Spring Applications, we Can implement our own framework type of controller interface and we can implement our own HandlerAdapter classes also.

```

public class SimpleControllerHandlerAdapter {
    public ModelAndView handle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        return ((Controller) handler).handleRequest(request, response);
        // handler points to StudentController
    }
}

```

- * HandlerAdapter classes will call Controller class .handleRequest() method so, we need to implement logic to call userdefined Controller interface implemented class method [StudentController] in Adapter class .

Steps To implement own Controller interface and Adapter class :

1. provide an interface.

```

public interface Controller {
    public ModelAndView handleStudentRequest (HttpServletRequest
        request, HttpServletResponse response) throws Exception;
}

```

- * It is used to provide the contract between framework and developer team .

2. Implement HandlerAdapter interface and override getLastModified(), handle() & supports() methods .

- * We need to implement the logic to call the developer implemented Controller class handleRequest() method from the handle() method

- * Using supports() method, we implement the logic, what type of classes this Adapter class has to work .

- * This example supports for the Controller classes , which are implementing userdefined Controller interface .

3. Configure the Adapter class in the Spring Configuration file .

```

<bean name = "studentHandlerAdapter"
      class = "org.springframework.web.servlet.HandlerAdapter">

```

How To Implement An preProcess() & The postProcess() for Controller Classes:

How To Implement ExceptionResolver:

- * If we are not handling Any Exception in the Controller ExceptionResolver will forward request to the corresponding Error page

HandlerInterceptors

Steps To Configure:

1. Implement HandlerInterceptor interface and override preHandle(), postHandle() and afterCompletion() methods.

Before handleRequest() preHandle() is called and After handleRequest() postHandle() is called, After the View afterCompletion() is called

2. Configure Interceptor Details in Spring configuration file

- 2.1. Configure Interceptor Bean

```
<bean id = "studentInterceptor"  
      class = "edu.interceptor.StudentInterceptor"/>
```

- 2.2. Register Interceptor with HandlerMapping

```
<bean id = "beanNameOfHandlerMapping"  
      class = "org.springframework.web.servlet.handler  
          BeanNameOfHandlerMapping">
```

```
  <property name = "interceptors">  
    <list>  
      <ref local = "studentInterceptor"/>  
    </list>  
  </property>
```

```
</bean>
```

```
public class BeanNameOfHandlerMapping {
```

```
  public void setInterceptors (Object[] interceptors)
```

```
  {  
    this.interceptors = addAll (Array.asList (interceptors));  
  }
```

```
}
```

```
// It is there in xxHandlerMapping super class.
```

Steps To Configure ExceptionResolver:

- * If we are missing Any ExceptionHandling in the Controller class those Exceptions is handle by ExceptionResolver:

1. Inject Exception type and the Error Page in the form of Properties object to ExceptionResolver.

```
<bean id = "exceptionResolver"
      class = "org.springframework.web.servlet.handler.
SimpleMappingExceptionResolver">
    <property name = "exceptionMappings">
      <props>
        <prop key = "java.lang.Exception">error</prop>
      </props>
    </property>
</bean>
```

Controller Classes

1. AbstractController:

- (1)- In Servlet, how to control, It should be executed for only get() method (or) only post() method.

```
<form action = "/StudentServlet" method = "GET">
```

- * Then there must doGet() method, If it is not available, then it will throw Exception.

- * If we want to implement the logic for only doPost() method

```
public class StudentServlet extends HttpServlet
```

```
{ public void doPost (Request, response)
```

```
{ String method = request.getMethod();
```

```
if ("Get".equals(method))
```

```
{ throw new Exception("....");}
```

```
Logic....
```

* If it is Struts Applications,

<form action = " /StudentAction" method = " GET " >

public class StudentAction extends Action

{ ... execute (...) }

{ String method = request.getMethod();

if (" GET ".equals(method))

{ throw Exception(" ... "); }

Logic ...

}

(2) How to check the pre-existing session and, how to process our

Servlet (or) Action class for only pre-existing session. If there
is no session, then throw Exception.

public class StudentServlet extends HttpServlet

{ public void doPost (req, res)

{ HttpSession session = req.getSession (false);

if (session == null)

{ throw Exception(); }

}

}

* In Struts also, we need to implement the same logic in Action class.

(3) How to execute . Action class execute() method in a Synchronized

block.

class StudentAction extends Action

{ ... execute (...) }

{ synchronized (...)

{ }

}

}

class EmployeeAction extends Action

{ ... execute (...) }

{ synchronized

{ }

}

}

- * In the above classes we are repeating Synchronized block code in Synchronized logic.
- * Instead of repeating the logic using Inheritance and Polymorphism we can reduce it.

class abstract StudentAction extends Action

```

    {
        ...
        execute(...)

        {
            synchronized()
            {
                handleExecute(...,...,...)
            }
        }

        public abstract ... handleExecute(...,...,...);
    }

```

class StudentAction extends StudentBaseAction

```

    {
        ...
        handleExecute(...,...,...)
        {
            ...
        }
    }

```

(4) How to Set Browser Cache

response.setDateHeader("Expires", 1L);

response.setHeader("Cache-Control", "no-store"); ...etc.

Note: We should not apply browser cache in the projects, The value suppose to be "-1", But "-1" value we need to set in the projects.

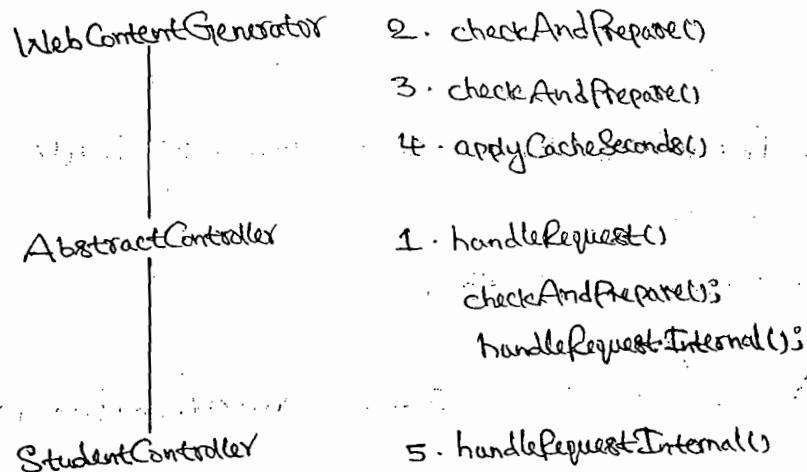
- * In Spring Applications, we don't require All the Above logic. Just we need to insert required values whether the logic should be executed (or) not.

Name	default	Description
supportedMethods	GET, POST	comma-separated (CSV) List of methods supported by this Controller, such as <u>GET, POST & PUT</u>
requireSession	false	Whether a session should be required for requests to be able to be handled by this Controller. This ensures that derived Controller can -without fear to null pointers - call <u>request.getSession()</u> to receive a session. If no session can be found while processing the <u>request</u> , a <u>ServletException</u> will be <u>thrown</u> .
cacheSeconds	-1	Indicates the <u>amount of seconds</u> to include the cache header for the response following on this request. <u>Zero (0)</u> will include <u>headers</u> for <u>no caching at all</u> . -1 [default] will <u>not generate any headers</u> and <u>Any positive number</u> will generate <u>headers</u> that state the amount indicated as seconds to <u>cache the content</u> .
synchronizeOnSession	false	Whether the call to <u>handleRequestInternal</u> should be synchronized around the <u>HttpSession</u> , to serialize invocations from the same client. No effect if there is no <u>HttpSession</u>

* Except synchronizeOnSession, the other 3 logics is performed by WebContentGenerator. synchronizeOnSession logic is performed by AbstractController.

* All the Above 4 logics is controlled & the flow starts internally using handleRequest() method and everything is satisfied, It will call handleRequestInternal(). So we need to override handleRequestInternal

Internal Workflow flow for AbstractController:



Containers

[HandlerAdapter]

```
handle()
{
    if ((Controller) handler).handleRequest(request, response);
}

public abstract class WebContentGenerator ...
{
    ... checkAndPrepare ...
    {
        checkAndPrepare();
    }

    protected final void checkAndPrepare()
    {
        String method = request.getMethod();
        if (!this.supportedMethods.contains(method))
            throw new HttpRequestMethodNotSupportedException(
                method);
    }

    if (this.requireSession)
        if (request.getSession(false) == null)
            throw new HttpSessionRequiredException(
                "pre-existing session required but none found");
    applyCacheSeconds(response, cacheSeconds, lastModified);
}
```

```

public abstract class AbstractController {
    public final ModelAndView handleRequest (...) {
        // Delegate to WebContentGenerator for checking & preparing
        // checkAndPrepare (request, response, this instanceof LastModified);
        // Execute handleRequestInternal in synchronized block if required
        if (this.synchronizeOnSession) {
            synchronized (...) {
                return handleRequestInternal (request, response);
            }
        }
        return handleRequestInternal (request, response);
    }
}

```

Explanation:

- * AbstractController class is a Convenient superclass for Controller implementations, Using the TemplateMethod DesignPattern.
- * A lot of functionality is already provided by certain abstract BaseControllers. The AbstractController is one of the most important abstract class Controller providing Basic features such as the Generation of Caching headers and the Enabling or Disabling of supported methods [GET/POST].

Workflow [And that Defined By Interface]:

1. handleRequest() will be called by the SimpleControllerHandlerAdapter handle() method.
2. SimpleControllerHandlerAdapter handle() method will be called by the DispatcherServlet.
3. Inspection of supported methods [ServletException, if request method is not supported].
4. If session is required, try to get it [ServletException if not found].
5. Set Cache headers, if needed According to cacheSeconds property
6. Call abstract method handleRequestInternal() [optionally synchronizing around the call on the HttpSession], which should be

implemented by extending classes to [StudentController] provide actual functionality to return ModelAndView objects.

Testing The Example :

1. Hit the URL:

http://localhost:8081/AbstractController

2. Change supportedMethods Value to POST in DispatcherServlet-servlet and Hit the URL. Then it will throw Request method "GET" not Supported

3. requireSession is true, so it supposed to throw the Exception. But if we Hit the URL it does not throw the Exception because we are Calling the Controller from the JSP. So JSP will create implicit session object.

Hit the URL in a new browser it gives Exception

http://localhost:8081/AbstractController/studentController.htm?
studentNo=1 & studentName=Neeru

4. First Hit the URL:

http://localhost:8081/AbstractController/

- * Provide the values, then it gives success page. Then take the URL from the success page and from a new browser submit the request
- * If we hit the URL before 10 seconds we don't get the exception because Browser Cache is Applied.

21/10
Saturday

MultiActionController

index.jsp

<u>InsertStudent</u>
<u>UpdateStudent</u>

insertStudent.jsp

Enter the Student Details	
Student No:	<input type="text"/>
Student Name:	<input type="text"/>
<input type="submit" value="InsertStudent"/>	

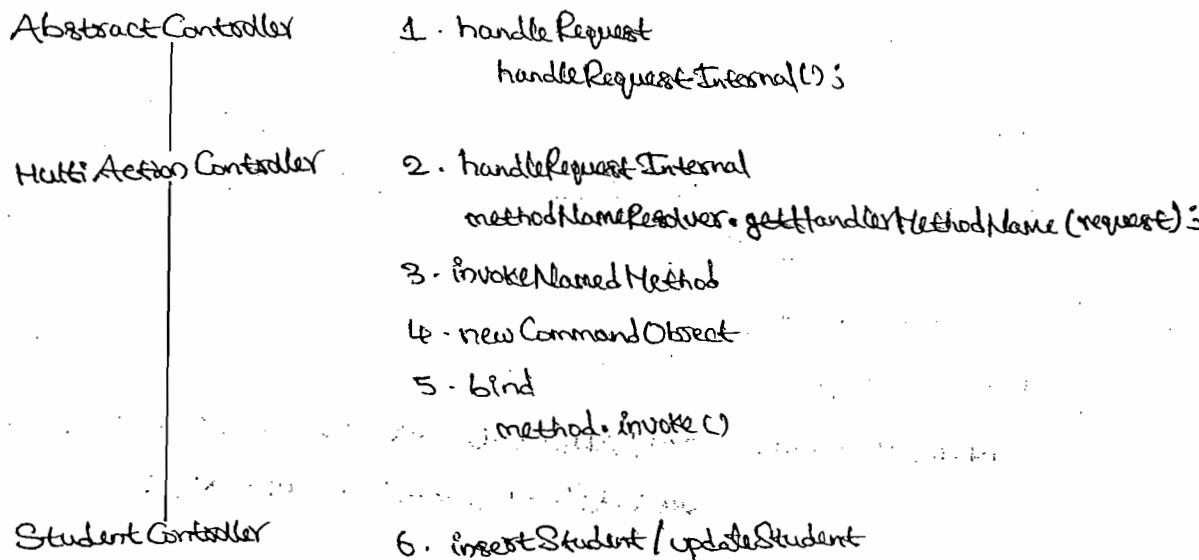
updateStudent.jsp

Enter the Student Details	
Student No:	<input type="text"/>
Student Name:	<input type="text"/>
<input type="submit" value="updateStudent"/>	

- * There are 2 JSP's; insertStudent & updateStudent for those we need to write 2 controllers instead of writing 2, we can write one controller with different methods and the method name will be identified using button value.
- * MultiActionController Means for Multiple Actions (or) requests there will be one controller

3/11/10
Sunday

Internal Workflow Flow for Multi Action Controller:



Containers:

```
((Controller) handler).handleRequest (request, response);
```

```
public class ParameterMethodNameResolver implements MethodNameResolver
```

```
{ public String getHandlerMethodName (request)
```

```
{ return request.getParameter (this.parameterName);
```

```
}
```

```
public class MultiActionController ...
```

```
{ protected ModelAndView handleRequestInternal (...)
```

```
{ String methodName = this.methodNameResolver.
```

```
getHandlerMethodName (request);
```

```
// MethodNameResolver points to ParameterMethodNameResolver Object
```

```
so it will call ParameterMethodNameResolver getHandlerMethodName  
method ...
```

```
return invokeNamedMethod (methodName, request, response);
```

```
}
```

protected final ModelAndView invokeHandlerMethod(...)

{ Method method = (Method) this.handlerMethodMap.get(methodName);

Object command = new CommandObject(...);

params.add(command);

bind(request, command);

Object returnValue = method.invoke(...);

protected Object newCommandObject(Class clazz)

// Internally creates the object using Class.forName(...);

return BeanUtils.instantiateClass(clazz);

protected void bind(HttpServletRequest request, Object command)

{ ServletRequestDataBinder binder = createBinder(request, command);

binder.bind(request);

ValidationUtils.invokeValidator(this.validators[i], command,
binder.getBindingResult());

newCommandObject(): Student object is created

bind(): It will set request values to Student object

method.invoke(): saveStudent() or updateStudent is called.

Steps To Configure:

1.

<input type="submit" name="method" value="insertStudent"/>

M-II

* Button [Action] name should be injected to MethodNameResolver. Means

To get the Controller [Because Controller extends MultiActionController]

<property name="paramName" value="method"/>

2. Controller should extends MultiActionController and we need to implement Different Methods.

class StudentController extends MultiActionController

{ insertStudent(...) M-II }

{ }

- * MethodNameResolver is used to Identify Button Value Based on the Button Name.
- * ServletRequestDataBinder is used to Bind the UI Values with the Java object.

BaseCommandController :

```

handleRequestInternal()
{
    Object command = getCommand(request);
    bindAndValidate(request, command);
    Student student = (Student) command;
    String studentName = student.getStudentName();
    SDP("studentName" + studentName);
}

```

- * Creates an object [the command object], on receipt of a request and attempts to populate this object with request parameter.
- * This Controller is the Base for all Controllers wishing to populate JavaBeans based on request parameters. Validate the content of such JavaBeans using Validators.

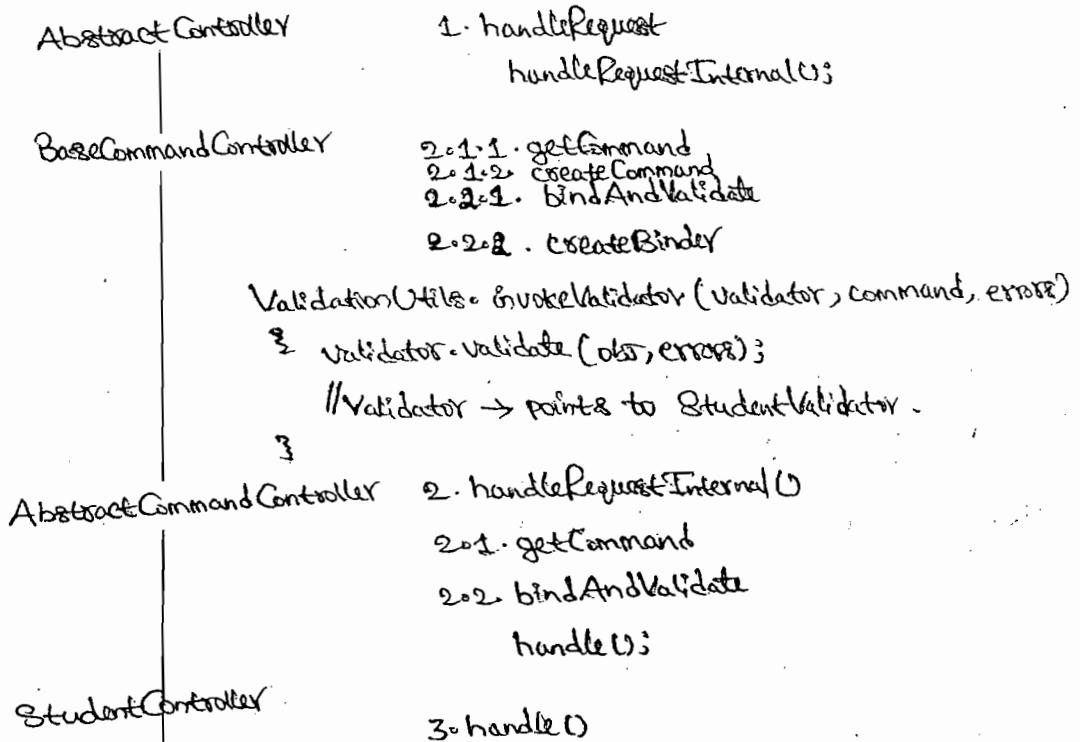
Command class :

- * An instance of the command class will be created for each request and populated with request parameters.
- * A command class can basically be any Java class. The only requirement is a no-arg constructor.
- * The command class should preferably be a JavaBean in order to be able to populate bean properties with request parameters.

Populating Using Request Parameters :

- * Upon receiving a request, Any BaseCommandController will attempt to fill the command object using the request parameters. This is done using the typical and well-known JavaBean Property notation. When a request parameter named "studentNo" exists,
- * The framework will attempt to call studentNo([value]) passing the value of the parameter. Nested properties are of Address supported.

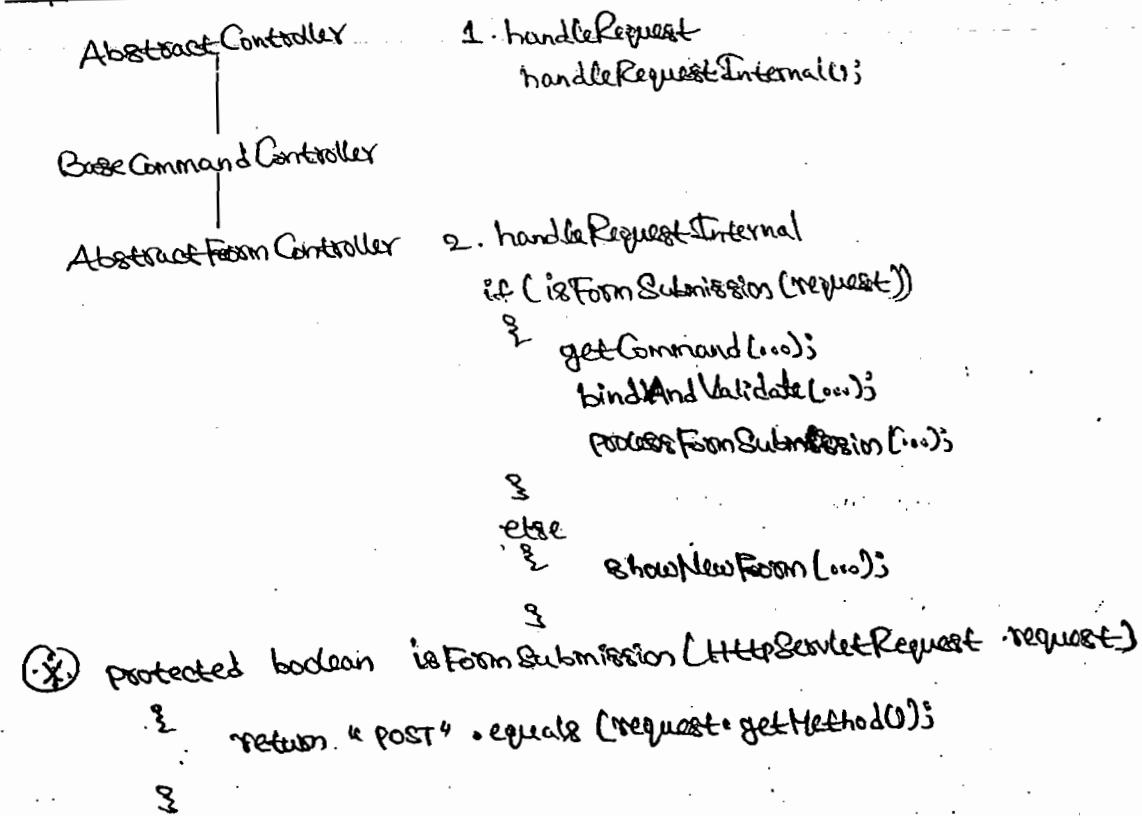
Workflow 3



* If we are overriding initBinder(), onBind() method. Then our own bind process is executed.

AbstractFormController

Workflow 3



- * For instance a parameter named "student.address" will result in a getStudent() & setAddress([value]) call on the command class.

Validators:

- * After the Controller has successfully populated the command object with parameters from the request, it will use any configured validators to validate the object.
- * Validation results will be put in a Error object which can be used in a View to render any input problems.

Workflow [And that Defined by Superclass]:

- * Since this class is an AbstractBase class for more specific implementation, it does not override the handleRequestInternal() method and also has no actual workflow.
- * Implementing classes like AbstractFormController, AbstractCommandController provide actual functionality and workflow.

CommandName command : The name to use when binding the instantiated command class to the request.

commandClass null : The class to use upon receiving a request and which to fill using the request parameters. What object is used and whether or not it should be created is defined by extending classes and their configuration properties and methods.

- * Inject Validator object to Validator property, then it will call validate() method on Validator class.

AbstractCommandController:

- * Autopopulates a command bean from the request. For Command Validation, a Validator [property inherited from BasicCommandController] can be used
- * In most cases, this Command Controller should not be used to handle form submission, because functionality for forms is offered in more detail

```

protected final Object getCommand (HttpServletRequest request)
{
    if (!isSessionFormMode, create a new form-backing object)
        if (!isSessionForm())
            return formBackingObject(request);
    return currentFormObject(request, sessionFormObject);
}

```

* If the request is coming first time isFormSubmission() returns false then internally it will call formBackingObject() method then it will display the JSP [first time if it is post type of request then the page will be submitted].

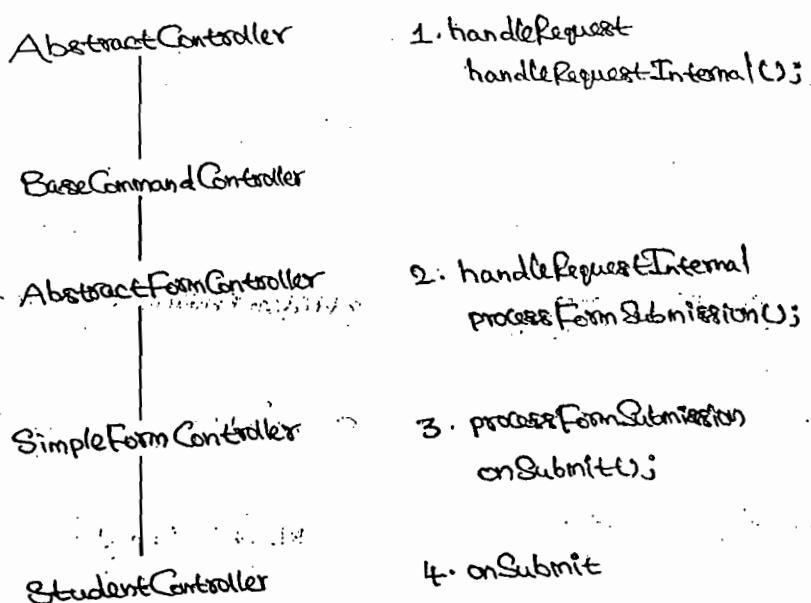
* If the request is coming from the Student.jsp, Then the method type will be post then it will submit the page.

* From Student.jsp, If the method type is get, Then the request never be submitted if we want to submit GET type of request also we need to override isFormSubmission() method.

SimpleFormController

* If we are working with SimpleFormController, we override onSubmit() method it will be called from the Super class processFormSubmission() method.

Workflow:



formView Property:

```
<property name = "formView">
    <value> student </value>
</property>
```

- * If the request is "Coming GET", then based on the formView value It is going to display the corresponding JSP.
- * If Any error comes, then what page it should be displayed that is decided by this value.

formView null Indicates what view to use, when the user asks for a new form or when validation errors have occurred on form submission.

SuccessView null Indicates what view to use, when the successful form submission have occurred. Such as success view could.

Eg: Display a submission summary

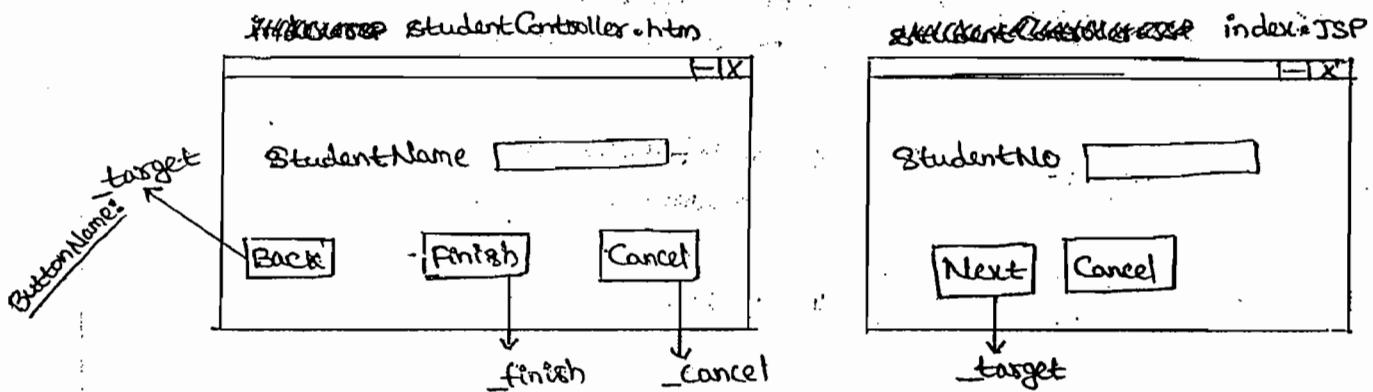
- * More sophisticated actions can be implemented by overriding one of the onSubmit() methods.

Session Form:

- * Indicates whether the form object should be kept in the session, when a user asks for a new form. This allows you.

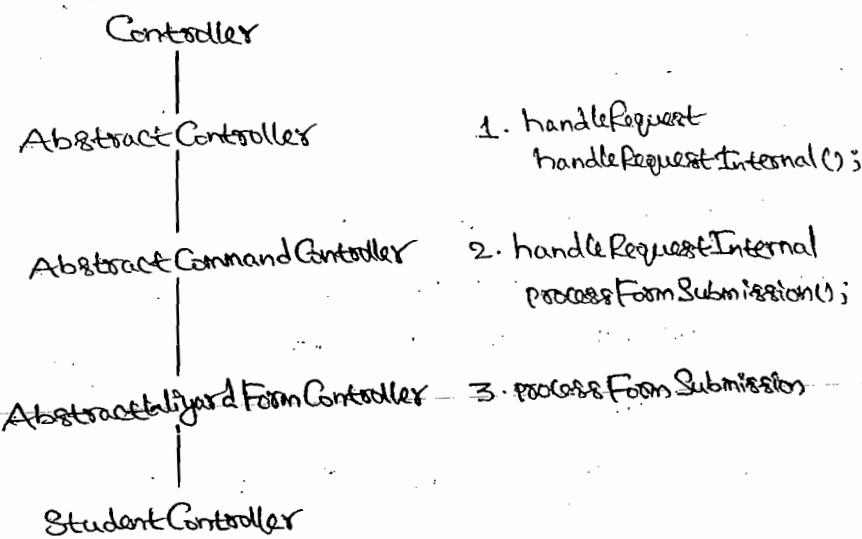
Eg: To retrieve an object from the Database, Let the user edit it and then persist it again. otherwise, a new command object will be created for each request. [Even when showing the form again after validation errors].

AbstractWizardFormController



- * If a page is having N No. of input values, then scrollbar will come.
So, Instead of keeping All the values in the same page, we can split the page into multiple pages.
- * The final submission to the Controller processFinish is done from the last page. From the other pages, the request is submitted to the controller. But
- * It won't process the request means it will not call processFinish() method.
- * It will call the processFormSubmission() method, then it display the next page (or) the previous page. Here also we need to inset command object and the Validator object to the controller internally it is done using BaseCommandController only.

Internal Flow:



- * If any request comes DispatcherServlet will execute handleRequest().
Note: handleRequestInternal will create command object, validate & process, then it will call processFormSubmission().

```

.... processFormSubmission(...)

  if (isPostionForm())
    request.getSession().removeAttribute("pageAfterName");

  // Cancel?
  if (isCancelRequest(request))
    return processCancel(...);

  // finish?
  if (isFinalPage(request))
  
```

```

// internal Submit : Validate current page and show specified target page.

if (!suppressValidation(...))
{
    ValidatePage(command, errors, currentPage, false);
}

int targetPage = getTargetPage(...);
return showPage(request, errors, targetPage);

private ModelAndView validatePagesAndFinish(...)
{
    // No remaining errors → proceed with finish.
    return processFinish(request, response, command, errors);
}

protected boolean isCancelRequest(...)
{
    return WebUtils.hasSubmitParameter(request, PARAM_CANCEL);
}

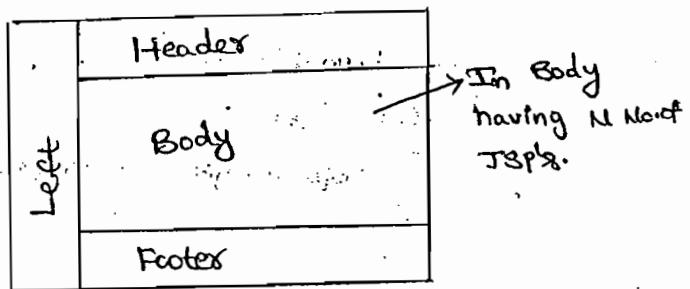
protected boolean isFinishRequest(...)
{
    return WebUtils.hasSubmitParameter(request, PARAM_FINISH);
}

public static final String PARAM_FINISH = "finish";
public static final String PARAM_CANCEL = "cancel";
public static final String PARAM_TARGET = "target";

public abstract class WebUtils
{
    public static boolean hasSubmitParameter(HttpServletRequest request, String name)
    {
        return request.getParameter(name + suffix);
    }
}

```

Steps To Integrate with Tiles [Working With Tiles] :



* We can implement the above design, using FrameSets in HTML

* We can implement the above design, using JSP: include tag.

* We can implement the above design, otherwise code duplication will be there, problem with JSP: include tags

is the JSP names, we need to write it in the JSP's only.

- * If we are implementing it using Tiles framework, there will be less configuration [Coding...etc] & If we are changing the JSP's It will not be effected to the other JSP's.

Steps To Configure:

ctrl+shift+F

1. Design the JSP's and Configure tiles-def.xml

- 1.1. header.jsp, leftNav.jsp, footer.jsp & content.jsp (Body.jsp)
the Basic Layout should be designed using layout.jsp.

- 1.2. Configure the basic layout in tiles-def.xml .

```
<tiles-definitions>
<definition name="template" page="/WEB-INF/tiles/Layout.jsp">
  <put name="title" value="Spring Tiles Example" />
  <put name="header" value="/WEB-INF/tiles/header.jsp" />
  <put name="footer" value="/WEB-INF/tiles/footer.jsp" />
  // Refer Material In Line No : 105
</definition>
<definition name="courseDetails" extends="template">
  <put name="content" value="/WEB-INF/tiles/courseDetails.jsp" />
  // Refer Material In Line No : 105
</definition>
</tiles-definitions>
```

* Using extends we are taking the basic template [It is like import or include]. In courseDetails, Content will be the new one and other [LeftNav, footer...] will be taken from base template.

- 1.3. Configure the basic layout.jsp and include leftNav, header

... JSP's
19-I

```
<tiles:insert name="header" />
<tiles:insert name="footer" />
```

2. Configure tiles-def.xml with TilesConfigurer and View should be TilesView.

2.1 Configure TilesConfigurer

```
<bean id="tilesConfigurer" class="TilesConfigurer">
```

factoryClass → I18nFactorySet

definitions → List → tiles-def.xml

2.2 View class should be TilesView

ViewResolver → InternalResourceViewResolver

viewClass → TilesView

2.3 Model And View ("courseDetails") → Controller class returned View should be configured in tiles-def.xml [with the corresponding JSP [View]]

Integrating Spring with Velocity:

* Velocity will be faster than JSP's [30% to 40%] Because the JSP will be converted to servlet means it will work with request and response objects.

* Velocity doesn't deal with request and response objects.

Steps To Configure:

1. Configure the velocity file location to VelocityConfigurer.

```
<bean id="velocityConfig" class="....VelocityConfigurer">
```

```
  <property name="resourceLoaderPath" value="WEB-INF/velocity/"/>
```

```
  </bean>
```

2. Configure ViewResolver as VelocityViewResolver, Internally it takes Velocity View.

```
<bean id="viewResolver" class="....VelocityViewResolver">
```

```
  <property name="suffix" value=".vm"/>
```

```
  </bean>
```

Note: Controller returns view name and Model object, In the vm use the model object to display the data [using Velocity tag library]

Note: To know the Classname if we don't know the jar file then
go to **ctrl+shift+T**

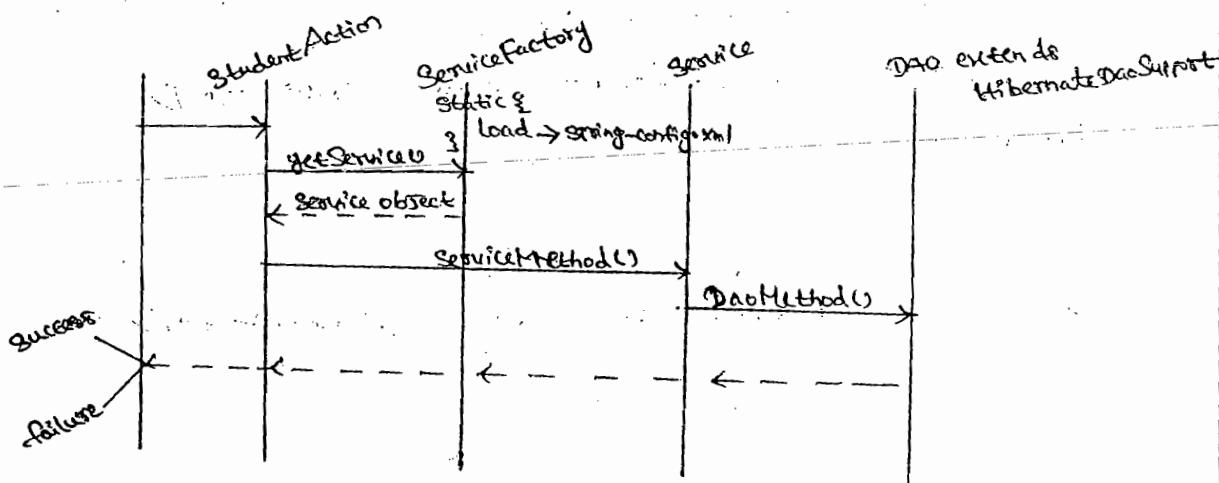
Generating pdf And Excel Sheets:

1. Extends AbstractPdfView (or) Extends AbstractExcelView and override buildPdfDocument() and buildExcelDocument().
use third party API iText (PDF) & POI (Excel) "jar files to generate the content".
2. The View class should be identified using beanNameViewResolver
Because here the view we are generating using a simple java Bean class

Integrating Struts With Spring:

* There are 4 ways, To integrate Struts with Spring.

1st Way:



* In the Above Architecture, we are integrating Spring with Struts.
But we are not using Any Spring provided API to integrate it

* To integrate Struts with Spring, we need to configure Spring provided contextLoaderPlugIn class in struts-config.xml.

* contextLoaderPlugIn will read Spring-configuration xml using init() method and it will create WebApplication Context object.

* If we are not providing Any Spring Configuration xml file, the Spring

```

<plug-in class="com.opensymphony.xwork2.plugin.ContextLoaderPlugin">
    <set-property property="contextConfigLocation"
        value="WEB-INF/actions-servlet.xml"/>
</plug-in>

public class ContextLoaderPlugin [Spring] implements Plugin [Struts]
{
    public static final String DEFAULT_NAMESPACE_SUFFIX = "-Servlet";
    public final void init(ActionServlet, moduleConfig)
    {
        this.webApplicationContext = initWebApplicationContext();
        actionServlet.getServletName() + DEFAULT_NAMESPACE_SUFFIX
            [should be Spring-config.xml]
    }
}

```

2nd Way:

struts-config.xml:

H-I

```

<action path="/studentAction" type="com.opensymphony.xwork2.DelegatingActionProxy"
    name="studentForm">
    <forward name="success" path="/success.jsp"/>
</action>

```

spring-config.xml:

H-I

```

<bean name="/studentActions" class="com.opensymphony.xwork2.DelegatingActionProxy"
    name="studentAction"/>

```

Advantage:

* We can inject Service & DAO objects.

* The other ways for Integrating Struts with Spring are using,

(3) ActionSupport

(4) DelegatingRequestProcessor

Note: In the second way of Configuration All the Actions or All the requests should be forwarded DelegatingActionProxy & It takes care of forwarding the request to the Actual Action class using Spring configuration file.

```
public class DelegatingActionProxy extends Action
{
    ... execute(...)

    {
        Action delegateAction = getDelegateAction(mapping);
        return delegateAction.execute(mapping, form, request, response);
    }
}
```

4/11/10
Monday
Way:

Steps To Integrate With Using DelegatingRequestProcessor:

struts-config.xml:

1. <controller processClass = "... DelegatingRequestProcessor" />
2. We don't configure Any ActionClassName , Here it should be configured in spring-config.xml file.
M-I
<action name = "StudentForm" path = "/StudentAction" />

spring-config.xml:

M-I
<bean name = "/StudentAction" class = "edu.actions.StudentAction" />
</bean>

* DelegatingRequestProcessor will control Creating the Action class object using Spring & Calling the Action class logic is done by the Struts.

```
public class DelegatingRequestProcessor extends RequestProcessor
{
    protected Action processActionCreate (request, response, mapping)
    {
        Action action = getDelegateAction(mapping);
        return super.processActionCreate (request, response, mapping);
    }
}
```

4th Way using Action Support:

- * All the Action classes has to extend ActionSupport instead of Action class
Using ActionSupport we can get Spring context object, using that we can get service object.

Steps To Configure:

1. public class StudentAction extends ActionSupport

{ execute(...)

ApplicationContext context = getWebApplicationContext();

Service service = (Service) context.getBean("service");

- * Here we don't inject service to action class.

Note: 2, 3, & 4th Examples, ContactLoaderPlugin should be configured in struts-config.xml.

- (*) * In projects, Delegating ActionProxy [2nd] OR 1st is the suggestible one.

Aspect Oriented Programming [AOP]

public class StudentService

{ public void insertStudent(Student student)

{ Session session = SessionUtil.openSession();

Transaction tx = session.beginTransaction(); // Before

StudentDao studentDao = DaoFactory.getStudentDao();

try

{ studentDao.insertStudent(student);

tx.commit(); // After

} catch (Exception exception)

{ tx.rollback(); // Catch

}

false

{ SessionUtil.close();

}

Before | After | Catch → Advice classes

```

class TransactionAdvice implements ...Advice
{
    ... Methods()
    {
        tx.beginTransaction();
        tx.commit();
        tx.rollback();
    }
}

```

```

public class StudentService
{
    public void insertStudent ( Student student )
    {
        studentDao.insertStudent ( student );
    }
}

```

* Using proxy object, we can configure Advice with Service [Target]...

AOP Overview

Cross-cutting Concerns:

- * Functionality whose implementation spans multiple modules
Eg: Logging, Transaction Management, Security, Auditing, Locking & Event Handler.
- * AOP is a programming methodology to help with cross-cutting concerns.

- (*) * Aspects can be Added or Removed as needed without changing your code.

Many AOP Implementations:

1. Java:

- * The Spring framework
- * JBoss AOP
- * AspectJ
- * Jakarta Hivemind
- * Many more...

2. .NET:

- * The Spring .NET Framework
- * Aspect.NET
- * ACA.NET

3. Many more for all other languages.

AOP In Spring

Join point: Model An Identifiable point in the execution of a program.

Advice: Model An Advice as an interceptor, Maintaining a chain of interceptors around the join point.

Pointcut: Spring uses the AspectJ pointcut language by default.

Aspect: Aspects are implemented using regular classes.

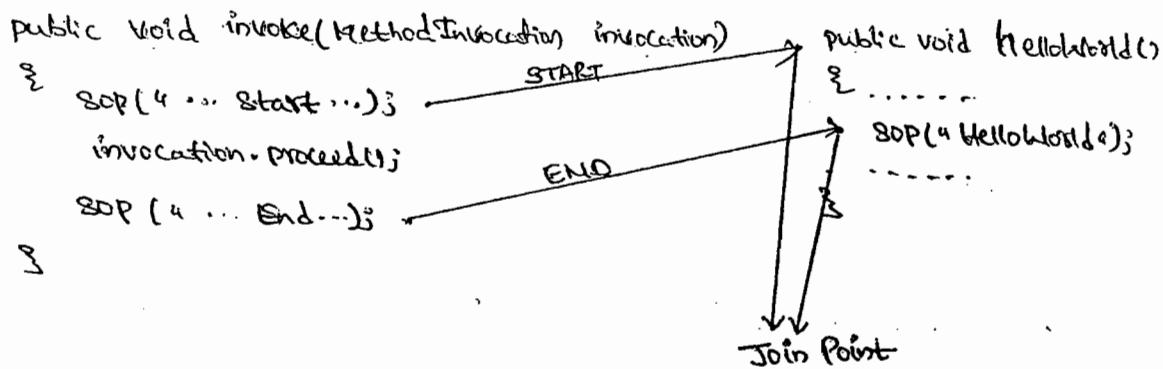
Weaving: Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime.

Target Object: Spring AOP is implemented using runtime proxies,
This object will always be a proxied object.

Introduction: Spring AOP allows you to introduce new interfaces
[and a corresponding implementation] to any proxied object

AOP Proxy: Object created by the AOP framework, Including Advice.
In Spring, An AOP Proxy will be a JDK Dynamic
Proxy or a CGLIB Proxy.

Advice



Join Point:

- * Well-defined point during the execution of your Application
- * You can insert Additional logic at Joinpoints.

Advice:

* The code that is executed at a particular Joinpoint.

Types of Advice:

1. before Advice, which executes before Joinpoint
2. after Advice, which executes after Joinpoint
3. around Advice, which executes around Joinpoint

Before Advice:

* Advice executes before a Joinpoint, 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 joinpoint completes normally.
Eg: 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 joinpoint exits (Normal or Exceptional return).

Around Advice:

* Advice that surrounds a joinpoint such as a Method Invocation.
This is the most powerful kind of Advice.

* Around Advice can perform Custom behaviour before and after the Method Invocation. It is also responsible for choosing whether to Proceed to the joinpoint (or) To Shorten the Advised method Execution by returning its own return value (or) throwing an exception.

Pointcuts:

- * A collection of joinpoints that you use to define when Advice should be executed.
- * By creating Pointcuts, You gain fine-grained control over how you apply Advice to the component.

Eg: 1. A typical joinpoint is a MethodInvocation.

2. A typical pointcut is a collection of All MethodInvocation in a particular class.

- * Pointcuts can be composed in complex relationships to further constrain when Advice is executed.

Aspect: An Aspect is the combination of Advice + Pointcuts.

$$\text{Aspect} = \text{Advice} + \text{Pointcuts}$$

Weaving:

- * process of actually inserting Aspects into the Application code at the appropriate point.

Types of weaving:

1. compile time weaving
2. Runtime weaving

Target:

- * An object whose execution flow is modified by some AOP process.
- * They are sometimes called Advised Object.

AOP Concepts: Introduction

- * process by which you can modify the structure of an object by introducing additional methods or fields to it.
- * You can use the introduction to make any object implement a specific interface without needing the object class to implement that interface explicitly.

5/1/10
Tuesday

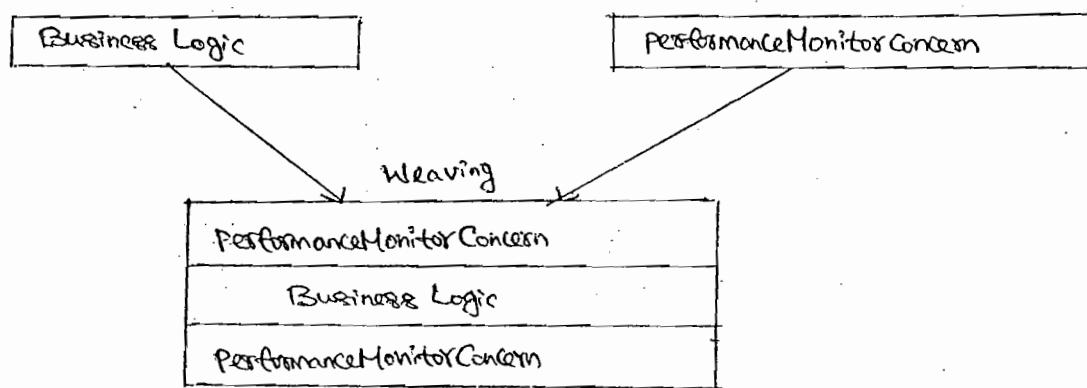
AOP In Spring

* Spring AOP has built-in Aspects such as providing transaction Management and performance monitoring.

Two options:

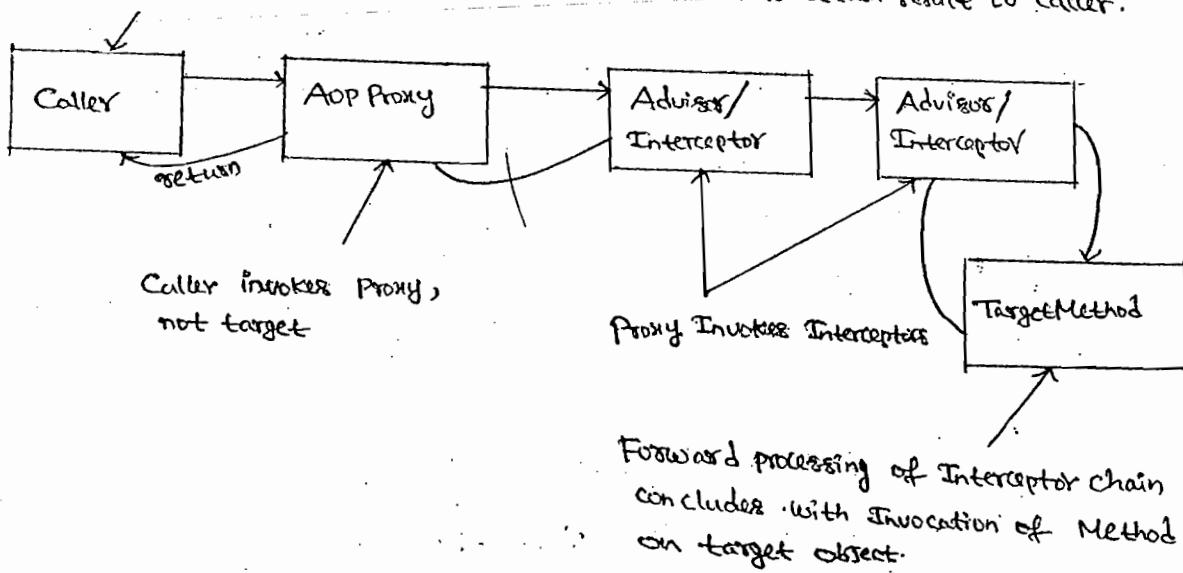
1. Spring AOP
2. AspectJ

IoC + AOP is a great combination that is non-invasive



AOP Control Flow

Control flows back through Interceptor chain to return result to caller.



- Advisor = pointcut + Advice

BeforeAdvice:

1. Implement the Business Layer [DAOImpl]
It is supposed to be service classes.
2. Implement MethodBeforeAdvice / AfterReturningAdvice / ThrowsAdvice / MethodInterceptor [AroundAdvice] and
override before / after / afterThrowing / invoke() methods ...
3. Implement Pointcut Logic [Implement Pointcut interface & override getClassFilter() and getMethodMatcher() methods]
getClassFilter(): It will decide on what class Advice should be Applied.

getMethodMatcher(): It will decide on what method Advice should be Applied.

(Note: If we are not configuring, then it takes the value as true.
Then on all Business Methods Advice will be Applied.

4. Main()

- * We need to take ProxyBusiness object, Then only at runtime the advices are going to Applied on Business object.
- * If we are takes Business object directly, then Advice never to be Applied on Business object. So, Always we need to take Proxy Business object only.

ProxyFactory proxyFactory = new ProxyFactory();

Advice advice = new LoggingMethodBeforeAdvice();

Advisor advisor = new DefaultPointcutAdvisor (Pointcut.TRUE, advice);

Advisor = Advice + pointcut

proxyFactory.setTarget (new DaoImpl());

ProxyFactory.addAdvisor (advisor);

```
Dao dao = (Dao) proxyFactory.getProxy();  
dao.daoMethod();  
dao.daoMethod();
```

(OR)

- * We can Advice directly to the proxy but internally it takes Adviser only. Means Always we are giving Advisor to the Proxy not the Advice..

```
ProxyFactory proxyFactory = new ProxyFactory();  
proxyFactory.setTarget(new DaoImpl());  
proxyFactory.addAdvice(new LoggingMethodBeforeAdvice());  
Dao dao = (Dao) proxyFactory.getProxy();  
dao.daoMethod();
```

Internal Code:

ProxyFactory:

```
public void addAdvice(Advice advice)  
{  
    addAdvice(POS, advice);  
}  
  
public void addAdvice(int pos, Advice advice)  
{  
    addAdvisor(pos, new DefaultPointcutAdvisor(advice));  
}
```

DefaultPointcutAdvisor:

```
public DefaultPointcutAdvisor(Advice advice)  
{  
    this(pointcut • TRUE, advice);  
}
```

XML Configuration:

1. Configure the Business object [Business Bean]
2. Configure the Advice object [Advice Bean]
3. Configure the proxy object [Proxy Bean]

bean	— ProxyFactoryBean
Property	— ProxyInterface — edu.dao.Dao
Property-target	— ref=...

Proxy Interfaces : Business Interface

-target : Business Object

* Advice we call it as Interceptor

Note: While Testing Throws Advice Pass true or false Value to the Business Method

Around Advice :

* From Invoke() method we need to call Business Method

method invocation.proceed(); // It will call Business Method

Around → Before + After Throws [Transaction Management internally uses Around Advice]

Pointcut :

NameMatchMethod Pointcut :

* Using this method, we can Configure the Business Method on those Business Methods only the Advice will be Applied on the other Business Methods. Advice is not going to be Applied. Means Pointcut will decide whether pointcut is Applied or not

```
<bean id = "nameMatchMethodPointcuts"
      class = "com.NameMatchMethodPointcut">
```

```
    <property name = "mappedName">
```

```
        <list>
```

```
            <value> daoMethod</value>
```

```
        </list>
```

```
    </property>
```

```
</bean>
```

* Here if there are N No of daoMethods, Advice will be Applied on daoMethod only.

```
<bean id = "defaultPointcutAdvisor" class = "DefaultPointcutAdvisor">  
    <property name = "advice">  
        <ref bean = "loggingMethodBeforeAdvice"/>  
    </property>  
    <property name = "pointcut">  
        <ref bean = "nameMatchMethodPointcut"/>  
    </property>  
</bean>
```

JdkRegexpMethodPointcut:

- * In JdkRegexpMethodPointcut we are going to give special character based on that Advice will be Applied on the Business Method.
 - If we give * on All the business methods Advice will be Applied
- Note: Check the Spring reference for the other special characters

```
<property name = "patterns">  
    <list>  
        <value>.*insert.*</value>  
        <value>.*doMethod.*</value>  
    </list>  
</property>
```

Note: The Above Two are Static Pointcuts & ControlFlowPoint is Dynamic Pointcut.

Annotation Based Configuration:

- * For the Advice class we need to give @Aspect Annotation and the Pointcut will be specified using @Pointcut using within we specify what is the Business class. Using @After we can specify ~~@~~ AfterAdvice.

④ Aspect

⑤ Pointcut ("within (edu.dao.DaoImpl);")

⑥ After ("edu.advice = DaoAfterReturningAdvice; daoMethod();")

⑦ Before

⑧ Around

⑨ AfterThrowing (pointcut = "within (edu.advice; DaoThrowsAdvice)")

Schema

Using SchemaBased Also we can Configure AOP:

<aop:config> //Root

<aop:aspect id = "daoAfterReturningAspect">

ref = "daoAfterReturningAdvice" //Advice

<aop:after-returning method = "afterAdvice">

pointcut = "within(edu.dao.DaoImpl)"/> //AfterAdvice

</aop:aspect>

//Business Object [Target]

</aop:config>

<aop:before ... />

<aop:around ... />

<aop:after-throwing ... />

expression

expression = execution >--> // It is like Regular Expression

* We specify point using pointcut and here we give pointcut and target object.

(*) Note: ContextLoaderListener is used to integrate Spring with JSF and it will read the Spring configuration file in the same way how the ContextLoaderPlugIn and DispatcherServlet is working.

(*) The End (*)

Kalyan IT Training Pvt., Ltd., Hyderabad

2ND FLOOR, SRINIVAS NAGAR (WEST), OPP. S.R.NAGAR BUS STOP, HYD - 38

T. Raghunatha

C & DS

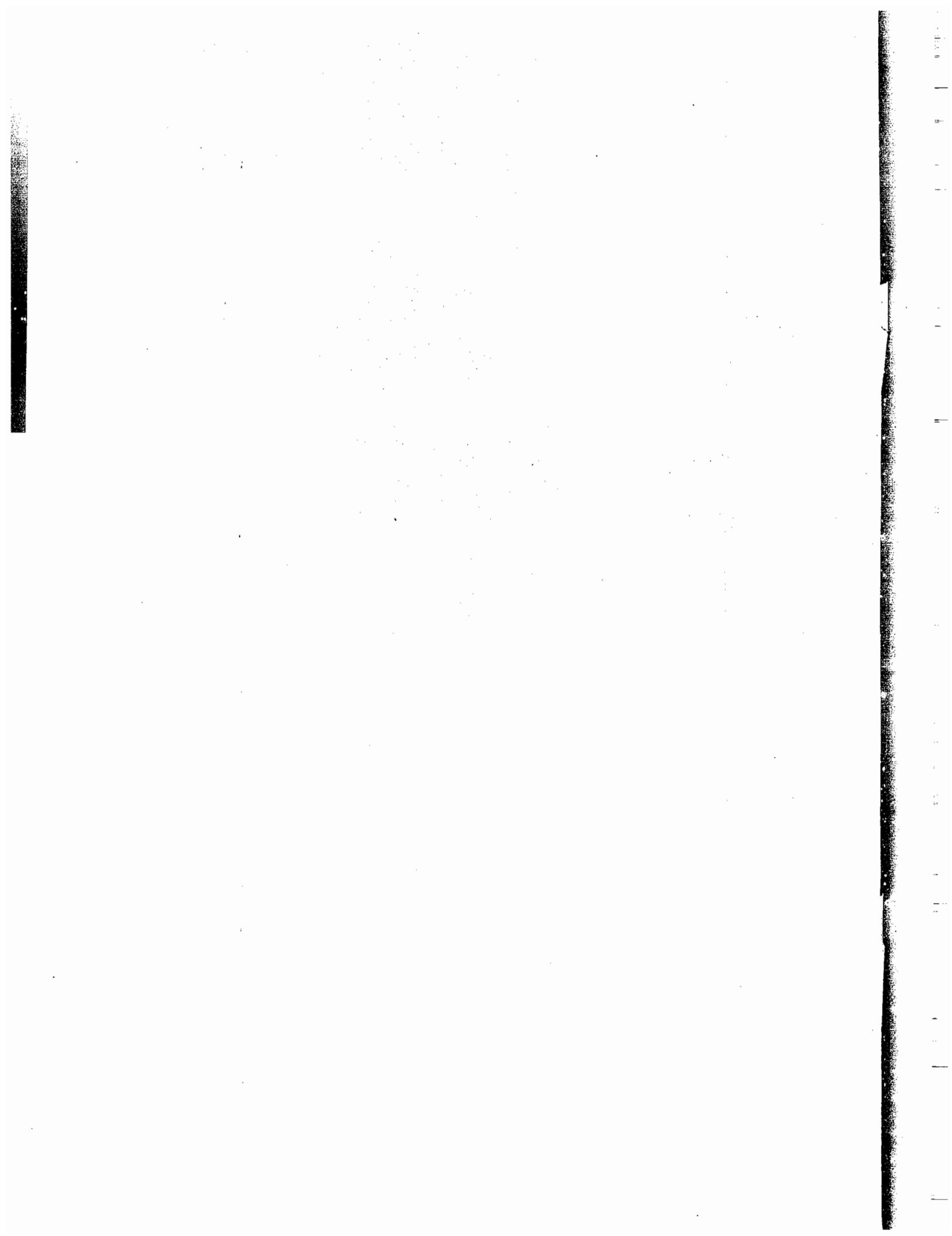
By

Mr. Kalyan Reddy

Kalyan

Ph: **6662 2789, 6662 3789**

www.kalyanit.com



C PROGRAMMING AND DATA STRUCTURES

UNIT - I

Algorithm / pseudo code, flowchart, program development steps, structure of C program, A Simple C program, identifiers, basic data types and sizes, Constants, variables, arithmetic, relational and logical operators, increment and decrement operators, conditional operator, bit-wise operators, assignment operators, expressions, type conversions, conditional expressions, precedence and order of evaluation. Input-output statements, statements and blocks, if and switch statements, loops- while, do-while and for statements, break, continue, goto and labels, programming examples.

UNIT - II

Designing structured programs, Functions, basics, parameter passing, storage classes- extern, auto, register, static, scope rules, block structure, user defined functions, standard library functions, recursive functions, header files, C preprocessor, example c programs.

UNIT - III

Arrays- concepts, declaration, definition, accessing elements, storing elements, arrays and functions, two dimensional and multi-dimensional arrays, applications of arrays. pointers-concepts, initialization of pointer variables, pointers and function arguments, address arithmetic, Character pointers and functions, pointers to pointers, pointers and multidimensional arrays, dynamic memory managements functions, command line arguments, c program examples.

UNIT - IV

Derived types- structures- declaration, definition and initialization of structures, accessing structures, nested structures, arrays of structures, structures and functions, pointers to structures, self referential structures, unions, typedef, bitfields, C program examples.

UNIT - V

Input and output – concept of a file, text files and binary files, streams, standard I/o, Formatted I/o, file I/O operations, error handling, C program examples.

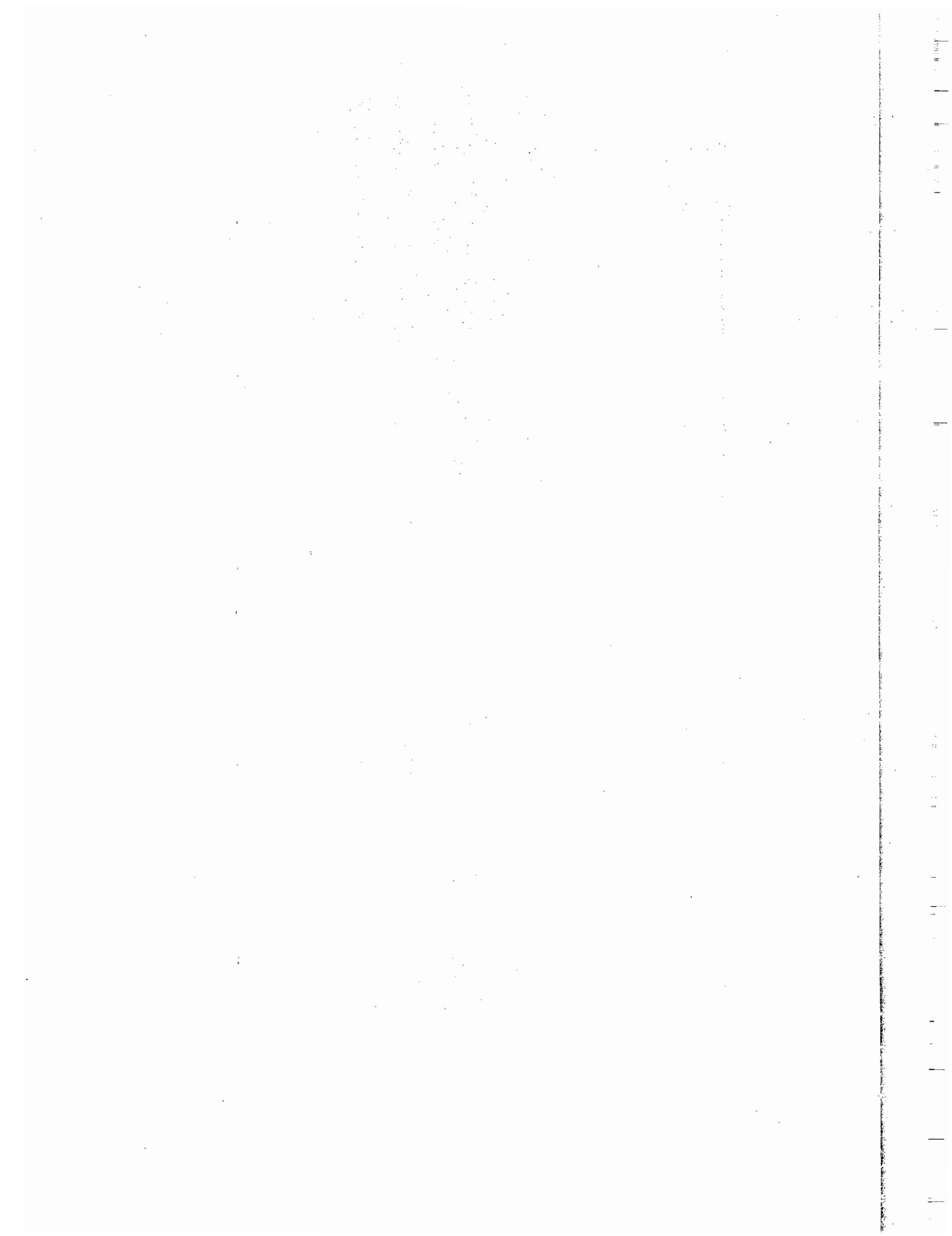
TEXT BOOKS :

1. Computer science, A structured programming approach using C, B.A. Forouzan and R.F. Gilberg Third edition, Thomson.
2. DataStructures Using C – A.S.Tanenbaum, Y. Langsam, and M.J. Augenstein, PHI/Pearson education.

REFERENCES :

1. C& Data structures – P. Padmanabham, B.S. Publications.
2. The C Programming Language, B.W. Kernighan, Dennis M.Ritchie, PHI/Pearson Education
3. C Programming with problem solving, J.A. Jones & K. Harrow, dreamtech Press
4. Programming in C – Stephen G. Kochan, III Edition, Pearson Education.

*www.kalyanit.com
ph: 6662 2789, 6662 3789*



Kalyan IT Training Pvt., Ltd., Hyderabad

2ND FLOOR, SRINIVAS NAGAR (WEST), OPP. S.R.NAGAR BUS STOP, HYD - 38

UNIT - I

C & DS

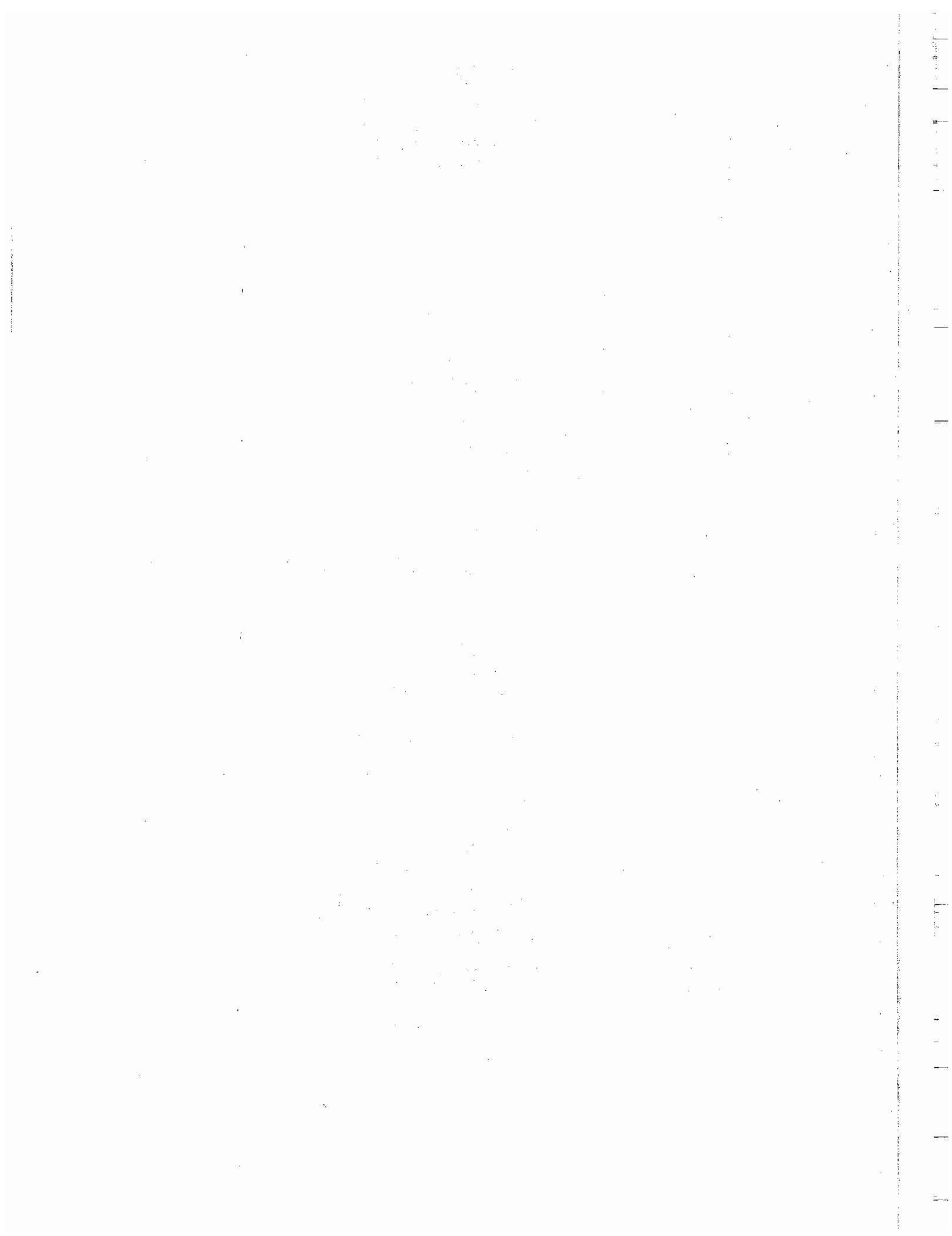
By

Mr. Kalyan Reddy

Kalyan

Ph: 6662 2789, 6662 3789

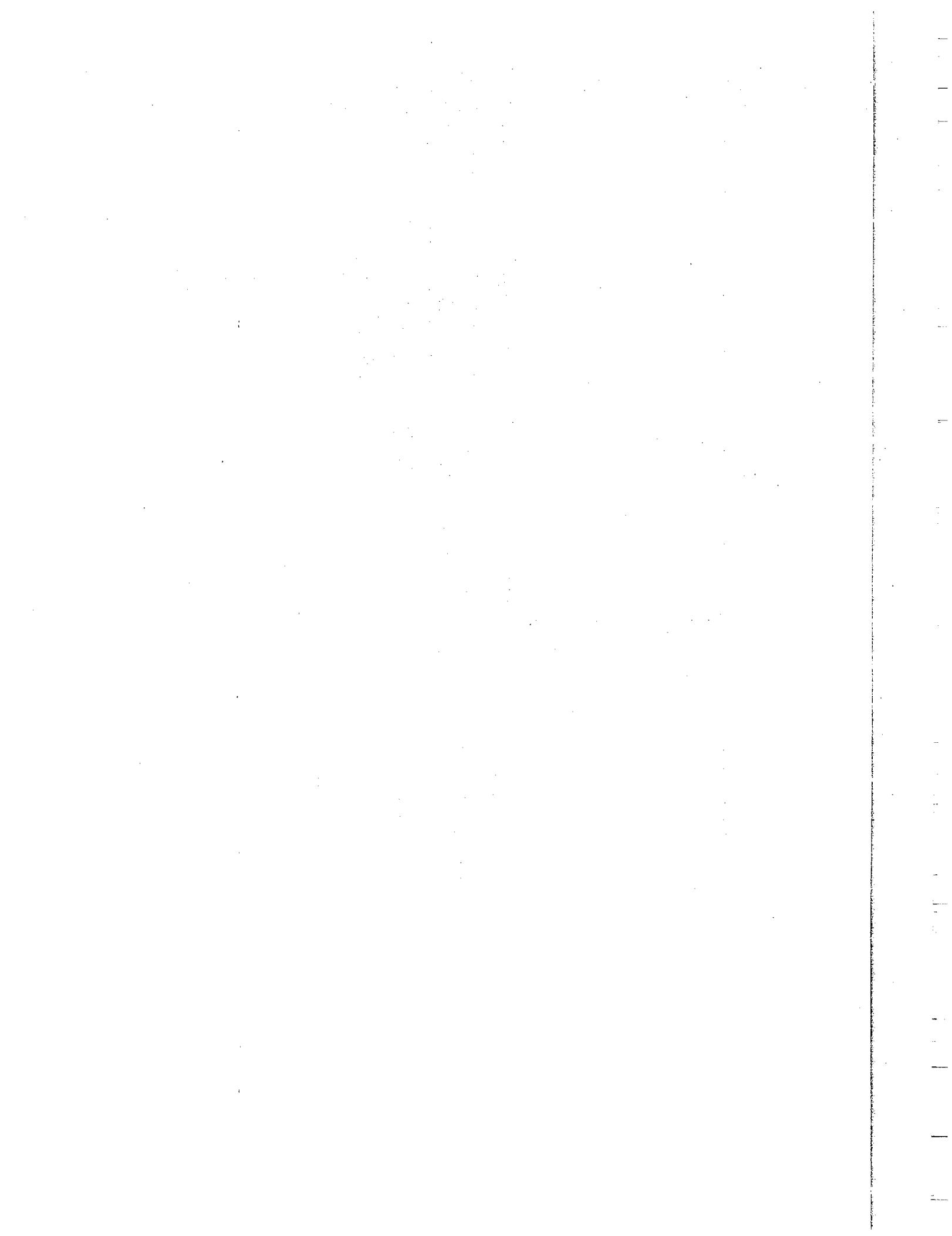
www.kalyanit.com



Kalyan IT Training (Hyd) Pvt Ltd

UNIT - 1

<u>Topic</u>	<u>Page No</u>
Algorithm	1
Flow chart	2
Program execution steps	3
Structure of C program	4
Characteristics of C	5
Token, Keywords	5
Character set, delimiters, identifiers, variables	6
Data types	7
Constants	9
Type conversion, operators	11
Expressions	17
I/Os, Statements	18
If – else	19
Conditional operator	22
Switch statement	23
Goto	24
Loops	25
Break – continue	30



Algorithm

An algorithm is a method of representing the step-by-step procedure for solving a problem using a pseudo code. An algorithm is defined as a finite set of steps that provide a chain of actions for solving a definite nature of problem.

Pseudo-code

Pseudo code is an artificial and informal language that helps programmers develops algorithms. Algorithms written in English like language are called pseudo-code.

Ex: Average of three numbers.

- a. Read a,b,c.
- b. Sum a,b,c.
- c. Divide sum by 3.
- d. Store result in d.
- e. Print d.
- f. End program.

Steps in algorithm: steps in algorithm can be divided into three categories.

- a. Sequence
- b. Selection
- c. Iteration

- a. **Sequence:** The steps described in an algorithm are performed successively one by one without skipping any step. The sequence of steps defined in an algorithm should be simple and easy to understand. Each instruction of such an algorithm is executed, because no selection procedure or conditional branching exists in a sequence algorithm.

Ex:

1. Dial the phone number
2. Phone rings at the called party.
3. Caller waits for response.
4. Called party picks up the phone.
5. Conversation begins between them.
6. Release of connection.

- b. **Selection:** the disadvantage with sequence algorithm is not reliable. There must be a procedure to handle operation failure occurring during execution. The selection statement can be as:

```
if (condition)
statement 1;
else
statement2;
```

Ex:

1. Dial phone number.
If (busy tone)
Goto step1
2. Phone rings at the called party.
3. Caller waits for response.

4. Called party picks up the phone.
5. Conversation begins between them.
6. Release of connection.

The **selection** block handles the situation and run-time failures can be avoided.

C. Iteration: In a program sometimes it is very necessary to perform the same action for a number of times. If the same statement is written repetitively, it will increase the program code. To avoid this problem, iteration mechanism is applied. The statement written in an iteration block is executed for a given number of times based on certain condition.

For	while	do-while
for (Expression1; Expression2; Expression3) Statement;	while Expression1; while (Expression2) { Statement; Expression 3; }	do Expression1; do { Statement; Expression 3; } while (Expression 2);

Properties:

1. Finiteness
2. Definiteness
3. Effectiveness
4. Generality
5. Input/Output

Finiteness: An algorithm should terminate in a finite no. of steps.

Definiteness: each step of algorithm must be precisely stated.

Effectiveness: each step should be easily convertible into program statement and can be performed exactly in a finite amount of time.

Generality: Algorithm should be complete in itself so that it can be used to solve all problems of a given type for any input data.

I/O: Each algorithm must take zero, one or more quantities as input data and yield one or more output values.

Flow Chart:

Flow chart is diagrammatic representation of an algorithm and built using different types of boxes and symbols. A flow chart is a visual representation of the sequence of steps for solving a problem. A flowchart is an alternative technique for solving a problem. Once an algorithm is written, its pictorial representation can be done using flowchart symbols. In other words, a pictorial representation of a textual algorithm is done using a flowchart.

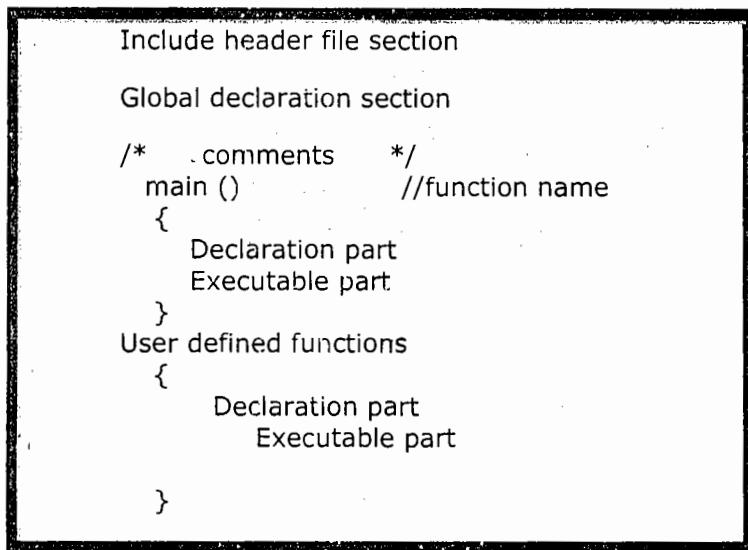
Oval		Terminal	Start/stop/begin/end.
Parallelogram		Input/output	Making data available for processing or recording of the processed information.
Document		Print out	Show data output in the form of document.
Rectangle		Process	Any processing to be done. A process changes or moves data. An assignment operation.
Diamond		Decision	Decision or switching type of operations.
Circle		Connector	Used to connect parts of flow chart.
Arrow		Flow	Joins two symbols and also represents flow of execution.

Various steps in C program execution:

- a. Editing
 - b. Compiling
 - c. Linking
 - d. Execution
- a. **Editing:** In the editing phase, the C program is entered into a file through a text editor. UNIX OS provides text editor 'vi' editor, modern compiler vendors provide IDEs which consist of both editor and compilers. The file is saved on the disk with .c extension. Corrections in program at later stages are done through these editors. Once program has been written, it requires to be translated into machine language.
- b. **Compiling:** This phase is carried out by compiler. Compiler translates the source code into the object code i.e. machine understandable code. The compilation phase cannot be proceeded successfully until and unless the source code is error-free. The compiler generates messages if it encounters syntactic errors in the source code. The error free source code is translated into object code and stored in a separate file with an extension '.obj'.
- c. **Linking:** The linker links all the files and functions with the object code under execution. For example, if a user uses a 'printf' function in the program, the linker links the user programs object code with the object code of the printf function which is in the stdio.h library.
- d. **Execution:** here, the executable object code is loaded into the memory and program execution begins. We may encounter errors during the execution phase

even though compilation phase is successful the errors may be run-time errors and logical errors.

Structure of 'C' Program:



- a. **Include header file section:** a C program depends upon some header files for function definition that are used in the program. Each header file by default has a extension '.h'. The file should be included using #include directive. **Ex:** #include <stdio.h>
- b. **Global declaration:** this section declares some variables that are used in more than one function. These variables are known as global variables. This section must be declared outside of all the functions.
- c. **Function main():** every program written in C language must contain the **main()** function. Empty parentheses after main are necessary. The function main() is the starting point of every 'C' program. The execution of the program always begins with the function main(). Except the main() function, other sections may not be necessary. The program execution starts from the opening brace ({) and ends with the closing brace (}). Between these two braces, the programmer should declare declarations and executable parts.
- d. **Declaration part:** the declaration part declares all variables that are used in the executable part. Initialization of variables is also done in this section. Initialization means providing initial value to the variables.
- e. **Executable part:** this part contains the statements following the declaration of the variables. This part contains set of statements or a single statement. These statements are enclosed between the braces.
- f. **User-defined function:** the functions defined by the user are called user-defined functions. These functions are generally defined after the main() function. They can also be defined before the main() function. This portion is not compulsory.
- g. **Comments:** comments are not necessary in the program. However, to understand the flow of the program, programmer can include comments in the program. Comments are to be inserted by the programmer. These are useful for documentation.

Characteristics of 'C':

1. **Generality:** C language is general-purpose language. It is used to write programs varying from simple to complex. Nowadays, it is used in developing applications which are effectively used in academic campus, multinational companies etc.
2. **Middle level language:** C language possess the capabilities of both low level languages and high level languages. Therefore, C can be used for writing system software's and application software's.
3. **Extensive library functions:** C language has huge set of built in library functions making it a robust language. Any complex programs can be easily written using these built-in functions.
4. **Efficiency and speed:** C language has rich set of data types and operators making the language more efficient and fast. The language provides some operators (increment, decrement) which speed up the execution to large extent. Further, it also provides user defined data types (Ex: structures) through which miscellaneous data can be easily manipulated.
5. **Portability:** C language is portable. It means that the programmes written on one machine can be executed on different machine with or without minor changes in the program.
6. **Structured programming:** a C program can be defined as a collection of function modules (i.e., performing a unit of task) and blocks. It also provides basic control flow statements which are essential for structured programming.

Simple program: The First C program

```
void main( )
{
    printf("WELCOME TO KALYAN IT");
}
```

Learning C is similar and easier. Instead of straight-away learning how to write programs, we must know keywords, operators, separators, constants, predefine functions and syntax of C language.

Token: the smallest individual units in C program are known as C tokens. C has 6 types of tokens: **Keywords, constants, identifiers, strings, special symbols** and **operators**. C programs are written using these tokens and syntax of the language.

1. **Keywords:** Keyword is the word whose meaning has already been explained to the C compiler. In ANSI standardizes C language total number of keywords are 32.

Ex: while, if, else, case, break, etc.

Auto	double	int	struct
Break	else	long	switch
Case	enum	register	typedef
Char	extern	return	union
Const	float	short	unsigned
Continue	for	signed	void

Default	goto	sizeof	volatile
Do	If	static	while

2. **Character set:** the characters used to form words, numbers and expressions depend upon the computer on which the program runs. These are classified into 4 categories.

1) Letters	Capital A to Z Small a to z
2) Digits	All decimal digits 0 to 9
3) White spaces	Blank space, horizontal tab, vertical tab,
4) Special characters	New line, form feed , ; : ` " ! / \ ~ _ \$? & ^ * - + < > () [] { } % # = @

3. **Delimiters:** These are special kind of symbols, which are called as delimiters. In general terms these delimiters known as **separators**.

Delimiters	Use
:	Useful for label
;	Terminates statement
()	Used in expression and function.
[]	Used for array declaration
{}	Scope of statement
#	Pre-processor directive
,	Variable separator

4. **Constants:** the constants in C are applicable to the values, which do not change during the execution of a program. Constants are classified in to two types.

a. **Numeric constants**

1. Integer constants: Ex: 10, +30, -15 etc.
2. Real constants: Ex: 2.5, 5.521, 3.14 etc.

b. **Character constants**

1. Single character constants: Ex: 'a', '8', ' ' etc. (Alpha numeric)
2. String constants: Ex: "Hello", "444", "a" etc.

5. **Identifiers:** Identifiers are names of variables, functions, and arrays. They are user-defined names, consisting of sequence of letters and digits, with the letter as the first character. Lower case letters are preferred. However, the upper case letters are also permitted. The (_) under score symbol can be used as an identifier. In general underscore is used as a link between two words in long identifiers. **Ex:** total_salary

6. **Variables:** a variable is a data name used for storing a data value. Its value may be changed during the program execution. The variables value keeps on changing during the execution of a program. In other words, a variable can be assigned

different values at different times during the execution of a program. **Ex:** height, average, sum etc.

7. **Data types:** A data type is essential to identify the storage representation and the type of operations that can be performed on that data. In general, data type associated with variable indicates,

- Type of value stored in the variable.
- Amount of memory (size) allocated for data variable.
- Type of operations that can be performed on the variable.

'C' supports four basic data type which can be grouped under two groups of data types.

1. integral data type
2. Floating point data type.

1. Integral Data Type: Integral Data type is used to stored whole numbers and characters. It can be further classification as,

- a. Integer data type
- b. Character data type.

a. Integer data type: This data type is used to store the whole numbers. The integers do not contain the decimal points. They take the binary form for storage. The size of the integer depended on the word length of a machine i.e., it can be either 16 or 32 bits. On a 16-bit machine. The range of integer value is -32768 to +32767.

'C' provides control over the range of integer value and the storage space occupied by these values through the data type 'short int', 'int', 'long int' in both 'signed' and 'unsigned' form.

signed integer: signed integers are those integers which uses 15 bits for storing the magnitude of a number and 1 bit to store its sign. The left-most bit i.e., 16th bit used the sign, it is 1 for negative number and '0' for the positive number.

Unsigned integer: unsigned integer use all 16 bit i.e., (15+1) bits to store the magnitude i.e., these are the whole numbers that always hold positive values and the sign bit also contains the value.

b. Character Data Type: The data type indicates that value stored in the associated variable is a character. Each character has an equivalent ASCII value. These characters are internally stored as integers.

'char' data type occupies one byte of memory for the storage of character. The qualifiers signed and unsigned can be applied on 'char' data type.

character Data Type			
Type	Size	Range	Format specifier
char (or) signed char	1	-128 to +127	%c
unsigned char	1	0 to 255	%c

2. Floating-point Data type: Floating point data type is used to store real numbers. The keyword 'float' is used to declare a variable holding a floating-point numbers with 6 digit of precision occupies 4 byte for storage. For greater precision we can use the 'double' data type for further precision, we have long double.

DATA Type qualifiers: A qualifier is an adjective describing the data type. They are used in combination with the basic data types. This qualifier along with data types. These qualifiers along with data types can be used to have control over storage space and range of numbers.

'C' language provides a set of qualifiers as given below,

- a. short
- b. long
- c. signed
- d. unsigned

Short: this qualifier can be used only with the data type int.

Syntax: short int variable;

'short int' is used for small ranger values (-128 to +127) and it occupies 50% of storage space (1 byte) as compared to regular int (2 bytes).

long: this qualifiers is used along with the data types 'int' and'double'. This qualifier is used to increase the range of values, thus consuming more storage space.

Syntax: long data-type variable;

- The qualifier 'long' when associated with 'int' data type takes a range of integer values and occupies 32 bits (4 byte) for storage.
- The qualifier 'long' when associated with 'double' data type takes a range of floating point values and occupies 80 bits (10 byte) for storage.

signed: the qualifier can be used with 'int' and 'char' data types. This qualifier effects the internal representation of the data value.

When signed qualifier is used with the basic data type the starting bit of storage gets reserved for the sign.

unsigned: this qualifier can be applied with 'int' and 'char' data type. This qualifier when used, the total bits get reserved for the magnitude, thus the range of values gets increased.

Constants: A quantity or number that does not vary during the execution of program is known as constants. These constants are classified into two types

1. numeric constants
2. character constants

1. Numeric constants: Numeric constants can be classified as,

- a. integer constants
- b. real constants

a. **Integer constants:** The integer constants are confined to the following features.

- o It contains minimum of one digit.
 - o It does not contain a decimal point.
 - o Blank spaces and commas are not allowed within the integer constant.
 - o An integer constant can be either positive or negative value.
- Ex: 400, +789, -100, etc.

The range of integer constants depending on the machine. 'C' provides a range of -32768 to 32767 integer constants for 16 bit word length machine and -1247483648 to 2147483647 for 32 bit.

a long integer, unsigned integer and unsigned long integer constant can be represented by appending the l or L, u or U or UL respectively.

Ex: long integer constants: 98491560l, 98491560L

unsigned integer constants: 55596U, 55596u

unsigned long integer constants: 955713345I, 955713345L

An integer constant can be represented in any one of the three numbers system i.e. decimal, octal, hexadecimal.

Decimal integers: these constant consists of numbers from 0 to 9 preceded by optional sign. Ex: +129, -95, 145

An octal integer: these constant consists of any digits from 0 to 7 with leading zero ('0').

Ex: 037, 067, 050

Hexa-decimal: these integer constants consist of combination of digits from 0 to 9 and alphabets, i.e.a-f or A-F. The alphabets a to f represent numbers 10 to 15 respectively. Hexadecimal numbers are preceded by 0x or 0X.

Ex: 0xA5, 0XA5, 0xa6c

b. Real constants: Real constants are the quantities which contains fractional part i.e., decimal point. Real Constants are also known as floating-point constant. Real constants can be represented in two ways.

- i. Decimal (fractional) form
- ii. Exponential (scientific) form

Decimal form: The real constants represented in decimal form are confined to the following rules.

- The constant must contain a whole number followed by decimal point and fraction part.
- It can be either +ve or -ve.
- blank spaces and commas are not with in the real constant
- The digits before and after the decimal point can be omitted

Ex: 110.1,+305.25,-400.50,-.10.

Exponential form: Exponential form is used to represented the real constants both either too large or too small value.

Ex: 8500000000 can be represented as 8.5e9.
0.00000786 can be represented as 7.86e-7.

The real constant represented in exponential form consist of two part, mantissa (i.e. part appearing before the latter e or E) and exponent (i.e. part appearing after e or E). A real constant represented in exponential form is confined to the following rule

- The letter e or E must separate the mantissa and exponential part.
- mantissa part could be either a real number or an integer .mantissa part could be either +ve or -ve (default is +ve)
- Exponential part must be contain at least one digit, which could be either a +ve or -ve integer.
- The real constant can be represented as float (i.e. single -precision) and long double (i.e. extend double -precision) by appending the terminals 'f' or 'F' and 'l' or 'L' respectively (note: default is double-precision).

2. Character Constants: The character constants can be divided into two types,

- a. single character constants
- b. string constants

Character constants: A single character constant is a single digits, alphabet or special symbol, enclosed within single quotes.

Ex: 'a', '1', '+'

A character constant represents numeric value. (i.e., ASCII value of the character). A character constant is therefore arithmetic operations.

Maximum of one character is taken within a character is taken within a character constants.

String constants: A string constant contains group of character (i.e., alphabets, digits, special symbols and blank spaces)
Ex: "welcome", "+,-" "10+2".

8. **Type conversion:** sometimes programmer needs the result in certain data type. Example, division of 5 with 2 should return float value. Practically compiler always returns integer values because both the arguments are of integer data type. In those cases type conversion is required.

Ex: `printf ("\ntwo integers (5&2) : %d", 5/2); → 2`
`printf ("\none int and one float (5.5&2) : %f", 5.5/2); → 2.75`
`printf ("\ntwo integers (5&2) : %f", (float) 5/2); → 2.5`

9. **Operators:** Operators are 'C' tokens which can joined together individual constants; variables array elements and function references. Operators act upon data items called as operands.

Classification:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operator
5. Increment and decrement operator
6. Conditional operator
7. Bitwise operator
8. Special operator

1. Arithmetic Operators

Arithmetic operator	Meaning
+	Adding or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

- For arithmetic operations, operands must be numeric values.
- Here modulo division produces remainder of integer division.

- arithmetic operations are classified as,
 - integer arithmetic
 - real arithmetic
 - mixed mode arithmetic

Integer arithmetic: both the operands are integer in integer arithmetic. It always yields an integer value.

Ex: int a=10,b=5;
 $a+b, a-b, a*b, a/b, a\%b$ etc.

Real arithmetic: both the operands are real in real arithmetic. It yields real values as result. It cannot be applied to % operator.

Ex: float a=10.0, b=5.0;
 $a+b=10.0+5.0=15.0$ i.e. 15.000000
 $a-b=10.0-5.0=5.0$ i.e. 5.000000
 $a*b=10.0*5.0=50.0$ i.e. 50.000000
 $a/b=10.0/5.0=2.0$ i.e. 2.000000
 $a\%b=10.0\%5.0$ (invalid expression)

Mixed Mode Arithmetic: when one operand is integer and other is real it is known as mixed mode arithmetic. Here the result will always be real.

Ex: int a=10;
 float b=5.0;
 $a+b=10+5.0=15.0$
 $a-b=10-5.0=5.0$
 $a*b=10*5.0=50.0$
 $a/b=10/5.0=2.0$
 $a\%b=10\%5.0$ (invalid expression)

2. Relational Operators: These are used to compare two quantities.

- These operators always returns 1(true) or 0 (false).
- if the expression is true than it returns with 1 else 0

relational Operator	action
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Expression	Result
10<5	0 (false)
10>5	1 (true)
10==15	0 (false)
10!=15	1 (true)
(10+5)==(3*5)	1 (true)

3. Logical Operators

logical Operator	action
&&	logical AND
	logical OR
!	logical NOT

- Logical expression combines two or more relational expressions.
- Logical operators are used to test more than one condition and make decision.

Logical AND: the result of logical AND expression will be true only when both the relational expression are true. Syntax: exp1 && exp2;

Logical OR: the result of logical OR expression will be false only when both relational expressions are false. Syntax: exp1 || exp2;

Logical NOT: the result of the expression will be true if the expression is false and vice versa. Syntax: !exp1;

Ex:

```
int a=10,b=15, c=20,i;
i= (a>b) && (b<c);
```

The value of i in this expression will be 0.

```
i= (a<b) && (b<c);
```

The value of i in this expression will be 1.

```
i= (a>b) || (b<c);
```

The value of i in this expression will be 1.

```
i= ! (a>b);
```

The value of i in this expression will be 1.

4. Assignment Operators

The operators are used to assign result of an expression to a variable

Syntax: Variable = exp;

Assignment Operator	Action
int a,b; a=10;b=5; a=a+1; or a+=1;	
	adds 1 to a and assigns the value to a
a=a-1; or a-=1;	decrements a by 1 and assigns the value to a
a=a/(b+1) ora/=b+1;	divides a by (b+1) and assigns result to a
a*=b+2; or a=a*(b+2)	multiplies (b+2) with a and assigns result to a

5. Increment and Decrement operators

These operators are represented as '++' and '--'. ++ Increments operand by 1 and -- decrement the value by 1. These are the unary operators and takes the following forms,

Operator	Action	Result
int a,b; a=1; b=++a;	pre- increment	increments a value by 1 and assigns the value to b. i.e., a=2 and b=2
b=a++;	post- increment	Assigns the value to b then increments a value by 1. i.e., a=2 and b=1

a=2;		
b=-a;	pre- decrement	decrements a value by 1 and assigns the value to b. i.e., a=1 and b=1
b=a--;	post- decrement	Assigns the value to b then decrements a value by 1. i.e., a=1 and b=2

6. Conditional Operator: It is also known as ternary operator

Syntax: exp1 ? exp2 : exp3;

Where exp1, exp2 and exp3 are expressions

Ex:

```
int a=5,b=10,c=15,y;
y=(a>b) ? b : c;
The value of y is 15
y=(a<b) ? b : c;
The value of y is 10
```

Greatest of three numbers using conditional operation.

y=(a>b && a>c) ? a: (b>c) ? b: c;

Here greatest of three numbers is stored in x;

7. Bitwise Operators

Bitwise operators are similar to that of logical operators except that they work on binary bits. When bitwise operators are used with variables, they are internally converted to binary numbers and then bitwise operators are applied on individual bits.

Bitwise operators do manipulation on bit stored in memory. These operators work with char and int type only. They cannot use with floating point numbers.

Bitwise Operators	Action
\sim	1's compliment
$ $	Bitwise OR
\wedge	Bitwise exclusive OR
$\&$	Bitwise AND
$<<$	Left shift
$>>$	Right shift

Ex:

int a=10,b=8;

The equivalent binary value of a is 0000 1010

The equivalent binary value of b is 0000 1000

1. the value of a & b= 0000 1000 i.e.,8
2. the value of a | b = 0000 1010 i.e., 10
3. the value of a \wedge b= 0000 0010 i.e., 2
4. a<<2. It left shift the binary bits twice. i.e.,0010 1000
5. a>>2. It right shifts the binary bits twice. i.e., 0010

8. Special Operator

There are some special operators available in 'C' such as sizeof operator and member selection operators (. and \square).

sizeof Operator: It returns numbers of bytes the operand occupies; operand may be a variable, a constant, or a data type qualifier.

Syntax: sizeof(operand)

Ex:

int y;

y= sizeof (int); here value of y will be 2 byte.

Member Operators: These are used to access members of structure and unions.

Operators priority wise:

Operators	Operation	Clubbing	Priority
()	Function call		
[]	Square bracket	Left to Right	1 st
->	Structure operator		
.	Structure operator		
+	Unary plus		
-	Unary minus		
++	Increment		
--	Decrement		
!	Not operator	Right to Left	2 nd
~	Ones complement		
*	Pointer operator		
&	Address operator		
sizeof	Size of an object		
type	Type case		
*	Multiplication		
/	Division	Left to Right	3 rd
%	Modular division		
+	Addition	Left to Right	4 th
-	Subtraction		
<<	Left shift	Left to Right	5 th
>>	Right shift		
<	Less than		
<=	Less than or equal to		
>	Greater than	Left to Right	6 th
>=	Greater than or equal to		
==	Equality	Left to right	7 th
!=	Inequality		
&	Bitwise AND	Left to right	8 th
^	Bitwise XOR	Left to right	9 th
	Bitwise OR	Left to right	10 th
&&	Logical AND	Left to right	11 th
	Logical OR	Left to right	12 th
? :	Conditional operator	Right to left	13 th
=, +=, -=, *=, /=, %=&, &=, ^=, =, <<=, >>=	Assignment operators	Right to left	14 th

10. Expressions:

An expression in C is constructed using operators, variables and /or constants.

Ex: $x+y*3.5$

$X < y$

$X < y \& \& x > 5$

11. **I/Os:** Reading the data from the input devices and displaying the results on the screen are the two main tasks of any program. When a program needs data, it takes the data through the input functions and sends results obtained through the output functions. Thus, the input/output functions are the link between the user and the terminal. The I/O functions are classified into two types:

- a. Formatted functions
 - b. Unformatted functions
- a. **Formatted functions:** the formatted I/O functions read and write all types of data values. They require conversion symbol to identify the data type. Hence, they can be used for both reading and writing of all data values. The formatted functions return the values after execution. The return value is equal to number of variables successfully read/written. Using this value the user can find out the errors occurring during reading or writing the data.
Ex: printf(), scanf();
- b. **Unformatted functions:** the unformatted I/O only works with the character data type. They do not require conversion symbol for identification of data types because they work only with character data type. There is no need to convert the data. In case values of other data types are passed to these functions, they are treated as the character data. The unformatted functions also return values, but the return value of unformatted function is always same.
- Types:
- (a). **Character I/O.**
Ex: getchar(), putchar(), getch() & getche(), putch().
 - (b). **String I/O.**
Ex: gets(), puts(), cgets(), cputs().
 - (c). **File I/O.**

12. **Statements:** a statement is an instruction that causes an action to be performed when executed. The C statements ends with ';'. In C there are 6 types of statements.

- a. **Selection statement:** selection statements are statements that are executed depending on the condition. These include **if** and **switch**.
- b. **Iteration statement:** these are executed repeatedly until a condition is satisfied. These include **for, while, do-while**.
- c. **Jump statements:** jump statements alter the flow of execution. These are **break, continue, goto** and **return**.
- d. **Label statements:** label statement is used to give the location of the target statement to which control must transfer. **Ex: case, default and label.**
- e. **Expression statement:** the statements composed of valid expression are expression statements.
- f. **Block statement:** a block consists of a set of statements enclosed within flower braces. These are also called **compound statements**.

13. **Decision control instructions in C:** We have seen that a C program is a set of statements which are normally executed sequentially in order in which they appear. By default, the instructions in a program are executed sequentially. In serious programming situations, we want a set of instructions to be executed in one situation, and different set of instructions to be executed in another situation. This kind of situation is dealt with in C programs using a decision control instruction. A decision control instruction can be implemented in C using:

- a. if statement.
- b. conditional operator statement.
- c. switch statement.
- d. goto statement.

A. If statement: The **if** statement is a powerful decision-making statement and is used to control the flow of execution of statement. The keyword **if** tells the compiler that what follows is a decision control instruction. The condition following the keyword **if** is always enclosed within a pair of parentheses. If the condition, whatever it is true, then the statement is executed. If the condition is not true, then the statement is not executed. As a general rule, we express a condition using C's 'relational and logical operators'. The **if** statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are:

- 1. Simple if statement.
- 2. if-else statement.
- 3. Nested if-else statement.
- 4. else if ladder.

Simple IF Statement

```
if ( condition )
{
    ....statement-block;
}
Statement-x;
```

The above figure describes the general form of a simple if statement. The '**statement-block**' may be a single statement or a group of statements. If the condition is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x. Remember, when the condition is true both the statement-block and 'statement-x' are executed in sequence.

Note: In simple if statement, only one statement is there no need to open body.

IF ELSE Statement

The if statement by itself will execute a single statement, or a group of statements, when the if condition true, when the condition is false, can we execute another group of statements! This is the purpose of else statement. The if else statement is an extension of the simple if statement. The general form is

```
if ( condition )
{
    TRUE-Block.....statement (s);
}
else ( condition )
{
    FALSE-Block.....statement (s);
}
```

If the condition is true, the true-block statement(s), immediately following the if statements are executed; otherwise, the false-block statement(s) are executed.

NESTED IF-ELSE STATEMENT

When a series of decisions are involved, we may have to use more than one if-else statement in nested form. If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the second condition. If the condition-2 is true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control jumps to statement-4.

Note: Else is always matching with the most recent unpaired if.

```
if ( condition-1 )
{
    if( condition-2 )
    {
        ...statement1;
    }
    else
    {
        ...statement2;
    }
}
else
{
    ...statement3;
}
statement4;
```

Different Forms of if – else

```
if ( condition)
```

```
    statement1;
```

```
if ( condition )
```

```
{
```

```
    statement1;
```

```
.....n;
```

```
}
```

```
if ( condition )
    statement1;
else
    statement2;
```

```
if ( condition )
{
    statement1;
    ....n;
}
else
{
    statement1;
    ....n;
}
```

```
if( condition )
    statement1;
else
{
    if ( condition )
        statement1;
    else
    {
        if ( condition )
            statement1;
        else
        {
            statements;
        }
    }
}
```

```
if ( condition )
{
    if ( condition )
        statement1;
    else
    {
        statements;
    }
}
else
    statement3;
```

B. Conditional operator: The conditional operators ? and : are sometime called ternary operators since they take three arguments. Their general form is,

expression 1 ? expression 2 : expression 3;

What this expression says is: "if expression 1 is true (that is, if its value is nonzero), then the value returned will be expression 2, otherwise the value returned will be expression 3".

- Note:** 1. Number of question marks and colon should be equal.
 2. Every colon should match with just before question marks.

C. Switch statement: The control statement that allows us to make a decision from the number of choice is called a switch.

Rules for switch statement:

- The switch expression must be an integer type.
- Case labels must be constants or constants expression.
- Case labels must be unique.
- Case labels must be end with semicolon.
- The break statement transfers the control out of the switch statement.
- The break statement is optional. That is, two or more case labels may belong to the same statement.
- The default label is optional. If present, it will be executed when the expression does not find a matching case label.
- There can be at most one default label.
- The default may be placed anywhere but usually placed at the end.
- It is permitted to next switch statement.

Note:

1. Using case and default must be within switch body.
2. Using break is optional you can use in switch or loop body.

The integer expression following the keyword switch is any C expression that will yield an integer value or characters. It could be an integer constant or an expression that evaluates to an integer. The keyword case is followed by an integer or a character constant. Each of these values should be unique. Block-1, block-2.. are statement lists and may contain zero or more statements. There is no need to put braces around these blocks. Note that case labels must end with a colon (:).

What happens when we executed a program containing a switch? First, the integer expression following the keyword switch is evaluated, and compared against the values constant1, constant2,... if a matching case is found then the block of statements that follows the case are executed up to break(optional if it is there) or all statements from matching case including default, and transferring the control to the statement-x following the switch.

```
switch ( integer expression )  
{  
    case constant1:  
        block-1;  
        break;  
    case constant2:  
        block-2;  
        break;
```

```
.....  
default:  
    statement;  
    break;  
}  
statement-x;
```

The **default** is an optional case. It will be executed if the value of the integer expression does not match with any of the case values. If not present, no action takes place if all matches fails and the control goes to the statement-x.

D. Goto statement: The goto statement to branch unconditionally from one point to another point in the program.

The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by a colon. The label is placed immediately before the statement where the control is to be transferred. The general form of goto and label statements is shown below:

Note that a goto breaks the normal sequential execution of the program. If the label: is before the statement goto label; a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a backward jump. On the other hand, if the label: is placed after the goto label; some statements will be skipped and jump is known as a forward jump.

goto label; label: statement;	label: statement; goto label
--	--

15. Decision making and looping

In the all programs written so far, execution begins at the start of main() function and proceeds line by line until end of the program. But if you want to print something repeatedly, say printing something on screen five times, you can include 5 print statements. But if you want to print 100 times it cannot be done like this. To do repetitive tasks, we use loops. In C there are three types of loops. Such as,

- a. while loop.
- b. do-while loop.
- c. for loop.

Looping Process includes four steps. They are As follows,

1. Setting And initializing variables (counter)
2. Evaluate the test EXPRESSION
3. Execute The body Of loop if test expression is TRUE
4. Increment or decrement the counter variable.

while Loop

The while statement will be executed repeatedly as long as the Expression Remains true. Its syntax is as follows,

The body of the loop will be executed repeatedly as long as the condition remains true. When the while is reached, the computer evaluates condition. If it is found to be false, body of loop will not be executed, and the loop ends. Otherwise, the body of loop will be executed, and then condition is checked again and so on, till condition becomes false.

```
while ( condition )
{
    ..statement1;
    body of loop;
}
```

Example1: /* calculating sum of first 5 Natural numbers */

```
void main()
{
    int i,sum=0;
    sum=0;
    i=1;          /* initialization */
    while( i<=5)           /* condition */
    {
        sum=sum+i;
        i=i+1;           /* incrementing */
    }
    printf("sum=%d",sum);
}
```

The code given above finds sum of first 5 Natural numbers. The execution of this while statement is done as follows,

- Variables i and sum are initialized.
- Condition ($i \leq 5$) is evaluated. The result is true i.e. hence body of while loop is executed.
- Sum is evaluated and i is incremented by 1.
- Again expression ($i \leq 5$) is evaluated. The result is true so body of loop is executed again. This process is continued until the condition becomes false.

do-while Loop

It is similar to while loop except of while loop except that it always executes at least once. The test of expression for repeating is done after each time body of loop is executed. Syntax for do-while is as follows,

```
do
{
    ....statement1;
    body of loop;
}while( condition );
```

The below example is used to find sum numbers from 1 to given input numbers this do-while loop is executed as given below,

- Variables sum and i are initialized, and n value will enter at runtime (n=5).
- Body of loop is executed and sum, i values are computed.
- Expression ($i \leq n$) is evaluated. It is true so the body of loop is executed again.
- The loop is executed repeatedly until the condition " $i \leq n$ " ($n=5$) becomes false (it happens when $i=6$).

Example2: /* calculating sum of the values from 1 to given input number*/

```
void main()
{
    int i,n,sum;
    printf("ENTER A NUMBER");
    scanf("%d",&n);
    sum=0;
    i=1; /* initialization */
    do
    {
        sum=sum+i;
        i=i+1; /* incrementing */
    } while( i<=n); /* condition */
    printf("\nsum=%d",sum);
}
```

Output:

ENTER A NUMBER 5

sum=15

for Loop

For statement is another type of looping statement. the difference between for and while, do-while statement is that, in while statements that number of passes are not know in advance whereas in for- statement the number of passes are know in advance. The general syntax to for loop is

```
for( initialization; condition ; iteration )
{
    ....statement1;
    body of loop;
}
```

This loop contained three parts; these are initialization, condition and iteration. It is possible to omit all the three expressions in the for statement but it is necessary to have two semicolon. If assignment expression and unary expression are omitted, then other way should be present to initialize and modify the initial value. If the conditional expression is omitted then it is assumed that, it have a static value 1 which is true. This static value makes the looping structure to be executed infinitely until terminated by using break or return statement.

- When for loop is executed for the first time the value will initialize.
- Then it checks the condition if the condition is true and the body of the loop is executed for first time.
- Upon reaching the closing brace of **for**, control is sent back to iteration part of the loop.
- Again it checks the condition, than condition is true and the body of the loop is executed otherwise loop is terminated and execution continue with the next statement that immediately following the loop.

Example3: /* calculating sum of the values from 1 to given input number*/

```
void main()
{
    int i,n,sum=0;
    printf("ENTER A NUMBER");
    scanf("%d",&n);
    for( i=1 ; i<=n ; i++ )
        sum=sum+i;
    printf("\nsum=%d",sum);
}
```

Output:

ENTER A NUMBER 5

sum=15

The above loop where $i=1$ is an expression that specifies the initial value and that controls the looping action. $i \leq 5$ is a condition, that is tested at each pass. The program is executed till this condition remains true. $i++$ is an unary expression, that is either increment or decrement to change the initial value. The conditional expression is evaluated and tested at the beginning while unary expression is evaluated at the end of each pass.

Nesting the loops

Nesting of loop, that is, one loop statement placing into another loop.

```
while ( condition )
{
    while loop body;

    for( initialization; condition ; iteration )
    {
        for loop body;
    }
}
```

break and continue:

break and continue are unconditional statement which are used within a program to alter the flow of control.

break: 'break' is a keyword used to terminate the loop or exit from switch statement. It is written as,

statements;

break;

It can be used with for, while, do-while and switch statement.

Ex: 1

```
char choice;
switch(choice)
{
    case 'y': printf("yes");
                break;
    case 'n': printf("no");
                break;
}
```

Here break statement is used in order to transfer control out of switch statement.

Ex: 2

```
int i;
i=1;
while( i<5 )
{
    if( i%2==0)
        break;
    i++;
}
```

when i=2, i%2==0

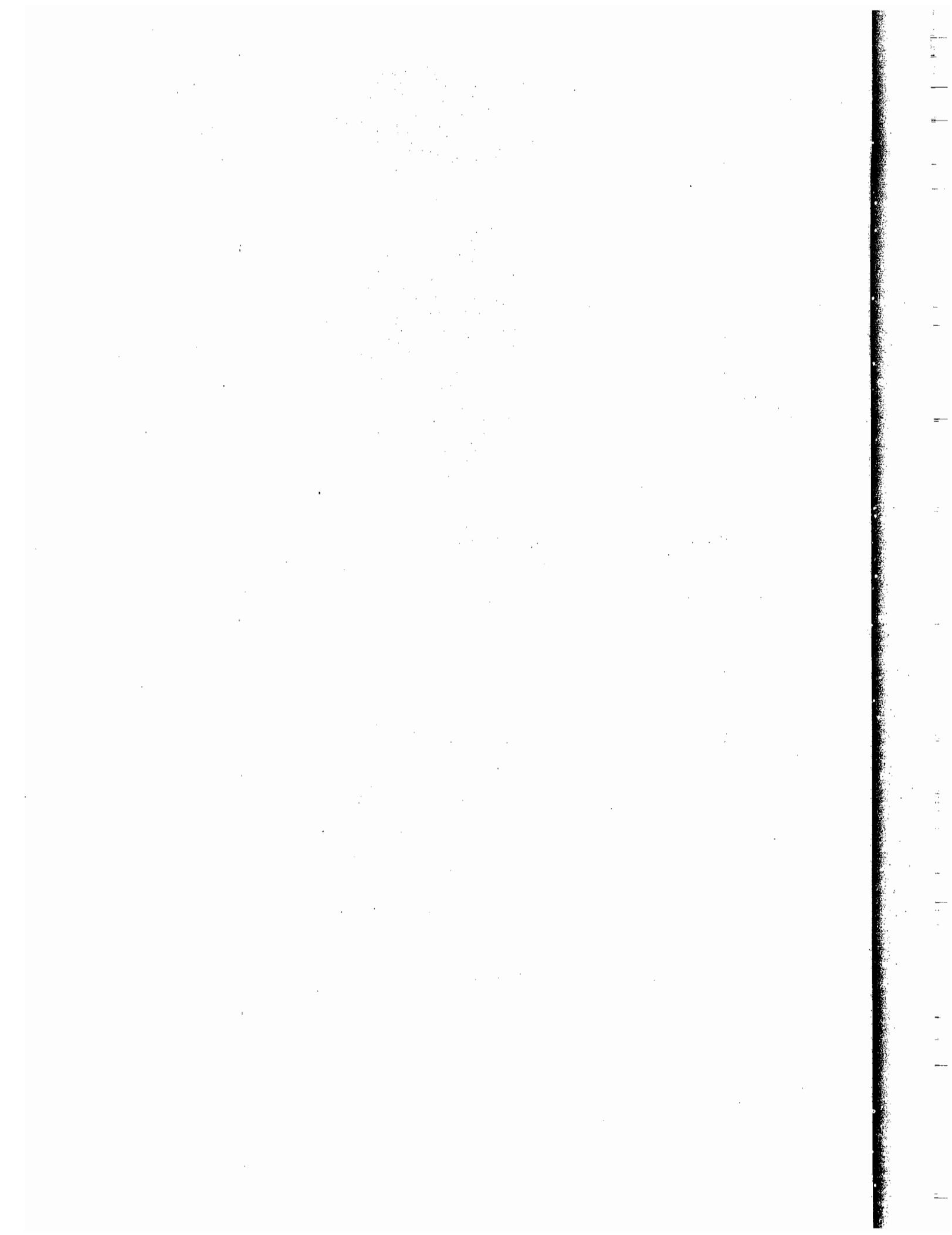
Hence, break statement is executed and loop terminates.

continue: 'Continue' is a keyword used to Skip same specific statement in loop body. It must be used with while, for and do-while.

Ex:1

```
int i;
i=0;
while( i<= 100 )
{
    i+=2;
    if( i>40 && i<60)
        continue;
    printf("%d",i);
}
```

In the above loop, when value of 'i' is 42, continue statement is executed. So, i=i+2 is skipped and control is transferred to beginning of while loop.



Kalyan IT Training Pvt., Ltd., Hyderabad

2ND FLOOR, SRINIVAS NAGAR (WEST), OPP. S.R.NAGAR BUS STOP, HYD - 38

UNIT -II

C & DS

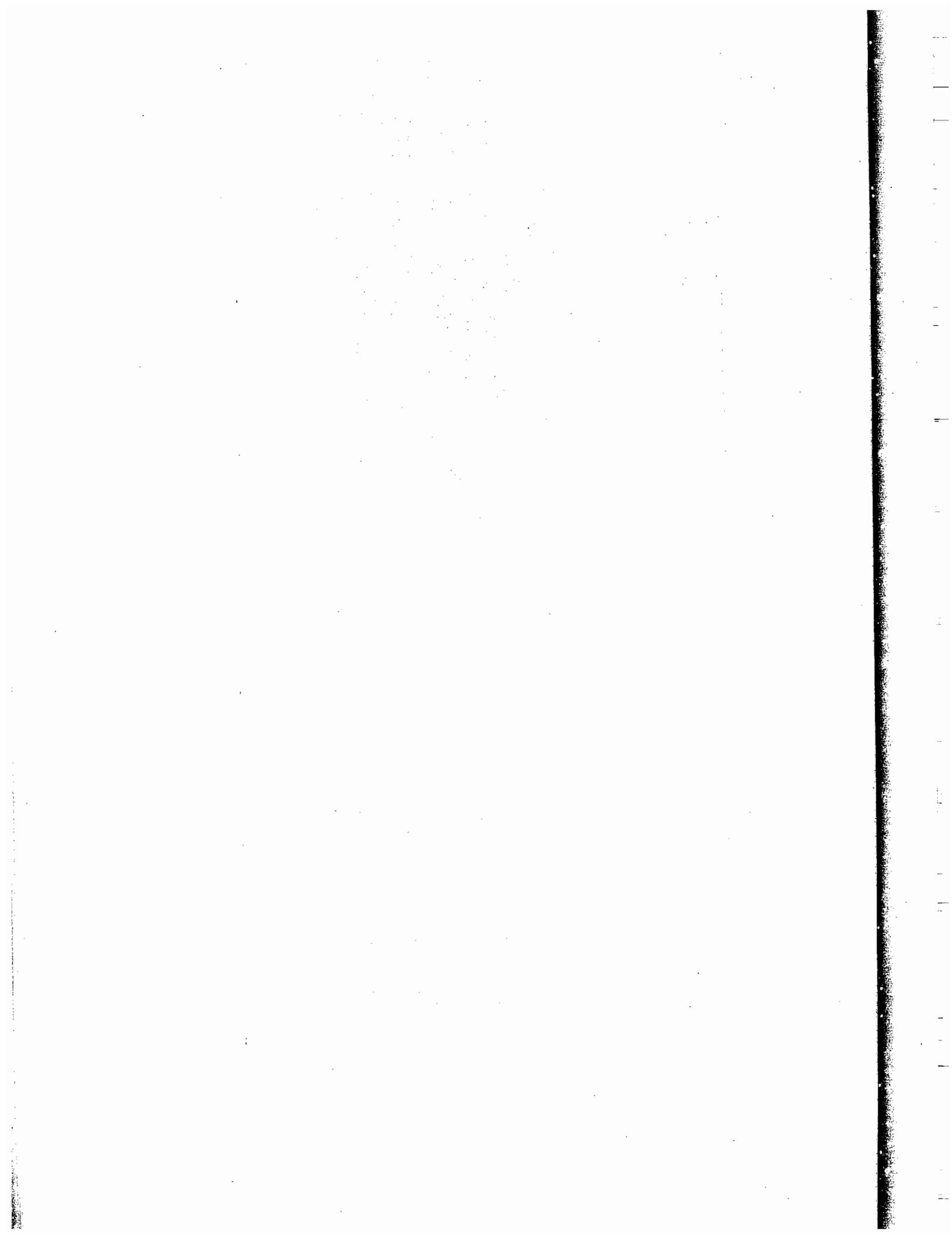
By

Mr. Kalyan Reddy

Kalyan

Ph: **6662 2789, 6662 3789**

www.kalyanit.com

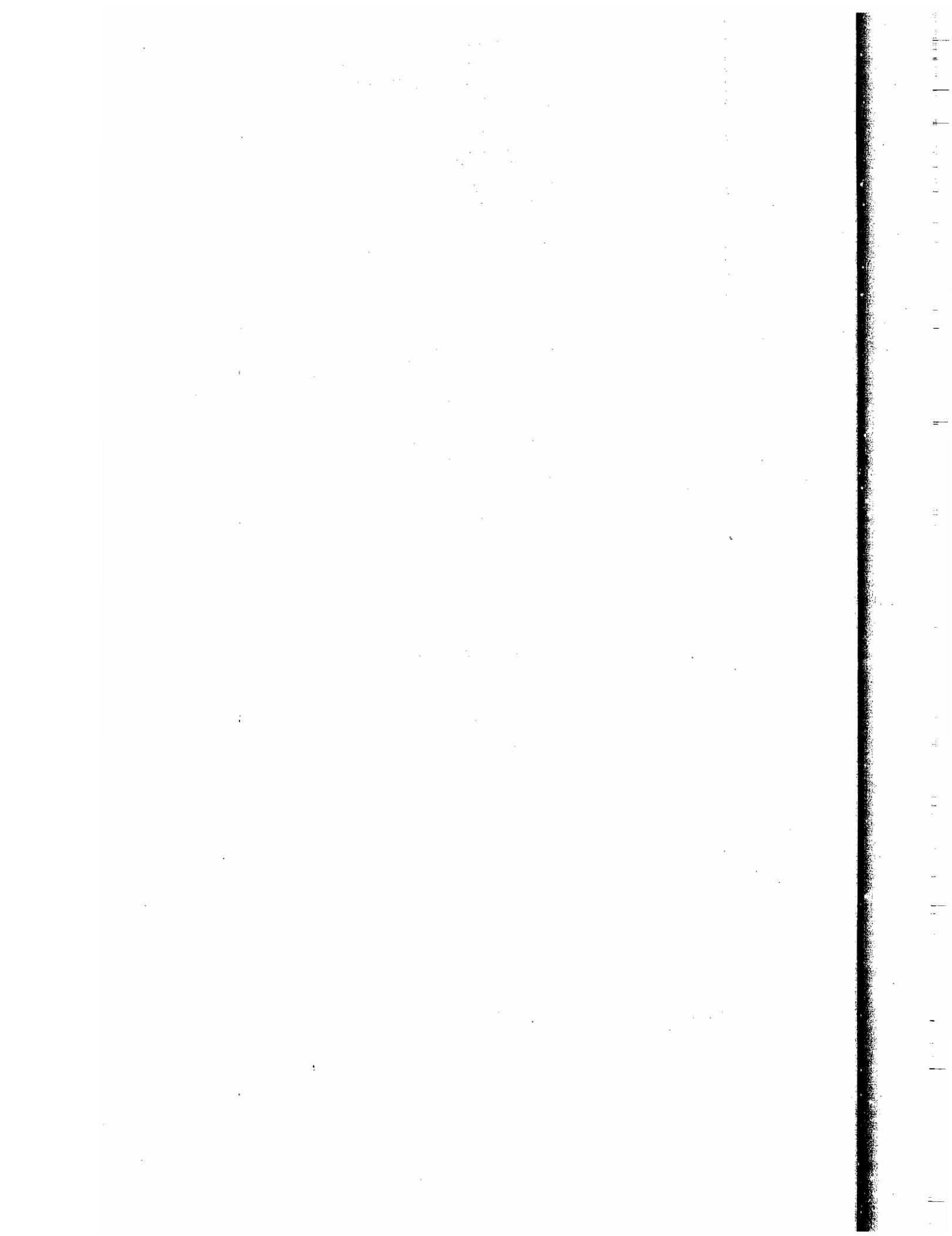


Kalyan IT Training (Hyd) Pvt Ltd

UNIT - II (FUNCTIONS)

<u>Topic</u>	<u>Page No</u>
Definition	- 32
Why use function?	- 32
Advantage of function	- 32
How function works?	- 32
Structure of function	- 33
Parameters in function	- 34
Function prototyping	- 34
Function defining	- 35
Call by value and Reference	- 36
Return or return(Expression)	- 36
Type of functions	- 38
Recursion	- 40
Scope variables, Block structure	- 41
Storage class	- 42
Library functions	- 46
Header files and preprocessors	- 48
Miscellaneous	- 55
a. Global Vs Local variables	
b. Static Vs automatic variables	
c. User-defined Vs built-in functions	
d. Necessity of main() function	





Functions:

1. Definition:

A function is a self-contained block or a sub-program of one or more statements that performs a special task (specific and well defined tasks) when called. A function is a block of statement that performs some kind of task. Every C program can be thought of as a collection of these functions.

2. Why Use functions? :

1. Writing functions avoids rewriting the same code over and over.
2. By using functions it becomes easier to write programs and keep track of what they are doing.
3. Using functions large programs can be reduced to smaller ones. It is easy to debug and find out the errors in it. It also increases readability.

Note:

- a. Purpose of function is code reuse.
- b. We can define the function randomly.
- c. From any function we can call any other function.
- d. Compilation process always starts from top to bottom.
- e. Execution process will start from main and end with main.
- f. Calling a function which is defined later, and then need to declare it (for avoiding the compilation errors).
- g. Function declaration means need to mention return type, name of the function and taking type.
- h. In function defining the first line is called function decelerator.
- i. Function declaration must match with function decelerator.

3. Advantages of Functions:

- a. When programs are very lengthy, handling those programs become difficult. It is also difficult to debug and understand such lengthy programs. In order to overcome these drawbacks, complex programs can be subdivided into smaller modules. These modules are implemented using functions. Hence these subprograms or functions are easy to understand, debug and test.
- b. The part of program which has to be repeated many times can be written as a function and can be called wherever needed to avoid repetition. This helps in reducing size of the program.
- c. A function can be shared by many programs.
- d. It facilitates top-down modular programming.

4. How function works? :

- a. Once a function is defined and called, it takes some data from the calling function and returns a value to the called function.

- b. The detail of inner working of a function is unknown to the rest of the program. Whenever a function is called, control passes to the called function and working of the calling functions is stopped. When the execution of the called function is completed, control returns back to the calling function and executes the next statement.
- c. The values of the actual arguments passed by the calling function are received by the formal arguments of the called function. The number of actual and formal arguments should be the same. Extra arguments are discarded if they are defined. If the formal arguments are more than the actual arguments then the extra arguments appear as a garbage. Any mismatch in the data type will produce the unexpected result.
- d. The function operates on formal arguments and sends back the result to the calling function. The return() statement performs this task.

5. Structure of a function:

```
void message( ); /* function prototype declaration */  
void main( )  
{  
    Function_name (parameter list1); /* function calling */  
    //Ex: message( a,b,c);  
}  
  
Return_value_type function_name(parameter_list2 ) /*  
 function definition */  
//Ex: int message (x,y,z)  
{  
    Local variable Declarations  
}
```

Statements

```
-----  
Return (expression);  
}
```

- a. **Return_value_type:** it is the type that the function returns, according to the value.
- b. **Function_name:** every function is known by a particular name.
- c. **Parameter_list:** parameters are also known as arguments. This list that is enclosed within the parenthesis can have variables that are separated by a comma. These arguments have communication between the calling function and the called function.
The arguments of calling functions are **actual arguments** (parameter_list1).
The arguments of called function are **formal arguments** (parameter_list2).
- d. **Variable declarations:** Local/global variables are declared.
- e. **Statements:** these refer to different statements such as, the printf(), scanf() statements of other assignment statements.
- f. **Return (expression):** returns the value of expression to the calling function.
- g. **Function call:** a compiler executes the function when a semi-colon (;) is followed by function name. a function can be called simply using its name like other C statement, terminated by semi colon (;).

6. Parameters in functions:

Depending on the location of parameters in functions, they are categorized as actual parameters and formal parameters.

Actual parameters: The parameters that are passed in function call are called actual parameters. These parameters are defined in the calling function. Actual parameters can be variables, constants or expressions.

Formal parameters: The parameters that are used in function definition are called formal parameters. These parameters belong to the called function. These can be only variables but not expression or constants.

7. Function Prototyping:

A prototype statement helps the compiler to check the return type and argument type of the function. The prototypes of built-in functions are given in the respective header files. Those we include using `#include directive`. while defining user-defined functions it is a must to declare its prototype.

Kalyan IT Training (Hyd) Pvt Ltd

A function prototype is a declaration of a subroutine (or a function) that specify the function name, an argument list with data types and the return type of a function without the function body. A function prototype specifies the function name, the argument list and return type that a function expects, but it does not specify what the function does.

When the programmer defines the function, the definition of the function must be same like its prototype declaration. If the programmer makes any mistake, the compiler flags an error message. The function prototype declaration statement is always terminated with semi-colon.

Syntax: Return_type function_name (parameter_list);

Ex: float sum (float, int);
 Float sum (float x, int y);

Note:

- If the function is not returning any value, and then specifies the return type as **void**.
- **void** means **nothing**.
- If the functions return type is void then we can't return values.

8. Function Definition:

A function definition is the complete description of a function. Actually a function definition tells what a function does and how it performs. A function definition contains a function body (a block of statements) in addition to function name, arguments list and return type.

Syntax: Return_value_type function_name(parameter_list2)

```
{  
    Local variable Declarations
```

```
    Statements
```

```
        Return (expression);  
    }
```

Ex: float sum (float x, float y)

```
{  
    Float z;  
    Z=x+y;  
    Return (z);  
}
```

9. CALL BY VALUE AND REFERENCE: There are two ways in which we can pass arguments to the function. 1. Call by value 2. Call by reference

- a. **Call by Value:** In this type value of actual arguments are passed to the formal arguments and the operation is done on the formal arguments. Any change made in the formal argument does not affect the actual arguments because formal arguments are photocopy of actual arguments. Hence, when function is called by the call by value method, it does not affect the actual contents of the actual arguments. Changes made in the formal arguments are local to the block of the called function. Once control returns back to the calling function the changes made vanish.
- b. **Call by Reference:** in this type instead of passing values, addresses (reference) are passed. Function operates on addresses rather than values. Here the formal arguments are pointers to the actual arguments. In this type formal arguments point to the actual argument. Hence changes made in the arguments are permanent.

Using call by reference method function can returns more than one value per call.

- Built suitable examples.

10.'Return' or return(expression) :

- The first type **return** does not return any values, it acts much as the closing brace of the function. When a return is encountered, the control is immediately passed back to the calling function.
- When a value is **returned**, it is automatically convert the function's type.

```
float sum(int v1,float v2)
{
    return v1+v2;
}

void main( )
{
    int I;
    float f;
    i=sum(5,5);
    f=sum(5,5);
```

```
printf("%d %f",i,f);
}
```

Calling	i(int)	f(float)
sum(5,5);	10	10
sum(5,5,5);	Error: Too many argument.	
sum(5);	Error: required argument missing.	
sum(7.5,7.5);	14	14.5
sum(7,7.5);	14	14.5
sum(7.5,7);	14	14.0

Note:

- When the function returns integer it is not required to declare it even if it defined later.

Declaration	Calling
Int sum(int,int) ;	i1=sum(i2,i3);
float sum(float,float);	f1=sum(f2,f3);
char sum(char,int,float);	c1=sum(c2,i1,f1);
void sum(); or void sum(void);	sum();
void sum(int,char);	sum(i1,c1);
float sum(char,int,float,float);	f1=sum(c1,i1,f2,f3);

11.Types of functions: There are two functions in C-language.

- a. **Library functions:** The library functions are pre-defined set of functions. Their task is limited. A user can not understand the internal working of these functions. The user can only use the functions but can't change or modify them.
- b. **User defined functions:** these functions allow the programmer to define their own functions according to the requirement of the program. The user has full scope to implement his/her own ideas in the function. The user certainly understands the internal working of the function.

Types of User Define function: A function, depending on whether arguments are present or not and whether a value is returned or not classified into **four** types.

1. Function with no arguments and no return type.
2. Function with arguments and no return type.
3. Function with no arguments and one return value.
4. Function with arguments and return type.

Note:

- If the function return type other than void and returning a value to the calling place is optional.
- Even thought function is returning the value collecting the value also optional.
- Default return type to any function is int.

i. Function with no arguments and no return type: When a function has no arguments, it does not receive any data from the calling function. Similarly, when it does not return a value, the calling function does not receive any data from the called function. There is no data transfer between the calling function and called function. The function is only executed and nothing is obtained. Such functions may be useful to print some messages, draw a line or split the line etc.

Calling function	Analysis	Called function	Example
Main() { - - - - - - Abc(); - - - - - - }	No arguments are passed. No values are sent back.	Abc () { - - - - - - }	void abc() { printf("\nhello xyz"); } void main() { abc(); printf("\nhello main"); }

ii. Function with arguments and no return type: Function receiving data from the calling function. But it does not return a value; the calling function receives data from the called function. And called function is does not returning any values. There is one direction data transfer between the calling function and called function.

Kalyan IT Training (Hyd) Pvt Ltd

Calling function	Analysis	Called function	Example
Main() { - - - - - - Abc(x); - - - - - - }	Arguments are passed. No values are sent back.	Abc(y) { - - - - - - }	Void main() { void sum(int,int); sum(10,20); } void sum(int a, int b) { printf("sum of a,b=%d",a+b); } Out put: sum of a,b=30

iii. Function with no arguments and return value: When a function has no arguments, it does not receive any data from the calling function. But called function returning the value back to the calling place. There is one direction data transfer between the called function and calling function.

Calling function	Analysis	Called function	Example
Main() { Int z; - - - - - - Z= Abc(); - - - - - - }	No arguments are passed. values are sent back.	Abc() { Int y=5; - - - - - - Return (y); }	int get_number() { int n; scanf("%d",&n); return n; } void main() { int i; i=get_number(); printf("%d",i); }

iv. Function with arguments and return type: Function receiving data from the calling function, and called function returning the value back to the calling place. There is two direction data transfer between the calling function and called function.

Calling function	Analysis	Called function	Example
Main() { Int z; - - - - - - Z= Abc(); - - - - - - }	Argument(s) are passed. values are sent back	Abc(y) { - - - Y++; - - - Return(y); }	int sum(int v1,int v2) { return v1+v2; } void main() { int i; i=sum(10,20); printf("%d",i); }

Note:

- Parameters are used in three places, in declaration, function call and function definition.
- The argument (parameter) used in prototypes (declaration) and function definitions are called **formal arguments** and those used in function calls are called **actual arguments**. Actual argument may be simple constants, variables or expressions.
- The formal and actual parameters must match exactly in type, order and number.
- Any function is returning the value to the calling place it is done through the **return** statement.
- The return statement can use in two ways.

12.Recursion: When a called function in turn calls another function a process of '**chaining**' occurs. In C, it is possible for the functions to call themselves. A function is called '**recursive**' if a statement within the body of a function calls the same function. Recursion is the process of defining something in terms of '**itself**'.

```
/* Non-recursive function for calculating the factorial values of an integer */

int fac ( int n )
{
    int f=1,i;

    for ( i=n ; i>=1 ; i-- )
        f=f*i;
    return(f);
}

void main( )
```

```
{  
int a,fact;  
printf("\nEnter any number");  
scanf("%d",&a);  
  
fact=fac(a);  
printf("Factorial value=%d",fact);  
}
```

Output:

Enter any number 3

Factorial value=6

13.Scope variables: scope of a variable is defined as a vicinity or space within a program, where value of a given variable can be accessed.

The area or block of the C program from where the variable can be accessed is known as the scope of variable. The area or scope of the variable depends on its storage class i.e. where and how it is declared. There are four scope variables:

- a. Function
- b. File
- c. Block
- d. Function prototype

Block structure: Block generally refers to a segment within a program providing certain functionality to that program. The ways these blocks are organized are referred as block structures.

A block may hold variables, structures, expressions, control structures etc., which are necessary for a program. It initiates with an opening brace and terminates with a closing brace. A program can hold any number of blocks, also it is possible to insert a block within another block.

14. Storage classes: The storage class of a variable tells the compiler,

- a. The storage area of the variable.
- b. The initial value of variable if not initialized.
- c. The scope of the variable.
- d. Life of the variable i.e. how long the variable would be active in the program.

Types of storage classes: There are four types of storage class in c.

- a. Automatic
- b. External
- c. Static
- d. register

Note: Automatic, static, and external storage classes are kept memory type.

a. **Automatic storage class:** Automatic variables are declared inside a function in which they are to be utilized. They are created when the function is called and destroyed automatically when the function is exited; hence the name automatic variables are private (local) to the function in which they are declared.

void main () { int a;;; }	void main () { auto int a;;; }
---	--

A variable declared inside a function without storage class specification is by default an automatic variable.

We may also use the keyword **auto** to declare automatic variable.

Note:

- Auto variables are kept memory type.
- Default value for this variable is an unpredictable value, i.e. called as garbage value.

- Scope of the auto variable within the block, which the variable is defined.
 - Life of the auto variable till the control remains within the block, in which the variable is defined.
- b. **External storage class:** Variable that are both alive and active throughout the program are known as **external variable**. They are also known as global variable can be accessed by any function in the program. External variable are declared out side a function.
- Once a variable has been declared as global, any function can use it and change its value. Then, subsequent functions can reference only that new value.
- One other aspect of a global variable is that it is available only from the point of declaration to the end of the program.

```
void fun( );
void main ( )
{
    y=5;
    ....;
    ....;
}
int y;
void fun ( )
{
    y=y+1;
    ....;
}
```

We have a problem in above program. As far as main is concerned, y is not defined. So, the compiler will issue an error message.

```
void fun();  
void main ( )  
{  
    extern int y;  
    y=5;  
    ....;  
    ....;  
}  
int y;  
void fun ( )  
{  
    y=y+1;  
    .....;  
}
```

If you want to access a global variable into the our **main()** function by using the keyword **extern**; i.e. **external declaration**.

Note:

- Extern variables are kept memory type.
- Default value for this variable is **zero**.
- Scope of the extern variable is **global**; i.e. **all functions**.
- Life of the extern variable is, as long the program's execution doesn't come to an end.

C. Static storage class: As the name suggests, the value of static variables persists until the end of the program. A variable can be declared static using the keyword **static**.

A static variable may be either an internal type or external type depending on the place of declaration. Internal static variable are those, which are declare inside a function, the scope of internal static variable, extend up to the end of the function in which they are defined. Therefore, internal static variables are similar to auto variables, except that they remain in existence throughout the program.

An external static variable is declared variable is declared outside of all functions and is available to all the functions in the program. The difference between a static external variable and external variable is that the static external variable is available only within the file where it is defined while the external variable can be accessed by other file.

A static variable is initialized only once, when the program is compiled. It is never initialized again.

Note:

- Static variables are kept type.
- Default value for this variable is zero.
- Local to the block in which the variable is defined.
- Life of the static variable is, as long the program's execution doesn't come to an end.

```
void state ( );
void main( )
{
    int I;
    for (i=0 ; i<=3 ; i++ )
        state ( );
}
void state( )
{
    static int x ;
    x=x+1;
```

```
printf("\nx= %d",x);  
}
```

output:

```
x= 1  
x= 2  
x=3
```

D. Register storage class: It is created at the nearest memory available to the CPU named as register. A value stored in a CPU register can always be accessed faster than the one that is stored in memory.

Limitations:

- We can't create n no of register variables (maximum 10 or 20) depending on CPU.
- We can't collect the address of register variable.

Note:

- Default value for this variable is garbage.
- Scope of the register variable within the block, which the variable is defined.
- Life of the register variable till the control remains within the block, in which the variable is defined.

15. Library Functions: C has a large set of useful built-in library functions which are ready to be used in our programs. The function prototypes for these functions are available in general header files. Therefore the appropriate header file for a standard function must be included at the beginning of our programs.

Kalyan IT Training (Hyd) Pvt Ltd

a. Mathematical Functions: These functions are available in math.h

Function	Description	Prototyping	Example
Abs	Returns the absolute value of an integer number	Int abs (int num);	Abs(-5)=5
Fabs	Returns the absolute value of a floating point number	Double fabs(double num);	Fabs (3.2) = 3.2
Labs	Returns the absolute value of a long integer number	Long labs(long num);	Labs(3.2L)=3.2
Ceil	Returns smallest integer value greater than or equal to a number	Double ceil (double num);	Ceil (3.005)=4
Floor	Returns the largest integer value less than or equal to a number	Double floor(double num);	Floor (3.9999)=3.0
Pow	Returns the value of a raised to the power b. an error occurs if a =0 and b<= 0 or if a<0 and b is not an integer.	Double pow(double a, double b);	Pow (2.0, 5.0)=32.0
Sqrt	Returns the square root of a number	Double sqrt(double num);	Sqrt(144.0) = 12

b. String Handling Functions: These functions are available in string.h

Function	Description	Prototyping	Example
Strcpy	Strcpy (target, source)	Copies source string to target string and returns target.	S=strcpy(s2,s1);
Strcat	Strcat (target, source)	Concatenates source string to end of the target and returns target.	S=strcat(s1,s2);
Strlen	Strlen (source)	Returns the length of the source string.	N=strlen(s);

strcmp	strcmp (str1, str2)	Compares two strings and returns values are: 0 – if str1==str2 <0 – if str1 < str2 >0 – if str1>str2	R = strcmp (s1,s2);
--------	---------------------	--	---------------------

c. General Library Functions: These functions are available in stdlib.h

Function	Description	Prototyping	Example
Srand	Creates the first seed for a pseudorandom number series. Seed is a variable that is used by the random number generator to generate next number series.	Void srand (unsigned int seed);	Srand (920);
Rand	Returns a pseudorandom integer in the range 0 to RAND_MAX. in ansi/iso standard the RAND_MAX value is 32,767	Int rand (void);	Rand();

16.Header files & Preprocessors: The header file section and global declaration section are valued for providing unique functions to carryout program execution smoothly in standard C program format.

Header files: Every C program consists of certain provision of header files, provided at the top of main() function. They generally begin with symbol "#" followed by a predefined word "include". With the inclusion of header file prior to main() function facilitates us to use or declare various functions, prototypes ... etc which are nothing but the constituent entities of declared header files. Example, stdio.h header file facilitates us to carryout various I/O functions in our "C" program. Functions like printf(), scan() possess a predefined code in the stdio library.

- Note:**
1. every header file possess '.h' extension.
 2. It is possible to create our own header files.
 3. In some compilers few header files are pre included. We need not require to write include before main().

Kalyan IT Training (Hyd) Pvt Ltd

Preprocessors: In general terms the word preprocessor is a code written to provide input to a processor hence, in other words, a preprocessor is a processor to a processor. Inclusion of preprocessor also begins with a hash (#) symbol. But follows by a special word "**define**". Hence, preprocessor declaration is also known as global declaration, since, the value once defined in this section can be applied once cannot be changed in the remaining program. Though they can be declared anywhere in the program, but it is often good practice to declare it at the beginning.

The preprocessor offers several features called preprocessor directives. These directives can be divided into **four categories**:

1. **Macro substitution directives.**
2. **File inclusion directives.**
3. **Conditional compilation.**
4. **Miscellaneous directives.**

Macro substitution directives: Macro substitution is a process where an identifier in a program is replaced by a predefined string composed of one or more tokens. a macro definition takes the following general form: **#define Identifier String**

Note:

- #define is written just as shown followed by the identifier and a string.
- It should not be ended semicolon.
- At least one blank space required between identifier and a string.
- The string may be any text, which the identifier must be a valid name.

File inclusion directives: This directive causes one file to be included in another. The preprocessor command for file inclusion looks like this: **#include "filename"**. And it simply causes the entire contents of filename to be inserted into the source code at the point in the program. It can be used in two cases:

1. If we have a very large program, the code is best divided into several different files, each containing a set of related function. It is very good programming practice to keep different sections of a larger program separate. These files are **#include** at the beginning of main program file.
2. There are some function and macro definitions that we need almost in all programs that we write. These commonly needed functions and macro definitions can be stored in a file, which would add all the statements in this file to our program as if we have typed them in.

Conditional compilation: Conditional compilation is a preprocessor directive which is used to tell the compiler to skip some part of a source code. This is achieved by inserting the preprocessing commands **#ifdef** and **#endif**.

Advantages of conditional compilation:

1. Conditional compilation allows to "comment out" obsolete lines of code.
2. It permits to have different versions of a same code in the same source file.
3. It makes the program portable i.e., it makes a program to work on two totally different computers.
4. It helps to ensure that header files are only once included in the program.

We can, if we want, have the compiler skip over of a source code by inserting the preprocessing commands #ifdef and #endif, which have the general form:

```
#define macroname

void main( )
{
    #ifdef macroname
        Statement1;
        Statement2;
        Statement3;
    #endif
    Statement4;
}
```

If **macroname** has been #define, the block of code will be processed as usual, otherwise not.

Kalyan IT Training (Hyd) Pvt Ltd

Where would **#ifdef** be useful? When you like to compile only a part of your program? In these cases:

1. To "comment out" obsolete code.
2. A more sophisticated use of **#ifdef** has to do with making the program portable, i.e. to make them work on two totally different computers.

Ex:

```
void main( )
```

```
{
```

```
    printf("KIT\n");
```

```
#ifdef KIT
```

```
    printf("WELCOME\n");
```

```
    printf("HELLO\n");
```

```
#endif
```

```
    printf("KALYAN  
IT\n");
```

```
}
```

```
#define KIT
```

```
void main( )
```

```
{
```

```
    printf("KIT\n");
```

```
#ifdef KIT
```

```
    printf("WELCOME\n");
```

```
    printf("HELLO\n");
```

```
#endif
```

```
    printf("KALYAN IT\n");
```

```
}
```

```
#define KIT
```

```
void main( )
```

```
{
```

```
    printf("KIT\n");
```

```
#ifdef KIT
```

```
    printf("WELCOME\n");
```

```
    printf("HELLO\n");
```

```
#else
```

```
    printf("C AND C++\n");
```

```
#endif
```

```
    printf("KALYAN IT\n");
```

```
}
```

#if and #elif Directives

The **#if** directive can be used to test whether an expression evaluates to a nonzero or not. If the result of the expression is nonzero, then subsequent lines up to a **#else**, **#elif** or **#endif** are compiled, otherwise they are skipped.

```
void main( )
{
    #if condition
        statement1;
        statement2;
        statement3;
    #elif
        statement4;
        statement5;
    #else
        statement6;
        statement7;
    #endif
}
```

If the condition evaluates to true, then staternet 1,2 and 3 are compiled, otherwise **#elif** will evaluates to true, then statement 4 and 5 are compiled , otherwise statement 6 and 7 are compiled.

Miscellaneous Directives: There are more preprocessor directives available, though they are not very commonly used. They are

1. #undef
2. #pragma
3. #error

Undefining a Macro: A defined macro can be undefined, using the statement

#undef identifier

This is useful when we want to restrict the definition only to a particular part of the program.

#pragma Directive: The #pragra is an implementation oriental directive that you can use to turn on or off features. Pragma vary form one compiler to another. Turbo C/C++ compiler has got a pragma that allows you to suppress warning generated by the compiler.

- #pragma startup and #pragma exit: These directives allow us to specify functions that are called upon program startup (before main()) or program exit.
- #pragma warn: On compilation the compiler reports Errors and Warnings in the program, if any. Errors provide the programmer with no option, apart from correcting them. Warnings, on the other hand, offer the programmer a hint or suggestion that something may be wrong with a particular statement. Two most common situation when warning are displayed are us under:
 - Return type warnings.
 - Run-time errors, such as assigning a value that never used.

```
#pragma warn-rvl      /* return valu*/
#pragma warn-par      /*parameter not used */
#pragma warn-rch      /*unreachable code */
```

```
void kit ( );
void abc( );
#pragma startup kit
#pragma exit abc

void main ( )
{
    printf("inside main\n");
}
```

```
void kit ( )  
{  
    printf("inside kit\n");  
}  
void abc( )  
{  
    printf("inside abc\n");  
}
```

Output:

```
insid kit  
inside main  
inside abc
```

Note: That the function kit() and abc() should not return any value. If we want two functions to get executed at startup then their pragmas should be defined in the reverse order in which you want to get them called.

Stringizing Operator # : ANSI C providing operator # called stringizing operator to be used in the definition of macro functions. This operator allows a formal argument within a macro definition to be converted to a string.

```
#define sum(xy) printf(#xy "%f\n",xy)  
void main()  
{  
    ...;  
    sum(a+b);  
}
```

This preprocessor will convert the line sum(a+b) into printf ("a+b""=%f\n",a+b); which is equivalent to printf ("a+b=%f\n",a+b);

Token Pasting Operator ##: The token pasting operator ## defined by ANSI standard enables us to combine two tokens within a macro definition to form a single token.

Ex: // Hours to seconds conversion program//

```
#include <stdio.h> //header file inclusion declaration
#define SEC 60           //Preprocessor declaration
#define MIN 60           // Preprocessor declaration
Void main()
{
    Int HOUR=0, TIME_IN_SEC;
    Printf("enter hours:");
    Scanf("%d",&HOURS);
    TIME_IN_SEC=HOUR*MIN*SEC;
    Printf("The time in seconds is %d",TIME_IN_SEC);
}
```

17. Miscellaneous:

A. Global variable Vs Local variable

Global Variable	Local variable
a. The variables which are declared above any function are called global variables b. These variables can be accessed by all the functions in the program. c. Default value for global variable is zero.	a. The variables which are declared within any function are called local variables. b. These variables can be accessed only by the function in which they are declared c. Default value for local variable is garbage value.

B. Necessity of main() in C

- A C program consists of one or more functions. One of them must be a main() function.

Kalyan IT Training (Hyd) Pvt Ltd

- The main() function is a user-defined function which is called by the compiler to execute the program.
- In other words the execution of a C program starts and ends with main().
- The main() function can call other functions to perform special tasks. Therefore every C program must include a main() function.
- Every function returns some value. The return value for main() void because it returns nothing.

C. Static Vs automatic variables

Variable	Storage class	Keyword	Storage Location	Scope	Extent	Initial value
Automatic variables	Automatic	Auto	Memory	Local to the block or function in which the automatic variable is declared	The automatic variable will be alive during execution of block or function in which it is declared.	Garbage
Static variables	Static	Local static	Memory	Local to the block or function in which static variable is declared	It is alive between two function calls.	Zero
		Global static	Memory	Accessible from the place of declaration to entire program	It is alive during execution of entire program.	Zero

D. User-defined Vs built-in functions

User-defined functions	Built-in functions
The functions that are defined by the programmers according to the requirements of the program are called "user-defined functions"	Built-in function are commonly required functions that are not grouped together and are stored in a library.
These functions can be modified by the programmers i.e., they are modifiable.	These function are not modifiable
They are not predefined.	They are predefined
These are written by the programmers.	These are not written by the programmers.
White space between the function names and parenthesis () causes an error.	White spaces between the function names and parenthesis () are ignored in case of build-in functions
User-defined functions can be overloaded.	Built-in functions cannot be overloaded.
The programmer is responsible for defining the user-defined functions, hence their definitions increases the programmer's overhead.	Programmer's overhead is reduced here, as they are available as "ready-to-use" functions.
Not already available, hence are created when required.	Already available for the programmers to call.
Their definitions can appear anywhere in a program	Their definitions appear in standard libraries.
They must be declared before they are used.	Declarations of the built-in functions are not required.
Example: display(); area(); etc..	Example: cos(x), sqrt (x), sin(x), etc.

Kalyan IT Training Pvt., Ltd., Hyderabad

2ND FLOOR, SRINIVAS NAGAR (WEST), OPP . S.R.NAGAR BUS STOP , HYD - 38

UNIT -III

C & DS

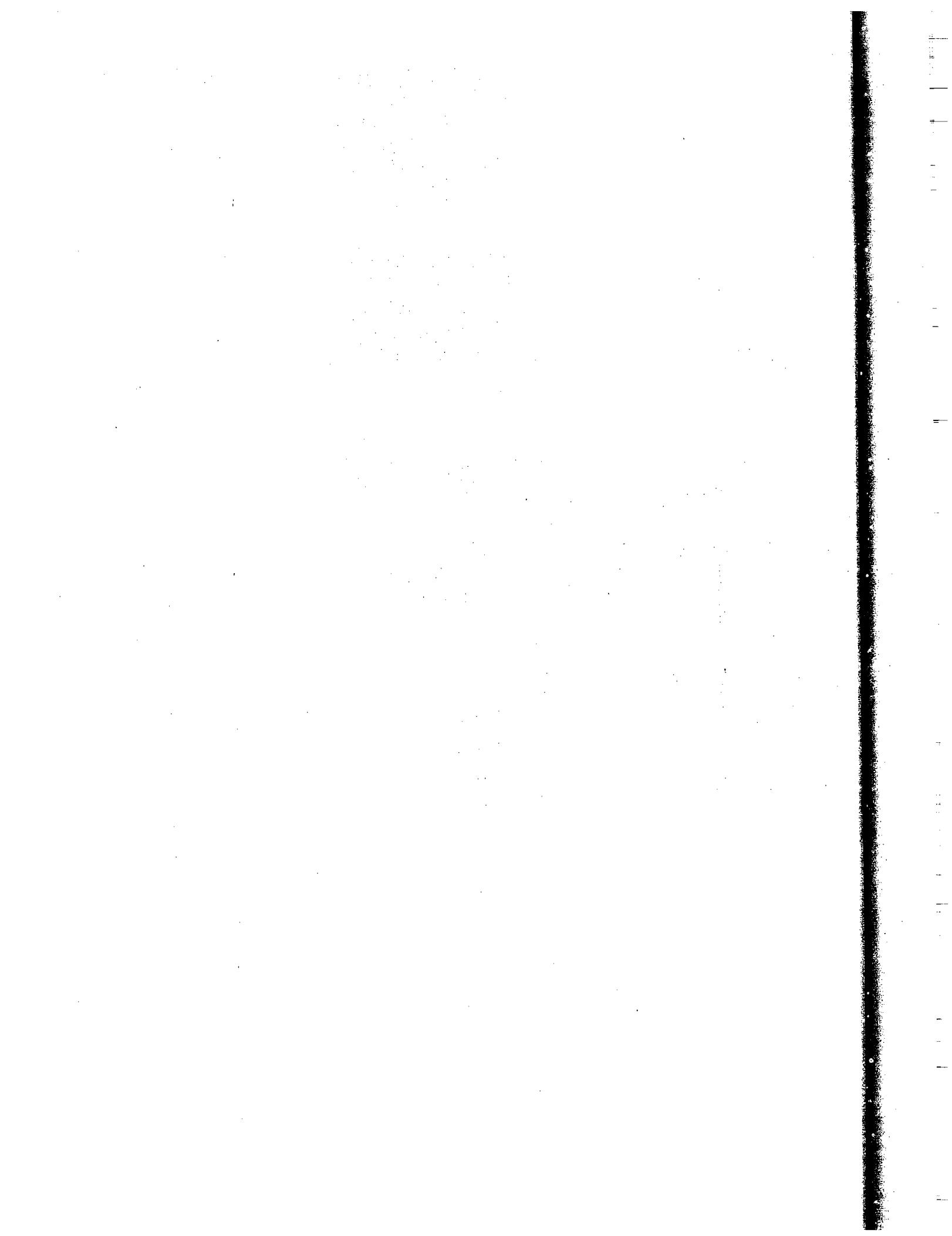
By

Mr. Kalyan Reddy

Kalyan

Ph: 6662 2789, 6662 3789

www.kalyanit.com

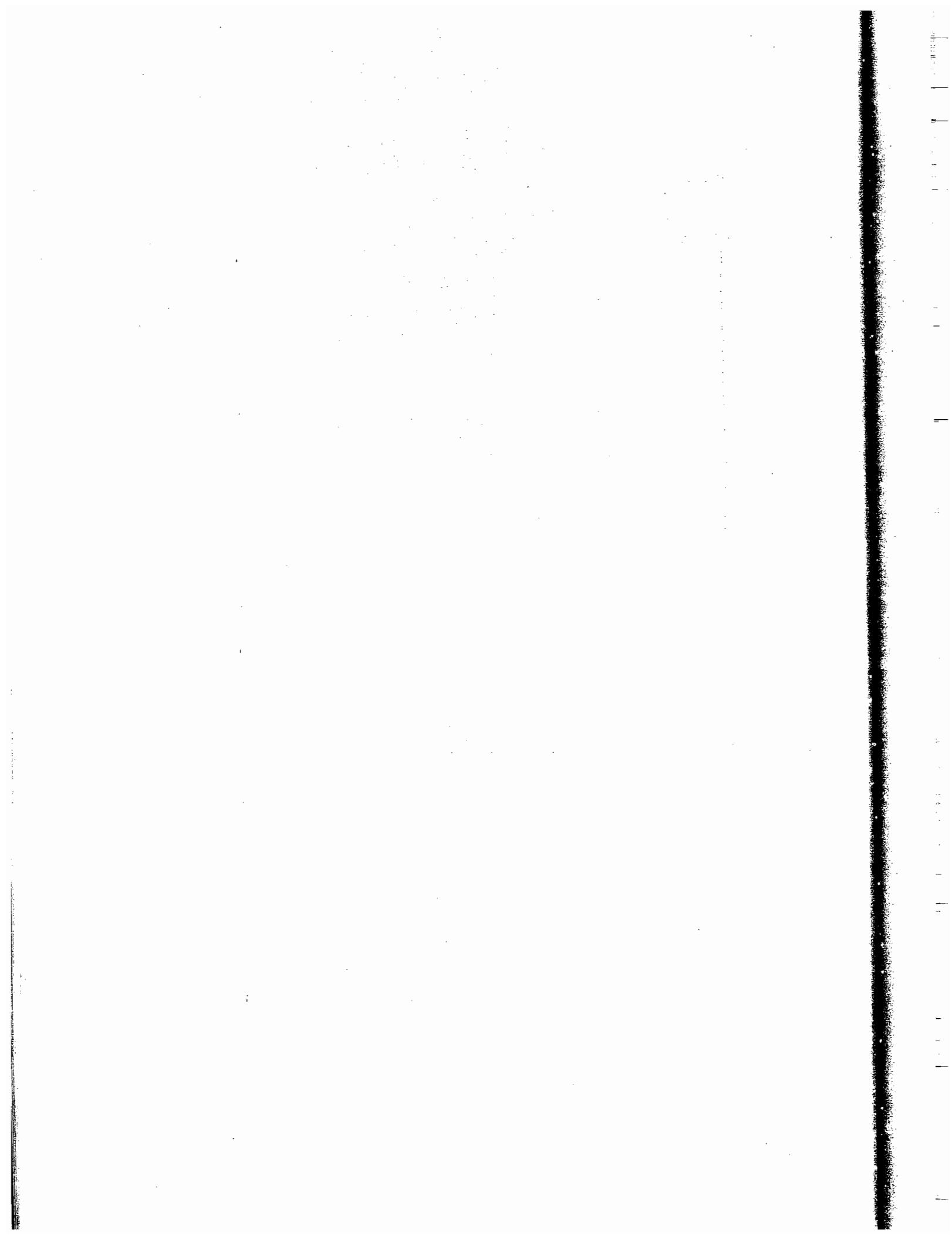


Kalyan IT Training (Hyd) Pvt Ltd

UNIT - III (Arrays)

<u>Topic</u>	<u>Page No</u>
Definition	58
Characteristics of array	58
One dimensional array	48
Array initialization	59
Memory allocation for array	60
2-dimensional array	60
3-dimensional array	62
Arrays with functions	63
Arrays with pointers	65
Strings	66
Reading strings from terminal	67
String standard functions	68
Pointers	70
Initialization of pointer variables	71
Arithmetic operations with pointers	73
Pointer as function argument	74
Pointer to function	76
Pointers & arrays	76
Arrays of pointers	77
Pointers to pointers	78
Pointers & strings	78
Miscellaneous	79
DMA management	80
Command line arguments	81





Arrays

Definition: array is a collection of similar data types in which each element is unique one and located in separate memory locations. These similar elements could be all **ints**, or all **floats**, or all **chars**, etc. usually, the array of characters is called a '**string**'.

We can use arrays to represent not only simple list of values but also tables of data in **two**, **three** or **more** dimension.

- o One-dimensional arrays.
- o Two-dimensional arrays.
- o Multidimensional arrays.

Characteristics of Array: let us take the example **int a[5]={1,2,3,4,5};**

1. The declaration **int a[5]** is nothing but creation of 5 variables of integer types in the memory. Instead of declaring five variables for five values, the programmer can define them in an array.
2. All the elements of an array share the same name, and they are distinguished from one another with the help of an element number.
3. The element number in an array plays major role for calling each element.
4. Any particular element of an array can be modified separately without disturbing other elements. If a programmer needs to replace 8 with 10, he/she need not require to change all other numbers except 8. To carry out this task the statement **a[4]=10** can be used. Here other three elements are not disturbed.
5. Any element of an array **a[]** can be assigned/equated to another ordinary variable or array variable of its type.

Ex: **b=a[2];**
a[2] = a[3];

in the statement **b=a[2]** or vice versa, value of **a[2]** is assigned to '**b**', where '**b**' is an integer. In the statement **a[2]=a[3]** or vice versa, value of **a[2]** is assigned to **a[3]**, where both the elements are of same array. The array elements are stored in continuous memory locations. The amount of storage required for holding elements of the array depends on its type and size. The total size in bytes for a single dimensional array is computed as:

total bytes=sizeof (data types) * size of array.

One Dimensional arrays:

Array Declaration: simple example of array declaration.

int marks[30];

Here, **int** specifies the type of the variable, just as it does with ordinary variables and the word **marks** specifies the name of the variable. The number 30 tells how many elements of the type **int** will be in our array. This number is often called the 'dimension' or 'subscript' of the array. The bracket (**[]**) tells the compiler that we are dealing with an array.

Subscript: subscript of an array is an integer expression, integer constant or integer variables like 'i', 'n' etc., that refers to different elements in the array. In the above example array subscripts start at zero. Therefore the elements are from marks[0] to marks[29]. Subscript have a range, starting from '0' upto the "size of array - 1". Subscripts can also be reflected in loops that print the array elements.

Array Initialization: Following are a few examples that demonstrate this.

```
int number[6]={ 2,4,6,8,10,12};  
int n[]={ 1,11,21,31,41,51};  
float pass[]={ 12.3, 22.3, 33.3, 44.3};  
int kit[5]={ 5,15,25,35,45,55,65};
```

Note:

- Till the array elements are not given any specific values, they are supposed to contain garbage values.
- If the array is initialized where it is declared, mentioning the dimension of the array is optional as in the 2nd and 3rd example above.
- If the array is initialized more than size of the array that is error.

Accessing array elements: the elements of an array can be accessed using an index known as *array index*. The smallest possible value of array index is know as *lower bound*. The largest value is known as *upper bound*. In "C" lower bound is always 0 and upper bound is N-1, where N is size of array.

Entering Data at Run Time: Here is the example to understand easily

Here is the section of code that places data into an array at run time.

```
for( i = 0 ; i <=29 ; i++ )  
{  
    printf ( "\nEnter marks ");  
    scanf ( "%d", &marks[i] );  
}
```

The for loop causes the process of asking for and receiving a student's marks from the user to be repeated 30 times. The first time through the loop, i has a value 0, so the scanf() function will cause the value typed to be stored in the array element marks[0], the first element of the array. This process will be repeated until become 29. This is the last time through the loop, which is a good thing, because there is no array element like marks[30].

In scanf() function, we have used the "address of" operator (&) on the element marks[i] of the array, just as we have used it earlier on other variable. In so doing, we are passing the address of the address of this particular array element to the scanf() function, rather than its value; which is what scanf() requires.

Reading Data from an Array:

Kalyan IT Training (Hyd) Pvt Ltd

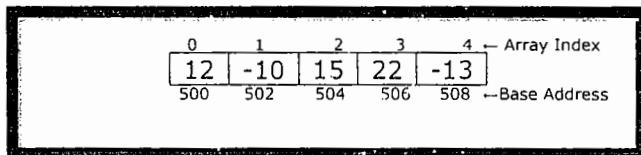
The balance of the program reads the data back out of the array and used it to calculate the average. The for loop is much the same, but now the body of the loop causes each student's marks to be added to a running total stored in a variable called sum. When all the marks have been added up, the result is divided by 30, the number of students, to get average.

```
for ( i= 0 ; i <= 29; i++ )  
    sum=sum+marks[i];  
  
    avg=sum/30;  
    printf( "\nAverage marks = %d " ,avg);
```

Memory allocations for array: Consider the following array declaration:

```
int arr[5];
```

What happens in memory when we make this declaration? 10 bytes get immediately reserved in memory, 2 bytes each for the 5 integers. And since the array is not being initialized, all five values present in it would be garbage values. This so happens because the storage class of this array is assumed to be **auto**. If the storage class is declared to be **static**, then all the array elements would have a default initial value as **zero**. Whatever be the initial values, all the array elements would always be present in contiguous memory locations. This arrangement of array element in memory is shown below.



Boundary Checking In Array: In C, C++ there is no checks to see if the subscript used for an array exceeds the size of the array. Data entered with a subscript exceed the size of the array. Data entered with a subscript exceeds the array size will simply be placed in memory outside the array; probably on top of other data, or on the program itself. This will lead to unpredictable results, to say the least, and there will be no error message to warn you that you are going beyond the array size. In some cases, the computer may just hang. Thus, the following program may turn out to be suicidal.

```
void main ( )  
{  
    int num[10],i;  
  
    for ( i=0 ; i<= 20 ; i++ )  
        num[i]=i;  
}
```

Thus, to see to it that we do not reach beyond the array size, is entirely the programmer's botheration and not the compiler's.

Two-Dimensional Array: The two-dimensional array is also called a matrix. Two-dimensional arrays are declared as follows:

Kalyan IT Training (Hyd) Pvt Ltd

Type array_name [row_size][column_size];

Ex: int arr[4][3];

Initialisation: When we are going to initializing the values we need to maintain row and columns by using curly braces and when the row is compiled close the braces, if you are using more rows and use separator (,) between.

```
int arr[4][3]={  
    {1,11,21},  
    {2,12,22},  
    {3,13,33},  
    {4,14,44}  
};
```

Once an array is declared or initialized, let us see how individual elements in the array can be referred. This is done by using two subscripts, the number in fist subscript referred the row address and second one for column following the array name. This number specifies the element's position in the array. All the array elements are numbered, starting with 0 and 0. Thus, arr[0][0] is the first element, arr[1][0] is fourth element and arr[3][2] is the last element.

It is important to remember that, while initializing a 2-D array, it is necessary to mention the second dimension (column), whereas the first dimension is optional (row). Thus the declarations,

```
int arr[2][2]={2,12,22,32};  
int arr[ ][2] ={3,13,23,33}; are perfectly acceptable. Whereas  
  
int arr[2][]={4,14,24,34};  
int arr[ ][ ]={5,15,25,35}; are not acceptable.
```

Entering Data at Run Time

Here is the section of code that places data into an array at run time.

```
for( i = 0 ; i <=3 ; i++ )  
{  
    for( j = 0 ; j <=2 ; j++ )  
    {  
        scanf ( "%d", &arr[i][j] );  
    }  
}
```

The outer for loop to be repeated 4 times that is equal to number of rows and inner for loop repeated 3 times that is equal to number of column in each row. The first time through the loop, i and j has a value 0, so the scanf() function will cause the value typed to be stored in the array element arr[0][0], the first element of the array. This process will be repeated until j value become 2. Then outer for loop variable will be increment with one again inner loop repeated 3 times. This process repeated until outer loop becomes false.

Memory allocations for 2-D array

Kalyan IT Training (Hyd) Pvt Ltd

Let us reiterate the arrangement of array element in a two-dimensional array of students, which contains roll no's: in one column and the marks in other.

The array arrangement shown in below figure is only conceptually true. This is because memory contains rows and columns. In memory, whether it is a 1-D or 2-D array, the array elements are stored in one continues chain. The arrangement of array element of a 2-D array in memory is shown below:

```
int arr[3][2]={1,11,21,31,41,51};
```

	[0][0]	[0][1]	[1][0]	[1][1]	[2][0]	[2][1]	Array Index
1	450	2	391	3	481		
	500	502	504	506	508	510	Address

We can easily refer to the marks obtained by the marks obtained by the second student using the subscript notation as shown below:

```
printf("Marks of second student=%d",arr[1][1]);
```

Three or Multi-dimensional arrays:

We aren't going to show a programming example that uses a 3-D array. This is because, in practice, one rarely uses this array. However, an example of initializing a 3-D array will consolidate your understanding of subscripts:

```
int arr[3][3][2]={
{
    {1,11},
    {2,22},
    {3,33}
},
{
    {4,44},
    {5,55},
    {6,66}
},
{
    {7,77},
    {8,88},
    {9,99}
};
};
```

A 3-D array can be thought of as an array of arrays of arrays. The outer array has three elements; each of which is a 2-D array of three one-dimensional arrays, each of which contains two integers. In other words, a one-dimensional array of two elements is constructed first. Then four such one-dimensional arrays are placed one below the other to give a two-dimensional arrays are placed one behind the other to yield a three-dimensional array containing three 2-D arrays. In the array declaration, note how the commas have been given.

0 th 2-D Array	1 st 2-D Array	3 rd 2-D Array
1 11 2 22 3 33 4 44 5 55 6 66 7 77 8 88 9 99		
500	512	524

How would you refer to the array element 99 in the above array? The first subscript should be [2], since the element is in third 2-D array. The second subscript should be [2] since the element is in third row of the two-dimensional array; and the third subscript should be [1] since the element is in second position in the one-dimensional array. We can, therefore, say that the element 1 can referred as **arr[2][2][1]**. It may be noted here that the counting of array element even for a 3-D array begins with zero. Can we not refer to this element using pointer notation? Of course, yes. For example, the following two expressions refer to the same element in the 3-D array. **arr[2][2][1] *(*(*arr+2)+2)+1)**

C allows of 3-D or more dimensions. The exact limit is determined by the compiler. The general form of a multi-dimensional array is

type array_name[s₁][s₂][s₃].....[s_m];

Where s_i is the size of the ith dimension.

Arrays with functions

Array element can be passed to a function by calling the function by value, or by reference. In the call by value, we pass values of array elements to the function, whereas in the call by reference, we pass address of array element to the function. These two calls are illustrated below:

```
/* Demonstrated of call by value */
void KIT(int);
void main( )
{
int i;
int marks[6]={55,66,77,88,99,100};
for ( i = 0 ;i <= 5 ; i++ )
    KIT( marks[i] );
}
void KIT( int r)
{
printf("%d",r);
}
```

Here, we are passing an individual element at a time to the function KIT() and getting it printed in the function KIT(). Note that, since at a time only one element is being passed, this element is collected in an ordinary integer variable r, in the function KIT().

```

/* Demonstrated of call by reference */
    void display(int *);
void main( )
{
int i;
int marks[6]={55,66,77,88,99,100};
for ( i = 0 ;i <= 5 ;i++ )
    display( &marks[i] );
}
void display( int *r)
{
printf("%d",*r);
}

```

Here, we are passing address of individual array element to the function `display()`. Hence, the variable in which this address is collected `r`, is declared as a pointer variable. And since `n` contains the address of array element, to print out the array element, we are using the 'value at address' operator `*`.

Passing an Entire Array to a Function

Like the value of simple variables, it is also possible to pass the values of an array to a function. To pass an array to a called function, it is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments. For example, the call `largest(arr,n)`; will pass all the elements contained in the array `arr` of size `n`. The called function expecting this call must be appropriately defined. The `largest` function header might look like:

```
int largest(int array[],int size);
```

The function `largest` is defined to take two arguments, the array name and the size of the array to specify the number of elements in the array. The declaration of the formal argument `array` is made as `int array[]`; the pair of brackets informs the compiler that the argument `array` is an array of numbers. It is not necessary to specify the size of the array here.

Let us consider a problem of finding the largest value in an array of elements. The program is as follows:

```

float largest ( int*,int);
void main ( )
{
float arr[5]={2.5,-8.2,4.4,5.8,8.2};
printf("\n%d",largest(arr,5));
}
float largest( int a[ ],int size)
{
int i;
float max;
max=a[0];
for( i = 1 ; i <size ; i++ )

```

```
{
if(max < a[i])
    max= a[i];
}
return (max);
}
```

When the function call largest (arr,5) is made, the values of all elements of the arr are passed to the corresponding element a in the called function. The largest function finds the largest value in the array and returns the result to main.

Arrays with pointers

When we array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the element of the array in contiguous memory locations. The base address is the location of the first element (index 0) of the array. The compiler also defines the array names as a constant pointer to the first element. Suppose we declare an array x as follows:

int x[5]={1,2,3,4,5};

Suppose the base address of x is 500 and assuming that each integer requires two bytes, the five elements will be stored as follows:

Elements	x[0]	x[1]	x[2]	x[3]	x[4]
Address	500	502	504	506	508

The name x is defined as a constant pointer pointing to the first element, x[0] and therefore the value of x is 1000, the location where x[0] is stored. That is,

$$x = \&x[0] = 500$$

If we declare p as pointer of type integer, then we can make the pointer p to point to the array x by the following assignment: **p=x;** this is equivalent to **p=&x[0];**

Now, we can access every value of x using p++ to move from one element to another. The relationship between p and x is shown below:

p	=&x[0]=500
p+1	=&x[1]=502
p+2	=&x[2]=504
p+3	=&x[3]=506
p+4	=&x[4]=508

You may notice that the address of an element is calculated using its index and the scale factor of the data type. For instance,

$$\begin{aligned} \text{Address of } x[3] &= \text{base address} + (3 * \text{scale factor of int}) \\ &= 500 + (3 * 2) \\ &= 506 \end{aligned}$$

When handling arrays, instead of using array indexing, we can use pointers to access array elements. Note that *(p+3) gives the value of x[3]. The pointer accessing method is much faster than array indexing.

Pointer can be used for 2-D, 3-D and multi-dimensional arrays as well. We know that in a one-dimensional array x, the expression ***(x+i)** or ***(p+i)** represent the element x[i]. Similarly, an element in a 2-D array can be represented by the pointer expression as follows:

***(*($a+i$)+ j) or *(*($p+i$)+ j)**

The below figure illustrates how this expression represents the element $a[i][j]$. The base address of the array a is $\&a[0][0]$ and starting at the address, the compiler allocates contiguous space for all the elements, row-wise. That is, the first element of the second row is placed immediately after the last element of the first row, and so on. Suppose we declare an array a as follows.

```
int a[3][4]={  
    {1,11,21,31},  
  
    {2,12,22,32},  
  
    {3,13,23,33}  
};
```

The element of a will be stored as shown below:

	row 0				row 1				row 2				
	1	11	21	31	2	12	22	32	3	13	23	33	
	500				508				516				

If we declare p as an int pointer with the initial address of $\&a[0][0]$, than $a[i][j]$ is equivalent to $*(p+4 * i+j)$.

You may notice that, if we increment i by 1, the p is incremented by 4, the size of each row, making p element $a[2][3]$ is given by $*(p+4 * 2+3) = *(p + 11)$.

This is the reason why, when a 2-D array is declared, we must specify the size of each row so that the compiler can determine the correct storage mapping.

Strings

Definition:

Group of characters, digits, and symbols enclosed within quotation marks are called as **string**. The string is always declared as character array. In other words character arrays are called strings. Example:

“Welcome to Usha Rama”

Character strings are often used to build meaningful and readable programs. The common operations performed on character string are:

- Reading and Writing strings.
- Combining strings together.
- Copying one string to another.
- Comparing strings for equality.
- Extracting a portion of a string.

Declaring and Initializing Strings

A string variable is any valid C variable name and is always declared as an array. The general form of declaration of a string variable is:

```
char string_name[size];
```

Kalyan IT Training (Hyd) Pvt Ltd

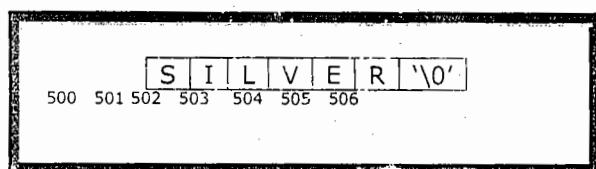
The size determines the number of characters in the string_name. Some examples are:

```
char city[10];  
char name[10];
```

A string constant is a one-dimensional array of characters terminated by a NULL ('\0') character. For example: **char name[] = {'S','I','L','V','E','R','\0'};**

Each character in the string occupies one byte of memory and the last character is always '\0'. The '\0' character looks like two characters, but it is actually only one character, with the \ indicating that what follows it is something special. '\0' is called NULL character. Note that '\0' and '0' are not same. ASCII value of '\0' is 0, whereas ASCII value of '0' is 48.

The terminating NULL is important, because it is the only one way the functions that works with a string can know where the string ends.



C concedes the fact that you would use strings very often and hence provides a shortcut for initializing strings. For example, the string used above can also be initialized as, **char name[] = {"SILVER"};**

Note that, in this declaration '\0' is not necessary. C inserts the null character automatically.

Reading Strings from Terminal:

Reading Words: The familiar input function **scanf()** can be used with %s format specification to read in a string of characters.

```
char name[10];  
scanf("%s",name);
```

The problem with **scanf** function is that it terminates its input on the first white space it finds. (A white space includes blanks, tabs, carriage returns, form feeds, and new lines.) Therefore, if the following line of text is typed in at the terminal, for example if we enter a string **Usha Rama** then only the string "Usha" will be read into the string name. Since the blank space after the word "Usha" will terminate the string.

Note that unlike previous **scanf** calls, in the case of character arrays, the ampersand (&) is not required before the variable name. The **scanf** function automatically terminates the string that is read with a NULL character and therefore the character array should be large enough to hold the input string plus the NULL character.

If we want to read the entire line "Usha Rama", then we may use two character array of appropriate sizes. That is,

```
char name1[10];  
char name2[10];  
scanf("%s%s",name1,name2);
```

with the line of text Usha Rama will assign the string "Usha" to name1 and "Rama" to name2.

Reading a Line of Text: In many texts processing application, we need to read in an entire line of the text from the terminal. It is not possible to use scanf function to read a line containing more than one word. This is because the scanf terminates reading as soon as soon as a space is encountered in input.

We can read a line of text by using getchar or gets functions. By using getchar function we can read a single character from terminal. We can use this function repeatedly to read successive single characters from the input and place them into character array. Thus, an entire line of text can be read and stored in an array. The reading is terminated when the newline character ('\n') is entered and the NULL character is then inserted at the end of string.

The function scanf() is not capable of receiving multi-word strings. Therefore, names such as "Usha Rama" would be unacceptable. The way to get around this limitation is by using the function **gets()**.

Putting String on screen: We have used extensively the printf function with %s format to print strings to screen. The format %s can be used to display an array of character that is terminated by NULL character. For example, the statement
printf("%s",name);

Can be used to display the entire contents of the string name. We can also specify the precision with which the string is displayed. For instance, the specification %10.4 indicates that the first four characters are to be printed in a field width of 10 columns.

However, if we include the minus sign in the specification (e.g., %-10.4), the string will be printed left justified.

For printing the string on the screen we can use another function is called puts. the fact that, puts() can display only one string at a time. Also, on displaying a string, unlike printf(), puts() places the cursor on the next line.

String formats with different precision:

Sr.No	Statement	Output
1.	printf("%s\n",text);	PRABHAKAR
2.	printf("%.5s\n",text);	PRABHA
3.	printf("%.8s\n",text);	PRABHAKA
4.	printf("%.15s\n",text);	PRABHAKAR
5.	printf("%-10.4s\n",text);	PRAB
6.	printf("%11s\n",text);	PRABHAKAR

String standard functions:

The "C" library provides a large number of string handling functions that can be used for performing string manipulations. These functions are available in the file **string.h**. this header file is used or included whenever these function are used.

Functions	Description
Strlen()	Determines length of a string.
Strcpy()	Copies a string from source to destination.

Kalyan IT Training (Hyd) Pvt Ltd

Strncpy()	Copies characters of a string to another string upto the specified length.
strcmp()	Copies characters of a string to another string upto the specified length.
Stricmp()	Compares two strings. (Doesn't discriminate b/w small, capital letters)
Strncmp()	Compares characters of two strings upto the specified length.
Strnicmp()	Compares characters of two strings upto the specified length. Ignores cases.
Strlwr()	Converts uppercase characters of a string to lowercase.
strupr()	Converts lowercase characters of a string to uppercase.
Strdup()	Duplicates a string.
Strchr()	Determines first occurrence of a given character in a string.
Strrchr()	Determines last occurrence of a given character in a string.
Strstr()	Determines first occurrence of a given string in another string.
Strcat()	Appends source string to destination string.
Strncat()	Appends source string to destination string upto specified length.
Strrev()	Reverses all characters of a string.
Strset()	Sets all characters of string with a given argument or symbol
Strnset()	Sets specified numbers of characters of string with a given argument or symbol.
Strspn()	Finds upto what length two strings are identical.
Strpbrk()	Searches the first occurrence of the character in a given string and then it displays the string starting from that character.

Program examples:

a. strcpy():

```
main()
{
    char ori[20],dup[20];
    clrscr();
    printf("enter ur name:");
    gets(ori);
    strcpy(dup,ori);
    printf("original string:%s",ori);
    printf("duplicate string:%s",des);
}
```

Output:

```
enter ur name: SACHIN
original string: SACHIN
duplicate string: SACHIN
```

b. strupr():

```
main()
{
    char upper[15];
    clrscr();
    printf("\nEnter string in lower case:");
    gets(upper);
    printf("after strupr():%s",strupr(upper));
}
```

Output:

```
Enter string in lower case: abcde
After strupr(): ABCDE
```

c. strlen():

```
#include<string.h>
void main()
{
    char str[10]="Usha Rama";
    int L;
    L=strlen(str);
    printf("String lenght=%d",L);
```

d. strchr():

```
void main()
{
    char string[30],ch,*chp;
    printf("enter test:");
    gets(string);
    printf("character to find");
    ch=getchar();
    chp=strchr(string,ch);
```

<pre>}</pre> <p><u>output:</u> String lenght=9</p>	<pre>if(chp) printf("character %c found",ch); else printf(character %c not found",ch); }</pre>
<p>e. strcmp():</p> <pre>void main() { char source[10]="Usha Rama"; char destination[10]="Usha Rama"; int i; i=strcmp(s1,s2); printf("\n%d",i); i=strcmp(source,"UshaRama"); printf(" %d",i); }</pre> <p><u>Output:</u></p> <p>0 -50</p>	<p>f. strcat():</p> <pre>void main() { char source[10]="Rama"; char destination[10]="Usha"; strcat(destination,source); printf("\n Source string=%s", source); printf("\n Destination string=%s", destination); }</pre> <p><u>Output:</u></p> <p>Source string=Rama Destination string=Usha Rama</p>

Pointers

Definition: a pointer is a memory variable that stores a memory address. Pointer can have any name that is legal for other variable and it is declared in the same fashion like other variables but it is always by '*' operator.

A pointer is a derived data type in C. it is built from one of the fundamental data type available in C. Always a pointer stores memory address.

Features:

- a pointer save the memory space.
- Execution time with pointer is faster because data is manipulated with the address i.e. direct access to memory location.
- The memory is accessed efficiently with the pointers. The pointer assigns the memory space and it also releases. Dynamically memory is allocated.
- Pointers are used with data structures. They are useful for representing two-dimensional and multi-dimensional arrays.

Pointer declaration: The declaration of a pointer variable takes the following form:

Data_type * pointer_name;

- The asterisk (*) tells that the variable pointer_name is a pointer variable.

- Pointer_name need a memory location.
- Pointer_name points to a variable of type data_type.

Ex: int *x;
 Float *f;
 Char *y;

- a. In the first statement 'x' is an integer pointer and it tells to the compiler that it holds the address of any integer variable. In the same way 'f' is a float pointer which stores the address of any float variable and 'y' is a character pointer that stores the address of any character variable.
- b. The indirection operator (*) is also called the deference operator. When a pointer is dereferenced, the value at the address stored by the pointer is retrieved.
- c. Normal variable provides direct access to their own values whereas a pointer provides indirect access to the values of the variable whose address it stores.
- d. The indirection operator (*) is used in two distinct ways with pointers, declaration and reference.
- e. When a pointer is declared, the star indicates that it is a pointer, not a normal variable.
- f. When the pointer is dereferenced, the indirection operator indicates that the value at that memory location stored in the pointer is to be accessed rather than the address itself.
- g. Also note that * is the same operator that can be used as the multiplication operator. The compiler knows which operator to call, based on the context.
- h. The '&' is the address operator and it represents the address of the variable. The %u is used with printf() function for printing the address of a variable. The address of any variable is a whole number. The operator '&' immediately preceding the variable returns the address of the variable.
- i. We can declare the pointer variable in 3 forms:
`int *ptr;
int* ptr;
int * ptr;`

Initialization of pointer variables:

The process of accessing the address of a variable to a pointer variable is known as initialization. All uninitialized pointer will have some unknown values that will be interpreted as memory address.

Once a pointer variable has been declared we can use the assignment operator to initialize the variable.

```
void main ( )
{
int i;
int *ptr;
ptr=&i;
}
```

Note:

- We can combine the initialization with the declaration. That is int *ptr=&i;

Accessing a variable through pointer:

Once a pointer has been assigned the address of a variable, the question remains as to how to access the value of the variable using the pointer? This is done by using another operator *, usually known as the indirection operator or dereference operator, or object at that location.

```
void main ( )
{
int i,n,*ptr;
ptr=&i;
i=25;
n=*ptr;

}
```

The first line declares **i**, and **n** as integer variables and **ptr** as pointer variable which points to integer value.

The second line assigns the address of **i** to a pointer variable **ptr** and the third line assigns the value 25 to **i**. The fourth line contains the **indirection operator ***. When the operator * is placed before a pointer variable in an expression, the pointer returns the values of the variable of which the pointer value is the address. In this case ***ptr** returns the value of the variable **I**, because **ptr** is the address of **i**.

```
/* simple example with pointer */
void main ( )
{
int a,b;
int *ptr;
a=10;
b=20;
ptr=&a;
*ptr=30;
ptr=&b;
*ptr=40;
printf("%d %d %d ",a,b,*ptr);
}
```

Output:

30 40 40

Note:

- Any type of pointer size is same because it stores the address.
- On **DOS** based compilers pointer size is 2 bytes. This all are unsigned values **0 to 65535**. Total address are allocated at runtime is **65536**.
- On **UNIX** based compiler millions of addresses are available, and pointer size is **4** bytes.

All the pointers have same sizes because they are storing similar values. But there is necessity to store address of character in **char***, address of integer in **int*** and float address in **float***. The difference is seen when we pointing the variable by using ***** operator. For example if we are using **char*** pointer to hold integer address then, when we using ***** operator than it points first 1 byte only.

Rules for pointer Operation:

1. Two pointer variables cannot be added.
2. A pointer variable can be added or subtracted with an integer value.
3. A pointer variable can be subtracted with an integer value.
4. Two pointer variables, a pointer variable with an integer value cannot be multiply.
5. A pointer variable can be assigned the address of another variable.
6. A pointer variable can be pre-fixed or post-fixed with increment or decrement operator.
7. We can compare pointers using equality and inequality operator.
8. A value cannot be assigned to an arbitrary address (i.e. **&x=10;** is illegal).

Arithmetic operations with pointers:

Arithmetic operation on pointer variables is also possible. Increase, decrease, prefix & postfix operations can be performed with the help of the pointers.

Data type	Initial Address	Operation	Address after operations	Required bytes
-----------	-----------------	-----------	--------------------------	----------------

Int i=2	4046	++	--	4048	4044	2
Char c='x'	4053	++	--	4054	4053	1
Float f=2.2	4058	++	--	4062	4054	4
Long l=2	4060	++	--	4064	4056	4

Ex: arithmetic operations using pointers.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a=25, b=10, *p, *j;
    p=&a;
    j=&b;
    clrscr();
    printf("\n addition a+b=%d",*p+b);
    printf("\n subtraction a-b=%d",*p-b);
    printf("\n product a*b=%d",*p**j);
    printf("\n division a/b=%d",*p / *j);
    printf("\n a mod b =%d",*p % *j);
}
```

Output: Addition a+b = 35
Subtraction a-b = 15
Product a*b = 250
Division a/b = 2
A mod b = 5

Addition: a number can be added to a pointer or addition of two variables through pointers is also possible. In the first printf() statement value of 'b' is added to pointer of 'a' i.e. *p.

Subtraction: a number can be subtracted from a pointer. Subtraction of two variables through pointers is also possible. In the second printf() statement value of 'b' is subtracted from pointer of 'a' i.e. *p.

Multiplication: multiplication of two pointers or a multiplication of number with pointer variable can be done. In the third printf() statement multiplication of variable 'a' and 'b' is done through their pointers '*p' and '*j'.

Note: the following things are not possible.

1. Addition of two addresses (pointers).
2. Multiplication of addresses or multiplication of address with a constant.
3. Division of address with a constant.

Pointer as function argument: Arguments can generally be passed to function in one of the two ways:

1. **Sending the values of the arguments.**
2. **Sending the addresses of the arguments.**

In the first method, the 'value' of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function. With

Kalyan IT Training (Hyd) Pvt Ltd

this method, the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the called function. The following program illustrates the '**call by value**' or '**pass by value**'.

That the value of v1 and v2 remain unchanged even after exchanging the value of n1 and n2.

```
void swap (int n1, int n2)
{
    int t;
    t=n1;
    n1=n2;
    n2=t;

    printf ("\nn1=%d n2=%d",n1,n2);
}

void main( )
{
    int v1,v2;
    v1=10;
    v2=20;
    swap(v1,v2);
    printf("\nv1=%d v2=%d",v1,v2);
}
Output:
n1=20 n1=10
v1=10 v2=20
```

In the second method (call by reference), the address of actual argument in the calling function are copied into the formal argument of the called function. This means that, using these addresses, we would have an access to the actual arguments and hence we would be able to manipulate them. The following program illustrates the '**call by address**' or '**call by reference**'.

```
void swap (int *p1, int *p2)
{
int t;
t=*p1;
*p1=*p2;
*p2=t;
}

void main( )
{
int v1,v2;
v1=10;
v2=20;
```

```
swap(&v1,&v2);
printf("\nv1=%d v2=%d",v1,v2);
}
```

The output of the above program would be:
a=20 b=10

Note:

- That this program manages to exchange the values of v1 and v2 using their addresses stored in p1 and p2.

Pointer to function: A function, like a variable, has a type and an address location in the memory. It is therefore, possible to declare a pointer to a function, which can then be used as an argument in another function. A pointer which is pointing to dead location is called **dangling pointer**.

```
int *fun ( )
{
int a;
a=25;
return &a;
}

void main ( )
{
int *ptr;
ptr=fun( );
*ptr=35;
printf("%d",*ptr);
}
```

Pointers and Arrays: Array name by itself is an address or pointer. It points to the address of the first element (0th) element of an array. The elements of the array together with their addresses can be displayed by using array name itself. Array elements are always stored in contiguous memory locations.

Ex:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int arr[5]={10,20,30,40,50}, p=0;
clrscr();
for (p=0;p<5;p++)
{
    printf("value of arr[%d]=",p);
    printf("%d | ",arr[p]);
    printf("%d | ",*(arr+p));
    printf("%d | ",*(p+arr));
}
```

```

        printf("%d | ", p[arr]);
        printf("address of arr[%d]=%u\n", p, &arr[p]);
    }
}

```

Arr[p]: displays various array elements. Here 'arr' refers to the address and 'p' refers to element number.

***(arr+p):** arr+p is addition of a constant with the base address of an array.

***(p+arr):** this is same as above.

P[arr]: this is same as arr[p]. here, 'p' refers to the element number and 'arr' refers to the base address. By varying 'p' and 'arr' the various elements of the array are displayed.

Ex: two dimensional arrays with pointer

```

#include <stdio.h>
#include <conio.h>
Void main()
{
    Int I;
    Int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    Clrscr();
    Printf("\t elements of an array with their addresses.\n\n");
    For (i=0;i<9;i++)
    {
        Printf("%8u", &a[0][0]+i);
        Printf("[%d]", *(&a[0][0]+i));
        If(i==2 || i==5)
            Printf("\n");
    }
}

```

Output: elements of an array with their addresses.

1[4052]	2[4054]	3[4056]
4[4058]	5[4060]	6[4062]
7[4064]	8[4066]	9[4068]

Array of pointers: it is nothing but a collection of addresses. Here, we store address of variables for which we have to declare an array as a pointer.

Ex: void main()

```

{
    Int *arrp[3];
    Int arr[3]={5,10,15},k;
    For (k=0;k<3;k++)
        Arrp[k] = arr+k;
    Clrscr();
    Printf("\n\taddress element\n");
    For (k=0;k<3;k++)
    {
        Printf("\t%u", arrp[k]);
        Printf("\t%7d\n", *(arrp[k]));
    }
}

```

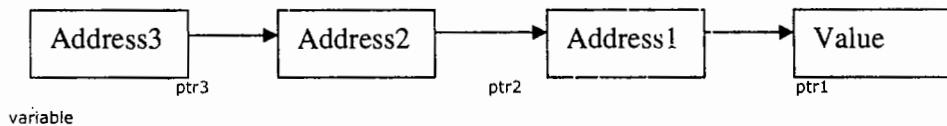
}

Output: address element
 4060 5
 4062 10
 4064 15

Element No.	Array of values	Element No.	Array of addresses
Arr[0]	5	Arrp[0]	4060
Arr[1]	10	Arrp[1]	4062
Arr[2]	15	Arrp[2]	4064

Pointers to pointers:

Pointer is known as a variable containing address of another variable. The pointer variables also have an address. The pointer variable containing address of other pointer variables is called as pointer to pointer. This chain can be continued to any extent.



Here, the pointer variable ptr3 contains the address of the pointer variable ptr2 and ptr2 contains the address ptr1 which is pointing to a variable. This is known as multiple indirections.

A variable that is a pointer to a pointer to pointer must be declared using additional indirection operator symbol in front of the name.

```

/* Example for pointer to pointer concept*/

void main ( )
{
    int i;
    int *ptr;
    int **pptr;
    int ***ppptr;
    ptr=&i;
    pptr=&ptr;
    ppptr=&pptr;
    i=25;
    printf("%d %d %d %d",i,*ptr,**pptr,***ppptr);
}
  
```

Pointers and strings: pointer usage on strings is similar to arrays only.

Ex: main()
 {
 char name[15], *ch;

```
printf("enter ur name:");
gets(name);
ch=name;
while(*ch]!='\0')
{
    printf("%c",*ch);
    ch++;
}
}
```

Miscellenious:

1. Arrays vs ordinary variable

An array is a collective name given to a group of similar elements whereas a variable is any entity that may change during program execution.

An array is a variable that is capable of holding many values. Whereas, an ordinary variable can hold a single value at a time. For example, an array initialization would be, **int number[8];** for which there are eight storage locations reserved where eight different values can be stored belonging to the same type. Whereas an ordinary variable initialization would be, **int number;** and only one storage location is reserved and can hold only one value at a time.

2. Effect of following statements:

- a. **int a, *b=&a;**: 'a' is an integer type i.e., 'a' is capable of holding an integer type value. The variable 'b' is a pointer variable of type int containing the address of 'a'.
- b. **int p, *p;;**: show error by compiler. Because same variable is used both normal and pointer variable.
- c. **char *s;;**: specifies 's' as a 'char' pointer i.e., variable 's' contains the address of a 'char', value.
- d. **a=(float *)&x;;**: This statement assigns the address of the variable x to 'a'. the contents of 'a' is casted with float, hence, 'a' contains float values.

3. Effect of following statements where m and n declared as integers and p1 and p2 as pointers to integers:

- a. **P1=&m;**: this statement assigns the address of variables m to the pointer variable p1 and p2 is said to "point to" m.
- b. **P2=n;;**: assigns the value of variable 'n' to the pointer variable p2. Since, 'n' is not initialized; a garbage value stored in 'n' is assigned to pointer p2.
- c. **M=p2-p1;;**: finds the difference of two pointers and assigns the resultant address as a value to the variable m.
- d. ***p1=&n;;**: assigns the address of n to the pointer variable p1 so the value at pointer p1 is the address of n.

4. Some important functions functionality:

- a. **gets() - puts();**

Kalyan IT Training (Hyd) Pvt Ltd

`gets()` collects a string of characters terminated by a new line from the standard input stream `stdin` and puts it into '`s`'. `gets` replaces the newline by a null character (`\0`) in '`s`'; it also allows input strings to contain certain whitespace characters (spaces, tabs). `gets` returns when it encounters a newline; everything up to the newline is copied into '`s`'.

`puts` copies the null-terminated string '`s`' to the standard output stream `stdout` and appends a newline character.

Return Value: `gets()` returns the string argument '`s`'. On end-of-file or error, `gets` returns null. `Puts()` returns a non-negative value. on error, `puts` returns a value of EOF.

b. `getc()` – `putc()`:

`getc` is a macro that gets one character from a stream. `Getc()` returns the next character on the given input stream and increments the stream's file pointer to point to the next character. `getc` returns the character read, after converting it to an int without sign extension.

`putc` is a macro that outputs a character to a stream. `putc` outputs the character given by `c` to the stream given by `stream`. `putc` returns the character given by `c`. On error (and on end-of-file for `getc`); both functions return EOF.

c. `getchar()` – `putchar()`: `getchar() == scanf("%c",ch);`

`getchar()`: only enter key is separator. Whereas for `scanf()` tab, space, enter key are separators. `getchar` is a macro that gets a character from `stdin`. `getchar` is a macro defined as `getc(stdin)` `getchar` returns the next character on the input stream `stdin`. `getchar` returns the character read, after converting it to an int without sign extension.

`putchar()`:`putchar` is a macro that outputs a character on `stdout`. `putchar` is a macro defined as `putc(c, stdout)` `putchar` puts the character given by `c` on the output stream `stdout`. `putchar` returns the character given by `c`.

On error (and on end-of-file for `getchar`), both macros return EOF.

d. `getch()` – `getche()`:

DMA Management

The Dynamic memory management is basically used to calculate runtime memory requirements. With this we can not only saves the amount of effort needed while developing the program but also saves us from recruiting the

entire program. Basically there are two major functions namely malloc() and calloc() referred to be dynamic memory management functions.

Malloc(): it reserves contiguous block of memory whose size in bytes is specified at "memory size". When a program obtains a memory block through malloc(), its contents remains undefined. Malloc() takes one argument. And allocates garbage value initially.

Syntax: void *malloc(size_t memory_size);
Ex: int *arr;
 Arr=(int *) malloc(no * size of (int));

Calloc(): reserves a block of memory whose size in bytes is at least count x size. In other words, the block is large enough to hold an array of count elements, each of which takes up size in bytes. Calloc() takes two arguments and allocates zeroes initially.

Syntax: void *calloc(size_t count, size_t size);
Ex: int *arr;
 Arr=(int *) calloc (no, size of (int));

Realloc(): causes resizing of allocated memory block. The below example goes to new memory address and first place occupied with old data and returns memory address to 'arr'. (here 100 is required extra memory).

Ex: int *arr;
 arr=(int *) realloc(arr,100);

Free(): causes releases of already allocated memory block.

Ex: int *arr;
 free(arr);

above all four functions available in **stdlib.h** header file.

Command line arguments

Arguments passed to a program from the command line are known as command line arguments.

Programs written in 'C' language can also be invoked with the command line arguments. In this case, the operating system provides the information about the arguments to the main() function. On the other hand the main() function has to be declared with two parameters i.e., **argc** and **argv[]**. Argv[] stands for number of arguments.

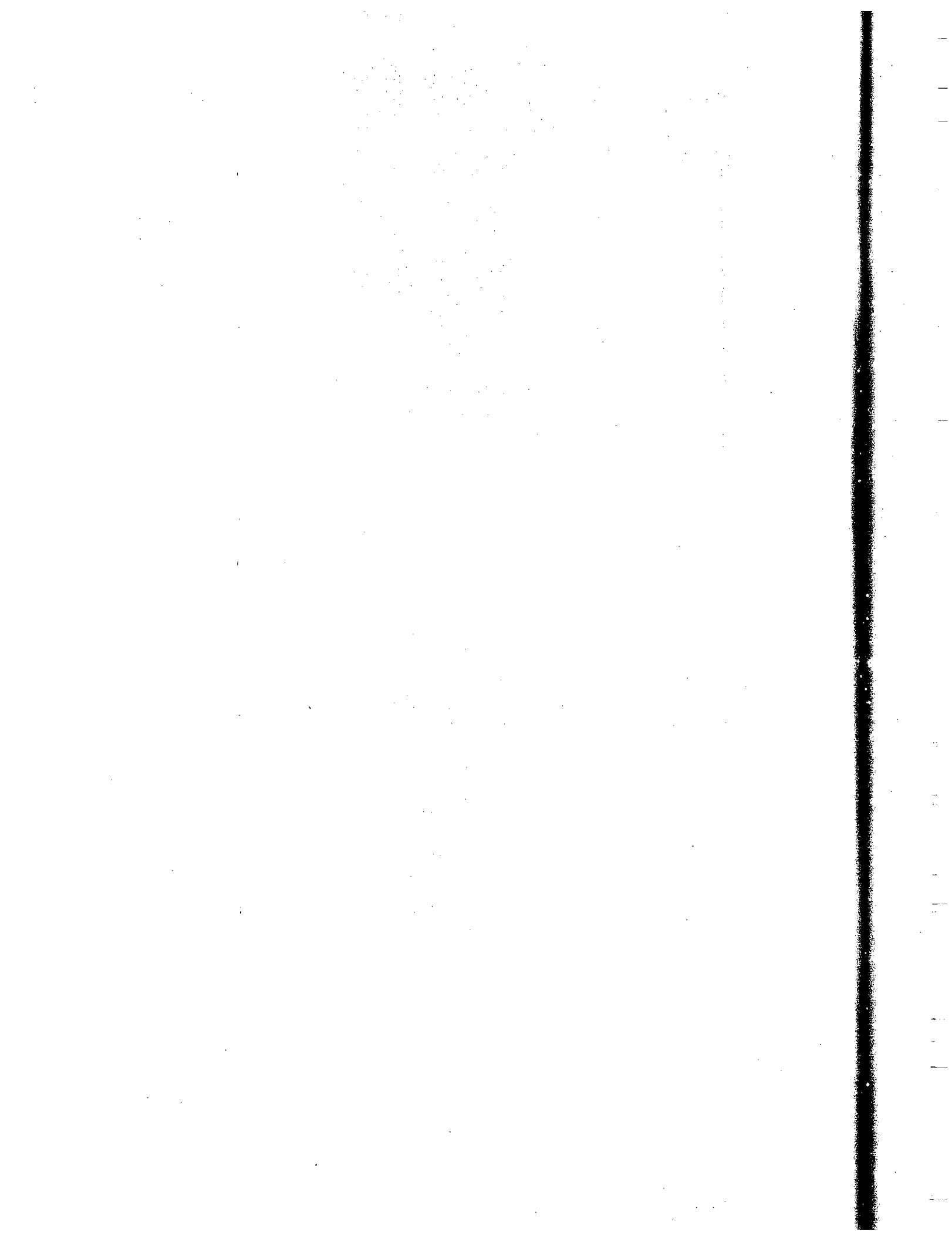
Default value of argc=1 in C and C++. Whereas it is 0 in java. Using command line arguments it is possible to create your own commands and can run in DOS operating system.

Ex:

Kalyan IT Training (Hyd) Pvt Ltd

```
#include <stdio.h>
#include <conio.h>

main(int argc, char *argv[])
{
    int I;
    printf("\n number of arguments = %d \n",
    argc);
    printf("the arguments passed are");
    printf("\n");
    for (i=0;i<=argc-1;i++)
    printf("%s\n",argv[i]);
    getch();
}
```



Kalyan IT Training Pvt., Ltd., Hyderabad

2ND FLOOR, SRINIVAS NAGAR (WEST), OPP. S.R.NAGAR BUS STOP, HYD - 38

UNIT -IV

C & DS

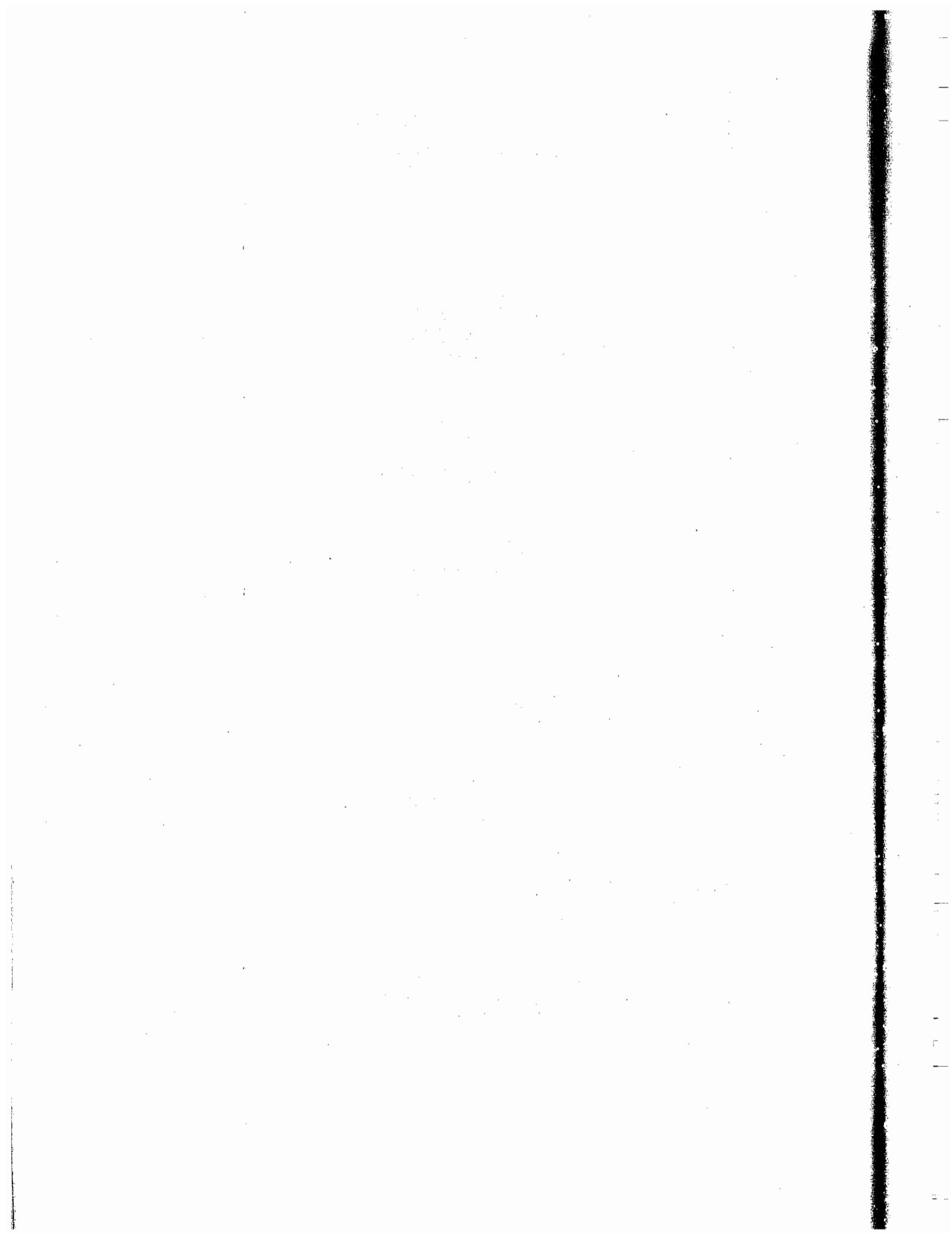
By

Mr. Kalyan Reddy

Kalyan

Ph: 6662 2789, 6662 3789

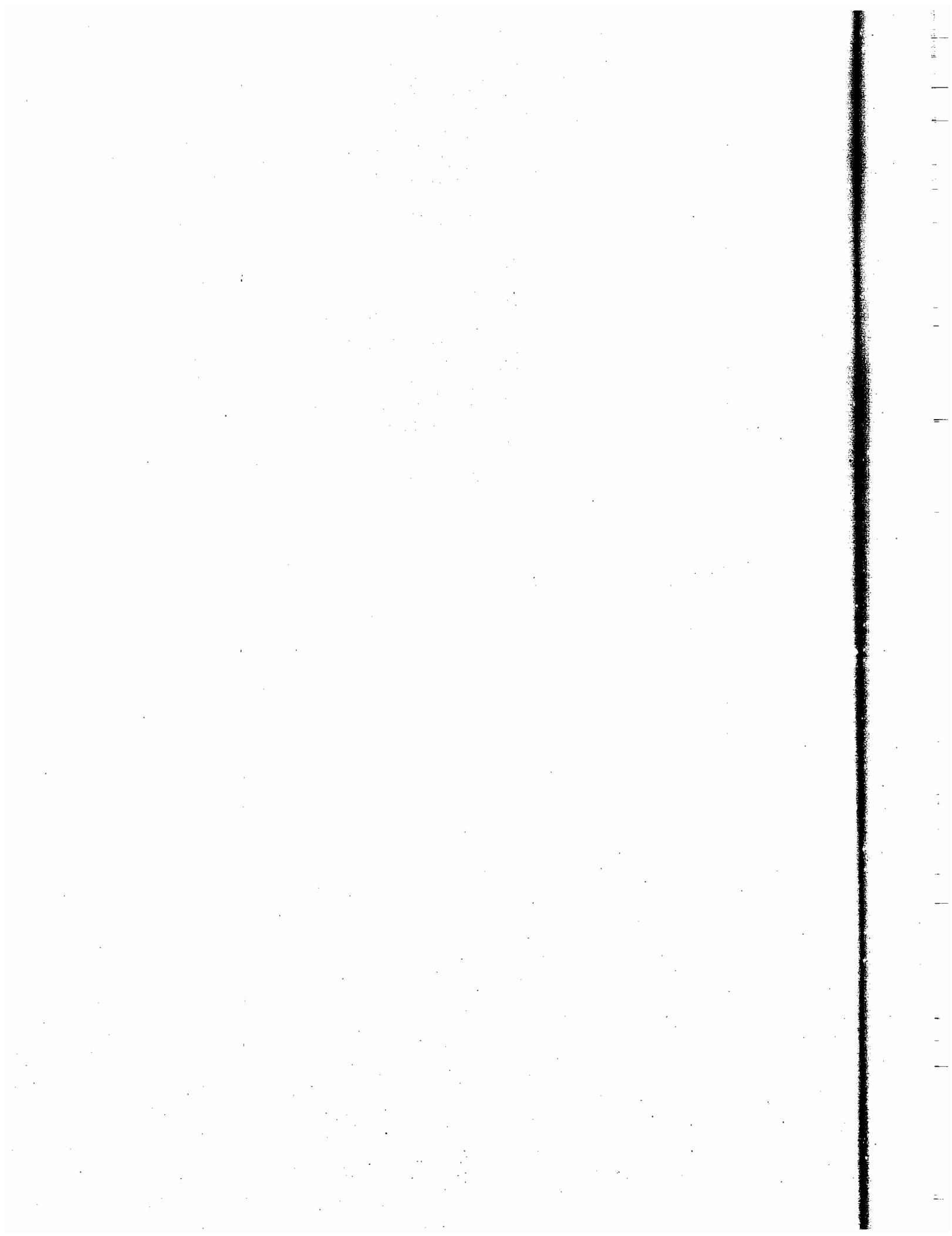
www.kalyanit.com



UNIT - IV (Structures)

<u>Topic</u>	<u>Page No</u>
Defination	83
Declaration & initialization	83
Arrays of structures	84
Structures with in structures	85
Pointers to structures	86
Structures & functions	86
Self referential structures	87
Union	87
Typedef	88
Bit fields	88
Miscellaneous	89

Kalyan



Kalyan IT Training (Hyd) Pvt Ltd

Structures

Definition: Structure is a collection of one or more variables of different data types, grouped together under a single name. By using structures we can make a group of variables, arrays, pointers etc.

Features:

- a. It is possible to copy the contents of all structure elements of different data types to another structure variable of its type using assignment (=) operator. It is possible because the structure elements are stored in successive memory locations.
- b. Nesting of structures is possible i.e. one can create structure within the structure.
- c. It is possible to pass structure elements to a function.
- d. It is also possible to create structure pointers.

Declaration and Initialization:

1. Struct declaration always starts with keyword **struct** and is enclosed within a pair of curly braces. Using struct and tag (struct_type) user can declare structure variables like variable1, variable2 and so on. (structure declaration using tag name is optional)

```
struct struct_type
{
    Type variable1;
    Type variable2;
};
```

2. after defining structure we can create variables:**struct struct_type v1,v2;**

3. the memory allocation takes places only when variables are declared.

```
struct book1
{
    Char book[30];
    Int pages;
    Float price;
};
struct book1 bk1;
```

4. to access the structure members the **period (.)** sign is used.

```
bk1.book="srinivas";
bk1.pages=500;
bk1.price=385.00;
```

5. Like any other data Type, a structure variable can be initialized.

```
struct emp
{
```

Kalyan IT Training (Hyd) Pvt Ltd

```
int id;
char name[25];
int sal;
} e1={101, "RAJESH",10000};
```

Another method is to initialize a structure variable inside the *main* function.

```
struct emp
{
    int id;
    char name[25];
    int sal;
} e1={101, "RAJESH",10000};
void main ()
{
    struct emp e2={108,"JEEVAN",5500};
    .....
    .....
    .....
}
```

Note that C language does not permit the initialization of individual structure members within structure. The initialization must be done only in the declaration of actual variables.

Array of structures: array is a collection of similar data types. In the same way we can also define array of structures. In such type of array elements is of structure type.

1. In the below example t[3] is an array of 3 elements containing three objects of time structure.

```
struct time
{
    int second;
    int minutes;
    int hour;
} t[3];
```

2. We use structure to describe the format of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases, we many declare an array of structures, each element of the array representing a structure variables. For example

```
struct class student[50];
```

Defines an array called student that consists of 50 elements. Each element is defined to be of the type struct class. Consider the following declaration:

```
struct class
{
    int subject1;
    int subject2;
    int subject3;
};

void main( )
{
    struct class students[3]={ 
        {75,85,95},
        {95,75,85},
        {99,88,77}
    };
}
```

This declaration the student as an array of three elements student[0], student[1], and student[2] and initializes their member as follows:

```
student[0].subject1=75;
student[0].subject2=85;
student[0].subject3=95;
.....
.....
student[2].student3=77;
```

Note that the array is declared just as it would have been, with any other array. Since student is an array, we use the usual array-accessing methods to access individual elements and then the member operator to access member.

Structures within structures: we can take any data type for declaring structure members like int,float,char etc. in the same way we can also take object of one structure as member in another structure. Thus, structure within structure can be used to create complex data applications. Structures within a structure mean nesting of structures. Nesting of structures is permitted in C.

```
struct time
{
    int second;
    int minute;
    int hour;
};

struct t
{
    int car;
```

Kalyan IT Training (Hyd) Pvt Ltd

```
    struct time st;
    struct time et;
};
```

Pointers to structure: we know that pointer is a variable that holds the address of another data variable. The variable may be of any data type i.e. int, float or double. In the same way we can also define pointer to structure. Here, starting address of the member variables can be accessed. Thus, such pointers are called structure pointers.

Ex: struct book

```
{  
    Char name[25];  
    Char author[25];  
    Int pages;  
};  
Struct book *ptr;
```

- In the above example *ptr is pointer to structure book. The syntax for using pointer with member is as:
1) Ptr -> name 2) ptr -> author 3) ptr -> pages.

Structures and functions: like variables of standard data type structure variables also can be passed to the function by value or address.

```
Ex: /* passing address of structure variable */  
#include <stdio.h>  
#include <conio.h>  
struct book {  
    char name[35];  
    char author[35];  
    int pages;  
};  
  
void main()  
{  
    struct book b1={"JAVA COMPLETE REFERENCE","P.NAUGHTON",886};  
    show(&b1);  
}  
  
show(struct book *b2)  
{  
    clrscr();  
    printf("\n %s by %s of %d pages ",b2->name,b2->author,b2->  
    pages);  
}
```

There are different ways of passing structure type arguments to functions. These are given below:

Kalyan IT Training (Hyd) Pvt Ltd

1. Passing structure members individually: each member of structure can be passed individually as arguments in the function call and retained through return statement.
2. A copy of complete structure can be passed as argument using call by value method: using this method, a copy of entire structure is passed to function but any modifications made to it will not be reflected in the called function.
3. Pass the address of structure to the called function; this method is more efficient than the second method because structure need not be returned to called function. In this approach, the address location of the structure is passed to the called function.

Self referential structures:

1. a self-referential structure is one that includes within its structure at least one member which is a pointer to the same structure type.
2. Self referential structures are mainly used in the implementation of data structures. Ex: linked lists, trees, etc.
3. **Ex:** struct student
{
 char name[50];
 int rollno;
 struct student *ptr;
};

Union

Union is a variable, which is similar to the structure. It contains number of members like structure but it holds only one object at a time. In the structure each member has its own memory location whereas, members of unions have same memory locations. It can accommodate one member at a time in a single area of storage. Union also contains members of types int,float,long,arrays,pointers etc. it allocates fixed specific bytes of memory for access of data types irrespective of any data type.

Union requires bytes that are equal to the number of bytes required for the largest members. For example the union contains char,integer and long integer then the number of bytes reserved in the memory for the union is 4 bytes.

Declaration: union <tag> //tag is optional
{
 Datatype member1;
 Datatype member2;

 Datatype member n;
};

Access: accessing of members of union is done similar to that of structure, it is done by using dot(.) operator for unions and using '->' for pointer to unions.

Kalyan IT Training (Hyd) Pvt Ltd

```
Example: union organization
{
    Char name[25];
    Char designation[10];
    Int sal;
};

Union organization emp1={"ramu","secretary".8000};
Union organization emp2={"gopal","manager".15000};
```

TypeDef: By using typedef we can create new data type (Eg: int, pointers and structures). The declaration for the typedef keyword is:

```
typedef <datatype> <newname>;
```

Ex: a) creating new data type 'int'

```
TypeDef int integer;
```

Now integer is the synonym for 'int'. integer can be used in declarations.

Ex: integer x,y,z;

b) using typedef for pointers.

```
TypeDef int * integer;
```

This statement makes integer as a data type which is similar to integer pointer (int *). Now this integer can be used for the declaration of integer pointer. Ex: integer P1;

c) using typedef for structures.

```
TypeDef struct student * st_pointer
```

Struct student

```
{
    Char name[50];
    Int rollno;
    St_pointer ptr;
}
```

Bit-fields: bit field provides exact amount of bits required for storage of values. A bit-field is a set of adjacent bits within a single machine word. Bit-fields are mainly used to save the memory space or to combine several states of information within a single machine word.

To hold the information we use the variables. The variables occupy a minimum of one byte for char and two bytes for integer. Instead of using complete integer if bits are used, space of memory can be saved.

Declaration: declaring structure using bit-fields.

```
Struct student
```

```
{
    Unsigned gender : 1;
```

```
    Unsigned grade : 2;  
};
```

The colon indicates the usage of bit-fields and the number following it represents (tells to compiler) the width of the field in bits.

However there are restrictions on bit fields when arrays are used. Arrays of bit fields are not permitted. Also the pointer cannot be used for addressing the bit field directly, although the use of the member access operator (->) is acceptable.

Miscellaneous:

- a. Enumerated date type:** the **enum** is a keyword. It is used for declaring enumeration types. The programmer can create his/her own data type and define what values the variables of these data types can hold. This enumeration data type helps in reading the program.

```
enum month {jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec};
```

This statement creates user defined data type. The enumerators are the identifiers jan, feb, mar, apr and so on. Their values are constant unsigned integers and start from 0. The identifier jan refers to 0, feb to 1 and so on.

The identifiers are not to be enclosed with in quotation marks. Please also note that integer constants are also not permitted.

b. Advantages of structures over arrays:

1. We can group items of different types within a single entity, which is not possible with arrays, as array stores similar elements.
2. The position of a particular structure type variable within a group is not needed in order to access it, whereas the position of an array member in the group is required, in order to refer to it.
3. In order to store the data about a particular entity such as a 'book', using an array type, we need three arrays; one for storing name, another for price and third one for number of pages etc., hence, the overhead is high. This overhead can be reduced by using structure type variable i.e. array of structures.
4. Once a new structure has been defined, one or more variables can be declared to be of that type.
5. A structure type variable can be used as a normal variable for accepting the user's input, for displaying the output etc.
6. The assignment of one 'struct' variable to another, reduces the burden of the programmer in filling the variable's fields again and again.
7. It is possible to initialize some or all fields of a structure variable at once, when it is declared.
8. Structure type allows the efficient insertion and deletion of elements but arrays cause them inefficiently.
9. For random array accessing, large hash tables are needed. Hence, large storage space and costs are required.
10. When a structure variable is created, all of the member variables are created automatically and are grouped under the given variable's name.

Kalyan IT Training (Hyd) Pvt Ltd

c. Structure Vs Union:

Structure	Union
<p>1. Declaration Struct <tag> { Data type member1; Data type member2; };</p> <p>2. Every structure member is allocated memory when a structure variable is defined. Ex: struct { Char c; Int x; Float y; }s; Memory allocated for s is 7 (1+2+4) bytes.</p> <p>3. All the members can be assigned values at a time.</p> <p>4. Values assigned to one member will not cause the change in other members.</p> <p>5. All structure variables can be initialized at a time.</p> <p>6. The usage of structure is efficient when all members are actively used in the program.</p>	<p>1. Declaration union <tag> { Data type member1; Data type member2; };</p> <p>2. the memory equivalent to the largest item is allocated commonly for all members. Ex: union <tag> { Char c; Int x; Float y; }u; Memory allocated to us is 4 bytes.</p> <p>3. Only one member can be assigned value at a time.</p> <p>4. value assigned to one member may cause the change in value of other members.</p> <p>5. only one union member can be initialized at a time.</p> <p>6. the usage of union is efficient when members of it are not required to be accessed at the same time.</p>

- d. **sizeof:** We normally use structures, unions, and arrays to create variables of large sizes. The actual size of these variables in terms of bytes may be change from machine to machine. We may use the unary operator 'sizeof' to tell us the size of a structure. The expression **sizeof(struct x)** will evaluate the number of bytes required to hold all the member of the structure x.

Kalyan IT Training Pvt., Ltd., Hyderabad

2ND FLOOR, SRINIVAS NAGAR (WEST), OPP. S.R.NAGAR BUS STOP, HYD - 38

UNIT -V

C & DS

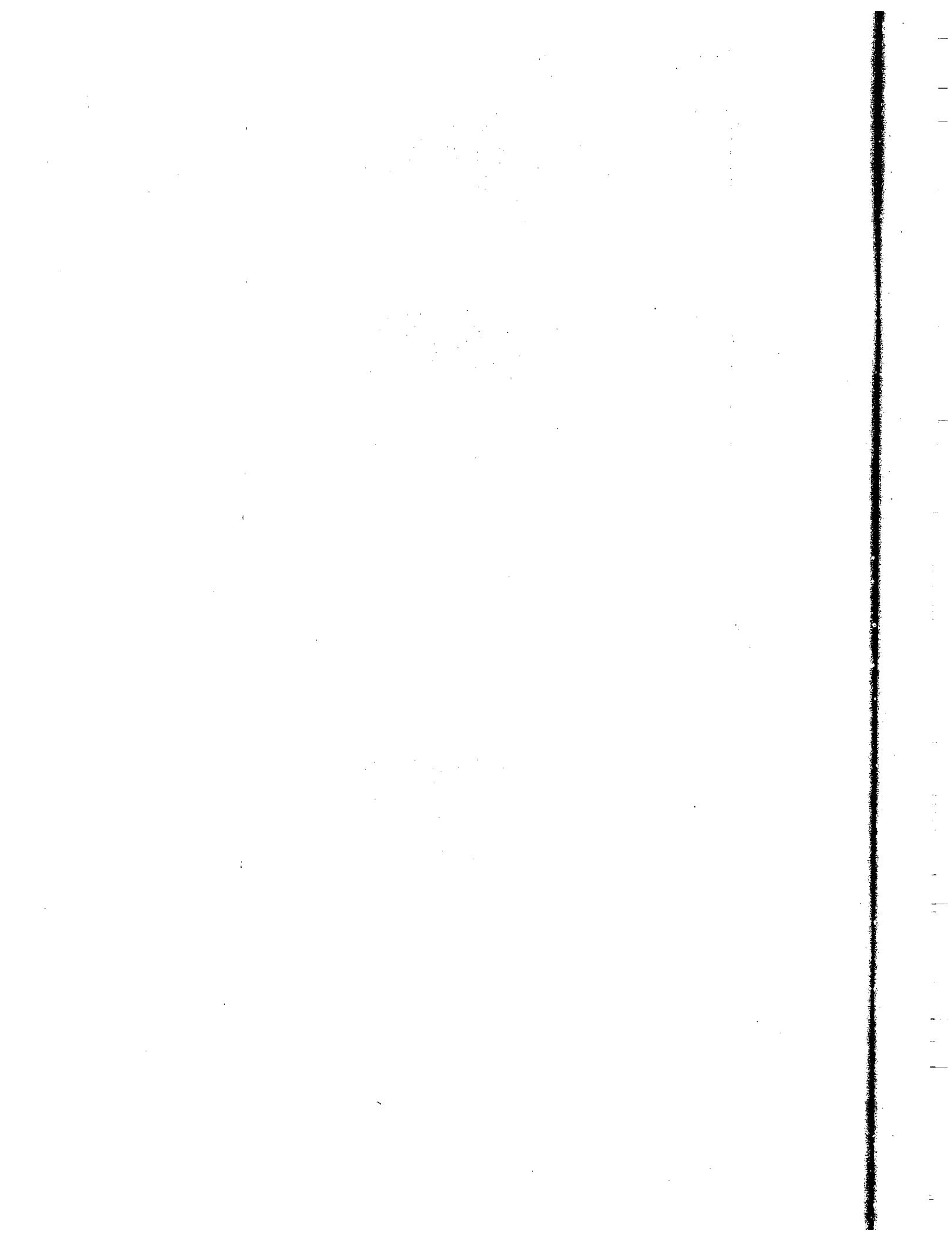
By

Mr. Kalyan Reddy

Kalyan

Ph: 6662 2789, 6662 3789

www.kalyanit.com

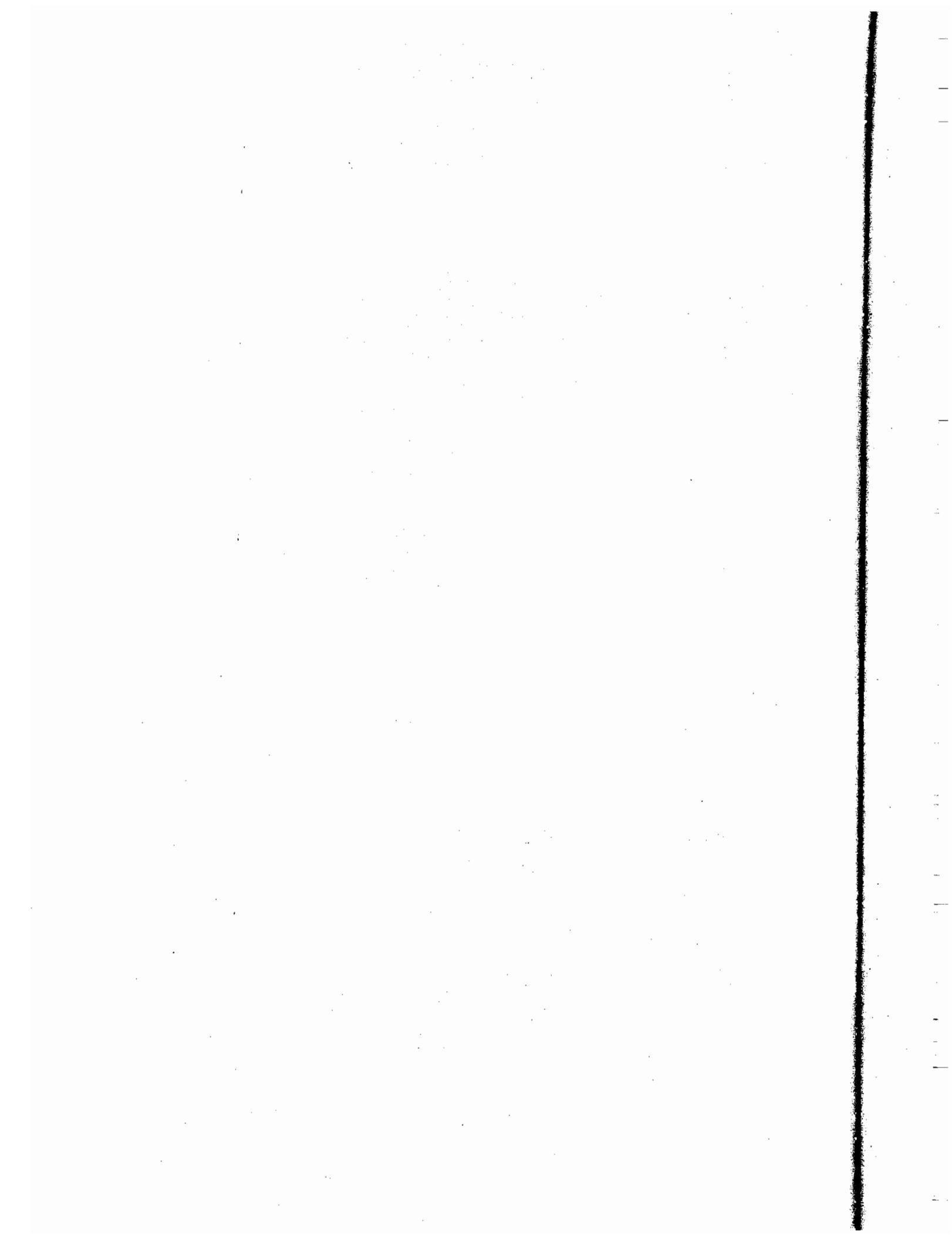


Kalyan IT Training (Hyd) Pvt Ltd

UNIT - V (Files)

<u>Topic</u>	<u>Page No</u>
Definition	91
Basic operations on file	91
Files modes	92
Binary modes	93
File functions	93
Streams	96
Types of files	96
Ios	97
Error handling in c	97

Kalyan



Kalyan IT Training (Hyd) Pvt Ltd

Files

Definition: File is a set of records that can be accessed through the set of library functions.

Basic operations on a file: the basic operations that are performed on the file are,

- Opening a file
- Reading or writing a file
- Closing file.

And also need to fulfill three important points:

- Naming of file
- Data structures of file
- Purpose of file.

1. Naming of file: the file name must be specified for a particular file. The file name is typically a string of characters. If the file is used for storing text we use '.txt', for C programs we give '.c', for header file we give '.h' extension and so on.
Ex: "data.txt", "add.c" or "pr1.h" and so on.

2. Data structure of file: FILE is the data type defined for a file.

Syntax: FILE pointer type.

Ex: FILE *p1; where 'p1' is the pointer to the FILE structure.

3. Purpose of File: here the file should be defined whether the file is to be read, written or appended by defining the modes.

- i. Opening file: file is opened by open() function with two parameters 1. File name, 2. File open mode. For opening a file, request has to be made to operating system. The operating system grants the request by pointing to the structure 'FILE' and rejects the request by returning a 'NULL' values.

Syntax: FILE fp;
fp=fopen(file_name, mode);

Ex: FILE *fp;
Fp=fopen("file.txt","r");

The fopen() function is defined in the "stdio.h" header file. The "file_name" parameter refers to any name of the file with an extension such as "file.txt" or "abcd.c" and so on. File open mode refers to the mode of opening the file. The file can be opened in 'read mode', 'write mode' or 'append mode' and also it can be opened with a combination of any two modes out of these three.

- ii. Reading or writing a file: once the file is opened the associated pointer points to the starting character of the file contents. This file can be read wholly by using various functions provided by the 'C' standard library (Eg: fgetc()); fgetc() reads the character pointed by fp, and the pointer position gets incremented, 'ch' holds the character read from the file.

Kalyan IT Training (Hyd) Pvt Ltd

For writing data into the opened file we can use the fputc() function.
Ex: fputc (ch, fp); where 'ch' is the character to be written into the file.

- iii. Closing a file: after the required operations such as reading, writing or appending operations on a file are done, the file needs to be closed with the associated pointer.

Syntax: fclose (file_pointer);
Ex: fclose (fp1);

To close more than one file at a time, we can use fcloseall() function.

File Modes: a. Text modes b. binary modes

- a. **Text modes:** write, read, append, write+read, append+read, read + write modes available in text modes.

i.write (w): this mode opens a new file on disk for writing. If the file already exists, it will be over written without confirmation.

Ex: fp=fopen ("data.txt", "w");

ii.read (r): this mode opens a pre-existing file for reading. If the file doesn't exist, then compiler returns NULL to the file pointer.

Ex: fp=fopen ("data.txt", "r");
If (fp==NULL)
 Printf("File does not exist");

iii.append (a): this mode opens a pre-existing file for appending data. If the file doesn't exist then new file is opened i.e. if the file does not exist then the mode of "a" is same as "w".

Ex: fp=fopen ("data.txt","a");

iv.write+read (w+): it searches for file, if found its contents are destroyed. If the file isn't found a new file is created. Returns NULL if fails to open the file. In this mode file can be written and read.

Ex: fopen("data.txt","w+");

v.append+read (a+): in this mode file can be read and records can be added at the end of file.

Ex: fp=fopen("data.txt","a+");

vi.read+write (r+): this mode is used for both reading and writing. We can read and write the record in the file. If the file does not exist, the compiler returns NULL to the file pointer.

Kalyan IT Training (Hyd) Pvt Ltd

```
Ex: fopen ("data.dat","r+");  
    If(fp==NULL)  
        Printf("File not found");
```

b. Binary modes:

i.wb (write): this mode opens a binary file in write mode.

```
Ex: fp=fopen("data.dat","wb");
```

ii.rb (read) : this mode opens a binary file in read mode.

```
Ex: fp=fopen("data.dat","rb");
```

iii.ab (append) : binary file opened in append mode i.e. data can be added at the end of file.

```
Ex: fp=fopen("data.dat","ab");
```

iv.r+b (read+write) : this mode opens a pre-existing file in read and write mode. i.e. file can be read and written.

```
Ex: fp=fopen("data.dat","r+b");
```

v.w+b (read+write) : this mode creates a new file in read and write mode.

```
Ex: fp=fopen("data.dat",'w+b');
```

vi.a+b (append+write) : this mode opens a file in append mode i.e. data can be written at the end of file. If file does not exist a new file is created.

```
Ex: fp=fopen("data.dat","a+b");
```

File functions:

Function	Operation
Fopen()	Creates a new file for read/write operation
Fclose()	Closes a file associated with file pointer
Closeall()	Closes all opened files with fopen()
Fgetc()	Reads the character from current pointer position and advances the pointer to next character
Getc()	Same as fgetc()
Fprintf()	Writes all types of data values to the file
Fscanf()	Reads all types of data values from a file
Putc()	Writes character one by one to a file
Fputc()	Same as putc()
Gets()	Reads an integer from the file
Puts()	Writes string to the file
Putw()	Writes an integer from the file
Getw()	Reads an integer from the file
Fread()	Reads structured data written by fwrite() function

Kalyan IT Training (Hyd) Pvt Ltd

Fwrite()	Writes block of structured data to the file
Fseek()	Sets the pointer position anywhere in the file
feof()	Detects the end of file
Ferror()	Reports error occurred while read/write operations
Perror()	Prints compilers error messages along with user defined messages
Ftell()	Returns the current pointer position
Rewind()	Sets the record pointer at the beginning of the file
Unlink()	Removes the specified file from the disk
Rename()	Changes the name of the file

File handling functions: file handling functions are used to handle users files efficiently. These functions are part of "C" standard library. File handling functions are mainly used for opening, reading or writing, closing the file. Some of these functions are:

- a. Fopen(): this function is used to create new file for reading/writing purpose.
Ex: fp=fopen(file,"r");
- b. Fclose(): this function closes the file with associated points. This closes one file at a time.
Ex: fclose(fp);
- c. Fscanf(): it is used for reading all types of data from a file associated with a file pointer.
Ex: fscanf(fp,"%s%d",name,&number);
- d. Fprintf(): this function is used for writing all types of data to the file with associated file pointer.
Ex: fprintf(fp,"%s%d",name,number);
- e. Fseek(): it positions file pointer on the stream. We can pass three arguments through this function. 1. File pointer 2. Negative or positive long integer number to reposition the file pointer towards the backward or forward direction. 3. The current position of file pointer.

Integer value	Constant	Location in the file
0	SEEK_SET	Beginning of the file
1	SEEK_CUR	Current position of the file pointer.
2	SEEK_END	End of the file

- Ex: 1. fseek(fp,10,0) or fseek(fp,10,SEEK_SET)
2. fseek(fp,-n,SEEK_END) → to read from backward direction from End of file
- f. feof(): the macro feof() is used for detecting the file pointer whether it is at the end of file or not. It returns a non-zero if the file pointer is at the end of file, otherwise it returns zero. This feof() function is a binary function that helps in detecting I/O errors in the files.
Syntax: feof(file_pointer);
Ex: if(feof(fp));
 printf ("reached end of file\n");

Kalyan IT Training (Hyd) Pvt Ltd

File I/O functions: these are used for various I/O operations that take place on files.

- a. Fprintf(): used for writing characters, strings, integers, floats etc. to the file. It contains the pointer parameter, which points the opened file.
Syntax: fprintf(file_pointer,"format string", list);
Ex: fprintf(R1,"%d %s %f",age,name,marks);
- b. Fscanf(): this function reads character, strings, integer, floats etc. from the file pointed by file pointer. This function analogous to the scan() function except that it works on files.
Syntax: fscanf(file_pointer,"format string", list);
Ex: fprintf(R1,"%d %s %f",&age,&name,&marks);
- c. Getc(): this function reads single character from the opened file and moves the file pointer. It returns EOF, if end of file is reached. The getc() function can handle only one character or integer at a time.
Syntax: getc(file_pointer);
Ex: getc(f1);
- d. Putc(): this function is used to write a single character into a file opened in write mode. If an error occurs it returns EOF. This function can also handle only one character or integer at a time.
Syntax: putc(character variable, file_pointer);
Ex: putc(a, f1);
- e. Fgets() – fputs(): fgets() function reads string from a file pointed by file pointer. It also copies the string to a memory location referred by an array. Fputs() is useful when we want to write a string into the opened file.
- f. Putw(): this function is used to write an integer value to file pointed by file pointer.
Syntax: putw(variable, file_pointer);
Ex: putw(I,fp);
- g. Getw(): this function returns the integer value from a file and increases the file pointer. Means reads an integer from a file.
Syntax: getw(file_pointer);
Ex: getw(fp);
- h. Fwrite() – fread(): fwrite() function is used for writing an entire structure block to a given file. Fread() function is used for reading an entire block from a given file.

File Error handling functions: while any I/O operations are performed there is a possibility of an error to occur. Some of the common situation where an error can occur are,

1. When read operation is performed though end of file is reached.

2. when a operation is being performed on a file which is not opened.
3. when a file is opened using incorrect name.
4. when a operation is performed on a file, but that file is opened to perform some other operation.

Kalyan IT Training (Hyd) Pvt Ltd

5.when a write operation is performed on a write protected file.

In these situations the program terminate exceptionally which causes deviation from the expected output or may lead to early termination of a program. In order to handle such situation two library function feof() and ferror() are used to detect I/O error.

- a. Ferror(): this function is used to find out error when the file read/write operation is carried out. It takes file pointer as its argument that returns zero if read/write operation is performed successfully and non-zero value if read/write operation is terminated unsuccessfully.

Ex: if (ferror(fp) !=0)

```
    Printf("there is an error in file");
```

- b. Ftell(): it returns the current position of the file pointer. It returns the pointer from the beginning of file.

Syntax: n=f.tell(fp); here 'n' gives the current position of file pointer in terms of bytes from beginning.

- c. Rewind(): this function resets the file pointer at the beginning of the file.

Syntax: rewind (fp);

- d. Remove() or unlink(): these functions delete the given file in the director. It is similar to 'del' command in dos.

- e. Rename(): this function changes the name of the given file. It is similar to dos command 'rename'.

Streams: stream means reading and writing of data. If the flow is from an input device then the stream is called an input stream and if the flow is to the output device then the stream is called an output stream. Stream is nothing but reading/writing data from/to the file. With the help of stream users can access the files efficiently. All information regarding streams are stored in a object file.

Types of files: there are two types of files 1.sequential 2.random access file.

1. Sequential file: In this type data is kept sequentially. If we want to read the last record of the file we need to read all the records before that record. It takes more time. Or if we desire to access the 10th record then the first 9 records should be read sequentially for reaching to the 10th record.

Sequential access of files is achieved using functions fprintf(), fscanf(), getc(), putc() etc.

2. Random access file: in random access file, records are stored regardless of the order and can be retrieved or modified randomly. If a read operation is to be performed on 'n' th record, then there is no need to access the first (n-1) records. The 'n' th record can be accessed directly because of which a lot of time is saved.

Kalyan IT Training (Hyd) Pvt Ltd

Random access of files is achieved with the help of functions fseek(), ftell() and rewind() are available in I/O library.

I/Os: In C, I/O functions are generally classified as formatted I/O and low-level I/O.

1. Formatted I/O: the I/O functions which are dependent on the C runtime library are known as formatted I/O.

The formatted I/O is independent of operating system, but uses C standard library. The process of interaction of I/O function with operating system is hidden in formatted I/O. the advantages of formatted I/O is the portability. A program using formatted I/O can be easily ported to another operating system.

The formatted I/O are also known as high-level I/O.

2. Low-level I/O: The I/O operations which directly use the system calls are known as low-level I/O, there is a direct interaction with the operating system, thus increasing the speed of I/O function. But the disadvantage of low-level I/O function is that the program using low-level I/O cannot be easily ported to another operating system.

A user has to use a buffer to carry out low-level I/O operations.

3. Standard I/O: the standard I/O files are standard input, standard output and standard error. These files are automatically opened by the operating system when a 'C' program is standard.

The standard input file is for the input from the keyboard. The standard output file is for the output on the screen. The standard error file is for alerting the error on the screen. A user can change input, output, error file for a particular program. The standard file are handled through 'C' library functions scanf(), printf() etc.

Error Handling in C: C does not support any special error handling constructs as that of certain advance programming languages like C++ and java. In general "error" refers to a "cause" which causes deviation from the expected output. Errors are classified into two types:

1. Runtime errors: the errors which get reflected while a given 'C' code is under execution process are runtime errors. These errors cause abnormal results. Following are frequently occurring runtime errors:
 - a. Stack overflow
 - b. Operation with invalid pointer
 - c. Floating point underflow/overflow
 - d. Floating point divide to zero/integer divide by zero
 - e. Log or ln of negative number of zero

Kalyan IT Training (Hyd) Pvt Ltd

- f. Attempt to determine tangent of 90
 - g. Square root of negative number
 - h. Array reference out of bounds
 - i. Number being too small or too large to convert to integer etc.
2. **Compile time errors:** all the errors which gets reflected on the screen after the compilation of the program are compile time errors. They indicate errors due to "syntax violation" or "data type mismatch" etc. These errors can be easily managed since they are descriptively revealed by the compiler.

It is quite tedious process to sort out such errors, and often they are not revealed even though the program gets successfully executed several times.

Miscellaneous:

Text vs binary modes

A text file contains only textual information like alphabets, digits and special symbols. The ASCII code of these characters are stored in text files.

A binary file is a collection of bytes. This collection is a compiled version of a C program. E.g.: program1.exe, or music data stored in a wave file or a picture stored in a graphic file.

Printf vs fprintf

Printf	Fprintf
<ul style="list-style-type: none">1. The printf() function is used for printing all types of data values to the console.2. Printf() function does not have a file pointer.	<ul style="list-style-type: none">1. Fprintf() function is used for printing all types of data values to the file.2. Fprintf() function contains a parameter called file pointer which points the opened file.

Getchar vs scanf

Scanf()	Getchar()
<ul style="list-style-type: none">1. It can accept multiple characters at a time.2. Entering of each character should be followed by pressing of enter key3. Continuous stream of characters	<ul style="list-style-type: none">1. It can accept only single character at a time.2. Pressing of enter key is not required.3. Continuous stream of characters can be directly supplied using

<ul style="list-style-type: none"> cannot be directly supplied using scanf function 4. It uses '%s' as an argument followed by a reference location variable in its syntax representation. 5. Scan function can be used to explicitly provide data at the execution time irrespective of its type 	<ul style="list-style-type: none"> getchar function. 4. It does not use '%s' as an argument in its specification. 5. Getchar() function is dedicated to be used only with character type.
--	--

feof vs ferror

Feof	Ferror
<ul style="list-style-type: none"> 1. Feof() is a macro that determines whether the file pointer is at the end of file or not. 2. Feof() returns a zero if end of file is not reached and returns a non-zero when the end of file is reached. 3. Example of feof(): <pre>While(!feof(fp)) { Printf("%c", c); C=getc(fp); }</pre> When, feof() returns a zero if end of file is not reached, then the '!' operator is used to negate this zero to the truth value. 	<ul style="list-style-type: none"> 1. Ferror() is a standard library function which finds the error when read/write operations are carried out in a file. 2. Ferror() returns a zero if read/write operation is successful and a non-zero value in case of failure of read/write operation. 3. Example of ferror(): <pre>While(!feof(fp)) { C=fgetc(fp); If(ferror()) { Printf("Error"); Break; } }</pre>

