Collection

- **Collection**: If we want to represent a group of individual objects as a single entity then we should go for collection.
- It is used to store, retrieve, manipulate and communicate collective data.
- **Collection Framework**: It defines several classes and interfaces which can be used a group of objects as single entity.

Arrays	Collections
Arrays are in fixed size.	Growable in nature.
For storing elements array performance is good.	Performance is not so good.
It holds only homogenous data.	It holds homogenous and heterogenous data.
It holds primitives as well as objects.	It holds only objects.

List

1. **ArrayList**: It is child interface of List. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged.
    - Constructors:
        1. ArrayList()
        2. ArrayList(Collection c)
        3. ArrayList(int capacity)
    - Methods:
        1. boolean add(Object o)
        2. void clear()
        3. Object clone()
        4. boolean contains(Object o)
        5. int size()
        6. boolean isEmpty()

7. Boolean remove(Object obj)

2. **LinkedList**: It is child interface of list. Insertion order is preserved.
    - Constructors:
        1. LinkedList()
        2. LinkedList(Collection c)
    - Methods:
        1. boolean add(Object o)
        2. void addFirst(Object o)
        3. void addLast(Object o)
        4. void clear()
        5. Object getFirst()
        6. Object getLast()
        7. Object removeFirst()
        8. Object removeLast()
        9. int size()

Stack

- Stack is a subclass of Vector that implements a standard LIFO stack. Stack only defines the default constructor, which creates an empty stack. Stack includes all the methods defined by Vector, add adds several of its own.
- Constructors:
    1. stack()
- Methods:
    1. boolean empty()
    2. Object peek()
    3. Object pop()
    4. Object push(Object element)
    5. int search(Object element)

Vector

- Vector implements a dynamic array. It is similar to ArrayList, but with few differences -
    1. Vector is synchronized.
    2. Vector contains many legacy methods that are not part of the collections framework.
    3. Vector proves to be very useful if you don’t know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.
- Constructors:
    1. Vector()
    2. Vector(int size)
    3. Vector(int size, int incr)
- Methods:
    1. void add(int index, Object element)
    2. boolean add(Object o)
    3. boolean addAll(Collection c)
    4. void addElement(Object obj)
    5. int capacity()
    6. void clear()
    7. Object firstElement()
    8. Object lastElement()

Set

- It is child interface of collection. Duplicates are not allowed in set and insertion order is preserved.
- Set does not contains new methods, so we have to use methods of collection interface.

1. **HashSet**:

- A hash table stores information by using a mechanism called hashing. In hashing, the informational content of a key is used to determine a unique value, called its hashcode.
- The underlying data structure is Hash Table.
- Duplicates are not allowed.
- Insertion order is not preserved and objects will be inserted on hash-code of objects.
- Heterogeneous objects are allowed.
- Null insertion is possible.
- Constructor:
    1. HashSet h = new HashSet();
    2. HashSet h = new HashSet(int initial capacity);
    3. HashSet h = new HashSet(initial capacity, float loadfactor);
    4. HashSet h = new HashSet(Collection c);
- Methods:
    1. boolean add(Object o)
    2. void clear()
    3. boolean contains(Object o)
    4. boolean isEmpty()
    5. Iterator iterator()
    6. boolean remove(Object o)
    7. int size()

2. **LinkedHashSet**:
    - It is child class of HashSet.
    - It is exactly same as HashSet except following differences.
    - It is the best choice to develop cache based applications, where duplicates are not allowed and insertion order must be preserved.

HashSet	LinkedHashSet

HashSet	LinkedHashSet
Underline data structure is HashTable.	Underlying data structure is LinkedList + HashTable.
Insertion order is not preserved.	Insertion order is preserved.
Introduction in 1.2 verison.	Introduced in 1.4 verison.

- Constructors:
    1. LinkedHashSet()
    2. LinkedHashSet(Collection c)
    3. LinkedHashSet(int capacity)
    4. LinkedHashSet(int capacity, float fillratio)

3. **TreeSet**:
    - The underlying data structure for TreeSet is Balanced tree.
    - Duplicate objects are not allowed.
    - We can insert NULL element but only once.
    - Insertion order is not preserved but objects are inserted in some sorted order.
    - Heterogeneous objects are not allowed.
    - Constructors:
        1. TreeSet()
        2. TreeSet(Collection c)
        3. TreeSet(Comparator comp)
        4. TreeSet(SortedSet ss)
    - Methods:
        1. void add(Object o)
        2. void clear()
        3. boolean contains(Object o)
        4. Object first()
        5. boolean isEmpty()

6. Object last()
7. int size()

Map

- Map is not the child interface of collection.
- Key value do not repeats where as, value part may contains duplicate values.
- Each key-value pair is considered as entry, hence map is called as collection of entry objects.

1. **HashMap**:
    - Underlying data structure is HashTable.
    - Duplicates key are not allowed but values can be duplicated.
    - Insertion order is not preserved.
    - Heterogeneous objects are allowed for both key and values.
    - Null is allowed for key(only once),for values (any no of times).
    - It implements serializable and clonable interfaces.
    - Efficient for search operation.
    - Constructors:
        1. HashMap()
        2. HashMap(int capacity)
        3. HashMap(int intitial Capacity, float loadFactor)
    - Methods:
        1. Object put(Object key, Object value)
        2. void putAll()
        3. void clear()
        4. Set entrySet()
        5. boolean isEmpty()
        6. int size()
2. **LinkedHashMap**:

- This class extends HashMap and maintains a linked list of the entries in the map, in the order in which they were inserted.
- When iterating a LinkedHashMap, the elements will be returned in the order in which they were inserted.
- Constructors:
    1. LinkedHashMap()
    2. LinkedHashMap(int capacity)
    3. LinkedHashMap(int capacity, float fillRatio)

3. **HashTable**:
    - Like HashMap, Hashtable stores key/value pairs in a hash table.
    - When using a Hashtable, you specify an object that is used as a key, and the value that you want linked to that key.
    - Only one thread can apply.
    - Null is not allowed
    - Constructors:
        1. Hashtable()
        2. Hashtable(int size)
        3. Hashtable(int size, float fillRatio)
    - Methods:
        1. void clear()
        2. boolean isEmpty()
        3. Object put(Object key, Objectvalue)
        4. int size()
4. **TreeMap**:
    - The TreeMap class implements the Map interface by using a tree.
    - A TreeMap provides an efficient means of storing key/value pairs in sorted order, and allows rapid retrieval.
    - Constructors:
        1. TreeMap()

2. TreeMap(Comparator comp)
3. TreeMap(SortedMap sm)

- Methods:
  1. void clear()
  2. Compartor comparator()
  3. Object get(Object key)
  4. int size()

HashMap	HashTable
Every method is not synchronous.	Every method is synchronous.
Mutli-thread can apply, not a thread safe.	Only one thread can apply, thread safe.
Performance is high.	Performance is low.
Null is available for key and value.	Null is not allowed.
It is not a legacy class.	It is legacy class.

Cursors of Collections

1. **Enumeration**:
   - It is used to get an element.
   - Methods:
     1. hasMoreElements();
     2. nextElement();
2. **Iterator**:
   - It is an universal cursor i.e. applicable to any class in collection.
   - We can perform read() and remove() operations.
   - We can traverse in only one direction.
   - Methods:
     1. public boolean hasNext()
     2. public object next()
     3. public void remove()

3. **ListIterator**:
    - ListIterator extends Iterator to allow bidirectional traverse of list.
    - It also allows modification of elements.
    - Methods:

For previous	For next	More functionalities
hasPrevious()	hasNext()	add()
previous()	next()	remove()
previousIndex()	previousIndex()	set()

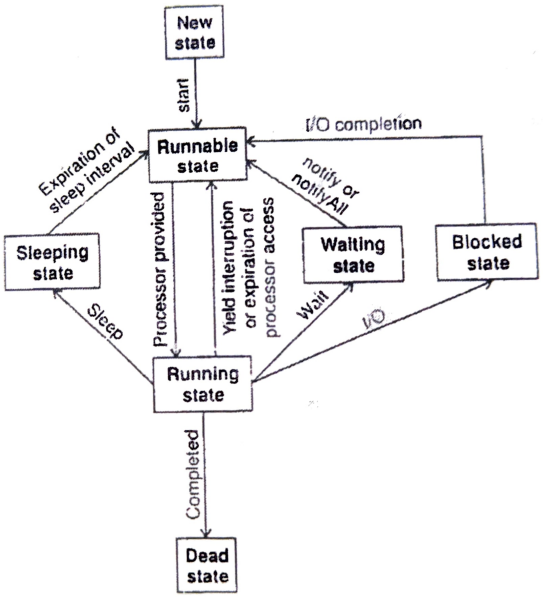
4. **Comparator**:
    - It is used to order the objects of user-defined class.
    - To order elements in ascending and descending, it is very useful interface.
    - Comparator defines two methods as
        a. compare(): It compare two elements for order.
        b. equals(): It shows whether object equals the invoking comparator.

Multithreading

▼ What is a thread?

- A **thread** is a lightweight process that can be executed independently within an application.
- Advantages of multithreading:
  1. CPU idle time is kept to minimum.
  2. Efficient use of I/O resources.
  3. Threads start and stop independently.
  4. Improves the performance of the application.
- Disadvantages of multithreading:
  1. Difficult to code.
  2. Difficult in debugging.
  3. Difficult to manage concurrency and requires precise context switching.
  4. Multithreading support must be provided by the OS.

▼ Thread Lifecycle



- **New state**: When a thread object is created it is in the new state.

- **Runnable**: Thread enters this state when *start()* is called. Here, thread is ready to execute as soon as a CPU is allotted.
- **Running state**: Thread is assigned to a processor and is now running.
- **Sleeping state**: Thread enters sleeping state invoked by *sleep()* and returns when sleep interval ends.
- **Waiting state**: Thread enters waiting state invoked by *wait()* and returns to runnable state when *notify()* or *notifyAll()* is issued.
- **Blocked state**: Thread enters this state when it is waiting for resources acquired by another thread.
- **Dead state**: When a thread has completed its execution.

▼ Implementing Runnable Interface

- java.lang.Runnable interface is used to create a thread. It has only one method *run()*.
- Steps to create a thread using Runnable interface.
    1. Define a class which implements Runnable.
    2. Define run() in the above class.
    3. Create an object of the above class.
    4. Create an object of the thread class and pass the Runnable object to the constructor.
    5. Invoke the start() method on the Thread object to execute the thread.

```
class MyRunnable implements Runnable
{
  public void run()
  {
    for(int i = 1; i <= 5; i++)
      System.out.println("Hello " + i);
  }
}

class RunnableTest
{
  public static void main(String[] args)
  {
    MyRunnable r = new new MyRunnable();
    Thread t = new Thread(r);
    System.out.println(t); //Displays thread information
    t.start();
```

```
  }
}
```

▼ Extending Thread Class

- We can also create thread by extending Thread class.
- java.lang.Thread class implements Runnable interface and contains many methods.
- *run()* method in thread class does nothing. We should overwrite it when extending Thread class.

```
class MyThread extends Thread
{
  public void run()
  {
    for(int i = 1; i <= 5; i++)
      System.out.println("Hello " + i);
  }
}

class ThreadTest
{
  public static void main(String[] args)
  {
    MyThread t = new MyThread();
    System.out.println(t); //Displays thread information
    t.start();
  }
}
```

- Constructors:
  1. Thread()
  2. Thread(Runnable runnOb)
  3. Thread(String name)
- Methods:

Name	Description
String getName()	Returns thread’s name.
int getPriority()	Returns thread’s priority.
void interrupt()	Interrupts thread.
boolean isAlive()	Tests if thread is alive.

Name	Description
void join()	Wait for thread to end.
void setName(String name)	Changes thread name.
void setPriority(int newPriority)	Sets new priority.
void sleep(long mSec)	Causes thread to sleep.
void start()	Begins execution.
String toString()	Returns thread’s name, priority and thread group.
static void yield()	Temporarily pause current thread and allow other thread execution.

▼ Thread Priorities

- Priority is an integer from 1 to 10 inclusive. 10 is the highest priority, 5 is normal and 1 is lowest.
- *getPriority()* method is used to get priority and *setPriority()* is used to set priority.
  - Thread.MIN_PRIORITY : Minimum priority i.e. 1
  - Thread.MAX_PRIORITY : Maximum priority i.e. 10
  - Thread.NORM_PRIORITY : Default priority i.e. 5
- eg: t.setPriority(Thread.MIN_PRIORITY);

▼ The “main” thread

- When a program starts running, a thread called “main” starts running.
- The main threads spawns other threads - called child threads.
- We can obtain the main thread using the *currentThread()*

▼ sleep() Method

- This method causes a thread to sleep for specified number of milliseconds.
- *InterruptedException* is thrown while using this method.

```
class MyThread extends Thread
{
  String message;
  MyThread(String message)
  {
```

```
    this.message = message;
  }
  public void run()
  {
    try
      {
        for(int i = 1; i <= 5; i++)
        {
          System.out.println(message + " - " + i);
          Thread.sleep(2000); //sleep thread for 2 seconds
        }
      }
    catch(InterruptedException ie) {}
  }
}

public class ThreadDemoWithSleep
{
  public static void main(String[] args)
  {
    MyThread t = new MyThread("Hello");
    t.start();
  }
}
```

▼ Running Multiple Threads

```
class MyThread extends Thread
{
  String message;
  MyThread(String message)
  {
    this.message = message;
  }
  public void run()
  {
    for(int i = 1; i <= 5; i++)
      System.out.println(message + " - " + i);
  }
}

public class MultipleThreadDemo
{
  public static void main(String[] args)
  {
    MyThread t1 = new MyThread("Hello"); //first thread
    MyThread t2 = new MyThread("Goodbye"); //second thread
    t1.start();
    t2.start();
  }
}
```

- How multiple threads are executed:

1. Thread priorities.
2. Load on the CPU.
3. How OS implements multitasking and multithreading.
4. Availability of resources.

▼ Synchronization

- Each thread may access shared resources at the same time, this may lead the resource to be in inconsistent state. Such situation is called as **Race Condition**.
- To ensure that the resource is used by only one thread at a time we use a mechanism called **Synchronization**.
- A synchronized block acquires a mutually-exclusive lock (semaphore) on behalf of the executing thread, it executes the thread and then releases the lock. Only one thread can acquire the semaphore at a time and all other threads go to waiting state.
- Syntax:

```
public synchronized function(arguments)
{
        //code
}
```

▼ Interthread Communication

- When thread within a single program talk to one another, it is called **Interthread Communication**.
- *wait()*, *notify()* and *notifyAll()* methods of the Object class must be used within a synchronized block.
- **wait()**: This method causes a thread to exit the lock and sleep until some other thread enters the same lock and calls *notify()*.
- **notify()**: This method wakes up the first thread that called *wait()* on the same object. Thread that wakes up goes to runnable state.
- **notifyAll()**: This method wakes up all threads that called *wait()* on the same object. Highest priority thread will run first.

```
class AddThread extends Thread
{
  ArrayList a;
  AddThread(ArrayList a)
  {
    this.a = a;
  }
  public synchronized void run()
  {
    for(int i = 0; i < 5; i++)
      addItem("Item " + i);
  }
  void addItem(String item
  {
    synchronized(a)
    {
      a.add("Item " + i);
      System.out.println("Item added by Add thread");
      a.notifyAll();
    }
  }
}

class WaitNotfiyDemo
{
  public static void main(String[] args)
  {
    ArrayList a = new ArrayList();
    AddThread t = new AddThread(a);
    t.start();
  }
}
```

Solve questions from 2-23 and onwards.

Database Programming

- Interacting with database requires database connectivity, which can be achieved by using the **Open Database Connectivity (ODBC) driver**. This driver is used with **Java Database Connectivity (JDBC)** to interact with various types of databases, such as *Oracle, MS Access, My SQL, POSTGre SQL* and *SQL Server*.
- **What is JDBC?**
  - JDBC is a specification from Sun Microsystems that provides a standard abstraction (API/ Protocol) for Java application to communicate with different databases.
  - JDBC is the Java API that can handle any kind of tabular data stored in a Relational Databases.
  - JDBC is a java API to connect and execute query with the database. JDBC API uses JDBC drivers to connect with the database.
  - The JDBC API allows java programs to execute SQL statements and retrieve results.
- **JDBC Characteristics**:
  - Supports a wide level of probability.
  - Provides Java interfaces that are compatible with Java applications. These providers are also responsible for providing the driver services.
  - Provides a higher level APIs for application programmers. The JDBC API specification is used as an interface for the application and DBMS.
  - Provides a support for ODBC using JDBC-ODBC Bridge.
  - Provides support for using SQL statements that are responsible for the entire communication with the database.
- **JDBC Architecture/ JDBC Design**:
  - JDBC driver process the SQL requests and generate the results. JDBC API provides classes and interfaces to handle database.

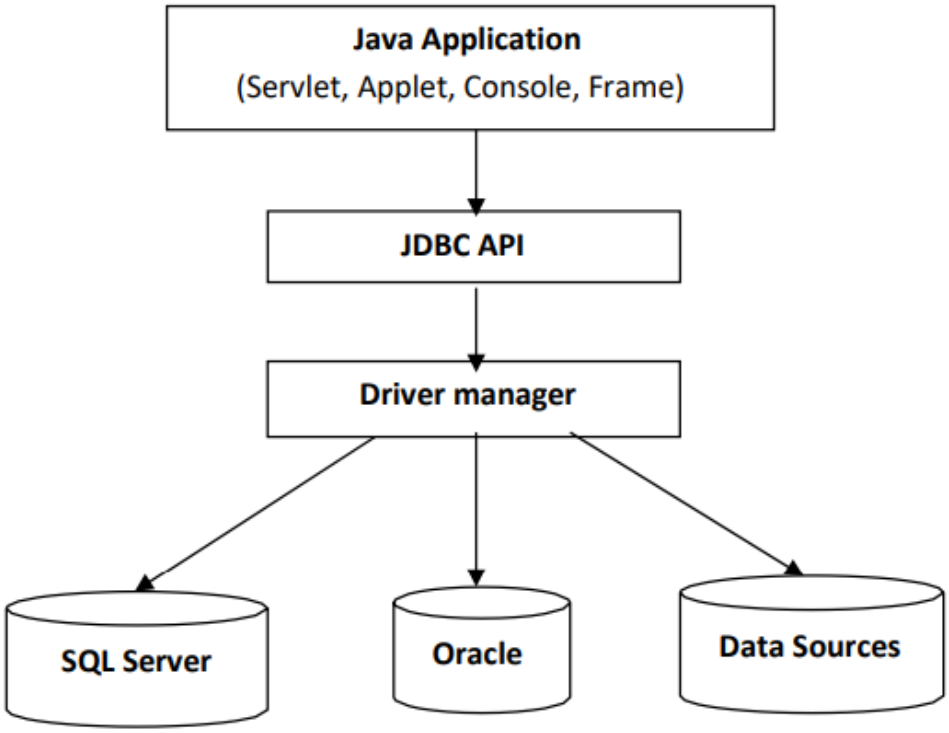


fig. **JDBC Architecture**

- The DriverManager class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol.
- The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database connection.
- As shown in diagram, any type of java application, either client side or server side, never communicates with a database, directly. It communicates using JDBC API.
- JDBC driver is added in the java application for JDBC support.
- After loading required driver, any Java application can communicate with any type of database or data source i.e. Oracle, SQL server or data sources.

- **Types of JDBC Drivers**:
  1. JDBC-ODBC Bridge Driver
  2. Native API/partly Java Driver
  3. All Java/Net-protocol Driver
  4. Native protocol/all-java Driver

JDBC-ODBC Bridge Driver

- It translates all JDBC calls into ODBC calls and sends them to ODBC driver.

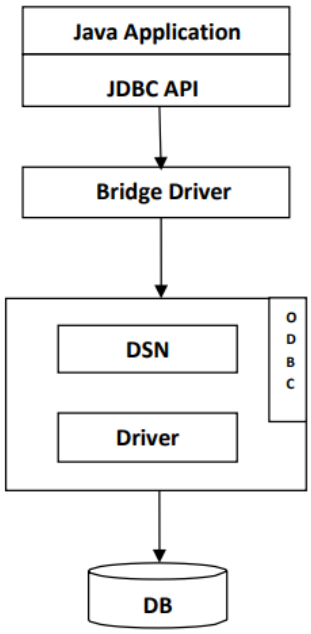
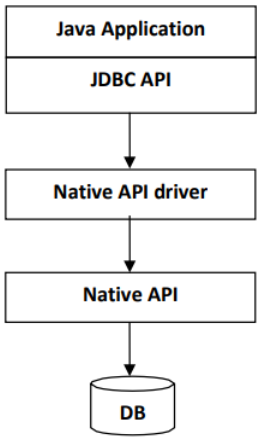


fig. JDBC Type-I driver

- Advantages:
  1. Allows us to communicate with all databases that supports ODBC driver.
  2. It is vendor independent driver.
- Disadvantages:
  1. This driver is not portable, because it is not fully written in java.
  2. Slowest performance.
  3. The internal performance is also a very slow process.
  4. The ODBC installation is necessary on client system to use this driver.
  5. Not recommended for the Web services.

Native API/partly Java Driver

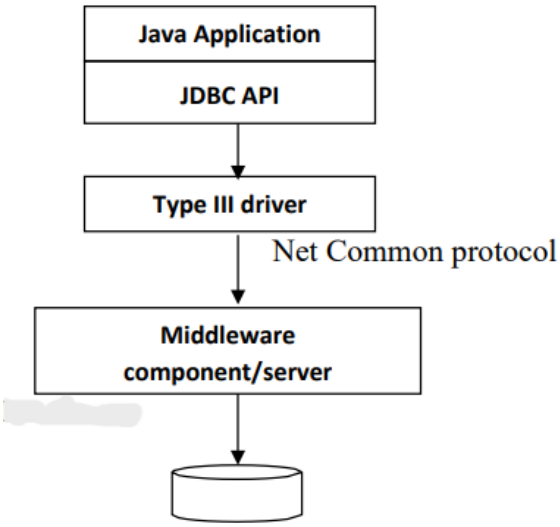
- These drivers convert JDBC calls into database-specific calls i.e. to a particular database.



- Advantages:
  1. Communicate faster as compared to type-I drivers.
  2. Contains additional features provided by the specific database vendor.
- Disadvantages:
  1. Not useful for internet application because it requires pre-installation of native API on client system.
  2. Not portable.
  3. If we change the database, we have to change the native API.
  4. Mostly outdated.
  5. Not thread safe.

Network protocol Driver

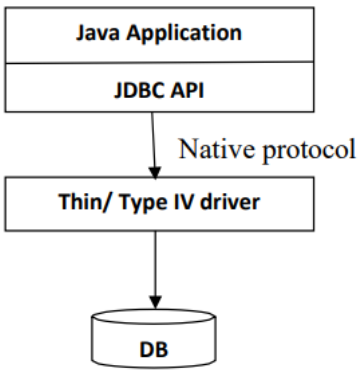
- With the help of middleware server, this driver translates the JDBC calls into database server independent and middleware-server-specific calls.



- Advantages:
  1. No need for any vendor database library to be present on client machine.
  2. Server based driver.
  3. Fully written in Java, hence portable.
  4. Suitable for the Web applications.
  5. Very small and faster to load.
  6. Very flexible and allows access to multiple databases using a single driver.
- Disadvantages:
  1. Requires supporting server application to install and monitor the connection.
  2. The data comes from backend server; hence this makes the traversal of recordset a big time consuming.

Thin Driver

- This driver uses Java networking libraries to directly communicate with the database server.



- Advantages:
    1. Platform independent and eliminate deployment administration issues.
    2. Suitable for the Web.
    3. Number of translation layers is very less.
    4. Do not need to install special software on the client or server.
- Disadvantages:
    1. User needs a different driver for each database.

Major Classes and Interfaces of JDBC:

1. **Class Class:**
    - This class is used to load JDBC driver for connecting databases.
    - This class has no public constructor.
    - Instead its objects are constructed automatically by the JVM & calls to the defineClass() method.
    - Methods of this class:
        - Static Class forName(String className) : Returns the Class object associated with the invoking object specified by class name.
        - Method getDeclaredMethods() : Returns an array of methods, declared by the invoking class object.
        - String getName() : Returns the string, specifying name of the entity(class, interface, array class, primitive type or void) represented by invoking class object.

- URL getResource(String name) : Returns a resource with a given name and returns URL.
- Boolean isArray() : Checks and returns whether invoking Class ibject represents an array class or not.

2. **DriverManager Class:**
   - This class will attempt to load the driver classes referenced in the *jdbc.drivers* system property.
   - This class is defined as: *public class DriverManager extends Object*
   - Methods of this class:
     - static void deregisterDriver(Driver *driver*) : De-registers a driver from the DriverManager's list.
     - static connection getConnection(String URL) : Establishes and returns a connection to the given class.
     - static connection getConnection(String URL,String user_name, String password) : Establishes and returns a connection to the given database URL, with specified user name.
     - static PrintWriter getLogWriter() : Retieves the log writer as PrintWriter instance.
3. **Connection Interface:**
   - A connection is the session between Java application and database.
   - SQL statements are fired and results are returned over the established connection.
   - The objects of Statement, PreparedStatement, DatabaseMetaData, CallableStatement are retrieved using object of connection interface.
   - The interface is defined as: *public interface Connection extends Wrapper, AutoCloseable*
   - Ways of establishing connection:
     i. By using Data Sources
     ii. By manually providing database driver & database name
   - Methods of this interface:
     - void abort(Executor executor) : Terminates an open connection.

- void clearWarnings() : Clears all warnings that are reported for invoking connection object.
- void close() : Closes the open connection.
- void commit() : Assigns all changes into the database, made since the last commit or rollback.
- Blob createBlob() : Creates and returns an object that implements the Blob interface.
- Clob createClob() : Creates and returns an object that implements the Clob interface
- Statement createStatement() : Creates and returns a Statement object for firing SQL queries to the connected database.
- boolean isClosed() : Checks and returns whether invoking Connection object is closed or not.
- boolean isReadOnly() : checks and returns whether invoking Connection object is in read-only mode or not.
- void releaseSavepoint(Savepoint savepoint) : Removes the specified Savepoint object.
- void setReadOnly(Boolean readOnly) : Sets the invoking connection as read-only.
- SavePoint setSavepoint(String name) : Creates and returns a save-point object with the given name.

4. **Statement Interface:**
    - Object of this interface is used for firing a static SQL or PL/SQL queries returning the results it has produced.
    - Defined as: *public interface Statement extends Wrapper, AutoCloseable*
    - Methods of this interface:
        - void cancel() : Cancels this Statement object.
        - void close() : Immediately closes the connection with the statement, instead of waiting for till it is automatically closed.
        - void closeOnCompletion() : Sets the statement to be automatically closed when ResultSet are closed.

- Boolean execute(String sql) : Executes the given SQL statement.
- ResultSet executeQuery(String sql) : Executes the given SQL's select query, which returns a ResultSet object.
- int executeUpdate(String sql) : Executes the given SQL's insert, update or delete query and returns the numbers of records that are affected by this query.
- Connection getConnection() : Returns the connection object that produced invoking Statement object.
- boolean isClosed() : Checks and returns whether invoking Statement object is closed or not.

5. **PreparedStatement Interface:**

- This interface is used when we plan to use the same SQL statement many times. PreparedStatement interface accepts input parameters at runtime.
- Defined as: *public interface PreparedStatement extends Statemenmt*
- Methods of this interface:
  - void clearParameters() : Immediately clears all the parameter values for invoking PreparedStatement instance.
  - boolean execute() : Fires the SQL statement in invoking PreparedStatement object.
  - void setDouble(int index, double val) : Sets the specified double value for the specified index.
  - void setFloat(int index, float val) : Sets the specified float value for the specified index.
  - void setInt(int index, int val) : Sets the specified int value for the specified index.
  - void setLong(int index, long val) : Sets the specified long value for the specified index.
  - void setShort(int index, short val) : Sets the specified short value for the specified index.
  - void setString(int index, String val) : Sets the specified String value for the specified string.

6. **CallableStatement interface:**
    - This interface is used to execute SQL stored procedures.
    - The stored procedures are a group of queries with variables in it.
    - Defined as: *CallableStatement cs=con.prepareCall();*
    - The developer can prepare stored procedure for multiple uses and hence it allows variable declaration also.
    - Methods of this interface:
        - byte getByte(int index) : Retrieves the value present at specified index as byte value.
        - double getDouble(int index) : Retrieves the value present at specified index as double value.
        - double getDouble(String attribute_name) : Retrieves the value present at specified attribute as double value.
        - float getFloat(int index) : Retrieves the value present at specified index as float value.
        - float getFloat(String attribute_name) : Retrieves the value present at specified attribute as float value.
        - int getInt(int index) : Retrieves the value present at specified attribute as int value.
7. **ResultSet interface:**
    - In case of select query, it returns set of selected records, a ReultSet object maintains cursor pointing to its current row of data.
    - Defined as: *public interface ResultSet extends Wrapper, AutoCloseable*
    - Methods of this interface:
        - boolean absolute(int row) : Moves the cursor to the specified row number. Returns false if no such row is found.
        - void afterLast() : Moves the cursor to the end of these ResultSet object, i.e. just after the last row.
        - void beforeFirst() : Moves the cursor to the front of this Resultset object, i.e. just before the first row.

- void close() : Immediately releases invoking ResultSet object's database and JDBC resources.
- void deleteRow() : Deletes the current row from invoking Resultset object and also from the database.
- boolen first() : Moves the cursor to the first row in this ResultSet object.
- float getFloat(int index) : Retrieves the value present at specified index as float value.
- int getInt(int index) : Retrieves the value present at specified index as int value.

MetaData

- MetaData is data of data, i.e. we can get further information from the data.

a. In case of ResultSet, we have metadata of retrieved ResultSet called as **ResultSetMetaData**.

b. In case of Database, we have metadata of database called as **DatabaseMetaData.**

1. **ResultSetMetaData:**
   - This interface provides an object that can be used to get information about the types and properties of the columns in a ResultSet object.
   - Defined as: *public interface ResultSetMetaData extends Wrapper*
   - Methods of this interface:
     - int getColumnCount() : Returns the number of columns available in ResultSet object.
     - String getColumnType(int col_num) : Returns the SQL type of the column.
     - getPrecision(int col_num) : Returns the max size for the column.
     - Boolean isAutoIncrement(int col_num) : Checks & returns whether the specified column is set for auto increment or not.
     - String getTableName(int col_num) : Returns the table name to which the specified column belongs.

    - boolean isNullable() : Checks and returns whether the specified column is allowed to keep null or not.
    - boolean isReadOnly() : Checks and returns whether specified column is read only or not.

2. **DatabaseMetaData:**
    - An instance of this interface provides comprehensive information about the database.
    - Defined as: *public interface DatabaseMetaData extends Wrapper*
    - Methods of this interface:
        - boolean allProceduresAreCallable() : Retrieves whether all the stored procedures are callable or not.
        - Boolean allTablesAreSelectable() : Retrieves whether the user can fire SELECT query on all connected tables.
        - int getDatabaseMinorVersion() : Retrieves the minor version of connected database.
        - int getDatabaseMajorVersion() : Retrieves the major version of connected database.
        - String getDatabaseProductName() : Retrieves the name of this database product.
        - String getDatabaseProductVersion() : Retrieves the product version of the connected database.

Transactions

- Similar to RDBMS, Java's JDBC also provides transactions and its commit, rollback and save-point commands as methods of Connection interface.
- All the JDBC connections are by default in auto-commit mode, i.e. all the SQL statements are committed to the database at the end.
- We can change the settings invoking: *con.setAutoCommit(false)*
- Main advantages of transactions is that we can manage if and when, the SQL queries should affect the database.
- Transaction processes are mainly used in e-commerce related applications.

Commit and Rollback

- The two simple methods provided by Connection interface are commit() and rollback().
- **commit()**
  - This method will put the changes in the database table and changes the database schema.
  - This command permanently saves any transaction into database.

```
public void commit();
```

- **rollback()**
  - This method will undo the last made changes.
  - Restores the database to last committed state.
  - It is also used with savepoint command to jump to a savepoint in a transaction

```
public void rollback();
```

- **savepoint**
  - savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

```
public void rollback(String Savepoint_name);
```

Solve problems from 3-24 and onwards.

Chapter 1

Q1 Explain map interface. Give its implementation.

- The Map interface maps unique keys to values. A key is an object that you use to retrieve a value at a later date.
- Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key.
- Several methods throw a No Such Element Exception when no items exist in the invoking map.
- There are following three main implementations of Map interfaces:

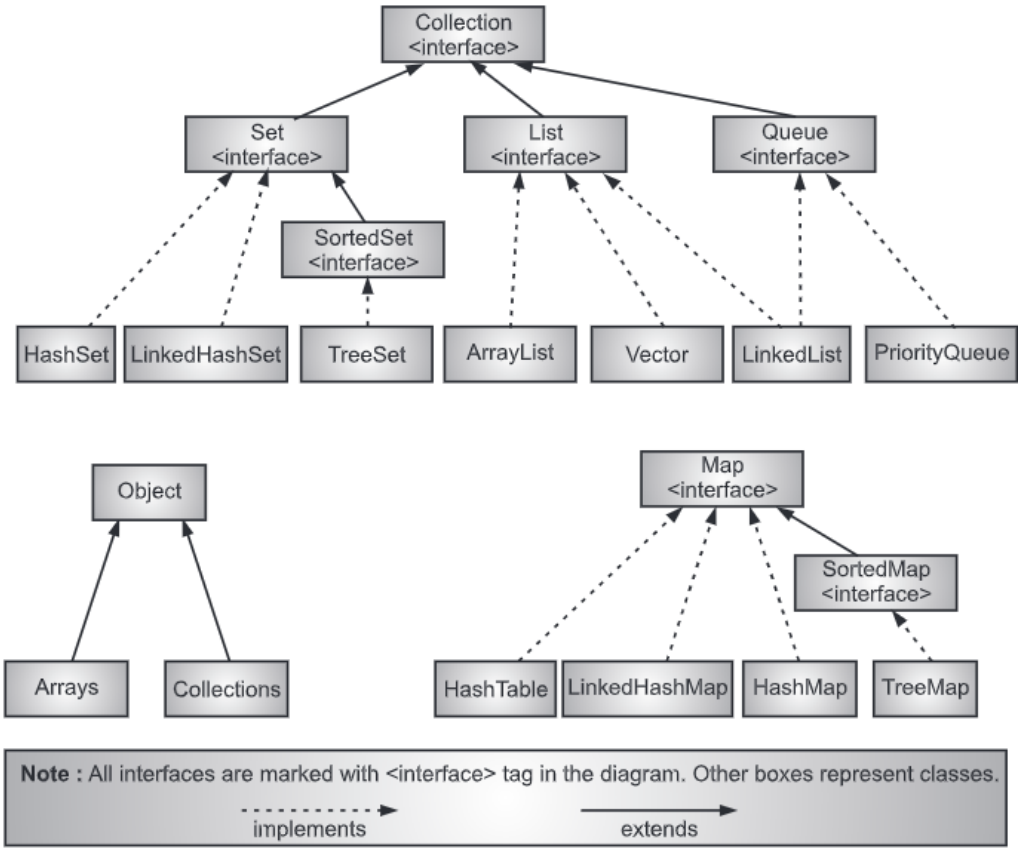
1. **HashMap**: It makes no guarantees concerning the order of iteration
2. **Tree Map**: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.
3. **Linked HashMap**: Its carders its elements based on the order in which they were inserted into the set (insertion-order).

Q2 Differentiate between iterator, list iterator and enumeration interface

Iterator	Enumeration	List iterator
Iterator is implemented on all Java collection classes.	Iterator is implemented on all Java collection classes.	ListIterator is implemented only for List type classes
Iterator uses iterator() method.	Enumeration uses elements() method.	ListIterator uses listIterator() method.
Enumeration can traverse in forward direction only.	Iterator can traverse in forward direction only.	ListIterator can traverse in forward and backward directions.
Iterator having methods like hasNext(), next(), remove(). .	Enumeration having methods like hasMoreElement(),nextElement().	ListIterator having methods like hasNext() next() previous()

Q3 what is collection framework? Explain with diagram. Also state its advantages

- The Collection framework which is contained in the java.util package is one of Java's most powerful systems.
- The Collection framework defines a set of interfaces and their implementations to manipulate Collections, which serve as a container for a group of objects such as a collection of mails.
- A Collection is an object that holds other objects. A Collection is a group of objects. In Java, these objects are called elements of the Collection.
- The Collections framework was designed to meet following goals:
    1. The Collection framework had to extend and/or adapt a collection easily.
    2. The framework had to be high-performance.
    3. The framework had to allow different types of Collections to work in a similar manner and with a high degree of interoperability.



Advantages of collection framework

1. We need not to learn multiple ad hoc collection APIs.
2. It provides a standard interface for collections that fosters software reuse and also provides algorithms to manipulate them.
3. Reduces the effort required to design and implement APIs by eliminating the need to produce ad hoc collections APIs.
4. It provides useful data structures and algorithms that reduces programming effort due to which we need not to write them ourselves.

Q4 Difference between array list and link list

Sr no	Array list	Link list
1	internally uses dynamic array to store the elements.	LinkedList internally uses doubly inked list to store the elements
2	Manipulation with Array List is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is faster than Array List because it uses doubly linked list so no bit shifting is required
3	Array List class can act as a list only because it implements List only.	In memory. LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
4	Array List is better for storing and accessing data.	LinkedList is better for manipulating data.

Q4 – a Explain treeset in detail

- TreeSet class provides an implementation of the Set interface that uses a tree for storage. Objects are stored in sorted, ascending order.
- Access and retrieval times are quite fast, which makes TreeSet an excellent choice
- when storing large amounts of sorted information that must be found quickly.

Constructors:

1. TreeSet() constructor constructs an empty tree set that will be sorted in ascending order according to the natural order of its elements.
2. TreeSet(Collection c) constructor builds a tree set that contains the elements of
3. TreeSet(Comparator comp) constructs an empty tree set that will be sorted according to the comparator specified by comp.
4. TreeSet(SortedSet ss) builds a tree set that contains the elements of ss.

Q4 – b What is Hashtable ? Describe with example

- Hashtable was part of the original java.util and is a concrete implementation of a Dictionary.
- A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method.
- A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.
- It contains only unique elements. It may have not have any null key or value. It is synchronized.

Constructors:

1. Hashtable() default constructor.
2. Hashtable(int size) creates a hash table that has an initial size specified by size. table that
3. Hashtable(int size, float fillRatio) creates a hash table that has an initial size specified by size and a fill ratio specified by fillRatio. This ratio must be between 0.0 and 1.0, and it determines how full the hash table can be before it is resized upward.

Q4 – c What is tree map class

- The TreeMap class implements the Map interface by using a tree. A TreeMap provides an efficient means of storing key/value pairs in sorted order, and allows rapid retrieval.
- TreeMap class implements Map interface similar to HashMap class. The main. difference between them is that HashMap is an unordered collection while TreeMap is sorted in the ascending order of its keys.
- TreeMap is unsynchronized collection class which means it is not suitable for thread safe operations until unless synchronized explicitly.

Constructors:

1. TreeMap() constructs an empty tree map that will be sorted by using the natural order of its keys.
2. TreeMap (Comparator comp) constructs an empty tree-based map that will be sorted by using the Comparator comp.
3. TreeMap (Map m) initializes a tree map with the entries from m, which will be sorted by using the natural order of the keys.
4. TreeMap(SortedMap sm) initializes a tree map with the entries from sm, which will be sorted in the same order as sm

Chapter 2

Q5 write a note on thread priorities.

- Thread priorities are the integers which decide how one thread should be treated with respect to the others.
- Thread priority decides when to switch from one running thread to another, process is called context switching.
- Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.
- Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX PRIORITY (a constant of 10). By default, every thread is given priority NORM PRIORITY (a constant of 5).
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependent.
- To set the priority of the thread setPriority0 method is used which is a method of the class Thread Class.

Q5- a What is thread? State its advantages and disadvantages

- A thread is a lightweight process that can be executed independently within an application.
- A thread is a program's path of execution. A thread is a line of execution. A thread is a lightweight sub process, a smallest unit of processing.
- Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.
- A thread shares the code section, data section and other operating system resources [like open files] with other threads belonging to the same process.

Advantages of multithreading:

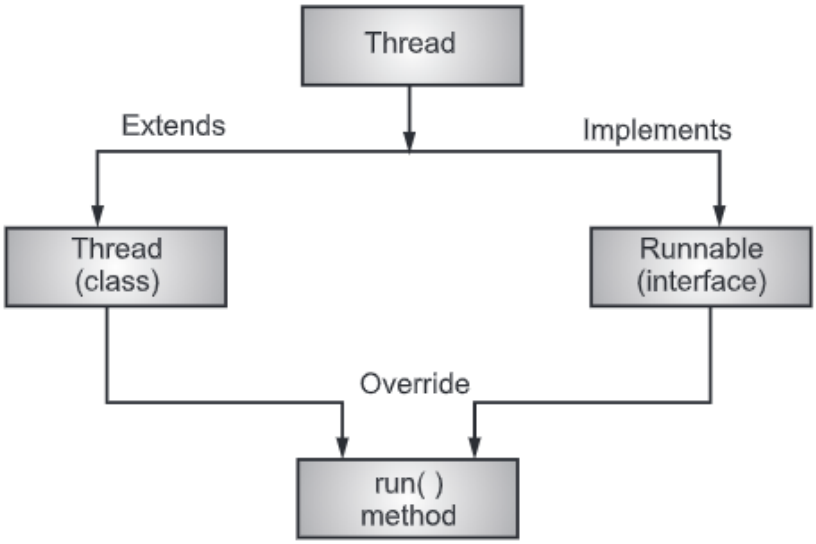
1. CPU idle time is kept to minimum.
2. Efficient use of I/O resources.
3. Threads start and stop independently.
4. Improves the performance of the application.

Disadvantages of multithreading:

1. Difficult to code.
2. Difficult in debugging.
3. Difficult to manage concurrency and requires precise context switching.
4. Multithreading support must be provided by the OS.

Q6 What are two different ways used to implement threading in Java? Explain with example.

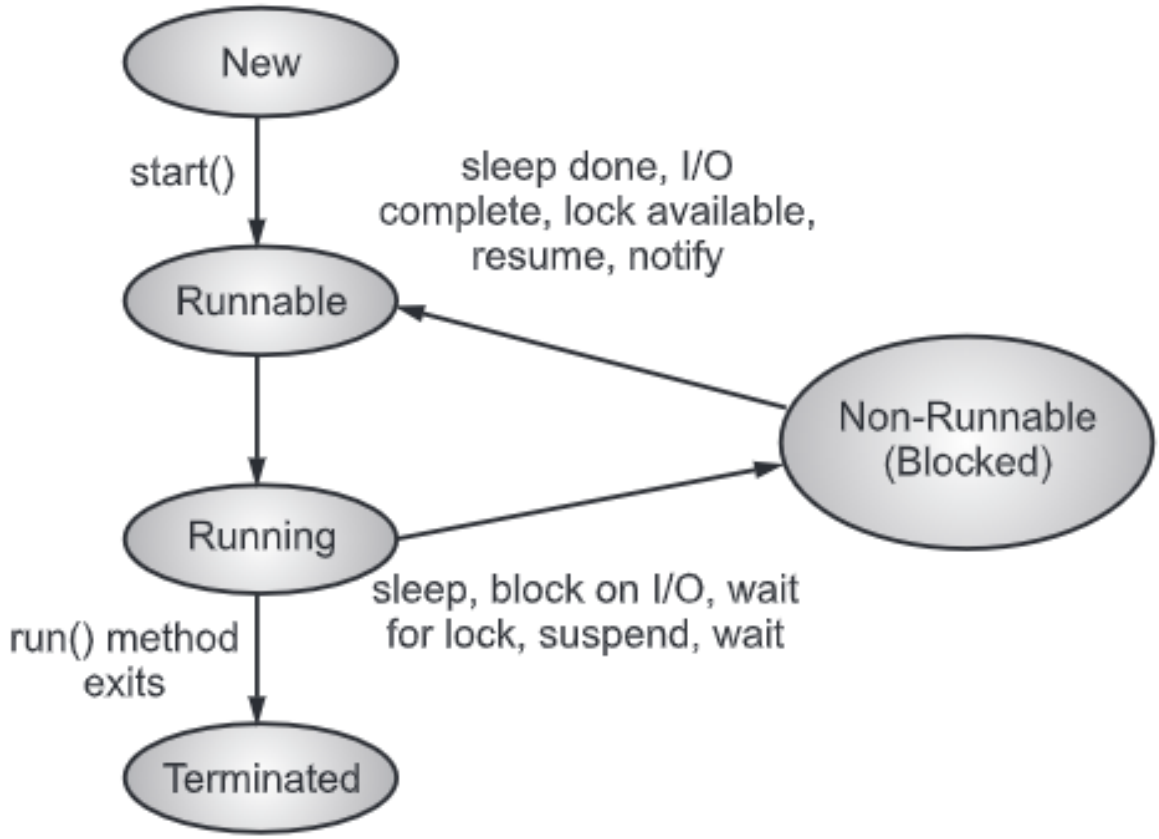
- We can create thread in java with two different ways
  - Extending the thread class, and
  - Implements runnable interface.



Using a thread class	Runnable Interface
• The one way to create a thread is to create a new class that extends Thread class using the following two simple steps. • B This approach provides more flexibility in handling multiple threads created using available methods in Thread class. **Step 1**: We will need to override run0 method available in Thread class. This method provides entry point for the thread and we will put you complete business logic inside this method. Following is simple syntax of run() method: public void run() **Step 2**: Once, Thread object is created, you can start it by calling start() method, which executes a call to run() method. Following is simple syntax of start() method: void start();	This is another easiest way of creating threads and it contains abstract method. If the class is intended to be executed as a thread then we can achieve this by implementing Runnable interface. **We will need to follow three basic steps:** **Step 1**: As a first step you need to implement a run0) method provided by Runnable interface. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run0 method: public void run() **Step 2**: At second step you will instantiate a Thread object using the following constructor: Thread (Runnable thread0bj, String thread Name); Where, thread Obj is an instance of a class that implements the Runnable interface and thread Name is the name given to the new thread.

Q7 Explain thread life cycle with its methods.

- A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. This is also known as life cycle of a thread.
- The life cycle of the thread in java is controlled by JVM.



- **New**: A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread. The thread is in new state if you create an instance of Thread class but before the invocation of star to method.
- **Runnable**: The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- **Running:** The thread is in running state if the thread scheduler has selected it.
- **Non-Runnable (Blocked)**: This is the state when the thread is still alive, but is currently not eligible to run
- **Terminated:** A thread is in terminated or dead state where its run0 method exits

Q8 Explain thread synchronization with an example.

- Multithreading introduces asynchronous behaviour to the programs. If a thread is writing some data another thread may be reading the same data at that time. This may bring inconsistency.
- When two or more threads wants to access a shared resource, then it must ensure that the resource will be used by only one thread at an instant of time. The mechanism of this process is called synchronization.
- Synchronization is the concept of the monitor or semaphore. Monitor works as mute and restrict to one thread to own a monitor at a given time.
- As the thread acquires the lock, all the threads that want to acquire the monitor will be suspended.

Example

```
class mythread extends Thread
{
String msg[]{"Java", "Supports", "Multithreading", "Concept"} mythread
(String name)
{
super (name) ;
}
public void run()
{
display (getName());
System.out.println("Exit from "+getName());
}
synchronized void display(String name) //Synchrinized method
{
for(int i=0;i<msg.length;i++)
{
System.out.println(name+msg[1]);
}
}
class MySynchro
public static void main(String args[])
{
mythread t1=new mythread ("Thread 1: ");
mythread t2=new mythread("Thread 2: ");
ti.start():
t2.start():
System.out.println( "Main thread exited");
}  }
```

Q8-a Explain Interthread Communication

- When thread within a single program talk to one another, it is called Interthread Communication.
- wait(), notify() and notifyAll() methods of the Object class must be used within a synchronized block.

  - wait(): This method causes a thread to exit the lock and sleep until some other thread enters the same lock and calls notify().
  - notify(): This method wakes up the first thread that called wait() on the same object. Thread that wakes up goes to runnable state.
  - notifyAll(): This method wakes up all threads that called wait() on the same object. Highest priority thread will run first

Q8-b Difference between multitasking and multithreading

Parameters	**multitasking**	**multithreading**
Basics	The process of multi-tasking lets a CPU execute various tasks at the very same time.	The process of multi-threading lets a CPU generate multiple threads out of a task and process all of them simultaneously.
Working	A user can easily perform various tasks simultaneously with their CPU using multi-tasking	A CPU gets to divide a single program into various threads so that it can work more efficiently and conveniently. Thus, multi-threading increases computer power.
Switching	There is the constant switching between various programs by the CPU	The CPU constantly switches between the threads and not programs.
Multiprocessing	It involves multiprocessing among the various components	It does not involve multiprocessing among its various components.
Speed of Execution	Executing multi-tasking is comparatively slower.	Executing multi-threading is comparatively much faster.

Chapter 3

Q9 What is metadata? How do you create database and ResultSet metadata? Explain in detail

- JDBC Metadata is the collective information about the data structure and property of a column available in table.
- The metadata of any table tells we the name of the columns, data type used in column and constraint used to enter the value of data into column of the table.

Database metadata

- Applications sometime need to know something about the underlying database. For example, an application might want to know the version of JDBC, even the maximum number of open connections such a information can be retrieved using DatabaseMetaData.
- It provides data about a database's structure, such as the table names, primary and foreign keys, and data types etc. In addition, you can use a DatabaseMetaData object to probe the database to determine its attributes.

Resultset metadata

- Represents the information about a particular result. Here, you can find information such as column-header names, the numbers of columns present in the return results, and whether the values are read-only.
- A ResultSetMetaData object is useful when you want to create a generic method with which to process resultsets.
- By using metadata you can determine the data types of the result set columns and call the correct getXXX() method to retrieve the data.
- The ResultSetMetaData interface provides descriptive information about the columns in a result set such as the number of columns it contains or each column's data type

Q10 What is ResultSet? Explain scrollable and updatable resultsets.

- All query that we fire on database return some result if there is no problem in query. Even if there is no item matching with the values specified in query, whatever result return by query should be store somewhere. That result is gets stored in ResultSet in Java.
- When result is returned from database, database has a choice to construct it:

1. A read-only: As the name suggested here ResultSet enables you to read through all of the values from the first to last. Once you reach the end, you cannot reread the values.
2. **A scrollable**: This type you can reread SesultSet again and again. With it you can jump to any known row or just move back and forth through the current list.
3. **An updateable**: This form is a live representation of the underlying database that enables you to insert, update, and delete rows. This form must also be scrollable.

Q11 What is use of the following in JDBC?

(i) Executed()
(ii) executeUpdate()
(iii) executeQuery()
(iv) statement()
(v) preparedStatement()

Use of Executed()

String query="drop table"; st.execute(query);

Use of executeUpdate()

String query="Update employee SET ename='AMAR' WHERE eid='10' ";
rs=st.executeUpdate(query);

Use of executeQuery ()

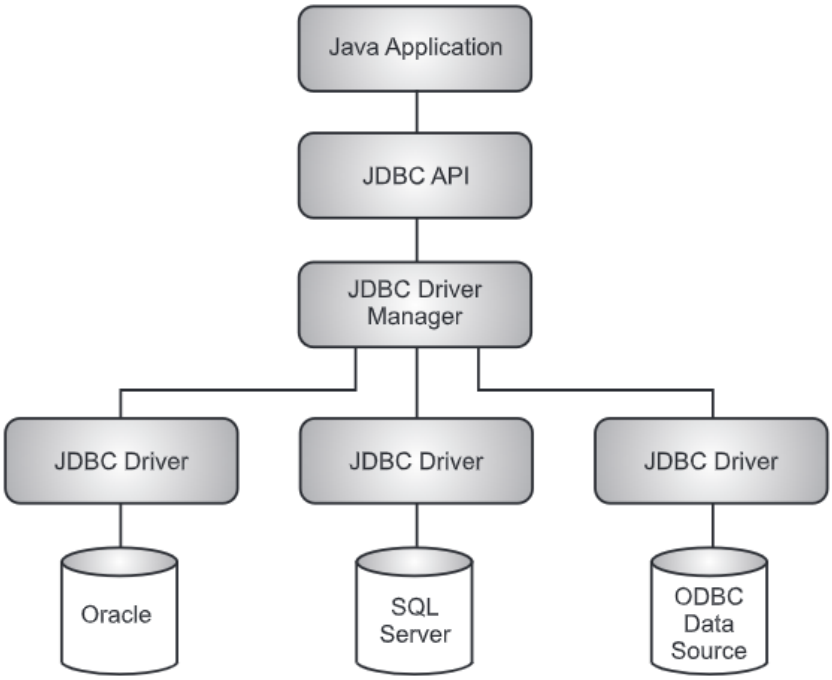
- Private Connection con;
- Statement st;
- ResultSet rs;
- Try {
- Class.forName("org.postgresql.Driver");
- con= DriverManager.getConnection("jdbc:postgresql:EMP", "postgres","");
- String query "Select from employee";
- st-con.createStatement();
- rs-st.executeQuery(query);
- con.close()

Use of preparedStatement()

- SQL queries are precompiled and executed using PreparedStatement.
- PreparedStatement is used when we want to execute SQL queries repeatedly but with different value.
- select from employee where eid=?
- In above example we want to retrieve employee information by specifying the employee id. Here, preparedStatement() method of connection object is used to precompile query which we pass an an argument to this method.
- In the following example we have given "?" at the place of eid, which is later on get replaced by actual eid using setxxx() method which requires two parameters, first is an integer that specifies the position of the question mark placeholder and second the value that replaces the question mark.

Q12 Explain JDBC architecture with types of JDBC drivers.

- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers:
    1. JDBC API: This provides the application-to-JDBC Manager connection.
    2. JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.



Drivers

1.**DriverManager** class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.

2. **Driver Interface** handles the communications with the database server. We will interact directly with Driver objects very rarely, instead, we use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

3. **Connection Interface** with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

Q12-a What is the jdbc drivers? List its types . explain with diagram

- JDBC Driver is a software component that enables Java application to interact with the database. JDBC Driver translates the standard JDBC API calls to the DBMS specific API calls.
- JDBC drivers implement the defined interfaces in the JDBC API, for interacting with the database server.

Type 1: JDBC-ODBC Bridge Driver:

- It is also referred as JDBC - ODBC bridge plus ODBC driver. It is a JavaSoft product.
- In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine.
- Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

Type 2: JDBC-Native API Driver:

- It is also referred as Native-API partly-Java driver. This driver converts JDBC calls into calls on the client API for Oracle, Sybase, Db2, informix or other DBMS.
- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database.
- These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

Type 3: JDBC-Net Pure Java Driver:

- In a Type 3 driver, a Three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server.
- The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

Type 4: Native Protocol Pure Java Drivers (100% Pure Java):

- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection.
- This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible; you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Chapter 4

Q13 Write a note on JSP Expressions with example

- Expression is used to insert values directly to the output. An expression tag places an expression to be evaluated inside the java servlet class.
- A JSP expression element contains a scripting language expression that is evaluated, converted to a String and inserted where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- The expression element can contain any expression that is valid according to the Java language specification but you cannot use a semicolon to end an expression.

Example

```
<html>
<head><title>A Comment Test</title> </head>
<body>
<p>
Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

Q14 What are advantages of JSP?

1. **vs. Active Server Pages (ASP**): The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
2. **vs. Pure Servlets**: It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
3. **vs. Server-Side Includes (SSD):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
4. **vs. JavaScript**: JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
5. **vs. Static HTML**: Regular HTML, of course, cannot contain dynamic information.

Q15 Explain the purpose of GET and POST method is Servlet request. Differentiate between doGetQ and doPostQ methods.

Get method

- The GET method sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the ? character as follows: http://www.test.com/hello?key1-value1&key2-value2
- The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location box.
- Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be in a request string.

Post method

- A generally more reliable method of passing information to a backend program is the POST method.
- This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a? in the URL it sends it as a separate message.
- This message comes to the backend program in the form of the standard input which we can parse and use for your processing. Servlet handles this type of requests using doPost() method

Difference between doGetQ and doPostQ methods.

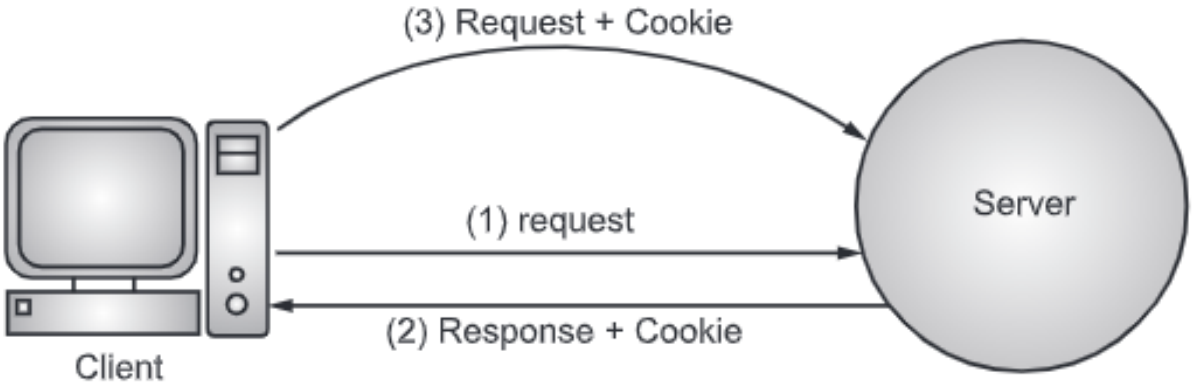
The doGet() Method
: A GET request results from a normal request for a URL or from an HTML form that has no method specified and it should be handled by doGetO method.
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
{
// Servlet code ...
}
	The doPost0 Method:
	A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.
	public void doPost(HttpServletRequest request,
	HttpServletResponse response)
	throws ServletException, IOException
	f
	// Servlet code....
	}

Q16 What is cookie? Explain how cookie can be created and accessed in servlet.

- The third technique that we can use to manage user sessions is by using cookies. A cookie is a small piece of information that is passed back and forth in the HTTP request and response.
- Even though a cookie can be created on the client side using some scripting language such as JavaScript, it is usually created by a server resource, such as a servlet.
- The cookies sent by a Servlet to the client will be passed back to the server when the client requests another page from the same application.

How Cookie Works?

- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the Servlet. So cookie is stored in the cache of the browser



Types of cookies in Servlets:

1. Non-persistent Cookie: It is valid for single session only. It is removed each time when user closes the browser.

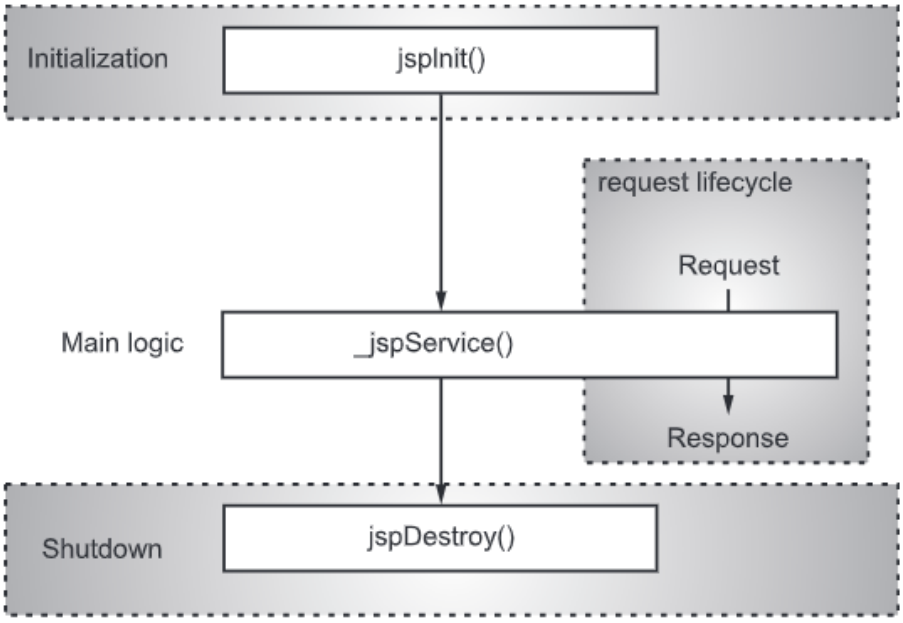
2. Persistent Cookie: It is valid for multiple session. It is not removed each time when user closes the browser. It is removed only if user logout or sign out.

Q16 -a Difference between get and post method

Sr No	GET request	Post request
1	In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
2	Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar..
3	Get request can be bookmarked.	Post request cannot be bookmarked
4	Get request is idempotent. It means second request will be ignored until response of first request is delivered.	Post request is non-idempotent.

Q17 Explain jsp lifecycle with the help of a diagram.

- A JSP life cycle can be defined as "the entire process from its creation till the destruction which is similar to a Servlet life cycle with an additional step which is required to compile a JSP into Servlet".
- A JSP page services requests as a servlet. Thus, the life cycle and many of the capabilities of JSP pages (in particular the dynamic aspects) are determined by Java Servlet technology.



1. JSP Compilation:

- When a browser asks for a JSP, the JSP engine first checks to see whether it needs to
- compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

2. JSP Initialization:

- When a container loads a JSP it invokes the jsplnit() method before servicing any requests.
- Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files and create lookup tables in the jsplnit() method.

3. JSP Execution:

- This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
- Whenever, a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the jspservid method in the JSP.

4. **JSP Cleanup:** The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

- The jspDestroy0 method is the JSP equivalent of the destroy method for servlets.

Q18 Explain scripting elements in JSP.

- When we are developing java server page, the JSP allows us to use the scriptlets (JSP script tags) and also allows us to invoke java components.
- The JSP page is built up using the following components:
  - **Directives,**
  - **Declarations,**
  - **Scriptlets,**
  - **Expressions,**
  - **Standard Actions, and**
  - **Custom Tags.**

Declarations

- A declaration declares one or more variables or methods that we can use in Java code later in the JSP file.
- We must declare the variable or method before you use it in the JSP file.
- Following is the syntax of JSP Declarations:
  < declaration; [declaration;]+... %>

Expressions

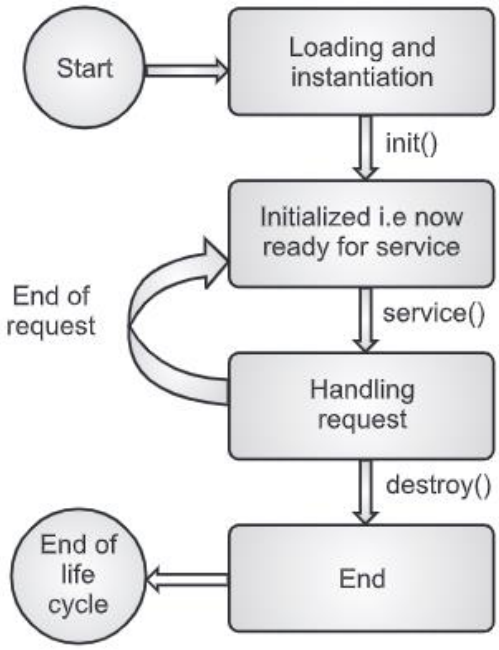
- Expression is used to insert values directly to the output. An expression tag places an expression to be evaluated inside the java servlet class.
- A JSP expression element contains a scripting language expression that is evaluated, converted to a String and inserted where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- The expression element can contain any expression that is valid according to the Java language specification but you cannot use a semicolon to end an expression.

Scriptlets

- A scriptlet can contain any number of java language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- JSP Scriptlets begins with <% and ends %>. We can embed any amount of java code in the JSP Scriptlets.
- **Syntax for Scriptlet:**
  <* //any java source code here ... %>

- A scriptlet is a fragment of Java code that is run when the user requests the page.

Q19 Explain Servlet lifecycle with the help of a diagram.



- The mechanism by which a Servlet works is controlled through a well-defined procedure called the Servlet life cycle.
- The life cycle includes the loading of a Servlet, instantiation and initialization of the Servlet, handling of requests and taking the Servlet out of service after generating the output.

The lifecycle methods in Serviet are explained below:

1. Int method is invoked: The init0 method is designed to be called only once. It is called when the Servlet is first created, and not called again for each user request.

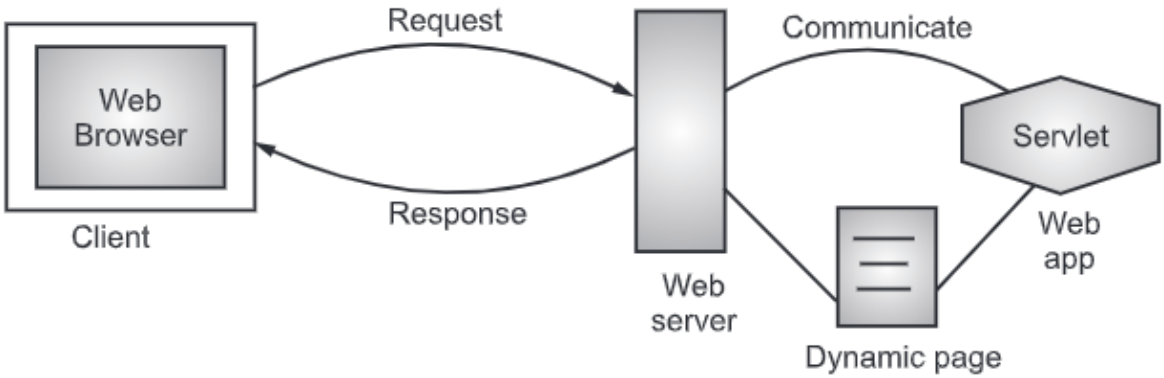
- So, it is used for one-time initializations, just as with the int method of applets. The Servlet is normally created when a user firs: invokes a URL corresponding to the Serviet, but you can also specify that the Servlet be loaded when the server is first started
- When a user invokes a Servlet, a single instance of each Servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The into method simply creates or loads some data that will be used throughout the life of the Servlet.

2. Service() method is invoked: The service0 method is the main method to perform the actual task. The Servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a Servlet, the server spawns a new thread and calls service. The service0 method checks the HTTP request type (GET, POST, PUT, DELETE)

Q20 what is servlet? State its advantages.

- With the growth of commercial activities on the Web, organizations wanted to deliver dynamic content to their customers such as Online Banking, Online Airline, Online Railway Ticket Booking, Online Shopping, News, Marketing and so on.
- A Servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP (HyperText Transfer Protocol).
- Java Servlets are programs that run on a web or application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.



ADVANTAGES

1. **Performance:** The performance of Servlets is superior to CGI because there is no process creation for each client request. Instead, each request is handled by the Servlet container process. After a Servlet is finished processing a request, it stays resident in memory, waiting for another request.
2. **Portability**: Similar to other Java technologies, Servlet applications are portable. You can move them to other operating systems without serious hassles.
3. **Rapid development cycle**: As a Java technology, Servlets have access to the rich Java library, which helps speed up the development process.
4. **Robustness**: Servlets are managed by the Java Virtual Machine (JVM). As such, we do not need to worry about memory leak or garbage collection, which helps we write robust applications.

Q21 List implicit objects in jsp and explain any two implicit objects in jsp.

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

Sr No	Object	Description
1	**request**	This is the HttpServletResponse object associated with the request.
2	**response**	This is the HttpServletResponse object associated with the response to the client.
3	**out**	This is the PrintWriter object used to send output to the client.
4	**session**	This is the HttpSession object associated with the request.
5	**application**	This is the ServletContext object associated with application context.
6	**config**	This is the ServletConfig object associated with the page.
7	**page**	This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
8	**exception**	The Exception object allows the exception data to be accessed by designated JSP.

Q23 differentiate between servlet and jsp

servlet	jsp
Servlet is a java code.	JSP is a HTML based code.
Writing code for servlet is harder than JSP as it is HTML in java.	JSP is easy to code as it is java in HTML.
Servlet plays a controller role in the has MVC approach.	JSP is the view in the MVC approach for showing output
Servlet is faster than JSP.	JSP is slower than Servlet because the first step in the hasJSP lifecycle is the translation of JSP to java code and then compile.
Servlet can accept all protocol requests.	JSP only accepts HTTP requests.
It does not have inbuilt implicit objects.	In JSP there are inbuilt implicit objects. .

Chapter 5

Q24 what is spring framework? State its advantages and disadvantages

- The Spring Framework is one of the most popular Java-based application Frameworks. It is an application framework and Inversion of Control (IoC) container for the Java platform.
- The Spring is the most popular application development framework for enterprise Java.
- Millions of developers around the world use Spring Framework to create high performing, easily testable and reusable code.
- The Spring Framework is a mature, powerful and highly flexible framework focused on building Web applications in Java.
- The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

Advantages

1. Spring Framework is lightweight in nature due to its POJO (Plain Old Java Object) implementation which does not need an enterprise container like an application server.
2. Spring support the use of both XML configuration as well as Java-based annotations for configuring the Spring Beans. Therefore, it provides the flexibility of using any of them for developing the application.
3. The Spring application is loosely couple due to Dependency Injection. Dependency Injection (or sometime called wiring) helps in gluing the classes in Java together and at the same time keeping them independent.
4. Spring provides a consistent transaction management interface that can scale down to a local transaction (for example, using a single database) and scale up to global transactions (for example, using JTA).

Disadvantages

1. Developing a Spring application requires lots of XML coding.
2. The learning curve for Spring Framework is very high as most developers find it hard to understand and apply.
3. For many its time-consuming process as Spring Framework has lots of integration with another framework due to which it is hard to know all the options which are available.
4. The nature of Spring Framework keeps changing over the time which makes it harder to grasp, (for example, the annotation-based Spring).

Q25 What are the application of spring

1. **POJO Based**: Spring enables developers to develop enterprise-class applications using POJOS. The benefit of using only POJOS is that you do not need an EJB container product such as an application server but you have the option of using only a robust Servlet container such as Tomcat or some commercial product.
2. **Modular**: Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.
3. **Integration with existing Frameworks:** Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.
4. **Testability**: Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using Java Bean style POJOS, it comes easier to use dependency injection for injecting test data.
5. **Web MVC**: Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.

Q26 how to validate form in spring. Explain with example

- To display the input form, we are going to use <form:form/> tag of Spring Framework.
- The form tag renders as HTML form. By default, a form executes a GET method to a given action. This means that the data input by the user will be sent to a URL stated within the form.
- The form tag might contain one or more tags inside such as input fields, radio buttons, or checkboxes - to retrieve data from its users. We cover them one by one in this section.
- To help with the binding, the form tag exposes a binding path to its inner tags with the modelAttribute attribute of the form, which states under which name this model class will be.
- The Spring<form:form> declares two attributes. The method-"post" attribute used to indicate a form performs an HTTP POST request upon submission. And the modelAttribute="reservation" attribute used.

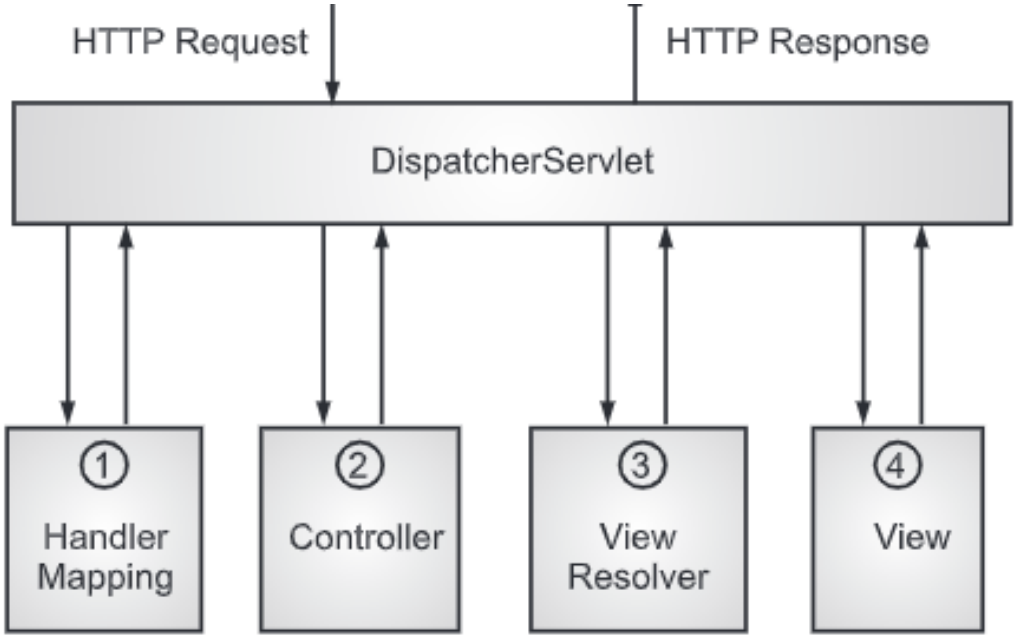
<form:input/>	<form:input path="name" placeholder="Task Name"/>
<form:textarea/>	<form:textarea path="comments" id="txtComments" rows="5" cols="30"/>

Q27 Describe MVC spring architecture. With the help of diagram

- A Spring MVC is a Java Framework which is used to build Web applications. The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications**.**

The Spring MVC consists of:

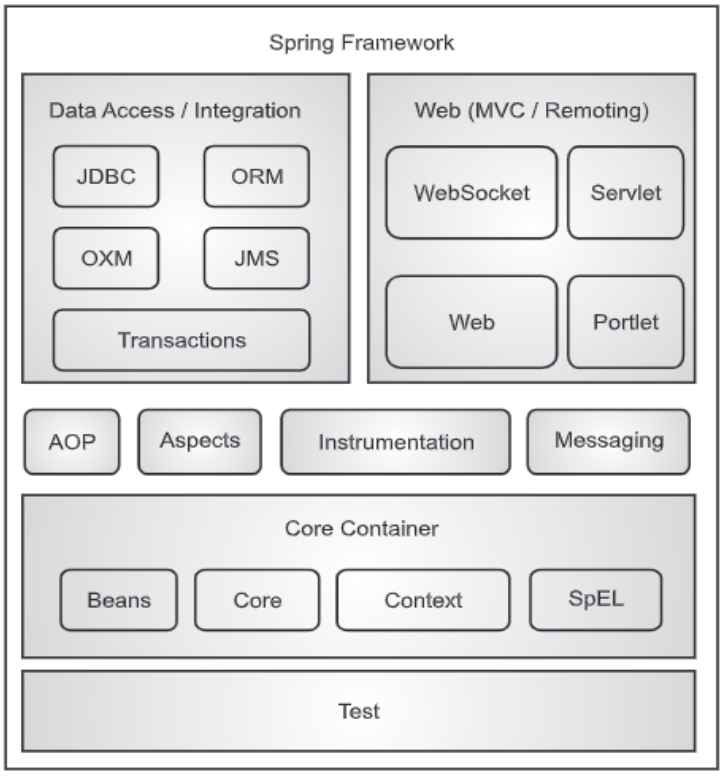
- **Model:** A model contains the data of the application. A data can be a single object or a collection of objects. The Model encapsulates the application data and in general they will consist of POJO.
- **View:** View is responsible for presenting data to the end user. A view represents the provided information in a particular format. The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret**.**
- **Controller:** The controller is a logic that is responsible for processing and acting on user requests. The Controller is responsible for processing user requests and building an appropriate model and passes it to the view for rendering



Following is the sequence of events corresponding to an incoming HTTP request to Dispatcher Servlet:

1. After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller.
2. The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.
3. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.
4. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

Q28 with the help of diagram describe module architecture in spring



- **JDBC Module**: This module provides the JDBC abstraction layer and helps to avoid tedious JDBC coding.
- **ORM Module**: This module provides integration for object relational mapping APIs such as JPA, Hibernate, JDO, etc.
- **JMS (Java Messaging Service) Module**: This module contains features for producing and consuming messages.
- **OXM Module**: This module provides Object/XML binding.
- **Transaction Module:** This model supports programmatic and declarative transaction management for classes that implement special interfaces and for all the POJOS.

1. The AOP module provides an aspect-oriented programming implementation allowing us to define method-interceptors and point-cuts to cleanly decouple code that implements functionality that should be separated.
2. The Aspects module provides integration with Aspect], which is again a powerful and mature AOP framework.
3. The Instrumentation module provides class instrumentation support and class loader implementations to be used in certain application servers.
4. The Messaging module provides support for STOMP as the WebSocket sub protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
5. The Test module supports the testing of Spring components with JUnit or TestNG frameworks.

One marks

Q1 State contructors of TreeSet class.

- Treeset(), treeset(collection c), Treeset (comparator comp), Treeset(sorted set)

Q2 List any four collection interfaces.

- Collection, Set, List , Queue

Q3 What is the purpose of Rollback ( ) in transactions in Java Database Connectivity?

- A rollback operation undoes all the changes done by the current transaction i.e. If you call a rollBack () method of the Connection interface, all the modifications are reverted until the last commit.Con.rollback ()

Q4 What is collection?

- It is an implied simply an object that holds collection of group or container objects

Q5 Which interfaces are implemented by TreeSet class?

- Hierarchy of TreeSet class. As shown in the above diagram, Java TreeSet class implements the NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

Q6 State four constructors of HashSet class.

- Hashset (), Hastset (collection c), Hashset (int capacity), Hashset(float fillratio)

Q7 What is ResultSet Interface? List any two methods.

- The ResultSet interface in the java.sql package in Java represents the result of a database query as a tabular data set
- Next (), previous ()

Q8 Which packages are required for Java Database connectivity?

- java.sql.*; 2. javax.sql.*;

Q9 State the use of Iterator and ListIterator.

- An Iterator can be used in these collection types like List, Set, and Queue whereas ListIterator can be used in List collection only.

Q10 When to use execute update (string sql) method?

- Use this method to execute SQL statements, for which you expect to get a number of rows affected – for example, an INSERT, UPDATE, or DELETE statement.

Q11 Name the statement types used for executing SQL queries.

- Creating a statement, Executing the Statement object, Prepared Statement, Callable Statement

Q12 What is Inversion of Control?

- Inversion of control is a pattern used for decoupling components and layers in the system. The pattern is implemented through injecting dependencies into a component when it is constructed.

Q13 List the modules in the core container of Spring.

- The core module is the core of the component model; it provides fundamental features like dependency injection

Q14 What is the main purpose of the Spring Framework.

- **Spring framework** helps develop various types of applications using the Java platforms**. It provides an extensive level of infrastructure support.**

Q15 What is Aspect Oriented Programming?

- AOP (Aspect Oriented Programming) AOP is a **programming paradigm** whose main aim is to increase modularity by allowing the separation of cross-cutting concerns.

Q16 what is collection framework?

- Java collection framework represents a hierarchy of set of interfaces and classes that are used to manipulate group of objects**.**

Q17 state two ways in creating thread.

- Extending the thread class
- Implements runnable interface

Q18 What is the purpose of join method in the context of thread?

- The current thread invokes this method on second thread causing the current thread to block until the second thread terminates.

Q19 What is multithreading?

- Multithreading in java is the process of executing multiple threads simultaneously

Q20 state any two methods of inter thread communication.

- Wait() , Notify()

Q21 What is the use of setautocommit().

- it saves all the changes that have been done till that particular point.

Chapter 1

Q1 Write a Java program to accept 'n' employee information (id, name) and store into Hashtable. Display all employee details.

```
class Employee
{
public string name;
public int id;
public Employee()
{
}
public Employee(string name, int id)
{
this.name = name;
this.id = id;
}
public void GetEmployeeData()
{
{
Console.WriteLine("Enter the name of employee: ");
name = Convert.ToString(Console.ReadLine());
Console.WriteLine("Enter the id of employee: ");
salary = Convert.ToInt32(Console.ReadLine());
}
}
public void DisplayEmployee()
{
Console.WriteLine("The name of employee is: "+ name);
Console.WriteLine("The salary of employee is: "+id);
}
}
class Program
{
static void Main(string[] args)
{
Employee[] e = new Employee[10];
for(int i = 0; i < 10; i++)
{
Console.WriteLine($"Enter the data of employee with id: {i}
e[i] = new Employee();
e[i].GetEmployeeData();
```

```
}
Console.WriteLine("**********
The data of given employees is: ");
for(int i = 0; i < 10; i++)
{
e[i].DisplayEmployee();
}
}
}
```

Q2 Write a program that uses Hashtable for storing and retrieving students records, (containing Name and Percentage) Display the details of students having highest percentage.

```
import java.io.*;
import java.util.*;
class DemoStudent
Ff 1
public static void main(String args[]) throws Exception
{
BufferedReader br = new BufferedReader (new
InputStreamReader(System.in));
Hashtable ht=new Hashtable();
float sper;
String sname=null;
System.out.println("\n Enter no of Students : ");
int n=Integer. parseInt(br.readLine());
for(int i=@;i¢n;i++)
{
System.out.print("Enter Student name :");
sname=br.readLine(};
system.out.print("Enter Student's per :");
sper = Float.parseFloat(br.readLine());
ht.put(sname, sper);
}
System.out.println("Student Name");
Enumeration keys = ht.keys();
while( keys. hasMoreElements() }
System.out.println( keys.nextElement() );
System.out.println("Student Percentage");
Enumeration values = ht.elements();
while( values. hasMoreElements({))
System.out.println( values.nextElement());
System.out.println("Enter the student name to be searched:");
BufferedReader br1 =new BufferedReader(new
InputStreamReader(System.in));
if(ht.containsKey(m))
{
System.out.printlin("Record found and percentage is:");
System.out.println(ht.get(m));
}
else
    {
    system.out.printin("no such student found");
    }
    } }
```

Q3Write a program to create a linklist of four integer objects and do the following operations. (i) Add element at first position (ii) Delete last element.

```
import java.util.LinkedList;
import java.util.Scanner;
public class C2 {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
LinkedList<Object> ll = new LinkedList<>();
ll.add(1);
ll.add(2);
ll.add(3);
System.out.println("\nElements in List :\n" + ll);
ll.addFirst(0);
System.out.println("\nList after adding Elements at First :\n" + ll);
ll.removeLast();
System.out.println("\nList after deleting Last Element :\n" + ll);
System.out.println("\nSize of the List :\n" + ll.size());
sc.close();
}
}
```

Q4 Write a Java program to accept names of n cities, insert the same into array list collection and display the content of the same array list, also remove all these elements.

```
import java.util.*;
public class A1 {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
ArrayList<Object> al = new ArrayList<>();
System.out.println("Enter How many cities :");
int n = sc.nextInt();
System.out.println("Enter the Cities :");
sc.nextLine();
for (int i = 0; i < n; i++) {
        String c = sc.nextLine();
        al.add(c);
}
System.out.println("Cities :" + al);
System.out.println("ArrayList after removing the elements  :");
al.clear();
sc.close();
} }
```

Chapter 2

Q1 Write a program to display "examination" 50 times using multithreading.

import

```
java.lang.*:
class HelloThread extends Thread { public HelloThread(String name) {
super(name):
}
public void run() {
try {
for(int i=0;i<25:1-)( System.out.println(Thread.currentThread().getName
()-"Hello");
sleep(500):
}
}catch(InterruptedException e) {
System.out.println(ie);
}
}
}
public class ThreadMain (
public static void main(String args[])
HelloThread = new HelloThread("My Thread");
t.start();
try{
for (int i=0;i<25;i++){
System.out.println(Thread.currentThread().getName ()+"Examination");
Thread.sleep(500);
}
} catch (InterruptedException e) {
System.out.println(e);
}
}
}
```

Chapter 3

Q1 Write a JDBC program to insert following 'n' records to employee table having structure emp_no, ernp_name and salary and display it.

```
importjava.sql.*;
public class Slip3_1
{
public static void main(String args[])
{
Connection con;
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:odbc:dsn");
if(con==null)
{
System.out.println("Connection Failed....");
System.exit(1);
}
Statement stmt=con.createStatement();
ResultSetrs=stmt.executeQuery("select * From employee Where dept='computer
science'");
System.out.println("eno\t"+"ename\t"+"department\t"+"sal");
while(rs.next())
{
System.out.println("\n"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+""+rs.getI
nt(4));
}
con.close();
rs.close();
stmt.close();
}
catch(Exception e)
{
System.out.println
}
}
}
```

Q2 Write a JDBC program to read, update any record from database. Database about element Oxygen and Hydrogen has the following fields: (Atomic weight, name, chemical symbol). The input should provided through command line. For example, Java pgm_name R will Read and show the contents of the table Java pgm_name U10 'oxygen' 137 will update the record according to the name specified.

```
import java.sql.*;
import java.io.*;
class Slip23_2
{
static BufferedReader br =new BufferedReader(new
InputStreamReader(System.in));
Connection con;
PreparedStatement ps;
Statement st;
ResultSet rs
void modify()throws Exception
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:dbc:dsn");
System.out.print("Enter Roll no to be updated:");
int rno=Integer.parseInt(br.readLine());
System.out.print("Enter new Percentage :");
float per=Float.parseFloat(br.readLine());
String sql="update student set per=? where rno=?";
ps=con.prepareStatement(sql);
ps.setFloat(1,per);
ps.setInt(2,rno);
int n=ps.executeUpdate();
if(n>0)
System.out.println("Record Updated...");
}
void delete()throws Exception
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:dbc:dsn");
System.out.println("Enter Roll No to be deleted :");
int rno=Integer.parseInt(br.readLine());
String sql="delete from student where rno=?";
ps=con.prepareStatement(sql);
ps.setInt(1,rno);
int n=ps.executeUpdate();if(n>0)
System.out.println("RecordDeleted...");
}
void viewAll()throws Exception
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection
("jdbc:dbc:dsn");
String sql="select * from student";
st=con.createStatement();
rs=st.executeQuery(sql);
System.out.println("Roll No\t Name \t
 Percentage");
while(rs.next
{
System.out.println(rs.getInt(1)+
"\t"+rs.getString(2)+"\t"+rs.getFloat(3));
}
}
public static void main(String a[]) throws
Exception
{
Slip23_2 ob = new Slip23_2();
NR CLASSES, PUNE (8796064387/90)
char ch;
do
{
String str = (a[0]);
ch=str.charAt(0);
switch(ch)
{
case 'U' : ob.modify();
break;
case 'D' : ob.delete();
break;
case 'R' : ob.viewAll();
break; case 'E' : break;
                }
                }
            while(ch!='E');
                } }
```

Q3 Write a JDBC program that accept department from user and display the employee information, who have maximum salary from that department.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
class JDBCmaximumSalary {
	public static void main(String[] args) {
		System.out.println("Finding Highest salary Example...");
		Connection conn = null;
		String url = "jdbc:mysql://localhost:3306/";
		String dbName = "employees";
		String driverName = "com.mysql.jdbc.Driver";
		String userName = "root";
		String password = "root";
		Statement statement = null;
		ResultSet rs;
		try {
			Class.forName(driverName);
			conn = DriverManager
					.getConnection(url + dbName, userName, password);
			statement = conn.createStatement();
			String sql = "SELECT * FROM employee WHERE salary=(SELECT
			MAX(salary) FROM employee)";
			rs = statement.executeQuery(sql);
			System.out.println("EmpId\tName\tSalary");
			System.out.println("----------------------");
			while (rs.next()) {
				int roll = rs.getInt("emp_id");
				String name = rs.getString("name");
				long salary = rs.getLong("salary");
				System.out.println(roll + "\t" + name + "\t" + salary);
			}
			conn.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Chapter 4

Q1 Write a Servlet program to count the number of times a Servlet has been invoked.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet
{
int c=0;
        public void doGet(HttpServletRequest r1,HttpServletResponse r2)throws
        ServletException,IOException
{
r2.setContentType("text/html");
        PrintWriter pw=r2.getWriter();
        c++;
        pw.println("Servlet Is Invoked For ");
        pw.println(""+c + "Times");
        pw.close();
}
}
```

Q2 Write a Servlet program that accept teacher id, name and subject and also print the same.

```
<!doctype html>
   <body>
      <form action="servlet/Register" method="post">
         <fieldset style="width:20%; background-color:#ccffeb">
            <h2 align="center">Registration form</h2><hr>
            <table>
               <tr>
                  <td>TId</td>
                  <td><input type="text" name="TId" required /></td>
               </tr>
               <tr>
                  <td>Name</td>
                  <td><input type="text" name="Name" required /></td>
               </tr>
               <tr>
                  <td>Address</td>
                  <td><textarea name="address" placeholder="Enter address
here..."></textarea></td>
               </tr>
```

```
                <tr>
                    <td><input type="reset" value="Reset"/></td>
                    <td><input type="submit" value="Register"/></td>
                </tr>
            </table>
        </fieldset>
    </form>
  </body>
</html>
```

Java file

```
import java.io.*;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class Register extends HttpServlet
{
   public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      int id = Integer.parseInt(request.getParameter("TId"));
      String name = request.getParameter("name");
      String address = request.getParameter("address");
      try
      {
         //load the driver
         Class.forName("oracle.jdbc.driver.OracleDriver");
         //create connection object
         Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","local","test");
         // create the prepared statement object
         PreparedStatement ps=con.prepareStatement("insert into TeacherDetails values(?,?,?)");

         ps.setInt(1, id);
         ps.setString(2,name);
         ps.setString(3,address);

         int i = ps.executeUpdate();
         if(i>0)
         out.print("You are successfully registered...");
      }
      catch (Exception ex)
      {  ex.printStackTrace();
      }
```

```
         }
         out.close();
      }
   }
```

Q3 Write a JSP program which takes Muliiplicant and Multiplier from user as input and after clicking on "Multiply" button it will display Multiplicant, *, Multiplier and product.

```
<html>
<body bgcolor="cyan">
<br>
<form
action="http://localhos8080/nult.jsp"
method="GET">
<center>
<h2>Enter the First Number :<input type =
text name="t1">
<br>Enter the Second Number: <input type
= text name= "t2">
<br>
<br>
<input type="Submit" value="Mult">
<input type="reset" value="Cancel">
</form>
</body>
```

```
</html>
//Mult.jsp
e page language="Java">
<html>
//Mult.jsp
e page language="Java">
<html>
<body bgcolor="pink">
int x
Integer.parseInt(request.getParameter("t1"));
int y
Integer.parseInt(request.getParameter("t2"));
=
int z = x y;
out.println("Multiplication = "+2);
%>
</body>
</html>
```

Q4 Write a Servlet program which display the current data and time.

```
import java.io.*;
import javax.servlet.*;

public class DateSrv extends GenericServlet
{
   //implement service()
   public void service(ServletRequest req, ServletResponse res) throws IOException,
ServletException
   {
      //set response content type
      res.setContentType("text/html");
      //get stream obj
      PrintWriter pw = res.getWriter();
      //write req processing logic
      java.util.Date date = new java.util.Date();
      pw.println("<h2>"+"Current Date & Time: " +date.toString()+"</h2>");
      //close stream object
      pw.close();
   }
}
```

Q5 Write a Servlet program to get information about the server such as name, server port number and server version.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
public class ServerSnoop extends GenericServlet {
public void service(ServletRequest req, ServletResponse res)
                        throws ServletException, IOException {
res.setContentType("text/plain");
PrintWriter out = res.getWriter();
out.println("req.getServerName(): " + req.getServerName());
out.println("req.getServerPort(): " + req.getServerPort());
out.println("getServletContext().getServerInfo(): " +
    getServletContext().getServerInfo());
out.println("getServerInfo() name: " +
    getServerInfoName(getServletContext().getServerInfo()));
out.println("getServerInfo() version: " +
    getServerInfoVersion(getServletContext().getServerInfo()));
out.println("getServletContext().getAttribute(\"attribute\"): " +
    getServletContext().getAttribute("attribute"));
}
private String getServerInfoName(String serverInfo) {
int slash = serverInfo.indexOf('/');
if (slash == -1) return serverInfo;
else return serverInfo.substring(0, slash);
}
private String getServerInfoVersion(String serverInfo) {
int slash = serverInfo.indexOf('/');
if (slash == -1) return null;
else return serverInfo.substring(slash + 1);
}
}
```

Q6 Write JSP program to demonstrate all three scripting tags used in JSP.

```
<html>
<body>
<center>
<%@ page language="java" %>
<%! int count = 0; %>
<% count++; %>
Welcome! You are visitor number
<= count %>
</center>
</body>
</html>
```

Q6-a JSP program for addition of two numbers.

```
<html>
<body bgcolor="cyan">
< int x, y, z ;\%>
<%
x = 100
y = 400
z = x + y
%>
<%= " x ="4 x%><br>
<%= " y = "+ y%><br>
<%= " z ="+Z\%>
</body>
</html>
```

Q 6 -b Program to display date and message.

```
//MyDate.Hisp
<html>
<head>
<title>A Comment Test
</title>
</head>
<body>
<p>
Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

Q7 Write JSP program to accept student name, address and class from HTML and display it on another page.

```
<html>
<body>
<form method=get action="Student.jsp">
<fieldset>
<legend>Enter Student Details...!!!</legend>
Enter Roll No :            <input type=text
name=rno><br><br>
Enter Student Name:  <input type=text name=sname><br><br>
Enter Gender:             <input type=text
name=gen><br><br>
Computer Knowledge:<input type=text name=ckn><br><br>
Enter Class:                <input
type=text name=cl><br><br>
</fieldset>
<div align=center>
<input type=submit value="Save">
</div>
</form>
</body>
</html>
```

student.html

```
<html>
<body>
<%@ page import="java.io.*;" %>
<%! int rno;
String name,gen,ckn,cl; %>
<%
try      {
rno=Integer.parseInt(request.getParameter("rno"));
name=request.getParameter("sname");
gen=request.getParameter("gen");
ckn=request.getParameter("ckn");
46       cl=request.getParameter("cl");
out.println("Roll No: "+rno+"<br>");
out.println("Name : "+name+"<br>");
out.println("Gender :"+gen+"<br>");
out.println("Comp Knowledge : "+ckn+"<br>");
out.println("Class :"+cl+"<br>");
```

```
}
catch(Exeption e)
{
out.println(e);
}
%>
</body>
</html>
```