

**T. Y. B. Sc.
COMPUTER SCIENCE
SEMESTER-VI**

**NEW SYLLABUS
CBCS PATTERN**

OBJECT ORIENTED PROGRAMMING USING JAVA-II

**Dr. Ms. MANISHA BHARAMBE
Ms. MANISHA GADEKAR**



NIRALI[®]
PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

OBJECT ORIENTED PROGRAMMING USING JAVA-II

**For T.Y.B.Sc. Computer Science : Semester – VI
[Course Code CS 365 : Credits – 2]**

CBCS Pattern

As Per New Syllabus, Effective from June 2021

Dr. Ms. Manisha Bharambe

M.Sc. (Comp. Sci.), M.Phil. Ph.D. (Comp. Sci.)

Vice Principal, Associate Professor, Department of Computer Science

MES's Abasaheb Garware College

Pune

Ms. Manisha Gadekar

M.C.A. (Science), UGC-NET

Assistant Professor, Annasaheb Magar College

Pune

Price ₹ 330.00



N5950

OBJECT ORIENTED PROGRAMMING USING JAVA - II**ISBN 978-93-5451-261-2**

First Edition : January 2022
© **Authors**

The text of this publication, or any part thereof, should not be reproduced or transmitted in any form or stored in any computer storage system or device for distribution including photocopy, recording, taping or information retrieval system or reproduced on any disc, tape, perforated media or other information storage device etc., without the written permission of Authors with whom the rights are reserved. Breach of this condition is liable for legal action.

Every effort has been made to avoid errors or omissions in this publication. In spite of this, errors may have crept in. Any mistake, error or discrepancy so noted and shall be brought to our notice shall be taken care of in the next edition. It is notified that neither the publisher nor the authors or seller shall be responsible for any damage or loss of action to any one, of any kind, in any manner, therefrom. The reader must cross check all the facts and contents with original Government notification or publications.

Published By :
NIRALI PRAKASHAN

Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS

Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041

DISTRIBUTION CENTRES**PUNE**

Nirali Prakashan
(For orders outside Pune)

S. No. 28/27, Dhayari Narhe Road, Near Asian College
Pune 411041, Maharashtra
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

Nirali Prakashan
(For orders within Pune)

119, Budhwar Peth, Jogeshwari Mandir Lane
Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

MUMBAI**Nirali Prakashan**

Rasdhara Co-op. Hsg. Society Ltd., 'D' Wing Ground Floor, 385 S.V.P. Road
Girgaum, Mumbai 400004, Maharashtra
Mobile : 7045821020, Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

DISTRIBUTION BRANCHES**DELHI****Nirali Prakashan**

Room No. 2 Ground Floor
4575/15 Omkar Tower, Agarwal Road
Darya Ganj, New Delhi 110002
Mobile : 9555778814/9818561840
Email : delhi@niralibooks.com

BENGALURU**Nirali Prakashan**

Maitri Ground Floor, Jaya Apartments,
No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003
Karnataka; Mob : 9686821074
Email : bengaluru@niralibooks.com

NAGPUR**Nirali Prakashan**

Above Maratha Mandir, Shop No. 3,
First Floor, Rani Jhansi Square,
Sitabuldi Nagpur 440012 (MAH)
Tel : (0712) 254 7129
Email : nagpur@niralibooks.com

KOLHAPUR**Nirali Prakashan**

438/2, Bhosale Plaza, Ground Floor
Khasbag, Opp. Balgopal Talim
Kolhapur 416 012, Maharashtra
Mob : 9850046155
Email : kolhapur@niralibooks.com

JALGAON**Nirali Prakashan**

34, V. V. Golani Market, Navi Peth,
Jalgaon 425001, Maharashtra
Tel : (0257) 222 0395
Mob : 94234 91860
Email : jalgaon@niralibooks.com

SOLAPUR**Nirali Prakashan**

R-158/2, Avanti Nagar, Near Golden
Gate, Pune Naka Chowk
Solapur 413001, Maharashtra
Mobile 9890918687
Email : solapur@niralibooks.com

marketing@pragationline.com | www.pragationline.com

Also find us on  www.facebook.com/niralibooks

Preface ...

We take an opportunity to present this Text Book on "**Object Oriented Programming using Java-II**" to the students of Third Year B.Sc. (Computer Science) Semester-VI as per the New Syllabus, June 2021.

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts. The book covers theory of Collections, Multithreading, Database Programming, Servlets and JSP, and Spring Framework.

A special word of thank to Shri. Dineshbhai Furia, and Mr. Jignesh Furia for showing full faith in us to write this text book. We also thank to Mr. Amar Salunkhe and Mr. Akbar Shaikh of M/s Nirali Prakashan for their excellent co-operation.

We also thank Ms. Chaitali Takle, Mr. Ravindra Walodare, Mr. Sachin Shinde, Mr. Ashok Bodke, Mr. Moshin Sayyed and Mr. Nitin Thorat.

Although every care has been taken to check mistakes and misprints, any errors, omission and suggestions from teachers and students for the improvement of this text book shall be most welcome.

Authors



Syllabus ...

- | | |
|---|----------------------|
| 1. Collections | (6 Lectures) |
| <ul style="list-style-type: none">• Introduction to the Collection Framework• List - ArrayList, LinkedList• Set - HashSet, TreeSet,• Map - HashMap and TreeMap• Interfaces such as Comparator, Iterator, ListIterator, Enumeration | |
| 2. Multithreading | (6 Lectures) |
| <ul style="list-style-type: none">• What are Threads?• Life cycle of Thread• Creating Threads:<ul style="list-style-type: none">◦ Thread Class◦ Runnable Interface• Thread Priorities• Running Multiple Threads• Synchronization and Interthread Communication | |
| 3. Database Programming | (6 Lectures) |
| <ul style="list-style-type: none">• The Design of jdbc• Types of Drivers• Executing SQL Statements, Query Execution• Scrollable and Updatable Resultset | |
| 4. Servlets and JSP | (12 Lectures) |
| <ul style="list-style-type: none">• Introduction to Servlet and Hierarchy of Servlet• Life Cycle of Servlet• Handing Get and Post Request (HTTP)• Handling Data from HTML to Servlet• Retrieving Data from Database to Servlet• Session Tracking:<ul style="list-style-type: none">◦ User Authorization◦ URL Rewriting◦ Hidden Form Fields◦ Cookies and HttpSession• Introduction to JSP, Life Cycle of JSP• Implicit Objects | |

- Scripting Elements:
 - Declarations
 - Expressions
 - Scriptlets
 - Comments
- JSP Directives - Page Directive, Include Directive
- Mixing Scriptlets and HTML
- JSP Actions:
 - `jsp:forward`
 - `jsp:include`
 - `jsp:useBean`
 - `jsp:setProperty`
 - `jsp:getProperty`

5. Spring Framework

(6 Lectures)

- Introduction of Spring Framework
- Spring Modules / Architecture
- Spring Applications
- Spring MVC
- Spring MVC Forms, Validation



Contents ...

| | |
|--------------------------------|-------------------|
| 1. Collections | 1.1 – 1.74 |
| 2. Multithreading | 2.1 – 2.52 |
| 3. Database Programming | 3.1 – 3.48 |
| 4. Servlets and JSP | 4.1 – 4.98 |
| 5. Spring Framework | 5.1 – 5.48 |

■ ■ ■

Collections

Objectives...

- To understand Collections and Collection Framework
- To study Collection Interfaces and Classes
- To learn Set and List Interfaces
- To Study Map and Comparator Interfaces

1.0 INTRODUCTION

- 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.
- Java Collection framework is a collection of interfaces and classes used to storing and processing a group of individual objects as a single unit.
- The Java Collection framework holds several classes that provide a large number of methods to store and process a group of objects.
- 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.

1.1 OVERVIEW TO COLLECTION FRAMEWORK [April 17, Oct. 18]

- A Collection, as its name implied, is simply an object that holds a collection (or a group or a container) of objects. Each item in a collection is called an element.

- In short, Collection is simply an object that groups multiple elements into a single unit. All the operations that we perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.
- The Java Collections framework is a collection of interfaces and classes which helps in storing and processing the data efficiently.
- Java Collection means a single unit of objects. A collection represents a group of objects, known as its elements.
- A collections framework is a unified architecture for representing and manipulating Collections.
- All Collections frameworks has:
 1. **Interfaces:** Interfaces are abstract data types that represent collections. The Collection interface is available inside the java.util package. The Collection interface (java.util.Collection) root interface in the Collections hierarchy.
 2. **Implementations i.e., Classes:** Java provides core implementation classes for Collections. The Java Collection framework holds several classes that provide a large number of methods to store and process a group of objects. We can use them to create different types of collections in our program.
 3. **Algorithms:** Algorithms are the useful methods to provide some common functionality such as searching and sorting.

1.1.1 Benefits of Collection Framework

- The Java Collections Framework provides the following benefits:
 1. **Reduces programming effort:** It comes with almost all common types of collections and useful methods to iterate and manipulate the data. So we can concentrate more on business logic rather than designing our collection APIs.
 2. **Increases program speed and quality:** Using core collection classes that are well tested increases our program quality rather than using any home developed data structure. Because we are freed from the work of writing our own data structures, we will have more time to devote to improving program's quality and performance.
 3. **Allows interoperability among unrelated APIs:** The collection interfaces are the dialect by which APIs pass collections back and forth.
 4. **Reduces effort to learn and to use new APIs:** Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.

5. **Reduces effort to design new APIs:** This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.
 6. **Fosters software reuse:** Like any New data structures that conform to the standard collection interfaces are also nature reusable.

1.1.2 Collections Framework Hierarchy

[April 18, 19]

- Fig. 1.1 shows Collections Framework hierarchy.

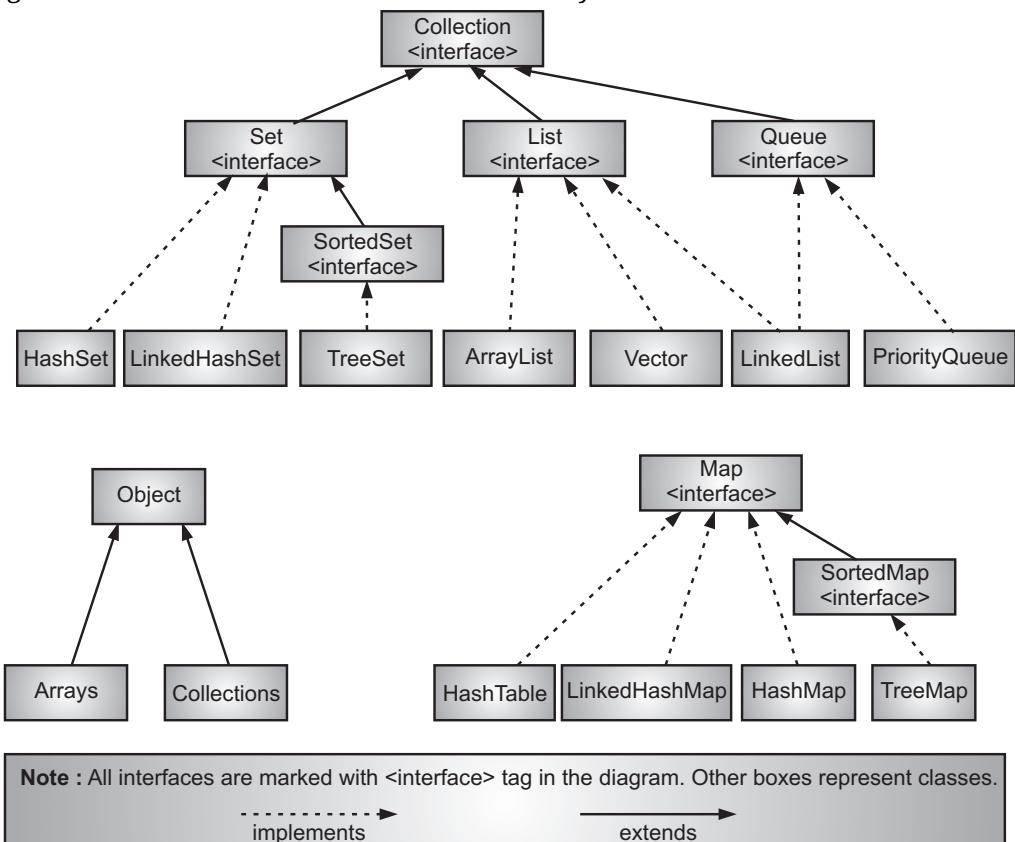


Fig. 1.1: Collections Framework Hierarchy

- The Collections Framework defines several interfaces. The collection interfaces are necessary because they determine the fundamental nature of the collection classes.
 - The core collection interfaces are:
[April 18]
 1. **Collection:** It is a root of the collection hierarchy. A collection represents a group of objects, known as its elements. It enables us to work with groups of objects.
 2. **Set:** It is a collection that cannot contain the duplicate elements. It means that set handles unique elements. It extends collection.

- 3. **List:** List is an ordered collection called as sequence. List may contain duplicate elements. The user of a list generally has precise control over where the list each element is inserted and their integer index (positions), e.g. vector. This extends from collection.
 - 4. **Queue:** The java.util.Queue interface is subtype of the java.util.Collection interface. It represents an ordered list of objects. A queue is designed to have elements inserted at the end of the queue and elements removed from the beginning of the queue. Just like a queue in a supermarket.
 - 5. **Map:** An object that maps keys to values maps cannot contain duplicate keys. Each key can map to at the most one value. e.g. Hashtable. The map cannot extend collection, but maps can be viewed as collections.
 - 6. **SortedSet:** A set which maintains its elements in ascending order. Various additional operations are provided here to take the advantage of the ordering. The sorted sets are used for naturally ordered set, e.g. word lists or rolls of members. This extends the set.
 - 7. **SortedMap:** A map that maintains its mappings in ascending key order. This is the map analog of sorted set. Sorted maps are used for naturally ordered collections of key/value pairs like dictionaries, telephone directories etc. It extends Map.
- The java.util package also provides several useful concrete implementations classes some of them are:
- [April 18]
- 1. **HashSet:** A set implemented using a hashtable. A good general purpose implementation for which adding, searching and deleting are mostly sensitive.
 - 2. **TreeSet:** A sortedset implemented using a balanced binary tree. It is slower to search and modify than HashSet. But it always keeps the elements sorted.
 - 3. **ArrayList:** A list implemented using a resizable array. It is very expensive to add or delete an element near the beginning if list is large. But it is cheap to create and fast for random access.
 - 4. **LinkedList:** A doubly-linked List implementation. It is useful for queues. Modification is cheap at any size but random access is slow.
 - 5. **HashMap:** A Hash table implementation of map. It is useful collection which is cheap for its lookup and insertion time.
 - 6. **TreeMap:** It is an implementation of Sorted Map. It is using a balanced binary tree that keeps its elements ordered by key. It is very useful for ordered data set. Searching is done through key.

1.1.3 Collection Classes

- Collection classes are the utility classes. It has all static methods which are used to specify different applications. The methods takes parameter of no_threadsafe method and returns a thread safe method.

- The collection class defines a range of static utility method that operates on the collections. The utility methods are classified into two groups:
 - Those methods that provide wrapped collections.
 - Those methods that does not provide wrapped collections.
- The wrapped collections allow us to present different view of collections by synchronizing access to the collection or just by removing all the methods which could modify the collection.
- There are various collection classes which are shown below:

| Class | Description |
|------------------------|--|
| AbstractCollection | Implements various collection interfaces. |
| AbstractList | Extends AbstractCollection and implements various List interfaces. |
| AbstractSequentialList | Extends AbstractList which uses sequential access of its elements. |
| LinkedList | Implements linked list by extending AbstractSequentialList. |
| ArrayList | Implements dynamic array by extending AbstractList. |
| AbstractSet | Extends AbstractCollection and implements various set interface. |
| HashSet | Extends AbstractSet for use with hash table. |
| LinkedHashSet | Extends HashSet to allow insertion order iterations. |
| TreeSet | Implements a set stored in a tree. It extends the AbstractSet. |

- The above classes of collection are shown in Fig. 1.2 as a full type tree for collections.

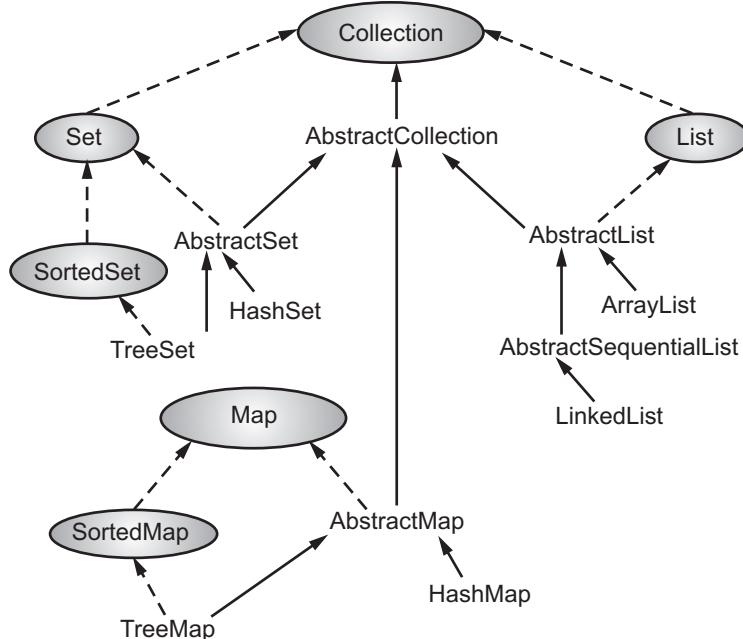


Fig. 1.2: Tree of Collection Classes

1.1.4 Collection Interface

[April 18]

- The Collection interface is used to pass around collections of objects where maximum generality is desired.
- For example, by convention all general-purpose collection implementations have a constructor that takes a Collection argument. This constructor, known as a conversion constructor, initializes the new collection to contain all of the elements in the specified collection, whatever the given collection's sub-interface or implementation type. In other words, it allows us to convert the collection's type.
- Suppose, for example, that we have a Collection<String> c, which may be a List, a Set, or another kind of Collection. This idiom creates a new ArrayList (an implementation of the List interface), initially containing all the elements in c.

```
List<String> list = new ArrayList<String>(c);
```

Methods for Collection Interface:

| Method | Description |
|-----------------------------------|---|
| boolean add(Object obj) | Adds obj to the invoking collection. Returns true if obj was added to the collection. Returns false if obj is already a member of the collection, or if the collection does not allow duplicates. |
| boolean addAll(Collection c) | Adds all the elements of c to the invoking collection. Returns true if the operation succeeded (i.e., the elements were added). Otherwise, returns false. |
| void clear() | Removes all elements from the invoking collection. |
| boolean contains(Object obj) | Returns true if obj is an element of the invoking collection. Otherwise, returns false. |
| boolean containsAll(Collection c) | Returns true if the invoking collection contains all elements of c. Otherwise, returns false. |
| boolean equals(Object obj) | Returns true if the invoking collection and obj are equal. Otherwise, returns false. |
| int hashCode() | Returns the hash code for the invoking collection. |
| boolean isEmpty() | Returns true if the invoking collection is empty. Otherwise, returns false. |
| Iterator iterator() | Returns an iterator for the invoking collection. |

contd. ...

| | |
|---|--|
| <code>boolean remove(Object obj)</code> | Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false. |
| <code>boolean removeAll(Collection c)</code> | Removes all elements of c from the invoking collection. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false. |
| <code>boolean retainAll(Collection c)</code> | Removes all elements from the invoking collection except those in c. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false. |
| <code>int size()</code> | Returns the number of elements held in the invoking collection. |
| <code>Object[] toArray()</code> | Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements. |
| <code>Object[] toArray(Object array[])</code> | Returns an array containing only those collection elements whose type matches that of array. |

Program 1.1: Program for collection interface.

```

import java.util.*;
public class CollectionInterfaceExample
{
    public static void main(String[] args)
    {
        List list_1 = new ArrayList();
        List<String> list_2 = new ArrayList<String>();
        list_1.add(10);
        list_1.add(20);
        list_2.add("BTech");
        list_2.add("Smart");
        list_2.add("Class");
        list_1.addAll(list_2);
        System.out.println("Elements of list_1: " + list_1);
        System.out.println("Search for BTech: " + list_1.contains("BTech"));
        System.out.println("Search for list_2 in list_1: " +
                           list_1.containsAll(list_2));
    }
}

```

```
System.out.println("Check whether list_1 and list_2 are equal: " +
                   list_1.equals(list_2));
System.out.println("Check is list_1 empty: " + list_1.isEmpty());
System.out.println("Size of list_1: " + list_1.size());
System.out.println("HashCode of list_1: " + list_1.hashCode());
list_1.remove(0);
System.out.println(list_1);
list_1.addAll(list_2);
System.out.println(list_1);
list_1.removeAll(list_2);
System.out.println(list_1);
list_2.clear();
System.out.println(list_2);
}
```

```
}
```

Output:

```
Elements of list_1: [10, 20, BTech, Smart, Class]
Search for BTech: true
Search for list_2 in list_1: true
Check whether list_1 and list_2 are equal: false
Check is list_1 empty: false
Size of list_1: 5
HashCode of list_1: -764556324
[20, BTech, Smart, Class]
[BTech, Smart, Class]
[]
[]
```

1.2 LIST

- A List is an ordered Collection (sometimes called a sequence). The List interface extends Collection and declares the behavior of a collection that stores a sequence of elements.
 - Lists may contain duplicate elements. Elements can be inserted or accessed by their position in the list, using a zero-based index.
-

Methods of List Interface:

| Method | Description |
|--|--|
| <code>void add(int index, Object obj)</code> | Inserts obj into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. |
| <code>Object get(int index)</code> | Returns the object stored at the specified index within the invoking collection. |
| <code>boolean addAll(int index, Collection c)</code> | Inserts all elements of c into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise. |
| <code>int indexOf(Object obj)</code> | Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list, .1 is returned. |
| <code>int lastIndexOf(Object obj)</code> | Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list, .1 is returned. |
| <code>ListIterator listIterator()</code> | Returns an iterator to the start of the invoking list. |
| <code>ListIterator listIterator(int index)</code> | Returns an iterator to the invoking list that begins at the specified index. |
| <code>Object remove(int index)</code> | Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one. |
| <code>Object set(int index, Object obj)</code> | Assigns obj to the location specified by index within the invoking list. |
| <code>List subList(int start, int end)</code> | Returns a list that includes elements from start to end. The 1 in the invoking list. Elements in the returned list are also referenced by the invoking object. |

Program 1.2: Program for List interface.

```
import java.util.*;
public class ListDemo
{
    public static void main(String[] args)
    {
        List a1 = new ArrayList();
        a1.add("Zuber");
        a1.add("Mahesh");
        a1.add("Ayush");
        System.out.println(" ArrayList Elements:");
        System.out.print("\t" + a1);
        List l1 = new LinkedList();
        l1.add("Zuber ");
        l1.add("Mahesh ");
        l1.add("Ayush");
        System.out.println();
        System.out.println(" LinkedList Elements:");
        System.out.print("\t" + l1);
    }
}
```

Output:

```
ArrayList Elements:
[Zuber, Mahesh, Ayush]
LinkedList Elements:
[Zuber, Mahesh, Ayush]
```

1.2.1 ArrayList

[April 18]

- ArrayList class uses a dynamic array for storing the elements.
- It extends AbstractList class and implements List interface.

Constructors:

1. `ArrayList()` constructor builds an empty array list.
2. `ArrayList(Collection c)` constructor builds an array list that is initialized with the elements of the collection c.
3. `ArrayList(int capacity)` constructor builds an array list that has the specified initial capacity. The capacity is the size of the underlying array that is used to store the elements. The capacity grows automatically as elements are added to an array list.

Methods of ArrayList:

| Method | Description |
|--|---|
| <code>void add(int index, Object element)</code> | Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 index > size()). |
| <code>boolean add(Object o)</code> | Appends the specified element to the end of this list. |
| <code>boolean addAll(Collection c)</code> | Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null. |
| <code>boolean addAll(int index, Collection c)</code> | Inserts all of the elements in the specified collection into this list, starting at the specified position. It throws NullPointerException if the specified collection is null. |
| <code>void clear()</code> | Removes all of the elements from this list. |
| <code>Object clone()</code> | Returns a shallow copy of this ArrayList. |
| <code>boolean contains(Object o)</code> | Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null? e==null: o.equals(e)). |
| <code>void ensureCapacity(int minCapacity)</code> | Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument. |
| <code>Object get(int index)</code> | Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 index >= size()). |
| <code>int indexOf(Object o)</code> | Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element. |

contd. ...

| | |
|---|--|
| <code>int lastIndexOf(Object o)</code> | Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |
| <code>Object remove(int index)</code> | Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 index >= size()). |
| <code>protected void removeRange(int fromIndex, int toIndex)</code> | Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive. |
| <code>Object set(int index, Object element)</code> | Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 index >= size()). |
| <code>int size()</code> | Returns the number of elements in this list. |
| <code>Object[] toArray()</code> | Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null. |
| <code>Object[] toArray(Object[] a)</code> | Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array. |
| <code>void trimToSize()</code> | Trims the capacity of this ArrayList instance to be the list's current size. |

Program 1.3: Program for ArrayList.

```

import java.util.*;
class DemoArrayList
{
    public static void main(String args[])
    {
        ArrayList b = new ArrayList();
        b.add("Pune");
        b.add("Patna");
        b.add("Parbhani");
        b.add("Satara");
        System.out.println(b);
    }
}

```

```

        b.remove("patna");
        System.out.println(b);
    }
}

```

Output:

[Pune, Patna, Parbhani, Satara]
[Pune, Patna, Parbhani, Satara]

1.2.2 LinkedList**[April 18]**

- The LinkedList class extends AbstractSequentialList and implements the List interface.
- It provides a linked-list data structure.

Constructors:

- LinkedList() constructor builds an empty linked list.
- LinkedList(Collection c) constructor builds a linked list that is initialized with the elements of the collection c.

Methods for Linked List:

| Method | Description |
|---|---|
| void add(int index, Object element) | Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 index > size()). |
| boolean add(Object o) | Appends the specified element to the end of this list. |
| boolean addAll(Collection c) | Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null. |
| boolean addAll(int index, Collection c) | Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null. |
| void addFirst(Object o) | Inserts the given element at the beginning of this list. |

contd. ...

| | |
|---|---|
| <code>void addLast(Object o)</code> | Appends the given element to the end of this list. |
| <code>void clear()</code> | Removes all of the elements from this list. |
| <code>Object clone()</code> | Returns a shallow copy of this LinkedList. |
| <code>boolean contains(Object o)</code> | Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (<code>o==null? e==null: o.equals(e)</code>). |
| <code>Object get(int index)</code> | Returns the element at the specified position in this list. Throws <code>IndexOutOfBoundsException</code> if the specified index is out of range (<code>index < 0 index >= size()</code>). |
| <code>Object getFirst()</code> | Returns the first element in this list. Throws <code>NoSuchElementException</code> if this list is empty. |
| <code>Object getLast()</code> | Returns the last element in this list. Throws <code>NoSuchElementException</code> if this list is empty. |
| <code>int indexOf(Object o)</code> | Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element. |
| <code>int lastIndexOf(Object o)</code> | Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |
| <code>ListIterator listIterator(int index)</code> | Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Throws <code>IndexOutOfBoundsException</code> if the specified index is out of range (<code>index < 0 index >= size()</code>). |
| <code>Object remove(int index)</code> | Removes the element at the specified position in this list. Throws <code>NoSuchElementException</code> if this list is empty. |
| <code>boolean remove(Object o)</code> | Removes the first occurrence of the specified element in this list. Throws <code>NoSuchElementException</code> if this list is empty. Throws <code>IndexOutOfBoundsException</code> if the specified index is out of range (<code>index < 0 index >= size()</code>). |

contd. ...

| | |
|--|--|
| <code>Object removeFirst()</code> | Removes and returns the first element from this list. Throws NoSuchElementException if this list is empty. |
| <code>Object removeLast()</code> | Removes and returns the last element from this list. Throws NoSuchElementException if this list is empty. |
| <code>Object set(int index, Object element)</code> | Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 index >= size()). |
| <code>int size()</code> | Returns the number of elements in this list. |
| <code>Object[] toArray()</code> | Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null. |
| <code>Object[] toArray(Object[] a)</code> | Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array. |

Program 1.4: Program for LinkedList.

```

import java.util.*;
class DemoLinkedList
{
    public static void main(String args[])
    {
        LinkedList linkedlist1 = new LinkedList();
        linkedlist1.add("Item 2");
        linkedlist1.add("Item 3");
        linkedlist1.add("Item 4");
        linkedlist1.add("Item 5");
        linkedlist1.addFirst("Item 0");
        linkedlist1.addLast("Item 6");
        linkedlist1.add(1, "Item 1");
        System.out.println(linkedlist1);
        linkedlist1.remove("Item 6");
    }
}

```

```

        System.out.println(linkedlist1);
        linkedlist1.removeLast();
        System.out.println(linkedlist1);
        System.out.println("\nUpdating linked list items");
        linkedlist1.set(0, "Red");
        linkedlist1.set(1, "Blue");
        linkedlist1.set(2, "Green");
        linkedlist1.set(3, "Yellow");
        linkedlist1.set(4, "Purple");
        System.out.println(linkedlist1);
    }
}

```

Output:

```

[Item 0, Item 1, Item 2, Item 3, Item 4, Item 5, Item 6]
[Item 0, Item 1, Item 2, Item 3, Item 4, Item 5]
[Item 0, Item 1, Item 2, Item 3, Item 4]
Updating linked list items
[Red, Blue, Green, Yellow, Purple]

```

Difference between ArrayList and LinkedList:**[April 18]**

| Sr. No. | ArrayList | LinkedList |
|---------|---|--|
| 1. | ArrayList internally uses dynamic array to store the elements. | LinkedList internally uses doubly linked list to store the elements. |
| 2. | Manipulation with ArrayList 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 ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3. | ArrayList class can act as a list only because it implements List only. | LinkedList class can act as a list and queue both because it implements List and Deque interfaces. |
| 4. | ArrayList is better for storing and accessing data. | LinkedList is better for manipulating data. |

1.2.3 Vector**[April 15]**

- The Vector class is analogous to ArrayList. It is a legacy class which is made to implement List and so it works as a collection.

- Vector implements a dynamic array. It is similar to ArrayList, but with two differences:
 1. Vector is synchronized.
 2. Vector contains many legacy methods that are not part of the collections framework.
- Vector proves to be very useful if we don't know the size of the array in advance or we just need one that can change sizes over the lifetime of a program.

Constructors:

1. `Vector()` constructor creates empty vector. Its size is 10 and capacity is 0. It is analogous to `ArrayList`.
2. `Vector(Collection c)` constructor creates a vector containing the elements of the collection `c`. The elements are accessed in the order returned by the collections iterator.
3. `Vector(int capacity)` constructor creates an empty vector with initial capacity.
4. `Vector(int capacity, int incr)` constructor creates an empty vector with initial capacity specified by `capacity` and capacity increment specified by `incr`.

Methods for Vector:

| Method | Description |
|---|---|
| <code>void addElement(Object element)</code> | Inserts the element at the end of the Vector. |
| <code>int capacity()</code> | Returns the current capacity of the vector. |
| <code>int size()</code> | Returns the current size of the vector. |
| <code>void setSize(int size)</code> | Changes the existing size with the specified size. |
| <code>Boolean contains(Object element)</code> | Checks whether the specified element is present in the Vector. If the element is been found it returns true else false. |
| <code>Boolean containsAll(Collection c)</code> | Returns true if all the elements of collection <code>c</code> are present in the Vector. |
| <code>Object elementAt(int index)</code> | Returns the element present at the specified location in Vector. |
| <code>Object firstElement()</code> | Used for getting the first element of the vector. |
| <code>Object lastElement()</code> | Returns the last element of the array. |
| <code>Object get(int index)</code> | Returns the element at the specified index. |
| <code>boolean isEmpty()</code> | Returns true if Vector doesn't have any element. |
| <code>boolean removeElement(Object element)</code> | Removes the specified element from vector. |
| <code>boolean removeAll(Collection c)</code> | Removes all those elements from vector which are present in the Collection <code>c</code> . |
| <code>void setElementAt(Object element, int index)</code> | Updates the element of specified index with the given element. |

Program 1.5: Program for Vector.

```
import java.util.*;
class VectorDemo
{
    public static void main(String args[])
    {
        Vector vector = new Vector(5);
        System.out.println("Size: " + vector.size());
        System.out.println("Capacity: " + vector.capacity());
        vector.addElement(new Integer(0));
        vector.addElement(new Integer(1));
        vector.addElement(new Integer(2));
        vector.addElement(new Integer(3));
        vector.addElement(new Integer(4));
        vector.addElement(new Integer(5));
        vector.addElement(new Integer(6));
        vector.addElement(new Integer(5));
        vector.addElement(new Integer(6));
        vector.addElement(new Double(3.14159));
        vector.addElement(new Float(3.14159));
        System.out.println("Capacity: " + vector.capacity());
        System.out.println("Size: " + vector.size());
        System.out.println(vector);
        System.out.println("First item:"+(Integer)vector.firstElement());
        System.out.println("Last item: "+(Float) vector.lastElement());
        if(vector.contains(new Integer(3)))
            System.out.println("Found 3");
    }
}
```

Output:

```
Size: 0
Capacity: 5
Capacity: 20
Size: 11
[0, 1, 2, 3, 4, 5, 6, 5, 6, 3.14159, 3.14159]
First item:0
Last item: 3.14159
Found 3
```

1.2.4 Stack

- Stack is a subclass of Vector that implements a standard Last-In, First-Out (LIFO) stack.
- Stack only defines the default constructor Stack(), which creates an empty stack. Stack includes all the methods defined by Vector, and adds several of its own.

Methods for Stack:

| Method | Description |
|-----------------------------|---|
| boolean empty() | Tests if this stack is empty. Returns true if the stack is empty, and returns false if the stack contains elements. |
| Object peek() | Returns the element on the top of the stack, but does not remove it. |
| Object pop() | Returns the element on the top of the stack, removing it in the process. |
| Object push(Object element) | Pushes element onto the stack. element is also returned. |
| int search(Object element) | Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, .1 is returned. |

Program 1.6: Program for Stack.

```

import java.util.*;
class DemoStack
{
    public static void main(String args[])
    {
        Stack stack1 = new Stack();
        try
        {
            stack1.push(new Integer(0));
            stack1.push(new Integer(1));
            stack1.push(new Integer(2));
            stack1.push(new Integer(3));
            stack1.push(new Integer(4));
            stack1.push(new Integer(5));
            stack1.push(new Integer(6));
            System.out.println("Pop->"+(Integer) stack1.pop());
            System.out.println("Pop->"+(Integer) stack1.pop());
        }
    }
}

```

```

        System.out.println("Pop->" + (Integer) stack1.pop());
        System.out.println("Pop->" + (Integer) stack1.pop());
        System.out.println("Pop->" + (Integer) stack1.pop());
        System.out.println("Pop->" + (Integer) stack1.pop());
        System.out.println("Pop->" + (Integer) stack1.pop());
    }
    catch (EmptyStackException e) {}
}
}

Output:
Pop->6
Pop->5
Pop->4
Pop->3
Pop->2
Pop->1
Pop->0

```

1.2.5 Queue

- Queue interface is in java.util package of java. Queue is a linear collection that supports element insertion and removal at both ends.
- A Queue interface extends the Collection interface to define an ordered collection for holding elements in a FIFO (first-in-first-out) manner to process them i.e. in a FIFO queue the element that is inserted firstly will also get removed first. Queue does not require any fix dimension like String array and int array.
- Besides basic collection operations, queues also provide additional operations such as insertion, removal, and inspection.

Methods for Queue:

| Method | Description |
|--------------|--|
| poll() | Returns and removes the element at the front end of the container. |
| remove() | Removes element at the head of the queue and throws NoSuchElementException if queue is empty. |
| peek() | Returns the element at the head of the queue without removing it. Returns null if queue is empty. |
| element() | Same as peek(), but throws NoSuchElementException if queue is empty. |
| offer(E obj) | Adds object to queue. This method is preferable to add() method since this method does not throw an exception when the capacity of the container is full since it returns false. |

- The remove() and poll() methods differ only in their behavior when the queue is empty: the remove() method throws an exception, while the poll() method returns null.

Program 1.7: In following program we use the LinkedList class which implements the Queue interface and add items to it using both the add() and offer() methods.

```
import java.util.*;
public class DemoQueue
{
    public static void main(String[] args)
    {
        Queue oqueue=new LinkedList();
        oqueue.add("1");
        oqueue.add("2");
        oqueue.add("3");
        oqueue.add("4");
        oqueue.add("5");
        Iterator itr=oqueue.iterator();
        System.out.println("Initial Size of Queue:"+oqueue.size());
        while(itr.hasNext())
        {
            String iteratorValue=(String)itr.next();
            System.out.println("Queue Next Value:"+iteratorValue);
        }
        // get value and does not remove element from queue
        System.out.println("Queue peek:"+oqueue.peek());
        // get first value and remove that object from queue
        System.out.println("Queue poll:"+oqueue.poll());
        System.out.println("Final Size of Queue:"+oqueue.size());
    }
}
```

Output:

```
Initial Size of Queue:5
Queue Next Value:1
Queue Next Value:2
Queue Next Value:3
Queue Next Value:4
Queue Next Value:5
Queue peek:1
Queue poll:1
Final Size of Queue:4
```

1.3 SET

- Set interface defines a Set. A set is a collection that cannot contain duplicate elements. It models the mathematical set abstraction.
- The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
- Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

Methods for Set:

| Method | Description |
|---------------------|--|
| add() | Adds an object to the collection. |
| clear() | Removes all objects from the collection. |
| contains() | Returns true if a specified object is an element within the collection. |
| isEmpty() | Returns true if the collection has no elements. |
| Iterator iterator() | Returns an Iterator object for the collection which may be used to retrieve an object. |
| remove() | Removes a specified object from the collection. |
| size() | Returns the number of elements in the collection. |

Program 1.8: Program for Set.

```

import java.util.*;
public class DemoSet
{
    public static void main(String args[])
    {
        int count[] = {34, 22, 10, 60, 30, 22};
        Set<Integer> set = new HashSet<Integer>();
        Try
        {
            for(int i = 0; i<5; i++)
            {
                set.add(count[i]);
            }
            System.out.println(set);
            TreeSet sortedSet = new TreeSet<Integer>(set);
        }
    }
}

```

```

        System.out.println("The sorted list is:");
        System.out.println(sortedSet);
        System.out.println("The First element of the set is: "+
        (Integer)sortedSet.first());
        System.out.println("The last element of the set is: "+
        (Integer)sortedSet.last());
    }
    catch(Exception e){}
}
}

```

Output:

```

[34, 22, 10, 60, 30]
The sorted list is:
[10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60

```

1.3.1 HashSet**[April 16]**

- HashSet class extends AbstractSet and implements the Set interface.
- It creates a collection that uses a hash table for storage.
- 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 hash code.
- The hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically.

Constructors:**[April 16]**

1. HashSet() constructor form constructs a default hash set.
2. HashSet(Collection c) constructor initializes the hash set by using the elements of c.
3. HashSet(int capacity) constructor initializes the capacity of the hash set to capacity. The capacity grows automatically as elements are added to the Hash.
4. HashSet(int capacity, float fillRatio) initializes both the capacity and the fill ratio (also called load capacity) of the hash set from its arguments.

Methods for HashSet:

| Method | Description |
|----------------------------|--|
| boolean add(Object o) | Adds the specified element to this set if it is not already present. |
| void clear() | Removes all of the elements from this set. |
| Object clone() | Returns a shallow copy of this HashSet instance: the elements themselves are not cloned. |
| boolean contains(Object o) | Returns true if this set contains the specified element. |
| boolean isEmpty() | Returns true if this set contains no elements. |
| Iterator iterator() | Returns an iterator over the elements in this set. |
| boolean remove(Object o) | Removes the specified element from this set if it is present. |
| int size() | Returns the number of elements in this set (its cardinality). |

Program 1.9: Program for HashSet (output in sorted order).

```
import java.util.*;
class DemoHashset
{
    public static void main(String args[])
    {
        hashSet hashset1 = new HashSet();
        hashset1.add("Item 0");
        hashset1.add("Item 1");
        hashset1.add("Item 2");
        hashset1.add("Item 3");
        hashset1.add("Item 4");
        hashset1.add("Item 5");
        hashset1.add("Item 5");
        hashset1.add("Item 6");
        hashset1.add("Item 6");
        hashset1.add("Item 6");
        System.out.println(hashset1);
    }
}
```

Output:

[Item 0, Item 4, Item 3, Item 2, Item 1, Item 6, Item 5]

1.3.2 TreeSet**[April 17, 18; Oct. 17]**

- 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 c.
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.

Methods for TreeSet:

| Method | Description |
|--|--|
| void add(Object o) | Adds the specified element to this set if it is not already present. |
| boolean addAll(Collection c) | Adds all of the elements in the specified collection to this set. |
| void clear() | Removes all of the elements from this set. |
| Object clone() | Returns a shallow copy of this TreeSet instance. |
| Comparator comparator() | Returns the comparator used to order this sorted set, or null if this tree set uses its elements natural ordering. |
| boolean contains(Object o) | Returns true if this set contains the specified element. |
| Object first() | Returns the first (lowest) element currently in this sorted set. |
| SortedSet headSet(Object toElement) | Returns a view of the portion of this set whose elements are strictly less than toElement. |
| boolean isEmpty() | Returns true if this set contains no elements. |
| Iterator iterator() | Returns an iterator over the elements in this set. |
| Object last() | Returns the last (highest) element currently in this sorted set. |
| boolean remove(Object o) | Removes the specified element from this set if it is present. |
| int size() | Returns the number of elements in this set (its cardinality). |
| SortedSet subSet(Object fromElement, Object toElement) | Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. |
| SortedSet tailSet(Object fromElement) | Returns a view of the portion of this set whose elements are greater than or equal to fromElement. |

Program 1.10: Program for TreeSet.

```

import java.util.*;
class DemoTreeSet
{
    public static void main(String args[])
    {
        TreeSet treeset1 = new TreeSet();
        treeset1.add("Monday");
        treeset1.add("Tuesday");
        treeset1.add("Wednesday");
        treeset1.add("Thursday");
        treeset1.add("Friday");
        treeset1.add("Saturday");
        treeset1.add("Sunday");
        System.out.println(treeset1);
    }
}

```

Output:

[Friday, Monday, Saturday, Sunday, Thursday, Tuesday, Wednesday]

1.3.3 LinkedHashSet

- This class extends HashSet, but adds no members of its own. LinkedHashSet maintains a linked list of the entries in the set, in the order in which they were inserted. This allows insertion-order iteration over the set.
- LinkedHashSet is also an implementation of Set interface, it is similar to the HashSet and TreeSet except the below mentioned differences:
 1. HashSet doesn't maintain any kind of order of its elements.
 2. TreeSet sorts the elements in ascending order.
 3. LinkedHashSet maintains the insertion order. Elements get sorted in the same sequence in which they have been added to the Set.

Constructors:

1. LinkedHashSet() constructs a default hash set.
 2. LinkedHashSet(Collection c) initializes the hash set by using the elements of c.
 3. LinkedHashSet(int capacity) initializes the capacity of the hash set to capacity. The capacity grows automatically as elements are added to the Hash.
 4. LinkedHashSet(int capacity, float fillRatio) initializes both the capacity and the fill ratio (also called load capacity) of the hash set from its arguments.
-

Program 1.11: Program for LinkedHashSet.

```
import java.util.*;
public class DemoLinkedHashSet
{
    public static void main(String args[])
    {
        LinkedHashSet hs = new LinkedHashSet();           // create a hash set
        // add elements to the hash set
        hs.add("B");
        hs.add("A");
        hs.add("D");
        hs.add("E");
        hs.add("C");
        hs.add("F");
        System.out.println(hs);
    }
}
```

Output:

[B, A, D, E, C, F]

1.4 MAP INTERFACE

[April 17]

- 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 NoSuchElementException 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. **TreeMap:** It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.
 3. **LinkedHashMap:** It orders its elements based on the order in which they were inserted into the set (insertion-order).

Methods for Map:

| Method | Description |
|--|---|
| <code>void clear()</code> | Removes all key/value pairs from the invoking map. |
| <code>boolean containsKey(Object k)</code> | Returns true if the invoking map contains k as a key. Otherwise, returns false. |
| <code>boolean containsValue(Object v)</code> | Returns true if the map contains v as a value. Otherwise, returns false. |
| <code>Set entrySet()</code> | Returns a Set that contains the entries in the map. The set contains objects of type Map.Entry. This method provides a set-view of the invoking map. |
| <code>boolean equals(Object obj)</code> | Returns true if obj is a Map and contains the same entries. Otherwise, returns false. |
| <code>Object get(Object k)</code> | Returns the value associated with the key k. |
| <code>int hashCode()</code> | Returns the hash code for the invoking map. |
| <code>boolean isEmpty()</code> | Returns true if the invoking map is empty. Otherwise, returns false. |
| <code>Set keySet()</code> | Returns a Set that contains the keys in the invoking map. This method provides a set-view of the keys in the invoking map. |
| <code>Object put(Object k, Object v)</code> | Puts an entry in the invoking map, overwriting any previous value associated with the key. The key and value are k and v, respectively. Returns null if the key did not already exist. Otherwise, the previous value linked to the key is returned. |
| <code>void putAll(Map m)</code> | Puts all the entries from m into this map. |
| <code>Object remove(Object k)</code> | Removes the entry whose key equals k. |
| <code>int size()</code> | Returns the number of key/value pairs in the map. |
| <code>Collection values()</code> | Returns a collection containing the values in the map. This method provides a collection-view of the values in the map. |

Program 1.12: Program for Map interface.

```

import java.util.*;
public class DemoMap
{
    public static void main(String[] args)

```

```

{
    Map m1 = new HashMap();
    m1.put("Rudra", "8");
    m1.put("Mahi", "31");
    m1.put("Ayush", "12");
    m1.put("Dhruthik", "14");
    m1.put("Atharva", "14");
    System.out.println();
    System.out.println(" Map Elements");
    System.out.print("\t" + m1);
}
}

```

Output:

```

Map Elements
{Mahi=31, Dhruthik=14, Rudra=8, Ayush=12, Atharva=14}

```

1.4.1 HashMap

- HashMap contains values based on the key. This class implements the Map interface and extends AbstractMap class.
- It contains only unique elements. It may have one null key and multiple null values. It maintains no order.

Constructors:

1. `HashMap()` constructs a default hash map.
2. `HashMap(Map m)` initializes the hash map by using the elements of m.
3. `HashMap(int capacity)` initializes the capacity of the hash map to capacity.
4. `HashMap(int capacity, float fillRatio)` initializes both the capacity and fill ratio of the hash map by using its arguments.

Methods for HashMap:

| Method | Description |
|--|---|
| <code>void clear()</code> | Removes all mappings from the map. |
| <code>Object clone()</code> | Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned. |
| <code>boolean containsKey(Object key)</code> | Returns true if this map contains a mapping for the specified key. |
| <code>boolean containsValue(Object value)</code> | Returns true if this map maps one or more keys to the specified value. |

contd. ...

| | |
|---|---|
| <code>Set entrySet()</code> | Returns a collection view of the mappings contained in this map. |
| <code>Object get(Object key)</code> | Returns the value to which the specified key is mapped in this identity hash map, or null if the map contains no mapping for this key. |
| <code>boolean isEmpty()</code> | Returns true if this map contains no key-value mappings. |
| <code>Set keySet()</code> | Returns a set view of the keys contained in this map. |
| <code>Object put(Object key, Object value)</code> | Associates the specified value with the specified key in this map. |
| <code>putAll(Map m)</code> | Copies all of the mappings from the specified map to this map. These mappings will replace any mappings that this map had for any of the keys currently in the specified map. |
| <code>Object remove(Object key)</code> | Removes the mapping for this key from this map if present. |
| <code>int size()</code> | Returns the number of key-value mappings in this map. |
| <code>Collection values()</code> | Returns a collection view of the values contained in this map. |

Program 1.13: Program for HashMap.

```

import java.util.*;
import java.io.*;
public class DemoHashMap
{
    public static void main(String args[])throws IOException
    {
        HashMap hm = new HashMap();          // Create a HashMap
        // Put elements to the map
        hm.put("Ziva", new Double(3434.34));
        hm.put("Malhar", new Double(123.22));
        hm.put("Adhira", new Double(1378.00));
        hm.put("Kismat", new Double(99.22));
        hm.put("Akash", new Double(-19.08));
        // Get a set of the entries
    }
}

```

```

Set set = hm.entrySet();
// Get an iterator
Iterator i = set.iterator();
// Display elements
while(i.hasNext())
{
    Map.Entry me = (Map.Entry)i.next();
    System.out.print(me.getKey() + ": ");
    System.out.println(me.getValue());
}
System.out.println();
// Deposit 1000 into Zara's account
double balance = ((Double)hm.get("Ziva")).doubleValue();
hm.put("Ziva", new Double(balance + 1000));
System.out.println("Ziva's new balance: " + hm.get("Ziva"));
}
}

```

Output:

```

Ziva: 3434.34
Kismat: 99.22
Akash: -19.08
Adhira: 1378.0
Malhar: 123.22
Ziva's new balance: 4434.34

```

1.4.2 LinkedHashMap

- LinkedHashMap class extends HashMap and maintains a linked list of the entries in the map, in the order in which they were inserted.
- This allows insertion-order iteration over the map. That is, when iterating a LinkedHashMap, the elements will be returned in the order in which they were inserted.
- We can also create a LinkedHashMap that returns its elements in the order in which they were last accessed.

Constructors:

1. `LinkedHashMap()` constructs a default LinkedHashMap.
2. `LinkedHashMap(Map m)` initializes the LinkedHashMap with the elements from m.
3. `LinkedHashMap(int capacity)` initializes the capacity.

4. `LinkedHashMap(int capacity, float fillRatio)` initializes both capacity and fill ratio. The meaning of capacity and fill ratio are the same as for `HashMap`.
5. `LinkedHashMap(int capacity, float fillRatio, boolean Order)` allows us to specify whether the elements will be stored in the linked list by insertion order, or by order of last access. If Order is true, then access order is used. If Order is false, then insertion order is used.

Methods:

1. `void clear()` method removes all mappings from this map.
2. `boolean containsKey(Object key)` method returns true if this map maps one or more keys to the specified value.
3. `Object get(Object key)` method returns the value to which this map maps the specified key.
4. `protected boolean removeEldestEntry(Map.Entry eldest)` method returns true if this map should remove its eldest entry.

Program 1.14: Program for `LinkedHashMap`.

```
import java.util.*;
class DemoLinkedHashMap
{
    public static void main(String args[])
    {
        LinkedHashMap<Integer, String> lhmap = new LinkedHashMap<Integer,
                                                String>();
        lhmap.put(500,"Goa");
        lhmap.put(510,"Pune");
        lhmap.put(520,"Delhi");
        System.out.println("Keys: "+lhmap.keySet());      //Fetching key
        System.out.println("Values: "+lhmap.values());    //Fetching value
        System.out.println("Key-Value pairs: "+lhmap.entrySet()); //Fetching
                                                               key-value pair
    }
}
```

Output:

```
Keys: [500, 510, 520]
Values: [Goa, Pune, Delhi]
Key-Value pairs: [500=Goa, 510=Pune, 520=Delhi]
```

1.4.3 Hashtable

[Oct. 17, April 19]

- 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.
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.
4. Hashtable(Map m) creates a hash table that is initialized with the elements in m. The capacity of the hash table is set to twice the number of elements in m. The default load factor of 0.75 is used.

Methods for HashTable:

| Method | Description |
|-------------------------------------|---|
| void clear() | Resets and empties the hash table. |
| Object clone() | Returns a duplicate of the invoking object. |
| boolean contains(Object value) | Returns true if some value equal to value exists within the hash table. Returns false if the value isn't found. |
| boolean containsKey(Object key) | Returns true if some key equal to key exists within the hash table. Returns false if the key isn't found. |
| boolean containsValue(Object value) | Returns true if some value equal to value exists within the hash table. Returns false if the value isn't found. |
| Enumeration elements() | Returns an enumeration of the values contained in the hash table. |
| Object get(Object key) | Returns the object that contains the value associated with key. If key is not in the hash table, a null object is returned. |

contd. ...

| | |
|---|---|
| <code>boolean isEmpty()</code> | Returns true if the hash table is empty; returns false if it contains at least one key. |
| <code>Enumeration keys()</code> | Returns an enumeration of the keys contained in the hash table. |
| <code>Object put(Object key, Object value)</code> | Inserts a key and a value into the hash table. Returns null if key isn't already in the hash table; returns the previous value associated with key if key is already in the hash table. |
| <code>void rehash()</code> | Increases the size of the hash table and rehashes all of its keys. |
| <code>Object remove(Object key)</code> | Removes key and its value. Returns the value associated with key. If key is not in the hash table, a null object is returned. |
| <code>int size()</code> | Returns the number of entries in the hash table. |
| <code>String toString()</code> | Returns the string equivalent of a hash table. |

Program 1.15: Program for Hashtable.

```

import java.util.*;
class DemoHashtable
{
    public static void main(String args[])
    {
        Hashtable<Integer,String> mt=new Hashtable<Integer,String>();
        mt.put(1,"January");
        mt.put(2,"February");
        mt.put(4,"April");
        mt.put(5,"March");
        System.out.println("Initial Hashtable: "+mt);
        mt.put(3,"March"); //Inserts, as the pair
        System.out.println("Updated Hashtable: "+mt);
    }
}

```

Output:

Initial Hashtable: {5=March, 4=April, 2=February, 1=January}

Updated Hashtable: {5=March, 4=April, 3=March, 2=February, 1=January}

1.4.4 TreeMap

[Oct. 18]

- 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:

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

Methods for TreeMap:

| Method | Description |
|-------------------------------------|--|
| void clear() | Removes all mappings from this TreeMap. |
| Object clone() | Returns a shallow copy of this TreeMap instance. |
| Comparator comparator() | Returns the comparator used to order this map, or null if this map uses its keys' natural order. |
| boolean containsKey(Object key) | Returns true if this map contains a mapping for the specified key. |
| boolean containsValue(Object value) | Returns true if this map maps one or more keys to the specified value. |
| Set entrySet() | Returns a set view of the mappings contained in this map. |
| Object firstKey() | Returns the first (lowest) key currently in this sorted map. |
| Object get(Object key) | Returns the value to which this map maps the specified key. |

contd. ...

| | |
|---|--|
| <code>SortedMap headMap(Object toKey)</code> | Returns a view of the portion of this map whose keys are strictly less than toKey. |
| <code>Set keySet()</code> | Returns a Set view of the keys contained in this map. |
| <code>Object lastKey()</code> | Returns the last (highest) key currently in this sorted map. |
| <code>Object put(Object key, Object value)</code> | Associates the specified value with the specified key in this map. |
| <code>void putAll(Map map)</code> | Copies all of the mappings from the specified map to this map. |
| <code>Object remove(Object key)</code> | Removes the mapping for this key from this TreeMap if present. |
| <code>int size()</code> | Returns the number of key-value mappings in this map. |
| <code>SortedMap subMap(Object fromKey, Object toKey)</code> | Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| <code>SortedMap tailMap(Object fromKey)</code> | Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |
| <code>Collection values()</code> | Returns a collection view of the values contained in this map. |

Program 1.16: Program for TreeMap.

```

import java.util.*;
public class DemoTreeMap
{
    public static void main(String args[])
    {
        TreeMap<Integer, String> map=new TreeMap<Integer, String>();
        map.put(1,"Tiger");
        map.put(2,"Lion");
        map.put(5,"Monkey");
        map.put(3,"Elephant");
        System.out.println("Initial TreeMap");
        for(Map.Entry m:map.entrySet())
        {
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}

```

```

map.remove(5);
System.out.println("After remove() method");
for(Map.Entry m:map.entrySet())
{
    System.out.println(m.getKey()+" "+m.getValue());
}
}

```

Output:

```

Initial TreeMap
1 Tiger
2 Lion
3 Elephant
5 Monkey
After remove() method
1 Tiger
2 Lion
3 Elephant

```

1.5 COMPARATOR INTERFACE

- Comparator interface is used to order the objects of user-defined class.
- Both TreeSet and TreeMap store elements in sorted order. However, it is the comparator that defines precisely what sorted order means.
- The Comparator interface defines two methods i.e., compare() and equals().

1. The compare() Method:

- The compare() method, shown here, compares two elements for order.

```
int compare(Object obj1, Object obj2)
```

where, obj1 and obj2 are the objects to be compared. This method returns zero if the objects are equal. It returns a positive value if obj1 is greater than obj2. Otherwise, a negative value is returned.

- By overriding compare(), we can alter the way that objects are ordered. For example, to sort in reverse order, we can create a comparator that reverses the outcome of a comparison.

2. The equals() Method:

- The equals() method, shown here, tests whether an object equals the invoking comparator:

```
boolean equals(Object obj)
```

where, obj is the object to be tested for equality. The method returns true if obj and the invoking object are both Comparator objects and use the same ordering. Otherwise, it returns false.

- Overriding equals() is unnecessary, and most simple comparators will not do so.
-

Program 1.17: Program for comparator interface.

```
import java.util.*;
class Horse implements Comparator<Horse>, Comparable<Horse>
{
    private String name;
    private int age;
    Horse()
    {
    }
    Horse(String n, int a)
    {
        name = n;
        age = a;
    }
    public String getHorseName()
    {
        return name;
    }
    public int getHorseAge()
    {
        return age;
    }
    // Overriding the compareTo method
    public int compareTo(Horse h)
    {
        return (this.name).compareTo(h.name);
    }
    // Overriding the compare method to sort the age
    public int compare(Horse h, Horse h1)
    {
        return h.age - h1.age;
    }
}
```

```

public class DemoComparator
{
    public static void main(String args[])
    {
        // Takes a list o Horse objects
        List<Horse> list = new ArrayList<Horse>();
        list.add(new Horse("Shaggy",3));
        list.add(new Horse("Lacy",2));
        list.add(new Horse("Roger",10));
        list.add(new Horse("Tommy",4));
        list.add(new Horse("Tammy",1));
        Collections.sort(list);// Sorts the array list
        for(Horse a: list)//printing the sorted list of names
            System.out.print(a.getHorseName() + ", ");
        // Sorts the array list using comparator
        Collections.sort(list, new Horse());
        System.out.println(" ");
        for(Horse a: list)           //printing the sorted list of ages
            System.out.print(a.getHorseName() + ":" +
                a.getHorseAge() + ", ");
    }
}

```

Output:

Lacy, Roger, Shaggy, Tammy, Tommy,
 Tammy: 1, Lacy: 2, Shaggy: 3, Tommy: 4, Roger: 10,

1.5.1 Iterator**[April 16, 17, 18]**

- Iterator is used to iterate through elements of a collection class. Using Iterator we can traverse in one direction (forward).
- When, we will want to cycle through the elements in a collection. For example, we might want to display each element.
- The easiest way to do this is to employ an iterator, which is an object that implements either the Iterator or the ListIterator interface.
- Iterator enables us to cycle through a collection, obtaining or removing elements. ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.

- Before we can access a collection through an iterator, you must obtain one. Each of the collection classes provides an iterator() method that returns an iterator to the start of the collection.
- By using this iterator object, we can access each element in the collection, one element at a time.
- In general, to use an iterator to cycle through the contents of a collection, follow following steps:
 1. Obtain an iterator to the start of the collection by calling the collection's iterator() method.
 2. Set up a loop that makes a call to hasNext(). Have the loop iterate as long as hasNext() returns true.
 3. Within the loop, obtain each element by calling next().
- For collections that implement List, you can also obtain an iterator by calling ListIterator.

Methods:

1. boolean hasNext() method returns true if there are more elements. Otherwise, returns false.
 2. Object next() method returns the next element. Throws NoSuchElementException if there is not a next element.
 3. void remove() method removes the current element. Throws IllegalStateException if an attempt is made to call remove() that is not preceded by a call to next().
- Iterator is used for iterating (looping) various collection classes such as HashMap, ArrayList, LinkedList etc.

Program 1.18: Program for iterator.

```
import java.util.ArrayList;
import java.util.Iterator;
public class DemoIterator
{
    public static void main(String args[])
    {
        ArrayList names = new ArrayList();
        names.add("Chaitanya");
        names.add("Shiva");
        names.add("Jai");
        Iterator it = names.iterator();
```

```

        while(it.hasNext())
        {
            String obj = (String)it.next();
            System.out.println(obj);
        }
    }
}

```

Output:

Chaitanya
Shiva
Jai

1.5.2 ListIterator**[April 16, 17]**

- ListIterator used to iterate through elements of a collection class.
- Using ListIterator we can traverse the collection class on both the directions i.e., backward and forward.

Methods for ListIterator:

| Method | Description |
|-----------------------|---|
| void add(Object obj) | Inserts obj into the list in front of the element that will be returned by the next call to next(). |
| boolean hasNext() | Returns true if there is a next element. Otherwise, returns false. |
| boolean hasPrevious() | Returns true if there is a previous element. Otherwise, returns false. |
| Object next() | Returns the next element. A NoSuchElementException is thrown if there is not a next element. |
| int nextIndex() | Returns the index of the next element. If there is not a next element, returns the size of the list. |
| Object previous() | Returns the previous element. A NoSuchElementException is thrown if there is not a previous element. |
| int previousIndex() | Returns the index of the previous element. If there is not a previous element, returns -1. |
| void remove() | Removes the current element from the list. An IllegalStateException is thrown if remove() is called before next() or previous() is invoked. |
| void set(Object obj) | Assigns obj to the current element. This is the element last returned by a call to either next() or previous(). |

Program 1.19: Program of ListIterator.

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
public class DemoListIterator
{
    public static void main(String a[])
    {
        ListIterator<String> litr = null;
        List<String> names = new ArrayList<String>();
        names.add("Shrimant");
        names.add("Ruha");
        names.add("Prapti");
        names.add("Tanaji");
        names.add("Kavya");
        //Obtaining list iterator
        litr=names.listIterator();
        System.out.println("Traversing the list in forward direction:");
        while(litr.hasNext())
        {
            System.out.println(litr.next());
        }
        System.out.println("\nTraversing the list in backward direction:");
        while(litr.hasPrevious())
        {
            System.out.println(litr.previous());
        }
    }
}
```

Output:

```
Traversing the list in forward direction:
Shrimant
Ruha
Prapti
Tanaji
Kavya
```

Traversing the list in backward direction:

Kavya
Tanaji
Prapti
Ruha
Shrimant

1.5.3 Enumeration Interface

- The Enumeration interface defines the methods by which you can enumerate, (obtain one at a time) the elements in a collection of objects.
- This legacy interface has been superceded by Iterator. Although not deprecated, Enumeration is considered obsolete for new code.
- However, it is used by several methods defined by the legacy classes such as Vector and Properties, is used by several other API classes, and is currently in widespread use in application code.

Methods:

1. `boolean hasMoreElements()`: When implemented, it must return true while there are still more elements to extract, and false when all the elements have been enumerated.
 2. `Object nextElement()`: This returns the next object in the enumeration as a generic Object reference.
-

Program 1.20: Program for Enumeration Interface.

```
import java.util.Vector;
import java.util.Enumeration;
public class DemoEnumeration
{
    public static void main(String args[])
    {
        Enumeration days;
        Vector dayNames = new Vector();
        dayNames.add("Sunday");
        dayNames.add("Monday");
        dayNames.add("Tuesday");
        dayNames.add("Wednesday");
        dayNames.add("Thursday");
        dayNames.add("Friday");
        dayNames.add("Saturday");
        days = dayNames.elements();
```

```
        while(days.hasMoreElements())
        {
            System.out.println(days.nextElement());
        }
    }
```

Output:

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

ADDITIONAL PROGRAMS

Program 1: Program for ArrayList and methods interface.

```
import java.util.*;
class ArrayListDemo
{
    public static void main (String args[])
    {
        ArrayList a1 = new ArrayList();
        System.out.println("Initial size of ArrayList " + a1.size());
        a1.add("First");
        a1.add("Second");
        a1.add("Third");
        a1.add("Fourth");
        a1.add("Fifth");
        a1.add(2, "Middle");
        System.out.println("Size of ArrayList after addition is " +
                           a1.size());
        System.out.println("Elements in ArrayList are " + a1);
        a1.remove("Third");
        a1.remove(1);
        System.out.println("Size of ArrayList after deletion is " +
                           a1.size());
        System.out.println("Elements in ArrayList are " + a1);
    }
}
```

Output:

```
Initial size of ArrayList 0
Size of ArrayList after addition is 6
Elements in ArrayList are [First, Second, Middle, Third, Fourth, Fifth]
Size of ArrayList after deletion is 4
Elements in ArrayList are [First, Middle, Fourth, Fifth]
```

Program 2: Program to display the sum of ArrayList elements.

```
import java.util.*;
import java.io.*;
class ArrayListToArray
{
    public static void main (String args[])throws IOException
    {
        ArrayList <Integer>a = new ArrayList <Integer>();
        a.add(11);
        a.add(22);
        a.add(33);
        a.add(44);
        a.add(55);
        System.out.println("Contents of ArrayList are"+ a);
        int[] ia = new int[a.size()];
        int sum = 0;
        for(int i = 0; i < a.size(); i++)
        {
            ia[i] = a.get(i).intValue();
            sum+= ia[i];
        }
        System.out.println("Sum of the elements is " + sum);
    }
}
```

Output:

```
Contents of ArrayList are[11, 22, 33, 44, 55]
Sum of the elements is 165
```

Program 3: Program to read ‘n’ cities names from user store them into ArrayList, the program should not allow duplicate city and display it in ascending order.

```
import java.util.*;
import java.io.*;
public class DemoCity
{
    public static void main(String args[])throws Exception
    {
        BufferedReader br = new BufferedReader(new
                                            InputStreamReader(System.in));
        ArrayList a = new ArrayList();
        System.out.println("\nHow many city? ");
        int n = Integer.parseInt(br.readLine());
        System.out.println("\nEnter City Name: ");
        for(int i = 1; i <= n; i++)
        {
            a.add(br.readLine());
        }
        TreeSet tr = new TreeSet(a);
        System.out.println("Sorted city names are : "+tr);
    }
}
```

Output:

```
How many city?
5
Enter City Name:
Kerla
Goa
Shimla
Mumbai
Kerla
Sorted city names are : [Goa, Kerla, Mumbai, Shimla]
```

Program 4: Program to accept ‘n’ Employee from user, store them into the linkedlist class and display them by using ListIterator interface and Iterator interface.

```
import java.util.*;
import java.io.*;
public class EmpLinkList
{
    public static void main(String args[])throws Exception
    {
        int n;
        BufferedReader br = new BufferedReader(new
                                                InputStreamReader(System.in));
        LinkedList li = new LinkedList ();
        System.out.println("\nEnter number of Employee : ");
        n = Integer.parseInt(br.readLine());
        System.out.println("\nEnter name : ");
        for(int i = 1; i <= n; i++)
        {
            li.add(br.readLine());
        }
        System.out.println("\nLink List Content : ");
        Iterator it = li.iterator();
        while(it.hasNext())
        {
            System.out.println(it.next());
        }
        System.out.println("\nReverse order : ");
        ListIterator lt = li.listIterator();
        while(lt.hasNext())
        {
            lt.next();
        }
        while(lt.hasPrevious())
        {
            System.out.println(lt.previous());
        }
    }
}
```

Output:

```
Enter number of Employee :
```

```
5
```

```
Enter name :
```

```
Rudra
```

```
Niket
```

```
Maya
```

```
Harsh
```

```
Lavanya
```

```
Link List Content :
```

```
Rudra
```

```
Niket
```

```
Maya
```

```
Harsh
```

```
Lavanya
```

```
Reverse order :
```

```
Lavanya
```

```
Harsh
```

```
Maya
```

```
Niket
```

```
Rudra
```

Program 5: Program for collection class ArrayList.

```
import java.util.*;
public class CollectionsDemo
{
    public static void main(String[] args)
    {
        List a1 = new ArrayList();
        a1.add("Zuber");
        a1.add("Mahesh");
        a1.add("Ayush");
        System.out.println(" ArrayList Elements:");
        System.out.print("\t" + a1);
        List l1 = new LinkedList();
        l1.add("Zuber");
        l1.add("Mahesh");
```

```

        l1.add("Ayush");
        System.out.println();
        System.out.println(" LinkedList Elements:");
        System.out.print("\t" + l1);
        Set s1 = new HashSet();
        s1.add("Zuber");
        s1.add("Mahesh");
        s1.add("Ayush");
        System.out.println();
        System.out.println(" Set Elements:");
        System.out.print("\t" + s1);
        Map m1 = new HashMap();
        m1.put("Zuber", "8");
        m1.put("Mahesh", "31");
        m1.put("Ayush", "12");
        m1.put("Arnav", "14");
        System.out.println();
        System.out.println(" Map Elements:");
        System.out.print("\t" + m1);
    }
}

```

Output:

```

ArrayList Elements:
[Zuber, Mahesh, Ayush]
LinkedList Elements:
[Zuber, Mahesh, Ayush]
Set Elements:
[Mahesh, Zuber, Ayush]
Map Elements:
{Arnav=14, Mahesh=31, Zuber=8, Ayush=12}

```

Program 6: Program for LinkedHashMap.

```

import java.util.*;
public class LinkedHashMapDemo
{
    public static void main(String args[])

```

```
{  
    LinkedHashMap lhm = new LinkedHashMap(); // Create a LinkedHashMap  
    // Put elements to the map  
    lhm.put("Roja", new Double(4534.96));  
    lhm.put("Meet", new Double(321.22));  
    lhm.put("Guru", new Double(3421.00));  
    lhm.put("Dia", new Double(77.88));  
    lhm.put("Pia", new Double(-89.18));  
    // Get a set of the entries  
    Set set = lhm.entrySet();  
    // Get an iterator  
    Iterator i = set.iterator();  
    // Display elements  
    while(i.hasNext())  
    {  
        Map.Entry me = (Map.Entry)i.next();  
        System.out.print(me.getKey() + ": ");  
        System.out.println(me.getValue());  
    }  
    System.out.println();  
    // Deposit 1000 into Zara's account  
    double balance = ((Double)lhm.get("Meet")).doubleValue();  
    lhm.put("Meet", new Double(balance + 1000));  
    System.out.println("Meet's new balance: " + lhm.get("Meet"));  
}  
}
```

Output:

```
Roja: 4534.96  
Meet: 321.22  
Guru: 3421.0  
Dia: 77.88  
Pia: -89.18  
Meet's new balance: 1321.22
```

Program 7: Program to create a Hashtable containing student name and percentage. Display the details of hash table. Also search for the specific student and display percentage of that student.

```
import java.io.*;
import java.util.*;
class DemoStudent
{
    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=0;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));
            String m = br1.readLine();
        }
    }
}
```

```
if(ht.containsKey(m))
{
    System.out.println("Record found and percentage is:");
    System.out.println(ht.get(m));
}
else
{
    System.out.println("no such student found");
}
}
```

Output:

```
Enter no of Students :
5
Enter Student name :Adhya
Enter Student's per :79
Enter Student name :Arjit
Enter Student's per :60
Enter Student name :Kavi
Enter Student's per :87
Enter Student name :Atharva
Enter Student's per :95
Enter Student name :Veer
Enter Student's per :77
Student Name
    Kavi
    Arjit
    Adhya
    Veer
    Atharva
Student Percentage
    87.0
    60.0
    79.0
    77.0
    95.0
Enter the student name to be searched:
    Adhya
Record found and percentage is:
    79.0
```

Program 8: Write a program that uses the Hashtable class for storing and retrieving employee records.

```
import java.util.*;
class HTDemo1
{
    public static void main(String args[])
    {
        Hashtable employee = new Hashtable();
        Enumeration names;
        String str;
        employee.put("Amar", "Manager");
        employee.put("Deep", "Supervisor");
        employee.put("Sham", "Head");
        employee.put("Deepa", "Cleaner");
        employee.put("Khushi", "Cleaner");
        names = employee.keys();
        while(names.hasMoreElements())
        {
            str = (String) names.nextElement();
            System.out.println(str + ": " +employee.get(str));
        }
    }
}
```

Output:

```
Sham: Head
Deepa: Cleaner
Khushi: Cleaner
Deep: Supervisor
Amar: Manage
```

Program 9: Write a program to read 10 strings from console and add them in a collection. The strings should be stored in the collection in the descending order. Display the elements of the collection using iterator.

```
import java.util.*;
import java.util.ArrayList;
import java.util.Iterator;
```

```
public class Collection1
{
    public static void main(String[] args)
    {
        ArrayList<String> lst = new ArrayList<String>();
        for(int i=0;i<10;i++)
        {
            lst.add(args[i]);
        }
        System.out.println(lst);
        Iterator iter = lst.iterator();
        while (iter.hasNext())
        {
            String str = (String)iter.next();
            System.out.println(str);
        }
        Collections.sort(lst, Collections.reverseOrder());
        System.out.println("After Sorting:");
        for(String counter: lst)
        {
            System.out.println(counter);
        }
    }
}
```

Output:

```
[ww, xx, yy, qq, aa, bb, cc, dd, jj, mm]
ww
xx
yy
qq
aa
bb
cc
dd
jj
mm
```

After Sorting:

yy
xx
ww
qq
mm
jj
dd
cc
bb
aa

Program 10: Program to read n strings into Array list Collection. Display the elements of collection in reverse order.

```
import java.util.*;  
import java.io.*;  
class ArrayListEx  
{  
    public static void main(String ar[])throws IOException  
        int n;  
        BufferedReader br = new BufferedReader(new  
                                         InputStreamReader(System.in));  

```

```
lit.next();
System.out.println("The contents of arraylist in reverse order...");
Collections.sort(a, Collections.reverseOrder());
System.out.println("After Sorting:");
for(String counter: a)
{
    System.out.println(counter);}
}
```

Output:

```
Enter size of arraylist
5
Enter element
august
Enter element
march
Enter element
june
Enter element
july
Enter element
september
The contents of arraylist using iterator..
august
march
june
july
september
The contents of arraylist in reverse order...
After Sorting:
september
march
june
july
august
```

Program 11: Program to accept ‘n’ movie names from the user and sort the list in descending order.

```
import java.util.*;
import java.io.*;
class Moviename
{
    public static void main(String ar[])throws IOException
    {
        int n;
        BufferedReader br = new BufferedReader(new
                                                InputStreamReader(System.in));
        System.out.println("\t How many movie names you want ");
        n=Integer.parseInt(br.readLine());
        ArrayList <String> a = new ArrayList<String>(n);
        for(int i=0;i<n;i++)
        {
            System.out.println("Entre Movie names: ");
            a.add(br.readLine());
        }
        System.out.println(" The movie names in list using iterator..");
        Iterator <String> it = a.iterator();
        while(it.hasNext())
            System.out.println(" \t " + it.next());
        ListIterator <String> lit = a.listIterator();
        while(lit.hasNext())
            lit.next();
        System.out.println("The movie names of arraylist in reverse order... ");
        Collections.sort(a, Collections.reverseOrder());
        System.out.println("After Sorting:");
        for(String counter: a){
            System.out.println(counter);}
    }
}
```

Output:

```
How many movie names you want
7
Entre Movie names:
DDLJ
Entre Movie names:
DIL
Entre Movie names:
Bahubali
Entre Movie names:
Dabang
Entre Movie names:
Lunchbox
Entre Movie names:
Dhoom
Entre Movie names:
Raazi
The movie names in list using iterator..
    DDLJ
    DIL
    Bahubali
    Dabang
    Lunchbox
    Dhoom
    Raazi
The movie names of arraylist in reverse order...
After Sorting:
Raazi
Lunchbox
Dhoom
Dabang
DIL
DDLJ
Bahubali
```

Program 12: Java program to read n strings into ArrayList collection and sort the elements of collection in descending order (use comparator).

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.io.*;
public class MyArrayListSort
{
    public static void main(String a[]) throws IOException
    {
        List<Empl> list = new ArrayList<Empl>();
        list.add(new Empl("Ram",3000));
        list.add(new Empl("Jai",6000));
        list.add(new Empl("Krish",2000));
        list.add(new Empl("Tina",2400));
        Collections.sort(list,new MySalaryComp());
        System.out.println("Sorted list entries: ");
        for(Empl e:list)
        {
            System.out.println(e);
        }
    }
    class MySalaryComp implements Comparator<Empl>
    {
        public int compare(Empl e1, Empl e2)
        {
            if(e1.getSalary() < e2.getSalary())
            {
                return 1;
            }
            else
            {
                return -1;
            }
        }
    }
}
```

```
class Empl
{
    private String name;
    private int salary;
    public Empl(String n, int s)
    {
        this.name = n;
        this.salary = s;
    }
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    public int getSalary()
    {
        return salary;
    }
    public void setSalary(int salary)
    {
        this.salary = salary;
    }
    public String toString()
    {
        return "Name: "+this.name+"-- Salary: "+this.salary;
    }
}
```

Output:

```
Sorted list entries:
Name: Jai-- Salary: 6000
Name: Ram-- Salary: 3000
Name: Tina-- Salary: 2400
Name: Krish-- Salary: 2000
```

Program 13: Create the Hashtable that will maintain the mobile number and student name. Display the contact list.

```
import java.util.*;
import java.io.*;
class HTDemo
{
    public static void main(String args[])
    {
        Hashtable student = new Hashtable();
        Enumeration names;
        String str;
        student.put("Amar",new Long(8999999999L));
        student.put("Anil",new Long(9751555327L));
        student.put("Asif",new Long(9751555327L));
        student.put("Aman",new Long(9751555327L));
        student.put("Atul",new Long(9751555327L));
        student.put("Ajay",new Long(9751555327L));
        names = student.keys();
        while(names.hasMoreElements())
        {
            str = (String) names.nextElement();
            System.out.println(str + ": " +student.get(str));
        }
    }
}
```

Output:

```
Aman: 9751555327
Asif: 9751555327
Ajay: 9751555327
Atul: 9751555327
Anil: 9751555327
Amar: 8999999999
```

Program 14: Java program to accept the details of ‘n’ employees (EName, Salary) from the user, store them into the Hashtable and displays the Employee Names having maximum Salary.

```
import java.util.*;
import java.io.*;
public class DemoEmpHT
{
    public static void main(String args[])throws Exception
    {
        int n,esal=0;
        String ename="";
        BufferedReader br = new BufferedReader(new
                                                InputStreamReader(System.in));
        Hashtable ht = new Hashtable();
        System.out.println("\nEnter number of Employee : ");
        n = Integer.parseInt(br.readLine());
        System.out.println("\nEnter Employee Name and Salary : ");
        for(int i = 1; i <= n; i++)
        {
            ename = br.readLine();
            esal=Integer.parseInt(br.readLine());
            ht.put(ename,esal);
        }
        Enumeration v = ht.elements(); //employee salary
        Enumeration k = ht.keys(); //employee name
        while(k.hasMoreElements())
        {
            System.out.println(k.nextElement()+" "+v.nextElement());
        }
        int emax = 0;
        String empstr="";
        k = ht.keys();
        v = ht.elements();
        while(v.hasMoreElements())
        {
            esal=(Integer)v.nextElement();
            ename = (String)k.nextElement();
            if(esal>emax)
                emax=esal;
        }
        System.out.println("Employee with Maximum Salary is "+ename);
    }
}
```

```
        if(esal>emax)
        {
            emax = esal;
            empstr = ename;
        }
    }
    System.out.println(empstr +" has maximum salary is "+emax);
}
}
```

Output:

```
Enter number of Employee :
5
Enter Employee Name and Salary :
AAA
55000
BBB
79000
CCC
30000
DDD
22000
ZZZ
45000
AAA 55000
ZZZ 45000
DDD 22000
CCC 30000
BBB 79000
BBB has maximum salary is 79000
```

Program 15: Program to sorts the TreeSet elements and display it descending order.

```
import java.util.Comparator;
import java.util.SortedSet;
import java.util.TreeSet;
```

```

public class ExTreeSetDesOrder
{
    public static void main(String[] args)
    {
        SortedSet<String> fruits = new TreeSet<>(new Comparator<String>()
        {
            public int compare(String s1, String s2)
            {
                return s2.compareTo(s1);
            }
        });
        // Adding new elements to a TreeSet
        fruits.add("Banana");
        fruits.add("Jackfruit");
        fruits.add("Apple");
        fruits.add("Mango");
        fruits.add("Pineapple");
        fruits.add("Orange");
        fruits.add("Berries");
        System.out.println("Fruits Set : " + fruits);
    }
}

```

Output:

Fruits Set : [Pineapple, Orange, Mango, Jackfruit, Berries, Banana, Apple]

Program 16: Program to display the stack (PUSH and POP) elements.

```

import java.util.*;
import java.io.*;
class StackDemo
{
    public static void main (String args[])
    {
        Stack <Integer>st = new Stack <Integer>();
        st.push(new Integer(20));
        st.push(new Integer(33));
        st.push(new Integer(62));
        st.push(new Integer(88));
    }
}

```

```

        st.push(new Integer(44));
        System.out.println("Contents of stack are : " + st);
        System.out.println("Popped element : " + st.pop());
        System.out.println("Contents of stack are : " + st);
        System.out.println("Popped element : " + st.pop());
        try
        {
            System.out.println("Popped element : " + st.pop());
        }
        catch (EmptyStackException e)
        {
            System.out.println("Empty Stack");
        }
    }
}

```

Output:

```

Contents of stack are : [20, 33, 62, 88, 44]
Popped element : 44
Contents of stack are : [20, 33, 62, 88]
Popped element : 88
Popped element : 62
Popped element : 33
Popped element : 20

```

Program 17: Program to explain the methods of vector.

```

import java.util.*;
class VectorDemo
{
    public static void main (String args[])
    {
        Vector v = new Vector(3, 3);
        System.out.println("Initial size :" + v.size());
        System.out.println("Initial capacity : " + v.capacity());
        System.out.println();
        v.addElement(new Integer(700));
    }
}

```

```
v.addElement(new Integer(59))
v.addElement(new Integer(61));
v.addElement(new Integer(88));
v.addElement(new Float(562.617));
System.out.println("Size after insertion : " + v.size());
System.out.println("Capacity after insertion : " + v.capacity());
System.out.println();
v.addElement(new Float(56.87));
v.addElement(new Integer(143));
System.out.println("Capacity after insertion : " + v.capacity());
System.out.println();
System.out.println("First element : " + (Integer) v.firstElement());
System.out.println("Last element : " + (Integer) v.lastElement());
System.out.println();
if (v.contains(new Integer(92)))
System.out.println("Vector contains 92");
System.out.println();
Enumeration en = v.elements();
System.out.println("Elements in Vector are");
while (en.hasMoreElements())
System.out.print(en.nextElement() + " ");
System.out.println();
}
}
```

Output:

```
Initial size :0
Initial capacity : 3
Size after insertion : 5
Capacity after insertion : 6
Capacity after insertion : 9
First element : 700
Last element : 143
Elements in Vector are
700 59 61 88 562.617 56.87 143
```

Program 18: Program for ArrayList and Iterator interface.

```
import java.util.*;
class IteratorDemo
{
    public static void main (String args[])
    {
        ArrayList a1 = new ArrayList();
        a1.add("First");
        a1.add("Second");
        a1.add("Third");
        a1.add("Fourth");
        a1.add("Fifth");
        System.out.println("Elements in ArrayList are ");
        Iterator <String>itr = a1.iterator();
        while (itr.hasNext())
        {
            Object ele = itr.next();
            System.out.print(ele + ", ");
        }
        System.out.println();
        System.out.println();
        ListIterator <String>litr = a1.listIterator();
        while (litr.hasNext())
        {
            Object ele = litr.next();
            litr.set(ele + " Element");
        }
        System.out.println("Modified Elements in ArrayList are ");
        Iterator <String>itr1 = a1.iterator();
        while (itr1.hasNext())
        {
            Object ele = itr1.next();
            System.out.print(ele + ", ");
        }
        System.out.println();
        System.out.println();
        System.out.println("Elements in ArrayList in backward order are ");
```

```

        while (litr.hasPrevious())
        {
            Object ele = litr.previous();
            System.out.print(ele + ", ");
        }
        System.out.println();
    }
}

```

Output:

```

Elements in ArrayList are
First, Second, Third, Fourth, Fifth,
Modified Elements in ArrayList are
First Element, Second Element, Third Element, Fourth Element, Fifth Element
Elements in ArrayList in backward order are
Fifth Element, Fourth Element, Third Element, Second Element, First Element

```

Program 19: Program of TreeMap.

```

import java.util.*;
class TreeMapDemo
{
    public static void main (String args[])
    {
        TreeMap t= new TreeMap();
        t.put("Asmita", new Integer(56));
        t.put("Rakesh", new Integer(75));
        t.put("Manisha", new Integer(80));
        t.put("Yash", new Integer(45));
        t.put("Jaya", new Integer(63));
        t.put("Hemant", new Integer(72));
        Set s = t.entrySet();
        Iterator i = s.iterator();
        while (i.hasNext())
        {
            Map.Entry m = (Map.Entry) i.next();
            System.out.print(m.getKey() + " ");
            System.out.println(m.getValue());
        }
    }
}

```

```

        int marks = ((Integer)t.get("Jaya")).intValue();
        t.put("Jaya", (new Integer(marks + 10)));
        System.out.println("Modifiend Marks of Jaya are " + t.get("Jaya"));
    }
}

```

Output:

```

Asmita 56
Hemant 72
Jaya 63
Manisha 80
Rakesh 75
Yash 45
Modifiend Marks of Jaya are 73

```

Program 20: Program for Hashtable.

```

import java.util.*;
public class HashtableDemo
{
    public static void main(String args[])
    {
        Hashtable<Integer,String> hm=new Hashtable<Integer,String>();
        hm.put(100,"Amit");
        hm.put(102,"Ravi");
        hm.put(101,"Vijay");
        hm.put(103,"Rahul");
        for(Map.Entry m:hm.entrySet())
        {
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}

```

Output:

```

103 Rahul
102 Ravi
101 Vijay
100 Amit

```

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which in Java refer to a collection of individual objects that are represented as a single unit?
(a) Collections (b) Interfaces
(c) Classes (d) Methods
2. Java collection framework provides an architecture to store and manipulate a group of objects which includes,
(a) Interfaces in Java allow Collections to be manipulated independently from the details of their representation.
(b) Classes in Java are the implementation of the collection interface.
(c) Algorithm refers to the methods which are used to perform operations such as searching and sorting, on objects that implement collection interfaces.
(d) All of the mentioned
3. Which interface is a root of the collection hierarchy?
(a) Map (b) Collection
(c) List (d) None of the mentioned
4. What is Collection?
(a) A group of component (b) A group of classes
(c) A group of objects (d) A group of operator
5. Which of these classes is not part of Java's collection framework?
(a) Map (b) Array
(c) Stack (d) Queue
6. Which of these packages contain all the collection classes?
(a) java.lang (b) java.util
(c) java.net (d) java.awt
7. Which of these iterators can be used only with List?
(a) SetIterator (b) ArrayIterator
(c) MapIterator (d) ListIterator
8. Which of this interface must contain a unique element?
(a) Set (b) List
(c) Array (d) Collection
9. Which of this method is used to make all elements of an equal to specified value?
(a) add() (b) fill()
(c) all() (d) set()
10. Which of these exceptions is thrown by remover() method?
(a) IOException (b) SystemException
(c) ObjectNotFoundException (d) IllegalStateException
11. Which of these return type of hasNext() method of an iterator?
(a) Collections Object (b) Boolean
(c) Double (d) Integer

12. Which of these methods can be used to move to next element?
 (a) next()
 (c) shuffle()
 (b) move()
 (d) hasNext()
13. Which of this interface is not a part of collection framework?
 (a) List
 (c) SortedMap
 (b) Set
 (d) SortedList
14. In Java which class creates a collection that uses a hash table for storage.
 (a) HashSet
 (b) HashMap
 (c) HashTree
 (d) None of the mentioned

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|--------|--------|--------|--------|--------|---------|
| 1. (a) | 2. (d) | 3. (b) | 4. (c) | 5. (a) | 6. (b) | 7. (d) | 8. (a) | 9. (b) | 10. (d) |
| 11. (b) | 12. (a) | 13. (d) | 14. (a) | | | | | | |

Q. II Fill in the Blanks:

1. The collection is a Framework of _____.
2. _____ interface doesn't extend collection interface.
3. The _____ interface is the root interface for all the collection classes.
4. Queue maintains element in _____.
5. _____ is the implementation of map and sortedmap.
6. _____ class can generate an array which can increase and decrease in size automatically.
7. _____ methods can randomize all elements in a list.
8. ArrayList, LinkedList and Vector are all _____.
9. _____ can traverse in both forward and backward direction on lists.
10. List, Set and Queue _____ Collection.
11. A _____ is an array of list.
12. _____ interface is used to order the objects of user-defined class.
13. The _____ interface defines the methods by which we can enumerate the elements in a collection of objects.
14. _____ is simply an object that groups multiple elements into a single unit.

Answers

| | | | |
|---------------------------|----------------|---------------|----------------|
| 1. interfaces and classes | 2. Map | 3. Iterable | 4. FIFO |
| 5. TreeMap | 6. ArrayList | 7. shuffle | 8. classes |
| 9. ListIterator | 10. extebds | 11. Hashtable | 12. Comparator |
| 13. Enumeration | 14. Collection | | |

Q. III State True or False:

1. Collection is a class.
2. LinkedHashSet class is non-synchronized.
3. Map doesn't allow duplicate keys.
4. A collection is an interface.

5. HashSet class contains unique elements.
6. Map allows duplicate values.
7. HashSet class does not maintain insertion order.
8. HashSet class is synchronized.
9. LinkedList is suited for manipulating data.
10. ArrayList is suited for storing and accessing data.
11. A collection is an object that can hold references to other objects.
12. Classes allow collections to be manipulated independently of the details of their representation.
13. Iterator interface provides the facility of iterating the elements in a forward direction only.
14. A Set is a Collection that can contain duplicate elements.

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|--------|--------|--------|--------|--------|---------|
| 1. (F) | 2. (T) | 3. (T) | 4. (F) | 5. (T) | 6. (T) | 7. (F) | 8. (F) | 9. (T) | 10. (T) |
| 11. (T) | 12. (F) | 13. (T) | 14. (F) | | | | | | |

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define Collection.
2. What is Collection Framework?
3. Define List.
4. Define Set.
5. What is the use of Map interface?
6. List components for Collection framework.
7. Give purpose of ArrayList.
8. Define Iterator.
9. What is Enumeration interface?
10. Define comparator.
11. What is the use of LinkedList class?
12. Compare Map and Set interface with any two points.
13. What TreeSet?
14. List any methods for Collection interface.

(B) Long Answer Questions:

1. What is Collections Framework? Explain with diagram. Also state its advantages.
2. What are the core collection interfaces? Explain any two of them in detail.
3. What is the use of LinkedList class? Explain with example.
4. What is HashMap? Describe with example.
5. Write a java program to read n strings and store into linkedList, also display contents of same LinkedList.

6. Write a java program which reads in a series of first names and stores them in a LinkedList. The program should not allow to store duplicate names and it should allow the user to search for a first name.
7. Explain TreeSet in detail.
8. What is Iterator interface? Describe with example.
9. What is Comparator Interface? Explain in detail example.
10. With the help of Enumeration interface.
11. Differentiate between Comparator Interface and Enumeration interface.
12. "ArrayList is a list implemented using a resizable array". Comment.
13. Write a program to accept employee information as (id, name). Store it into Hashtable. Display only names of all employees.
14. Write a java program to read n strings into ArrayList collection. Display the elements of collection in reverse order.
15. Write a Java program to read n strings into ArrayList collection and sort the elements of collection in descending order (use comparator).
16. Create the Hashtable that will maintain the mobile number and student name. Display the contact list.
17. Write a program that uses the Hashtable class for storing and retrieving employee records.
18. Write a program to read 10 strings from console and add them in a collection. The strings should be stored in the collection in the descending order. Display the elements of the collection using iterator.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. What is the difference between an Iterator and List Iterator? [1 M]
- Ans.** Refer to Sections 1.5.1 and 1.5.2.
2. Write a Java program to accept 'n' employee information (id, name) and store into Hashtable. Display all employee details. [4 M]
- Ans.** Refer to Additional Programs.
3. State four constructors of HashSet class. [2 M]
- Ans.** Refer to Section 1.3.1.

April 2017

1. What is collection? [1 M]
- Ans.** Refer to Section 1.1.
2. Which interfaces are implemented by TreeSet class? [1 M]
- Ans.** Refer to Section 1.3.2.
3. Explain map interface. Give its implementation. [4 M]
- Ans.** Refer to Section 1.4.
4. State the use of Iterator and List Iterator. [2 M]
- Ans.** Refer to Sections 1.5.1 and 1.5.2.

October 2017

1. State constructors of TreeSet class.

[1 M]

Ans. Refer to Section 1.3.2.

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

[5 M]

Ans. Refer to Additional Programs.

April 2018

1. List any four collection interfaces.

[1 M]

Ans. Refer to Section 1.1.2.

2. State the use of iterator with its syntax.

[1 M]

Ans. Refer to Section 1.5.1.

3. Write 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.

[4 M]

Ans. Refer to Additional Programs.

4. Write any two differences between ArrayList and LinkedList.

[2 M]

Ans. Refer to Sections 1.2.1 and 1.2.2.

October 2018

1. What is collection framework?

[1 M]

Ans. Refer to Section 1.1.

2. Write a note on TreeMap class.

[2 M]

Ans. Refer to Section 1.4.4.

3. Write a code using Hashtable, that will maintain the names and phone nos of two students.

[2 M]

Ans. Refer to Section 1.4.3.

April 2019

1. Which method of Hashtable returns a duplicate of the invoking object?

[1 M]

Ans. Refer to Section 1.4.3.

2. Explain the use of collection Interface?

[1 M]

Ans. Refer to Section 1.1.2.

3. 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.

[5 M]

Ans. Refer to Additional Programs.

■ ■ ■

Multithreading

Objectives...

- To understand Concepts in Thread
- To learn Life Cycle of Thread and Implementation of Thread
- To study Thread Priorities
- To understand Synchronization and Inter-thread Communication Concepts

2.0 INTRODUCTION

[Oct. 17, April 18]

- Multithreading is one of the most important feature of Java. Multithreading in Java is a process of executing multiple threads simultaneously.
- Java provides built-in support for multithreaded programming. Java is a multi threaded programming language which means we can develop multi threaded program using Java.
- A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources especially when the computer has multiple CPUs.
- By definition multitasking is when multiple processes share common processing resources such as a CPU.
- Multi threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel.
- The OS (Operating System) divides processing time not only among different applications, but also among each thread within an application.
- Multi threading enables us to write in a way where multiple activities can proceed concurrently in the same program.

2.1 WHAT ARE THREADS?

[Oct. 17, 18]

- A task is completed by a program with the sequence of steps called process. Each specific task in a process is called thread.
- When there are multiple threads execute simultaneously, it is called multithreading.
- 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.

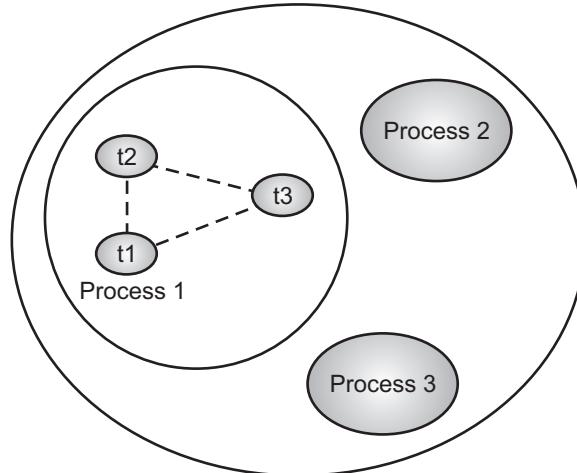


Fig. 2.1: Thread Process

- As shown in the Fig. 2.1, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.
- Conceptually, 'a thread is a basic unit of CPU utilization'. A thread is similar to the sequential programs.
- A single thread has a beginning, a sequence and an end. A thread comprises a thread ID, a program counter, a register set and a stack.
- A thread shares the code section, data section and other operating system resources [like open files] with other threads belonging to the same process.
- Multithreaded application has many threads executing concurrently. This is the facility given by Java virtual machine.
- Fig. 2.2 (a) illustrates single threaded process and Fig. 2.2 (b) illustrates multithreaded process.

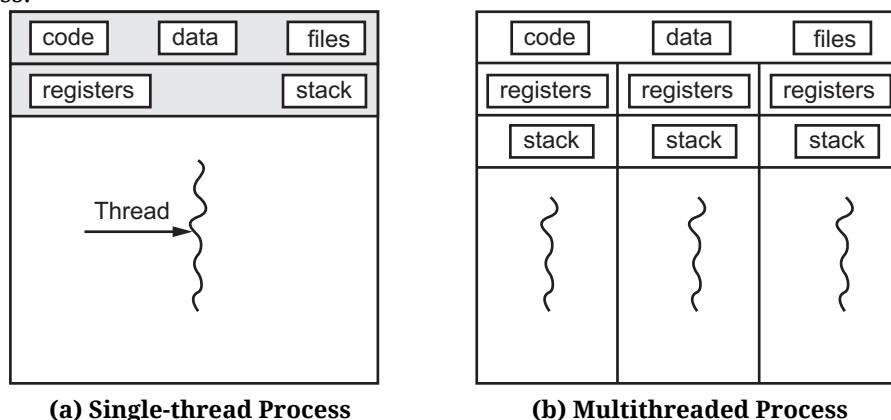


Fig. 2.2

Difference between Multitasking and Multithreading:**[Oct. 17, 18]**

- Multitasking let CPU to execute multiple tasks at the same time. Multithreading let CPU to execute multiple threads of a process simultaneously.
- A thread is the smallest unit in multi-threading. A process is the smallest unit in multitasking.
- Multitasking is when a single CPU performs several tasks (program, process, task, threads) at the same time. Multithreading allows multiple threads of a single task (program, process) to be processed by CPU at the same time.
- Multitasking allocates separate memory and resources for each process/program whereas, in multithreading threads belonging to the same process shares the same memory and resources as that of the process.
- Multithreading is less costly than multitasking as threads are easy to create than a process.
- Multithreading is fast in processing while multitasking is slower as comparison to multithreading.

2.1.1 Benefits of Multithreaded Programming

- Following are the benefits of multithreaded programming:
 1. **Resource sharing:** A thread shares memory and resources by default. The code sharing allows an application to have several thread activity within the same address space. So it is beneficial for memory.
 2. **Responsiveness:** Multithreaded application gives more response to the users. For example, a web browser could allow us to login even if it is downloading some other image with different thread.
 3. **Economy:** If we allocate memory and resources for each process then it is costly. But here, thread shares resources of the same process, so it becomes economical. For example, in Solaris, creating a process is about 30 times slower than creating a thread.
 4. **Utilization of multiprocessor architecture:** Multithreading increases the usage of multi-CPU architecture.
- In short, we can say that:
 1. Multithreading requires less overhead.
 2. Threads are light weight.
 3. They shares same address space.
 4. Inexpensive inter-process communication.

2.1.2 Disadvantages of Multithreading

- Various disadvantages of multithreading are:
 1. Programming and debugging is more complex.
 2. Programmer may face race conditions or deadlocks.
 3. Thread consumes more processor time.
 4. Operating system spends more time in managing and switching between the threads.

2.2 LIFE CYCLE OF THREAD / STATES OF THREAD [April 17, Oct. 18]

- 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. Fig. 2.3 shows the different states of threads.

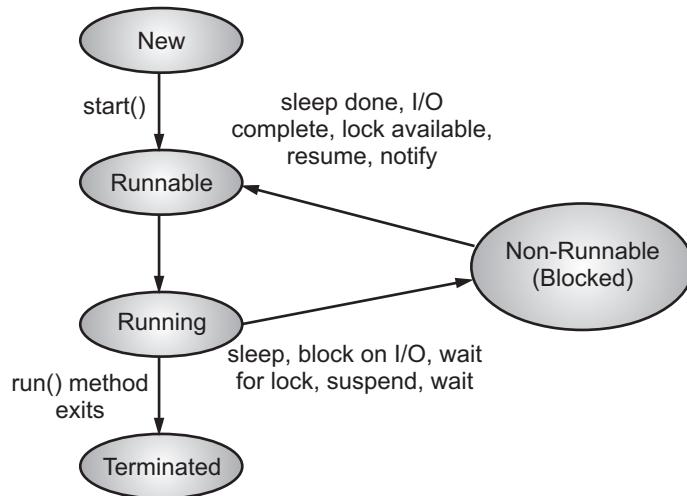


Fig. 2.3: Different states of thread

- Above mentioned stages are explained here:
 1. **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 start() method.
 2. **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.
 3. **Running:** The thread is in running state if the thread scheduler has selected it.
 4. **Non-Runnable (Blocked):** This is the state when the thread is still alive, but is currently not eligible to run.
 5. **Terminated:** A thread is in terminated or dead state when its run() method exits.

2.3 CREATING THREADS

[April 16, 18, 19]

- The Java programming language allows us to create a program that contains one or more parts that can run simultaneously at the same time.
- This type of program is known as a multithreading program. Each part of this program is called a thread.
- We can create thread in java with two different ways, (See Fig. 2.4):
 - Extending the thread class, and
 - Implements runnable interface.

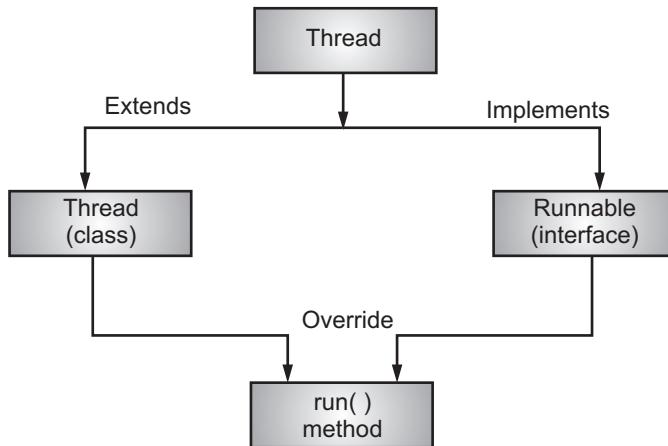


Fig. 2.4: Thread

2.3.1 Using a Thread Class

[April 16, 18, 19 Oct. 17]

- The one way to create a thread is to create a new class that extends Thread class using the following two simple steps.
- This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1: We will need to override run() 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();
```

Methods of Thread Class:

- public void start(): Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
- public void run(): If the Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.

3. `public final void setName(String name)`: Changes the name of the Thread object. There is also a `getName()` method for retrieving the name.
4. `public final void setPriority(int priority)`: Sets the priority of this Thread object. The possible values are between 1 and 10.
5. `public final void setDaemon(boolean on)`: A parameter of true denotes this Thread as a daemon thread.
6. `public final void join(long millisec)`: The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.

[April 16, Oct. 17]

7. `public void interrupt()`: Interrupts this thread, causing it to continue execution if it was blocked for any reason.
 8. `public final boolean isAlive()`: Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.
- The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static.
 - Invoking one of the static methods performs the operation on the currently running thread.
 1. `public static void yield()`: Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled. **[April 18]**
 2. `public static void sleep(long millisec)`: Causes the currently running thread to block for at least the specified number of milliseconds.
 3. `public static boolean holdsLock(Object x)`: Returns true if the current thread holds the lock on the given Object.
 4. `public static Thread currentThread()`: Returns a reference to the currently running thread, which is the thread that invokes this method.

Program 2.1: Program for creating a thread.

```
public class DemoThread extends Thread
{
    public static void main(String[] args)
    {
        Thread thread1 = new Thread("My Thread1");
        Thread thread2 = new Thread("My Thread2");
        thread1.start();
        thread2.start();
        System.out.println("Thread names are following:");
    }
}
```

```
        System.out.println(thread1.getName());
        System.out.println(thread2.getName());
    }
    public void run()
    {
    }
}
```

Output:

```
Thread names are following:
My Thread1
My Thread2
```

Program 2.2: Program for thread creation.

```
class ThreadDemo extends Thread
{
    private Thread t;
    private String threadName;
    ThreadDemo(String name)
    {
        threadName = name;
        System.out.println("Creating " + threadName );
    }
    public void run()
    {
        System.out.println("Running " + threadName );
        try
        {
            for(int i = 4; i > 0; i--)
            {
                System.out.println("Thread: " + threadName + ", " + i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
        }
        System.out.println("Thread " + threadName + " exiting.");
    }
}
```

```
public void start ()  
{  
    System.out.println("Starting " + threadName );  
    if(t == null)  
    {  
        t = new Thread (this, threadName);  
        t.start ();  
    }  
}  
}  
  
public class TestThread  
{  
    public static void main(String args[])  
    {  
        ThreadDemo T1 = new ThreadDemo( "Thread-1");  
        T1.start();  
        ThreadDemo T2 = new ThreadDemo( "Thread-2");  
        T2.start();  
    }  
}
```

Output:

```
Creating Thread-1  
Starting Thread-1  
Creating Thread-2  
Starting Thread-2  
Running Thread-1  
Running Thread-2  
Thread: Thread-1, 4  
Thread: Thread-2, 4  
Thread: Thread-1, 3  
Thread: Thread-2, 3  
Thread: Thread-1, 2  
Thread: Thread-2, 2  
Thread: Thread-2, 1  
Thread: Thread-1, 1  
Thread Thread-2 exiting.  
Thread Thread-1 exiting.
```

2.3.2 main Thread

- When a Java program starts up, one thread begins running immediately. This is usually called the main thread.
- The main thread creates automatically when program is started.
- The main thread is important for two reasons:
 1. It is the thread from which other “child” threads will be spawned.
 2. Often, it must be the last thread to finish execution because it performs various shutdown actions.
- In every program there will be at least one thread running and it is called the main thread.

Program 2.3: Simple java main thread program.

```
class MainthreadDemo
{
    public static void main (String args[])
    {
        System.out.println("main thread");
    } // end main
} //end class
```

Output:

```
main thread
```

- When this program is executed, it creates main thread. This reference is available in java-run-time system. This is represented in Fig. 2.5.

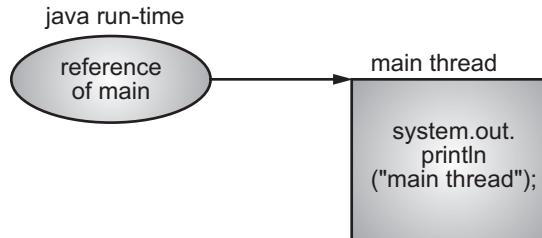


Fig. 2.5: main Thread

- Although the main thread is created automatically when your program is started, it can be controlled through a Thread object.
- To do so, we must obtain a reference to it by calling the method `currentThread()`, which is a public static member of Thread.
- Its general **form/syntax** is: `static Thread currentThread()`
- This method returns a reference to the thread in which it is called. Once you have a reference to the main thread, you can control it just like any other thread.

Program 2.4: Program for how to get reference of a thread. This is just to check whether a thread is available in a simple java program or not.

```
class Mythread
{
    public static void main(String args[])
    {
        Thread d = Thread.currentThread();      //to find reference
        System.out.println("current thread is" + d);
    }
} // end class
```

Output:

current thread isThread[main,5,main]

- In the above Program 2.4, we obtained the reference by calling currentThread() method. This reference is stored in a variable of type Thread.
- Here, it displays three things:
 - name of Thread,
 - its priority, and
 - group of Thread.
- In java, default name of mainThread is main. By default it's priority is 5. Last main is the name of group to which this thread belongs. This is represented in Fig. 2.6.

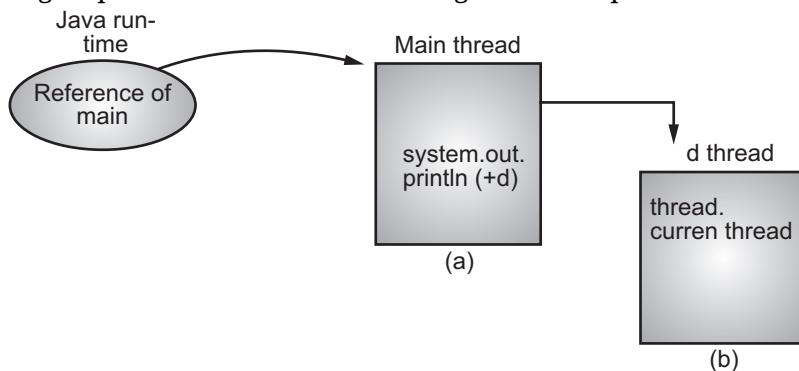


Fig. 2.6: (a) main Thread, (b) Created Thread (when object is created)

Program 2.5: Illustrates the usage of setName() and getPriority() methods.

```
import java.lang.*;
class MytestThread
{
    public static void main(String args[])
    {
        Thread thread = Thread.currentThread();
```

```

        System.out.println("current Thread is" + thread);
        thread.setName("mythread");
        System.out.println("New name" + thread);
        int priority = thread.getPriority();
        thread.setPriority(10);
        System.out.println("The priority is : " + thread.getPriority());
    }
}

```

Output:

```

current Thread isThread[main,5,main]
New nameThread[mythread,5,main]
The priority is : 10

```

2.3.3 Runnable Interface**[April 16, 18, 19]**

- 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 run() method provided by Runnable interface. This method provides entry point for the thread and you will put your complete business logic inside this method. Following is simple syntax of run() method:

```
public void run( )
```

Step 2 : At second step you will instantiate a Thread object using the following constructor:

```
Thread(Runnable threadObj, String threadName);
```

Where, threadObj is an instance of a class that implements the Runnable interface and threadName is the name given to the new thread.

Step 3 : 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( );
```

Program 2.6: Program creates a new thread and starts it running.

```

class RunnableDemo implements Runnable
{
    private Thread t;
    private String threadName;

```

```
RunnableDemo( String name)
{
    threadName = name;
    System.out.println("Creating " + threadName );
}
public void run()
{
    System.out.println("Running " + threadName );
try
{
    for(int i = 4; i > 0; i--)
    {
        System.out.println("Thread: " + threadName + ", " + i);
        // Let the thread sleep for a while.
        Thread.sleep(50);
    }
}
catch (InterruptedException e)
{
    System.out.println("Thread " + threadName + " interrupted.");
}
System.out.println("Thread " + threadName + " exiting.");
}
public void start ()
{
    System.out.println("Starting " + threadName );
if(t == null)
{
    t = new Thread (this, threadName);
    t.start ();
}
}
public class TestMyThread
{
    public static void main(String args[])
    {
        RunnableDemo R1 = new RunnableDemo( "Thread-1");
        R1.start();
```

```
RunnableDemo R2 = new RunnableDemo( "Thread-2");
R2.start();
}
}
```

Output:

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-2
Running Thread-1
Thread: Thread-2, 4
Thread: Thread-1, 4
Thread: Thread-2, 3
Thread: Thread-1, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-2 exiting.
Thread Thread-1 exiting.
```

2.3.4 Thread Priorities**[April 16, 17, 19]**

- 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 `setPriority()` method is used which is a method of the class Thread Class.

Program 2.7: Program of usage of Runnable interface with priority.

```
class TestMultiPriority extends Thread
{
    public void run()
    {
        System.out.println("Running thread name
                           is:"+Thread.currentThread().getName());
        System.out.println("Running thread priority
                           is:"+Thread.currentThread().getPriority());
    }
    public static void main(String args[])
    {
        TestMultiPriority m1=new TestMultiPriority();
        TestMultiPriority m2=new TestMultiPriority();
        m1.setPriority(Thread.MIN_PRIORITY);
        m2.setPriority(Thread.MAX_PRIORITY);
        m1.start();
        m2.start();
    }
}
```

Output:

```
Running thread name is:Thread-1
Running thread priority is:10
Running thread name is:Thread-0
Running thread priority is:1
```

2.4 RUNNING MULTIPLE THREADS

- The first way is to create a class which extends thread class and then create the instance of that class. This extended class must override method run().
- It must also call start() method. The Thread class is defined in package java.lang; so we have to import it.
- The following code segment tells the definition:

```
import java.lang.*;
public class Coun extends Thread
{
    public void run()
```

```
{  
    _____  
    _____  
    _____  
    _____  
}  
}  
}
```

- This will create a new class Coun and overrides method run(). The program 5.6 shows how to write Thread program.
-

Program 2.8: Program for multiple threads.

```
import java.lang.*;  
class Cons extends Thread  
{  
    //constructor  
    Cons()  
    {  
        start(); //starts the thread  
    }  
    public void run()  
    {  
        try  
        {  
            for (int k=1;k<=5;++k)  
            {  
                System.out.println("mythread" + k);  
                Thread.sleep(500);  
            }  
        }  
        catch(InterruptedException ob)  
        {  
        }  
        System.out.println("Thread exists");  
    } // end run  
} //end Cons
```

```
class Mainthread
{
    public static void main(String args[])
    {
        Cons c = new Cons();
        try
        {
            for (int i=1;i<=5;++i)
            {
                System.out.println("Main Thread" +i);
                Thread.sleep(1000);
            } //end for
        } //end try
        catch (InterruptedException ob)
        {
        }
        System.out.println("main Thread Exists");
    } //end main
} // end mainthread
```

Output:

```
mythread1
Main Thread1
mythread2
mythread3
Main Thread2
mythread4
mythread5
Main Thread3
Thread exists
Main Thread4
Main Thread5
main Thread Exists
```

- This output may change PC to PC. In this program, two threads mainthread and mythread (coun) runs simultaneously.
 - In the above example, mythread m suspends threads for 500 millisecond by calling its sleep method. The mainthread first get control of CPU and count as 1 and then suspended for 1000 milliseconds.
-

- The Program 2.9 illustrates the calling of the start method from main method and not from constructor. In a constructor, we will use a super method. It gets one argument as string which is name of a thread. In this program, we will create three different objects.

Program 2.9: Program to use of super in threading.

```
class mythread extends Thread
{
    mythread(String name)
    {
        super(name);
    }
    public void run()
    {
        try
        {
            for(int k=5;k>0;k--)
            {
                System.out.println(getName() + k);
                Thread.sleep(500);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println("Thread interrupted");
        }
        System.out.println("Thread exists");
    } //end run()
} //end mythread
//Main class for thread object
class Mainthread1
{
    public static void main(String args[])
    {
        mythread ob1 = new mythread ("First");
        mythread ob2 = new mythread ("Second");
        mythread ob3 = new mythread ("Third");
        ob1.start();
        ob2.start();
        ob3.start();
    }
}
```

```
try
{
    for(int k=5;k>0;k--)
    {
        System.out.println("main thread" +k);
        Thread.sleep(500);
    }
}
catch(InterruptedException e)
{
    System.out.println("interrupted thread");
}
System.out.println("main thread exists");
} //end main
} //end class mainthread
```

Output:

```
main thread5
Third5
First5
Second5
Third4
First4
Second4
main thread4
Third3
Second3
First3
main thread3
Third2
First2
Second2
main thread2
Third1
Second1
First1
main thread1
Thread exists
main thread exists
Thread exists
Thread exists
```

- In above Program 2.9, the constructor is parameterized which takes one argument of type string. This argument is the name of thread.
- First, second and third are the names of thread. So in this program, four threads are running simultaneously. This is shown in Fig. 2.7.

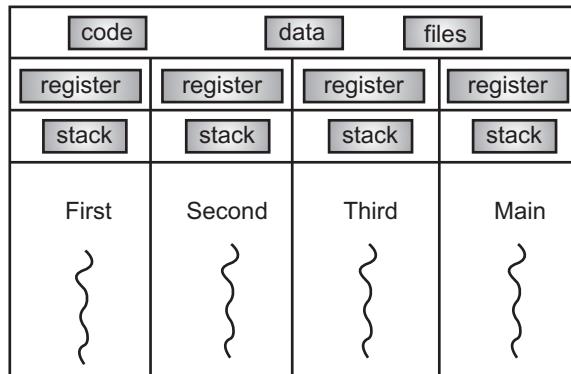


Fig. 2.7: Four threads with CPU

2.5 SYNCHRONIZATION, INTERTHREAD COMMUNICATION

[April 17, 19]

- In this section we will study synchronization and inter-thread communication in Java.

2.5.1 Synchronization

[April 17, April 19]

- Multithreading introduces asynchronous behavior 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 mutex 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.
- As the first thread exits from the monitor, one thread will acquire monitor from the waiting list of threads.
- Java programming language provides a very handy way of creating threads and synchronizing their task by using synchronized blocks.
- We keep shared resources within this block. Following is the general form of the synchronized statement:

```
synchronized(objectidentifier)
{
    // Access shared variables and other shared resources
}
```

- Here, the object identifier is a reference to an object whose lock associates with the monitor that the synchronized statement represents.
- Synchronization in java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Program 2.10: Program for synchronization.

```

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[i]);
        }
    }
}
/* Main class */
class MySynchro
{
    public static void main(String args[])
    {
        mythread t1=new mythread("Thread 1: ");
        mythread t2=new mythread("Thread 2: ");
        t1.start();
        t2.start();
        System.out.println("Main thread exited");
    }
}

```

Output:

```

Main thread exited
Thread 1: Java
Thread 2: Java
Thread 1: Supports
Thread 1: Multithreading
Thread 1: Concept
Thread 2: Supports
Thread 2: Multithreading
Thread 2: Concept
Exit from Thread 1:
Exit from Thread 2:

```

2.5.2 Inter-thread Communication**[April 17]**

- Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.
- Inter-thread communication is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.
- It is implemented by following methods like wait(), notify(), notifyAll() of Object class, let's see in detail.
 1. `wait()` method is used to tell the calling thread to give up the lock and go to sleep until some other thread enters the same lock and calls `notify()`. **[April 17, Oct. 18]**
Syntax: `public final void wait(long timeout)`
 2. `notify()` method wakes up the first thread that called `wait()` on the same object. **[April 16, 17, Oct. 18]**
Syntax: `public final void notify()`
 3. `notifyAll()` method wakes up all the threads that called `wait()` on the same object. Here, the highest priority thread will run first. **[April 16, 17, Oct. 18]**
Syntax: `public final void notifyAll()`

Program 2.11: Program shows how two thread can communicate using `wait()` and `notify()` method. We can create a complex system using the same concept.

```

class Chat
{
    boolean flag = false;
    public synchronized void Question(String msg)
    {

```

```
if (flag)
{
    try
    {
        wait();
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
System.out.println(msg);
flag = true;
notify();
}

public synchronized void Answer(String msg)
{
    if (!flag)
    {
        try
        {
            wait();
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
    System.out.println(msg);
    flag = false;
    notify();
}

class T1 implements Runnable
{
    Chat m;
    String[] s1 = { "Hi", "How are you?", "I am also doing fine!" };
}
```

```
public T1(Chat m1)
{
    this.m = m1;
    new Thread(this, "Question").start();
}
public void run()
{
    for (int i = 0; i < s1.length; i++)
    {
        m.Question(s1[i]);
    }
}
class T2 implements Runnable
{
    Chat m;
    String[] s2 = { "Hi", "I am good, what about you?", "Great!" };
    public T2(Chat m2)
    {
        this.m = m2;
        new Thread(this, "Answer").start();
    }
    public void run()
    {
        for (int i = 0; i < s2.length; i++)
        {
            m.Answer(s2[i]);
        }
    }
}
public class MyDemoThread
{
    public static void main(String[] args)
    {
        Chat m = new Chat();
        new T1(m);
        new T2(m);
    }
}
```

Output:

```
Hi  
Hi  
How are you?  
I am good, what about you?  
I am also doing fine!  
Great!
```

ADDITIONAL PROGRAMS

Program 1: Program to display the 100, 99, 98,.....1 using thread.

```
class MyThredDemo1  
{  
    public static void main(String args[])  
    {  
        Thread t = Thread.currentThread();  
        System.out.println("Current thread is : " + t);  
        t.setName("Demo Thread");  
        System.out.println("After changing the name thread is : " + t);  
        try  
        {  
            for(int n = 100; n > 0; n--)  
            {  
                System.out.println(n);  
                Thread.sleep(1000);  
            }  
        }  
        catch (InterruptedException e)  
        {  
            System.out.println("Thread interrupted");  
        }  
    }  
}
```

Output:

```
Current thread is : Thread[main,5,main]  
After changing the name thread is : Thread[Demo Thread,5,main]  
100
```

99
98
97
96
95
94
93
92
91
90
89
88
87
86
85
84
83
82
81
80
79
78
77
76
75
74
73
72
71
70
69
68
67
66
65
64

63
62
61
60
59
58
57
56
55
54
53
52
51
50
49
48
47
46
45
44
43
42
41
40
39
38
37
36
35
34
33
32
31
30
29
28

27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1

Program 2: Program to use of sleep() method.

```
class NewThread implements Runnable
{
    Thread t;
    NewThread()
    {
        t = new Thread(this, "Demo Thread");
        System.out.println("Child Thread :" + t);
        t.start();
    }
}
```

```
public void run()
{
    try
    {
        for (int i = 0; i < 5; i++)
        {
            System.out.println("Child Thread :" + i);
            Thread.sleep(1000);
        }
    }
    catch (InterruptedException e)
    {
        System.out.println("Child Thread Interrupted");
    }
    System.out.println("Exiting child thread");
}

class DemoMyThread2
{
    public static void main(String args[])
    {
        new NewThread();
        try
        {
            for (int i = 0; i < 5; i++)
            {
                System.out.println("Main Thread : " + i);
                Thread.sleep(500);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        System.out.println("Exiting main thread");
    }
}
```

Output:

```
Child Thread :Thread[Demo Thread,5,main]
Child Thread :0
Main Thread : 0
Main Thread : 1
Child Thread :1
Main Thread : 2
Main Thread : 3
Child Thread :2
Main Thread : 4
Exiting main thread
Child Thread :3
Child Thread :4
Exiting child thread
```

Program 3: Program to use super().

```
class NewThread extends Thread
{
    NewThread()
    {
        super( "Demo Thread");
        System.out.println("Child Thread :" + this);
        start();
    }
    public void run()
    {
        try
        {
            for (int i = 0; i < 5;  i++)
            {
                System.out.println("Child Thread :" + i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Child Thread Interrupted");
        }
        System.out.println("Exiting child thread");
    }
}
```

```
class DemoMyThread3
{
    public static void main(String args[])
    {
        new NewThread();
        try
        {
            for (int i = 0; i < 5; i++)
            {
                System.out.println("Main Thread : " + i);
                Thread.sleep(500);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
        System.out.println("Exiting main thread");
    }
}
```

Output:

```
Child Thread :Thread[Demo Thread,5,main]
Main Thread : 0
Child Thread :0
Main Thread : 1
Child Thread :1
Main Thread : 2
Main Thread : 3
Child Thread :2
Main Thread : 4
Exiting main thread
Child Thread :3
Child Thread :4
Exiting child thread
```

Program 4: Program to display "BYE CORONA..." message n times using Runnable Interface.

```
import java.io.*;
public class DemoMyThread4 implements Runnable
{
    int i,no;
    DemoMyThread4(int n)
    {
        no = n;
    }
    public void run()
    {
        for(i = 1; i<=no; i++)
        {
            System.out.println("BYE CORONA...");
            try
            {
                Thread.sleep(50);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
    public static void main(String args[])
    {
        try
        {
            int n;
            System.out.println("\nHow many time you want? ");
            BufferedReader br = new BufferedReader(new
                                                InputStreamReader(System.in));
            String str = br.readLine();
            n = Integer.parseInt(str);
            Thread t = new Thread(new DemoMyThread4(n));
            t.start();
        }
    }
}
```

```

        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}

```

Output:

```

How many time you want?
5
BYE CORONA...
BYE CORONA...
BYE CORONA...
BYE CORONA...
BYE CORONA...

```

Program 5: Program to define a thread for printing text on output screen for ‘n’ number of times. Create 3 threads and run them. Pass the text ‘n’ parameters to the thread constructor.

Example:

- (a) First thread prints “COVID19” 10 times.
- (b) Second thread prints “LOCKDOWN2020” 20 times
- (c) Third thread prints “VACCINATED2021” 30 times

```

import java.io.*;
import java.lang.String.*;
class TestPrint extends Thread
{
    String msg="";
    int n;
    TestPrint(String msg,int n)
    {
        this.msg=msg;
        this.n=n;
    }
    public void run()
    {
        try
        {

```

```
        for(int i=1;i<=n;i++)
        {
            System.out.println(msg+" "+i+" times");
        }
        System.out.println("\n ");
    }
    catch(Exception e)
    {
    }
}
}

class DemoMyThread5
{
    public static void main(String args[])
    {
        int n=Integer.parseInt(args[0]);
        TestPrint t1=new TestPrint("COVID19",n);
        t1.start();
        TestPrint t2=new TestPrint("LOCKDOWN2020",n+10);
        t2.start();
        TestPrint t3=new TestPrint("VACCINATED2021",n+20);
        t3.start();
    }
}
```

Output:

```
VACCINATED2021 1 times
LOCKDOWN2020 1 times
COVID19 1 times
LOCKDOWN2020 2 times
VACCINATED2021 2 times
LOCKDOWN2020 3 times
COVID19 2 times
LOCKDOWN2020 4 times
VACCINATED2021 3 times
LOCKDOWN2020 5 times
COVID19 3 times
LOCKDOWN2020 6 times
VACCINATED2021 4 times
```

LOCKDOWN2020 7 times
COVID19 4 times
LOCKDOWN2020 8 times
VACCINATED2021 5 times
LOCKDOWN2020 9 times
COVID19 5 times
LOCKDOWN2020 10 times
VACCINATED2021 6 times
LOCKDOWN2020 11 times
COVID19 6 times
LOCKDOWN2020 12 times
VACCINATED2021 7 times
LOCKDOWN2020 13 times
COVID19 7 times
LOCKDOWN2020 14 times
VACCINATED2021 8 times
LOCKDOWN2020 15 times
LOCKDOWN2020 16 times
VACCINATED2021 9 times
VACCINATED2021 10 times
LOCKDOWN2020 17 times
VACCINATED2021 11 times
VACCINATED2021 12 times
VACCINATED2021 13 times
VACCINATED2021 14 times
VACCINATED2021 15 times
VACCINATED2021 16 times
VACCINATED2021 17 times
VACCINATED2021 18 times
VACCINATED2021 19 times
VACCINATED2021 20 times
VACCINATED2021 21 times
VACCINATED2021 22 times
VACCINATED2021 23 times
VACCINATED2021 24 times
VACCINATED2021 25 times
VACCINATED2021 26 times
VACCINATED2021 27 times

Program 6: Program to use the method to suspend thread.

```
class NewThread implements Runnable
{
    String thName;
    Thread t;
    NewThread(String name)
    {
        thName = name;
        t = new Thread(this);
        System.out.println("Thread started :" + t);
        t.start();
    }
    public void run()
    {
        try
        {
            for (int i = 0; i < 10; i++)
            {
                System.out.println(t.getName() + " " + i);
                Thread.sleep(200);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println(t + " Interrupted");
        }
        System.out.println(thName + " Exiting");
    }
}
class DemoThread6
{
    public static void main(String args[])
    {
        NewThread th1 = new NewThread("First");
        NewThread th2 = new NewThread("Second");
```

```
try
{
    Thread.sleep(600);
    th1.t.suspend();
    System.out.println("Thread 1 suspended");
    Thread.sleep(600);
    th1.t.resume();
    System.out.println("Thread 1 resumed");
    th2.t.suspend();
    System.out.println("Thread 2 suspended");
    Thread.sleep(600);
    th2.t.resume();
    System.out.println("Thread 2 resumed");
}
catch (InterruptedException e)
{
    System.out.println("Main Thread Interrupted");
}
try
{
    System.out.println("Waiting for threads to terminate");
    th1.t.join();
    th2.t.join();
}
catch (InterruptedException e)
{
    System.out.println("Main Thread Interrupted");
}
System.out.println("Exiting main thread");
}
```

Output:

```
Thread started :Thread[Thread-0,5,main]
Thread started :Thread[Thread-1,5,main]
Thread-0  0
Thread-1  0
```

```
Thread-0 1
Thread-1 1
Thread-0 2
Thread-1 2
Thread 1 suspended
Thread-1 3
Thread-1 4
Thread-1 5
Thread 1 resumed
Thread-0 3
Thread 2 suspended
Thread-0 4
Thread-0 5
Thread-0 6
Thread 2 resumed
Thread-1 6
Waiting for threads to terminate
Thread-1 7
Thread-0 7
Thread-0 8
Thread-1 8
Thread-0 9
Thread-1 9
Second Exiting
First Exiting
Exiting main thread
```

Program 7: Program to use the synchronized() method.

```
import java.io.*;
import java.lang.*;
class A
{
    synchronized void first(B b)
    {
        String threadName = Thread.currentThread().getName();
        System.out.println(threadName + "Executing A.first()");
        System.out.println(threadName + "Trying to execute B.last()");
        b.last();
    }
}
```

```
        synchronized void last()
    {
        System.out.println("Inside A.last()");
    }
}

class B
{
    synchronized void first(A a)
    {
        String threadName = Thread.currentThread().getName();
        System.out.println(threadName + "Executing B.first()");
        System.out.println(threadName + "Trying to execute A.last()");
        a.last();
    }

    synchronized void last()
    {
        System.out.println("Inside B.last()");
    }
}

class Deadlock implements Runnable
{
    A a = new A();
    B b = new B();

    Deadlock()
    {
        Thread.currentThread().setName("Main Thread");
        Thread t = new Thread(this, "Child Thread");
        t.start();
        a.first(b);
        System.out.println("Back in main thread");
    }

    public void run()
    {
        b.first(a);
        System.out.println("Back in child thread");
    }
}
```

```

public static void main(String args[]) throws IOException
{
    new Deadlock();
}
}

```

Output:

```

Main ThreadExecuting A.first()
Main ThreadTrying to execute B.last()
Child ThreadExecuting B.first()
Child ThreadTrying to execute A.last()
-
-
-
```

Program 8: Program to use of wait(),notify() methods.

```

import java.io.*;
import java.lang.*;
class Shared
{
    int a;
    boolean valueChanged = false;
    synchronized int get_data()
    {
        if (!valueChanged)
            try
            {
                wait();
            }
            catch (InterruptedException e)
            {
                System.out.println(" Interrupted");
            }
        System.out.println("Read : " + a);
        valueChanged = false;
        notify();
        return a;
    }
}

```

```
    synchronized void put_data(int n)
    {
        if (valueChanged)
            try
            {
                wait();
            }
            catch (InterruptedException e)
            {
                System.out.println( "Interrupted");
            }
        this.a = n;
        valueChanged = true;
        System.out.println("Written : " + a);
        notify();
    }
}//Shared End
class Producer implements Runnable
{
    Shared ob;
    Producer (Shared ob)
    {
        this.ob = ob;
        new Thread(this, "Producer").start();
    }
    public void run()
    {
        int j = 0;
        while(true)
        {
            ob.put_data(j++);
        }
    }
}//Producer End
class Consumer implements Runnable
{
    Shared ob;
```

```
Consumer (Shared ob)
{
    this.ob = ob;
    new Thread(this, "Producer").start();
}
public void run()
{
    while(true)
    {
        ob.get_data();
    }
}
}//Consumer End
class DemoThread8
{
    public static void main(String args[])throws IOException
    {
        Shared ob = new Shared();
        new Producer(ob);
        new Consumer(ob);
        System.out.println("Press Cntl+c to stop");
    }
}
```

Output:

```
Press Cntl+c to stop
Written : 0
Read : 0
Written : 1
Read : 1
Written : 2
Read : 2
Written : 3
Read : 3
Written : 4
Read : 4
Written : 5
```

```
Read : 5
Written : 6
Read : 6
Written : 7
Read : 7
Written : 8
Read : 8
Written : 9
Read : 9
Written : 10
Read : 10
```

Program 9: Program to use of join() methods.

```
import java.io.*;
class Maths_op
{
    int a = 5;
    void add_op(int b)
    {
        try
        {
            Thread.sleep(500);
        }
        catch (InterruptedException e)
        {
            System.out.println(" Interrupted Thread");
        }
        a += b;
        System.out.println("a = " + a);
    }
}
class NewThread implements Runnable
{
    Thread t;
    Maths_op ob1;
    int op2;
```

```
        NewThread(Maths_op ob, int p)
    {
        ob1 = ob;
        op2 = p;
        t = new Thread(this);
        t.start();
    }
    public void run()
    {
        synchronized (ob1)
        {
            ob1.add_op(op2);
        }
    }
}
class DemoThread9
{
    public static void main(String args[])throws IOException
    {
        Maths_op ob = new Maths_op();
        NewThread th1 = new NewThread(ob, 100);
        NewThread th2 = new NewThread( ob, 200);
        NewThread th3 = new NewThread( ob, 300);
        try
        {
            th1.t.join();
            th2.t.join();
            th3.t.join();
        }
        catch (InterruptedException e)
        {
            System.out.println("Main Thread Interrupted");
        }
    }
}
```

Output:

```
a = 105
a = 405
a = 605
```

Program 10: Program to use of priority thread.

```
class NewThread implements Runnable
{
    Thread t;
    NewThread(String thname, int p)
    {
        t = new Thread(this);
        t.setName(thname);
        t.setPriority(p);
    }
    void start()
    {
        t.start();
    }
    public void run()
    {
        try
        {
            for (int i = 0; i <10;  i++)
            {
                System.out.println(t.getName()  +"  " + i);
            }
        }
        catch (Exception e)
        {
            System.out.println(" Interrupted");
        }
        System.out.println(t.getName()  + " Exiting");
    }
}
class DemoThread10
{
    public static void main(String args[])
    {
        NewThread th1 = new NewThread("COVISHIELD", Thread.NORM_PRIORITY + 2);
        NewThread th2 = new NewThread("COVAXI", Thread.NORM_PRIORITY - 2);
        th1.start();
        th2.start();
    }
}
```

```
try
{
    Thread.sleep(1000);
}
catch (InterruptedException e)
{
    System.out.println("Main Thread Interrupted");
}
try
{
    System.out.println("Waiting for threads to terminate");
    th1.t.join();
    th2.t.join();
}
catch (InterruptedException e)
{
    System.out.println("Main Thread Interrupted");
}
System.out.println("Exiting main thread");
}
```

}

Output:

```
COVISHIELD 0
COVISHIELD 1
COVISHIELD 2
COVISHIELD 3
COVISHIELD 4
COVISHIELD 5
COVISHIELD 6
COVISHIELD 7
COVISHIELD 8
COVISHIELD 9
COVISHIELD Exiting
COVAXI 0
COVAXI 1
COVAXI 2
```

```
COVAXI 3
COVAXI 4
COVAXI 5
COVAXI 6
COVAXI 7
COVAXI 8
COVAXI 9
COVAXI Exiting
Waiting for threads to terminate
Exiting main thread
```

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. What is multithreaded programming?
 - (a) It is a process in which two different processes run simultaneously.
 - (b) It is a process in which two or more parts of same process run simultaneously.
 - (c) It is a process in which many different process are able to access same information.
 - (d) It is a process in which a single process can access information from many sources.
2. Which is a single flow of control (or smallest unit of execution or processing) within a program?

| | |
|-----------------|----------------|
| (a) Collections | (b) Interfaces |
| (c) Thread | (d) Methods |
3. Which program contains two or more parts that can run concurrently?

| | |
|-----------------------|---------------------------|
| (a) processed program | (b) multithreaded program |
| (c) classed program | (d) None of the mentioned |
4. Inter-thread communication consists of following which methods,

| | |
|-----------------|--------------------------|
| (a) wait() | (b) notify() |
| (c) notifyAll() | (d) All of the mentioned |
5. Which of these methods of Thread class is used to suspend a thread for a period of time?

| | |
|---------------|-----------------|
| (a) sleep() | (b) terminate() |
| (c) suspend() | (d) stop() |
6. Which of this method can be used to make the main thread to be executed last among all the threads?

| | |
|------------|-------------|
| (a) stop() | (b) sleep() |
| (c) join() | (d) call() |

7. What is synchronization in reference to a thread?
 - (a) It is a process of handling situations when two or more threads need access to a shared resource.
 - (b) It is a process by which many threads are able to access same shared resource simultaneously.
 - (c) It is a process by which a method is able to access many different threads simultaneously.
 - (d) It is a method that allows too many threads to access any information required.
8. Which of these keywords are used to implement synchronization?

| | |
|-----------------|------------------|
| (a) synchronize | (b) syn |
| (c) synch | (d) synchronized |
9. Which of the following will ensure the thread will be in running state?

| | |
|-------------|-------------------------|
| (a) yield() | (b) notify() |
| (c) wait() | (d) Thread.killThread() |
10. Which of these methods is used to explicitly set the priority of a thread?

| | |
|-------------------|--------------------|
| (a) prioritySet() | (b) getPriority() |
| (c) setPriority() | (d) fillPriority() |
11. Which of this method is used to find out that a thread is still running or not?

| | |
|---------------|----------------|
| (a) isAlive() | (b) Alive() |
| (c) run() | (d) checkRun() |
12. Which of the following is a correct constructor for thread?

| | |
|--------------------------------------|---------------------------------------|
| (a) Thread(Runnable a, String str) | (b) Thread(int priority) |
| (c) Thread(Runnable a, int priority) | (d) Thread(Runnable a, ThreadGroup t) |
13. What is the default value of priority variable MIN_PRIORITY AND MAX_PRIORITY?

| | |
|--------------|---------------|
| (a) 0 and 1 | (b) 0 and 256 |
| (c) 1 and 10 | (d) 1 and 256 |
14. The ways to create a thread include,
 - (a) by extending the Thread class and overriding its run() method.
 - (b) to implement the Runnable interface.
 - (c) Both (a) and (b)
 - (d) None of the mentioned
15. Whenever a new thread is created, it is always in the,

| | |
|-------------------|--------------------|
| (a) waiting state | (b) runnable state |
| (c) running state | (d) new state |

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|---------|--------|--------|--------|--------|---------|
| 1. (a) | 2. (c) | 3. (b) | 4. (d) | 5. (a) | 6. (b) | 7. (a) | 8. (d) | 9. (c) | 10. (c) |
| 11. (a) | 12. (a) | 13. (c) | 14. (c) | 15. (d) | | | | | |

Q. II Fill in the Blanks:

1. A thread is a _____ sub-process.
2. _____ method is used to check if a thread is running.
3. _____ is the capability to control the access of multiple threads to any shared resource.
4. The notify() is used to _____ one waiting thread.
5. _____ method waits for a thread to die.
6. Thread either by extending _____ class or implementing _____ interface.
7. _____ is maximum thread priority.
8. _____ method is called internally by Thread start() method.
9. _____ method of Thread class is used to find out the priority given to a thread.
10. The process of executing multiple threads simultaneously is termed as _____.
11. A thread is a computational execution/processing unit within a _____.
12. Multithreading refers to a process of executing _____ threads simultaneously.
13. Thread class provides the methods and constructors to create and perform _____ on a thread.
14. Runnable interface is implemented whose instances are intended to be executed by a thread and has only _____ method run().
15. The priorities of threads are simply numbers (1 to 10) that are essential for determining the _____ of a thread.
16. The methods that are essential for _____ communication are wait(), notify(), and notifyAll().
17. When there is a need to access the _____ resources by two or more threads, then synchronization is used.
18. Multithreaded programs execute two or more threads run _____.
19. The _____ thread is created automatically when the program is started.

Answers

| | | | |
|------------------|---------------------|--------------------|------------------|
| 1. lightweight | 2. isAlive() | 3. Synchronization | 4. unblock |
| 5. join() | 6. Thread, Runnable | 7. 10 | 8. run() |
| 9. getPriority() | 10. Multithreading | 11. process | 12. two or more |
| 13. operations | 14. one | 15. priority | 16. inter-thread |
| 17. shared | 18. concurrently | 19. main | |

Q. III State True or False:

1. In multi-threaded program each part of a program is called as a thread and each thread defines separate path of execution.

2. The start() method is used to begin execution of the thread.
3. A thread can be formed by implementing Runnable interface only.
4. Multithreading CPU idle time is minimized, and we can take maximum use of it.
5. The run() method creates new thread.
6. A thread is the smallest segment/part/unit of execution in the process (can divided into a number of threads executing concurrently).
7. A thread can exist only in two states, running and blocked.
8. Two threads in Java can have the same priority.
9. A thread can be formed by a class that extends Thread class.
10. The run() method is used to begin execution of a thread before start() method in special cases.
11. The lifecycle of each thread in Java has five different stages such as New, Runnable, Running, Waiting and Terminated.
12. A thread in Java is a heavyweight process requiring fewer resources to create and share the process resources.
13. Multithreading in Java is a process of executing two or more threads simultaneously.
14. The JVM allows an application to have multiple threads of execution running concurrently.
15. Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.
16. All Java programs have at least one thread, known as the main thread, which is created by the Java Virtual Machine (JVM) at the program's start, when the main() method is invoked with the main thread.
17. A thread is a lightweight process or the smallest unit of a process.
18. To create a runnable thread, we can implement the Runnable interface and override the public start() method.
19. The setPriority() method enables setting of thread priority.
20. When a Java program starts up, one thread begin running immediately called main thread of the program.

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 1. (T) | 2. (F) | 3. (F) | 4. (T) | 5. (F) | 6. (T) | 7. (F) | 8. (T) | 9. (F) | 10. (T) |
| 11. (T) | 12. (F) | 13. (T) | 14. (T) | 15. (T) | 16. (T) | 17. (T) | 18. (F) | 19. (T) | 20. (T) |

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define thread.
2. What is multithreaded program?

3. Which states occurs in life cycle of multithreading.
4. Define thread priority.
5. Explain the purpose of join() method in the threads.
6. Define inter-thread communication.
7. Java's multithreading built upon the Thread class. State true or false.
8. Compare thread and process. Any two points.
9. What is the use of notifyAll () method?
10. Which are the two ways that are used to create a new thread in Java?
11. Name the methods used for inter-thread communication.
12. Give the purpose of synchronization.
13. "A thread is a lightweight process". Comment.
14. What is the purpose of multithreading.

(B) Long Answer Questions:

1. What is multithreading? Explain with diagram.
2. What is synchronization? Why it is needed? Describe in detail.
3. What is thread? Define multithreaded program. What is its purpose in Java?
4. What is inter-thread communication? Describe with a program.
5. Describe life cycle of thread with diagram.
6. What is thread priority? How they can be used in program? Explain in detail.
7. How to create a thread? Explain with suitable example.
8. What is synchronization? How can thread synchronization be achieved in Java? Explain with example.
9. What is main thread? Describe with example.
10. Write a short note on: Running multiple threads.
11. What will be the output of the program?

```
class Test extends Thread
{
    public
        void run()
    {
        System.out.println("Run");
    }
} class Myclass {
public
    static void main(String[] args)
    {
        Test t = new Test();
        t.start();
    }
}
```

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. State two-way in creating thread. [1 M]
- Ans.** Refer to Sections 2.3, 2.3.1 and 2.3.3.
2. What is the purpose of join method in the context of thread? [1 M]
- Ans.** Refer to Section 2.3.1, Point (6).
3. Write a note on thread priorities. [5 M]
- Ans.** Refer to Section 2.3.4.
4. What is the use of notify() and notifyAll() methods? [4 M]
- Ans.** Refer to Section 2.5.2, Points (2) and (3).
-

April 2017

1. State any two methods of inner-thread communication. [1 M]
- Ans.** Refer to Section 2.5.2, Points (2) and (3).
2. How do we set priority to thread? [1 M]
- Ans.** Refer to Section 2.3.4.
3. Explain thread life cycle with its methods. [2 M]
- Ans.** Refer to Section 2.2.
-

October 2017

1. Explain the purpose of join method in the context of threads. [1 M]
- Ans.** Refer to Section 2.3.1, Point (6).
2. Give two differences between Multithreading and Multitasking. [1 M]
- Ans.** Refer to Section 2.1.
3. Write a program to display "examination" 50 times using multithreading. [5 M]
- Ans.** Refer to Additional Programs.
-

April 2018

1. What is Multithreading? [1 M]
- Ans.** Refer to Section 2.0.
2. "Java can control different types of multitasking" True/False – Justify. [1 M]
- Ans.** Refer to Section 2.1.
3. What are two different ways used to implement threading in Java? Explain with example. [5 M]
- Ans.** Refer to Sections 2.3, 2.3.1 and 2.3.3.
4. Explain the purpose of yield() method in the context of thread. [2 M]
- Ans.** Refer to Section 2.3.1.
-

October 2018

1. "Multithreading is better than multiprocessing". Comment. [1 M]
Ans. Refer to Section 2.1.
2. State any two methods for inter-thread communication. [1 M]
Ans. Refer to Section 2.5.2.
3. Explain life cycle of thread. [4 M]
Ans. Refer to Section 2.2.
-

April 2019

1. Why synchronization is used to implement threading in Java? [1 M]
Ans. Refer to Section 2.5.1.
2. How do we set priority to threads? [1 M]
Ans. Refer to Section 2.3.4.
3. Explain thread synchronization with an example. [5 M]
Ans. Refer to Section 2.5.1.
4. State two ways in creating a Thread. [2 M]
Ans. Refer to Sections 2.3, 2.3.1 and 2.3.3.
-
- ■ ■

Database Programming

Objectives...

- To understand Database Programming
 - To Study Basic Concepts of JDBC
 - To Learn Design and Derives of JDBC
 - To Study ResultSet, MetaData and Transactions
-

3.0 INTRODUCTION

- Java offers several benefits to the software developer creating a front-end application for a database server.
- Java is ‘Write Once Run Everywhere (WORA)’ language, means that Java programs may be deployed without recompilation on any computer architectures and operating systems that possesses a Java Virtual Machine (JVM).
- Java interacts with database using a common database application programming interface called as JDBC (Java DataBase Connectivity).
- JDBC is a Java API to connect and execute the query with the database. JDBC API uses JDBC drivers (JDBC-ODBC Bridge Driver, Native Driver etc.) to connect with the database.
- JDBC is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases such as Oracle, PostgreSQL, Microsoft Access, DB2, Sybase, MySQL etc.
- The Java JDBC API is part of the core JavaSE (Java Standard Edition) SDK, making JDBC available to all Java applications that want to use it. The API (Application Programming Interface) is a document that contains description of all the features of a product or software.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
 1. Making a connection to a database.

- 2. Creating SQL or PostgreSQL statements.
- 3. Executing SQL or PostgreSQL queries in the database.
- 4. Viewing and modifying the resulting records.
- Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.
- Java can be used to write different types of executables, such as Java Applications, Java Applets, Java Servlets, Java ServerPages (JSPs), Enterprise JavaBeans (EJBs) etc.
- JDBC API enables the java application to interact with different database. This is shown in Fig. 3.1.

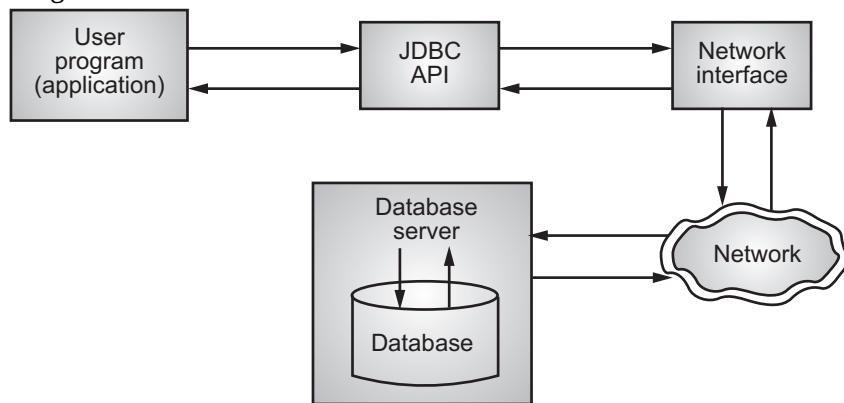


Fig. 3.1: Overview of JDBC

JDBC Architecture:

[April 18, Oct. 18]

- 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.
- Fig. 3.2 shows the location of the driver manager with respect to the JDBC drivers and the Java application.

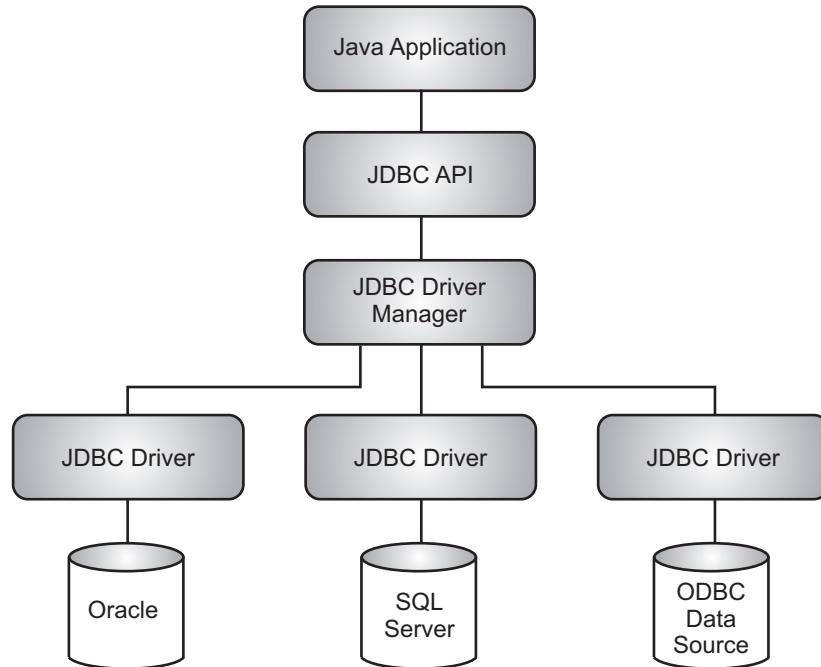


Fig. 3.2: JDBC Architecture

Common JDBC Components:

- The JDBC API provides the following interfaces and classes:
 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.
 4. **Statement** we use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
 5. **ResultSet Objects** hold data retrieved from a database after we execute an SQL query using Statement objects. It acts as an iterator to allow us to move through its data.
 6. **SQLException** class handles any errors that occur in a database application.

3.1 DESIGN OF JDBC

- There are many databases available in the market like MySQL, Sybase, DB2, Microsoft Access etc.
- In order to extending Java, Sun Microsystems wanted to create some API which can be common for all databases available in the market.
- The first problem for designing such API was language barrier i.e., each database vendor defines distinct way to communicate with application program. So the low level code written to communicate with an Oracle may be applicable for DB2.
- In 1996, Sun Microsystems creates JDBC Drivers and JDBC API. JDBC driver created by Sun was not drivers at all; it was specification that describes the detail functionality of JDBC Drivers.
- DBMS manufacturer and third party vendors were encouraged to build JDBC driver that conformed to sun's JDBC driver's specifications.
- These JDBC drivers require, it should translate SQL queries in to a language understood by JDBC API.
- JDBC drivers created by DBMS Manufacturers have to:
 1. Established a connection between database and java Component.
 2. Translate SQL command in to a form that can be understood by both database and java components.
 3. Returns any kind of error message that conforms to the JDBC specification to the JDBC driver.
 4. At the end it should close the connection between DBMS and Java Component.
- JDBC is an Application Programming Interface (API) used to connect Java application with database. Design of JDBC defines the components of JDBC, which is used for connecting to the database.
- JDBC is an API that establishes a connection between a standard database and Java application that intends to use that database. JDBC helps us to connect to a database and execute SQL statements against a database.
- Fig. 3.3 shows components for JDBC design. These components are explained below:
 1. **Application:** Application in JDBC like Java Applet or a Servlet that communicates with a data source. Application is the data processing application that intends to access the data from the different databases.
 2. **JDBC API:** JDBC API provides classes, methods, and interfaces that allow Java programs to execute SQL statements and retrieve results from the database. JDBC API provides various interfaces and methods to establish easy connection with different databases. JDBC API provides connection from Application to JDBC Manager. The JDBC API ensures that a stable connection is established between the data storage unit and the JDBC application.

- 3. Driver Manager:** The Driver Manager plays an important role in the JDBC design. JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database. The JDBC driver manager makes sure that the correct driver is being used to access the databases. It is also capable of handling multiple drivers connected to multiple databases simultaneously.
- 4. JDBC Drivers:** JDBC drivers help us to communicate with a data source (database) through JDBC. We need a JDBC driver that can intelligently interact with the respective data source.

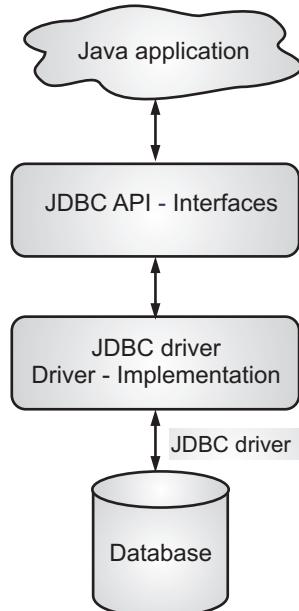


Fig. 3.3: Design of JDBC

3.2 JDBC DRIVERS

[April 17]

- 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.
- For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.
- Fig. 3.4 provides a high-level overview of the various types of drivers.
- The Types 1 and 2 Drivers rely heavily on additional software, (typically C/C++ DLLs) installed on the client computer to provide database connectivity. Java and JDBC use these components to interact with the database.

- The Types 3 and 4 Drivers are pure Java implementations and require no additional software to be installed on the client, except for the JDBC driver.

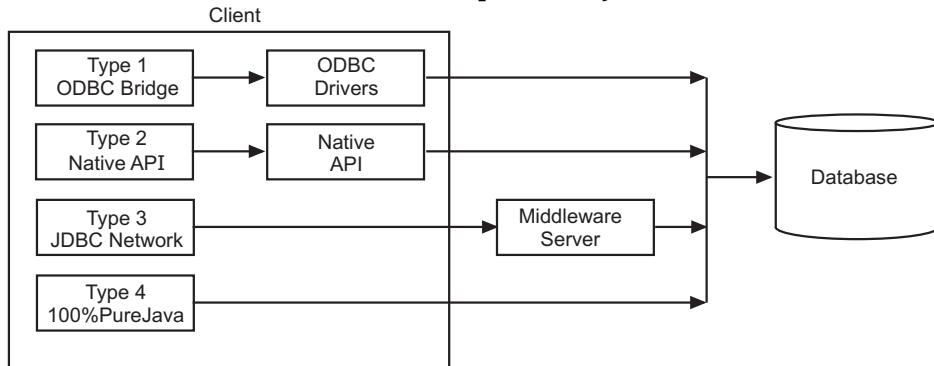


Fig. 3.4: Types JDBC Driver

- Fig. 3.5 shows the communication path between JDBC to database.

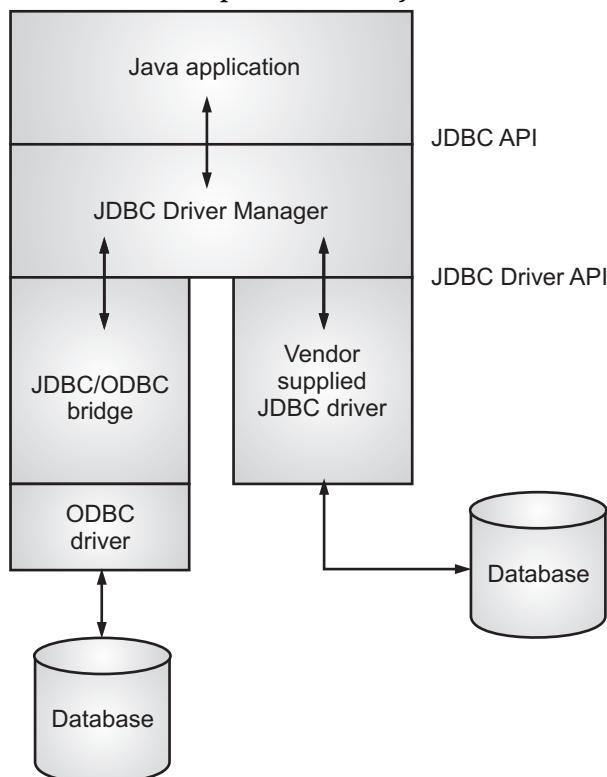


Fig. 3.5

3.2.1 Types of Drivers

[April 17, Oct. 18]

- The JDBC API encapsulates relational database access into a hierarchy of Java classes and interfaces.

- JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates.
- Sun has divided the implementation types into four categories of Drivers, Types 1, 2, 3, and 4, which is explained below:

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.
- When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.
- Fig. 3.6 shows Java JDBC-ODBC Bridge driver.

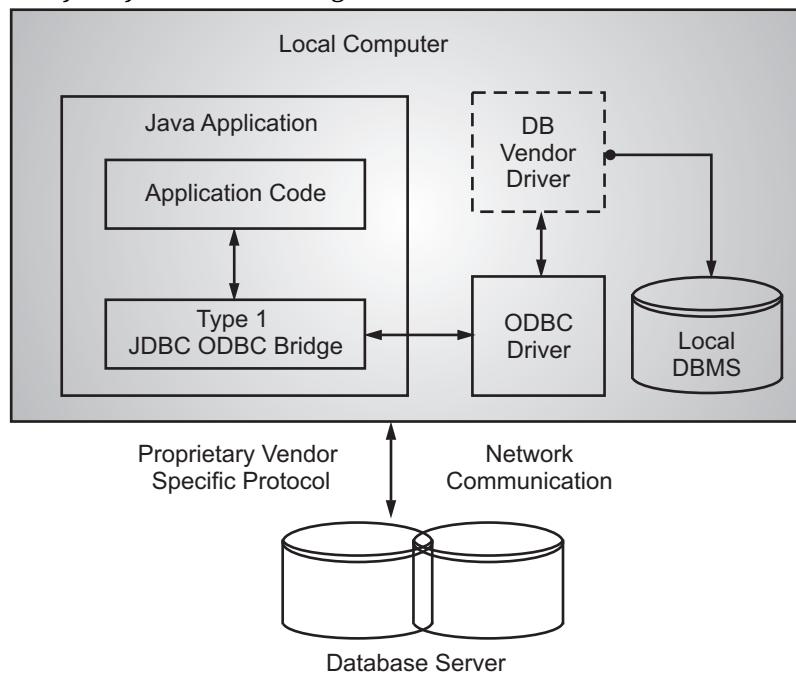


Fig. 3.6: JDBC - ODBC Bridge Driver

- The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Advantages:

1. Easy to use.
2. Can be easily connected to any database if that database has installed ODBC driver.

Disadvantages:

1. Performance degraded because JDBC method call is converted into the ODBC function calls.
2. The ODBC driver needs to be installed on the client machine.
3. As it is not written fully in java, so these drivers are not portable.
4. They are not much suitable for Web applications.

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.
- If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but they are fast as compare to Type 1 driver as Type 2 driver has no ODBC's overhead.
- Fig. 3.7 shows JDBC-Native API driver.

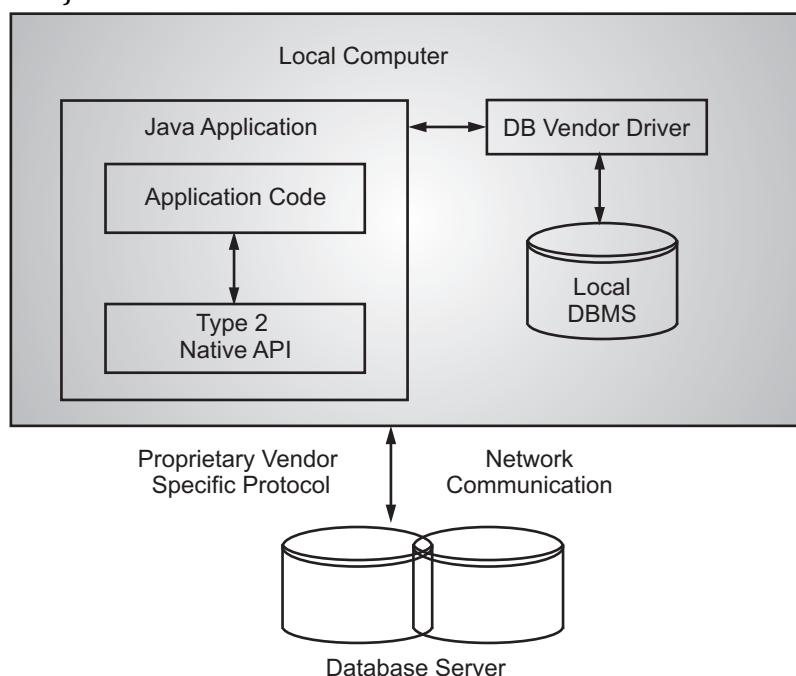


Fig. 3.7: JDBC-Native API Driver

- The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Advantages:

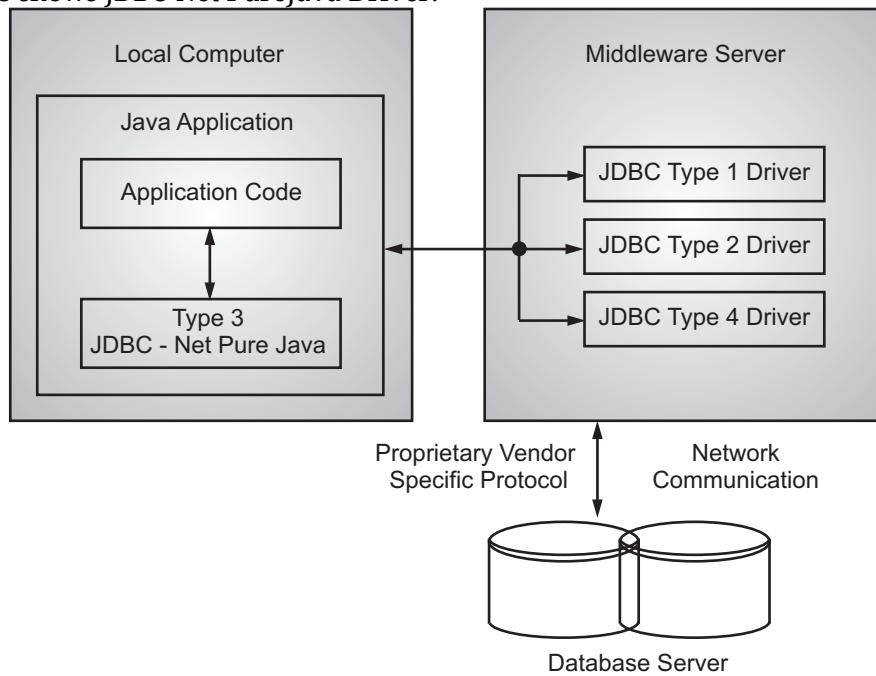
1. Faster as compared to Type 1 Driver as Type 2 drivers are database specific as well as they do not require translation.
2. Contains additional features.

Disadvantages:

1. Requires installation of native API on client machine so not suitable for web applications.
2. Type 2 drivers are also not written in java so they having portability issue.
3. Increased cost of application.
4. As we need to install database specific Native API on client machine, so if database change we need to change Native API.

Type 3: JDBC-Net Pure Java Driver:**[Oct. 18]**

- 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.
- Fig. 3.8 shows JDBC-Net PureJava Driver.

**Fig. 3.8: JDBC-Net Pure Java Driver**

- We can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application.
- As a result, we need some knowledge of the application server's configuration in order to effectively use this driver type.

- The application server might use a Type 1, 2, or 4 drivers to communicate with the database, understanding the nuances will prove helpful.

Advantages:

- It does not require any native library to be installed on the client.
- It has database independency.
- It provides facility to switch over from one database to another database.
- They support several networking options, such as HTTP tunneling.
- They are suitable for web applications.
- They are most suitable drivers type amongst all.

Disadvantages:

- Slow due to increase number of network call.
- They can be difficult to set up since; the architecture is complicated by the network interface.
- It is costliest JDBC driver.

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

[April 19]

- 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.
- Fig. 3.9 shows Native Protocol Pure Java driver.

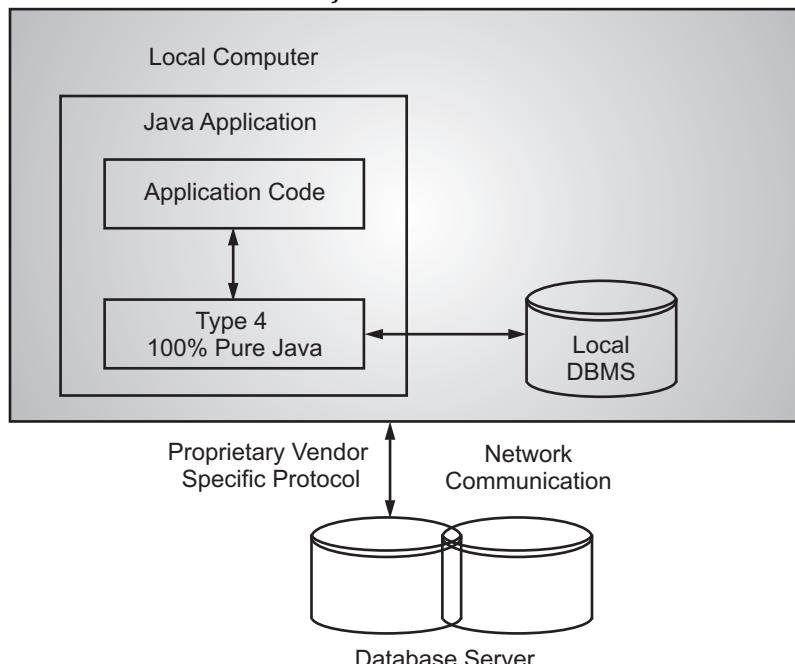


Fig. 3.9: Native Protocol Pure Java Drivers

- MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Advantages:**[April 19]**

- These are 100% pure java drivers so they are portable in nature.
- Better performance than all other drivers.
- No translation is required.
- No software is required at client side or server side.

Disadvantage:

- Drivers are depending on the Database.

JDBC Packages:**[Oct. 17]**

- JDBC API is contained in two packages:
 - java.sql**: This package provides basic classes which are used to connect with database and used to load and store data in database.
 - javax.sql**: It extends the java.sql and provides classes that are used to interact with Java Naming and Directory Interface (**JNDI**) and manages connection pooling among other advanced JDBC features.

3.3 EXECUTING SQL STATEMENTS**[April 17]**

- The next step after establishment of proper database connection is creating SQL statement, using them to work with database.
- One of the following three statements is used to execute database queries.

1. Statement:**[April 16, Oct. 17]**

- It is used to execute query immediately without compilation of the query.
- The `createStatement()` method is used to create a statement object.
- The statement object contains the `executeQuery()` method to which you can pass query as an argument.
- The `executeQuery()` method executes a query that we passed as an argument and returns a `ResultSet`.

[Oct. 17]

- The statement object also contains `execute()` and `executeUpdate()` methods.
- The `execute()` is used when there may be multiple results returned and `executeUpdate()` method is used when query contains update or delete operation that you want to perform on database.
- The `executeUpdate()` returns an integer value specifying the number of rows that are affected by query.

[April 18]

- **Use of executeQuery() Method:**

[Oct. 17]

```

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 execute() Method:**

```

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

```

- **Use of executeUpdate() Method:**

```

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

```

2. PreparedStatement:

[April 17, 18, Oct. 17]

- 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 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.
- Next there a executeQuery() method, here we need not to pass any parameter because the query that we want to execute is already associated with prepared statement object.

```

private Connection con;
PreparedStatement pst;
ResultSet rs;
Try {
    Class.forName("org.postgresql.Driver");
    con = DriverManager.getConnection("jdbc:postgresql:EMP",
                                    "postgres", " ");
    String query="Select * from employee where eid=?";
    pst=con.prepareStatement(query);
    pst.setString(1, "10");
    rs=pst.executeQuery();
    pst.close();
    con.close();
}

```

3. Callable Statement:

- This statement is used to call stored procedure from java component. Callable statement object uses three types of parameters when calling a stored procedure, i.e. IN, OUT, INOUT.
- The IN parameter is used to pass input value to the procedure, for which again we use setxxx() method.
- The OUT parameter is used to store any value return by procedure. For this OUT parameter should be registered using registerOutParameter() method.
- The INOUT parameter is a single parameter that is used for both, passing as well as retrieving information from procedure.
- The prepareCall() method is used to create an object of CallableStatement.

For example:

```

Private Connection con;
CallableStatement cst;
ResultSet rs;
Try {
    Class.forName("org.postgresql.Driver");
    con = DriverManager.getConnection("jdbc:postgresql:EMP",
                                    "postgres", " ");
    String query="{ call EMPSALARY (?) } ";
    cst=con. prepareCall(query);
    cst.registerOutParameter(1, java.sql.Types. VARCHAR);

```

```

        rs=pst.execute ();
        EMPSALARY=cst.getString(1);
        cst.close();
        con.close();
    }

```

3.4 EXECUTING QUERIES

- Executing a database using query (or Querying) means accessing data from database. In general, to process any SQL statement with JDBC, we follow these steps:

Step 1: Establishing a Connection: First, establish a connection with the data source we want to use. A data source can be a DBMS, a legacy file system, or some other source of data with a corresponding JDBC driver. This connection is represented by a Connection object.

Step 2: Create a Statement: A Statement is an interface that represents a SQL statement. We execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set. We need a Connection object to create a Statement object.

For example, `stmt = con.createStatement();`

There are three different kinds of statements:

- **Statement:** Used to implement simple SQL statements with no parameters.
- **PreparedStatement:** Used for pre-compiling SQL statements that might contain input parameters.
- **CallableStatement:** Used to execute stored procedures that may contain both input and output parameters.

Step 3: Execute the Query: To execute a query, call an execute method from Statement such as the following:

- **execute:** Returns true if the first object that the query returns is a ResultSet object. Use this method if the query could return one or more ResultSet objects. Retrieve the ResultSet objects returned from the query by repeatedly calling Statement.getResultSet.
- **executeQuery:** Returns one ResultSet object.
- **executeUpdate:** Returns an integer representing the number of rows affected by the SQL statement. Use this method if we are using INSERT, DELETE, or UPDATE SQL statements.

For example, `ResultSet rs = stmt.executeQuery(query);`

Step 4: Process the ResultSet Object: We access the data in a ResultSet object through a cursor. This cursor is a pointer that points to one row of data in the ResultSet object. Initially, the cursor is positioned before the first row. We call various methods such as next (to move the cursor forward by one row) defined in the ResultSet object to move the cursor.

Step 5: Close the Connection: When we are finished using a Connection, Statement or ResultSet object, call its close() method to close connection.

3.5

RESULTSET (SCROLLABLE AND UPDATABLE)

[April 16, 17, Oct. 17]

- 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 stored somewhere. That result is 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 ResultSet 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.
- To get the type of ResultSet we can use the getType() and getConcurrency() methods. The getType() method returns one of three types:
 1. **TYPE_FORWARD_ONLY:** The result set is the read-only version that enables you to move in forward direction only.
 2. **TYPE_SCROLL_SENSITIVE:** This ResultSet is Scrollable means you can move in any direction and directly access any row directly. Also this ResultSet is dynamic in nature means if any changes are made while ResultSet is open the values are reflected in ResultSet.
 3. **TYPE_SCROLL_INSENSITIVE:** This ResultSet is Scrollable means you can move in any direction and directly access any row directly. But when others make changes to the underlying data source, they will not change the order of values you see in the return results.
- Several methods of the ResultSet interface are given below:
 1. `public void beforeFirst() throws SQLException` method moves the cursor just before the first row.

2. `public void afterLast()` throws `SQLException` method moves the cursor just after the last row.
3. `public boolean first()` throws `SQLException` method moves the cursor to the first row.
4. `public void last()` throws `SQLException` method moves the cursor to the last row.
5. `public boolean absolute(int row)` throws `SQLException` method moves the cursor to the specified row.
6. `public boolean relative(int row)` throws `SQLException` method moves the cursor the given number of rows forward or backward, from where it is currently pointing.
7. `public boolean previous()` throws `SQLException` method moves the cursor to the previous row. This method returns false if the previous row is off the result set.
8. `public boolean next()` throws `SQLException` method moves the cursor to the next row. This method returns false if there are no more rows in the result set.
9. `public int getRow()` throws `SQLException` method returns the row number that the cursor is pointing to.
10. `public void moveToInsertRow()` throws `SQLException` method moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered.
11. `public void moveToCurrentRow()` throws `SQLException` method moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing.
12. `public int getInt(String columnName)` throws `SQLException` method returns the int in the current row in the column named `columnName`.
13. `public int getInt(int columnIndex)` throws `SQLException` method returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.
14. `public void updateString(int columnIndex, String s)` throws `SQLException` method changes the String in the specified column to the value of `s`.
15. `public void updateString(String columnName, String s)` throws `SQLException` method is similar to the previous method, except that the column is specified by its name instead of its index.
16. `public void deleteRow()` method deletes the current row from the database
17. `public void refreshRow()` method refreshes the data in the result set to reflect any recent changes in the database.
18. `public void cancelRowUpdates()` method cancels any updates made on the current row.
19. `public void insertRow()` method inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

3.6 METADATA**[April 16, 18]**

- JDBC MetaData is the collective information about the data structure and property of a column available in table.
- The metadata of any table tells us the name of the columns, data type used in column and constraint used to enter the value of data into column of the table.
- With the following interfaces we can retrieve some basic Metadata:

DatabaseMetaData:**[April 16]**

- Applications sometimes 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.
- The DatabaseMetaData object has many methods and properties that provide a lot of information about a database.

Creating DatabaseMetaData Objects:

- A Connection object represents a database connection and also instantiates a DatabaseMetaData object with the getMetaData() method.
- The DatabaseMetaData object holds information for the database to which the Connection object is connected.
- The following code shows how to create a DatabaseMetaData object:

```
DatabaseMetaData dmd = conn.getMetaData();
```

Methods of DatabaseMetaData:

1. `getTables()` method retrieves information about the tables.
2. `getColumns()` method retrieves information about table columns in a database.
3. `getTypeInfo()` method determines what data types your target database supports.
4. `getPrimaryKeys()` and `getImportedKeys()` methods provide the primary and foreign-key information.
5. `getProcedures()` and `getProcedureColumns()` methods provides information about the stored procedures in a database.

ResultSetMetaData:**[April 16]**

- 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.

Creating ResultSetMetaData Objects:

- Object or ResultSetMetaData can be created same as that of DatabaseMetaData, using getMetaData() method.

```
ResultSetMetaData rsmd = rs.getMetaData();
```

Methods of ResultSetMetaData:

1. `getTableName()` method returns the name of the table you queried.
2. `getColumnTypeName(int n)` method provides the name of the column's data type as defined by the database. The parameter n is the column position within the result set.
3. `getColumnType(int n)` method returns the JDBC data type for a column at position n of the result set.
4. `getColumnName(int n)` method returns the name of a column at position n in the result set.
5. `getColumnCount()` method returns the number of columns in the result set.
Returns an int.

ParameterMetaData:

- An object that can be used to get information about the types and properties for each parameter marker in a PreparedStatement object.
- For some queries and driver implementations, the data that would be returned by a ParameterMetaData object may not be available until the PreparedStatement has been executed.

RowSetMetaData:

- An object that contains information about the columns in a RowSet object. This interface is an extension of the ResultSetMetaData interface with methods for setting the values in a RowSetMetaData object.
- When a RowSetReader object reads data into a RowSet object, it creates a RowSetMetaData object and initializes it using the methods in the RowSetMetaData interface. Then the reader passes the RowSetMetaData object to the rowset.

3.7 TRANSACTIONS

[April 16, 19]

- Transaction is a grouping of work that an application wants to happen all together, or not at all.
- Transaction includes multiple updates to data, either in same database or in different database.
- If we have an application that needs to execute multiple SQL statements to fulfill one goal, suppose in ERP system we want to calculate employee's salary by passing his Eid, we probably want to use JDBC's transaction services to accomplish the goal.
- When the transaction is committed, all the updates are undone. This allows data to be kept in a consistent state.
- Working with a transaction involves the following steps:
Step 1: Start the transaction, perform require operations,
Step 2: Either commit the transaction if all the operations succeed, or
Step 3: ROLL it back if one of the operations fails.

Rollback() and Commit():

[April 16, 19, Oct. 17]

- Rollback a transaction is the key feature of a transaction. Instead of successfully reflecting changes in database with commit we can also undo performed work on database which is referred as "rollback".
- The syntax is:

```
connection.setAutoCommit(false);
connection.rollback();
```
- The Connection object in JDBC is responsible for transaction management. A new connection starts out in transaction auto-commit mode, which means that every SQL statement is executed as an individual transaction that is immediately committed to the database.
- To perform a transaction that uses multiple statements, you have to call the setAutoCommit() method with a false argument.
- The commit() method is applied to the database only when AutoCommit is set to false. If it is set to true, then the commit is implicitly applied.

Syntax: `connection.setAutoCommit(false);`

Example:

```
try {
    con.setAutoCommit(false);
    stmt.executeUpdate("UPDATE EMP SET Empbasic = 9000 WHERE Eid = 5");
    con.commit(); }
    catch (SQLException e) {
        con.rollback();}
```

- If auto-commit is set to false, you must call commit() or rollback() explicitly at the end of each transaction, otherwise your changes will be lost.

Savepoint:

- Once, user perform rollback operation all the operation he/she has performed may be lost. With the help of Save point user can save his changes even after rollback. So for user need to create save point of his operations.
- We can use setSavepoint() method to create save point and releaseSavepoint() method to cancel created save point.

Example:

```

con.setAutoCommit(false);
stmt.executeUpdate("UPDATE EMP SET Empbasic = 9000 WHERE Eid = 5");
Savepoint sp=con.setSavepoint("Savepoint");
stmt.executeUpdate("UPDATE EMP SET Empbasic = 12000 WHERE Eid = 6");
stmt.executeUpdate("UPDATE EMP SET Empbasic = 1200 WHERE Eid = 7");
if(Empbasic<2000)
{
    con.rollback(sp);
}
con.commit();
con.realiseSavepoint(sp);

```

Batch Updates:

- The main problem with original JDBC is if we try to load large amount of data into database, the performance of system decreases. By using batch update all of your queries are treated as a single command and executed as a batch.
- We can use addBatch() method of Statementwith which can execute number of update queries at a time. It can be used in following manner:

```

con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO Employee VALUES
                (1, "AMAR", "Manager", 10000, "IT");
stmt.addBatch("INSERT INTO Employee VALUES
                (2, "Swami", "Supervisor", 7000, "Workshop");
stmt.addBatch("INSERT INTO Employee VALUES
                (3, "Kiran", "loader ", 5200, "Workshop");
int[] upCounts = stmt.executeBatch();
con.commit();

```

- Note that we can use only SQL statements that return an update count (e.g., CREATE, DROP, INSERT, UPDATE, DELETE) as part of a batch.

- If we include a statement that returns a result set, such as SELECT, you get a SQLException when you execute the batch.
- If one of the statements in a batch cannot be executed for some reason, executeBatch() throws a BatchUpdateException. Where as if we use PreparedStatement or CallableStatement instead of Statement:

```

con.setAutoCommit(false);
PreparedStatement stmt = con.prepareStatement(
    "INSERT INTO Employee VALUES (?,?,?,?,?,?)");
stmt.setInt(1,1);
stmt.setString(2, "AMAR");
stmt.setString(3, "Manager");
stmt.setString(4, 10000);
stmt.setString(5, "IT");
stmt.addBatch();
int[] upCounts = stmt.executeBatch();
con.commit();

```

ADDITIONAL PROGRAMS

Program 1: JDBC program to create a STUDENT(STUDNO, STUDNAME, STUDPER) table, fetch the record and display the all in the record of students.

```

import java.sql.*;      //Step 1 : import package
import java.io.*;
class jdbcfirst
{
    public static void main(String a1[])
    {
        try
        {
            Class.forName("org.postgresql.Driver"); //Step 2 : Load Driver
            System.out.println("Driver Loaded...");
            Connection con =
                DriverManager.getConnection("jdbc:postgresql:tybov","postgres","");
                // Step 3 create connectin
            System.out.println("Connection Established...");
            Statement st = con.createStatement();
                // Step 4 create statement object

```

```

        System.out.println("Create statement...");
        ResultSet rs = st.executeQuery("select * from student");
                                //Step 5 : create resultset object
        System.out.println("Retrieved data");
        System.out.println("Student no \t student Name \t Student Per");
        while(rs.next())           //Step 6 Retrive Data
        {
            System.out.print(" "+rs.getInt(1));
            System.out.print("\t\t"+rs.getString(2));
            System.out.print("\t\t"+rs.getFloat(3));
            System.out.println();
        }//while
        con.close();      //Step 7 close the connection
    }//try
    catch(Exception e)
    {
        System.out.println("Error"+e);
    }
}//main
}//end of class

```

Output:

```

Driver Loaded...
Connection Established...
Create statement...
Retrieved data
Student no    student Name    Student Per
 11          sumit          95.0
 21          shubham         75.0
 51          Niki            77.0
 41          Akankshya       79.0
 90          MMM             87.0
 19          KKK             67.0

```

Program 2: Program to insert data into table student (using command Line) and search a record and display the information of students.

```

import java.sql.*;
import java.io.*;

```

```
class Jdbcins
{
    public static void main(String a[])
    {
        try
        {
            int sno = Integer.parseInt(a[0]);
            String sname = a[1];
            Float sper = Float.parseFloat(a[2]);
            int no = Integer.parseInt(a[3]);
            System.out.println("Student No = "+sno);
            System.out.println("Student Name = "+sname);
            System.out.println("Student Percentage = "+sper);
            Class.forName("org.postgresql.Driver");
            System.out.println("Driver Loaded ...");
            Connection con = DriverManager.getConnection
                ("jdbc:postgresql:tybov","postgres","");
            System.out.println("Connection Established");
            PreparedStatement ps = con.prepareStatement
                ("insert into student values(?, ?, ?)");
            ps.setInt(1,sno);
            ps.setString(2,sname);
            ps.setFloat(3,sper);
            int i = ps.executeUpdate();
            System.out.println("Value =" +i);
            System.out.println("Insert Data Succesfully ...");
            Statement st = con.createStatement();
            System.out.println("Statement Created...");
            ResultSet rs = st.executeQuery("select * from student");
            System.out.println("Student No \t Student Name \t Student Per ");
            while(rs.next())
            {
                System.out.print(""+rs.getInt(1));
                System.out.print("\t\t"+rs.getString(2));
                System.out.print("\t\t"+rs.getFloat(3));
                System.out.println();
            } //while
```

```
//searching using Constant
Statement st1 = con.createStatement();
System.out.println("Statement Created...");
ResultSet rs1 = st.executeQuery("select * from student where sno=7");
System.out.println("FOUND...");
System.out.println("Student No \t Student Name \t Student Per ");
while(rs1.next())
{
    System.out.print(""+rs1.getInt(1));
    System.out.print("\t\t"+rs1.getString(2));
    System.out.print("\t\t"+rs1.getFloat(3));
    System.out.println();
}
//while
//searching using command line Input
Statement st2 = con.createStatement();
System.out.println("Statement Created...");
ResultSet rs2 = st.executeQuery("select * from student
                                where sno="+ "+no+"");
System.out.println("FOUND...");
System.out.println("Student No \t Student Name \t Student Per ");
while(rs2.next())
{
    System.out.print(""+rs2.getInt(1));
    System.out.print("\t\t"+rs2.getString(2));
    System.out.print("\t\t"+rs2.getFloat(3));
    System.out.println();
}
//while
}//try
catch(Exception e)
{
    System.out.println("Error "+e);
}
//catch
}//end of main
} //end of class
```

Output:

```

20 Nirali 88 19
Student No = 20
Student Name = Nirali
Student Percentage = 88.0
Driver Loaded ...
Connection Established
Value =1
Insert Data Succesfully ...
Statement Created...
Student No      Student Name      Student Per
1               sumit            95.0
2               shubham          75.0
5               Nikita           77.0
4               Akankshya        79.0
7               Rudra            89.0
8               Atharva          90.0
9               MMM              87.0
19              KKK              77.0
20              Nirali           88.0
Statement Created...
FOUND...
Student No      Student Name      Student Per
7               Rudra            89.0
Statement Created...
FOUND...
Student No      Student Name      Student Per
19              KKK              77.0

```

Program 3: A JDBC program to create a EMP (EMPNO,EMPNAME,EMPSALARY)table get the information from user and insert into EMPLOYEE table and search the record by EMPNO and EMPNAME and display the information of EMPLOYEE.

```

import java.sql.*;
import java.io.*;
class Jdbcemp
{
    public static void main(String a[])

```

```
{  
    try  
    {  
        int eno,esal,lim;  
        String ename;  
        Class.forName("org.postgresql.Driver");  
        System.out.println("Driver Loaded ...");  
        Connection con = DriverManager.getConnection  
            ("jdbc:postgresql:ty2021","postgres","");
        System.out.println("Connection Established...");  
        BufferedReader br = new BufferedReader  
            (new InputStreamReader(System.in));  
        System.out.println("How many record want to be insert ?");  
        lim = Integer.parseInt(br.readLine());  
        for(int i=0;i<lim;i++)  
        {  
            System.out.println("Enter Employee No =");  
            eno = Integer.parseInt(br.readLine());  
            System.out.println("Enter Employee Name =");  
            ename = br.readLine();  
            System.out.println("Enter Employee Salary =");  
            esal = Integer.parseInt(br.readLine());  
            //insert  
            PreparedStatement ps = con.prepareStatement  
                ("insert into emp values(?, ?, ?)");  
            ps.setInt(1,eno);  
            ps.setString(2,ename);  
            ps.setInt(3,esal);  
            int j = ps.executeUpdate();  
            System.out.println("j = "+j);  
        }//for */  
        //Display (select )  
        Statement st = con.createStatement();  
        System.out.println("Statement Created...");  
        ResultSet rs = st.executeQuery("select * from emp");  
        System.out.println("*** Employee Record *** \n");  
        System.out.println("Emp No \t Emp Name \t Emp Salary");

---


```

```
while(rs.next())
{
    System.out.print(" "+rs.getInt(1));
    System.out.print("\t"+rs.getString(2));
    System.out.print("\t\t "+rs.getInt(3));
    System.out.println();
}
//Search by Employee Number
System.out.println("Enter Employee No to be Search =");
int no = Integer.parseInt(br.readLine());
Statement st1 = con.createStatement();
System.out.println("Statement Created...");
ResultSet rs1 = st1.executeQuery
                ("select * from emp where eno = "+no+"");
System.out.println("*** Employee Record *** \n");
System.out.println("Emp No \t Emp Name \t\t Emp Salary");
while(rs1.next())
{
    System.out.print(" "+rs1.getInt(1));
    System.out.print("\t"+rs1.getString(2));
    System.out.print("\t\t "+rs1.getInt(3));
    System.out.println();
}
//Search by Employee Name
System.out.println("Enter Employee Name to be Search =");
String name = br.readLine();
Statement st2 = con.createStatement();
System.out.println("Statement Created...");
ResultSet rs2 = st2.executeQuery
                ("select * from emp where ename = '"+name+"'");
System.out.println("*** Employee Record *** \n");
System.out.println("Emp No \t Emp Name \t\t Emp Salary");
while(rs2.next())
{
    System.out.print(" "+rs2.getInt(1));
    System.out.print("\t"+rs2.getString(2));
```

```
        System.out.print("\t\t "+rs2.getInt(3));
        System.out.println();
    }
}//try
catch(Exception e)
{
    System.out.println("Error "+e);
}
}
}
```

Output:

```
Driver Loaded ...
Connection Established...
How many record want to be insert ?
3
Enter Employee No =
101
Enter Employee Name =
Jiva
Enter Employee Salary =
65000
j = 1
Enter Employee No =
102
Enter Employee Name =
Anandi
Enter Employee Salary =
20000
j = 1
Enter Employee No =
103
Enter Employee Name =
Yash
Enter Employee Salary =
15000
j = 1
```

```

Statement Created...
*** Employee Record ***
Emp No    Emp Name    Emp Salary
101        Jiva         65000
102        Anandi       20000
103        Yash         15000
Enter Employee No to be Search =
102
Statement Created...
*** Employee Record ***
Emp No    Emp Name    Emp Salary
102        Anandi       20000
Enter Employee Name to be Search =
Jiva
Statement Created...
*** Employee Record ***
Emp No    Emp Name    Emp Salary
101        Jiva         65000

```

Program 4: Program to create the table student with the fields roll number ,name, percentage using Postgresql. Write a menu driven program to perform the following operations on student table.

- (a) Insert
- (b) Modify
- (c) Delete
- (d) Search
- (e) View All
- (f) Exit

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.*;
public class Jdbcstudent
{
    public static void main(String[] args)
    {
        int choice,choice1;
        Connection con=null;

```

```
Statement sm=null;
ResultSet rs=null;
BufferedReader br=new BufferedReader
                (new InputStreamReader(System.in));
try
{
    Class.forName("org.postgresql.Driver");
    con= DriverManager.getConnection
                ("jdbc:postgresql:ty2021","postgres","");
    sm=con.createStatement();
    do
    {
        System.out.println("1.Insert");
        System.out.println("2.Modify");
        System.out.println("3.Delete");
        System.out.println("4.Search");
        System.out.println("5.View All");
        System.out.println("6.Exit");
        System.out.println("Enter Your Choice:\t");
        choice=Integer.parseInt(br.readLine());
        switch (choice)
        {
            case 1:
                System.out.println("Enter Roll No:\t");
                int r=Integer.parseInt(br.readLine());
                System.out.println("Enter Name:\t");
                String name=br.readLine();
                System.out.println("Enter Percentage:\t");
                float
                    f=Float.parseFloat(br.readLine());
                int
                    result=sm.executeUpdate("insert into
                        Student values("+r+","+name+"','"+f+");");
                if(result>0)System.out.println("Insert Successfully");
                else
                    System.out.println("error");
                break;
        }
    }
}
```

```
        case 2:  
            System.out.println("Enter Roll No:\t");  
            r=Integer.parseInt(br.readLine());  
            System.out.println("1.Name");  
  
            System.out.println("2.Percentage");  
            System.out.println("Enter Your Choice:\t");  
            choice1=Integer.parseInt(br.readLine());  
            switch (choice1)  
            {  
                case 1:  
                    System.out.println("Enter Name:\t");  
                    name=br.readLine();  
                    result=sm.executeUpdate("update Student set  
                    sname='"+name+"' where sno="+r);  
                    if(result>0)  
                        System.out.println("Update Successfully");  
                    else  
                        System.out.println("error");  
                    break;  
                case 2:  
                    System.out.println("Enter Percentage:\t");  
                    f=Float.parseFloat(br.readLine());  
                    result=sm.executeUpdate("update Student  
                    set per="+f+" where sno="+r);  
                    if(result>0)  
                        System.out.println("Update Successfully");  
                    else  
                        System.out.println("error");  
                    break;  
            }  
            break;  
        case 3:  
            System.out.println("Enter Roll No:\t");  
            r=Integer.parseInt(br.readLine());  
            result=sm.executeUpdate("delete from  
            Student where sno="+r+";");
```

```
        if(result>0)
            System.out.println("Deleted Successfully");
        else
            System.out.println("error");
        break;
    case 4:
        System.out.println("Enter Roll No:\t");
        r=Integer.parseInt(br.readLine());
        rs=sm.executeQuery
            ("select * from Student where sno="+r+";");
        ResultSetMetaData rsmd=rs.getMetaData();
        int col=rsmd.getColumnCount();
        for(int i=1;i<=col;i++)
            System.out.print(rsmd.getColumnLabel(i) +"\t");
        System.out.println();
        while(rs.next())
        {
            for(int i=1;i<=col;i++)
                System.out.print(rs.getString(i)+"\t");
            System.out.println();
        }
        break;
    case 5:
        rs=sm.executeQuery("select * from Student;");
        rsmd=rs.getMetaData();
        col=rsmd.getColumnCount();

        for(int i=1;i<=col;i++)
            System.out.print(rsmd.getColumnLabel(i)+"\t");
        System.out.println();
        while(rs.next())
        {
            for(int i=1;i<=col;i++)
                System.out.print(rs.getString(i)+"\t");
            System.out.println();
        }
        break;
```

```
        case 6:System.exit(0);
    }
}while(true);
}
catch(Exception e)
{
    System.out.println(e);
}
}
}
```

Output:

1.Insert

2.Modify

3.Delete

4.Search

5.View All

6.Exit

Enter Your Choice:

1

Enter Roll No:

1

Enter Name:

AAA

Enter Percentage:

56

Insert Successfully

1.Insert

2.Modify

3.Delete

4.Search

5.View All

6.Exit

Enter Your Choice:

1

Enter Roll No:

2

```
Enter Name:  
BBB  
Enter Percentage:  
89  
Insert Successfully  
1.Insert  
2.Modify  
3.Delete  
4.Search  
5.View All  
6.Exit  
Enter Your Choice:  
1  
Enter Roll No:  
3  
Enter Name:  
ZZZ  
Enter Percentage:  
92  
Insert Successfully  
1.Insert  
2.Modify  
3.Delete  
4.Search  
5.View All  
6.Exit  
Enter Your Choice:  
5  
sno      sname      sper  
1        AAA        56  
2        BBB        89  
3        ZZZ        92  
1.Insert  
2.Modify  
3.Delete  
4.Search  
5.View All  
6.Exit
```

```
3
Enter Roll No:
1
Deleted Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
5
sno      sname      sper
2        BBB        89
3        ZZZ        92
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
1
Enter Roll No:
1
Enter Name:
AAA
Enter Percentage:
75
Insert Successfully
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
```

Enter Your Choice:

5

| sno | sname | sper |
|-----|-------|------|
| 2 | BBB | 89 |
| 3 | ZZZ | 92 |
| 1 | AAA | 75 |

1.Insert

2.Modify

3.Delete

4.Search

5.View All

6.Exit

Enter Your Choice:

2

Enter Roll No:

1

1.Name

2.Percentage

Enter Your Choice:

1

Enter Name:

XXX

Update Successfully

1.Insert

2.Modify

3.Delete

4.Search

5.View All

6.Exit

Enter Your Choice:

5

| sno | sname | sper |
|-----|-------|------|
| 2 | BBB | 89 |
| 3 | ZZZ | 92 |
| 1 | XXX | 75 |

```

1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
4
Enter Roll No:
3
sno      sname      sper
3        ZZZ        92
1.Insert
2.Modify
3.Delete
4.Search
5.View All
6.Exit
Enter Your Choice:
6

```

Program 5: Design a table admin(username, password) using postgresql. Also design an HTML login screen accepting a login name and password from the user. Write a JDBC program that validates accepted login or user name and password entered by user from the login table you have created and sends an appropriate response.

```

import java.sql.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;
import java.util.*;
public class login
{
    Connection con=null;
    ResultSet rs=null;
    Statement stmt=null;
    String username=null,password=null;

```

```
JTextField tf1;
JPasswordField tf2;
JButton b1,b2,b3;
JFrame f;
boolean status=true;
public login()
{
    f= new JFrame("Login Sreen");
    f.setLayout(null);
    JPanel panel = new JPanel();
    JLabel l1=new JLabel("UserName");
    l1.setBounds(50,50,100,25);
    panel.add(l1);
    f.setLocation(400,150);
    tf1=new JTextField();
    tf1.setBounds(175,50,100,25);
    panel.add(tf1);
    JLabel l2=new JLabel("Password");
    l2.setBounds(50,125,100,25);
    panel.add(l2);
    tf2=new JPasswordField();
    tf2.setBounds(175,125,100,25);
    panel.add(tf2);
    b1=new JButton("LOGIN");
    b1.setBounds(25,275,90,25);
    panel.add(b1);
    b2=new JButton("CLEAR");
    b2.setBounds(125,275,90,25);
    panel.add(b2);
    b3=new JButton("CANCEL");
    b3.setBounds(225,275,90,25);
    panel.add(b3);
    b1.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent ae)
        {
```

```
        b1actionperformed(ae);
    }
});
b2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ae)
    {
        b2actionperformed(ae);
    }
});
b3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ae)
    {
        b3actionperformed(ae);
    }
});
panel.setLayout(null);
panel.setBounds(10,10,440,440);
panel.setBorder(new LineBorder(Color.green,2));
panel.setBackground(new
java.awt.Color(255,219,251));
panel.setVisible(true);
f.add(panel);
f.setSize(480,500);
f.setVisible(true);
}//login
public static void main(String []venkat)
{
    new login();
}
//}
public void b1actionperformed(ActionEvent ae)
{
    try
    {
        Class.forName("org.postgresql.Driver");
    }
```

```
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }
        if(tf1.getText().equals("")){
        {
            JOptionPane.showMessageDialog(null,"Please fill data in all the
fields","Error Message",JOptionPane.OK_OPTION);
        }
        try
        {
            con=DriverManager.getConnection("jdbc:postgresql:ty2021",
                                              "postgres"," ");
            System.out.println("Connection Established!!!");
            stmt=con.createStatement();
            rs=stmt.executeQuery("select * from admin");
            while(rs.next())
            {
                if(tf1.getText().equals(rs.getString(1))
                   && tf2.getText().equals(rs.getString(2)))
                {
                    JOptionPane.showMessageDialog(null,"login sucessfully");
                    //new Gui1();
                    f.setVisible(false);
                }
                else
                {
                    JOptionPane.showMessageDialog(null,"incorrect username
or password","error Message",JOptionPane.OK_OPTION);
                }
            }//while
        }//try
        catch (Exception e)
        {
            e.printStackTrace();
        }
        clear();
    }
```

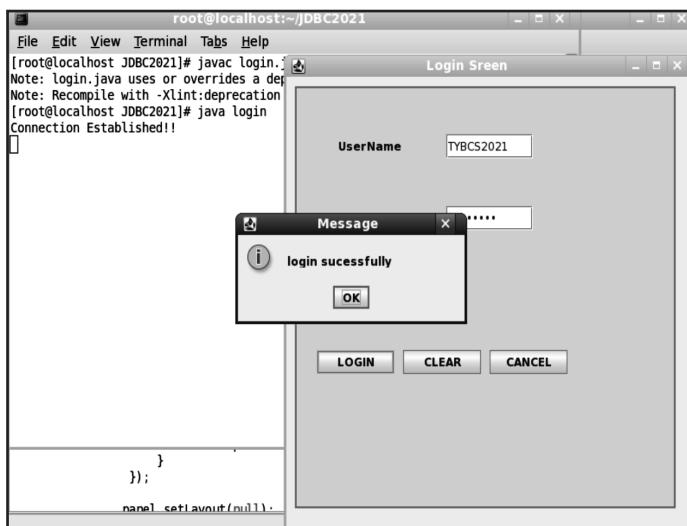
```

public void b2actionperformed(ActionEvent ae)
{
    clear();
}
public void b3actionperformed(ActionEvent ae)
{
    System.exit(0);
}
void clear()
{
    tf1.setText(null);
    tf2.setText(null);
}
}

```

Output:

Connection Established!!



PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. JDBC stands for,

| | |
|----------------------------------|------------------------------|
| (a) Java DataBase Connectivity | (b) Java DataBase Connection |
| (c) Java DataBase Communications | (d) None of the mentioned |

2. Which method is used to establish the connection with the specified url in a Driver Manager class?
 - (a) public static void registerDriver(Driver driver)
 - (b) public static void deregisterDriver(Driver driver)
 - (c) public static Connection getConnection(String url)
 - (d) public static Connection getConnection(String userName, String password)
 3. Which JDBC product components does the Java software provide?

| | |
|-----------------------------|--------------------------------|
| (a) The JDBC driver manager | (b) The JDBC driver test suite |
| (c) The JDBC-ODBC bridge | (d) All of the mentioned |
 4. Which result set generally does not show changes to the underlying database that are made while it is open. The membership, order, and column values of rows are typically fixed when the result set is created?

| | |
|---------------------------|-----------------------------|
| (a) TYPE_FORWARD_ONLY | (b) TYPE_SCROLL_INSENSITIVE |
| (c) TYPE_SCROLL_SENSITIVE | (d) All of the mentioned |
 5. Which of the following JDBC drivers is known as a partially java driver?

| | |
|-----------------------------|-----------------------|
| (a) JDBC-ODBC bridge driver | (b) Native-API driver |
| (c) Network Protocol driver | (d) Thin driver |
 6. In the following JDBC drivers which is known as fully Java driver.

| | |
|-----------------------|-----------------------------|
| (a) Native-API driver | (b) Network Protocol driver |
| (c) Thin driver | (d) Both (b) and (c) |
 7. Which class has traditionally been the backbone of the JDBC architecture?

| | |
|-------------------------|----------------------------|
| (a) JDBC driver manager | (b) JDBC driver test suite |
| (c) JDBC-ODBC bridge | (d) All of the mentioned |
 8. Which of the following contains both date and time?

| | |
|--------------------|------------------------|
| (a) java.io.date | (b) java.sql.date |
| (c) java.util.date | (d) java.util.dateTime |
 9. Which of the following is advantage of using PreparedStatement in Java?

| | |
|----------------------------|------------------------------|
| (a) Slow performance | (b) Encourages SQL injection |
| (c) Prevents SQL injection | (d) More memory usage |
 10. Which kind of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, IBM DB2 or other DBMS?

| | |
|-----------------------------------|---------------------------------------|
| (a) Native-API partly-Java driver | (b) JDBC-ODBC bridge plus ODBC driver |
| (c) JDBC-Net pure Java driver | (d) Native-protocol pure Java driver |
 11. JDBC is an Application Programming Interface (API) for the programming language,

| | |
|------------|----------|
| (a) Python | (b) C++ |
| (c) PHP | (d) Java |
-

12. Which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases?
- (a) JDBC (b) AODBC
(c) AJDBC (d) None of the mentioned
13. Which interface represents the result set of a database query?
- (a) ResultObject (b) ResultQuery
(c) ResultSet (d) All of the mentioned
14. Which is a unit of work that is performed against a database?
- (a) thread (b) transaction
(c) process (d) None of the mentioned
15. Which class handles any errors that occur in a database application?
- (a) SQLException (b) SQLError
(c) SQLExceptionError (d) None of the mentioned
16. Which command is the transactional command used to save changes invoked by a transaction to the database?
- (a) SAVEPOINT (b) COMMIT
(c) ROLLBACK (d) None of the mentioned

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|--------|--------|--------|---------|
| 1. (a) | 2. (c) | 3. (d) | 4. (b) | 5. (b) | 6. (d) | 7. (a) | 8. (d) | 9. (c) | 10. (a) |
| 11. (d) | 12. (a) | 13. (c) | 14. (b) | 15. (a) | 16. (b) | | | | |

Q. II Fill in the Blanks:

1. A _____ is a collection of information organized to provide efficient retrieval while a DataBase Management System (DBMS) enables to retrieval information on database.
 2. _____ is the return type of executeUpdate() method.
 3. _____ method is used to rollback a JDBC transaction
 4. A _____ represents a point that the current transaction can roll back to.
 5. _____ is the return type of execute() method.
 6. _____ is efficient than a statement due to the pre-compilation of SQL.
 7. _____ always maintains a cursor that points to a row in the database table.
 8. _____ method is used to execute a query for any type of SQL statements.
 9. _____ represents a single unit of work on a database.
 10. _____ interface with all methods for contacting a database.
 11. JDBC _____ is a software component that enables java application to interact with the database.
 12. JDBC _____ is contained in two packages namely, java.sql and javax.sql.
-

13. The `java.sql.Statement` and `java.sql.PreparedStatement` interfaces provide methods for _____ processing.
14. _____ class manages a list of database drivers.
15. We can use _____ method to create save point.

Answers

| | | | |
|----------------|----------------------|--------------------|--------------|
| 1. database | 2. integer | 3. rollback() | 4. savepoint |
| 5. Boolean | 6. PreparedStatement | 7. ResultSet | 8. execute() |
| 9. Transaction | 10. Connection | 11. Driver | 12. API |
| 13. batch | 14. DriverManager | 15. setSavepoint() | |

Q. III State True or False:

1. JDBC Type 2 drivers connect to the native API of the DBMS.
2. The performance of the application will be faster if you use `PreparedStatement` interface because query is compiled only once.
3. JDBC is a Java API that is used to connect and execute query to the database.
4. Drivers that are JDBC Compliant should normally support scrollable result sets, but they are not required to do so.
5. The `ResultSet.next` method is used to move to the next row of the `ResultSet`, making it the current row.
6. `ResultSet` object can be moved forward only and it is updatable.
7. The JDBC API is what allows access to a data source from a Java middle tier.
8. JDBC RowSet is the wrapper of `ResultSet`, It holds tabular data like `ResultSet` but it is easy and flexible to use.
9. The JDBC API has always supported persistent storage of objects defined in the Java programming language through the methods `getObject` and `setObject`.
10. The intent is for JDBC drivers to implement nonscrollable result sets using the support provided by the underlying database systems.
11. JDBC stands for Java Database Connectivity.
12. By using batch update all of your queries are treated as a single command and executed as a batch.
13. A `ResultSet` object maintains a cursor that points to the current row in the result set, in simple words, it refers to the row and column data contained in a `ResultSet` object.
14. The `getMetaData()` method of `ResultSet` interface returns the object of `ResultSetMetaData`.
15. The `java.sql` package contains classes and interfaces for ODBC API.

16. CallableStatement interface Use this when you want to access the database stored procedures.
17. The Statement interface provides methods to execute queries with the database.
18. The metadata means data about data i.e. we can get further information from the data.
19. Driver interface handles the communications with the database server.
20. We can use objects created from Statement interface to submit the SQL statements to the database.

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 1. (T) | 2. (T) | 3. (T) | 4. (T) | 5. (T) | 6. (F) | 7. (T) | 8. (T) | 9. (T) | 10. (F) |
| 11. (T) | 12. (T) | 13. (T) | 14. (T) | 15. (F) | 16. (T) | 17. (T) | 18. (T) | 19. (T) | 20. (T) |

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define database and DBMS.
2. What is JDBC?
3. Define JDBC driver.
4. What is the advantage of using PreparedStatement over Statement?
5. Which class manages list of database drivers.
6. What is the role of Connection interface.
7. What is ResultSet?
8. Define transaction in database.
9. What is the use of callable statement?
10. When to use executeUpdate()?
11. Name the class responsible for selecting database drivers and creating a new database connection.
12. Define metadata.
13. What is the use of SetAutoCommit()?
14. List any two methods for creating transaction.
15. What is the purpose of PreparedStatement?
16. List the types of JDBC drivers.

(B) Long Answer Questions:

1. With the help of diagram describe working of JDBC. Also explain JDBC architecture.
2. What are the components of JDBC? Explain with them with their role.
3. Write a short note on: Design of JDBC.
4. What is JDBC driver? List its types. Explain with diagram.

5. With the help of example describe how to connect database in Java using JDBC.
6. How to executing SQL statements? Describe in detail.
7. What is ResultSet? How it can be used in JDBC? Explain with example.
8. What is metadata? Explain in detail.
9. What is savepoint? What is its use? How to create it? Explain with example.
10. Write a short note on: Scrollable and Updatable ResultSet.
11. What is batch updates? How it works? Explain with example.
12. State the difference between statement, callable statement and prepared statement interface.
13. Write a JDBC program to display all details of the student table (roll, name, marks).
Also perform insertion and updation operation on student database.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. Write the Java method that creates scrollable and Updatable ResultSet. **[1 M]**
- Ans.** Refer to Section 3.5.
2. What is the use of SetAutoCommit()? **[1 M]**
- Ans.** Refer to Section 3.7.
3. What is metadata? How do you create database and ResultSet metadata? Explain in detail. **[5 M]**
- Ans.** Refer to Section 3.6.
4. Write a JDBC program to insert student details in student table (roll, name, marks) and display all details. **[5 M]**
- Ans.** Refer to Additional Programs.

April 2017

1. What are four types of JDBC drives? **[1 M]**
- Ans.** Refer to Section 3.2.
2. When to use PreparedStatement? **[1 M]**
- Ans.** Refer to Section 3.2, Point (2).
3. Write a JDBC program to insert following 'n' records to employee table having structure emp_no, emp_name and salary and display it. **[5 M]**
- Ans.** Refer to Additional Programs.
4. What is ResultSet? Explain scrollable and updatable resultsets. **[5 M]**
- Ans.** Refer to Section 3.5.

October 2017

1. Which packages are required of Java database connectivity? **[1 M]**
- Ans.** Refer to 3.2.1 Last Point.

2. What is purpose of rollback() in transactions in Java database connectivity? [1 M]

Ans. Refer to Section 3.7.

3. What is use of the following in JDBC?

- (i) execute()
- (ii) executeUpdate()
- (iii) executeQuery()
- (iv) statement()
- (v) preparedStatement().

[5 M]

Ans. Refer to Section 3.3.

5. 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 U 10 'oxygen' 137 will update the record according to the name specified. [5 M]

Ans. Refer to Additional Programs.

April 2018

1. When to use execute update (string sql) method? [1 M]

Ans. Refer to Section 3.3.

2. Name the statement types used for executing SQL queries. [1 M]

Ans. Refer to Section 3.3.

3. Write a JDBC program to display information about table such as column labels, number of columns and column type (Table name is stud and Database name is Exam). [5 M]

Ans. Refer to Additional Programs.

4. Explain JDBC architecture in detail. [5 M]

Ans. Refer to Section 3.0.

October 2018

1. State the advantage of using type 3 JDBC driver. [1 M]

Ans. Refer to Section 3.2.1.

2. Explain JDBC architecture with types of JDBC drivers. [5 M]

Ans. Refer to Section 3.0.

3. The project in a database has the following fields: Title (text), StudentName (text), duration (Number), Class (text), submissiondate (date).

Write a Java program to read the project details from user and add it in table. Continue the process until user press 'Y' to the prompt. Add more (Yes/No)? [5 M]

Ans. Refer to Section Additional Programs.

April 2019

1. What is use of commit() in Transactions in Java.

[1 M]**Ans.** Refer to Section 3.7.

2. Write Advantage of using Type 4 driver in Database connectivity.

[1 M]**Ans.** Refer to Section 3.2.1.

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

[5 M]**Ans.** Refer to Additional Programs.



Servlets and JSP

Objectives...

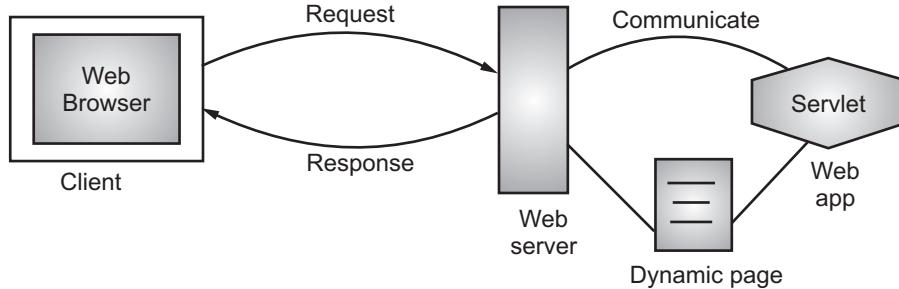
- To understand Concepts in Servlet
- To study Handling GET and POST Requests
- To learn Session Tracking and Retrieving Data from Database using Servlet
- To study JSP Concepts like Architecture, Advantages etc.
- To Learn Life Cycle of Servlet and JSP
- To study Scripting Element of JSP like Declarations, Comments, Expressions etc.
- To Learn JSP Directives like Page and Include and JSP Actions

4.0 INTRODUCTION

- The emergence of the Internet, Web technologies have become more and more important and Web applications become more and more common.
- In the earlier days web pages were static i.e. a user request a resource and the server returns it.
- A Web application is a web site with dynamic functionality on the server. Google, Facebook, Twitter are the examples of Web applications.
- A Web application is an application accessible from the Web. A web application is composed of Web components like Servlet, JSP etc., and other elements such as HTML, CSS, and JavaScript.
- Java Servlets and Java Server Pages (JSPs) are Java programs that run on a Java application server and extend the capabilities of the Web server.
- Servlet are the java programs that run on a Web server and act as a middle layer between a requests coming from HTTP client and databases or applications on the HTTP server.
- JSP is simply a text document that contains two types of text: static text which is predefined and dynamic text which is rendered after server response is received.

4.1 INTRODUCTION TO SERVLET**[April 17, 18, Oct. 18]**

- 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.
- Fig. 4.1 shows Servlet technology.
- Servlet technology is used to create web application. Servlet technology uses Java language to create web application.
- Web applications are helper application that resides at web server and build dynamic web pages.
- A dynamic page could be anything like a page that randomly chooses picture to display or even a page that display current time.
- When a user issues a request for a URL that corresponds to a java Servlets, the server hands the request off to the Servlet (program on server side) for processing.
- The Servlet dynamically produces a response to the request, typically an HTML web page and sends it back to the requesting web browser.
- Servlets provide a component-based, platform-independent method for building Web-based applications.
- Servlets provides an object-oriented and extensible middle tier for Web server based applications.
- Web servers generally cannot talk to databases – a web server alone cannot create web page content using data held in database.
- This information cannot make accessible to public through the web server. Here, Servlets helps to extend the functionality of a web server.
- Web server uses Servlets ability to access a database table, extract the data and convert this data into a format acceptable to a web server for delivery to a client browser through the Internet.

**Fig. 4.1: Servlet Technology**

- Servlet technology is robust and scalable as it uses the java language. Before Servlet, CGI (Common Gateway Interface) scripting language was used as a server-side programming language.
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, HttpServletRequest, HttpServletResponse etc.
- Servlet is an interface that must be implemented for creating any Servlet. Servlets have access to the entire family of Java APIs including the JDBC API to access enterprise databases.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming request. It can respond to any type of requests.

4.1.1 What is Servlet?

[April 17]

- A Java Servlet is a server side program that services HTTP requests and returns the result as HTTP responses.
- A Servlet:
 - is a program written using java that runs in a server application to answer clients requests.
 - is supported by virtually all web servers and application servers.
 - Solves the performance problem by executing all requests as thread in one process.
 - is a technology i.e. used to create web application.
 - is an API that provides many interfaces and classes including documentations.

4.1.2 Tasks of Servlets

- Various tasks of Servlets are listed below:
 1. **Read the explicit data sent by the clients (browsers):** This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
 2. **Read the implicit HTTP request data sent by the clients (browsers):** This includes cookies, media types and compression schemes the browser understands, and so forth.

3. **Process the data and generate the results:** This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
4. **Send the explicit data (i.e., the document) to the clients (browsers):** This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
5. **Send the implicit HTTP response to the clients (browsers):** This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

4.1.3 Advantages of Servlet

[April 18]

- Servlet offers the following benefits (advantages) that are not necessarily available in other technologies:
 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 us write robust applications.
 5. **Widespread acceptance:** Java is a widely accepted technology. This means that numerous vendors work on Java-based technologies. One of the advantages of this widespread acceptance is that you can easily find and purchase components that suit your needs, which saves precious development time.
 6. **Secure:** Servlet technology is very secured because it uses Java language.

4.1.4 How a Servlet Works?

- A Servlet is a Java program that runs on Web server. This program will start running when there is a request from the client side. A Servlet is loaded by the Servlet container the first time the Servlet is requested.
- When a client sends a request to the server, the user request is given to the server, the server runs the Servlet and gives the user request to the Servlet,
- Then the Servlet performs operations and generates results and returns the result as a response to the server, which in turn sends the response back to the user.

- After that, the Servlet stays in memory waiting for other requests - it will not be unloaded from the memory unless the Servlet container sees as hostage of memory.
- Each time the Servlet is requested, however, the Servlet container compares the timestamp of the loaded Servlet with the Servlet class file. If the class file timestamp is more recent, the Servlet is reloaded into memory.
- This way, we do not need to restart the Servlet container every time you update your Servlet.
- Fig. 4.2 shows working of Servlet.

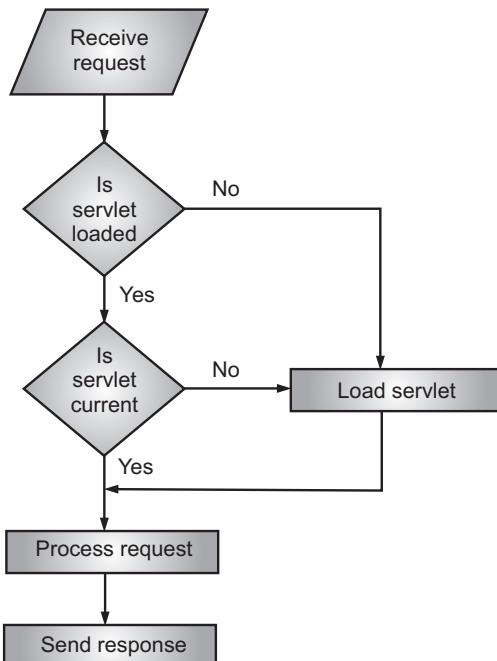


Fig. 4.2: Working of Servlet

4.1.5 Types of Servlet

- There are two types of Servlets, GenericServlet and HttpServlet. GenericServlet defines the generic or protocol independent Servlet.
 - The HttpServlet is subclass of GenericServlet and provides some http specific functionality like doGet and doPost methods.
1. **Generic Servlets:** It extends javax.servlet.GenericServlet. Generic Servlets are protocol independent. They contain no inherent HTTP support or any other transport protocol.
 2. **HTTP Servlets:** It extends javax.servlet.HttpServlet. They have built-in HTTP protocol support and are more useful in a Sun Java System Web Server environment.

- For both Servlet types, you implement the constructor method init() and the destructor method destroy() to initialize or deallocate resources.
- All Servlets must implement a service() method, which is responsible for handling Servlet requests.
- For generic Servlets, simply override the service method to provide routines for handling requests.
- HTTP Servlets provide a service method that automatically routes the request to another method in the Servlet based on which HTTP transfer method is used.
- So, for HTTP servlets, override doPost() to process POST requests, doGet() to process GET requests, and so on.

4.1.6 Servlet Hierarchy

- Fig. 4.3 shows the hierarchy of a typical HTTP Servlet. The Servlet is the root interface of the Servlet class hierarchy. All Servlets need to either directly or indirectly implement the Servlet interface.
- The GenericServlet class of the Servlet API implements the Servlet interface. The javax.servlet.GenericServlet class defines methods for building protocol-independent Servlets.
- The javax.servlet.http.HttpServlet class extends this class to provide HTTP specific methods.

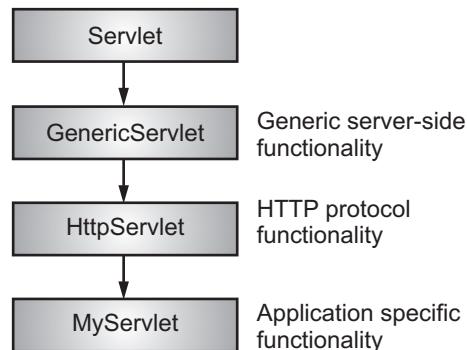


Fig. 4.3: Hierarchy of HTTP Servlet

- The GenericServlet class defines generic server-side operations:
 1. The HttpServlet extends this to define HTTP specific operations.
 2. Application specific Servlets extend this to provide application specific operations.

4.2 LIFE CYCLE OF SERVLET

[April 18, 19, Oct. 17]

- 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 life cycle of a Servlet consists of three methods namely, init(), service() and destroy(). These three methods belong to javax.servlet.Servlet interface.
- The lifecycle of a Servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a Servlet, the container performs the steps as shown in Fig. 4.4.

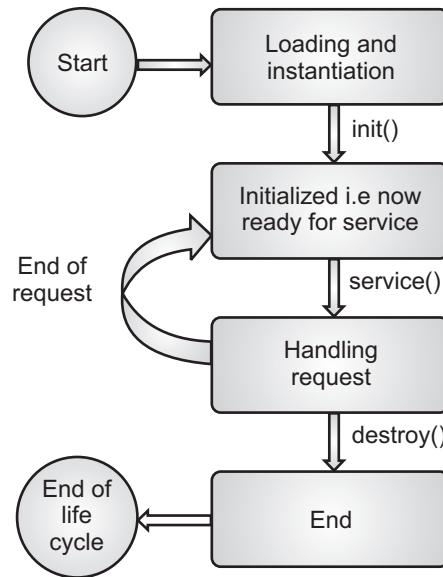


Fig. 4.4: Lifecycle of Servlet

- The lifecycle methods in Servlet are explained below:
 - init() method is invoked:** The init() 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 init method of applets. The Servlet is normally created when a user first invokes a URL corresponding to the Servlet, 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 init() method simply creates or loads some data that will be used throughout the life of the Sservlet.

The init() method definition looks like this:

```

public void init() throws ServletException
{
    // Initialization code...
}
  
```

2. **Service() method is invoked:** The service() 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 service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here, is the **syntax** of this method:

```
public void service(ServletRequest request,
                    ServletResponse response)
    throws ServletException, IOException
{
}
```

The service() method is called by the container and service method invokes doGet, doPost, doPut, doDelete etc., methods as appropriate. So we have nothing to do with service() method but we override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request.

- (i) **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 doGet() method. [April 16, 17, Oct. 17, 18]

```
public void doGet(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException
{
    // Servlet code ...
}
```

- (ii) **The doPost() Method:** A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method. [April 16, 17, Oct. 17, 18]

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException
{
    // Servlet code...
}
```

3. **destroy() method is invoked:** The destroy() method is called only once at the end of the life cycle of a Servlet. This method gives the Servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the Servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy()
{
    // Finalization code...
}
```

4.3 CREATING SERVLET

[April 17, 18]

- Servlets are Java classes which service HTTP requests and implement the javax.servlet.Servlet interface.
- Web application developers typically write Servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.
- To create a Servlet application you need to follow the below mentioned steps. These steps are common for the entire Web server. In our example we are using Apache Tomcat server.
- Apache Tomcat is an open source web server for testing Servlets and JSP technology. Download latest version of Tomcat Server and install it on your machine.
- After installing Tomcat Server on your machine follow the below mentioned steps:
 - Step 1:** Create directory structure for your application.
 - Step 2:** Create a Servlet.
 - Step 3:** Compile the Servlet.
 - Step 4:** Create Deployment Descriptor for your application.
 - Step 5:** Start the server and deploy the application.

Sample Code for Hello World:

- Following is the sample source code structure of a Servlet example to write Hello World:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
```

```
public class HelloWorld extends HttpServlet
{
    private String message;
    public void init() throws ServletException
    {
        // Do required initialization
        message = "Hello World 2021...";
    }
    public void doGet(HttpServletRequest request, HttpServletResponse
                      response) throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");
        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }
    public void destroy()
    {
        // do nothing.
    }
}
```

Compiling the Servlet:

- Let us put above code in Firstservlet.java file and put this file in /root/apache-tomcat-5.5.25/webapps/servlets-examples/WEB-INF/classes
- Assuming the environment is setup properly, go in ServletDevel directory and compile HelloWorld.java as follows:

```
$ javac HelloWorld.java
```

- If the Servlet depends on any other libraries, we have to include those JAR files on our CLASSPATH as well. We have included only servlet-api.jar JAR file because we are not using any other library in HelloWorld program.
 - This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK).
 - For this command to work properly, we have to include the location of the Java SDK that we are using in the PATH environment variable.
 - If everything goes fine, above compilation would produce HelloWorld.class file in the same directory. Next section would explain how a compiled servlet would be deployed in production.
-

Servlet Deployment:

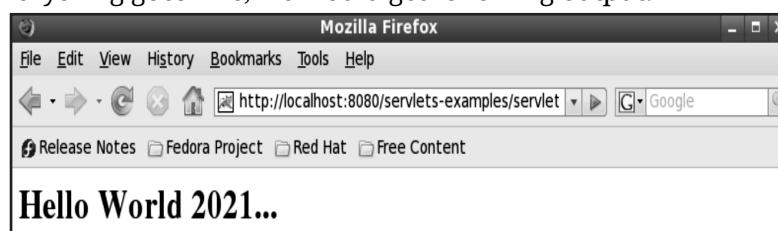
- Create following entries in web.xml file located in /root/apache-tomcat-5.5.25/webapps/servlets-examples/web.xml.

```
<servlet>
    <servlet-name> HelloWorld </servlet-name>
    <servlet-class> HelloWorld </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name> HelloWorld </servlet-name>
    <url-pattern>/servlet/HelloWorld </url-pattern>
</servlet-mapping>
```

- Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.
- We are almost done, now let us start tomcat server using:

```
[root@localhost bin]# sh shutdown.sh
Using CATALINA_BASE:      /root/apache-tomcat-5.5.25/
Using CATALINA_HOME:      /root/apache-tomcat-5.5.25/
Using CATALINA_TMPDIR:    /root/apache-tomcat-5.5.25//temp
Using JRE_HOME:           /root/jdk1.5.0_05/
[root@localhost bin]# sh startup.sh
Using CATALINA_BASE:      /root/apache-tomcat-5.5.25/
Using CATALINA_HOME:      /root/apache-tomcat-5.5.25/
Using CATALINA_TMPDIR:    /root/apache-tomcat-5.5.25//temp
Using JRE_HOME:           /root/jdk1.5.0_05/
```

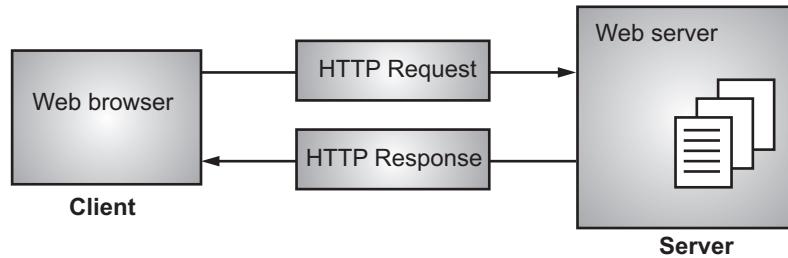
- If everything goes fine, we would get following output:



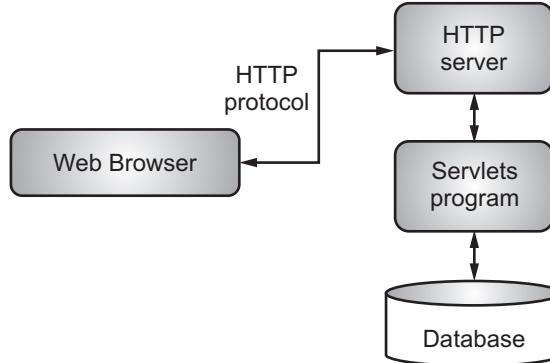
- A number of Web Servers that support Servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them.
- Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server.

4.4**HANDLING GET AND POST REQUESTS (HTTP)**

- The HTTP (HyperText Transfer Protocol) is a protocol that clients and servers use on the web to communicate.
- It is similar to other internet protocols such as SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) but there is one fundamental difference. HTTP is a stateless protocol i.e. HTTP supports only one request per connection.
- This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
- The client sends an HTTP request and the server answers (responses) with an HTML page to the client, using HTTP, (See Fig. 4.5).

**Fig. 4.5: HTML Page to the Client, using HTTP**

- Fig. 4.6 diagram shows the position of Servlets in a Web Application.

**Fig. 4.6: Servlet in a Web Application****HTTP Methods:**

- HTTP request can be made using a variety of methods, but the ones you will use most often are GET and POST.
- The method name tells the server the kind of request that is being made, and how the rest of the message will be formatted.

HTTP Methods:

1. **OPTIONS:** Request for communication options that are available on the request/response chain.
 2. **GET:** Request to retrieve information from server using a given URI (Uniform Resource Identifier).
 3. **HEAD:** Identical to GET except that it does not return a message-body, only the headers and status line.
 4. **POST:** Request for server to accept the entity enclosed in the body of HTTP method.
 5. **DELETE:** Request for the Server to delete the resource.
 6. **CONNECT:** Reserved for use with a proxy that can switch to being a tunnel.
 7. **PUT:** This is same as POST, but POST is used to create, PUT can be used to create as well as update. It replaces all current representations of the target resource with the uploaded content.
- Let us see get and post methods in detail.

1. GET Method:**[April 17]**

- 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.
- This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet() method.
- The **syntax of doGet()** is as follows:

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws IOException, ServletException
{ ..... }
```

2. Post Method:**[April 17]**

- 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.
- The **syntax of `doPost()`** is as given below:

```
public void doPost (HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException
{ ..... }
```

Program 4.1: Program to use of `doGet` and display message.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Hello1 extends HttpServlet
{
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp) throws ServletException, IOException
    {
        PrintWriter out;
        //set the content type
        resp.setContentType("text/html");
        out = resp.getWriter();
        out.println("<HTML><HEAD><TITLE>");
        out.println("Hello World Servlet");
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>Go Corona</H1>");
        out.println("<BODY></HTML>");
        out.close();
    }
}
```

Output:



Difference between GET and POST Requests:

| 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. |
| 5. | Get request is more efficient and used more than Post. | Post request is less efficient and used less than get. |

4.4.1 javax.servlet Package

- This contains various classes and interfaces which establish the framework in which servlet operate.

| Interface | Description |
|----------------------|--|
| Servlet | Declares life cycle methods for a Servlet. |
| ServletConfig | Allows Servlet to get initialization parameter. |
| ServletRequest | Used to read data from a client request. |
| ServletResponse | Used to write data to a client response. |
| SingleThreadModel | Indicates that Servlet is threadsafe. |
| GenericServlet | Implements Servlet and servletConfig interfaces. |
| ServletInputStream | Stream for reading request from a client. |
| ServletOutputStream | Stream for writing responses to a client. |
| ServletException | Indicates a Servlet error occurred. |
| UnavailableException | Indicates a Servlet is unavailable. |

4.4.2 Servlet Interface

- It has three methods init(), service() and destroy(). These methods are called by the server during the life cycle of a Servlet.

- The list of methods in this interface is given below:

| Method | Description |
|--|---|
| destroy() | Called at the time of unloading Servlet. |
| ServletConfig getServletConfig() | Returns ServletConfig object that contains any initialization parameters. |
| getServletInfo() | Returns a string describing the Servlet. |
| init(ServletConfig sc()) | It is called at the initialization of Servlet. |
| service(SevletRequest req, SevletResponse res) | It is called to process a request from a client. It reads requests and sends response back. |

4.4.3 ServletConfig Interface

- This interface allows a Servlet to obtain configuration data when loaded.
- The following methods are used in this interface:

| Method | Description |
|------------------------------------|--|
| ServletContext getServletContext() | Returns a context for the Servlet. |
| getInitParameter(String param) | Returns value of initialization. |
| getServletName() | Returns name of the invoking Servlets. |

4.4.4 ServletRequest Interface

- It allows getting the information about client request.
- The following methods are used in this interface:

| Method | Description |
|-------------------------------------|--|
| getAttribute(String attr) | Returns value of the attribute. |
| getContentLength() | Returns size of the request. – 1 is returned if size is unavailable. |
| getContentType() | Returns type of request. null if no type available. |
| getProtocol() | Returns description of the protocol. |
| ServletInputStream getInputStream() | Returns a stream used to read data. |
| getServerName() | Returns name of server. |
| getServerPort() | Returns port number. |
| getRemoteAddr() | Returns string equivalent to client IP address. |
| getRemoteHost() | Return string equivalent to client host name. |
| getScheme() | Returns transmission scheme of URL e.g. http, ftp etc. |

4.4.5 ServletResponse Interface

- This is used to return response back to the client.
- The methods of this interface are listed below:

| Method | Description |
|-----------------------------|---|
| getCharacterEncoding() | Returns character encoding for the response. |
| ServletOutputStream | Returns a stream used to write binary data to the response. |
| getOutputStream() | |
| PrintWriter getWriter() | Returns printwriter used to write character data to the response. |
| setContentLength(int size) | Sets the content length for the response to size. |
| setContentType(String type) | Sets the content type for the response to the type. |

4.4.6 javax.servlet.http Package

- This contains number of interfaces and classes commonly used by the developer.
- The following table shows interfaces and classes of this package:

| Interface | Description |
|---------------------------|--|
| HttpServletRequest | Reads data from HTTP request. |
| HttpServletResponse | Writes data to http response. |
| HttpSession | Allows session data to be read and written. |
| HttpSessionBindingListner | Informs an object it is bound or unbound from a session. |

| Classes | Description |
|-------------------------|--|
| Cookie | Allows state information to be stored on a client machine. |
| HttpServlet | Provides methods to handle http request and response. |
| HttpSessionEvent | Encapsulates session-changed events. |
| HttpSessionBindingEvent | Tells that a listener is bound to or unbound from a session value. |

HttpServletRequest Interface:

- This interface encapsulates information of the request made to a Servlet. This interface gets information from client so that it can be used in service() method.
- The HTTP protocol specified header information to be accessed from service() method.

- The different methods of this interface are shown below:

| Method | Description |
|-----------------------------|--|
| getMethod() | Returns the HTTP Get, Post method. |
| getQueryString() | Returns query string that is part of http request. |
| getRemoteUser() | Returns name of user making http request. |
| getRequestedSessionId() | Returns session id with http request. |
| getSession(boolean) | If boolean = false, it returns valid current session else creates a new session. |
| isRequestedSessionIdValid() | Checks whether the http request associated with a session is valid. |
| getCookies() | Returns array of cookies. |

HttpServletResponse Interface:

- This encapsulates the information of the response sent by a Servlet to a client.
- This allows a service() method to manipulate http protocol specified header information and return data to its client.
- There are various methods available which are shown below:

| Method | Description |
|----------------------|--|
| addCookie | Adds specified cookie to a response. |
| encodeUrl(String) | Encodes specified url by adding session to it. |
| sendRedirect(String) | Sends temporary redirect response to a client using specified redirect location url. |
| sendError(int) | Sends error Response to a client. |

4.5 READING SERVLET PARAMETERS (FORM)

- The ServletRequest class includes the methods which allows us to read the names and values of parameters those are included in a client request.
- This is shown by using two programs, one is for HTML code and other one is for accepting the values from the user.
- Servlets handles form data parsing automatically using the following methods depending on the situation:
 1. **getParameter():** Call request.getParameter() method to get the value of a form parameter.
 2. **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
 3. **getParameterNames():** Call this method if we want a complete list of all parameters in the current request.

- Let us see GET and POST methods in detail.

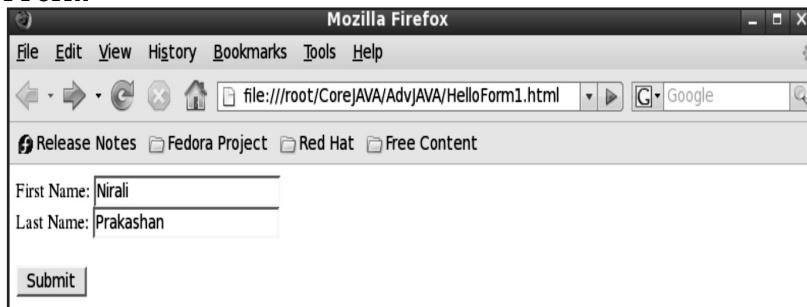
1. GET Method Example Using URL Form:

- Here, is a simple program which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
  <body>
    <form action="http://localhost:8080/servlets-examples/servlet
          /HelloForm" method="GET">
      First Name: <input type="text" name="first_name">
      <br />
      Last Name: <input type="text" name="last_name" /><br><br>
      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

- Keep this HTML in a file HelloForm1.htm and put it in <Tomcat-installation-directory>/webapps ROOT directory. When we would access http://localhost:8080/HelloForm1.htm, here is the actual output of the above form.

Top of Form



- Here, is a simple URL which will pass two values to HelloForm1 program using GET method.

```
http://localhost:8080/servlets-examples/servlet
/HelloForm1?first_name=Nirali&last_name=Prakashan
```

- Below is HelloForm.java servlet program to handle input given by web browser. We are going to use getParameter() method which makes it very easy to access passed information:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
```

```

public class HelloForm extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Using GET Method to Read Form Data";
        String docType = "<!doctype html public \"-//w3c//dtd html 4.0 \" +\n                           \"transitional//en\\\">\\n";
        out.println(docType +
                    "<html>\\n" +
                    "  <head><title>" + title + "</title></head>\\n" +
                    "  <body bgcolor=\"#f0f0f0\\\">\\n" +
                    "    <h1 align=\"center\\\">" + title + "</h1>\\n" +
                    "    <ul>\\n" +
                    "      <li><b>First Name</b>:" +
                    "        " + request.getParameter("first_name") + "\\n" +
                    "      <li><b>Last Name</b>:" +
                    "        " + request.getParameter("last_name") + "\\n" +
                    "    </ul>\\n" +
                    "  </body></html>");
    }
}

```

- Assuming the environment is setup properly, compile HelloForm.java as follows:

```
$ javac HelloForm.java
```

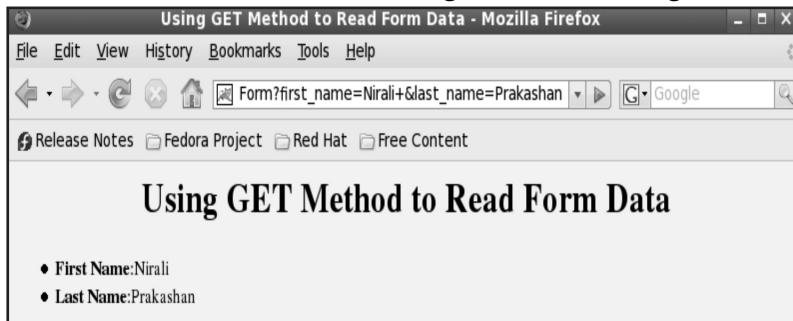
- If everything goes fine, above compilation would produce HelloForm.class file. Next we would have to copy this class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/.

```

<servlet>
    <servlet-name>HelloForm</servlet-name>
    <servlet-class>HelloForm</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloForm</servlet-name>
    <url-pattern>/servlet/HelloForm</url-pattern>
</servlet-mapping>

```

- Now type `http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI` in our browser's Location:box and make sure WE already started tomcat server, before firing above command in the browser. This would generate following result:



- Try to enter First Name and Last Name and then click submit button to see the result on the local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

2. POST Method Example Using Form:

- Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm1 extends HttpServlet
{
    // Method to handle GET method request.
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
        throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Using Post Method to Read Form Data";
        String docType =
        "<!doctype html public "-//w3c//dtd html 4.0 " +
        "transitional//en\">\n";
        out.println(docType +
```

```

        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<ul>\n" +
        "  <li><b>First Name</b>: "
        + request.getParameter("first_name") + "\n" +
        "  <li><b>Last Name</b>: "
        + request.getParameter("last_name") + "\n" +
        "</ul>\n" +
        "</body></html>");
    }
}

```

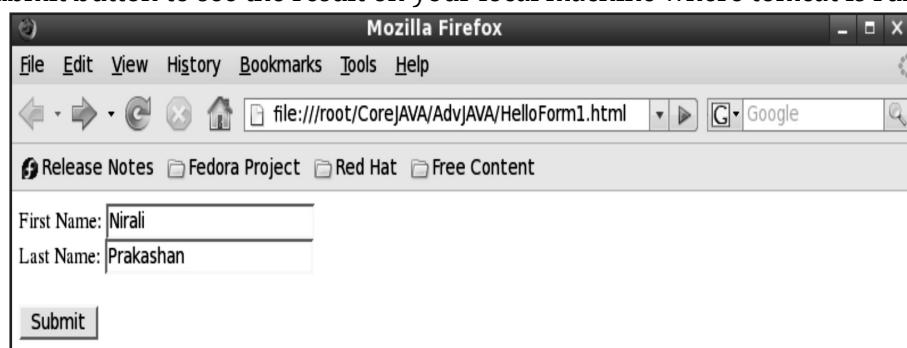
- Now compile, deploy the above Servlet and test it using Hello.htm with the POST method as follows:

```

<html>
  <body>
    <form action="http://localhost:8080/servlets-examples
           /servlet/HelloForm1" method="POST">
      First Name: <input type="text" name="first_name">
      <br />
      Last Name: <input type="text" name="last_name" /><br><br>
      <input type="submit" value="Submit" />
    </form>
  </body>
</html>

```

- Here, is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.



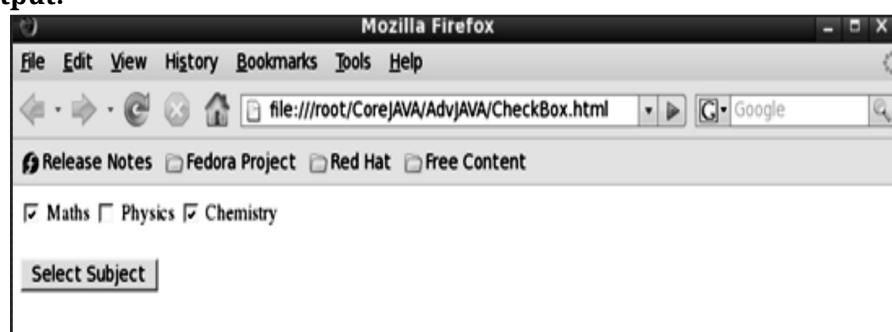


3. Passing Checkbox Data to Servlet Program:

- Checkboxes are used when more than one option is required to be selected.
- Here, is example HTML code, CheckBox.htm, for a form with two checkboxes.

```
<html>
  <body>
    <form action="http://localhost:8080/servlets/examples/servlet
          /CheckBox" method="POST" target="_blank">
      <input type="checkbox" name="maths" checked="checked" /> Maths
      <input type="checkbox" name="physics" /> Physics
      <input type="checkbox" name="chemistry" checked="checked" />
      Chemistry
      <br><br>
      <input type="submit" value="Select Subject" />
    </form>
  </body>
</html>
```

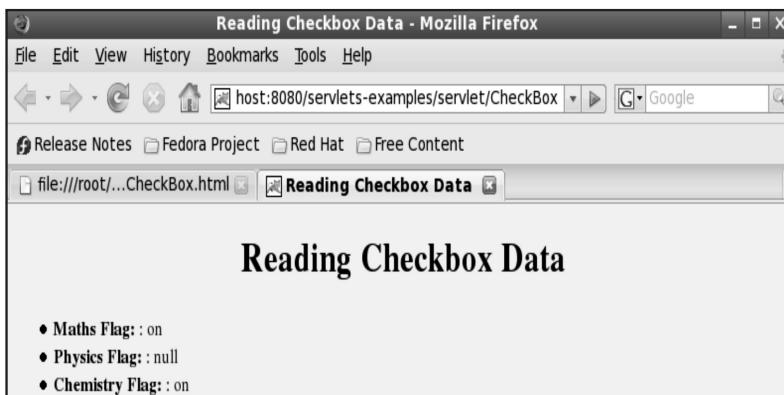
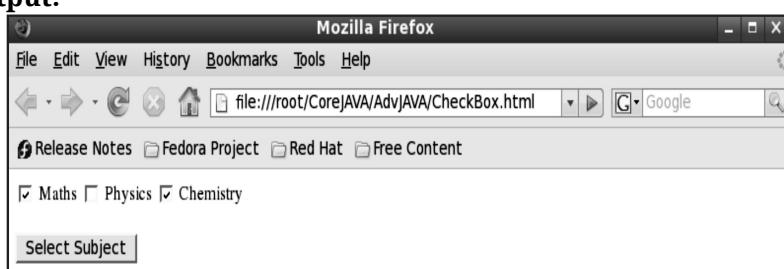
Output:



- Below is CheckBox.java Servlet program to handle input given by web browser for checkbox button.

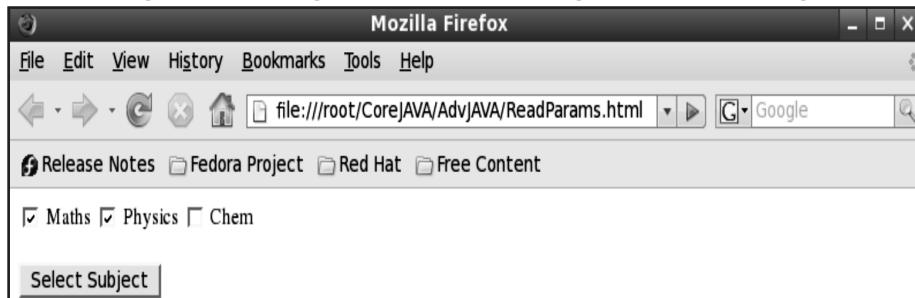
```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class CheckBox extends HttpServlet
{
    // Method to handle GET method request.
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Reading Checkbox Data";
        String docType =
            "<!doctype html public \"-//w3c//dtd html 4.0\" +
            \"transitional//en\"\">\n";
        out.println(docType +
                    "<html>\n" +
                    "<head><title>" + title + "</title></head>\n" +
                    "<body bgcolor=\"#f0f0f0\"\n" +
                    "<h1 align=\"center\"\>" + title + "</h1>\n" +
                    "<ul>\n" +
                    " <li><b>Maths Flag: </b>: " +
                    + request.getParameter("maths") + "\n" +
                    " <li><b>Physics Flag: </b>: " +
                    + request.getParameter("physics") + "\n" +
                    " <li><b>Chemistry Flag: </b>: " +
                    + request.getParameter("chemistry") + "\n" +
                    "</ul>\n" +
                    "</body></html>");
    }
}
```

```
// Method to handle POST method request.
public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, IOException
{
    doGet(request, response);
}
}
```

Output:**4. Reading all Form Parameters:****HTML Code**

```
<html>
  <body>
    <form action="http://localhost:8080/servlets/examples/servlet/ReadParams" method="POST" target="_blank">
      <input type="checkbox" name="maths" checked="checked" /> Maths
      <input type="checkbox" name="physics"/> Physics
      <input type="checkbox" name="chemistry" checked="checked" /> Chem
      <br><br>
      <input type="submit" value="Select Subject" />
    </form>
  </body>
</html>
```

- Now calling servlet using above form would generate following result:



- We can try above servlet to read any other form's data which is having other objects like text box, radio button or drop down box etc.
- Following is the generic example which uses `getParameterNames()` method of `HttpServletRequest` to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.
- Once, we have an Enumeration, we can loop down the Enumeration in the standard manner, using `hasMoreElements()` method to determine when to stop and using `nextElement()` method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class ReadParams extends HttpServlet
{
    // Method to handle GET method request.
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Reading All Form Parameters";
        String docType =
            "<!doctype html public "-//w3c//dtd html 4.0 " +
            "transitional//en\">\n";
        out.println(docType +

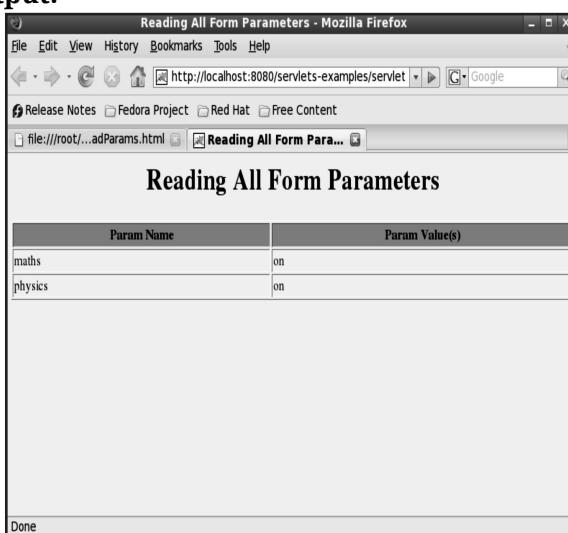
```

```
"<html>\n" +
"<head><title>" + title + "</title></head>\n" +
"<body bgcolor=\"#f0f0f0\">\n" +
"<h1 align=\"center\">" + title + "</h1>\n" +
"<table width=\"100%\" border=\"1\" align=\"center\">\n" +
"<tr bgcolor=\"#949494\">\n" +
"<th>Param Name</th><th>Param Value(s)</th>\n" +
"</tr>\n");
Enumeration paramNames = request.getParameterNames();
while(paramNames.hasMoreElements())
{
    String paramName = (String)paramNames.nextElement();
    out.print("<tr><td>" + paramName + "</td>\n<td>");
    String[] paramValues =
                    request.getParameterValues(paramName);
    // Read single valued data
    if (paramValues.length == 1)
    {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0)
            out.println("<i>No Value</i>");
        else
            out.println(paramValue);
    } else
    {
        // Read multiple valued data
        out.println("<ul>");
        for(int i=0; i < paramValues.length; i++)
        {
            out.println("<li>" + paramValues[i]);
        }
        out.println("</ul>");
    }
}
out.println("</tr>\n</table>\n</body></html>");
```

```

// Method to handle POST method request.
public void doPost(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
    doGet(request, response);
}
}

```

Output:

4.6 ACCESSING DATABASE

- To start with basic concept, let us create a simple table and create few records in that table as follows:

Create Table:

- To create the Employees table in TEST database, use the following steps:


```
[root@localhost ~]# su - postgres
-bash-3.2$ initdb TEST
```
- The files belonging to this database system will be owned by user "postgres". This user must also own the server process.
- The database cluster will be initialized with locale en_US.UTF-8. The default database encoding has accordingly been set to UTF8.


```
creating directory TEST ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
```

```
selecting default shared_buffers/max_fsm_pages ... 24MB/153600
creating configuration files ... ok
creating template1 database in TEST/base/1 ... ok
initializing pg_authid ... ok
initializing dependencies ... ok
creating system views ... ok
loading system objects' descriptions ... ok
creating conversions ... ok
setting privileges on built-in objects ... ok
creating information schema ... ok
vacuuming database template1 ... ok
copying template1 to template0 ... ok
copying template1 to postgres ... ok

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the -A option the
next time you run initdb.
```

Success. You can now start the database server using:

```
postgres -D TEST
or
pg_ctl -D TEST -l logfile start
-bash-3.2$
-bash-3.2$ pg_ctl -D TEST -l logfile start
server starting
-bash-3.2$ createdb TEST
CREATE DATABASE
-bash-3.2$ psql TEST
Welcome to psql 8.2.5, the PostgreSQL interactive terminal.
```

```
Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help with psql commands
      \g or terminate with semicolon to execute query
      \q to quit
```

```
TEST=# create table Employees(id int,age int,first text,last text);
CREATE TABLE
```

Create Data Records:

- Finally WE create few records in Employee table as follows:

```
TEST=# insert into Employees values(101,65,'Naren','Modi');
INSERT 0 1
TEST=# insert into Employees values(102,40,'Amar','Salunke');
INSERT 0 1
TEST=# insert into Employees values(103,20,'Ravi','Samant');
INSERT 0 1
TEST=# select * from Employees;
+----+----+----+----+
| id | age | first | last |
+----+----+----+----+
| 101 | 65 | Naren | Modi |
| 102 | 40 | Amar | Salunke |
| 103 | 20 | Ravi | Samant |
+----+----+----+----+
(3 rows)
```

Accessing a Database:

- Here, is an example which shows how to access TEST database using Servlet.

```
// Loading required libraries
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class DatabaseAccess extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
        throws ServletException, IOException
    {
        Connection conn = null;
        Statement stmt = null;

        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>DataBase Result</h1>");
```

```
try
{
    // Register JDBC driver
    Class.forName("org.postgresql.Driver");
    // Open a connection
    conn = DriverManager.getConnection
        ("jdbc:postgresql:TEST","postgres","");
    // Execute SQL query
    stmt = conn.createStatement();
    String sql;
    sql = "SELECT id, first, last, age FROM Employees";
    ResultSet rs = stmt.executeQuery(sql);
    // Extract data from result set
    while(rs.next())
    {
        //Retrieve by column name
        int id   = rs.getInt("id");
        int age = rs.getInt("age");
        String first = rs.getString("first");
        String last = rs.getString("last");
        //Display values
        out.println("ID: " + id + "<br>");
        out.println(", Age: " + age + "<br>");
        out.println(", First: " + first + "<br>");
        out.println(", Last: " + last + "<br>");
    }
    out.println("</body></html>");
    // Clean-up environment
    rs.close();
    stmt.close();
    conn.close();
}
catch(SQLException se)
{
    //Handle errors for JDBC
    se.printStackTrace();
}
```

```

        catch(Exception e)
        {
            //Handle errors for Class.forName
            e.printStackTrace();
        }
        finally
        {
            //finally block used to close resources
            try
            {
                if(stmt!=null)
                    stmt.close();
            }
            catch(SQLException se2)
            {
                // nothing we can do
            }
            try
            {
                if(conn!=null)
                    conn.close();
            }
            catch(SQLException se)
            {
                se.printStackTrace();
            } //end finally try
        } //end try
    }
}

```

- Now let us compile above servlet and create following entries in web.xml

```

.....
<servlet>
    <servlet-name>DatabaseAccess</servlet-name>
    <servlet-class>DatabaseAccess</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DatabaseAccess</servlet-name>
    <url-pattern>/servlet/DatabaseAccess</url-pattern>
</servlet-mapping>
.....

```

- Now call this servlet using URL <http://localhost:8080/DatabaseAccess> which would display following response:

```

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/servlets-examples/servlet
Release Notes Fedora Project Red Hat Free Content
 DataBase Result
ID: 101
, Age: 65
, First: Naren
, Last: Modi
ID: 102
, Age: 40
, First: Amar
, Last: Salunke
ID: 103
, Age: 20
, First: Ravi
, Last: Samant

```

4.7 SESSION MANAGEMENT

- Session management is a mechanism used by the Web container to store session information for a particular user.
- There are four different techniques used by Servlet application for session management. These techniques are Cookies, Hidden field, URL Rewriting and Session Object.

4.7.1 Concept of Session

- Session simply means a particular interval of time. Session is used to store everything that we can get from the client from all the requests the client makes.
- Fig. 4.7 shows how sessions work.

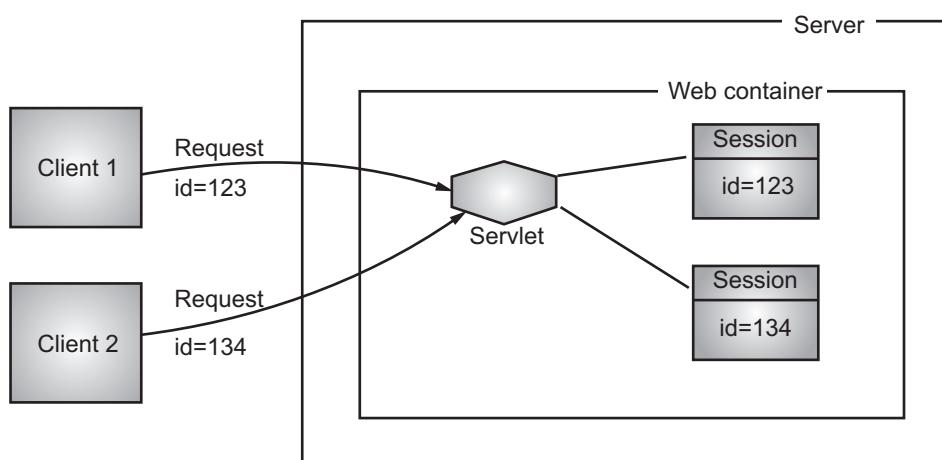


Fig. 4.7: Working of Session

4.7.2 Session Tracking

[Oct. 18, April 19]

- Session tracking is a way to maintain state (data) of an user. It is also known as session management in Servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new request. It is shown in the Fig. 4.8.

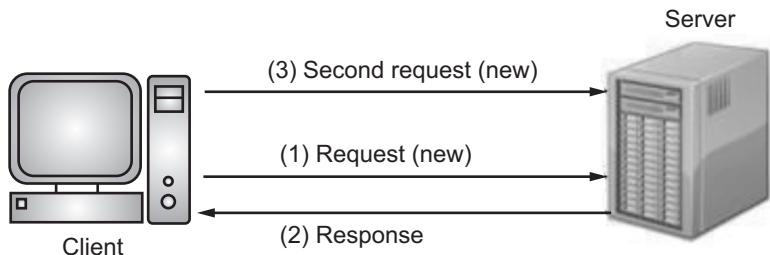


Fig. 4.8: Session Tracking

- There are following techniques used in session tracking:
 1. User authorization,
 2. Cookies,
 3. Hidden Form Field,
 4. URL Rewriting, and
 5. HttpSession.

4.7.2.1 User Authorization

- Users can be authorized to use the web application in different ways. Basic concept is that the user will provide username and password to login to the application. Based on that the user can be identified and the session can be maintained.
- One way to perform session tracking is to leverage the information that comes with user authorization.
- We can use the username to track a client session. Once a user has logged in, the browser remembers her username and resends the name and password as the user views new pages on the site.
- A Servlet can identify the user through her username and thereby track her session. For example, if the user adds an item to her virtual shopping cart, that fact can be remembered (in a shared class or external database, perhaps) and used later by another Servlet when the user goes to the check-out page.

- For example, a Servlet that utilizes user authorization might add an item to a user's shopping cart with code like the following:

```
String name = req.getRemoteUser();
if (name == null)
{
    // Explain that the server administrator should protect this page
}
else
{
    String[] items = req.getParameterValues("item");
    if (items != null)
    {
        for (int i = 0; i < items.length; i++)
        {
            addItemToCart(name, items[i]);
        }
    }
}
```

- Another servlet can then retrieve the items from a user's cart with code like this:

```
String name = req.getRemoteUser();
if (name == null)
{
    // Explain that the server administrator should protect this page
}
else
{
    String[] items = getItemsFromCart(name);
}
```

- The biggest advantage of using user authorization to perform session tracking is that it's easy to implement.
- Simply tell the server to protect a set of pages, and use getRemoteUser() to identify each client. Another advantage is that the technique works even when the user accesses the site from different machines. It also works even if the user strays from your site or exits her browser before coming back.
- The biggest disadvantage of user authorization is that it requires each user to register for an account and then log in each time she starts visiting your site.

- Most users will tolerate registering and logging in as a necessary evil when they are accessing sensitive information, but it's overkill for simple session tracking. We clearly need a better approach to support anonymous session tracking.
- Another small problem with user authorization is that a user cannot simultaneously maintain more than one session at the same site.

4.7.2.2 URL Rewriting

- URL rewriting is another way to support anonymous session tracking. With URL rewriting, every local URL the user might click on is dynamically modified or rewritten, to include extra information.
- The extra information can be in the form of extra path information, added parameters or some custom, server-specific URL change.
- Due to the limited space available in rewriting a URL, the extra information is usually limited to a unique session ID.
- With URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.
- We can send parameter name/value pairs using the following format:
`url?name1=value1&name2=value2&...`
- A name and a value is separated using an equal sign (=) a parameter name/value pair is separated from another parameter name/value pair using the ampersand (&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.
- From a Servlet, you can use the HttpServletRequest interface's getParameter method to obtain a parameter value.
- For instance, to obtain the value of the second parameter, we write the following:
`request.getParameter(name2);`
- The use of URL rewriting is easy and simple. When using this technique, however, we need to consider several things:
 1. The number of characters that can be passed in a URL is limited. Typically, a browser can pass up to 2,000 characters.
 2. The value that we pass can be seen in the URL. Sometimes, this is not desirable. For example, some people prefer their password not to appear on the URL.
 3. We need to encode certain characters, such as & and? characters and white spaces, that you append to a URL.

URL Rewriting for Session Management:

- If the client has disabled cookie in the browser then session management using cookie won't work. In that case URL rewriting can be used as a backup.

- In URL rewriting, a token (parameter) is added at the end of the URL. The token consist of name/value pair separated by an equal (=) sign.
- Fig. 4.9 shows an example of URL rewriting.

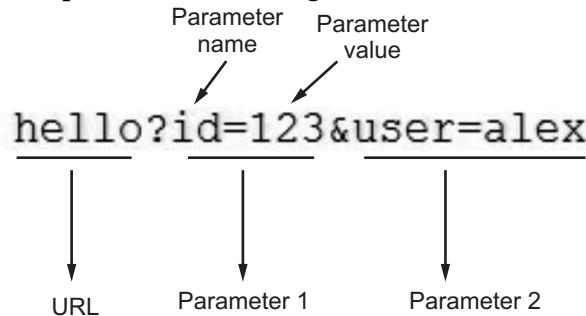


Fig. 4.9: URL Rewriting

- When the user clicks the URL links having parameters, the request goes to the Web Container with extra bit of information at the end of URL.
- The Web Container will fetch the extra part of the requested URL and uses it for session management.
- The `getParameter()` method is used to get the parameter value.

Advantage of URL Rewriting:

1. It will always work whether cookie is disabled or not, (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting:

1. It will work only with links.
2. It can send only textual information.

4.7.2.3 Hidden Form Fields

- Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state.
- In this case user information is stored in hidden field value and retrieve from another Servlet.
- In short, in Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another Servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- Let's see the code to store value in hidden field.

```
<input type="hidden" name="uname" value="Vimal Jaiswal">
```

Here, `uname` is the hidden field name and `Vimal Jaiswal` is the hidden field value.

- In Fig. 4.10 we are storing the name of the user in a hidden textfield and getting that value from another Servlet.

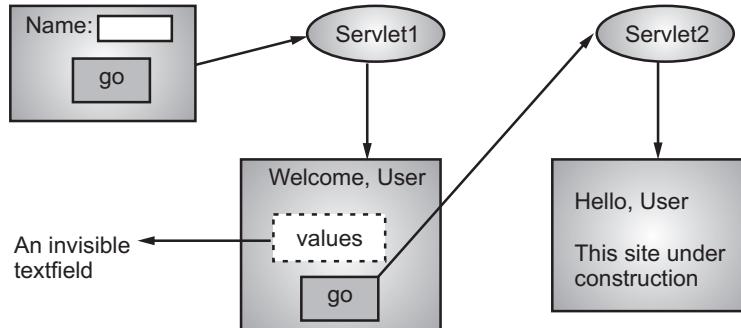


Fig. 4.10: Hidden Form Field

Advantage of Hidden Form Field:

- It will always work whether cookie is disabled or not.

Disadvantages of Hidden Form Field:

- It is maintained at server side.
- Extra form submission is required on each page.
- Only textual information can be used.

4.7.2.4 Cookies

[April 16, 17, 18, 19 Oct. 17]

- 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.
- After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.
- Fig. 4.11 shows working of cookies.

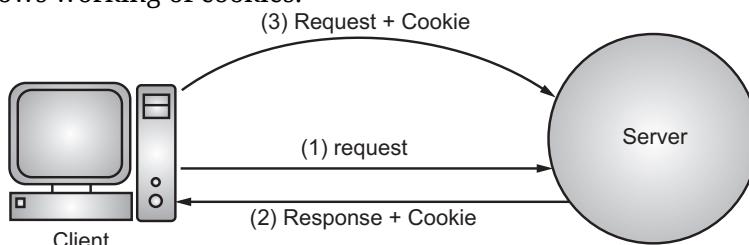


Fig. 4.11: Working of Cookies

- 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.

- In Servlet programming, a cookie is represented by the Cookie class in the javax.servlet.http package.
- We can create a cookie by calling the Cookie class constructor and passing two String objects i.e., the name and value of the cookie.
- For instance, the following code creates a cookie object called c1. The cookie has the name "myCookie" and a value of "secret":

```
Cookie c1 = new Cookie("myCookie", "secret");
```

- You then can add the cookie to the HTTP response using the addCookie method of the HttpServletResponse interface:

```
response.addCookie(c1);
```

- The following example shows how you can create two cookies called userName and password and illustrates how those cookies are transferred back to the server. The Servlet is called CookieServlet and its code is given below.
- When it is first invoked, the doGet method of the servlet is called. The method creates two cookies and adds both to the HttpServletResponse object, as follows:

```
Cookie c1 = new Cookie("userName", "tanmay");
Cookie c2 = new Cookie("password", "rashmi");
response.addCookie(c1);
response.addCookie(c2);
```

- Next, the doGet method sends an HTML form that the user can click to send another request to the servlet:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>Cookie Test</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("Please click the button to see the cookies sent to you.");
out.println("<BR>");
out.println("<FORM METHOD=POST>");
```

- ```

 out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
 out.println("</FORM>");
 out.println("</BODY>");
 out.println("</HTML>");

 - The form does not have any element other than a submit button. When the form is submitted, the doPost method is invoked.
 - To retrieve cookies, we use the getCookies method of the HttpServletRequest interface. This method returns a Cookie array containing all cookies in the request. It is your responsibility to loop through the array to get the cookie you want, as follows:

```

Cookie[] cookies = request.getCookies();
int length = cookies.length;
for (int i=0; i<length; i++)
{
    Cookie cookie = cookies[i];
    out.println("<B>Cookie Name:</B> " +cookie.getName() + "<BR>");
    out.println("<B>Cookie Value:</B> " +cookie.getValue() + "<BR>");
}

```

```

**Program 4.2:** Program to sending and receiving cookies.

```

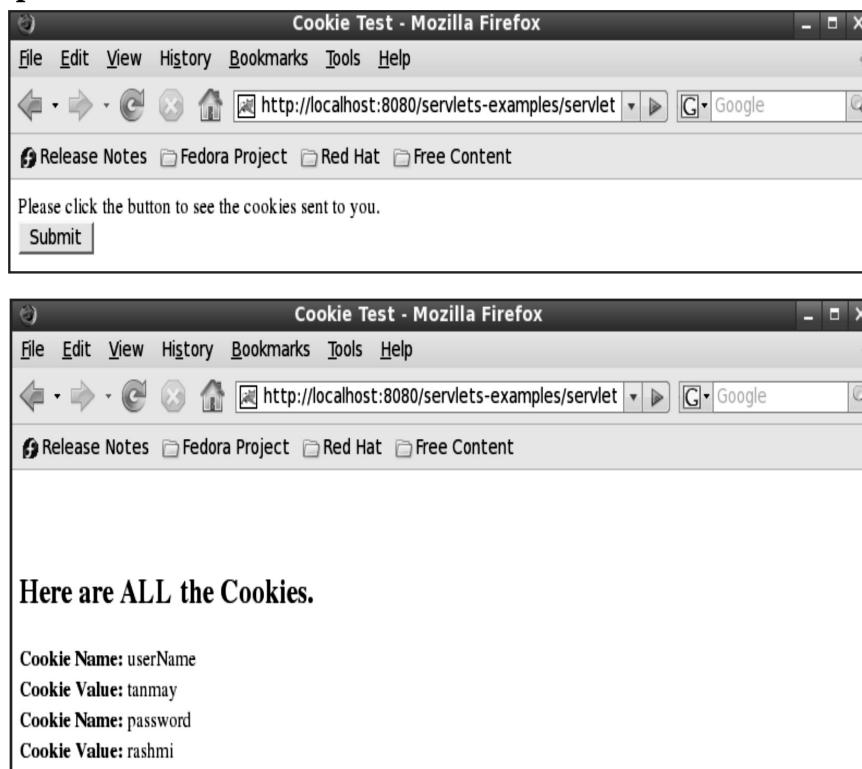
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class CookieServlet extends HttpServlet
{
 /*Process the HTTP Get request*/
 public void doGet(HttpServletRequest request, HttpServletResponse
 response) throws ServletException, IOException
 {
 Cookie c1 = new Cookie("userName", "tanmay");
 Cookie c2 = new Cookie("password", "rashmi");
 response.addCookie(c1);
 response.addCookie(c2);
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("<HTML>");
 out.println("<HEAD>");

```

---

```
 out.println("<TITLE>Cookie Test</TITLE>");
 out.println("</HEAD>");
 out.println("<BODY>");
 out.println("Please click the button to see the cookies sent to you.");
 out.println("
");
 out.println("<FORM METHOD=POST>");
 out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
 out.println("</FORM>");
 out.println("</BODY>");
 out.println("</HTML>");
 }
 /*Process the HTTP Post request*/
 public void doPost(HttpServletRequest request, HttpServletResponse
 response) throws ServletException, IOException
 {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("<HTML>");
 out.println("<HEAD>");
 out.println("<TITLE>Cookie Test</TITLE>");
 out.println("</HEAD>");
 out.println("<BODY>");
 out.println("

<H2> Here are ALL the Cookies.</H2>");
 Cookie[] cookies = request.getCookies();
 int length = cookies.length;
 for (int i=0; i<length; i++)
 {
 Cookie cookie = cookies[i];
 out.println("Cookie Name:" + cookie.getName() + "
");
 out.println("Cookie Value:" + cookie.getValue() + "
");
 }
 out.println("</BODY>");
 out.println("</HTML>");
 }
}
```

**Output:****Advantages of Cookies:**

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

**Disadvantages of Cookies:**

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

**Persisting Cookies:**

- The cookies we created in the previous last as long as the browser is open. When the browser is closed, the cookies are deleted.
- We can choose to persist cookies so that they last longer.
- The javax.servlet.http.Cookie class has the setMaxAge method that sets the maximum age of the cookie in seconds.

**4.7.2.5 HttpSession****[April 16, Oct. 17, 18]**

- The HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object.
- Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequest object.

- An object of HttpSession can be used to perform two tasks:
  1. Bind objects, and
  2. View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

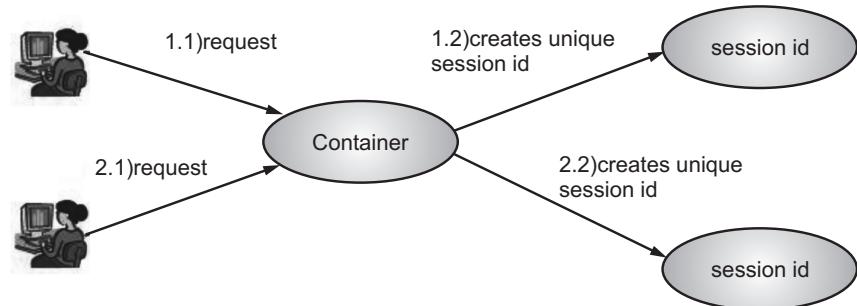


Fig. 4.12: HTTP Session

### How HttpSession Works?

- Fig. 4.13 shows working of HttpSession.

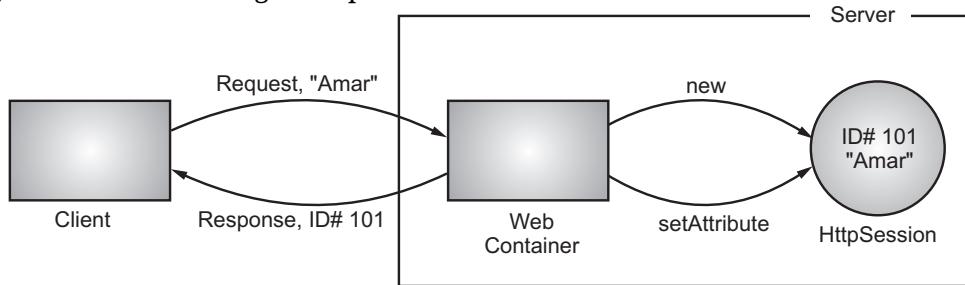
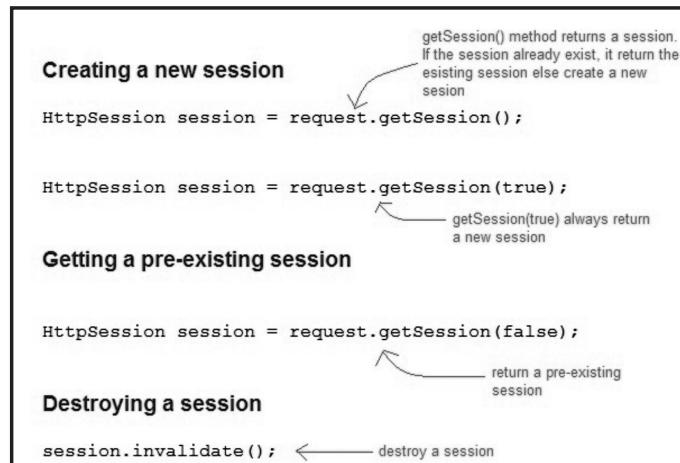


Fig. 4.13: Working of HttpSession

1. On client's first request, the Web Container generates a unique session ID and web container to identify where the request is coming from.
2. The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.

### HttpSession Interface:



**Some Methods of HttpSession:**

1. `long getCreationTime()` method returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.
2. `String getId()` method returns a string containing the unique identifier assigned to the session.
3. `long getLastAccessedTime()` method returns the last time the client sent a request associated with the session.
4. `int getMaxInactiveInterval()` method returns the maximum time interval, in seconds.
5. `void invalidate()` method destroys the session.
6. `boolean isNew()` method returns true if the session is new else false.
7. `void setMaxInactiveInterval(int interval)` method specifies the time, in seconds, after servlet container will invalidate the session.

**Program 4.3:** Complete program demonstrating usage of HttpSession. All the files mentioned below are required for the example.

**index.html**

```
<html>
<body>
<form method="post" action=
 "http://localhost:8080/servlets-examples/servlet/Validate">
 User: <input type="text" name="user" />

 Password: <input type="text" name="pass" />

 <input type="submit" value="submit">
</form>
</body>
</html>
```

**web.xml**

```
<servlet>
 <servlet-name>Validate</servlet-name>
 <servlet-class>Validate</servlet-class>
</servlet>
<servlet>
 <servlet-name>Welcome</servlet-name>
 <servlet-class>Welcome</servlet-class>
</servlet>
<servlet-mapping>
```

```
<servlet-name>Validate</servlet-name>
<url-pattern>/servlet/Validate</url-pattern>
</servlet-mapping>
<servlet-mapping>
 <servlet-name>Welcome</servlet-name>
 <url-pattern>/servlet/Welcome</url-pattern>
</servlet-mapping>
```

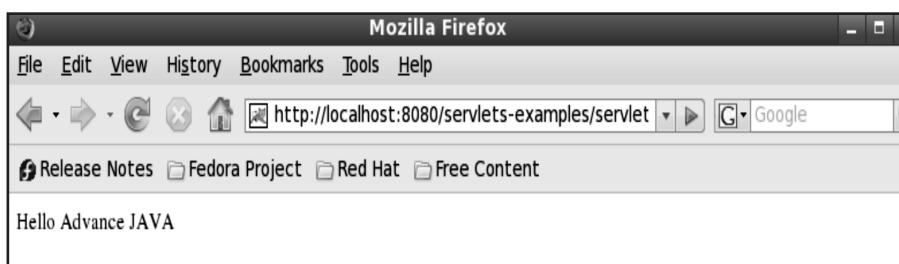
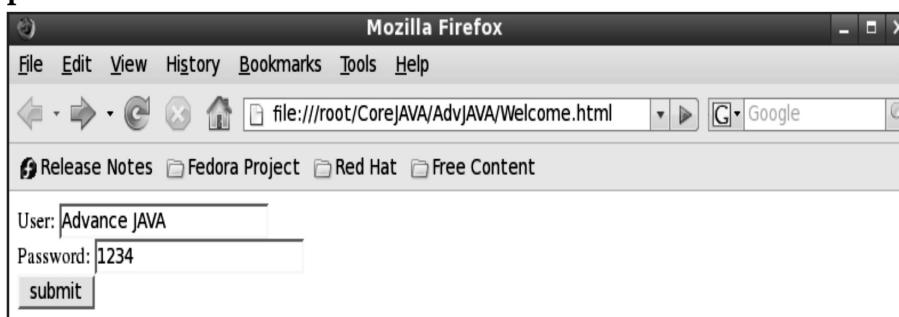
**Validate.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Validate extends HttpServlet
{
 protected void doPost(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException
 {
 response.setContentType("text/html");
 String name = request.getParameter("user");
 String pass = request.getParameter("pass");
 if(pass.equals("1234"))
 {
 //creating a session
 HttpSession session = request.getSession();
 session.setAttribute("user", name);
 response.sendRedirect("Welcome");
 }
 }
}
```

**Welcome.java**

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

```
public class Welcome extends HttpServlet
{
 protected void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException
 {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 HttpSession session = request.getSession();
 String user = (String)session.getAttribute("user");
 out.println("Hello "+user);
 }
}
```

**Output:****4.8 INTRODUCTION TO JSP**

[April 17, Oct. 18]

- JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- JSP is a standard for developing interactive Web applications, (pages containing dynamic content).

- A JSP web page may display different content based on certain parameters (information stored in a database, the user preferences etc.), while a classic webpage (with the .htm or .html extension) will continuously display the same information.
- JSP is actually a powerful scripting language executed on the server side (like CGI, PHP, ASP) and not on the client side (Java applets which run in the browser of the user connected to a site).
- JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than Servlet such as expression language, JSTL etc.
- A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

### Why Use JSP?

1. Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
2. JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
3. JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
4. JSP pages can be used in combination with Servlets that handle the business logic, the model supported by Java Servlet template engines.

### Comparison between JSP and Servlet:

[April 16]

| Sr. No. | Terms                               | JSP                                                                                    | Servlet                                                                                         |
|---------|-------------------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| 1.      | What are they?                      | JSP is a webpage scripting language, generally used to create the dynamic web content. | Servlets are Java programs that are already compiled and which also create dynamic web content. |
| 2.      | Typically                           | JSP is typically more oriented towards displaying information.                         | Servlet is more oriented towards processing information.                                        |
| 3.      | Role in MVC (Model View Controller) | JSP acts as a viewer.                                                                  | Servlet acts as a controller.                                                                   |

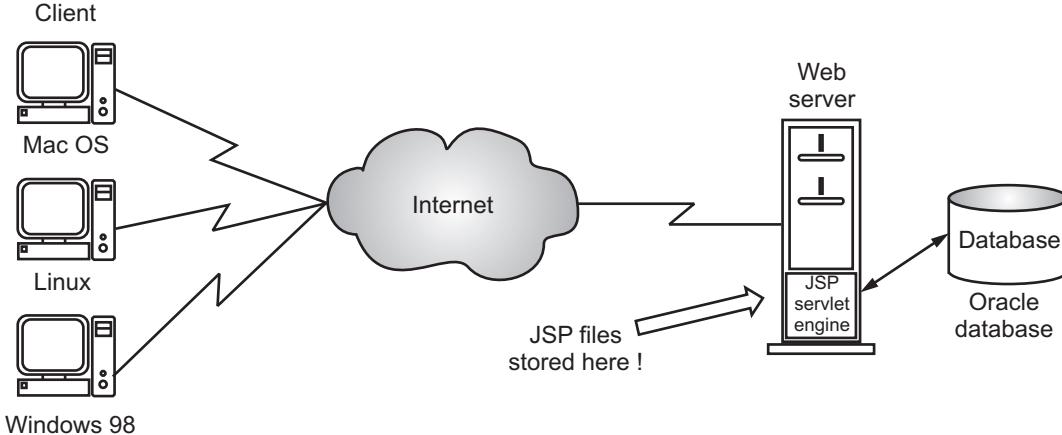
*contd. ...*

|     |                           |                                                                                  |                                                                                       |
|-----|---------------------------|----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| 4.  | Applicable at the time of | They are generally preferred when there is not much processing of data required. | They are generally preferred when there is more processing and manipulation involved. |
| 5.  | Running speed             | JSP runs slower as compared to a Servlet. JSP compiles into Java Servlets.       | Servlets run faster as compared to JSP.                                               |
| 6.  | Code complications        | The code programming is easy as compared to that of Servlets.                    | The code programming is difficult as compared to that of JSP.                         |
| 7.  | Facility                  | Here, we can build custom tags which can directly call Java beans.               | No such facility is available in servlets.                                            |
| 8.  | Consists of               | JSP are Java HTML representation mixed with JAVA scriptlets.                     | Servlet are full functional Java codes.                                               |
| 9.  | Consistence of objects    | JSP has Implicit objects.                                                        | Servlets does not have such type of objects.                                          |
| 10. | Examples                  | To display a report.                                                             | To process a user submitted form.                                                     |

**4.8.1 JSP Architecture**

[April 17]

- The Web server needs a JSP engine i.e., container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages.
- A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.
- Fig. 4.14 shows the position of JSP container and JSP files in a Web Application.

**Fig. 4.14: JSP Architecture**

## 4.8.2 JSP Processing

- The following steps explain how the web server creates the web page using JSP:
  - Step 1:** As with a normal page, the browser sends an HTTP request to the web server.
  - Step 2:** The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
  - Step 3:** The JSP engine loads the JSP page from disk and converts it into an Servlet content. This conversion is very simple in which all template text is converted to println() statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
  - Step 4:** The JSP engine compiles the Servlet into an executable class and forwards the original request to a Servlet engine.
  - Step 5:** A part of the web server called the Servlet engine loads the Servlet class and executes it. During execution, the Servlet produces an output in HTML format, which the Servlet engine passes to the web server inside an HTTP response.
  - Step 6:** The web server forwards the HTTP response to the browser in terms of static HTML content.
  - Step 7:** Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.
- All the above mentioned steps can be shown below in the Fig. 4.15.

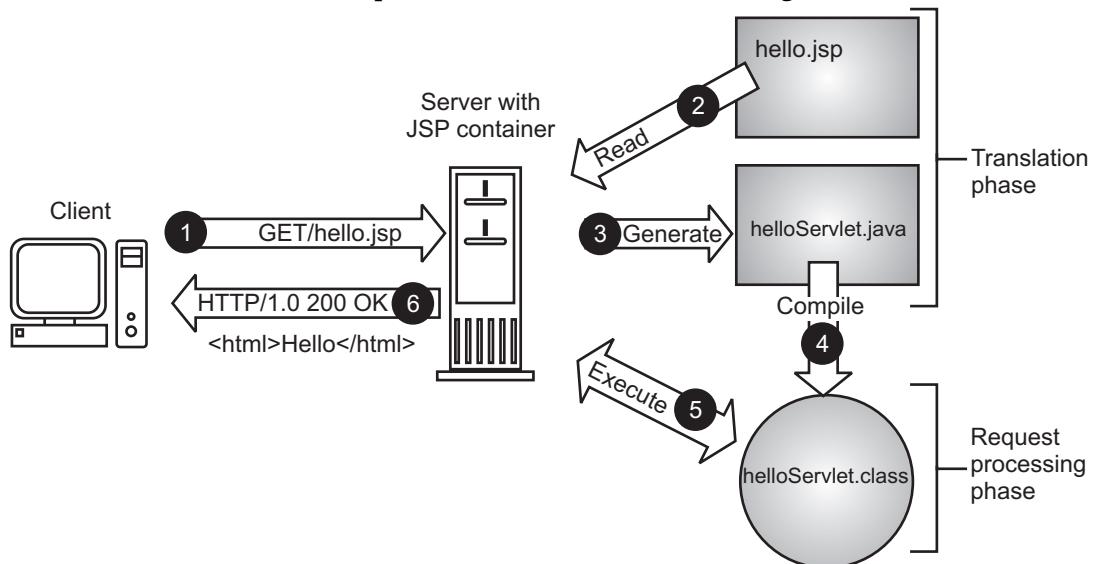


Fig. 4.15: Steps the Web Server creates Web Page using JSP

- Typically, the JSP engine checks to see whether a Servlet for a JSP file already exists and whether the modification date on the JSP is older than the Servlet.
- If the JSP is older than its generated Servlet, the JSP container assumes that the JSP hasn't changed and that the generated Servlet still matches the JSP's contents.
- This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.
- So in a way, a JSP page is really just another way to write a Servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular Servlet.

### 4.8.3 Accessing Model of JSP

- JSP is a server side technology which helps to create a webpage dynamically using java as the programming language.
- JSP is a specification from Sun Microsystems. It is an extension to Servlet API.
- The java server pages can be accessed in two different ways :
  1. Access through a client request.
  2. Access through a Servlet request.
- **Model 1: Architecture or Access through a Client Request:** In this Model, JSP plays a key role and it is responsible for processing the request made by client. Client (Web browser) makes a request; JSP then creates a bean object which then fulfills the request and passes the response to JSP. JSP then sends the response back to client. Unlike Model2 architecture in this Model, most of the processing is done by JSP itself.

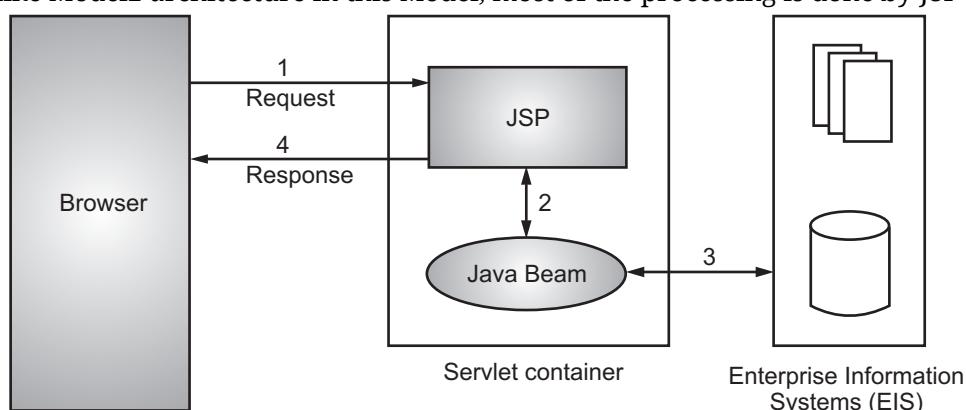


Fig. 4.16: Architecture of Client Request

- **Model 2: Architecture or Access through a Servlet Request:** In this Model Servlet plays a major role and it is responsible for processing the client's (web browser) request. Presentation part (GUI part) will be handled by JSP and it performs this with the help of bean as shown in the Fig. 4.17. The Servlet acts as controller and in charge

of request processing. It creates the bean objects if required by the jsp page and calls the respective JSP page. The JSP handles the presentation part by using the bean object. In this Model, JSP doesn't do any processing; Servlet creates the bean Object and calls the JSP program as per the request made by client.

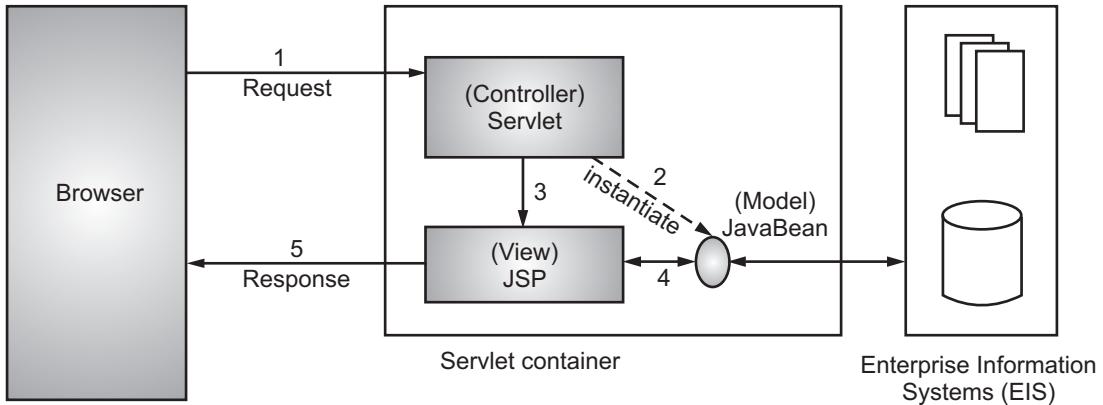


Fig. 4.17: Architecture of Servlet Request

#### 4.8.4 Advantages of JSP

[April 17]

- Following is the list of other advantages of using JSP over other technologies:
  - 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.
  - 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.
  - vs. Server-Side Includes (SSI):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
  - 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.
  - vs. Static HTML:** Regular HTML, of course, cannot contain dynamic information.

#### 4.9 SIMPLE JSP PROGRAM

- Let us start learning with a simple JSP program. All JSP programs are stored as a .jsp files.
- Following is a simple JSP code, MyJSP.jsp which prints Hello, Sample JSP code.
- As discussed JSP is used for creating dynamic webpages. Dynamic webpages can have two types of contents i.e., static and dynamic content.

- The static contents can have text-based formats such as HTML, XML etc and the dynamic contents are generated by JSP elements.

```
<%-- JSP comment --%>
<html>
 <head>
 <title>Message</title>
 </head>
 <body>
 <%out.print("Hello, Sample JSP code");%>
 </body>
</html>
```

#### **Analysis of the above code:**

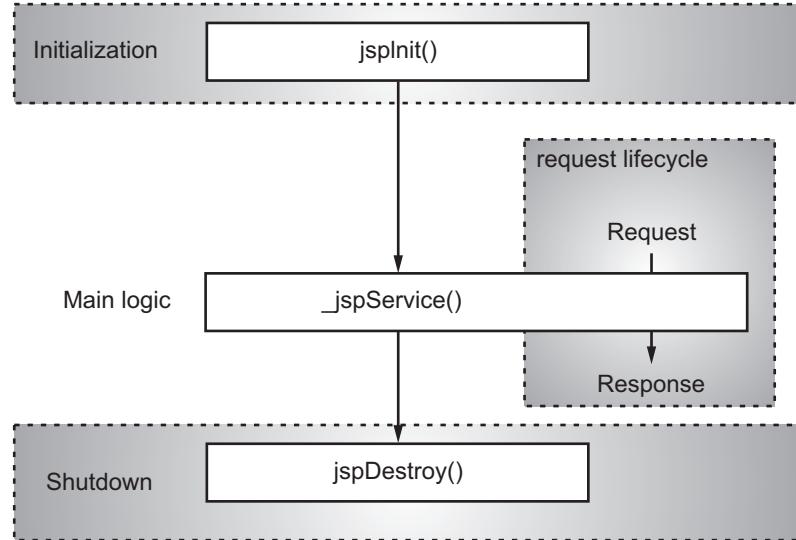
- The line **<%–JSP Comment–%>** represents the JSP element called JSP Comment, While adding comments to a JSP page you can use this tag. JSP Comments must starts with a tag **<%––** and ends with **–%>**.
- HEAD, TITLE and BODY tags are HTML tags:** They are HTML tags, frequently used for static web pages.
- <%out.print(“ Hello, Sample JSP code ”);%>** is a JSP element, which is known as Scriptlet. Scriptlets can contain Java codes. **Syntax** of scriptlet is: **<%Executable java code%>**. As the code in Scriptlets is java statement, they must end with a semicolon(;). **out.print(“ Hello, Sample JSP code ”)** is a java statement, which prints“ Hello, Sample JSP code”.

#### **4.10 LIFE CYCLE OF JSP**

[April 16, 17, 19, Oct. 17, 18]

- 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.
- The following are the paths followed by a JSP:
  - Compilation,
  - Initialization,
  - Execution, and
  - Cleanup.

- The four major phases of JSP life cycle are very similar to Servlet Life Cycle as shown in Fig. 4.18.



**Fig. 4.18: Life cycle of JSP**

- Fig. 4.18 shows following parts of a JSP page:

### 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.
- The compilation process involves three steps:
  - Parsing the JSP.
  - Turning the JSP into a servlet.
  - Compiling the servlet.

### 2. JSP Initialization:

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

```

public void jspInit()
{
 // Initialization code...
}

```

- 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 `jspInit()` 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 `_jspService()` method in the JSP.
- The `_jspService()` method takes an `HttpServletRequest` and an `HttpServletResponse` as its parameters as follows:

```

void _jspService(HttpServletRequest request,
 HttpServletResponse response)
{
 // Service handling code...
}
```

- The `_jspService()` method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods i.e. GET, POST, DELETE etc.

### 4. JSP Cleanup:

- The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
- The `jspDestroy()` method is the JSP equivalent of the `destroy` method for servlets.
- Override `jspDestroy()` when you need to perform any cleanup, such as releasing database connections or closing open files.

## 4.11 JSP IMPLICIT OBJECTS

[Oct. 17, April 18]

- 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.
- JSP Implicit Objects are also called pre-defined variables.
- There are total nine implicit objects available in JSP as given in following Table:

| Sr. No. | Object                | Description                                                                                     |
|---------|-----------------------|-------------------------------------------------------------------------------------------------|
| 1.      | <code>request</code>  | This is the <code>HttpServletRequest</code> object associated with the request.                 |
| 2.      | <code>response</code> | This is the <code>HttpServletResponse</code> object associated with the response to the client. |
| 3.      | <code>out</code>      | This is the <code>PrintWriter</code> object used to send output to the client.                  |
| 4.      | <code>session</code>  | This is the <code>HttpSession</code> object associated with the request.                        |

*contd. ...*

|    |                          |                                                                                                             |
|----|--------------------------|-------------------------------------------------------------------------------------------------------------|
| 5. | <code>application</code> | This is the ServletContext object associated with application context.                                      |
| 6. | <code>config</code>      | This is the ServletConfig object associated with the page.                                                  |
| 7. | <code>pageContext</code> | This encapsulates use of server-specific features like higher performance JspWriters.                       |
| 8. | <code>page</code>        | This is simply a synonym for this, and is used to call the methods defined by the translated servlet class. |
| 9. | <code>Exception</code>   | The Exception object allows the exception data to be accessed by designated JSP.                            |

**4.12 SCRIPTING ELEMENTS OF JSP****[April 18, 19, Oct. 17]**

- 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 script may contain HTML code. This document can be developed in any editor with .html or .html file. We can directly use it as a .jsp file.
- The JSP page is built up using the following components :
  - Directives,
  - Declarations,
  - Scriptlets,
  - Expressions,
  - Standard Actions, and
  - Custom Tags.

**4.12.1 Declarations****[April 18]**

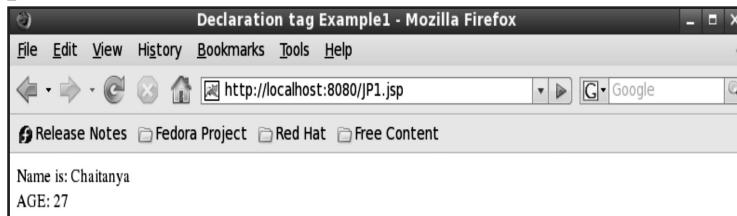
- A declaration declares one or more variables or methods that we can use in Java code later in the JSP file.
- We must declare the variable or method before you use it in the JSP file.
- Following is the **syntax** of JSP Declarations:  
`<%! declaration;[ declaration;]+... %>`
- We can write XML equivalent of the above **syntax** as follows:  
`<jsp:declaration>  
 code fragment...  
</jsp:declaration>`
- Following is the simple example for JSP Declarations:  
`<%!int i =0; %>  
<%!int a, b, c; %>  
<%!Circle a =newCircle(2.0); %>`

**Example 1: Variables Declaration:**

- In this example we have declared two variables inside declaration tag and displayed them on client using expression tag.

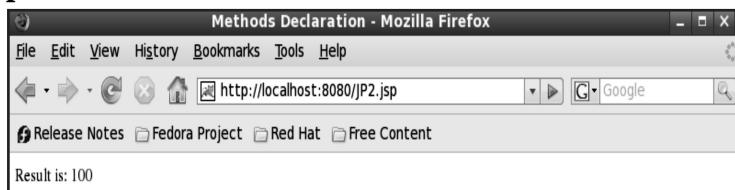
```
<html>
 <head>
 <title>Declaration tag Example1</title>
 </head>
 <body>
 <%! String name="Chaitanya"; %>
 <%! int age=27; %>
 <%= "Name is: " + name %>

 <%= "AGE: " + age %>
 </body>
</html>
```

**Output:****Example 2: Methods Declaration:**

- In this example we have declared a method sum using JSP declaration tag.

```
<html>
 <head>
 <title>Methods Declaration</title>
 </head>
 <body>
 <%!
 int sum(int num1, int num2, int num3)
 {
 return num1+num2+num3;
 }
 %>
 <%= "Result is: " + sum(10,40,50) %>
 </body>
</html>
```

**Output:****4.12.2 Expressions**

[April 16]

- 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.
- Following is the **syntax** of JSP Expression:

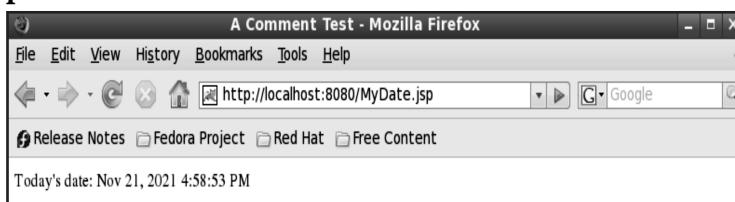
```
<%= expression %>
```

- We can write XML equivalent of the above **syntax** as follows:

```
<jsp:expression>
 expression
</jsp:expression>
```

- Following is the simple example for JSP Expression:

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

**Output:**

### 4.12.3 Scriptlets

[April 16, Oct. 18]

- 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.
- For example, any Java if/for/while blocks opened in one scriptlet element must be correctly closed in a later element for the page to successfully compile. Markup which falls inside a split block of code is subject to that code, so markup inside an if block will only appear in the output when the if condition evaluates to true; likewise, markup inside a loop construct may appear multiple times in the output depending upon how many times the loop body runs.

- The following would be a valid for loop in a JSP page:

```
<p>Counting to three:</p>
<% for(int i=1; i<4; i++){ %>
<p>This number is <%= i %> </p>
<% } %>
<p>OK</p>
```

- The output displayed in the user's web browser would be:

```
Counting to three:
This number is 1.
This number is 2.
This number is 3.
OK.
```

### 4.12.4 Comments

[April 17]

- JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.
- Following is the **syntax** of JSP comments:  

```
<%-- This is JSP comment ... --%>
```
- Following is the simple example for JSP Comments:

```
<html>
 <head><title>A Comment Test</title></head>
 <body>
```

```

<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page source --%>
</body>
</html>

```

**Output:** A Test of Comments

- There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary:

| Syntax            | Purpose                                                 |
|-------------------|---------------------------------------------------------|
| <%-- comment --%> | A JSP comment. Ignored by the JSP engine.               |
| <!-- comment -->  | An HTML comment. Ignored by the browser.                |
| <\%               | Represents static <% literal.                           |
| %\>               | Represents static %> literal.                           |
| \'                | A single quote in an attribute that uses single quotes. |
| \"                | A double quote in an attribute that uses double quotes. |

## 4.13 JSP DIRECTIVES

[April 17]

- JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing.
- JSP directive tag gives special information about the page to the JSP engine.
- A JSP directive affects the overall structure of the servlet class. It usually has the following **form/syntax**:

```
<%@ directive attribute="value" %>
```

- Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.
- The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.
- There are three types of directive tag:

| Directive          | Description                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------|
| <%@ page ... %>    | Defines page-dependent attributes, such as scripting language, error page, and buffering requirements. |
| <%@ include ... %> | Includes a file during the translation phase.                                                          |
| <%@ taglib ... %>  | Declares a tag library, containing custom actions, used in the page.                                   |

### 4.13.1 Page Directive

[Oct. 17]

- The page directive is used to provide instructions to the container that pertain to the current JSP page.
- We may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.
- Following is the basic **syntax** of page directive:  
`<%@ page attribute="value" %>`
- We can write XML equivalent of the above **syntax** as follows:  
`<jsp:directive.page attribute="value"/>`
- Attributes:** Following is the list of attributes associated with page directive:

| Attribute          | Purpose                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------|
| buffer             | Specifies a buffering model for the output stream.                                                             |
| autoFlush          | Controls the behavior of the servlet output buffer.                                                            |
| contentType        | Defines the character encoding scheme.                                                                         |
| errorPage          | Defines the URL of another JSP that reports on Java unchecked runtime exceptions.                              |
| isErrorPage        | Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.                       |
| extends            | Specifies a superclass that the generated servlet must extend.                                                 |
| import             | Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes. |
| info               | Defines a string that can be accessed with the servlet's getServletInfo() method.                              |
| isThreadSafe       | Defines the threading model for the generated servlet.                                                         |
| language           | Defines the programming language used in the JSP page.                                                         |
| session            | Specifies whether or not the JSP page participates in HTTP sessions.                                           |
| isELIgnored        | Specifies whether or not EL expression within the JSP page will be ignored.                                    |
| isScriptingEnabled | Determines if scripting elements are allowed for use.                                                          |

### 4.13.2 Include Directive

[April 19]

- The include directive is used to includes a file during the translation phase.
- This directive tells the container to merge the content of other external files with the current JSP during the translation phase.

- You may code include directives anywhere in your JSP page.
- The general usage **form/syntax** of this directive is as follows:  
`<%@ include file="relative url">`
- The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.
- You can write XML equivalent of the above **syntax** as follows:  
`<jsp:directive.includefile="relative url"/>`

**taglib Directive:**

- The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.
- The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.
- The taglib directive follows the following **syntax**:

```
<%@ taglib uri="uri" prefix="prefixOfTag">
```

Where, the uri attribute value resolves to a location the container understands and the prefix attribute informs a container what bits of markup are custom actions.

## **4.14 MIXING SCRIPTLETS AND HTML**

- We have already seen how to use the "out" variable to generate HTML output from within a scriptlet.
- For more complicated HTML, using the out variable all the time loses some of the advantages of JSP programming. It is simpler to mix scriptlets and HTML.
- Suppose we have to generate a table in HTML. This is a common operation, and you may want to generate a table from a SQL table, or from the lines of a file.
- But to keep our example simple, we will generate a table containing the numbers from 1 to N. Not very useful, but it will show you the technique.
- Here, is the JSP fragment to do it:

```
<table border=2>
<%
 for(int i = 0; i < n; i++)
 {
%
 <tr>
 <td>Number</td>
```

```

<td><%= i+1 %></td>
</tr>
<%
}
%>
</table>
```

- We would have to supply an int variable "n" before it will work, and then it will output a simple table with "n" rows.
- The important things to notice are how the %> and <% characters appear in the middle of the "for" loop, to let you drop back into HTML and then to come back to the Scriptlet.
- The concepts are simple here -- as we can see, you can drop out of the scriptlets, write normal HTML, and get back into the Scriptlet.
- Any control expressions such as a "while" or a "for" loop or an "if" expression will control the HTML also. If the HTML is inside a loop, it will be emitted once for each iteration of the loop.
- Another example of mixing Scriptlets and HTML is shown below -- here it is assumed that there is a Boolean variable named "hello" available. If we set it to true, you will see one output, if we set it to false, we will see another output.

```

<%
if(hello) {
 %>
 <p>Hello, world
 <%
} else {
 %>
 <p>Goodbye, world
 <%
}
%>
```

- It is a little difficult to keep track of all open braces and scriptlet start and ends, but with a little practice and some good formatting discipline, you will acquire competence in doing it.

## 4.15 JSP ACTIONS

- JSP actions use constructs in XML syntax to control the behavior of the Servlet engine. The JSP specification provides a standard tag called Action tag used within JSP code.

- Standard actions are predefined JSP elements that implement specific functionality. As is the case with all actions (or tags), they follow the rules for XML, and thus, they do not have alternative syntactical representations. With each new JSP specification, the list of standard actions increases, as do the actions complexity. Currently, standard actions can be used to perform dynamic resource inclusion, include applets, forward response-processing requests and include JavaBeans. Standard actions use an EML-like syntax with a start tag and end tag. Some actions have body content, while others only have attributes.
- There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks. The action tags are used to control the flow between pages.
- Action elements are predefined functions to carry out some actions in a JSP page such as:
  - Accessing Java applet code inside JSP page.
  - Using the Java Bean components.
  - Including the result of the external resources such as other JSP page, HTML and Servlet.
  - Forwarding the request to other JSP page and so on.
- Scripting elements are used during the request processing phase whereas action elements are used in the translation phase. As it is used at the translation phase, it can control the behaviour of the Servlet engine. Unlike directives, actions are re-evaluated each time the page is accessed. Each standard tag or action element must start with `jsp:prefix` and the Action elements work in both JSP document and in the regular JSP page.
- There is following only one **syntax** for the Action element, as it conforms to the XML standard.

```
<jsp:action_name attribute = "value" />
```

- Action elements are basically predefined functions. The available JSP actions are given below:
  1. **jsp:include** action includes a file at the time the page is requested.
  2. **jsp:useBean** action finds or instantiates a JavaBean.
  3. **jsp:setProperty** action sets the property of a JavaBean.
  4. **jsp:getProperty** action inserts the property of a JavaBean into the output.
  5. **jsp:forward** action forwards the requester to a new page.
  6. **jsp:plugin** action generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
  7. **jsp:element** action defines XML elements dynamically.
  8. **jsp:attribute** action defines dynamically-defined XML element's attribute.
  9. **jsp:body** action defines dynamically-defined XML element's body.
  10. **jsp:text** action used to write template text in JSP pages and documents.

- Two basic attributes are commonly used for all action tags are given below:
  - id:** The id attribute defines unique action elements and allows actions to be referenced within the JSP page. When the action creates an object's instance, the id attribute is used to refer to it.
  - Scope:** The scope attribute is used for identifying an action's life cycle. It correlates with the id attribute because the scope attribute is used to establish that particular object's lifespan associated with the ID.

### 4.15.1 jsp:forward Action

- The JSP forward action terminates the current page and allows us to forward the request to other resources.
- The forward action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.
- The jsp:forward action can be used to dispatch a request to a new source. In effect, the JSP container stops processing the current JSP page and starts processing the new resource.
- The **syntax** of the jsp:forward action is given below:
 

```
<jsp:forward page = "Relative URL" />
```
- The page attribute is the URL for the resource that will handle the forwarded request. Relative paths are evaluated relative to the current JSP page. The value of the page attribute can be dynamically evaluated at run-time.
- Example:** Let us use two files namely, date.jsp and main.jsp.

#### date.jsp

```
<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
```

#### main.jsp

```
<html>
 <head>
 <title>Example of jsp:forward Action</title>
 </head>
 <body>
 <h2>Example of jsp:forward Action</h2>
 <jsp:forward page = "date.jsp" />
 </body>
</html>
```

- Now keep above both files in the root directory and try to access main.jsp. This would display output something like as below.
- Here, it discarded the content from the main page and displayed the content from forwarded page only.

---

Today's date: 12-Sep-2021 14:54:22

---

## 4.15.2 jsp:include Action

- The JSP include action allows us to include another resource in the page being generated. This action lets us to insert files into the page being generated.

- The **syntax** of the jsp:include action is given below:

```
<jsp:include page = "relative URL" flush = "true" />
```

- In page attribute the relative URL of the page to be included while the flush attribute determines whether the included resource has its buffer flushed before it is included.

- Example:** Let us use two files namely, date.jsp and main.jsp.

### date.jsp

```
<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
```

### main.jsp

```
<html>
 <head>
 <title>Example of jsp:include Action</title>
 </head>
 <body>
 <h2>Example of jsp:include Action</h2>
 <jsp:include page = "date.jsp" flush = "true" />
 </body>
</html>
```

- Now keep above both files in the root directory and try to access main.jsp we will receive the following output:

## Example of jsp:include Action

Today's date: 12-Sep-2021 14:54:22

## 4.15.3 jsp:useBean Action

- The JSP useBean action is used for creating or locating bean objects.
- The JSP useBean action is used for creating or locating bean objects. The jsp:useBean action is used within a JSP page to create a new scripting variable that
- The **syntax** of the jsp:useBean action is given below:

```
<jsp:useBean id = "name" class = "package.class" />
```

The class attribute designates the full package name of the bean.

The type of attribute defines the actual type of scripting variable that is created.

The beanName attribute gives the name of the bean.

The id attribute is a case-sensitive name used to identify the object instantiated from the JavaBean class. This object is assigned to a scripting variable within the current JSP page that can then be used to directly access the JavaBean.

The scope attribute defines the visibility of the new instantiated object. This attribute can only be assigned one of the four legal scope values define positively.

- Once, a bean class is loaded, we can use `jsp:setProperty` and `jsp:getProperty` actions to modify and retrieve the bean properties.
- Example:** Let us use two files namely, `calculator.java` and `main.jsp`.

#### date.jsp

```
package com.javatpoint;
public class Calculator
{
 public int cube(int n){return n*n*n;}
}
```

#### main.jsp

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>
<%
 int m=obj.cube(3);
 out.print("cube of 3 is "+m);
%>
```

#### Output:

cube of 3 is 27

### 4.15.4 `jsp:setProperty` and `jsp:getProperty` Actions

- The `setProperty` and `getProperty` action tags/elements are used for developing web application with Java Bean.
- In Web development, bean class is mostly used because it is a reusable software component that represents data.
- The `jsp:setProperty` action tag sets a property value or values in a bean using the setter method. The `jsp:getProperty` action tag returns the value of the property.
- The `jsp:setProperty` action sets the properties of a Bean. The Bean must have been previously defined before this action.
- There are two basic ways to use the `setProperty` action:
  - In first context we can use `jsp:setProperty` after, but outside of a `jsp:useBean` element, as given below:

```
<jsp:useBean id = "myName" ... />
...
<jsp:setProperty name = "myName" property = "someProperty" .../>
```

In above case, the `jsp:setProperty` is executed regardless of whether a new bean was instantiated or an existing bean was found.

- A second context in which `jsp:setProperty` can appear is inside the body of a `jsp:useBean` element, as given below:

```
<jsp:useBean id = "myName" ... >
...
<jsp:setProperty name = "myName" property = "someProperty" .../>
</jsp:useBean>
```

Here, the `jsp:setProperty` is executed only if a new object was instantiated, not if an existing one was found.

- Attributes associated with the `setProperty` action:
  1. The `name` attribute is the name of the Bean as specified by the `jsp:useBean id` attribute.
  2. The `value` attribute is the actual value that should be assigned to the specific property of the name Bean. The `value` attribute can be used to assign dynamically evaluated expressions and cannot be used in conjunction with the `param` attribute.
  3. The `param` attribute names a request parameter that should be used as the value assigned to the specific property of the named Bean. The `param` attribute can be used in conjunction with the `value` attribute.
  4. The `property` attribute is the name of the Bean property (or variable) that should be modified. Using the wildcard character `*` as the value of the `property` attribute causes the JSP container to match request parameters directly to identically named Bean properties.
- To access the properties of a JavaBean, you use the `jsp:getProperty` action, which converts the property to a String value and places it into the output stream.
- The `getProperty` action is used to retrieve the value of a given property and converts it to a string, and finally inserts it into the output.
- The **syntax** of the `getProperty` action is given below:

```
<jsp:useBean id = "myName" ... />
...
<jsp:getProperty name = "myName" property = "someProperty" .../>
```
- Attributes associated with the `getProperty` action:
  1. The `name` attribute shows the name of the Bean that has a property to be retrieved. The Bean must have been previously defined.
  2. The `property` attribute is the name of the Bean property to be retrieved.

- **Example:** We define a test bean.

```
/* File: TestBean.java */
package action;
public class TestBean
{
 private String message = "No message specified";
 public String getMessage()
 {
 return(message);
 }
 public void setMessage(String message)
 {
 this.message = message;
 }
}
```

- Compile the above code to the generated TestBean.class file and make sure that we copied the

TestBean.class in C:\apache-tomcat-7.0.2\webapps\WEB-INF\classes\action folder and the CLASSPATH variable should also be set to this folder.

- Now we use the following code in main.jsp file. This loads the bean and sets/gets a simple String parameter:

```
<html>
 <head>
 <title>Use of JavaBeans in JSP</title>
 </head>
 <body>
 <h2>Using JavaBeans in JSP</h2>
 <jsp:useBean id = "test" class = "action.TestBean" />
 <jsp:setProperty name = "test" property = "message"
 value = "Hello JSP..." />
 <p>Got Message....</p>
 <jsp:getProperty name = "test" property = "message" />
 </body>
</html>
```

- To access main.jsp, it would display the following output:

## Use of JavaBeans in JSP

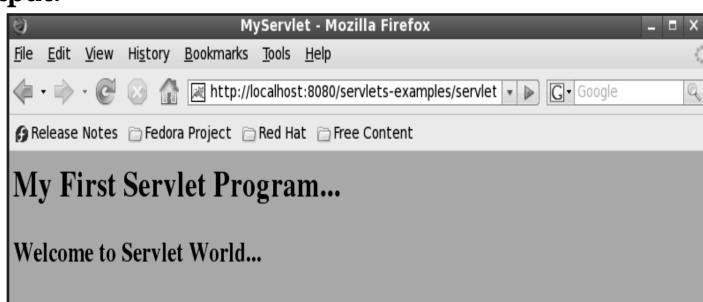
Got Message....  
Hello JSP...

## ADDITIONAL PROGRAMS

**Program 1:** Program for displaying the message using Servlet.

```
// Firstservlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Firstservlet extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res)
 throws IOException, ServletException
 {
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 out.println("<html>");
 out.println("<head>");
 out.println("<title>MyServlet</title>");
 out.println("</head>");
 out.println("<body bgcolor=\"cyan\">");
 out.println("<h1> My First Servlet Program...</h1>");
 out.println("<h2>Welcome to Servlet World...</h2>");
 out.println("</body>");
 out.println("</html>");
 }
}
```

**Output:**

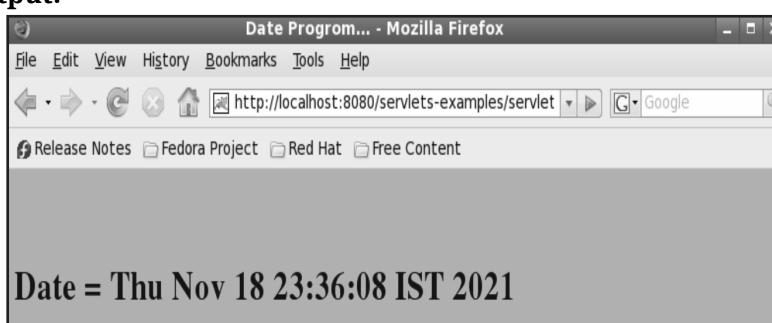


**Program 2:** Program for displaying the Date using Servlet.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyDate extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res)
 throws IOException, ServletException
 {
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 out.println("<html>");
 out.println("<head>");
 out.println("<title>Date Program...</title>");
 out.println("</head>");
 out.println("<body bgcolor=\"cyan\">");
 Date d = new Date();
 out.println("

<h1>Date = "+d+"</h1>");
 out.println("</body>");
 out.println("</html>");
 }
}
```

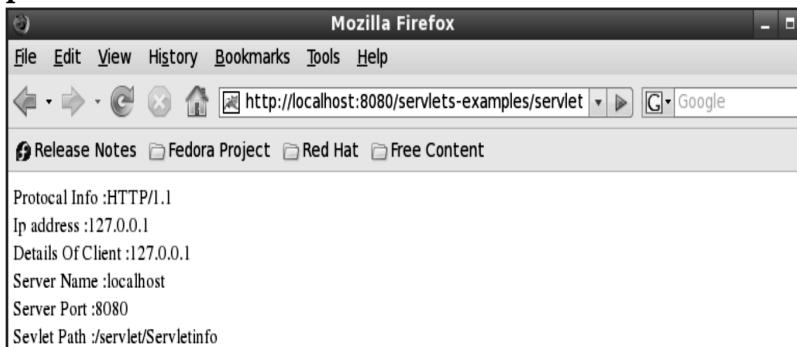
**Output:**



**Program 3:** Program to displaying the Servlet information.

```
// Servletinfo.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servletinfo extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res)
 throws ServletException,IOException
 {
 res.setContentType("text/html");
 PrintWriter out=res.getWriter();
 out.println("Protocal Info :" +req.getProtocol());
 out.println("
Ip address :" +req.getRemoteAddr());
 out.println("
Details Of Client :" +req.getRemoteHost());
 out.println("
Server Name :" +req.getServerName());
 out.println("
Server Port :" +req.getServerPort());
 out.println("
Sevlet Path :" +req.getServletPath());
 }
}
```

**Output:**



**Program 4:** Program for displaying the sum of two numbers gets the inputs from user.

```
//Sum.html
<html>
<head>
<title>Addtion</title>
</head>
```

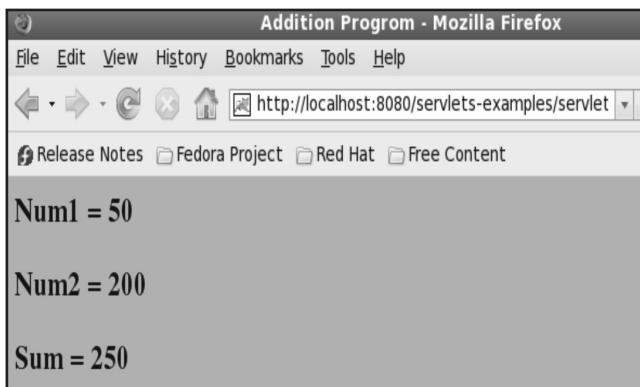
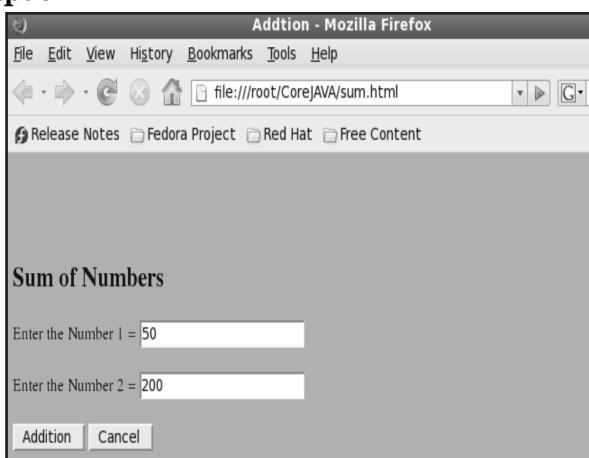
```
<body bgcolor = "cyan">
<form action = "http:
 //localhost:8080/servlets-examples/servlet/AddNumServlet" method ="GET">

<h2> Sum of Numbers </h2>
Enter the Number 1 = <input type = "text" name = "n1">

Enter the Number 2 = <input type = "text" name = "n2">

<input type = "submit" value = "Addition ">
<input type = "reset" value = "Cancel ">
</form>
</body>
</html>
// AddNumServlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AddNumServlet extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res)
 throws IOException, ServletException
 {
 int num1 = Integer.parseInt(req.getParameter("n1"));
 int num2 = Integer.parseInt(req.getParameter("n2"));
 int add = num1 + num2;
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 out.println("<html>");
 out.println("<head>");
 out.println("<title>Addition Program</title>");
 out.println("</head>");
 out.println("<body bgcolor=\"cyan\">");
 out.println("<h2> Num1 = "+num1+"</h2>");
 out.println("<h2> Num2 = "+num2+"</h2>");
 out.println("<h2> Sum = "+add+"</h2>");
 }
}
```

```
 out.println("</body>");
 out.println("</html>");
 }
}
```

**Output:**

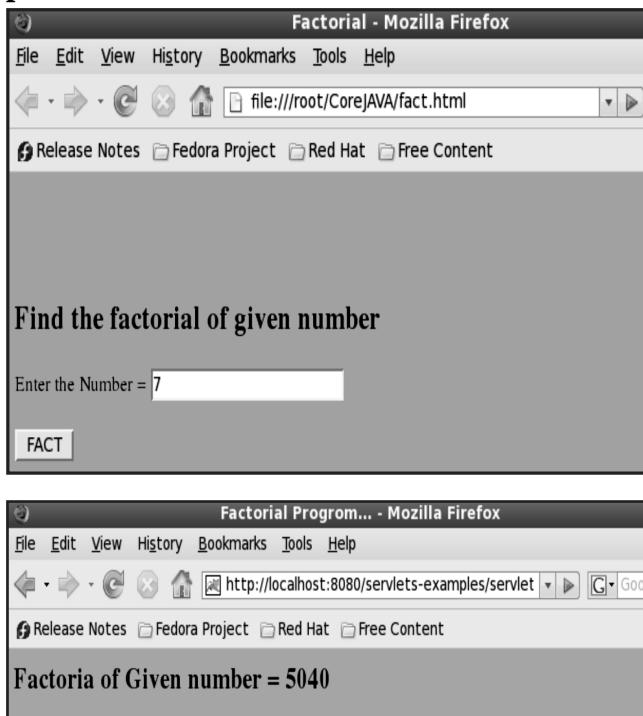
**Program 5:** Program to find the factorial of given number using Servlet program.

```
//fact.html
<html>
<head>
<title>Factorial</title>
</head>
<body bgcolor = "cyan">
<form action = "http://localhost:8080/servlets-examples/
servlet/Fact servlet" method ="GET">

<h2> Find the factorial of given number </h2>
```

```
Enter the Number = <input type = "text" name = "txt">

<input type = "submit" value = "FACT">
</form>
</body>
</html>
//Fact servlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Fact servlet extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res)
 throws IOException, ServletException
 {
 int no = Integer.parseInt(req.getParameter("txt"));
 int i,f=1;
 for(i=1;i<=no;i++)
 {
 f = f * i;
 }
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 out.println("<html>");
 out.println("<head>");
 out.println("<title> Factorial Program...</title>");
 out.println("</head>");
 out.println("<body bgcolor=\"cyan\">");
 out.println("<h2> Factorial of Given number = "+f+"</h2>");
 out.println("</body>");
 out.println("</html>");
 }
}
```

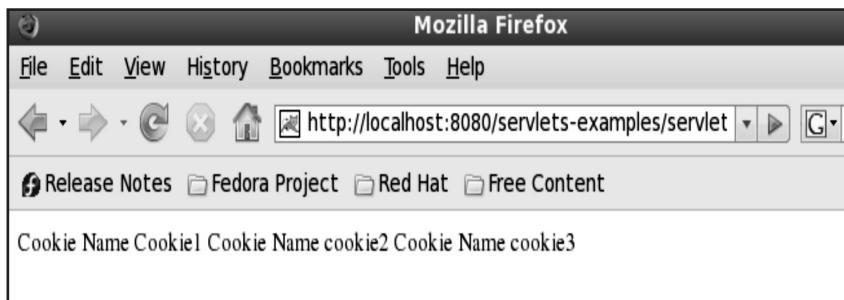
**Output:**

**Program 6:** Program of add Cookie and get the information of Cookies.

```
//Addcookie.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Addcookie extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res)
 throws ServletException,IOException
 {
 Cookie c1 = new Cookie("Cookie1","1");
 res.addCookie(c1);
 res.setContentType("text/html");
 PrintWriter pw = res.getWriter();
 pw.println("Cookie added with value 1");
 Cookie c2 = new Cookie("cookie2","2");
 res.addCookie(c2);
```

```
 pw.println("Cookie added with value 2");
 Cookie c3 = new Cookie("cookie3","3");
 res.addCookie(c3);
 pw.println("Cookie added with value 3");
 }
}

// Getcookie.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Getcookie extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res)
 throws ServletException,IOException
 {
 res.setContentType("text/html");
 PrintWriter out=res.getWriter();
 Cookie c[] = req.getCookies();
 for(int i=0;i<c.length;i++)
 out.println("Cookie Name "+c[i].getName());
 }
}
```

**Output:**

**Program 7:** Program to display login successfully or not get the input from user and check with respect to the admin table (use the Postgresql database).

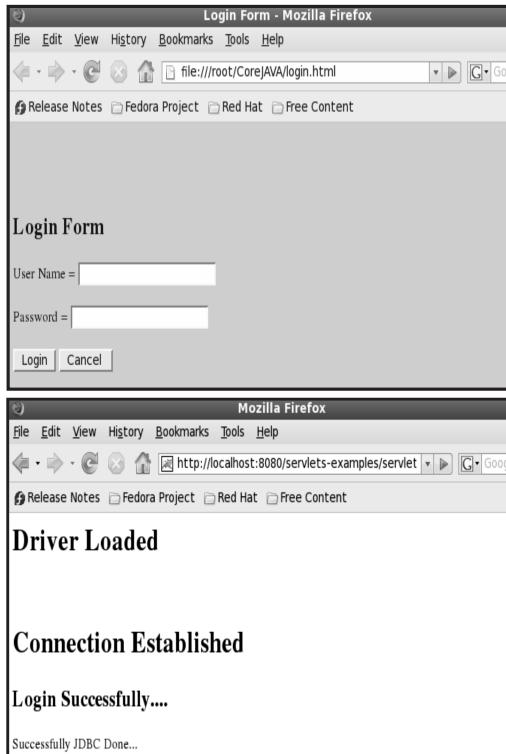
```
//Login.html
<html>
<head>
 <title>Login Form</title>
</head>
<body bgcolor = "pink">
 <form action = "http://localhost:8080/
 servlets-examples/servlet/LoginServlet" method ="GET">

 <h2> Login Form</h2>
 User Name = <input type = "text" name = "us">

 Password = <input type = "text" name = "pd">

 <input type = "submit" value = "Login">
 <input type = "reset" value = "Cancel ">
 </form>
</body>
</html>
// LoginServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LoginServlet extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res) throws
 IOException, ServletException
 {
 String uname = req.getParameter("us");
 String pwd = req.getParameter("pd");
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 try
 {
 out.println("<html>");
 out.println("<body>");
 Class.forName("postgresql.Driver");
 out.println("<h1>Driver Loaded</h1>");
 Connection c=DriverManager.getConnection
 ("jdbc:postgresql:mj","postgres","");
 }
 }
}
```

```
 out.println("
<h1>Connection Established</h1>");
 Statement st=c.createStatement();
 ResultSet rs=st.executeQuery("select * from admin");
 while(rs.next())
 {
 if(uname.equals(rs.getString(1))&&pwd.equals(rs.getString(2)))
 out.println("<h2>Login Successfully....</h2>");
 else
 out.println("<h3> NOT ! Try Again...</h3>");
 }
 }
 catch(Exception e)
 {
 out.println("error"+e);
 }
 out.println("Successfully JDBC Done...");
 out.println("</body>");
 out.println("</html>");
}
}
```

**Output:**

**Program 8:** Program for Servlet session.

```
//Sess.html
<html>
 <head>
 <title>Login Form</title>
 </head>
 <body bgcolor = "purple">
 <form action = "http://localhost:8080/servletsexamples
 /servlet/Sess1" method ="GET">

 <h2> Login Form</h2>
 User Name = <input type = "text" name = "us">

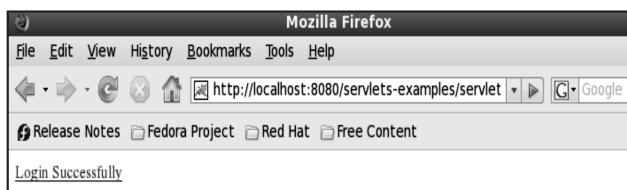
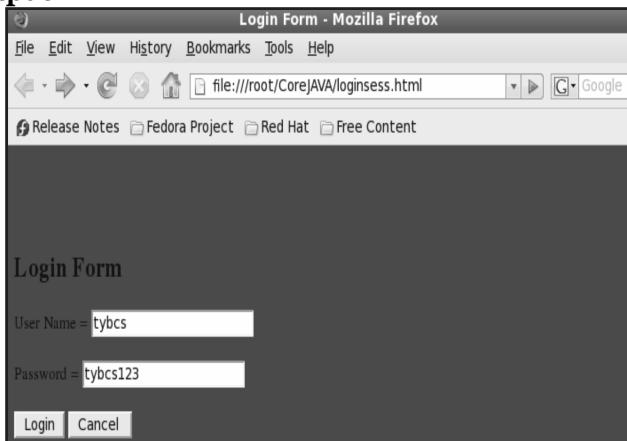
 Password = <input type = "text" name = "pd">

 <input type = "submit" value = "Login">
 <input type = "reset" value = "Cancel ">
 </form>
 </body>
</html>
// Sess1.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Sess1 extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res) throws
 IOException, ServletException
 {
 HttpSession hs = req.getSession(true);
 String uname = req.getParameter("us");
 String pwd = req.getParameter("pd");
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 if(uname.equals("tybcs")&&pwd.equals("tybcs123"))
 {
 out.println("<a href=
http://localhost:8080/servletsexamples/servlet/Sess.html> Login
Successfully ");
 hs.setAttribute("LoginID",uname);
 }
 }
}
```

```

 else
 {
 hs.setAttribute("LoginID",pwd);
 }
 }
}

```

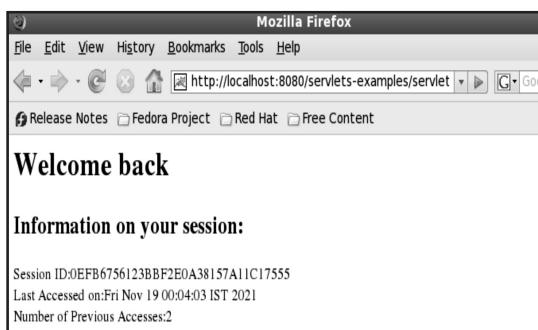
**Output:****Program 9:** Program for session.

```

// Webcount.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class Webcount extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res) throws
 ServletException,IOException
 {
 Integer count;
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 HttpSession session = req.getSession(true);
 String heading;
 count = (Integer) session.getAttribute("accessCount");

```

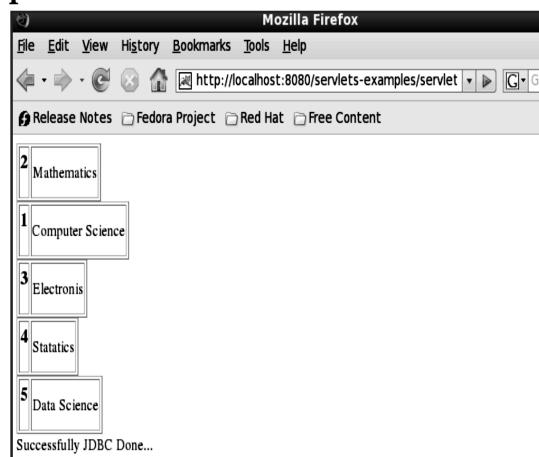
```
if(count==null)
{
 count=new Integer(0);
 heading="Welcome,newcomer";
}
else
{
 heading="Welcome back";
 count=new Integer((count.intValue())+1);
}
session.setAttribute("accessCount",count);
out.println("<BODY><HTML>"+<H1>" +heading+"</H1>\n"+<H2>Information
on your session:</H2>\n"+Session ID:"+session.getId()+"
Last
Accessed on:"+new Date(session.getLastAccessedTime())+"
Number of
Previous Accesses:"+count+"
</BODY></HTML>");
}
public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
{
 doGet(req,res);
}
}
```

**Output:**

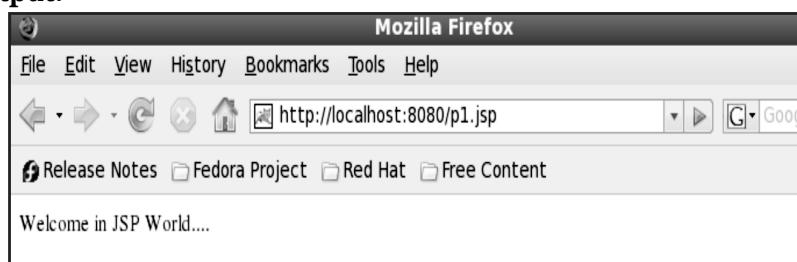
**Program 10:** Program to displaying the information of Department which store in the database (Servlet and Postgresql).

```
// Jdbcserver.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Jdbcserver extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res) throws
 IOException, ServletException
 {
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
 try
 {
 out.println("<html>");
 out.println("<body>");
 Class.forName("postgresql.Driver");
 out.println("<h1>driver loaded</h1>");
 Connection c=DriverManager.getConnection
 ("jdbc:postgresql:serv","postgres","");
 out.println("
<h1>connection created</h1>");
 Statement st=c.createStatement();
 ResultSet rs=st.executeQuery("select * from department");
 while(rs.next())
 {
 out.println("<table border= 1>");
 out.println("<tr>");
 out.print("<td><h3>" + rs.getInt(1) +
 "</td><td>" + rs.getString(2) + "</td></h3>");
 out.println("</tr>");
 out.println("</table>");
 }
 }
 }
}
```

```
 catch(Exception e)
 {
 out.println("error"+e);
 }
 out.println("Successfully JDBC Done...");
 out.println("</body>");
 out.println("</html>");
 }
}
```

**Output:****Program 11:** JSP program to display the string.

```
<html>
 <body>
 <%
 out.println("Welcome in JSP World....");
 %>
 </body>
</html>
```

**Output:**

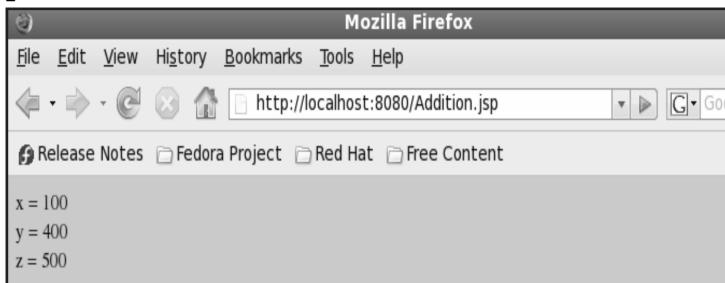
**Program 12:** JSP program for addition of two numbers.

```
<html>
 <body bgcolor="cyan">
 <%! int x,y,z;%>
 <%
 x = 100;
 y = 400;
 z = x + y;
 %>
 <%= "x = "+ x%>

 <%= "y = "+ y%>

 <%= "z = "+ z %>
 </body>
</html>
```

**Output:**



**Program 13:** Program to get the input from user and display the multiplication using JSP.

```
//Mult.html
<html>
 <body bgcolor="cyan">

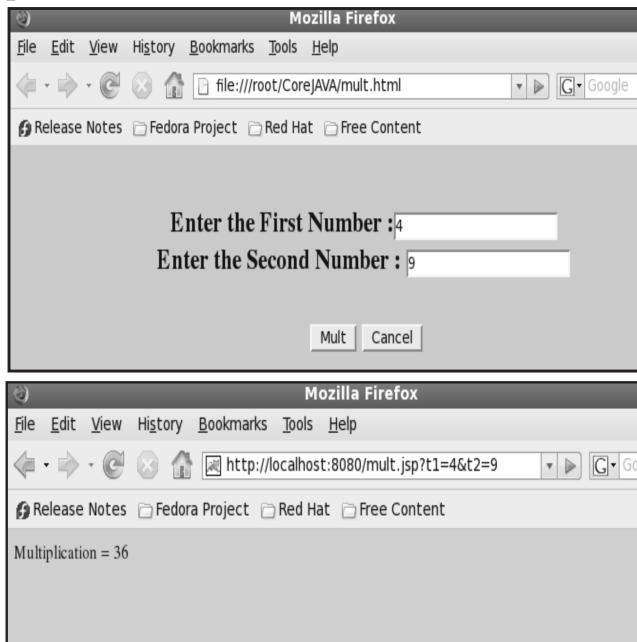
 <form action="http://localhost:8080/mult.jsp" method="GET">
 <center>
 <h2>Enter the First Number :<input type = text name = "t1">

Enter the Second Number : <input type = text name= "t2">

 <input type="Submit" value="Mult">
 <input type="reset" value="Cancel">
 </form>
```

```
</body>
</html>
//Mult.jsp
<%@ page language="java"%>
<html>
<body bgcolor="pink">
<%
 int x = Integer.parseInt(request.getParameter("t1"));
 int y = Integer.parseInt(request.getParameter("t2"));

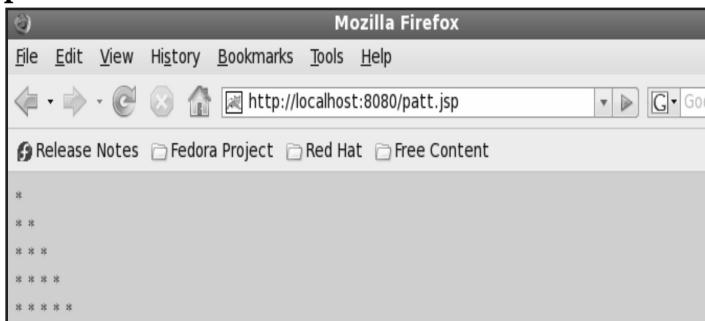
 int z = x * y;
 out.println("Multiplication = "+z);
%
</body>
</html>
```

**Output:****Program 14:** Program to display the pattern.

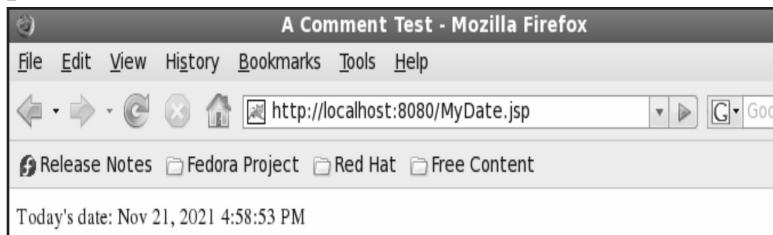
```
<html>
<body bgcolor="pink">
<%! int n,i,j;%>
<%
n = 5;
```

```
for(i=1;i<=n;i++)
{
 for(j=1;j<=i;j++)
 {
 %>
 <%= "*"%>
 <% }%>

 <%
 }
 %>
}
</body>
</html>
```

**Output:****Program 15:** Program to display date and message.

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

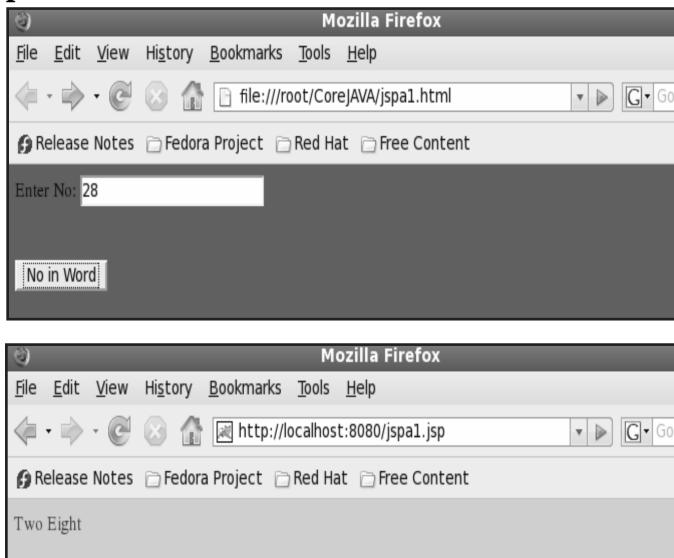
**Output:**

**Program 16:** Program to display the number into the word.

```
//jspa1.jsp
<html>
 <body bgcolor="green">
 <form action="http://localhost:8080/jspa1.jsp" method="post" >
 Enter No: <input type="text" name="txt">

 <input type="submit" value="No in Word">
 </form>
 </body>
</html>
//jspa1.jsp
<%@ page import="java.lang.*" %>
<html>
 <body bgcolor="pink" text="red">
 <%
 try
 {
 String wd=request.getParameter("txt");
 int no=Integer.parseInt(wd);
 int rem=0,rev=0,n;
 while(no!=0)
 {
 rem=no%10;
 no=no/10;
 rev=rev*10+rem;
 }
 }
```

```
while(rev!=0)
{
 n=rev%10;
 rev=rev/10;
 switch(n)
 {
 case 0:out.println("Zero");
 break;
 case 1:out.println("One");
 break;
 case 2:out.println("Two");
 break;
 case 3:out.println("Three");
 break;
 case 4:out.println("Four");
 break;
 case 5:out.println("Five");
 break;
 case 6:out.println("Six");
 break;
 case 7:out.println("Seven");
 break;
 case 8:out.println("Eight");
 break;
 case 9:out.println("Nine");
 break;
 }
}
catch(Exception e)
{
 out.println("Please Enter No.");
}
%>
```

**Output:**

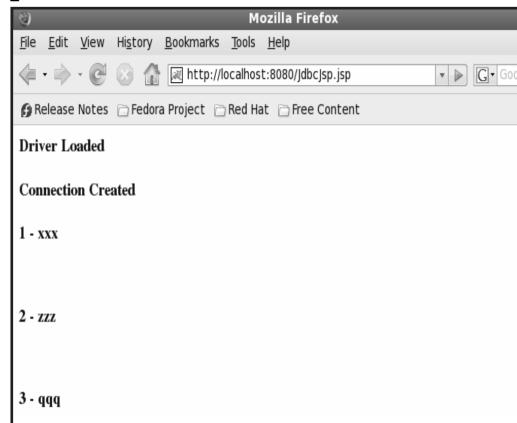
**Program 17:** Program to display the customer information like cust\_no and cust\_name with store in the table customer using JSP (JSP and Postgresql).

```
<%@page language="java" import="java.io.*" import="java.sql.*"%>
<html>
 <body>
 <%
 Connection con;
 Statement st;
 ResultSet rs;
 %>
 <%
 try
 {
 Class.forName("postgresql.Driver");
 out.println("<h3>Driver Loaded</h3>");
 con=DriverManager.getConnection
 ("jdbc:postgresql:TEST","postgres","");
 out.println("<h3>Connection Created</h3>");
 st=con.createStatement();
 rs=st.executeQuery("select * from customer");
 }
 %>
```

```

 while(rs.next())
 {
 out.print("<h3>" + rs.getInt(1)
 + " - " + rs.getString(2) + "</h3>");
 out.println("
");
 }
 }
 catch(Exception e)
 {
 out.println("error" + e);
 }
%>
</body>
</html>

```

**Output:****PRACTICE QUESTIONS****Q. I Multiple Choice Questions:**

1. Which is a java program that runs inside JVM on the web server and used for developing dynamic web applications?
 

|                |                           |
|----------------|---------------------------|
| (a) Servlet    | (b) Applet                |
| (c) Angular JS | (d) None of the mentioned |
2. Features of Servlet include,
 

|                            |                          |
|----------------------------|--------------------------|
| (a) Efficient and Scalable | (b) Portable and Robust  |
| (c) Powerful and flexible  | (d) All of the mentioned |
3. JSP stands for,
 

|                       |                           |
|-----------------------|---------------------------|
| (a) Java Save Pages   | (b) Java Server Pages     |
| (c) Java Single Pages | (d) None of the mentioned |

4. Which object of HttpSession can be used to view and manipulate information about a session?
  - (a) creation time
  - (b) last accessed time
  - (c) session identifier
  - (d) All of the mentioned
5. How constructor can be used for a Servlet?
  - (a) Initialization
  - (b) Constructor function
  - (c) Both (a) and (b)
  - (d) None of the mentioned
6. Which page directive should be used in JSP to generate a PDF page?
  - (a) contentType
  - (b) generatePdf
  - (c) typePDF
  - (d) contentPDF
7. In JSP Action tags which tags are used for bean development?
  - (a) jsp:useBean
  - (b) jsp:setProperty
  - (c) jsp:getProperty
  - (d) All of the mentioned
8. Which of the following is stored at client side?
  - (a) URL rewriting
  - (b) Hidden form fields
  - (c) Cookies
  - (d) Session
9. Which of the below is not a session tracking method?
  - (a) HTTP
  - (b) Cookies
  - (c) URL Rewriting
  - (d) SSL Sessions
10. Which of the following code is used to get an attribute in a HTTP Session object in Servlets?
  - (a) session.alterAttribute(String name)
  - (b) session.updateAttribute(String name)
  - (c) session.setAttribute(String name)
  - (d) session.getAttribute(String name)
11. Which technology do we mix our business logic with the presentation logic?
  - (a) Servlet
  - (b) JSP
  - (c) ASP
  - (d) Applet
12. Which tag should be used to pass information from JSP to included JSP?
  - (a) Using <%jsp:useBean> tag
  - (b) Using <%jsp:page> tag
  - (c) Using <%jsp:import> tag
  - (d) Using <%jsp:param> tag
13. Which in Java refer to a collection of individual objects that are represented as a single unit?
  - (a) Initialization, Cleanup, Compilation, Execution
  - (b) Initialization, Compilation, Cleanup, Execution
  - (c) Compilation, Initialization, Execution, Cleanup
  - (d) Cleanup, Compilation, Initialization, Execution

14. A Servlet life cycle can be defined as the entire process from its creation till the destruction. The paths followed by a Servlet include,
- The Servlet is initialized by calling the init() method.
  - The Servlet calls service() method to process a client's request.
  - The Servlet is terminated by calling the destroy() method.
  - All of the mentioned

### Answers

|         |         |         |         |        |        |        |        |        |         |
|---------|---------|---------|---------|--------|--------|--------|--------|--------|---------|
| 1. (a)  | 2. (d)  | 3. (b)  | 4. (d)  | 5. (c) | 6. (a) | 7. (d) | 8. (c) | 9. (a) | 10. (d) |
| 11. (a) | 12. (b) | 13. (c) | 14. (d) |        |        |        |        |        |         |

**Q. II Fill in the Blanks:**

- \_\_\_\_\_ HTTP Request method is non-idempotent.
- \_\_\_\_\_ method can be used to read a form parameter in JSP.
- \_\_\_\_\_ is used to pass configuration information to Servlet.
- \_\_\_\_\_ attributes are mandatory in <jsp:setProperty /> tag.
- \_\_\_\_\_ and javax.servlet.http packages provide interfaces and classes for writing our own Servlets.
- A \_\_\_\_\_ is a Java program that runs on a Web server.
- \_\_\_\_\_ method of servlet gets called.
- JSP stands on \_\_\_\_\_.
- \_\_\_\_\_ tag is used to execute java source code in JSP.
- \_\_\_\_\_ are the Java programs that run on the Java-enabled web server or application server.
- \_\_\_\_\_ is used by clients and servers to communicate on the web.
- The entire life cycle of a servlet is managed by the Servlet container which uses the \_\_\_\_\_ interface to understand the Servlet object and manage it.
- \_\_\_\_\_ simply means a particular interval of time.
- A \_\_\_\_\_ is a small piece of information that is persisted between the multiple client requests.
- A JSP \_\_\_\_\_ affects the overall structure of the Servlet class.
- A \_\_\_\_\_ page consists of HTML tags and JSP tags.
- A scriptlet tag is used to execute Java \_\_\_\_\_ code in JSP.
- The init() method is called only \_\_\_\_\_ and is called only when the servlet is created.
- The Servlet \_\_\_\_\_ (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.

20. The \_\_\_\_\_ action Includes a file at the time the page is requested.
21. Servlets are Java classes which service HTTP \_\_\_\_\_ and implement the javax.servlet.Servlet interface.
22. The GET method sends the encoded user information appended to the page request. Servlet handles this type of requests using \_\_\_\_\_ method.
23. Session \_\_\_\_\_ is a technique to maintain state (data) of a user also known as session management in Servlet.
24. A JSP \_\_\_\_\_ declares one or more variables or methods that you can use in Java code later in the JSP file.
25. The \_\_\_\_\_ method is called only once at the end of the life cycle of a Servlet.
26. JSP actions use constructs in \_\_\_\_\_ syntax to control the behavior of the Servlet engine.
27. A JSP \_\_\_\_\_ contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

### Answers

|                   |                           |                                |
|-------------------|---------------------------|--------------------------------|
| 1. POST           | 2. request.getParameter() | 3. javax.servlet.ServletConfig |
| 4. name, property | 5. javax.servlet          | 6. Servlet                     |
| 7. init()         | 8. Java Server Page       | 9. Scriptlet                   |
| 10. Servlets      | 11. HTTP                  | 12. javax.servlet.Servlet      |
| 13. Session       | 14. cookie                | 15. directive                  |
| 16. JSP           | 17. source                | 18. once                       |
| 19. container     | 20. jsp:include           | 21. requests                   |
| 22. doGet()       | 23. tracking              | 24. declaration                |
| 25. destroy()     | 26. XML                   | 27. expression                 |

### Q. III State True or False:

1. Is JSP technology extensible?
2. Default timeout value for session variable is 20 minutes.
3. Servlets execute within the address space of web server.
4. Objects are accessible only from the pages which are in same session.
5. Servlets execute within the address space of web server, platform independent and uses the functionality of java class libraries.
6. Can servlet class declare constructor with ServletConfig object as an argument?
7. SessionIDs are stored in cookies.
8. New session cannot be created for a new user.
9. Objects are accessible only from the page in which they are created

10. `_jspService()` method of `HttpJspPage` class should not be overridden.
11. Servlets are used to handle the request obtained from the Web server, process the request, produce the response, then send a response back to the Web server.
12. Cookies are text files stored on the client computer and they are kept for various information tracking purpose.
13. To read cookies, you need to create an array of `javax.servlet.http.Cookie` objects by calling the `getCookies()` method of `HttpServletRequest`.
14. A JSP directive affects the overall structure of the Servlet class.
15. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
16. URL rewriting is a better way to maintain sessions and it works even when browsers don't support cookies.
17. The Web server needs a JSP engine, i.e., a container to process JSP pages. The JSP container is responsible for intercepting responses for JSP pages.
18. Servlet provides `HttpSession` Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
19. Java Servlets are programs that run on a Web or Application server.
20. Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.
21. Servlets can be created using the `javax.servlet` and `javax.servlet.http` packages, which are a standard part of the Java's enterprise edition.
22. A JSP life cycle is defined as the process from its creation till the destruction. the paths followed by a JSP, Compilation, Initialization, Execution and Cleanup.
23. `jsp:setProperty` action gets the property of a JavaBean.
24. When a container loads a JSP it invokes the `jspInit()` method before servicing any requests.
25. JSP comment marks text or statements that the JSP container should ignore.

### Answers

|         |         |         |         |         |         |         |         |         |         |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 1. (T)  | 2. (F)  | 3. (F)  | 4. (T)  | 5. (T)  | 6. (F)  | 7. (T)  | 8. (T)  | 9. (F)  | 10. (T) |
| 11. (T) | 12. (T) | 13. (F) | 14. (T) | 15. (T) | 16. (T) | 17. (F) | 18. (T) | 19. (T) | 20. (T) |
| 21. (T) | 22. (T) | 23. (F) | 24. (T) | 25. (T) |         |         |         |         |         |

### Q. IV Answer the following Questions:

#### (A) Short Answer Questions:

1. What is Servlet?
2. Define cookie.
3. What is JSP?
4. Which method is used to send a cookie to the client from Servlet?

5. State the difference between doGet and doPost methods.
6. Which methods are used to obtain and store information from HttpSession object?
7. Which method is used to send the cookie from server to the client? Give its syntax.
8. What is session?
9. What is purpose of JSP directives?
10. Name the methods in the lifecycle of a Servlet.
11. What is scriptlet?
12. How to send cookie from server to client?
13. Name the method to get all values of a parameter passed from html to Servlet.
14. List the different ways by which session tracking can be done.
15. What are the parameters of the doGet method in Servlet?
16. State any two predefined variables provided by JSP.
17. What is JSP action?
18. Define session tracking.

**(B) Long Answer Questions:**

1. What is Servlet technology? Explain with example. What are its advantages and disadvantages?
  2. With the help of diagram describe Servlet hierarchy.
  3. Explain the life cycle of Servlet diagrammatically.
  4. How many types are there in Servlet? Explain.
  5. What is a cookie? How to create and access it? Explain with example.
  6. What is session? How to track it? Explain with example.
  7. Write a program for making a registration form which collects name and email address send this data to a Servlet that displays it.
  8. Describe the relationship between Servlet and JSP. Also compare them.
  9. Write a Servlet program which display the current date and time.
  10. Write a Servlet which accepts a font name and name of background color from user and use this information to display text in the specified font and change the background color of the HTML page to the specified color.
  11. Write a Servlet code to get information about the server such as name of server, server port number and server version.
  12. Explain architecture of JSP with the help of diagram.
  13. Write a short note on: JSP directives.
  14. What are the components of JSP? Describe two of them in detail
-

15. Explain the following terms in context of JSP:
  - (i) scriptlets
  - (ii) expression.
16. What is purpose of JSP directives? Explain with syntax and example.
17. What are JSP scripting elements? Describe them in brief.
18. What is the purpose of implicit objects? Explain any two implicit objects in JSP.
19. Differentiate between Servlet and JSP.

## UNIVERSITY QUESTIONS AND ANSWERS

**April 2016**

1. State the difference between doGet() and doPost() methods. [1 M]
- Ans.** Refer to Section 4.2, Point (2 (i) and (ii)).
2. How do you create a session using HttpSession object. [1 M]
- Ans.** Refer to Section 4.7.2.5.
3. List any two difference between JSP and Servlet. [1 M]
- Ans.** Refer to Section 4.8.
4. What is scriptlet? Write syntax of scriptlet. [1 M]
- Ans.** Refer to Section 4.12.3.
5. What is cookies? Explain how cookies are created and accessed in Servlet. [1 M]
- Ans.** Refer to Section 4.7.2.4.
6. Explain JSP life cycle. [5 M]
- Ans.** Refer to Section 4.10.
7. Write a note on JSP Expressions with example. [4 M]
- Ans.** Refer to Section 4.12.2.

**April 2017**

1. What is Servlet? [1 M]
- Ans.** Refer to Sections 4.1 and 4.1.1.
2. Write syntax of comments in JSP. [1 M]
- Ans.** Refer to Section 4.12.4.
3. What is purpose of JSP directives? [1 M]
- Ans.** Refer to Section 4.13.
4. Explain the purpose of GET and POST method is Servlet request. Differentiate between doGet() and doPost() methods. [5 M]
- Ans.** Refer to Section 4.4, Points (1) and (2) and Section 4.2, Point (2 (i) and (ii)).
5. Write a Servlet program to count the number of times a Servlet has been invoked. [5 M]
- Ans.** Refer to Additional Programs.

6. What is JSP? Explain JSP life-cycle. [5 M]
- Ans.** Refer to Sections 4.8 and 4.8.1..
7. What is cookie? Explain how cookie can be created and accessed in servlet. [4 M]
- Ans.** Refer to Section 4.7.2.4.
8. What are advantages of JSP? [4 M]
- Ans.** Refer to Section 4.8.4.
- 

**October 2017**

1. What is the use of page directives in JSP? [1 M]
- Ans.** Refer to Section 4.13.1.
2. List the parameters of the doPost method is Servlet. [1 M]
- Ans.** Refer to Section 4.2, Point (2 (ii)).
3. Write a Servlet program that accept teacher id, name and subject and also print the same. [5 M]
- Ans.** Refer to Additional Programs.
4. Explain Servlet lifecycle with the help of a diagram. [5 M]
- Ans.** Refer to Section 4.10.
5. Write a JSP program which takes Multiplicand and Multiplier from user as input and after clicking on "Multiply" button it will display Multiplicand, \*, Multiplier and Product. [4 M]
- Ans.** Refer to Additional Programs.
6. Differentiate between doGet() and doPost() methods. [2 M]
- Ans.** Refer to Section 4.2, Point (2 (i) and (ii)).
7. Explain scripting elements in JSP. [4 M]
- Ans.** Refer to Section 4.12.
8. What is cookie? How to set cookies explain with example. [4 M]
- Ans.** Refer to Section 4.7.2.4.
- 

**April 2018**

1. Which JSP tag is used to give declaration? [1 M]
- Ans.** Refer to Section 4.12.1.
2. Write a Servlet program which display the current date and time. [5 M]
- Ans.** Refer to Additional Programs.
3. Explain scripting elements in JSP. [5 M]
- Ans.** Refer to Section 4.12.
4. What is Servlet? State its advantages. [4 M]
- Ans.** Refer to Sections 4.1 and 4.1.3.
5. List implicit objects in JSP and explain any two implicit objects in JSP. [4 M]
- Ans.** Refer to Section 4.11.
-

**October 2018**

1. Write any two methods of HTTP session. [1 M]
- Ans.** Refer to Section 4.7.2.5.
2. Explain the session tracking. [1 M]
- Ans.** Refer to Section 4.7.2.
3. What is scriptlet? [1 M]
- Ans.** Refer to Section 4.12.3.
4. Write a Servlet program to get information about the server such as name, server port number and server version. [5 M]
- Ans.** Refer to Additional Programs.
5. Write JSP program to demonstrate all three scripting tags used in JSP. [5 M]
- Ans.** Refer to Additional Program.
6. Write JSP program to accept student name, address and class from HTML and display it on another page. [5 M]
- Ans.** Refer to Section Additional Program.
7. What is a Servlet? Differentiate doGet() and doPost() methods. [5 M]
- Ans.** Refer to Sections 4.1 and 4.2, Point (2 (i) and (ii)).
8. Write a Servlet program to count number of times a Servlet has been invoked. [4 M]
- Ans.** Refer to Additional Programs.
9. What is JSP? Explain JSP life cycle. [4 M]
- Ans.** Refer to Sections 4.8 and 4.10.

**April 2019**

1. Explain and write the syntax of getcookies() method in Servlet? [1 M]
- Ans.** Refer to Section 4.7.2.4.
2. State three methods which are used for session tracking? [1 M]
- Ans.** Refer to Section 4.7.2.
3. Write a general syntax of include directive in JSP? [1 M]
- Ans.** Refer to Section 4.13.2.
4. Write a Servlet to get information about the server such as name of server, port number of server. [5 M]
- Ans.** Refer to Additional Programs.
5. Explain the Servlet Life Cycle. [5 M]
- Ans.** Refer to Section 4.2.
6. Explain scripting elements in JSP? [4 M]
- Ans.** Refer to Section 4.12.
7. Explain life cycle of JSP. [4 M]
- Ans.** Refer to Section 4.10.

■ ■ ■

# Spring Framework

## Objectives...

- To understand Basic Concept of Spring
  - To learn Spring Architecture and MVC
  - To study different Applications of Spring
- 

### 5.0 INTRODUCTION

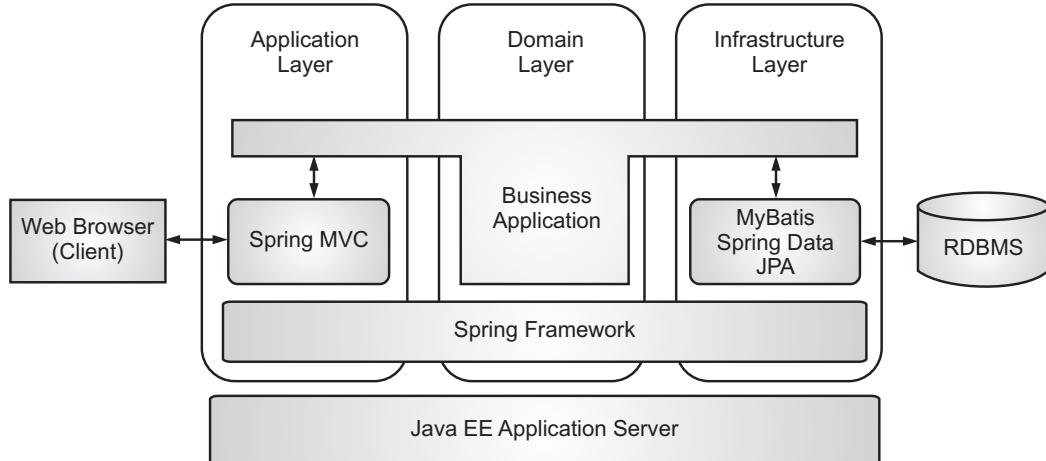
- Spring is an open source framework created to address the complexity of enterprise application development.
- Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.
- Spring was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.
- Spring Framework is an open source framework that we use to develop Java applications with very ease and with a rapid pace.
- Spring is a very lightweight framework which provides well-defined infrastructure support for developing Java application.

#### Why Spring?

- Java programs are complex and feature many heavyweight components. Heavyweight means the components are dependent on the underlying Operating System (OS) for their appearance and properties.
  - Spring is considered to be a secure, low-cost and flexible framework.
  - Spring improves coding efficiency and reduces overall application development time because it is lightweight and efficient at utilizing system resources.
  - Spring removes tedious configuration work so that developers can focus on writing program logic. Spring handles the infrastructure so developers can focus on the application.
-

## 5.1 INTRODUCTION TO SPRING FRAMEWORK

- 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.
- Fig. 5.1 shows the Spring Framework. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building Web applications on top of the Java EE platform.
- 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.
- The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications.
- One of the most popular Java Enterprise Edition (Java EE) frameworks, Spring helps developers create high performing applications using Plain Old Java Objects (POJOs).



**Fig. 5.1: Spring Framework**

### Advantages of Spring Framework:

- Advantages of Spring Framework are as follows:
  - 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 supports 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).
5. Spring Framework supports other frameworks and its integration makes Spring easier to develop.
6. Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean style POJOs, it becomes easier and simpler to use dependency injection for injecting test data.
7. Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC) into consistent, unchecked exceptions.
8. The Spring Framework provides modularity to Java developers/programmers which helps them to choose which packages or classes can be used or ignored. With tons of classes and packages, it comes as a boon to developers to identify and choose the packages or classes without any problems.
9. The Spring Framework AOP (Aspect Oriented Programming) module allows a Java developer/programmer to have different compilation unit or separate class loader. Along with that, it uses IoC (Inversion of Control ) for dependency injection which allows aspects to be configured normally.
10. Spring is lightweight, loosely coupled and open source.

#### **Disadvantages of Spring Framework:**

- Disadvantages of Spring Framework are as follow:
  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).
  5. No clear guidance in Spring Framework on cross-site scripting attacks and cross-site request attacks in Spring MVC documentation. Also, it suffers from a several security holes.

**Concept of Dependency Injection (DI) in Spring Framework:**

- DI is the fundamental aspect of spring framework through which the spring container injects the properties of one object into another object. Every object will have their dependencies or basic requirements.
- Dependency Injection (DI) also ensures loose coupling between classes. Spring Framework provides two options to inject the dependencies namely, Constructor Injection and Setter Injection.
  1. **Constructor Injection:** In this DI the container will invoke the constructor with an argument which represents dependency we want to set. We can inject multiple dependencies through the constructor. In order to inject dependencies through constructor we have to configure all the details inside the bean configuration file.
  2. **Setter Injection:** In this case dependencies will be injected by setter method. Setter injection can be used to inject partial dependencies of an object. Setter injection is more flexible than constructor injection.

## 5.2 SPRING MODULES/ARCHITECTURE

- Fig. 5.2 shows architecture of Spring Framework with its important modules. The modules of Spring Framework are organized into multiple containers like Core, Web and Data.
- The basic idea behind the development of Spring Framework was to make it a one-stop shop where we can integrate and use the modules according to the need/requirement of the application.
- Spring is modular, allowing us to pick and choose which modules are applicable to an application, without having to bring in the rest.
- The core container consists of core, beans, context and expression language modules that are fundamental to any Spring application.
- The **core module** is the core of the component model; it provides fundamental features like dependency injection.
- The Bean module provides Bean Factory, which is a sophisticated implementation of the factory pattern.
- The bean module is useful for the instantiation of the beans and is used extensively by the Spring component framework.
- The context module is used on top of the core and bean modules for initializing the context for the beans.
- The context module plays an important role in loading the components defined and configured for the application. The EL module is used for dynamic querying of components.
- The Spring 3.0 introduces a new expression language – Spring Expression Language (SpEL). It is a powerful expression language based on Java Server Pages (JSP) Expression Language (EL).
- The **Web** container is built on top of the core container. The Web and Web Servlet modules that are part of the web container are useful for building the Web components for the enterprise applications using the spring framework.

- The WebSocket module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The Web module provides the basic features of a Web application and the Web Servlet module provides an implementation of the spring MVC architecture for Web applications.
- The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.
- Modules in the data layer help to provide data access from the Web and business components. The **Data Layer or Data Access/Integration layer** consists of the following modules:
  - **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.

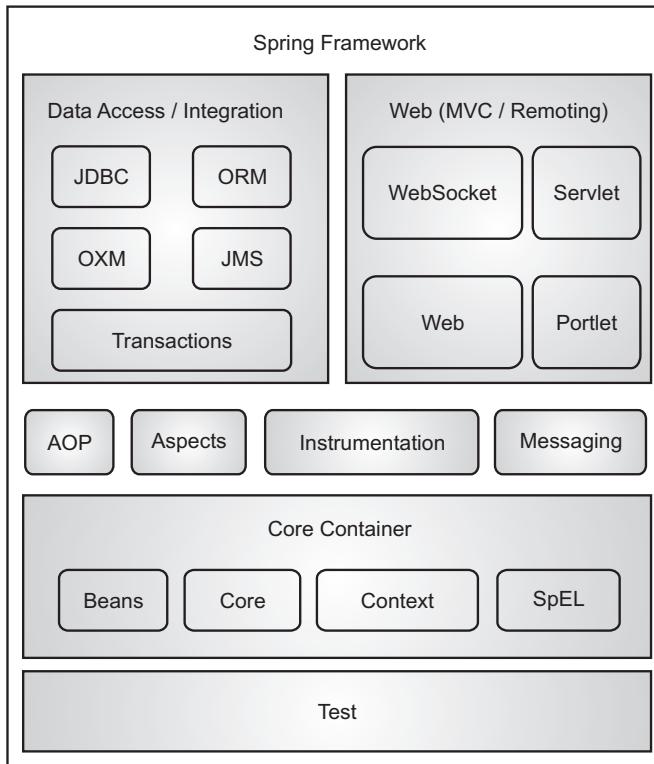


Fig. 5.2: Architecture of Spring Framework with Modules

- There are few **other important modules** which plays vital role in the Spring Framework namely, AOP, Aspects, Instrumentation, Web and Test modules. These modules are explained as follows:
  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 AspectJ, 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.
- Let us start actual programming with Spring Framework. First we create simple example in Spring with following steps. We first download the Spring Tool Suits 4 for that follow the link <https://spring.io/tools>.

**Step 1: Create Java Project:** The first step is to lunch the workspace and then create a simple spring starter project. Follow the option File → New → Spring Starter Project and finally select Java Project wizard from the wizard list. The Lunch workspace Window is shown in Fig. 5.3.

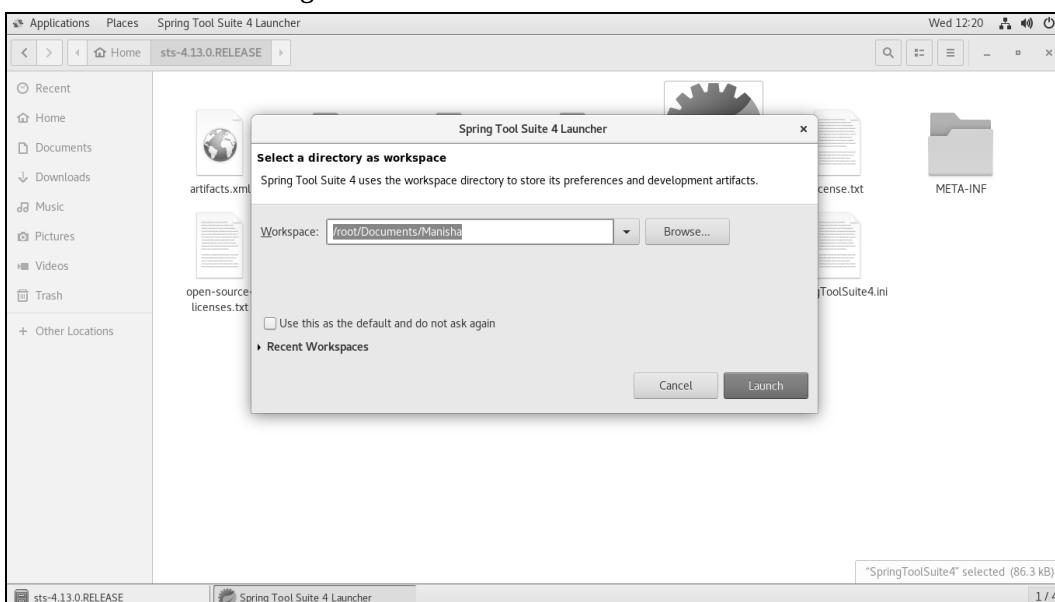


Fig. 5.3

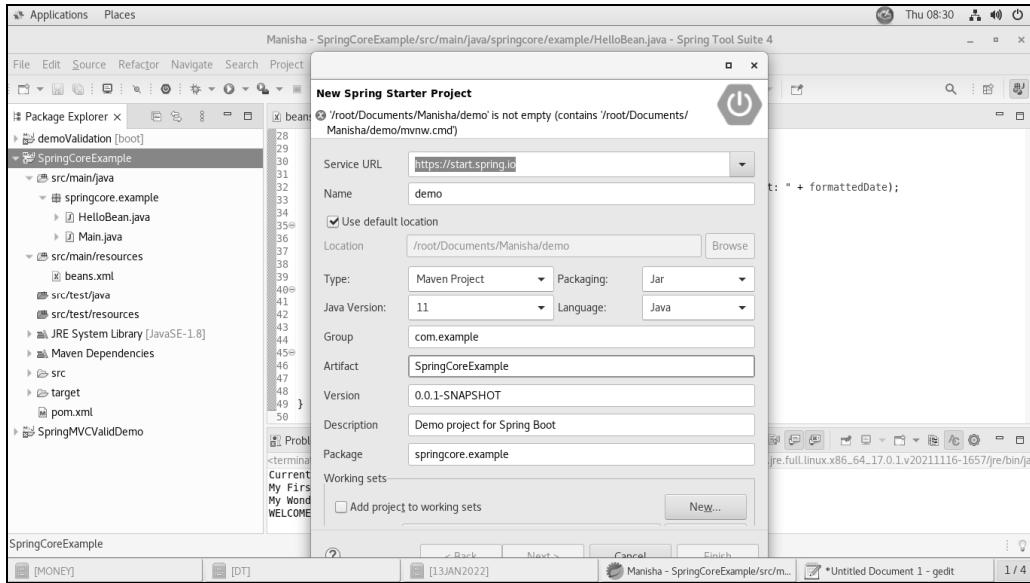


Fig. 5.4

Now name the project as SpringCore using the wizard. Once, the project is created successfully.

**Step 2: Add Required Libraries:** As a second step let us add Spring Framework and common logging API libraries in our project. To do this, right-click on the project name spring core and then follow the following option available in the context menu – Build Path → Configure Build Path to display the Java Build Path window.

Now use Add External JARs button available under the Libraries tab to add the following core JARs from Spring Framework and Common Logging installation directories:

- commons-logging-1.1.1
- spring-aop-4.1.6.RELEASE
- spring-aspects-4.1.6.RELEASE
- spring-beans-4.1.6.RELEASE
- spring-context-4.1.6.RELEASE
- spring-context-support-4.1.6.RELEASE
- spring-core-4.1.6.RELEASE
- spring-expression-4.1.6.RELEASE
- spring-instrument-4.1.6.RELEASE
- spring-instrument-tomcat-4.1.6.RELEASE
- spring-jdbc-4.1.6.RELEASE
- spring-jms-4.1.6.RELEASE
- spring-messaging-4.1.6.RELEASE
- spring-orm-4.1.6.RELEASE
- spring-oxm-4.1.6.RELEASE
- spring-test-4.1.6.RELEASE

- spring-tx-4.1.6.RELEASE
- spring-web-4.1.6.RELEASE
- spring-webmvc-4.1.6.RELEASE
- spring-webmvc-portlet-4.1.6.RELEASE
- spring-websocket-4.1.6.RELEASE

**Step 3: Create Source Files:** Now let us create actual source files under the HelloSpring project. First we need to create a package called com.niralibooks. To do this, right click on src in package explorer section and follow the options: New → Package. Next we will create HelloWorld.java and MainApp.java files under the com.

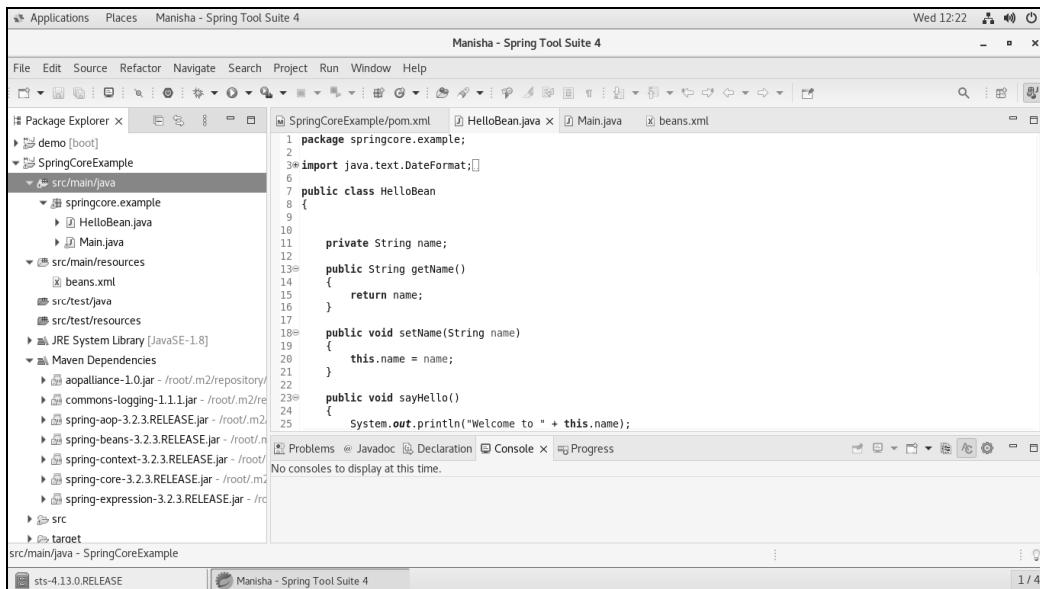


Fig. 5.5

Here, is the content of HelloBean.java file:

```

 package springcore.example;
 public class HelloBean
 {
 private String name;
 public String getName()
 {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 public void sayHello()
 {
 System.out.println("Hello" + this.name);
 }
 }

```

- Following is the content of the second file MainApp.java:

```
//Main.java
package springcore.example;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Main
{
 private static ApplicationContext context;
 public static void main(String[] args) {
 context = new ClassPathXmlApplicationContext("beans.xml");
 HelloBeanhelloBean = (HelloBean) context.getBean("HelloBean");
 helloBean.sayHello();
 }
}
```

**Step 4: Create Bean Configuration File:** We need to create a Bean Configuration file which is an XML file and acts as cement that glues the beans, i.e. the classes together. This file needs to be created under the src directory (Src/main/resources)beans.xml. Usually developers name this file as Beans.xml, but we are independent to choose any name we like. The Beans.xml is used to assign unique IDs to different beans and to control the creation of objects with different values without impacting any of the Spring source files.

For example, using the following file we can pass any value for "message" variable and we can print different values of message without impacting HelloBean.java and Main.java files. Let us see how it works.

```
//beans.xml
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p=
 "http://www.springframework.org/schema/p"
 xmlns:aop="http://www.springframework.org/schema/aop" xmlns:context=
 "http://www.springframework.org/schema/context"
 xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx=
 "http://www.springframework.org/schema/tx"
 xmlns:task="http://www.springframework.org/schema/task"
 xsi:schemaLocation="http://www.springframework.org/schema/aop
 http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
 http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
 http://www.springframework.org/schema/context"
```

```
http://www.springframework.org/schema/context/spring-context-3.2.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task-3.2.xsd">
<context:component-scan base-package="springcore.examples"/>
<bean id="HelloBean" class="springcore.example.HelloBean">
 <property name="name" value="Spring Programme"/>
</bean>
</beans>
<?xml version = "1.0" encoding = "UTF-8"?>
<beans xmlns = "http://www.springframework.org/schema/beans"
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation = "http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
 <bean id = "helloBean" class = "springcore.example.HelloBean">
 <property name = "message" value = "Hello World!"/>
 </bean>
</beans>
```

When Spring application gets loaded into the memory, Framework makes use of the above configuration file to create all the beans defined and assigns them a unique ID as defined in `<bean>` tag. We can use `<property>` tag to pass the values of different variables used at the time of object creation.

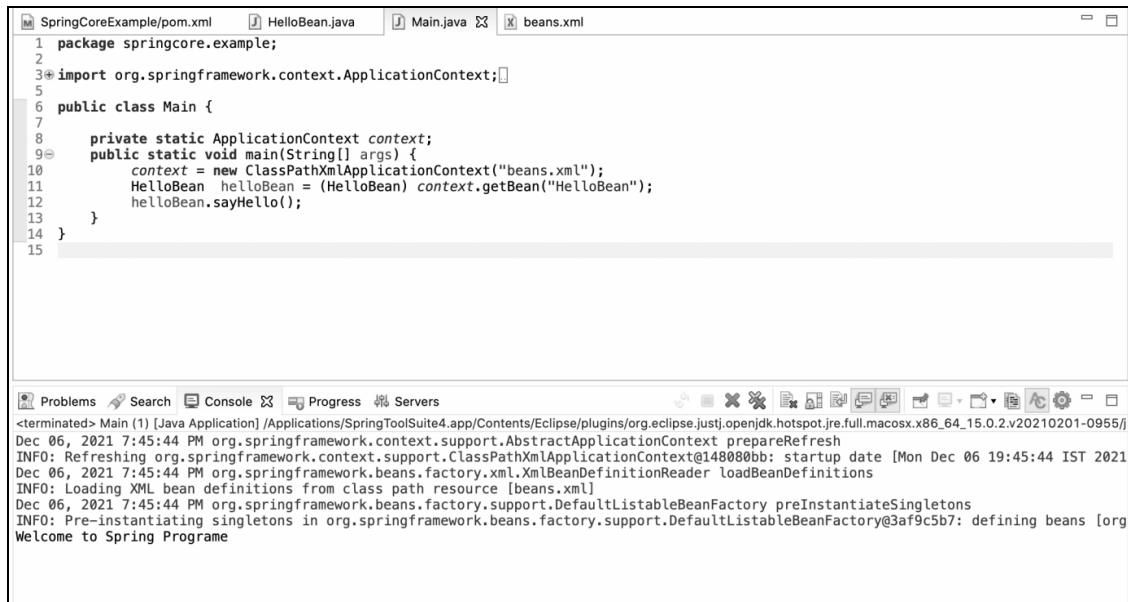
```
//pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>springcore_example</groupId>
 <artifactId>SpringCoreExample</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <!-- JDK 8 configuration below -->
 <properties>
 <spring.version>3.2.3.RELEASE</spring.version>
```

```

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
<!-- completed -->
<dependencies>
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-core</artifactId>
 <version>${spring.version}</version>
 </dependency>
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-context</artifactId>
 <version>${spring.version}</version>
 </dependency>
</dependencies>
</project>

```

**Step 5: Running the Program:** Once we are done with creating the source and beans configuration files, we are ready for this step, which is compiling and running the program. To do this, keep Main.java file tab active and use either Run option. If everything is fine with the application, this will print the following message in console.



```

1 package springcore.example;
2
3 import org.springframework.context.ApplicationContext;
4
5 public class Main {
6
7 private static ApplicationContext context;
8
9 public static void main(String[] args) {
10 context = new ClassPathXmlApplicationContext("beans.xml");
11 HelloBean helloBean = (HelloBean) context.getBean("HelloBean");
12 helloBean.sayHello();
13 }
14 }
15

```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows files: SpringCoreExample/pom.xml, HelloBean.java, Main.java, and beans.xml.
- Code Editor:** Displays the Main.java code.
- Console:** Shows the output of the application's execution.
- Output:**

```

<terminated> Main (1) [Java Application] /Applications/SpringToolSuite4.app/Contents/Eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_15.0.2.v20210201-0955/
Dec 06, 2021 7:45:44 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@14800bb: startup date [Mon Dec 06 19:45:44 IST 2021]
Dec 06, 2021 7:45:44 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [beans.xml]
Dec 06, 2021 7:45:44 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@3af9c5b7: defining beans [org
Welcome to Spring Program

```

### 5.3 SPRING APPLICATIONS

- Following is the list of few of the great benefits of using Spring Framework:
  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 JavaBeanstyle POJOs, it becomes 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.
  6. **Central Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
  7. **Lightweight:** Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
  8. **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

### 5.4 SPRING MVC

- 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.
- The Spring Web Model-View-Controller (MVC) framework is designed around a DispatcherServlet that dispatches requests to handlers, with configurable handler mappings, view resolution, locale and theme resolution as well as support for uploading files.
- The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.
- The request processing workflow of the Spring Web MVC DispatcherServlet is shown in the Fig. 5.6.

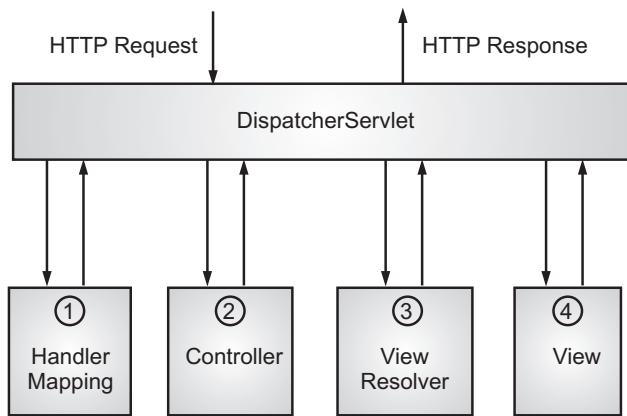
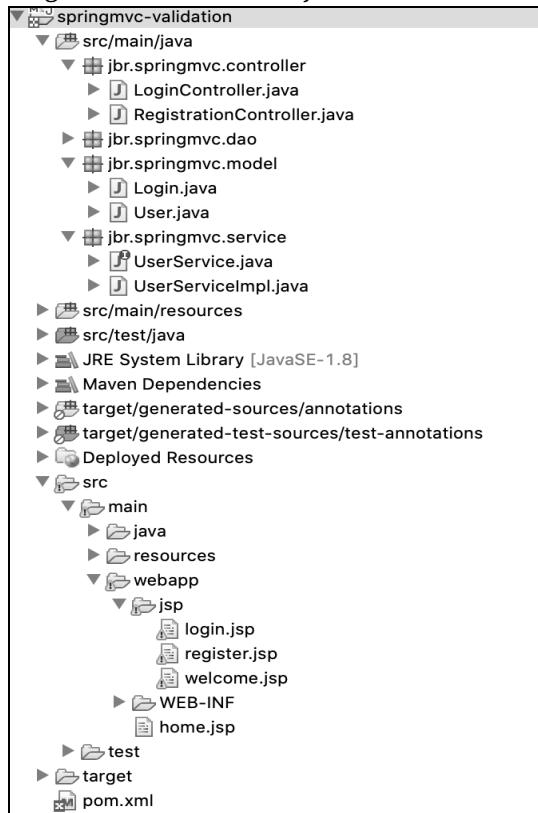


Fig. 5.6

- Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet:
  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.

- All the above-mentioned components, i.e. HandlerMapping, Controller and ViewResolver are parts of WebApplicationContext which is an extension of the plainApplicationContext with some extra features necessary for web applications.
- For example, we create a simple web-based Hello World application using Spring MVC framework.
- To start with it, let us have a working Eclipse IDE in place and take the following steps to develop a Dynamic Web Application using Spring Web Framework:
  - Step 1 :** Create a Dynamic Web Project with a name HelloWeb and create a package jbr.springmvc.controller under the src folder in the created project.
  - Step 2 :** Drag and drop below mentioned Spring and other libraries into the folder WebContent/WEB-INF/lib.
  - Step 3 :** Create a Java class HelloController under the jbr.springmvc.controller package.
  - Step 4 :** Create Spring configuration files web.xml and HelloWeb-servlet.xml under the WebContent/WEB-INF folder.
  - Step 5 :** Create a sub-folder with a name jsp under the WebContent/WEB-INF folder. Create a view file hello.jsp under this sub-folder.
  - Step 6 :** The final step is to create the content of all the source and configuration files and export the application as explained below.
- Here, is the content of RegistrationController.java file:



- Create file inside src/main/java.

**Package Name:** Jbr/springmvc/controller.

**Class Name:** RegistrationController.java.

```
package jbr.springmvc.controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import jbr.springmvc.model.User;
import jbr.springmvc.service.UserService;
@Controller
public class RegistrationController
{
 @Autowired
 public UserService userService;
 @RequestMapping(value = "/register", method = RequestMethod.GET)
 public ModelAndView showRegister(HttpServletRequest
 request, HttpServletResponse response)
 {
 ModelAndView mav = new ModelAndView("register");
 mav.addObject("user", new User());
 return mav;
 }
 @RequestMapping(value = "/registerProcess", method = RequestMethod.POST)
 public ModelAndView addUser(HttpServletRequest
 request, HttpServletResponse response,
 @ModelAttribute("user") User user)
 {
 userService.register(user);
 return new ModelAndView("welcome", "name", user);
 }
}
package jbr.springmvc.controller;
import javax.servlet.http.HttpServletRequest;
```

---

```
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import jbr.springmvc.model.Login;
import jbr.springmvc.model.User;
import jbr.springmvc.service.UserService;
@Controller
public class LoginController {
 @Autowired
 UserService userService;
 @RequestMapping(value = "/login", method = RequestMethod.GET)
 public ModelAndView showLogin(HttpServletRequest request,
 HttpServletResponse response)
 {
 ModelAndView mav = new ModelAndView("login");
 mav.addObject("login", new Login());
 return mav;
 }
 @RequestMapping(value = "/loginProcess", method = RequestMethod.POST)
 public ModelAndView loginProcess(HttpServletRequest request,
 HttpServletResponse response, @ModelAttribute("login") Login login)
 {
 ModelAndView mav = null;
 User user = userService.validateUser(login);
 if (null != user) {
 mav = new ModelAndView("welcome", "firstname", login);
 //mav.addObject("firstname", user.getFirstname(), user.getPassword());
 } else {
 mav = new ModelAndView("login");
 mav.addObject("message", "Username or Password is wrong!!!");
 }
 return mav;
 }
}
```

```
package jbr.springmvc.dao
package jbr.springmvc.dao;
import jbr.springmvc.model.User;
public interface UserDao
{
 int register(User user);
}
package jbr.springmvc.dao;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import jbr.springmvc.model.User;
public class UserDaoImpl implements UserDao
{
 @Autowired
 DataSource datasource;
 @Autowired
 JdbcTemplate jdbcTemplate;
 public int register(User user)
 {
 String sql = "insert into users values(?, ?, ?)";
 return jdbcTemplate.update(sql, new Object[] { user.getName(),
 user.getId(), user.getAge() });
 }
 class UserMapper implements RowMapper<User>
 {
 public User mapRow(ResultSet rs, int arg1) throws SQLException {
 User user = new User();
 user.setName(rs.getString("name"));
 user.setId(rs.getString("id"));
 user.setAge(rs.getString("age"));
 return user;
 }
 }
}
```

```
package jbr.springmvc.model;
public class User
{
 private Stringname;
 private Stringage;
 private Stringid;
 public String getName()
 {
 return name;
 }
 public void setName(Stringname)
 {
 this.name = name;
 }
 public String getAge()
 {
 return age;
 }
 public void setAge(Stringage)
 {
 this.age = age;
 }
 public String getId()
 {
 return id;
 }
 public void setId(Stringid)
 {
 this.id = id;
 }
}
package jbr.springmvc.service;
package jbr.springmvc.service;
import jbr.springmvc.model.User;
public interface UserService
{
 int register(User user);
}
```

```
package jbr.springmvc.service;
import org.springframework.beans.factory.annotation.Autowired;
import jbr.springmvc.dao.UserDao;
import jbr.springmvc.model.User;
public class UserServiceImpl implements UserService
{
 @Autowired
 public UserDao userDao;
 public int register(User user)
 {
 return userDao.register(user);
 }
}
```

- Following is the content of Spring Web configuration file web.xml:

**Web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/javaee"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
 <display-name>Archetype Created Web Application</display-name>
 <welcome-file-list>
 <welcome-file>home.jsp</welcome-file>
 </welcome-file-list>
 <servlet>
 <servlet-name>spring-mvc</servlet-name>
 <servlet-class>
 org.springframework.web.servlet.DispatcherServlet
 </servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>spring-mvc</servlet-name>
 <url-pattern>/</url-pattern>
 </servlet-mapping>
 <!-- <context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>/WEB-INF/spring-mvc-servlet.xml</param-value>
```

---

```
</context-param>
<listener>
 <listener-class>
 org.springframework.web.context.ContextLoaderListener
 </listener-class>
</listener> -->
</web-app>
```

**POM.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/maven-v4_0_0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>jbr</groupId>
 <artifactId>springmvc-student-reg-login</artifactId>
 <packaging>war</packaging>
 <version>0.1</version>
 <name>springmvc-student-reg-login</name>
 <url>http://maven.apache.org</url>
 <properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <spring.version>5.2.4.RELEASE</spring.version>
 <junit.version>4.12</junit.version>
 <servlet.version>3.1.0</servlet.version>
 <java.version>1.8</java.version>
 <postgresql.connector.version>8.0.17</postgresql.connector.version>
 </properties>
 <dependencies>
 <!-- Junit -->
 <dependency>
 <groupId>junit</groupId>
 <artifactId>junit</artifactId>
 <version>${junit.version}</version>
 <scope>test</scope>
 </dependency>
 <!-- Spring Framework -->
 <dependency>
 <groupId>org.springframework</groupId>
```

---

```
 <artifactId>spring-core</artifactId>
 <version>${spring.version}</version>
 </dependency>
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-beans</artifactId>
 <version>${spring.version}</version>
 </dependency>
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-context</artifactId>
 <version>${spring.version}</version>
 </dependency>
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-jdbc</artifactId>
 <version>${spring.version}</version>
 </dependency>
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-test</artifactId>
 <version>${spring.version}</version>
 <scope>test</scope>
 </dependency>
 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-webmvc</artifactId>
 <version>${spring.version}</version>
 </dependency>
 <!-- PostgreSQL database driver -->
 <dependency>
 <groupId>postgresql</groupId>
 <artifactId>postgresql-connector-java</artifactId>
 <version>${postgresql.connector.version}</version>
 </dependency>
 <!-- Servlet API -->
 <dependency>
 <groupId>javax.servlet</groupId>
```

- ```

<artifactId>javax.servlet-api</artifactId>
<version>${servlet.version}</version>
</dependency>
</dependencies>
<build>
<finalName>springmvc-user-reg-login</finalName>
<sourceDirectory>src/main/java</sourceDirectory>
<plugins>
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.5.1</version>
<configuration>
<source>${java.version}</source>
<target>${java.version}</target>
</configuration>
</plugin>
</plugins>
</build>
</project>

```
- Following is the content of another Spring Web configuration file HelloWeb-servlet.xml:

Spring-mvc-servlet.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation=" http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<import resource="classpath:jbr/config/user-beans.xml"/>
<context:component-scan base-package="jbr.springmvc"/>
<context:annotation-config/>
<bean
      class="org.springframework.web.servlet.
                                         view.InternalResourceViewResolver">
<property name="prefix" value="/jsp/" />

```

```
    <property name="suffix" value=".jsp" />
</bean>
</beans>
```

- Following is the content of Spring view file hello.jsp:

register.jsp

```
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
                                         charset=ISO-8859-1">
        <title>Student Registration</title>
    </head>
    <body>
        <form:form id="regForm" modelAttribute="user" action="registerProcess"
                    method="post">
            <table align="center">
                <tr>
                    <td><form:label path="name">name</form:label></td>
                    <td><form:input path="name" name="name" id="name"/></td>
                </tr>
                <tr>
                    <td><form:label path="id">id</form:label></td>
                    <td><form:input path="id" name="id" id="id"/></td>
                </tr>
                <tr>
                    <td><form:label path="age">age</form:label></td>
                    <td><form:input path="age" name="age" id="age"/></td>
                </tr>
                <tr>
                    <td></td>
                    <td>
                        <form:button id="register" name="register">Submit</form:button>
                    </td>
                </tr>
            </table>
        </form:form>
    </body>
</html>
```

```
<tr></tr>
<tr>
    <td></td>
    <td><a href="home.jsp">Home</a></td>
</tr>
</table>
</form:>form>
</body>
</html>
```

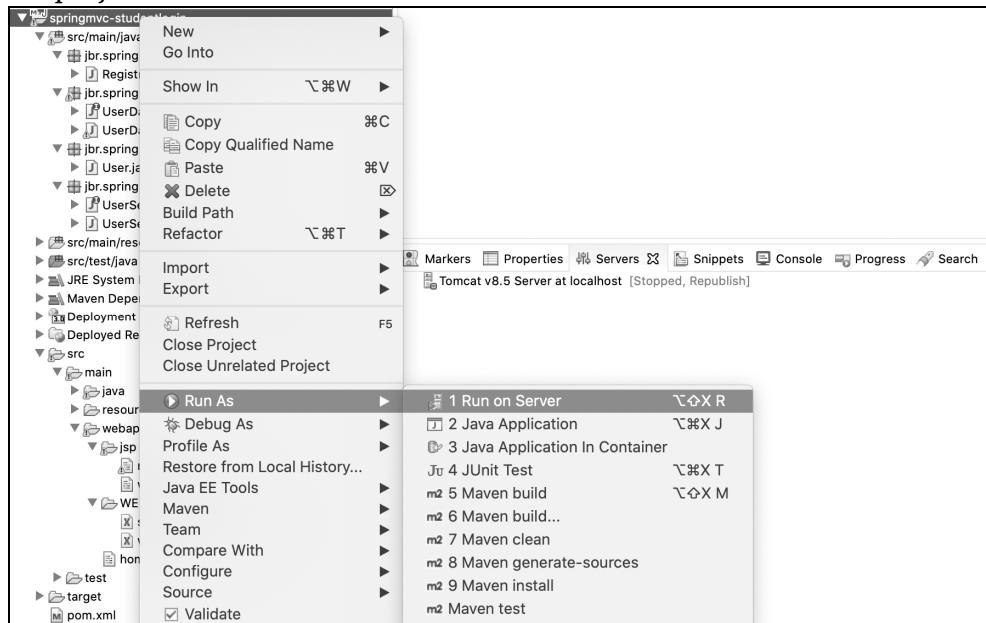
Welcome.jsp

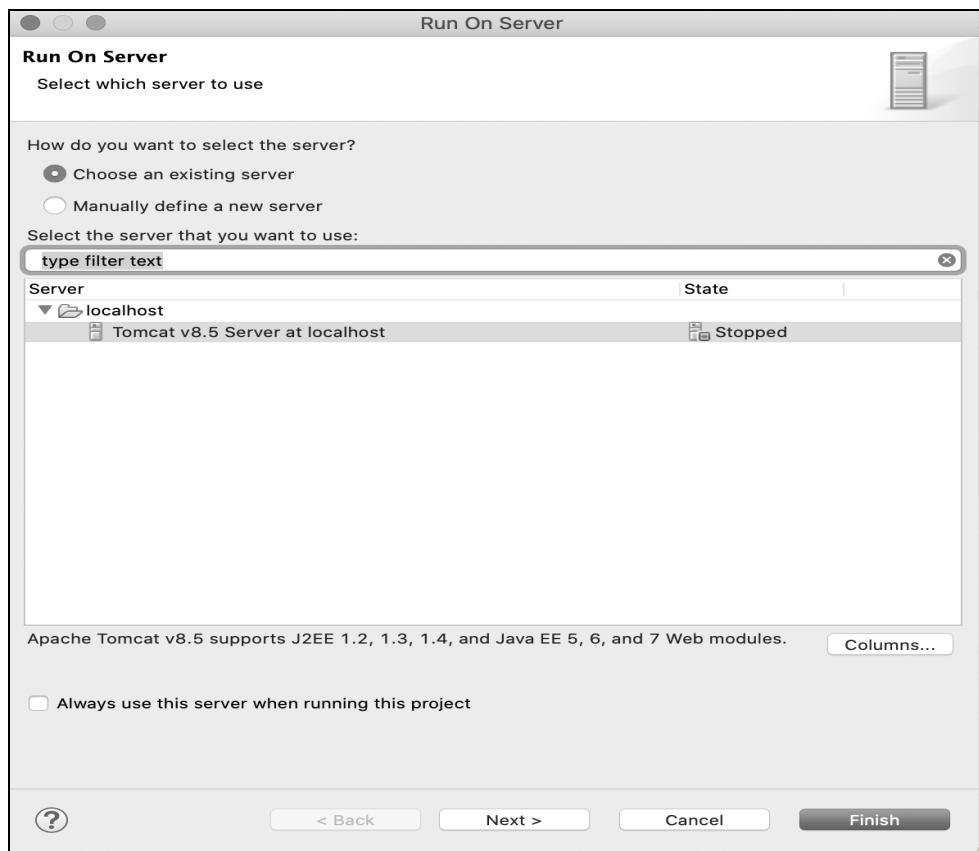
```
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
           pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
           Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
                           charset=ISO-8859-1">
        <title>Welcome</title>
    </head>
    <body>
        <table>
            <tr>
                <td>Name : ${user.name}</td>
            </tr>
            <tr>
                <td>Age : ${user.age}</td>
            </tr>
            <tr>
                <td>ID : ${user.id}</td>
            </tr>
            <tr>
                <td><a href="home.jsp">Home</a></td>
            </tr>
        </table>
    </body>
</html>
```

home.jsp

```
<%@page language="java"contentType="text/html; charset=ISO-8859-1"
          page Encoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
          Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"content="text/html;
          charset=ISO-8859-1">
<title>Welcome</title>
</head>
<body>
<table align="center">
<tr>
<td><a href="register">Register</a></td>
</tr>
</table>
</body>
</html>
```

- Now start the Tomcat server and make sure you are able to access other web pages from webapps folder using a standard browser.
- Run the project File-Run As - Run on Server.





- Try to access the URL <http://localhost:8080/springmvc-user-reg-login/login> and if everything is fine with your Spring Web Application, you should see the following output:

The screenshot shows a web browser window with a login form. The address bar shows the URL <http://localhost:8080/springmvc-user-reg-login/login>. The form has two text input fields: 'Username: satish' and 'Password:'. Below the password field is a 'Login' button. There is also a 'Home' link at the bottom of the form.

The screenshot shows a web browser window with a welcome message. The address bar shows the URL <http://localhost:8080/springmvc-user-reg-login/loginProcess>. The page content includes the heading 'Login Information' and the text 'UserName : satish'. There is also a 'Home' link at the bottom.

5.5 SPRING MVC FORMS AND VALIDATION

- 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.
- Spring form tags provide data binding for HTML forms. They have tight integration with request handlers in controllers. Generally, they represent similarly named HTML form elements and share common attributes:

| Type of Form Tag | Example |
|--|--|
| <form:input/> | <form:input path="name" placeholder="Task Name"/> |
| <form:textarea/> | <form:textarea path="comments" id="txtComments" rows="5" cols="30"/> |
| <form:select/> <form:option/> and <form:options /> | <form:select path="createdBy" id="selectCreatedBy"/> <form:option value="-1" label="Select"/> <form:options items="\${users}" itemValue="id" itemLabel="name"/> <form:select> |
| <form:label/> | <form:label for="txtTaskName" path="name">Task-names</form:label> |
| <form:hidden/> | <form:hidden path="tasked" id="hdnTaskId"/> |
| <form:password/> | <form:password path="userPassword"/> |
| <form:radiobutton/> | <form:radiobutton path="sex" value="Male"/> |
| <form:radiobuttons/> | <form:radiobuttons path="sex" items="\${sexOptions}"/> |
| <form:checkbox/> | <form:checkbox path="task.priority" value="1"/> |
| <form:checkboxes/> | <form:checkboxes path="task.priority" value="\${taskPriorities}"/> |
| <form:password/> <form:errors/> | <form:password path="password"/> <form:errors path="createdBy.id"/> |

- The following example shows how to write a simple Web application, which makes use of HTML forms using Spring Web MVC framework.
- To start with, let us have a working Eclipse IDE in place and take the following steps to develop a Dynamic Form-based Web Application using Spring Web Framework:
 - Step 1 :** Create a Dynamic Web Project with a name HelloWeb and create a package niralibooks.example under the src folder in the created project.
 - Step 2 :** Drag and drop below mentioned Spring and other libraries into the folder WebContent/WEB-INF/lib.
 - Step 3 :** Create a Java classes Student and StudentController under the niralibooks.example package.
 - Step 4 :** Create Spring configuration files Web.xml and HelloWeb-servlet.xml under the WebContent/WEB-INF folder.
 - Step 5 :** Create a sub-folder with a name jsp under the WebContent/WEB-INF folder. Create a view files student.jsp and result.jsp under this sub-folder.
 - Step 6 :** The final step is to create the content of all the source and configuration files and export the application as explained below.

- Here is the content of Student.java file:

```
package niralibooks.example;
public class Student
{
    private Integer age;
    private String name;
    private Integer id;
    public void setAge(Integer age)
    {
        this.age = age;
    }
    public Integer getAge()
    {
        return age;
    }
    public void setName(String name)
    {
        this.name = name;
    }
```

```
public String getName()
{
    return name;
}
public void setId(Integer id)
{
    this.id = id;
}
public Integer getId()
{
    return id;
}
```

- Following is the content of StudentController.java file:

```
package niralibooks.example;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.ui.ModelMap;
@Controller
public class StudentController
{
    @RequestMapping(value = "/student", method = RequestMethod.GET)
    public ModelAndView student()
    {
        return new ModelAndView("student", "command", new Student());
    }
    @RequestMapping(value = "/addStudent", method = RequestMethod.POST)
    public String addStudent(@ModelAttribute("SpringWeb")Student student,
                           ModelMap model)
    {
        model.addAttribute("name", student.getName());
        model.addAttribute("age", student.getAge());
        model.addAttribute("id", student.getId());
        return "result";
    }
}
```

- Here the first service method student(), we have passed a blank Student object in the ModelAndView object with the name "command" because the Spring Framework expects an object with the name "command" if we are using <form:form> tags in the JSP file. So, when student() method is called , it returns student.jsp view.
- The second service method addStudent() will be called against a POST method on the HelloWeb/addStudent URL.
- We will prepare the model object based on the submitted information.
- Finally a "result" view will be returned from the service method, which will result in rendering result.jsp.
- Following is the content of Spring Web configuration file web.xml:

```
<web-app id = "WebApp_ID" version = "2.4"
    xmlns = "http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>Spring MVC Form Handling</display-name>
    <servlet>
        <servlet-name>HelloWeb</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWeb</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

- Following is the content of another Spring Web configuration file HelloWeb-servlet.xml:

```
<beans xmlns = "http://www.springframework.org/schema/beans"
    xmlns:context = "http://www.springframework.org/schema/context"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation = "http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <context:component-scan base-package = "niralibooks.example" />
```

```
<bean class =
    "org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name = "prefix" value = "/WEB-INF/jsp/" />
    <property name = "suffix" value = ".jsp" />
</bean>
</beans>
• Following is the content of Spring view file student.jsp:
<%@taglib uri = "http://www.springframework.org/tags/form"
    prefix = "form"%>
<html>
    <head>
        <title>Spring MVC Form Handling</title>
    </head>
    <body>
        <h2>Student Information</h2>
        <form:form method = "POST" action = "/HelloWeb/addStudent">
            <table>
                <tr>
                    <td><form:label path = "name">Name</form:label></td>
                    <td><form:input path = "name" /></td>
                </tr>
                <tr>
                    <td><form:label path = "age">Age</form:label></td>
                    <td><form:input path = "age" /></td>
                </tr>
                <tr>
                    <td><form:label path = "id">id</form:label></td>
                    <td><form:input path = "id" /></td>
                </tr>
                <tr>
                    <td colspan = "2">
                        <input type = "submit" value = "Submit"/>
                    </td>
                </tr>
            </table>
        </form:form>
    </body>
</html>
```

- Following is the content of Spring view file result.jsp:

```
<%@page contentType = "text/html; charset = UTF-8" language = "java" %>
<%@page isELIgnored = "false" %>
<%@taglib uri = "http://www.springframework.org/tags/form"
           prefix = "form"%>

<html>
    <head>
        <title>Spring MVC Form Handling</title>
    </head>
    <body>
        <h2>Submitted Student Information</h2>
        <table>
            <tr>
                <td>Name</td>
                <td>${name}</td>
            </tr>
            <tr>
                <td>Age</td>
                <td>${age}</td>
            </tr>
            <tr>
                <td>ID</td>
                <td>${id}</td>
            </tr>
        </table>
    </body>
</html>
```

- Finally, following is the list of Spring and other libraries to be included in the web application. We simply drag these files and drop them in WebContent/WEB-INF/lib folder.
 - commons-logging-x.y.z.jar
 - org.springframework.asm-x.y.z.jar
 - org.springframework.beans-x.y.z.jar
 - org.springframework.context-x.y.z.jar
 - org.springframework.core-x.y.z.jar
 - org.springframework.expression-x.y.z.jar
 - org.springframework.web.servlet-x.y.z.jar
 - org.springframework.web-x.y.z.jar
 - spring-web.jar

- Once we are done with creating source and configuration files, export the application. Right click on the application and use the Export > WAR File option and save your SpringWeb.war file in Tomcat's webapps folder.
- Now start the Tomcat server and make sure we are able to access other web pages from webapps folder using a standard browser.
- Now try a URL <http://localhost:8080/SpringWeb/student> and we should see the following result if everything is fine with the Spring Web Application.

Student Information

| | |
|---------------|-------|
| Name | AMAR |
| Age | 10 |
| id | 12234 |
| Submit | |

- After submitting the required information, click the Submit button to submit the form. We should see the following result if everything is fine with the Spring Web Application.

Submitted Student Information

| | |
|------|-------|
| Name | AMAR |
| Age | 10 |
| ID | 12234 |

Spring Form MVC Validation:

- The Spring MVC Validation is used to restrict the input provided by the user. DispatcherServlet acts as the front controller in the Spring's MVC module. All the user requests are handled by this Servlet.
- In this example, we create a simple form that contains the input fields. Here, (*) means it is mandatory to enter the corresponding field. Otherwise, the form generates an error.

XML File: pom.xml

```

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>

```

```
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

Bean Class File: Employee.java

```
package niralibooks.example;
import javax.validation.constraints.Size;
public class Employee
{
    private String name;
    @Size(min=1,message="required")
    private String pass;
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    public String getPass()
    {
        return pass;
    }
}
```

```
public void setPass(String pass)
{
    this.pass = pass;
}
```

Create the Controller Class: In controller class:

- o The @Valid annotation applies validation rules on the provided object.
- o The BindingResult interface contains the result of validation.

```
package niralibooks.example;
import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class EmployeeController
{
    @RequestMapping("/hello")
    public String display(Model m)
    {
        m.addAttribute("emp", new Employee());
        return "viewpage";
    }
    @RequestMapping("/helloagain")
    public String submitForm(@Valid @ModelAttribute("emp")
                           Employee e, BindingResult br)
    {
        if(br.hasErrors())
        {
            return "viewpage";
        }
        else
        {
            return "final";
        }
    }
}
```

Provide the entry of controller in the web.xml file: web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
          xmlns=http://java.sun.com/xml/ns/javaee
          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
          http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
          id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servletclass>org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Define the bean in the xml file: spring-servlet.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- Provide support for component scanning -->
    <context:component-scan base-package="niralibooks.example;
" />
    <!--Provide support for conversion, formatting and validation -->
    <mvc:annotation-driven/>
```

```
<!-- Define Spring MVC view resolver -->
<bean id="viewResolver" class="org.springframework.web.servlet.view.
InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/jsp/"></property>
<property name="suffix" value=".jsp"></property>
</bean>
</beans>
```

Create the requested page: index.jsp.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<body>
<a href="hello">Click here...</a>
</body>
</html>
```

Create the other view components: viewpage.jsp.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<style>
.error{color:red}
</style>
</head>
<body>
<form:form action="helloagain" modelAttribute="emp">
    Username: <form:input path="name"/> <br><br>
    Password(*): <form:password path="pass"/>
    <form:errors path="pass" cssClass="error"/><br><br>
    <input type="submit" value="submit">
</form:form>
</body>
</html>
```

final.jsp

```
<html>
<body>
    Username: ${emp.name} <br><br>
    Password: ${emp.pass}
</body>
</html>
```

Output:

The screenshot shows a browser window with the URL <http://localhost:8082/SpringMVCValidation/>. Below the address bar is a single text input field containing the text "Click here...".

Let's submit the form without entering the password.

The screenshot shows a browser window with the URL <http://localhost:8082/SpringMVCValidation/helloagain>. It contains a login form with two fields: "Username" set to "AMAR" and "Password(*)" empty. A "submit" button is visible at the bottom.

Now, we entered the password and then submit the form.

The screenshot shows a browser window with the URL <http://localhost:8082/SpringMVCValidation/hello>. The login form has been submitted. The "Username" field still contains "AMAR", but the "Password(*)" field now contains three asterisks ("***"). A "submit" button is visible at the bottom.

The screenshot shows a browser window with the URL <http://localhost:8082/SpringMVCValidation/helloagain>. The page displays the submitted form data: "Username: AMAR" and "Password: jtp".

ADDITIONAL PROGRAMS

Program 1: Spring core example to print Welcome to Spring Programme.

HelloBean.java

```
package springcore.example;
public class HelloBean
{
    private String name;
    public String getName()
```

```
{  
    return name;  
}  
public void setName(String name)  
{  
    this.name = name;  
}  
  
public void sayHello()  
{  
    System.out.println("Welcome to " + this.name);  
}  
public void helloWorld() {  
    System.out.println("My First Spring Program.....");  
}  
public void getMsg() {  
    System.out.println("My WonderFul World...");  
}  
public void msgDisplay() {  
    System.out.println("WELCOME Spring PROGRAMMING 2022 ");  
}  
}  
//Main.java  
package springcore.example;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
public class Main  
{  
    private static ApplicationContext context;  
    public static void main(String[] args) {  
        context = new ClassPathXmlApplicationContext("beans.xml");  
        HelloBean helloBean = (HelloBean) context.getBean("HelloBean");  
        helloBean.sayHello();  
        helloBean.helloWorld();  
        helloBean.getMsg();  
        helloBean.msgDisplay();  
    }  
}
```

```
//beans.xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:task="http://www.springframework.org/schema/task"
       xsi:schemaLocation="http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
                           http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.2.xsd
                           http://www.springframework.org/schema/jee
                           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
                           http://www.springframework.org/schema/task
                           http://www.springframework.org/schema/task/spring-task-3.2.xsd">
    <context:component-scan base-package="springcore.example" />
    <bean id="HelloBean" class="springcore.example.HelloBean">
        <property name="name" value="Spring Programme" />
    </bean>
</beans>
//pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                              https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>springcore_example</groupId>
    <artifactId>SpringCoreExample</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <!-- JDK 8 configuration below -->
    <properties>
```

```

<spring.version>3.2.3.RELEASE</spring.version>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
<!-- completed -->
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
</dependencies>
</project>

```

Output:

The screenshot shows the Spring Tool Suite 4 interface. The left pane displays the Package Explorer with the project structure: src/main/java, src/main/resources, src/test/java, src/test/resources, Maven Dependencies, and pom.xml. The right pane shows the code editor for HelloBean.java and the terminal window.

```

public void sayHello()
{
    System.out.println("Welcome to " + this.name);
}

public void helloWorld() {
    System.out.println("My First Spring Program.....");
}

public void getMsg() {
    System.out.println("My WonderFull World...");
}

public void msgDisplay() {
    System.out.println("WELCOME Spring PROGRAMMING 2022 ");
}

```

The terminal window shows the following output:

```

terminated: Main [Java Application] /root/sts-4.13.0.RELEASE/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64-17.0.1.v20211116-1657/jre/bin/java
Welcome to Spring Program
My First Spring Program.....
My WonderFull World...
WELCOME Spring PROGRAMMING 2022

```

Program 2: Spring example for current date.

HelloBean.java

```
package springcore.example;
public class HelloBean
{
    private String name;
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    Public void sayHello()
    {
        System.out.println("Hello" + this.name);
    }
    public static void getCurrentTimeUsingDate()
    {
        Date date = new Date();
        String strDateFormat = "hh:mm:ss a";
        DateFormat dateFormat = new SimpleDateFormat(strDateFormat);
        String formattedDate= dateFormat.format(date);
        System.out.println("Current time of the day using Date -
                           12 hour format: " + formattedDate);
    }
}
```

Main.java

```
package springcore.example;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Main
{
    private static ApplicationContext context;
```

```
public static void main(String[] args)
{
    context = new ClassPathXmlApplicationContext("beans.xml");
    HelloBeanhelloBean = (HelloBean) context.getBean("HelloBean");
    helloBean.getCurrentTimeUsingDate();
    helloBean.sayHello();
}
```

src/main/resources**beans.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
           instance" xmlns:p="http://www.springframework.org/schema/p" xmlns:aop=
           "http://www.springframework.org/schema/aop" xmlns:context=
           "http://www.springframework.org/schema/context"
       xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx=
           "http://www.springframework.org/schema/tx"
       xmlns:task="http://www.springframework.org/schema/task"
       xsi:schemaLocation="http://www.springframework.org/schema/aop
           http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.2.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
           http://www.springframework.org/schema/task
           http://www.springframework.org/schema/task/spring-task-3.2.xsd">
<context:component-scan base-package="springcore.examples"/>
<bean id="HelloBean" class="springcore.example.HelloBean">
    <property name="name" value="Spring Programme"/>
</bean>
</beans>
```

Output:

```

11  private String name;
12
13     public String getName() {
14         return name;
15     }
16
17     public void setName(String name) {
18         this.name = name;
19     }
20
21     public void sayHello() {
22         System.out.println("Welcome to " + this.name);
23     }
24     public static void getCurrentTimeUsingDate() {
25         Date date = new Date();
26         String strDateFormat = "hh:mm:ss a";
27         DateFormat dateFormat = new SimpleDateFormat(strDateFormat);
28         String formattedDate = dateFormat.format(date);
29         System.out.println("Current time of the day using Date - 12 hour format: " + formattedDate);
30     }
31

```

Problems Search Console Progress Servers

```

<terminated Main (1) [Java Application] /Applications/SpringToolSuite4.app/Contents/Eclipse/plugins/org.eclipse.justopened.httpRuntime.full.macosx.x86_64.15.0.2+20210201-0955>
Dec 06, 2021 8:02:04 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@14800bb: startup date [Mon Dec 06 20:02:04 IST 2021]
Dec 06, 2021 8:02:04 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [beans.xml]
Dec 06, 2021 8:02:04 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@3af9c5b: defining beans [org
Welcome to Spring Program
Current time of the day using Date - 12 hour format: 08:02:04 pm

```

PRACTICE QUESTIONS**Q. I Multiple Choice Questions:**

1. What is spring?
 - (a) Proprietary framework.
 - (b) Spring is a development framework for .Net applications.
 - (c) Open source development framework for enterprise Java.
 - (d) Spring is a development framework for PHP based applications.
2. Attribute used to handle web flow requests.

| | |
|----------------------|---------------------|
| (a) servlet-requests | (b) servlet-attr |
| (c) servlet-flow | (d) servlet-mapping |
3. Component which additionally provides a pop-up date picker control for its enclosed input field.

| | |
|-------------------------|---------------------|
| (a) dateValidator | (b) clientValidator |
| (c) clientDateValidator | (d) Datevalidator |
4. Parameter used to specify a configuration file is ____.

| | |
|----------------------|---------------------------|
| (a) contextConfigure | (b) contextConfigLocation |
| (c) contextLocation | (d) contextTypeLocation |
5. What is the basic concept of Spring?

| | |
|--------------------------|------------------------------|
| (a) Inversion of control | (b) Factory pattern |
| (c) Singleton pattern | (d) Abstract factory pattern |

6. In Spring based applications, the objects lives in ____.
 - (a) Spring pool
 - (b) Spring heap
 - (c) Spring container
 - (d) Spring group
7. Spring Dependency Injection is useful because it makes the code ____.
 - (a) easier to understand
 - (b) simple
 - (c) easier to test
 - (d) All of the mentioned
8. The Spring container gets its instructions on what objects to instantiate, configure, and assemble by reading the ____.
 - (a) configuration text file
 - (b) spring integration data
 - (c) configuration metadata
 - (d) source code
9. Which of the following is the default scope in Spring?
 - (a) singleton
 - (b) session
 - (c) http
 - (d) request
10. Spring enables developers to develop enterprise-class applications using ____.
 - (a) TOJOs
 - (b) AOJOs
 - (c) POJOs
 - (d) None of the mentioned
11. Which Spring module provides a powerful expression language for querying and manipulating an object graph at runtime?
 - (a) Core module
 - (b) Web module
 - (c) Test module
 - (d) All of the mentioned
12. Which Spring module provides the testing of Spring components with JUnit or TestNG frameworks.
 - (a) Web
 - (b) Web module
 - (c) Test
 - (d) Data access
13. AOP stands for ____.
 - (a) Aspect Oriented Programming
 - (b) Aspect Originated Programming
 - (c) Aspect Oriented Paging
 - (d) None of the mentioned
14. The Data Access/Integration layer consists of ____.
 - (a) JDBC module
 - (b) ORM module
 - (c) OXM module
 - (d) All of the mentioned

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|--------|--------|--------|--------|--------|---------|
| 1. (c) | 2. (d) | 3. (c) | 4. (b) | 5. (a) | 6. (c) | 7. (d) | 8. (c) | 9. (a) | 10. (c) |
| 11. (d) | 12. (c) | 13. (a) | 14. (d) | | | | | | |

Q. II Fill in the Blanks:

1. To deal with the databases, spring framework provides JDBC _____.
2. Controller in spring is a _____.
3. In spring XML-based configuration metadata, we indicate a child bean definition by using the _____ attribute.
4. Dependency Injection is a _____.
5. Spring _____ is responsible to manage the objects life cycle.
6. The objects that form the backbone of your application and that are managed by the Spring container are called _____.
7. The Spring _____ module supports programmatic and declarative transaction management for classes that implement special interfaces and for all the POJOs.
8. The Web-MVC module contains Spring's Model-View-Controller (MVC) implementation for _____ applications.
9. Spring framework is an _____ source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.
10. _____ stands for Plain Old Java Object.
11. The Spring container is at the _____ of the Spring Framework used to create the objects, combine them together, configure them and manage their complete life cycle from creation till destruction.

Answers

| | | | |
|--------------|--------------|----------------|-------------------|
| 1. Template | 2. Interface | 3. Parent | 4. Design Pattern |
| 5. container | 6. Bean | 7. Transaction | 8. Web |
| 9. open | 10. POJO | 11. Core | |

Q. III State True or False:

1. Spring framework is an open source Java platform.
2. Spring framework itself also offers a remoting technology called HTTP Invoker.
3. With AOP objects are expected to acquire their dependencies on their own.
4. Spring is heavyweight.

5. Controller Class renders the objects passed by the controller's handler method.
6. By default, @RequestMapping gets all the POST requests.
7. Spring enables developers to develop enterprise-class applications using POJOs.
8. Testing an application written with Spring is simple because environment-dependent code is moved into this framework.
9. The Spring container uses DI (Dependency Injection) to manage the components that make up an application.
10. The AOP module of Spring Framework provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
11. The Bean module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
12. Spring supports tight coupling.
13. The Spring IoC (Inversion of Control) container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.
14. Spring BeanFactory container providing the basic support for DI and is defined by the org.springframework.beans.factory.BeanFactory interface.

Answers

| | | | | | | | | | |
|---------|---------|---------|---------|--------|--------|--------|--------|--------|---------|
| 1. (T) | 2. (T) | 3. (F) | 4. (F) | 5. (T) | 6. (T) | 7. (T) | 8. (T) | 9. (T) | 10. (T) |
| 11. (T) | 12. (F) | 13. (T) | 14. (T) | | | | | | |

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. What is Spring?
2. “Spring is lightweight, loosely coupled and open source”. Comment this statement.
3. List modules in Spring.
4. What is Spring MVC?
5. List any two applications of Spring.
6. Spring has MVC architecture. State true or false.
7. List modules for Spring.

(B) Long Answer Questions:

1. What is Spring framework? Explain in detail.
2. How to create an application in Spring? Explain with example.
3. With the help of diagram describe MVC Spring architecture.
4. What is form in Spring? How to create it? Explain with program.
5. What are the applications of Spring?
6. How to validate forms in Spring? Explain with example.
7. With the help of diagram describe module architecture of Spring.

■ ■ ■