Savitribai Phule Pune University

सावित्रीबाई फुले पुणे विद्यापीठ

T. Y. B. Sc.

(Computer Science)

CS-348 Laboratory Course II

(Java Programming - II)

(As per new syllabus w.e.f. academic year 2015-2016)

Semester II

Name ________________________________ **Roll No.** _______

College _______________________________ **Division** _______

Academic Year _____________

CO-ORDINATOR:
PROF. MS. POONAM PONDE
NOWROSJEE WADIA COLLEGE, PUNE

EDITED BY :
Ms. Poonam Ponde

INPUTS GIVEN BY :
Mr. Jeevan Tonde
Ms. Rasika Rahalkar

PREVIOUS EDITION AUTHORS:

Ms. Poonam Ponde
Mr. Jeevan Limaye
Mr. Sachin Bhoite
Ms. Kalpana Joshi

ABOUT THE WORK BOOK

- **OBJECTIVES OF THIS BOOK**

This lab-book is intended to be used by T.Y.B.Sc(Computer Science) students for Laboratory course – II (Java programming), Semester II.
The objectives of this book are

a. Covers the complete scope of the syllabus.
b. Bringing uniformity in the way course is conducted across different colleges.
c. Continuous assessment of the students.
d. Providing ready references for students while working in the lab.

- **Instructions to the students**

1. This book is mandatory for the completion of the laboratory course. Students should carry this book during practical sessions.
2. Student should read the topics mentioned in **Reading section** of this book before coming for the practical session.
3. Students should get the assignments completed and checked in time with marks assigned by the instructor in the index page.
4. Only Set A and B programs are compulsory. Programs in Set C are only for additional practice.
5. Each assignment will be assessed on a scale of 0 to 5 as indicated below.

i) Not done	0
ii) Incomplete	1
iii) Late Complete	2

iv) Needs improvement	3
v) Complete	4
vi) Well Done	5

- **Difficulty Levels**

Self Activity : Students should solve these exercises for practice only.
SET A - Easy : All exercises are compulsory.
SET B - Medium : All exercises are compulsory.
SET C - Difficult : Not Compulsory.

- **Instruction to the Instructors**

1) Make sure that students follow the instruction as given above.
2) Evaluate each assignment on a scale of 5 as specified above by ticking appropriate box.
3) The value should also be entered on assignment completion page of the respective Lab course.
4) Students can use IDE like Eclipse for performing the assignments.
5) The database used for JDBC assignment should be postgresql.
6) The assignments should be performed on Linux operating system.

Assignment Completion Sheet

Sr. No	Assignment Name	Marks
1	Collections	
2	Database programming	
3	Servlets	
4	Java Server Pages	
5	Multithreading	
6	Networking	
	Total	

Signature of Incharge

Date:

Internal Examiner **External Examiner**

Date:

Assignment 1: Collections

Objectives

- **Study the Collections framework in java**
- **Use various collections**

Reading

You should read the following topics before starting this exercise:

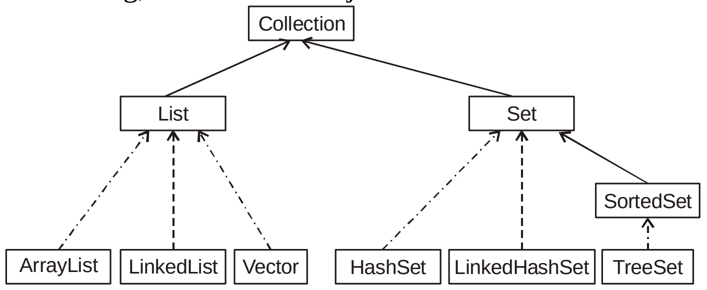
1. Concept of Collection
2. classes and interfaces in the Collections framework
3. Concept of iterator.
4. Creating and using collections objects.

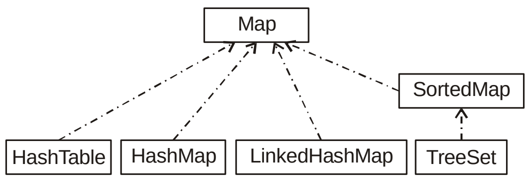
Ready Reference

A *collection* is an object that represents a group of objects. A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data.
A collections framework is a unified architecture for representing and manipulating collections, allowing them to be manipulated independently of the details of their representation.

It contains the following:

i. **Interfaces:** Interfaces allow collections to be manipulated independently of the details of their representation. Interfaces generally form a hierarchy.
ii. **Implementations:** These are the concrete implementations of the collection interfaces.
iii. **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on collection objects.





Interfaces:

Interface Name	Description
`Collection`	The most general collection interface type. A collection represents a group of objects known as its *elements*. The Java platform doesn't provide any direct implementations of this interface.
`List`	Represents an ordered collection. Lists can contain duplicate elements.

`Set`	Represents an unordered collection that does not permit duplicate elements.
`SortedSet`	Represents a set whose elements are maintained in a sorted order.
`Queue`	A collection used to hold multiple elements prior to processing. A Queue provides additional insertion, extraction and inspection operations.
`Map`	Represents key-value pairs. A Map cannot contain duplicate keys; each key can map to at most one value. Does not extend collection.
`SortedMap`	Represents a Map that maintains its mappings in ascending key order.

Classes:

Class name	Description
AbstractCollection	Implements most of the Collection interface.
AbstractList	Extends AbstractCollection and implements most of the List interface.
AbstractSequentialList	Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending AbstractSequentialList.
ArrayList	Implements a dynamic array by extending AbstractList.
AbstractSet	Extends AbstractCollection and implements most of the Set interface.
HashSet	Extends AbstractSet for use with a hash table.
TreeSet	Implements a set stored in a tree. Extends AbstractSet

The Collection interface:

This interface represents a general collection.

Method	Explanation
boolean add(Object element)	Method adds objects in the collection.
boolean remove(Object element)	Method removes objects in the collection.
The Collection interface also supports query operations:	
int size()	Returns the size of the collection.
boolean isEmpty()	Returns true if the collection is empty or false.
boolean contains(Object element)	Returns true if the collection contains the element passed in argument.
Other operations are tasks done on groups of elements or the entire collection at once:	
boolean containsAll(Collection collection)	The ***containsAll()*** method allows you to discover if the current collection contains all the elements of another collection, a *subset*.
boolean addAll(Collection collection)	ensures all elements from another collection are added to the current collection, usually a *union*.
void removeAll(Collection collection)	method is like *clear()* but only removes a subset of elements
void retainAll(Collection collection)	*This* method is similar to the *removeAll()* method but does what might be perceived as the opposite: it removes from the current

	collection those elements not in the other collection, an *intersection*.
void clear()	removes all elements from the current collection

List interface

A list stores a sequence of elements.

Method	Description
`void **add**(int index, Object element)`	Inserts the specified element at the specified position in this list (optional operation).
`Boolean **addAll**(int index, Collection c)`	Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
`Object **get**(int index)`	Returns the element at the specified position in this list.
`int **indexOf**(Object o)`	Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
`int **lastIndexOf**(Object o)`	Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.
`ListIterator **listIterator**()`	Returns a list iterator of the elements in this list (in proper sequence).
`ListIterator **listIterator**(int index)`	Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list.
`Object **remove**(int index)`	Removes the element at the specified position in this list (optional operation).
`Object **set**(int index, Object element)`	Replaces the element at the specified position in this list with the specified element (optional operation).
`int **size**()`	Returns the number of elements in this list.
`List **subList**(int fromIndex, int toIndex)`	Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.

Set Interface and SortedSet Interface

A set is a collection that contains no duplicate elements. It contains the methods of the Collection interface. A SortedSet is a `Set` that maintains its elements in ascending order. It adds the following methods:

Method	Description
`Comparator **comparator**()`	Returns the comparator associated with this sorted set, or null if it uses its elements' natural ordering.
`Object first()`	Returns the first (lowest) element currently in this sorted set.
`SortedSet headset(Object toElement)`	Returns a view of the portion of this sorted set whose elements are strictly less than toElement.
`Object last()`	Returns the last (highest) element currently in this sorted set.
`SortedSet subset(Object fromElement, Object toElement)`	Returns a view of the portion of this sorted set whose elements range from fromElement, inclusive, to toElement, exclusive.

`SortedSet tailSet(Object fromElement)`	Returns a view of the portion of this sorted set whose elements are greater than or equal to fromElement.

Map Interface

A Map represents an object that maps keys to values. Keys have to be unique.

Method	Description
`void clear()`	Removes all mappings from this map
`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.
`boolean equals(Object o)`	Compares the specified object with this map for equality.
`Object get(Object key)`	Returns the value to which this map maps the specified key.
`int hashCode()`	Returns the hash code value for this map.
`boolean isEmpty()`	Returns true if this map contains no key-value mappings
`Set keySet()`	Returns a set view of the keys contained in this map.
`Object put(Object key, Object value)`	Associates the specified value with the specified key in this map
`void putAll(Map t)`	Copies all of the mappings from the specified map to this map
`Object remove(Object key)`	Removes the mapping for this key from this map if it is present (optional operation).
`int size()`	Returns the number of key-value mappings in this map.
`Collection values()`	Returns a collection view of the values contained in this map

Implementations

The general-purpose implementations are summarized in the following table.

General-purpose Implementations

Interfaces	**Implementations**				
	Hash table	**Resizable array**	**Tree**	**Linked list**	**Hash table + Linked list**
`Set`	`HashSet`		`TreeSet`		`LinkedHashSet`
`List`		`ArrayList`		`LinkedList`	
`Map`	`HashMap`		`TreeMap`		`LinkedHashMap`

List Implementations

There are two general-purpose `List` implementations — `ArrayList` and `LinkedList`.

1. **ArrayList:** Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. Each `ArrayList` instance has a *capacity*. The capacity is the size of the array used to

store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically.

2. **LinkedList:** The LinkedList class implements the `List` interface. All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

ArrayList Method name	**Description**
addAll(int index, Collection c)	Inserts all of the elements in the specified Collection into this list, starting at the specified position.
ensureCapacity(int minCapacity)	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.
removeRange(int fromIndex, int toIndex)	Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
trimToSize()	Trims the capacity of this ArrayList instance to be the list's current size.
LinkedList Method name	**Description**
`addFirst(Object o)`	Inserts the given element at the beginning of this list.
`addLast(Object o)`	Appends the given element to the end of this list.
`Object getFirst()`	Returns the first element in the list
`Object getLast()`	Returns the last element in the list
`Object removeFirst()`	Removes and returns the first element from this list.
`Object removeLast()`	Removes and returns the last element from this list.
`ListIterator listIterator(int index)`	Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.

Set Implementations

There are three general-purpose `Set` implementations — `HashSet`, `TreeSet`, and `LinkedHashSet`.

1. TreeSet: The elements are internally stored in a search tree. It is useful when you need to extract elements from a collection in a sorted manner.
2. **HashSet: I**t creates a collection the uses a hash table for storage. The advantage of HashSet is that it performs basic operations (`add`, `remove`, `contains` and `size`) in constant time and is faster than TreeSet
3. **LinkedHashSet:** The only difference is that the LinkedHashSet maintains the order of the items added to the Set, The elements are stored in a doubly linked list.

Iterator:

The **Iterator** interface provides methods using which we can traverse any collection. This interface is implemented by all collection classes.

Methods:

`hasNext()`	`true` if there is a next element in the collection.
`next()`	Returns the next object.
`remove()`	Removes the most recent element that was returned by next()

ListIterator

`ListIterator` is implemented only by the classes that implement the `List` interface (`ArrayList`, `LinkedList`, and `Vector`). ListIterator provides the following.

Forward iteration	
`hasNext``()`	`true` if there is a next element in the collection.
`next``()`	Returns the next object.
Backward iteration	
`hasPrevious``()`	`true` if there is a previous element.
`previous``()`	Returns the previous element.
Getting the index of an element	
`nextIndex``()`	Returns index of element that would be returned by subsequent call to next().
`previousIndex``()`	Returns index of element that would be returned by subsequent call to previous().
Optional modification methods.	
`add(`*`obj`*`)`	Inserts obj in collection before the next element to be returned by `next()` and after an element that would be returned by `previous()`.
`set()`	Replaces the most recent element that was returned by `next` or `previous()`.
`remove()`	Removes the most recent element that was returned by `next()` or `previous()`.

- **HashTable**

This class implements a hashtable, which maps keys to values. It is similar to **HashMap**, but is synchronized. The important methods of the HashTable class are:

Method name	Description
`void clear()`	Clears this hashtable so that it contains no keys.
`boolean contains(Object value)`	Tests if some key maps into the specified value in this hashtable.
`boolean containsKey(Object key)`	Tests if the specified object is a key in this hashtable.
`containsValue(Object value)`	Returns true if this Hashtable maps one or more keys to this value.
`Enumeration elements()`	Returns an enumeration of the values in this hashtable.
`Set entrySet()`	Returns a Set view of the entries contained in this Hashtable.
`Object get(Object key)`	Returns the value to which the specified key is mapped in this hashtable.
`int hashCode()`	Returns the hash code value for this Map as per the definition in the Map interface.
`Boolean isEmpty()`	Tests if this hashtable maps no keys to values.
`Enumeration keys()`	Returns an enumeration of the keys in this hashtable.
`Set keySet()`	Returns a Set view of the keys contained in this Hashtable.
`Object put(Object key, Object value)`	Maps the specified key to the specified value in this hashtable.
`void putAll(Map m)`	Copies all of the mappings from the specified Map to this Hashtable These mappings will replace any mappings that this Hashtable had for any of the keys currently in the specified Map.
`void rehash()`	Increases the capacity of and internally reorganizes this hashtable, in order to accommodate and access its entries more efficiently.
`Object remove(Object key)`	Removes the key (and its corresponding value) from this hashtable.
`int size()`	Returns the number of keys in this hashtable.
`Collection values()`	Returns a Collection view of the values contained in this Hashtable.

For traversing a HashTable, use the **Enumeration** interface. The `Enumeration` interface has two methods, **`hasMoreElements`** and **`nextElement`** `which` are same as the `hasNext` and `next` methods of the `Iterator` interface.

Self Activity

1. Sample program

```
/* Program to demonstrate ArrayList and LinkedList */
import java.util.*;
class ArrayLinkedListDemo {
   public static void main(String args[])
   {
      ArrayList al = new ArrayList();
      LinkedList l1 = new LinkedList();
      System.out.println("Initial size of al: " + al.size());
      // add elements to the array list
      al.add("A");
      al.add("B");
      al.add("C");
      al.add(2, "AA");
      System.out.println("Contents of al: " + al);

      al.remove("B");
      al.remove(1);
      System.out.println("Contents of al: " + al);
        l1.add("A");
          l1.add("B");
          l1.add(new Integer(10));
          System.out.println("The contents of list is " + l1);
          l1.addFirst("AA");
          l1.addLast("c");
          l1.add(2,"D");
          l1.add(1,"E");
          l1.remove(3);
          System.out.println("The contents of list is " + l1);
        }
      }
```

2. Sample program

```
/* Program to demonstrate iterator */
import java.util.*;
public class IteratorDemo
{
  public static void main(String[] args)
{
     ArrayList a1 = new ArrayList();

al.add("C"); al.add("A"); al.add("E"); al.add("B"); al.add("D"); al.add("F");
  Iterator itr = a1.iterator();  //obtain iterator
  while(itr.hasNext())
  {
      String elt = (String)itr.next();
      System.out.println("Element = " + elt);
  }

  LinkedList l = new LinkedList();
  l.add("A");  l.add("B");  l.add("C");  l.add("D");
  ListIterator litr = l.listIterator();
  while(litr.hasNext())
  {
       String elt = (String)litr.next();
       System.out.println(elt);
   }
  System.out.println("Traversing Backwards : ");
  while(litr.hasPrevious())
        System.out.println(litr.previous());
}
}
```

3. Sample program

```
/* Program to demonstrate HashTable*/
import java.util.*;
public class HashtableDemo
{
  public static void main(String[] args)
{
    Hashtable hashtable = new Hashtable();
String str, name = null;
hashtable.put( "A", 75.2 ); // adding value into hashtable
hashtable.put( "B", 65.9 );
hashtable.put( "C", 95.1 );
hashtable.put( "D", 85.7 );

System.out.println("Retriving all keys from the Hashtable");
Enumeration keys = hashtable.keys();
while( keys. hasMoreElements() )
             System.out.println( keys.nextElement() );

  System.out.println("Retriving all values from the table");
Enumeration values = hashtable.elements();
while( values. hasMoreElements() )
                    System.out.println( values.nextElement() );
}
}
```

Lab Assignments

SET A

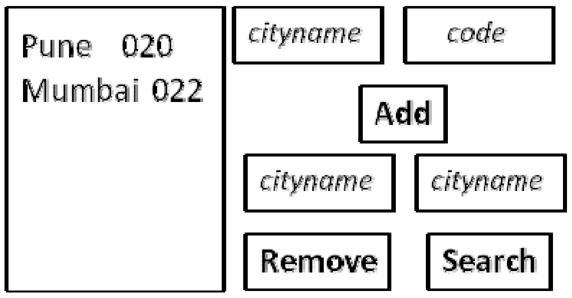
1. Accept 'n' integers from the user and store them in a collection. Display them in the sorted order. The collection should not accept duplicate elements. (Use a suitable collection). Search for an particular element using predefined search method in the Collection framework.

2. Construct a linked List containing names of colors: red, blue, yellow and orange. Then extend your program to do the following:
   - i. Display the contents of the List using an Iterator;
   - ii. Display the contents of the List in reverse order using a ListIterator;
   - iii. Create another list containing pink and green. Insert the elements of this list between blue and yellow.

3. Create a Hash table containing student name and percentage. Display the details of the hash table. Also search for a specific student and display percentage of that student.

SET B

1. Create a java application to store city names and their STD codes using an appropriate collection. The GUI ahould allow the following operations:
   i. Add a new city and its code (No duplicates)
   ii. Remove a city from the collection
   iii. Search for a cityname and display the code



SET C

1. Read a text file, specified by the first command line argument, into a `list`. The program should then display a menu which performs the following operations on the list:
1. Insert line 2. Delete line 3. Append line 4. Modify line 5. Exit
When the user selects Exit, save the contents of the list to the file and end the program.

Signature of the instructor Date / /

Assignment Evaluation **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 2: Database Programming

Objectives

- **To communicate with a database using java.**
- **To execute queries on tables.**
- **To obtain information about the database and tables**

Reading

You should read the following topics before starting this exercise:

1. The JDBC driver types
2. The design of JDBC
3. Statement, PreparedStatement, ResultSet
4. DatabaseMetaData and ResultSetMetaData

Ready Reference

JDBC : Java Database Connectivity

This API contains of a set of **classes** and **interfaces** to enable programmers to communicate with a database using java. These classes and interfaces are in the java.sql package.

The JDBC API makes it possible to do three things:

i. Establish a connection with a data source.
ii. Send queries and update statements to the data source.
iii. Process the results.

The classes and interfaces in the **java.sql** package are given below.

Interface Name	Description
`Array`	Maps to the SQL type ARRAY
`Blob`	Represents SQL BLOB Value
`CallableStatement`	To execute SQL stored procedures.
`Clob`	Represents SQL CLOB type
`Connection`	Represents a connection session with the database
`DatabaseMetaData`	Information about the database
`Driver`	Interface that every driver class must implement
`ParameterMetaData`	Information about parameters in PreparedStatement object
`PreparedStatement`	Represents precompiled SQL statement
`Ref`	Maps to SQL `REF` type
`ResultSet`	Table of data generated by executing a database query
`ResultSetMetaData`	Information about columns in a ResultSet
`Savepoint`	The representation of a savepoint, which is a point within the current transaction.
`SQLData`	For custom mapping of an SQL user-defined type (UDT) to a class in the Java programming language.
`SQLInput`	An input stream that contains a stream of values representing an instance of an SQL structured type.
`SQLOutput`	The output stream for writing the attributes of a user-defined type back to the database.
`Statement`	For executing a static SQL statement and returning the results it produces.

`Struct`	Maps to an SQL structured type.

Class Name	Description
`Date`	Represents an SQL `DATE` value.
`DriverManager`	The basic service for managing a set of JDBC drivers.
`DriverPropertyInfo`	Driver properties for making a connection.
`SQLPermission`	The permission for which the `SecurityManager` will check when code that is running in an applet calls the `DriverManager.setLogWriter` method or the `DriverManager.setLogStream` (deprecated) method.
`Time`	Represents an SQL `TIME` value.
`Timestamp`	Represents an SQL `TIMESTAMP` value.
`Types`	Defines constants that are used to identify generic SQL types, called JDBC types.

JDBC Drivers

To communicate with a database, you need a database driver. There are four types of drivers:
1. Type 1: JDBC-ODBC Bridge driver
2. Type 2: Native-API partly-Java driver:
3. Type 3: JDBC-Net pure Java driver:
4. Type 4: Native-protocol pure Java driver:

For postgresql, use the driver:
`org.postgresql.Driver`

To load the driver, use the following command:

```
Class.forName("driverName");
```

Example:
```
Class.forName("org.postgresql.Driver");
```

Establishing a connection

To establish a connection with the database, use the getConnection method of the DriverManager class. This method returns a Connection object.

```
DriverManager.getConnection("url", "user", "password");
```

Example:
```
Connection conn = DriverManager.getConnection
("jdbc:postgresql://192.168.100.4/TestDB", "scott", "tiger");
```

Methods of Connection class:

`void` **`close()`**	Releases this `Connection` object's database and JDBC resources immediately instead of waiting for them to be automatically released.
`void` **`commit()`**	Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this `Connection` object.
`Statement` **`createStatement()`**	Creates a `Statement` object for sending SQL statements to the database.
`Statement` **`createStatement`**`(int`	Creates a `Statement` object that will generate `ResultSet` objects

`resultSetType, int resultSetConcurrency)`	with the given type and concurrency.
`Boolean` **`getAutoCommit`**`()`	Retrieves the current auto-commit mode for this `Connection` object.
`DatabaseMetaData` **`getMetaData`**`()`	Retrieves a `DatabaseMetaData` object that contains metadata about the database to which this `Connection` object represents a connection.
`CallableStatement` **`prepareCall`**`(String sql)`	Creates a `CallableStatement` object for calling database stored procedures.
`CallableStatement` **`prepareCall`**`(String sql, int resultSetType, int resultSetConcurrency)`	Creates a `CallableStatement` object that will generate `ResultSet` objects with the given type and concurrency.
`PreparedStatement` **`prepareStatement`**`(String sql)`	Creates a `PreparedStatement` object for sending parameterized SQL statements to the database.
`PreparedStatement` **`prepareStatement`**`(String sql, int resultSetType, int resultSetConcurrency)`	Creates a `PreparedStatement` object that will generate `ResultSet` objects with the given type and concurrency.
`void` **`rollback`**`()`	Undoes all changes made in the current transaction and releases any database locks currently held by this `Connection` object.
`void` **`setAutoCommit`**`(boolean autoCommit)`	Sets this connection's auto-commit mode to the given state.

Executing Queries

To execute an SQL query, you have to use one of the following classes:

- Statement
- PreparedStatement
- CallableStatement

A Statement represents a general SQL statement without parameters. The method **createStatement()** creates a Statement object. A PreparedStatement represents a precompiled SQL statement, with or without parameters. The method **prepareStatement(String sql)** creates a PreparedStatement object. CallableStatement objects are used to execute SQL stored procedures. The method **prepareCall(String sql)** creates a CallableStatement object.

Executing a SQL statement with the Statement object, and returning a jdbc resultSet.

To execute a query, call an `execute` method from `Statement` such as the following:

- `execute`: Use this method if the query could return one or more `ResultSet` objects.
- `executeQuery`: Returns one `ResultSet` object.

- `executeUpdate`: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using `INSERT`, `DELETE`, or `UPDATE` SQL statements.

```
Examples:
ResultSet rs = stmt.executeQuery("SELECT * FROM Student");
int result = stmt.executeUpdate("Update authors SET name = 'abc' WHERE id = 1");
boolean ans = stmt.execute("DROP TABLE IF EXISTS test");
```

ResultSet provides access to a table of data generated by executing a Statement. The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The **next()** method is used to successively step through the rows of the tabular results.

Examples:

```
Statement stmt =  conn.prepareStatement();
ResultSet rs = stmt.executeQuery("Select * from student");
while(rs.next())
{
  //access resultset data
}
```

To access these values, there are getXXX() methods where XXX is a type *for example*, getString(), getInt() etc. There are two forms of the getXXX methods:

i. Using columnName: getXXX(String columnName)

ii. Using columnNumber: getXXX(int columnNumber)

Example

```
rs.getString("stuname"));
rs.getString(1); //where name appears as column 1 in the ResultSet
```

Using PreparedStatement

These are precompiled sql statements. For parameters, the SQL commands in a PreparedStatement can contain **placeholders** which are represented by '**?**' in the SQL command.

Example

```
String sql = "UPDATE authors SET name = ? WHERE id = ?";
PreparedStatement  ps = conn.prepareStatement(sql);
```

Before the sql statement is executed, the placeholders have to be replaced by actual values. This is done by calling a **setXXX(int n, XXX x)** method, where XXX is the appropriate type for the parameter *for example*, setString, setInt, setFloat, setDate etc, n is the placeholder number and x is the value which replaces the placeholder.

Example

```
String sql = "UPDATE authors SET name = ? WHERE id = ?";
PreparedStatement  ps = conn.prepareStatement(sql);
ps.setString(1,'abc'); //assign abc to first placeholder
ps.setInt(2,123); //assign 123 to second placeholder
```

ResultSet Scroll Types and Concurrency

The scroll type indicates how the cursor moves in the ResultSet. The concurrency type affects concurrent access to the resultset. The types are given in the table below.

Scroll Type

`TYPE_FORWARD_ONLY`	The result set is not scrollable.
`TYPE_SCROLL_INSENSITIVE`	The result set is scrollable but not sensitive to database changes.
`TYPE_SCROLL_SENSITIVE`	The result set is scrollable and sensitive to database changes.
Concurrency Type	
`CONCUR_READ_ONLY`	The result set cannot be used to update the database.
`CONCUR_UPDATABLE`	The result set can be used to update the database.

Example:

```
Statement stmt = conn.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

ResultSet Interface

The `ResultSet` interface provides methods for retrieving and manipulating the results of executed queries.

Method Name	Description
`beforeFirst()`	Default position. Puts cursor before 1st row of ResultSet.
`first()`	Puts cursor on 1st row of ResultSet.
`last()`	Puts cursor on last row of ResultSet.
`afterLast()`	Puts cursor after/beyond last row of ResultSet.
`absolute (int pos)`	Puts cursor at row number position where absolute (1) is a 1st row and absolute (-1) is last row of ResultSet.
`relative (int pos)`	Puts cursor at row no. position relative from current position.
`next()`	To move to the next row in ResultSet
`previous()`	To move to the previous row in ResultSet.
`void close()`	To close the ResultSet.
`deleteRow()`	Deletes the current row from the ResultSet and underlying database.
`getRow()`	Retrieves the current row number
`insertRow()`	Inserts the contents of the insert row into the ResultSet object and into the database.
`refreshRow()`	Refreshes the current row with its most recent value in the database.
`updateRow()`	Updates the underlying database with the new contents of the current row of this ResultSet object.
`getXXX(String columnName)`	Retrieves the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.
`moveToInsertRow()`	Moves the cursor to the insert row.
`close()`	Disposes the ResultSet.
`isFirst()`	Tests whether the cursor is at the first position.
`isLast()`	Tests whether the cursor is at the last position
`isBeforeFirst()`	Tests whether the cursor is before the first position
`isAfterLast()`	Tests whether the cursor is after the last position
`updateXXX(int columnNumber, XXX value)`	Updates the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.
`updateXXX(String columnName, XXX value)`	Updates the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.

DatabaseMetaData

This interface provides methods that tell you about the database for a given connection object.

Method Name	Description
`getDatabaseProductName()`	Retrieves the name of this database product.
`getDatabaseProductVersion()`	Retrieves the version number of this database product.
`getDriverName()`	Retrieves the name of this JDBC driver.
`getDriverVersion()`	Retrieves the version number of this JDBC driver as a String.
`getUserName()`	Retrieves the user name as known to this database.
`getCatalogs()`	Retrieves the catalog names available in this database.
`getSchemas(String catalog, String schemaPattern)`	Retrieves the schema names available in this database.
`getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)`	Retrieves a description of the tables available in the given catalog.
`getPrimaryKeys(String catalog, String schema, String table)`	Retrieves a description of the given table's primary key columns.
`getExportedKeys(String catalog, String schema, String table)`	Retrieves a description of the foreign key columns that reference the given table's primary key columns (the foreign keys exported by a table).
`getImportedKeys(String catalog, String schema, String table)`	Retrieves a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).
`getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)`	Retrieves a description of table columns available in the specified catalog.
`getProcedures(String catalog, String schemaPattern, String procedureNamePattern)`	Retrieves a description of the stored procedures available in the given catalog.
`getFunctions(String catalog, String schemaPattern, String functionNamePattern)`	Retrieves a description of the system and user functions available in the given catalog.

Example:
```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getTables(null, null, null,new String[] {"TABLE"});
while (rs.next())
         System.out.println( rs.getString("TABLE_NAME"));
```

ResultSetMetaData

The ResultSetMetaData interface provides information about the structure of a particular ResultSet.

Method Name	Description
`getColumnCount()`	Returns the number of columns in the current ResultSet object.
`getColumnDisplaySize(int column)`	Gives the maximum width of the column specified by the index parameter.
`getColumnLabel(int column)`	Gives the suggested title for the column for use in display and printouts.
`getColumnName(int column)`	Gives the column name associated with the column index.
`getColumnTypeName(int column)`	Gives the designated column's SQL type.
`isReadOnly(int column)`	Indicates whether the designated column is read-only.
`isWritable(int column)`	Indicates whether you can write to the designated column.

`isNullable(int column)`	Indicates the nullability of values in the designated column.

Example:
```
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData rsmd = rs.getMetaData();
int noOfColumns = rsmd.getColumnCount();
System.out.println("Number of columns = " + noOfColumns);
for(int i=1; i<=noOfColumns; i++)
{
   System.out.println("Column No : " + i);
   System.out.println("Column Name : " + rsmd.getColumnName(i));
   System.out.println("Column Type : " + rsmd.getColumnTypeName(i));
  System.out.println("Column display size : " + rsmd.getColumnDisplaySize(i));
}
```

Self Activity

1. Sample program to display employee data (id, name, salary)

```
import java.sql.*;
import java.io.*;
class JDBCDemo
{
  public static void main(String[] args) throws SQLException
  {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    try
    {
      Class.forName("org.postgresql.Driver");
      conn =
DriverManager.getConnection("jdbc:postgresql://192.168.100.254/employeeDB","student","");
      if(conn==null)
        System.out.println("Connection failed ");
                    else
             {
             System.out.println("Connection successful..");
             stmt = conn.createStatement();
          rs = stmt.executeQuery("Select * from emp");
          while(rs.next())
          {
          System.out.print("ID = " + rs.getInt(1));
          System.out.println("Name = " + rs.getString(2));
          System.out.println("Salary = " + rs.getInt(3));
          }
          conn.close();
     }
   }
    catch(Exception e)
    { System.out.println(e);}
  }
}// end of class
```

2. Sample program to perform insert and delete operations on employee table using PreparedStatement (id, name, salary)

```
import java.sql.*;
import java.io.*;
class JDBCDemoOp
{
  public static void main(String[] args) throws SQLException
  {
Connection conn = null;
Statement stmt =  null;
ResultSet rs = null;
PreparedStatement ps1 = null, ps2=null;
int id, sal;
String name;
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
Class.forName("org.postgresql.Driver");
conn =
DriverManager.getConnection("jdbc:postgresql://192.168.100.254/employeeDB","student","");
stmt = conn.createStatement();
ps1 = conn.prepareStatement("Insert into employee values(?,?,?)");
ps2 = conn.prepareStatement("Delete employee where ID = ?");
if(conn!=null)
  System.out.println("Connection successful..");
     System.out.println("Enter the ID, name and salary to be inserted ");
     id = Integer.parseInt(br.readLine());
     name = br.readLine();
     sal = Integer.parseInt(br.readLine());
     ps1.setInt(1,id); ps1.setString(2,name);ps1.setInt(3,sal);
     ps1.executeUpdate();

     System.out.println("Enter the ID to be deleted ");
     id = Integer.parseInt(br.readLine());
     ps2.setInt(1,id);
     ps2.executeUpdate();
     conn.close();
  }
}// end of class
```

Lab Assignments

SET A

1. Create a student table with fields roll number, name, percentage. Insert values in the table. Display all the details of the student table in a tabular format on the screen (using swing).

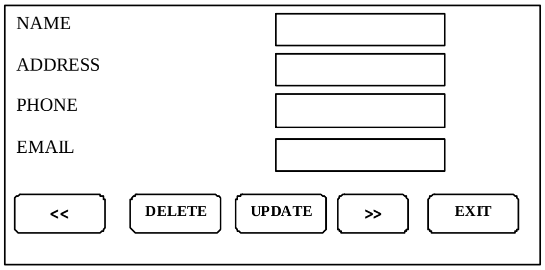
2. Write a program to display information about the database and list all the tables in the database. (Use DatabaseMetaData).

3. Write a program to display information about all columns in the student table. (Use ResultSetMetaData).

SET B

1. Write a menu driven program (Command line interface) to perform the following operations on student table.
   1. Insert 2. Modify 3. Delete 4. Search 5. View All 6. Exit

2. Design a following Phone Book Application Screen using swing & write a code for various operations like Delete, Update, Next, Previous. Raise an appropriate exception if invalid data is entered like name left blank and negative phone Number.



SET C

1. Create tables : Course (id, name, instructor) and Student (id, name). Course and Student have a many to many relationship. Create a GUI based system for performing the following operations on the tables:
   Course: Add Course, View All students of a specific course
   Student: Add Student, Delete Student, View All students, Search student

2. Design a GUI to perform the following operations on Telephone user data.
i. Add record   ii. Display current bill
Add record stores the details of a telephone user in a database. User has the following attributes: User (id, name, telephone number, number of calls, month, year).
Display current bill should Calculate and display the bill for a specific user (search by name or phone number) using the following rules. Provide button to Search user on basis of telephone number or name.
Rules: The first 100 calls are free and rent is Rs. 300)

No. Of Calls	Charge (per call)
> 100 and <=500	Rs. 1.00
> 500	Rs. 1.30

Signature of the instructor		Date	/ /

Assignment Evaluation		**Signature**			
0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 3: Servlets

Objectives

- **To understand server-side programming**
- **Defining and executing servlets**

Reading

You should read the following topics before starting this exercise:

1. Concept of servlet
2. Difference between applet and servlet
3. Introduction to Servlet (HTTP Servlet)
4. Lifecycle of a Servlet
5. Handling Get and Post requests (HTTP)
6. Data Handling using Servlet
7. Creating Cookies
8. Session Tracking using HTTP Servlet

Ready Reference

What are servlets?

Servlets are small programs that execute on the server side. Servlets are pieces of Java source code that add functionality to a web server

Servlet provides full support for sessions, a way to keep track of a particular user over time as a website's pages are being viewed. They also can communicate directly with a web server using a standard interface.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

Running servlets requires a server that supports the technologies. Several web servers, each of which has its own installation, security and administration procedures, support Servlets. The most popular one is the Tomcat- an open source server developed by the Apache Software Foundation in cooperation with Sun Microsystems version 5.5 of Tomcat supports Java Servlet.

Getting Tomcat

The software is available a a free download from Apache's website at the address http://jakarta.apache.org/tomcat. Several versions are available: Linux users should download the **rpm** of Tomcat.

The javax.servlet package

The important interfaces and classes are described in the table below.

Interface	Description
`Servlet`	A java servlet must implement the Servlet interface. This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods.

`ServletConfig`	The ServletConfig interface is used by the server to pass configuration information to a servlet. Its methods are used by the servlet to retrieve this information.
`ServletRequest`	The ServletRequest interface encapsulates a client request for service. It defines a number of methods for obtaining information about the server, requester, and request.
`ServletResponse`	The ServletResponse interface is used by a servlet to respond to a request by sending information back to the client.
`ServletContext`	The ServletContext interface defines the environment in which an applet is executed. It provides methods that are used by applets to access environment information.
`SingleThreadModel`	The SingleThreadModel interface is used to identify servlets that must be thread-safe. If a servlet implements this interface, the Web server will not concurrently execute the service() method of more than one instance of the servlet.
Class	**Description**
`GenericServlet`	The GenericServlet class implements the Servlet interface. You can subclass this class to define your own servlets.
`ServletInputStream`	The ServletInputStream class is used to access request information supplied by a Web client. An object of this class is returned by the getInputStream() method of the ServletRequest interface.
`ServletOutputStream`	The ServletOutputStream class is used to send response information to a Web client. An object of this class is returned by the getOutputStream() method of the ServletResponse interface.

The javax.servlet.http package

Interface	Description
`HttpServletRequest`	The HttpServletRequest interface extends the ServletRequest interface and adds methods for accessing the details of an HTTP request.
`HttpServletResponse`	The HttpServletResponse interface extends the ServletResponse interface and adds constants and methods for returning HTTP-specific responses.
`HttpSession`	This interface is implemented by servlets to enable them to support browser-server sessions that span multiple HTTP request-response pairs. Since HTTP is a stateless protocol, session state is maintained externally using client-side cookies or URL rewriting. This interface provides methods for reading and writing state values and managing sessions.
`HttpSessionContext`	This interface is used to represent a collection of HttpSession objects that are associated with session IDs.
Class	**Description**
`HttpServlet`	Used to create HTTP servlets. The HttpServlet class extends the GenericServlet class.
`Cookie`	This class represents an HTTP cookie. Cookies are used to maintain session state over multiple HTTP requests. They are named data values that are created on the Web server and stored on individual browser clients. The Cookie class provides the method for getting and setting cookie values and attributes.

Servlet Life Cycle

A servlet's life cycle methods function similarly to the life cycle methods of applets.

- The **init(ServletConfig)** method is called automatically when a web server first begins a servlet to handle the user's request. The init() method is called only once. ServletConfig is an interface in the javax.servlet package, containing the methods to find out more about the environment in which a servlet is running.
- The servlet action is in the **service()** method. The service() method checks the HTTP request type (GET, POST, PUT, DELETE etc.) and calls doGet(), doPost(),

doPut(), doDelete() etc. methods. A GET request results from normal request for a URL or from an HTML form that has no METHOD specified. The POST request results from an HTML form that specifically lists POST as the METHOD.

- The **destroy()** method is called when a web server takes a servlet offline.

Using Servlets

One of the main tasks of a servlet is to collect information from a web user and present something back in response. Collection of information is achieved using form, which is a group of text boxes, radio buttons, text areas, and other input fields on the web page. Each field on a form stores information that can be transmitted to a web server and then sent to a Java servlet. web browsers communicate with servers by using Hypertext Transfer Protocol (HTTP).

- Form data can be sent to a server using two kinds of HTTP requests: get and post. When web page calls a server using **get** or **post**, the name of the program that handles the request must be specified as a web address, also called uniform resource locator (URL). A get request affixes all data on a form to the end of a URL. A post request includes form data as a header and sent separately from the URL. This is generally preferred, and it’s required when confidential information is being collected on the form.
- Java servlets handle both of these requests through methods inherited from the HTTPServlet class: **doGet(HttpServletRequest, HttpServletResponse)** and **doPost(HttpServletRequest, HttpServletResponse).** These methods throw two kinds of exceptions: ServletException, part of javax.servlet package, and IOException, an exception in the java.io package.
- The **getparameter(String)** method is used to retrieve the fields in a servlet with the name of the field as an argument. Using an HTML document a servlet communicates with the user.
- While preparing the response you have to define the kind of content the servlet is sending to a browser. The **setContentType(String)** method is used to decide the type of response servlet is communicating. Most common form of response is written using an HTML as: **setContentType(“text/html”).**
- To send data to the browser, you create a servlet output stream associated with the browser and then call the **println(String)** method on that stream. The **getWriter()** method of HttpServletResponse object returns a stream. which can be used to send a response back to the client.

Example

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http.*;
public class MyHttpServlet extends HttpServlet
{
  public void doGet(HttpServletRequest req,HttpServletResponse res) throws
ServletException, IOException
{
    // Use “req” to read incoming request
    // Use “res” to specify the HTTP response status
    //Use req.getParameter(String) or getParameterValues(String) to obtain
parameters
         PrintWriter out = res.getWriter();//stream for output
    // Use "out" to send content to browser
  }
}
```

Request and Response methods

`ServletRequest` methods	
`String getParameter(String name )`	Obtains the value of a parameter sent to the servlet as part of a `get` or `post` request. The `name` argument represents the parameter name.
`Enumeration getParameterNames()`	Returns the names of all the parameters sent to the servlet as part of a `post` request.
`String[]getParameterValues(String name)`	For a parameter with multiple values, this method Returns an array of strings containing the values for a specified servlet parameter.
`String getProtocol()`	Returns the name and version of the protocol the request uses in the form *protocol/majorVersion.minorVersion*, for example, HTTP/1.
`String getRemoteAddr()`	Returns the Internet Protocol (IP) address of the client that sent the request.
`String getRemoteHost()`	Returns the fully qualified name of the client that sent the request.
`String getServerName()`	Returns the host name of the server that received the request.
`int getServerPort()`	Returns the port number on which this request was received.
`HttpServletRequest` methods	
`Cookie[] getCookies()`	Returns an array of Cookie objects stored on the client by the server.
`HttpSession getSession( boolean create )`	Returns an `HttpSession` object associated with the client's current browsing session. This method can create an `HttpSession` object (`True argument`) if one does not already exist for the client.
`String getServletPath()`	Returns the part of this request's URL that calls the servlet.
`String getMethod()`	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
`String getQueryString()`	Returns the query string that is contained in the request URL after the path.
`String getPathInfo()`	Returns any extra path information associated with the URL the client sent when it made this request.
String getRemoteUser()	Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
`ServletResponse` methods	
`ServletOutputStream getOutputStream()`	Obtains a byte-based output stream for sending binary data to the client.
`PrintWriter getWriter()`	Obtains a character-based output stream for sending text data (usually HTML formatted text) to the client.
`void setContentType(String type)`	Specifies the content type of the response to the browser. The content type is also known as MIME (Multipurpose Internet Mail Extension) type of the data. For examples, `"text/html"` , `"image/gif"` etc.
`String setContentLength(int len)`	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.

`HttpServletResponse` methods	
`void addCookie(Cookie cookie)`	Used to add a `Cookie` to the header of the response to the client.
`void sendError(int ec)`	Sends an error response to the client using the specified status.
`void sendError(int ec, String messg)`	Sends an error response to the client using the specified status code and descriptive message.
`void sendRedirect(Stirng url)`	Sends a temporary redirect response to the client using the specified redirect location URL.
`void setHeader(String name, String value)`	Sets a response header with the given name and value.

Writing, Compiling and Running Servlet
Type the first sample program of the self-activity section. After saving this servlet,compile it with the Java compiler as: javac SimpleServlet.java. After compilation a class file with name SimpleServlet.class is created.

To make the servlet available, you have to publish this class file in a folder on your web server that has been designated for Java servlets. Tomcat provides the classes sub-folder to deploy this servlet's class file. Copy this class file in this classes sub-folder, which is available on the path: tomcat/webapps/ WEB-INF/classes. Now edit the web.xml file available under WEB-INF sub-folder with the following lines:

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

Repeat the above sequence of line to run every newly created servlet. Remember, these line lines must be placed somewhere after the <web-app> tag and before the closing </web-app> tag.

After adding these lines, save web.xml file. Restart the Tomcat service and run the servlet by loading its address with a web browser as: http://localhost:8080/FirstServlet.

Using MySQL – Database Connectivity tool with servlets
Java's Servlet also provides support for data handling using MySQL database. For this you have to do few simple steps.

1. Copy the jar file mentioned in Database Connectivity assignment into the subfolder: tomcat/lib/common.
2. Edit the file .bash_profile of your login using command: vi .bash_profile.
3. Add the following line without removing any line.
   ```
   export CLASSPATH=$CLASSPATH:/$HOME/tomcat/common/lib/<jar file> used
   in database connectivity assignment.
   ```
   Example: if I have mysql-connector-java-5.1.6.jar file, I will type the line as
   ```
   export CLASSPATH=$CLASSPATH:/$HOME/tomcat/common/lib/mysql-connector-
   java-5.1.6.jar
   ```
4. Save this file. Logout from the terminal and re-login.

5. Create the table student(rno, sname) in your database. Insert few records into this table.

Session Handling

1. Using cookies
2. Using HttpSession class

1. Using Cookies

To keep the track of information about you and the features you want the site to display. This customization is possible because of a web browser features called cookies, small files containing information that a website wants to remember about a user, like username, number of visits, and other. The files are stored on the user's computer, and a website can read only the cookies on the user's system that the site has created. The default behavior of all the web browsers is to accept all cookies.

The **javax.servlet.http.Cookie** class allows us to create a cookie and send it to the browser. The methods are:

Method	Description
`int getMaxAge()`	Returns the maximum age of the cookie, specified in seconds, By default, `-1` indicating the cookie will persist until browser shutdown.
`String getName()`	Returns the name of the cookie.
`String getValue()`	Returns the value of the cookie.
`void setMaxAge(int s)`	Sets the maximum age of the cookie in seconds.
`void setValue (String value)`	Assigns a new value to a cookie after the cookie is created.

- The Cookie class in the javax.servlet.http package supports cookies. To create a cookie, call the Cookie(String,String) constructor. The first argument is the name you want to give the Cookie, and the second is the cookie's value.
- To send a cookie, call the addCookie(Cookie) method of an HttpServletResponse object. You can add more than one cookie to a response.
- In a servlet,call the getCookies() method of an HttpServletRequest object to receive an array of Cookie objects. Use getName() and getValue() methods to find out about cookie.

2.HttpSession class

Servlet can retain the state of user through **HttpSession**, a class that represents sessions. There can be one session object for each user running your servlet.

- A user's session can be created or retrieved by calling the **getSession(Boolean)** method of the servlet's request object. Use an argument true if a session should be created when one doesn't already exist for the user.

Example: HttpSession state=req.getSession(true);

```
public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
{
      HttpSession session = req.getSession(true);
// ...
}
```

- Objects held by session are called its attributes. Call the session's **setAttribute(String, Object)** method with two arguments: a name to give the attribute and the object.
- To retrieve an attribute, call the **getAttribute(String)** method with its name as the only argument. It returns the object, which must be cast from object to the desired class, or null if no attribute of that name exists.
- To remove an attribute when it's no longer needed, call **removeAttribute(String)** with its name as the argument.

Method	Description
`Object getAttribute(String name)`	Returns the object bound with the specified name in this session, or `null` if no object is bound under the name.
`Enumeration getAttributeNames()`	Returns an `Enumeration` of `String` objects containing the names of all the objects bound to this session.
`long getCreationTime()`	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
`long getLastAccessedTime()`	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
`int getMaxInactiveInterval()`	Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
`void RemoveAttribute(String name)`	Removes the object bound with the specified name from this session.
`void setAttribute(String name, Object value)`	Binds an object to this session, using the name specified.
`void setMaxInactiveInterval(int seconds)`	Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.
`void invalidate()`	Invalidates this session then unbinds any objects bound to it.
`Boolean isNew()`	Returns true if it is a new session.
`String getId()`	Returns a string containing the unique identifier assigned to this session.

Self Activity

1. Sample program

```
/* Program for simple servlet*/
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet
{
        public void service(HttpServletRequest req, HttpServletResponse rs)
        throws ServletException, IOException
  {
                rs.setContentType("text/html");
                PrintWriter pw=rs.getWriter();
                pw.println("<html>");
                pw.println("<body>");
                pw.println("<B>Hello, Welcome to Java Servlet's");
                pw.println("</body>");
                pw.println("</html>");
                pw.close();
  }
}
```

After saving this servlet, compile it with the Java compiler as: javac SimpleServlet.java.
Run the servlet using http://server-ip:8080/SimpleServlet

2. Sample program to read two numbers and return their sum

```
// Save the following code as Sum.html
<html>
<head>
<title></title>
</head>
<body>
<form method="post" action="http://server-ip:8080/AddServlet">
Enter the Number1 <input type="text" name="No1">
Enter the Number2 <input type="text" name="No2">
<br>
<input type="Submit">
</form>
</body>
</html>

// Save the following code as AddServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AddServlet extends HttpServlet{
      public void doGet(HttpServletRequest req,HttpServletResponse
res)
            throws ServletException, IOException{

            int no1=Integer.parseInt(req.getParameter("No1"));
            int no2=Integer.parseInt(req.getParameter("No2"));
            int total=no1+no2;

            res.setContentType("text/html");
            PrintWriter pw=res.getWriter();
            pw.println("<h1> ans </h1> <h3>"+total+"</h3>");
            pw.close();
      }
}
```

3. Sample program for database handling using servlet

```
//Create a student table (rno, name)
//The servlet displays all records from the student table on the client machine.
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class ServJdbc extends HttpServlet{

      String qry;
      ResultSet rs;
      Statement st;
      Connection cn;
      String ename,sal;

      public void init()
      {
            try{
                  Class.forName("org.gjt.mm.mysql.Driver");

      cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/root","root","");
            }
            catch(ClassNotFoundException ce){}
            catch(SQLException se){}

      }

      public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
      {
            res.setContentType("text/html");
            PrintWriter pw=res.getWriter();
            try{

                  qry="Select * from student";
                  st=cn.createStatement();
                  rs=st.executeQuery(qry);

                  while(rs.next())
                  {
                        pw.print("<table border=1>");
                        pw.print("<tr>");
                        pw.print("<td>" + rs.getInt(1) + "</td>");
                        pw.print("<td>" + rs.getString(2) + "</td>");
                        pw.print("</tr>");
                        pw.print("</table>");
                  }
            }
            catch(Exception se){}
            pw.close();
      }
}
```

Run this program as http://server-ip:8080/ServJdbc

4. Sample program for cookies

```
/*Save this program as AddCookie.java
*/
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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.print("Cookie added with value 1);
            Cookie c2=new Cookie("Cookie2","2");
            res.addCookie(c2);
            pw.print("Cookie added with value 2);
            pw.close();
      }
}
```

/* Save this program as GetCookie.java */

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

public class GetCookie extends HttpServlet{

      public void doGet(HttpServletRequest req,HttpServletResponse
res) throws ServletException, IOException{

            Cookie [] c=req.getCookies();
            res.setContentType("text/html");
            PrintWriter pw=res.getWriter();
            for(int i=0;i<c.length;i++)
                  pw.println("Cookie Name"+c[i].getName());
            pw.close();
      }
}
```

Run this program as http://server-ip:8080/AddCookie
Run this program as http://server-ip:8080/GetCookie

5. Sample program for sessions

```
/* Program for Session using Servlet
Save this program as SessionDemo.java
*/
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionDemo extends HttpServlet{

      String result1="success";
      String result2="failure";
      public void doGet(HttpServletRequest req,HttpServletResponse
res)
      throws ServletException, IOException{


      HttpSession hs=req.getSession(true);

      String lname=req.getParameter("txt1");
      String pwd=req.getParameter("txt2");

      res.setContentType("text/html");
      PrintWriter pw=res.getWriter();

      if((lname.equals("BCS"))&&(pwd.equals("CS")))
      {
            pw.print("<a href=http://localhost:8080/NewInfo.html>
Login Success</a>");
            hs.setAttribute("loginID",result1);
      }
      else
      {
            pw.print("<a href=http://localhost:8080/NewInfo.html>
Kick Out</a>");
            hs.setAttribute("loginID",result2);
      }
      pw.close();
      }
}
<!—HTML File for NewInfo.html -->
<html>
<head>
<title></title>
</head>
<body>
<form method="post" action="http://localhost:8080/SessionInfo">
<input type="Submit" value=”Read Session Value”>
</form>
</body>
</html>

/*Save this program as SessionInfo.java */

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

public class SessionInfo extends HttpServlet{
      String readloginid;
      public void doGet(HttpServletRequest req,HttpServletResponse
res)  throws ServletException, IOException{

      HttpSession hs=req.getSession(true);
      readloginid=hs.getId();

      res.setContentType("text/html");
      PrintWriter pw=res.getWriter();

      if(hs.getAttribute("loginID").equals("success"))
            pw.print("Your Session ID " + readloginid);
      else
            pw.print("<h1>Session Expired </h1>");
      pw.close();
      }
}
```

Create an html file for login and password and use http://server-ip:8080/SessionDemo in the Form Action tag.

Lab Assignments

SET A

1. Design a servlet that provides information about a HTTP request from a client, such as IP address and browser type. The servlet also provides information about the server on which the servlet is running, such as the operating system type, and the names of currently loaded **servlets.**

2. Write a servlet which counts how many times a user has visited a web page. If the user is visiting the page for the first time, display a welcome message. If the user is re-visiting the page, display the number of times visited. (Use cookies)

3. Design an HTML page which passes student roll number to a search servlet. The servlet searches for the roll number in a database (student table) and returns student details if found or error message otherwise.

SET B

1. Write a program to create a shopping mall. User must be allowed to do purchase from two pages. Each page should have a page total. The third page should display a bill, which consists of a page total of what ever the purchase has been done and print the total. (Use HttpSession)

2. Design an HTML page containing 4 option buttons (Painting, Drawing, Singing and swimming) and 2 buttons reset and submit. When the user clicks submit, the server responds by adding a cookie containing the selected hobby and sends a message back to the client. Program should not allow duplicate cookies to be written.

Set B: Additional Programs For Practice

1. Design the table Login(login_name, password) using MySQL. Also design an HTML login screen. Accept the login name and password from the user. The servlet accepts the login name and password and validates it from the database you have created. The servlet sends back an appropriate response. Also calculate the number of time that user has successfully performed the login activity. Use cookies.

SET C

1. Consider the following entities and their relationships
   Movie (movie_no, movie_name, release_year)
   Actor(actor_no, name)
   Relationship between movie and actor is many – many with attribute rate in Rs. Create a RDB in 3 NF answer the following:
   a) Accept an actor name and display all movie names in which he has acted along with his name on top.
   b) Accept a movie name and list all actors in that movie along with the movie name on top.

Signature of the instructor		Date	/ /

Assignment Evaluation **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 4: Java Server Pages

Objectives

- **To demonstrate the use of JSP**

Reading

You should read the following topics before starting this exercise

1. Concept of Servlets.
2. JSP life-cycle.
3. JSP Directives
4. Scripting elements.
5. Actions in JSP.

Ready Reference

What is JSP?

JSP is Java Server Page, which is a dynamic web page and used to build dynamic websites. To run jsp, we need web server which can be tomcat provided by apache, it can also be jRun, jBoss(Redhat), weblogic (BEA) , or websphere(IBM).

JSP is dynamic file whereas Html file is static. HTML can not get data from database or dynamic data. JSP can be interactive and communicate with database and controllable by programmer. It is saved by extension of **.jsp**. Each Java server page is compiled into a servlet before it can be used. This is normally done when the first request to the JSP page is made.

A JSP contains 3 important types of elements:-

1. **Directives:-** these are messages to the JSP container that is the server program that executes JSPs.
2. **Scripting elements:-** These enables programmers to insert java code which will be a part of the resultant servlet.
3. **Actions:-** Actions encapsulates functionally in predefined tags that programmers can embedded in a JSP.

JSP Directives:-

Directives are message to the JSP container that enable the programmer to specify page setting to include content from other resources & to specify custom tag libraries for use in a JSP.

Syntax:-

<%@ name attribute1=”….”, attribute2=”…”…%>

Directive	Description
`page`	Defines page settings for the JSP container to process.
`include`	Causes the JSP container to perform a translation-time insertion of another resource's content. The file included can be either static (HTML file) or dynamic (i.e., another tag file)
`taglib`	Allows programmers to use new tags from tag libraries that encapsulate more complex functionality and simplify the coding of a JSP.

Page Directive:-
The page directives specify global settings for the JSP in the JSP container. There can be many page directives, provided that there is only one occurrence of each attribute.

Syntax:-
```
<%@ page
  [ language="java" ]
  [ extends="package.class" ]
  [ import="{package.class | package.*}, ..." ]
  [ session="true|false" ]
  [ buffer="none|8kb|sizekb" ]
  [ autoFlush="true|false" ]
  [ isThreadSafe="true|false" ]
  [ info="text" ]
  [ errorPage="relativeURL" ]
  [ contentType="mimeType [ ; charset=characterSet ]" |
    "text/html ; charset=ISO-8859-1" ]
  [ isErrorPage="true|false" ]
  [ pageEncoding="characterSet | ISO-8859-1" ]    %>
```

Scripting Elements

1. Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file.

Syntax
```
<%! Java declaration statements %>
```
Example,
```
<%! private int count = 0; %>
<%! int i = 0; %>
```
2. Expressions
An expression element contains a java expression that is evaluated, converted to a `String`, and inserted where the expression appears in the JSP file.

Syntax
```
<%= expression %>
```
Example,
```
Your name is <%= request.getParameter("name") %>
```
3. Scriptlet
A scriptlet contains a set of java statements which is executed. A scriptlet can have java variable and method declarations, expressions, use implicit objects and contain any other statement valid in java.

Syntax
```
<% statements %>
```
Example
```
<%
   String name = request.getParameter("userName");
       out.println("Hello  " + name);
%>
```

Implicit objects used in JSP

Implicit object	Description
`application`	A javax.servlet.ServletContext object that represents the container in which the JSP executes. It allows sharing information between the jsp page's servlet and any web components with in

	the same application.
config	A javax.servlet.ServletConfig object that represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor (web.xml). The method getinitparameter() is used to access the initialization parameters.
exceptio n	A java.lang.Throwable object that represents an exception that is passed to a JSP error page. This object is available only in a JSP error page.
out	A javax.servlet.jsp.JspWriter object that writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
page	An `Object` that represents the current JSP instance.
pageCont ext	A javax.servlet.jsp.PageContext object that provides JSP programmers with access to the implicit objects discussed in this table.
request	An object that represents the client request and is normally an instance of a class that implements HttpServletRequest. If a protocol other than HTTP is used, this object is an instance of a subclass of javax.servlet.Servlet-Request. It uses the getParameter() method to access the request parameter.
response	An object that represents the response to the client and is normally an instance of a class that implements HttpServletResponse (package javax.servlet.http). If a protocol other than HTTP is used, this object is an instance of a class that implements javax.servlet.ServletResponse.
session	A javax.servlet.http.HttpSession object that represents the client session information. This object is available only in pages that participate in a session.

To run JSP files: all JSP code should be copied (Deployed) into webapps folder in the tomcat server. To execute the file, type: http://server-ip:8080/Filename.jsp

Self Activity

1. Sample program : Simple display on browser

```
/* type this as first.jsp */
<html> <body>
<%
out.print("Hello World!");
%>
</body> </html>
```

2. Sample program to display current date

```
<%@ page  language="java" import="java.util.*" %>
<html> <body>
Current Date time: <%=new java.util.Date()%>
</body> </html>
```

3. Sample program to add two numbers “AddNumbers.jsp”

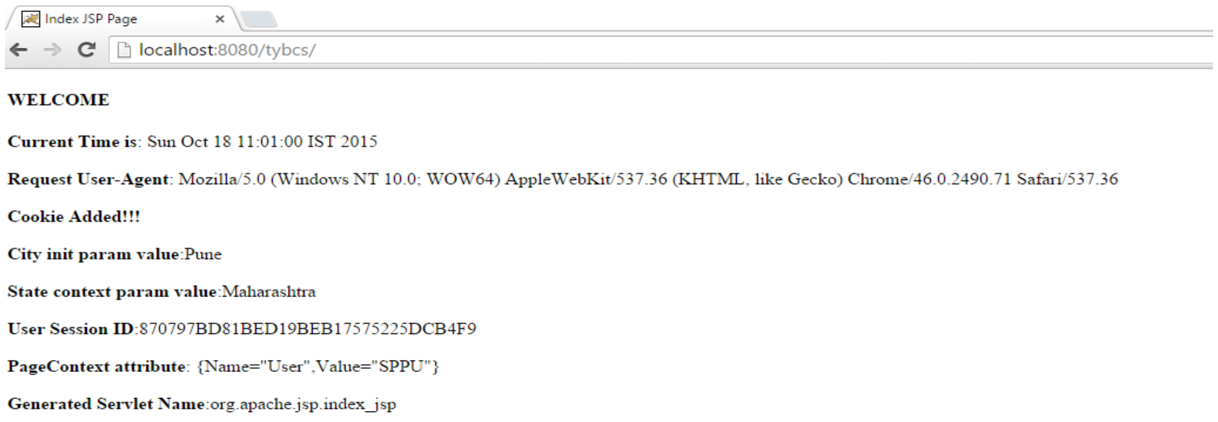
```
<%@ page language="java"%>
<html> <head>
<title>Add number program in JSP</title>
</head>
<body>
<form method = “post” action = “AddNumbers.jsp”>
Enter Number 1 <input type =”text” name = “No1”>
Enter Number 2 <input type =”text” name = “No2”>
<input type="submit" value="Get Result"/>
<%
 int a=Integer.parseInt(request.getParameter(“No1”));
 int b=Integer.parseInt(request.getParameter(“No2”));
  int result=a+b;
  out.print("Additon of a and b :"+result);
%>
```

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

Lab Assignments

SET A

1. Write a Program to make use of following JSP implicit objects:
    i. out: To display current Date and Time.
    ii. request: To get header information.
    iii. response: To Add Cookie
    iv. config: get the parameters value defined in <init-param>
    v. application: get the parameter value defined in <context-param>
    vi. session: Display Current Session ID
    vii. pageContext: To set and get the attributes.
    viii. page: get the name of Generated Servlet



```
WELCOME

Current Time is: Sun Oct 18 11:01:00 IST 2015

Request User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.71 Safari/537.36

Cookie Added!!!

City init param value:Pune

State context param value:Maharashtra

User Session ID:870797BD81BED19BEB17575225DCB4F9

PageContext attribute: {Name="User",Value="SPPU"}

Generated Servlet Name:org.apache.jsp.index_jsp
```

2. Create a JSP page which will accept the file extension and display all files in the current directory having that extension. Each filename should appear as a hyperlink on screen.

SET B

1. Create a JSP page, which accepts user name in a text box and greets the user according to the time on server side. Example: User name : ABC
Output : Good morning ABC / Good Afternoon ABC/ Good Evening ABC

2. Create a JSP page for an online multiple choice test. The questions are randomly selected from a database and displayed on the screen. The choices are displayed using radio buttons. When the user clicks on next, the next question is displayed. When the user clicks on submit, display the total score on the screen.

Signature of the instructor		Date	/ /

Assignment Evaluation **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 5: Multithreading

Objectives

- **To create and use threads in java**
- **To demonstrate multithreading**

Reading

You should read the following topics before starting this exercise:
1. Thread class
2. Runnable interface
3. Thread lifecycle
4. Thread methods

Ready Reference

Introduction

Nearly every operating system supports the concept of processes -- independently running programs that are isolated from each other to some degree. Threading is a facility to allow multiple activities to coexist within a single process. Java is the first mainstream programming language to explicitly include threading within the language itself.

A process can support multiple threads, which appear to execute simultaneously and asynchronously to each other. Multiple threads within a process share the same memory address space, which means they have access to the same variables and objects, and they allocate objects from the same heap.

Every Java program uses threads:-
Every Java program has at least one thread -- the **main** thread. When a Java program starts, the JVM creates the main thread and calls the program's main() method within that thread. The JVM also creates other threads that are mostly invisible to you -- for example, threads associated with garbage collection, object finalization, and other JVM housekeeping tasks. Other facilities create threads too, such as the AWT or Swing UI toolkits, servlet containers, application servers, and RMI (Remote Method Invocation).

Thread Lifecycle:

The lifecycle of thread consist of several states which thread can be in. Each thread is in one state at any given point of time.
1. New State: - Thread object was just created. It is in this state before the start () method is invoked. At this point the thread is considered not alive.
2. Runnable or ready state:- A thread starts its life from Runnable state. It enters this state the time start() is called but it can enter this state several times later. In this state, a thread is ready to run as soon as it gets CPU time.
3. Running State:- In this state, a thread is assigned a processor and is running. The thread enters this state only after it is in the Runnable state and the scheduler assigns a processor to the thread.

4. Sleeping state: - When the sleep method is invoked on a running thread, it enters the Sleeping state.
5. Waiting State:- A thread enters the waiting state when the wait method is invoked on the thread. When some other thread issues a notify () or notifyAll (), it returns to the Runnable state().
6. Blocked State: - A thread can enter this state because it is waiting for resources that are held by another thread – typically I/O resources.
7. Dead State:- This is the last state of a thread. When the thread has completed execution that is its run () method ends, the thread is terminated. Once terminated, a thread can't be resumed.

Thread Creation

There are two ways to create thread in java;

1. Implement the Runnable interface (java.lang.Runnable)
2. By Extending the Thread class (java.lang.Thread)

1. Implementing the Runnable Interface

One way to create a thread in java is to implement the Runnable Interface and then instantiate an object of the class. We need to override the run() method into our class which is the only method that needs to be implemented. The run() method contains the logic of the thread.

The procedure for creating threads based on the Runnable interface is as follows:

1. A class implements the Runnable interface, providing the run() method that will be executed by the thread. An object of this class is a Runnable object.
2. An object of Thread class is created by passing a Runnable object as argument to the Thread constructor. The Thread object now has a Runnable object that implements the run() method.
3. The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned.
4. The thread ends when the run() method ends, either by normal completion or by throwing an uncaught exception.

Example:

```
class MyRunnable implements Runnable
{
  public void run() //define run method
       {
              //thread action
       }
}
...
MyRunnable r = new MyRunnable(); //create object
Thread t = new Thread(r); //create Thread object
t.start(); //execute thread
```

2. Extending java.lang.Thread class

A class can also extend the `Thread` class to create a thread. When we extend the Thread class, we should override the run method to specify the thread action.

```
class MyThread extends Thread
```

```
{
  public void run() //override run method
       {
             //thread action
       }
}
...
MyThread t = new MyThread(); //create Thread object
t.start(); //execute thread
```

Important methods of the Thread class:

Method	Description
`static int activeCount()`	Returns the number of active threads in the current thread's thread group.
`static Thread currentThread()`	Returns a reference to the currently executing thread object.
`String getName()`	Returns this thread's name.
`int getPriority()`	Returns this thread's priority.
`ThreadGroup getThreadGroup()`	Returns the thread group to which this thread belongs.
`void interrupt()`	Interrupts this thread.
`boolean isAlive()`	Tests if this thread is alive.
`boolean isDaemon()`	Tests if this thread is a daemon thread.
`boolean isInterrupted()`	Tests whether this thread has been interrupted.
`void join()`	Waits for this thread to end.
`void setName(String name)`	Changes the name of this thread.
`void setPriority(int newPriority)`	Changes the priority of this thread.
`void sleep(long mSec)`	Causes the currently executing thread to sleep.
`void start()`	Causes this thread to begin execution.
`String toString()`	Returns a string representation of this thread, including the thread's name, priority, and thread group.
`static void yield()`	Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Thread priorities

Every thread has a priority. A priority is an integer from 1 to 10 inclusive, where 10 is the highest priority, referred to as the ***maximum priority***, and 1 is the lowest priority, also known as the ***minimum priority***. The normal priority is 5, which is the default priority for each thread.

To set a thread’s priority, use the setPriority() method. You can use an integer from 1 to 10, or you can use static, final variables defined in the Thread class. These variables are

i. Thread.MIN_PRIORITY: Minimum priority i.e. 1
ii. Thread.MAX_PRIORITY: Maximum priority i.e. 10
iii. Thread.NORM_PRIORITY: Default priority i.e. 5

Interthread Communication

When multiple threads are running, it becomes necessary for the threads to communicate with each other. The methods used for inter-thread communication are join(), wait(), notify() and notifyAll().

Self Activity

1. Below is a program that illustrates instantiation and running of threads using the Runnable interface.

```
class RunnableThread implements Runnable {
    Thread runner;
    public RunnableThread() {
    }
    public RunnableThread(String threadName) {
        runner = new Thread(this, threadName); // (1) Create a new
thread.
        System.out.println(runner.getName());
        runner.start(); // (2) Start the thread.
    }
    public void run() {
        //Display info about this particular thread
        System.out.println(Thread.currentThread());
    }
}

public class RunnableExample {
    public static void main(String[] args) {
        Thread thread1 = new Thread(new RunnableThread(), "thread1");
        Thread thread2 = new Thread(new RunnableThread(), "thread2");
        RunnableThread thread3 = new RunnableThread("thread3");
        //Start the threads
        thread1.start();
        thread2.start();
        try {
            //delay for one second
            Thread.currentThread().sleep(1000);
        } catch (InterruptedException e) {
        }
        //Display info about the main thread
        System.out.println(Thread.currentThread());
    }
}
```

2. Creating multiple threads using the Thread class.

```
class MyThread extends Thread
{
      String message;
      MyThread(String message)
      {
            this.message = message;
      }
      public void run()
      {
            try
            {
                  for(int i=1; i<=5; i++)
                  {
                        System.out.println(message + ”-” + i);
                        Thread.sleep(5000); //sleep for 5 seconds
                  }
            }
            catch(InterruptedException ie) { }
      }
}
public class MultipleThreadDemo
{
      public static void main( String[] args)
      {
            MyThread t1 = new MyThread("One");
        MyThread t2 = new MyThread("Two");
        System.out.println(t1);
        System.out.println(t2);
        t1.start();
        t2.start();
         }
}
```

Lab Assignments

SET A

1. Write a program that create 2 threads – each displaying a message (Pass the message as a parameter to the constructor). The threads should display the messages continuously till the user presses ctrl-c. Also display the thread information as it is running.

2. Write a java program to calculate the sum and average of an array of 1000 integers (generated randomly) using 10 threads. Each thread calculates the sum of 100 integers. Use these values to calculate average. [Use join method]

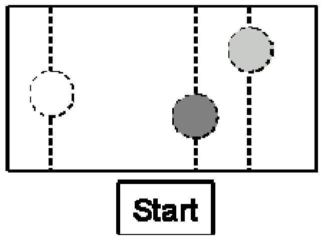
Set A Extra programs for practice

1. Define a thread called "PrintText_Thread" for printing text on command prompt for n number of times. Create three threads and run them. Pass the text and n as parameters to the thread constructor. Example:
   i. First thread prints "I am in FY" 10 times
   ii. Second thread prints "I am in SY" 20 times
   iii. Third thread prints "I am in TY" 30 times

SET B

1. Write a program for a simple search engine. Accept a string to be searched. Search for the string in all text files in the current folder. Use a separate thread for each file. The result should display the filename, line number where the string is found.

2. Define a thread to move a ball inside a panel vertically. The Ball should be created when user clicks on the Start Button. Each ball should have a different color and vertical position (calculated randomly). **Note**: Suppose user has clicked buttons 5 times then five balls should be created and move inside the panel. Ensure that ball is moving within the panel border only.



SET C

1. Write a java program to create a class called FileWatcher that can be given several filenames. The class should start a thread for each file name. If the file exists, the thread will write a message to the console and then end. If the filw does not exist, the thread will check for the existence of its file after every 5 seconds till the file gets created.

2.Write a program to simulate traffic signal using threads.

3. Write a program to show how three thread manipulate same stack , two of them are pushing elements on the stack, while the third one is popping elements off the stack.

Signature of the instructor Date / /

Assignment Evaluation **Signature**

0: Not done	2: Late Complete	4: Complete
1: Incomplete	3: Needs improvement	5: Well Done

Assignment 6: Networking

Objectives

- **Introduction to the java.net package**
- **Connection oriented transmission – Stream Socket Class**

Reading

You should read the following topics before starting this exercise

1. Networking Basics
2. The java.net package – InetAddress class, URL class, URLConnection class
3. Connection oriented communication –ServerSocket class, Socket class

Ready Reference

Introduction

One of the important features of Java is its networking support. Java has classes that range from low-level TCP/IP connections to ones that provide instant access to resources on the World Wide Web. Java supports fundamental networking capabilities through java.net package.

Networking Basics

Protocol

A protocol is a set of rules and standards for communication. A protocol specifies the format of data being sent over the Internet, along with how and when it is sent. Three important protocols used in the Internet are:
1. IP (Internet Protocol): is a network layer protocol that breaks data into small packets and routes them using IP addresses.
2. TCP (Transmission Control Protocol): This protocol is used for connection-oriented communication between two applications on two different machines. It is most reliable and implements a connection as a stream of bytes from source to destination.
3. UDP (User Datagram Protocol): It is a connection less protocol and used typically for request and reply services. It is less reliable but faster than TCP.

Addressing

1. **MAC Address**: Each machine is uniquely identified by a physical address, address of the network interface card. It is 48-bit address represented as 12 hexadecimal characters:
   For Example: 00:09:5B:EC:EE:F2
2. **IP Address**: It is used to uniquely identify a network and a machine in the network. It is referred as global addressing scheme. Also called as logical address. Currently used type of IP addresses is: Ipv4 – 32-bit address and Ipv6 – 128-bit address.
   For Example:
   Ipv4 – 192.168.16.1   Ipv6 – 0001:0BA0:01E0:D001:0000:0000:D0F0:0010
3. **Port Address:** It is the address used in the network to identify the application on the network.

Domain Name Service (DNS)

It is very difficult to remember the IP addresses of machines in a network. Instead, we can identify a machine using a “domain name” which is character based naming mechanism. Example: www.google.com. The mapping between the domain name and the IP address is done by a service called DNS.

URL

A URL (Uniform Resource Locator) is a unique identifier for any resource located on the Internet. The syntax for URL is given as:

```
<protocol>://<hostname>[:<port>][/<pathname>][/<filename>[#<section>]]
```

For Example: http://java.sun.com/j2se/1.5.0/download.jsp

Sockets

A socket represents the end-point of the network communication. It provides a simple read/write interface and hides the implementation details network and transport layer protocols. It is used to indicate one of the two end-points of a communication link between two processes. When client wishes to make connection to a server, it will create a socket at its end of the communication link.
The socket concept was developed in the first networked version of UNIX developed at the University of California at Berkeley. So sockets are also known as Berkeley Sockets.

Ports

A port number identifies a specific application running in the machine. A port number is a number in the range 1-65535.
Reserved Ports: TCP/IP protocol reserves port number in the range 1-1023 for the use of specified standard services, often referred to as “well-known” services.

Client-Server

It is a common term related to networking. A server is a machine that has some resource that can be shared. A server may also be a proxy server. Proxy server is a machine that acts as an intermediate between the client and the actual server. It performs a task like authentications, filtering and buffering of data etc. it communicates with the server on behalf of the client.
A client is any machine that wants to use the services of a particular server. The server is a permanently available resource, while the client is free to disconnect after it’s job is done.

The java.net Package

The java.net package provides classes and interfaces for implementing network applications such as sockets, network addresses, Uniform Resource Locators (URLs) etc. some important interfaces, classes and exceptions are :

InetAddress Class

This class is used to represent numerical IP addresses and domain name for that address i.e. it supports both numeric IP address and hostnames. There are no public constructors for InetAddress class. Instead, there are static methods that return InetAddress instances. Such methods are called as **factory methods**.

1. static InetAddress getLocalHost(): represents the IP address of the local host.
2. static InetAddress getByName(String hostname): represents the IP address of the host name passed to it. hostName can be "www.google.com", or hostname can be "130.95.72.134".
3. static InetAddress[] getAllByName(String hostname): represents an array of IP addresses of the specified host name.

All of these methods throw UnknownHostException.

Some of the instance methods of the InetAddress class are:

1. byte[] getAddress(): returns the raw IP address of the InetAddress object.
2. String getHostAddress(): returns the IP address string.
3. String getHostName(): Gets the host name for the IP address.

URL Class

As the name suggests, it provides a uniform way to locate resources on the web. Every browser uses them to identify information on the web. The URL class provides a simple API to access information across net using URLs.

The class has the following constructors:

1. **URL(String url_str)**: Creates a URL object based on the string parameter. If the URL cannot be correctly parsed, a MalformedURLException will be thrown.
2. **URL(String protocol, String host, String path)**: Creates a URL object with the specified protocol, host and path.
3. **URL(String protocol, String host, int port, String path)**: Creates a URL object with the specified protocol, host, port and file path.
4. **URL(URL urlObj, String urlSpecifier)**: Allows you to use an existing URL as a reference context and then create a new URL from that context.

Some important methods are given below:

Method	Description
URLConnection openConnection()	This method establishes a connection and returns a stream
Object getContent()	This method makes a connection and reads the contents
String getFile()	This method returns the filename part of the URL.
String getHost()	This method returns only the host name part from the URL.
String getPort()	This method returns the port number part from the URL. This is an optional component and returns –1 if not present.
String getProtocol()	This method returns the name of the protocol used in URL.
String getRef()	This method returns the anchor reference part from the URL.
int equals(Object)	This method compares whether two URL objects represents the same resource.
InputStream openStream()	This method establishes a connection and returns a stream for reading it.

URLConnection Class

This class is useful to actually connect to a website or resource on a network and get all the details of the website. Using the openConnection() method, we should establish a contact with the site on Internet. Method returns URLConnection object. Then using

URLConnection class methods, we can display all the details of the website and also content of the webpage whose name is given in URL.

Some important methods are:

Method	**Description**
void connect()	Establishes a connection between the application and the resource
Object getContent()	Reads the contents of the resource
int getContentLength()	Returns the value of the “content-length” header field, if such a field exists. Returns –1 if no content-length field was specified.
String getContentType()	Returns the value of the “content-type” header field, if such a field exists. Returns null if no content-type field was specified.
long getExpiration()	Returns the value of the “Expires” header field, expressed as the number of seconds since January 1, 1970 GMT. If no such a header field was specified, a value of zero will be returned.
InputStream getInputStream()	Returns an InputStream object that reads the contents of the resource pointed to by the URLConnection
OutputStream getOutputStream()	Returns an OutputStream object that writes to the remote connection
URL getURL()	Returns a URL object representing the location of the resource pointed by the URL connection
long getDate()	Returns the value of the “Date” header field, expressed as the number of seconds since January 1, 1970 GMT. Returns 0 if not specified.
long getLastModified()	Returns the date of the “Last-modified” header field, expressed as the number of seconds since January 1 1970 GMT. Returns 0 if not specified.

Connection Oriented Communication

Using Socket Java performs the network communication. Sockets enables to transfer data through certain port. Socket class of Java makes it easy to write the socket programs. Sockets are broken into two types:

1. **Datagram sockets (UDP Socket)**

Java uses java.net.DatagramSocket class to create datagram socket. The java.net.DatagramPacket represents a datagram.

2. **Stream Socket (TCP Socket)**

Java uses java.net.Socket class to create stream socket for client and java.net.ServerSocket for server.

ServerSocket Class

This class is used to create a server that listens for incoming connections from clients. It is possible for client to connect with the server when server socket binds itself to a specific port. The various constructors of the ServerSocket class are:

1. **ServerSocket(int port):** binds the server socket to the specified port number. If 0 is passed, any free port will be used. However, clients will be unable to access the service unless notified the port number.
2. **ServerSocket(int port, int numberOfClients)**: Binds the server socket to the specified port number and allocates sufficient space to the queue to support the specified number of client sockets.
3. **ServerSocket(int port, int numberOfClients, InetAddress address):** Binds the server socket to the specified port number and allocates sufficient space to the queue to support the specified number of client sockets and the IP address to which this socket binds.

The various methods of this class are:

Method	Description
accept()	Listens for socket connection. This is a blocking I/O operation, and will not return until a connection is made. When a connection is established a Socket object will be returned.
getLocalSocketAddresss()	Returns local socket information.
bind()	Binds a connection to destination.
close()	Closes the connection.
getChannel()	Returns channel for connection.
getInetAddress()	Returns address of connection.
getLocalPort()	Returns local port used for connection.
isBound()	Checks whether socket is bound.
isClosed()	Checks whether socket is closed.
toString()	Converts server socket to string.

Socket Class

This class is used to represents a TCP client socket, which connects to a server socket and initiate protocol exchanges. The various constructors of this class are:

1. **Socket(InetAddress address, int port):** creates a socket connected to the specified IP address and port. Can throw an IOException.
2. **Socket(String host, int port)**: creates a socket connected to the specified host and port. Can throw an UnknownHostException or an IOException.

The various methods of this class are:

Method	Description
getLocalSocketAddresss()	Returns local socket information.
bind()	Binds a connection to destination.
close()	Closes the connection.
connect()	Makes connection to destination
getChannel()	Returns channel for connection.
getInetAddress()	Returns address of connection.
getLocalPort()	Returns local port used for connection.
getInputStream()	Returns input stream for connection.
getKeepAlive()	Returns indication of keep alive option enabled.
getLocalAddress()	Returns local address
getOutputStream()	Returns output stream for connection.
getPort()	Returns remote used for connection.
getRemoteSocketAddress()	Returns remote socket information.
isBound()	Checks whether socket is bound.
isClosed()	Checks whether socket is closed.
isConnected()	Checks whether socket is connected.
toString()	Converts server socket to string.

Self Activity

1. Sample program to find IP address of a web-site (Internet connection required)

```
/* Program to find the IPAddress of a website*/
```

```
import java.io.*;
import java.net.*;
class AddressDemo
{
      public static void main(String args[]) throws IOException
      {
            BufferedReader br=new BufferedReader(new
             InputStreamReader(System.in));
            System.out.print(“Enter a website name: ”);
            String site=br.readLine();
            try
            {
                  InetAddress ip=InetAddress.getByName(site);
                  System.out.print(“The IP address is: ” + ip);
            }
            catch(UnknownHostException ue)
            {
                  System.out.println(“WebSite not found”);
            }
      }
}
```

2. Sample program for client-server communication

/* Program to display socket information on client machine*/

```
import java.net.*;
import java.io.*;
public class Server
{
      public static void main(String args[])throws   UnknownHostException ,
IOException
      {
            ServerSocket ss = new ServerSocket(4444);
            System.out.println("Server Started");
            Socket s = ss.accept();
            System.out.println("Connected to client");
      }
}

public class Client
{
      public static void main(String args[]) throws  UnknownHostException,
IOException
      {
            Socket s = new Socket ("localhost", 4444);
            System.out.println (s.getInetAddress());
            System.out.println (s.getPort());
            System.out.println (s.getLocalPort());
            s.close();
      }
}
```

Lab Assignments

SET A

1. Write a client-server program which displays the server machine’s date and time on the client machine.

2. Write a program which sends the name of a text file from the client to server and displays the contents of the file on the client machine. If the file is not found, display an error message.

SET B

1. Write a program to accept a list of file names on the client machine and check how many exist on the server. Display appropriate messages on the client side.

2. Write a server program which echoes messages sent by the client. The process continues till the client types "END".

SET C

1. Write a program for a simple GUI based chat application between client and server.

Signature of the instructor		Date	/ /

Assignment Evaluation **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	