CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 2– Mar. 2025

Sub:	**Advanced Java& J2EE**							**Sub Code:**	**22MCA341**
Date:	**13/3/2025**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**III**	**Branch:**	**MCA**

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE CO	OBE RBT
	PART I			
1	Describe ArrayList class and explain its constructors. Demonstrate its usage with an example program	10	CO1	L2
	OR			
2.	Create a class STUDENT with 2 private string members : USN,NAME using Linked List class in java. Write a program to add atleast 3 objects of aboveSTUDENT class. Also display the data in neat format.	10	CO2	L4
	PART II			
3	Explain the following interfaces in detail i)Queue ii)Sorted Set	10	CO1	L2
	OR			
4.	Explain the constructors of Tree Set class and write a java program to create Tree Set collection and access via iterator.	10	CO1,CO2	L2
	PART III			
5	Explain the following legacy classes with an example. i)Hash Table ii)Vector	10	CO1	L2
	OR			
6	List and Explain JDBC Driver types.	10	CO3	L2
	PART IV			
7	Describe the various steps of JDBC process with code snippets.	10	CO3	L2
	OR			
8	Discuss in detail the need of preparedStatement with an example program.	10	CO3	L2
	PART V			
9	Discuss the following map classes with example. a) Hash Map b)Tree Map	10	CO1	L2
	OR			
10	Develop a program to insert following data into music database using prepared Statement object. Table consists of music_id int(5),music_name varchar(20),music_author varchar(20)	10	CO3	L4

1. **Describe ArrayList class and explain its constructors. Demonstrate its usage with an example program**

The ArrayList class extends AbstractList and implements the List interface. ArrayList is a
generic class that has this declaration:
class ArrayList<E>
Here, E specifies the type of objects that the list will hold.
ArrayList supports dynamic arrays that can grow as needed. In Java, standard arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold. But, sometimes, we may not know until run time precisely how large an array we need. To handle this situation, the Collections Framework defines ArrayList. In essence, an ArrayList is a variable-length array of object references. That is, an ArrayList can dynamically increase or decrease in size. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array can be shrunk.
ArrayList has the constructors shown here:
ArrayList( )
ArrayList(Collection<? extends E> c)
ArrayList(int capacity)
The first constructor builds an empty array list. The second constructor builds an array list that is initialized with the elements of the collection c. The third constructor builds an array list that has the specified initial capacity. The capacity is the size of the underlying array that is used to store the elements. The capacity grows automatically as elements are added to an array list.

```
// Demonstrate ArrayList.
import java.util.*;
class ArrayListDemo {
public static void main(String args[]) {
// Create an array list.
ArrayList<String> al = new ArrayList<String>();
System.out.println("Initial size of al: " +
al.size());
// Add elements to the array list.
al.add("C");
al.add("A");
al.add("E");
al.add("B");
al.add("D");
al.add("F");
al.add(1, "A2");
System.out.println("Size of al after additions: " +
al.size());
// Display the array list.
System.out.println("Contents of al: " + al);
// Remove elements from the array list.
al.remove("F");
al.remove(2);
System.out.println("Size of al after deletions: " +
al.size());
System.out.println("Contents of al: " + al);
}
 }
```

2. **Create a class STUDENT with 2 private string members: USN, NAME using Linked List class in java. Write a program to add atleast 3 objects of above STUDENT class. Also display the data in neat format**

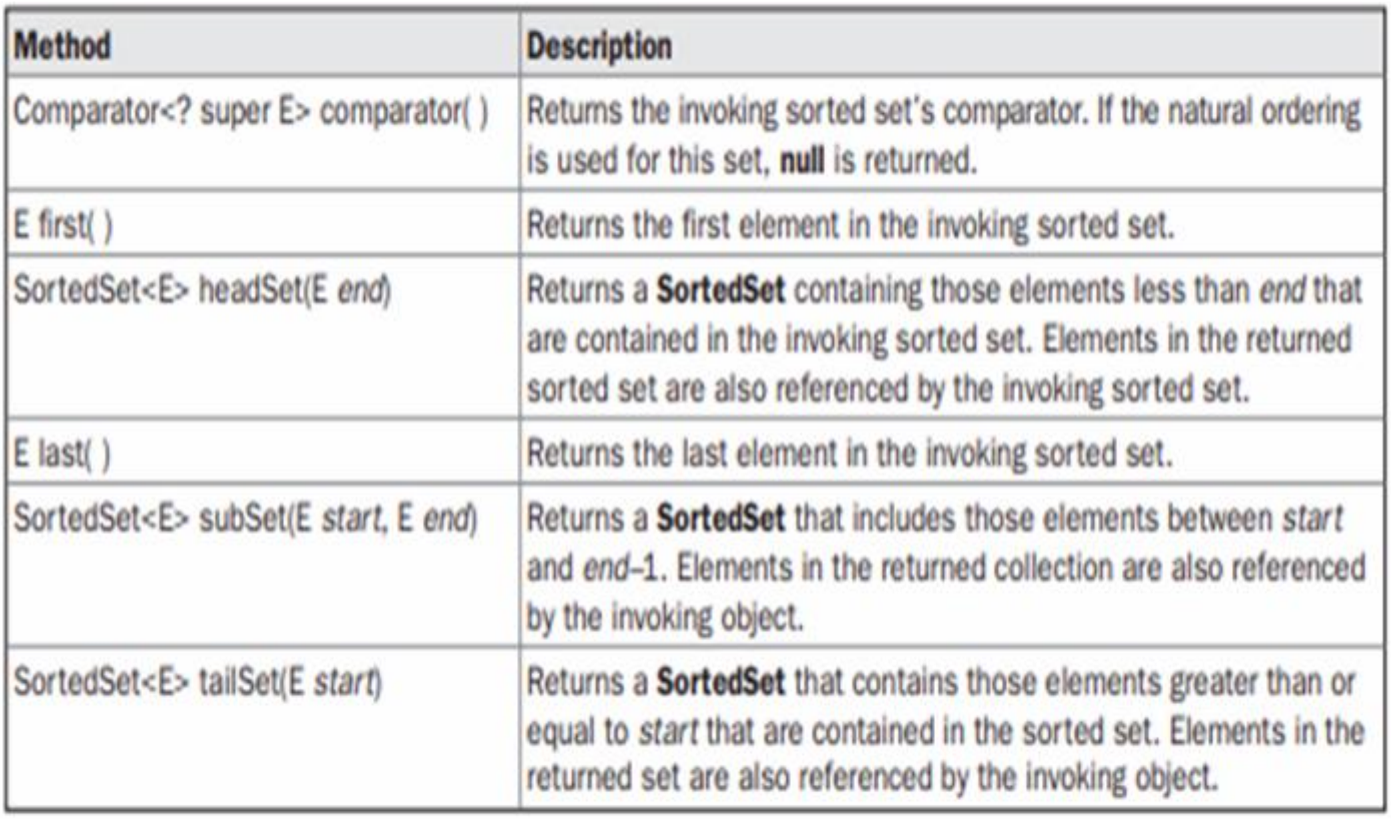
```
import java.util.LinkedList;
import java.util.Iterator;

class Student {
   private String USN;
   private String name;
```

```
    public Student(String USN, String name) {
        this.USN = USN;
        this.name = name;
    }

    public String getUSN() {
        return USN;
    }

    public String getName() {
        return name;
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList<Student> students = new LinkedList<>();

        students.add(new Student("1", "Alice"));
        students.add(new Student("2", "Bob"));
        students.add(new Student("3", "Charlie"));

        System.out.println("Student Details:");
        System.out.println("----------------");
        Iterator<Student> iterator = students.iterator();
        while (iterator.hasNext()) {
            Student student = iterator.next();
            System.out.println("USN: " + student.getUSN() + ", Name: " + student.getName());
        }
    }
}
```

3. Explain the following interfaces in detail
i) Queue ii) SortedSet

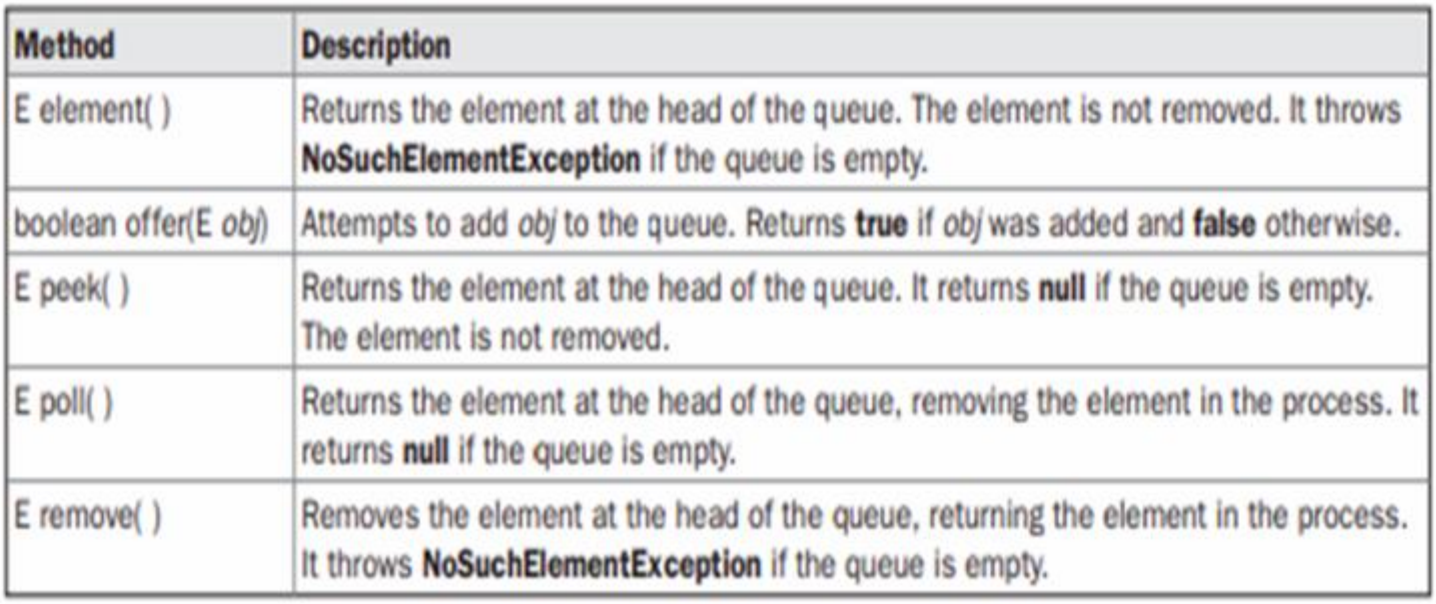
The SortedSet interface extends Set and declares the behavior of a set sorted in ascending order. SortedSet is a generic interface that has this declaration:
interface SortedSet<E>
Here, E specifies the type of objects that the set will hold.

SortedSet defines several methods that make set processing more convenient. To obtain the first object in the set, call first( ). To get the last element, use last( ). You can obtain a subset of a sorted set by calling subSet( ), specifying the first and last object in the set. If you need the subset that starts with the first element in the set, use headSet( ). If you want the subset that ends the set, use tailSet( ).



Method	Description
Comparator<? super E> comparator( )	Returns the invoking sorted set's comparator. If the natural ordering is used for this set, **null** is returned.
E first( )	Returns the first element in the invoking sorted set.
SortedSet<E> headSet(E *end*)	Returns a **SortedSet** containing those elements less than *end* that are contained in the invoking sorted set. Elements in the returned sorted set are also referenced by the invoking sorted set.
E last( )	Returns the last element in the invoking sorted set.
SortedSet<E> subSet(E *start*, E *end*)	Returns a **SortedSet** that includes those elements between *start* and *end*–1. Elements in the returned collection are also referenced by the invoking object.
SortedSet<E> tailSet(E *start*)	Returns a **SortedSet** that contains those elements greater than or equal to *start* that are contained in the sorted set. Elements in the returned set are also referenced by the invoking object.

Queue:
The Queue interface extends Collection and declares the behavior of a queue, which is often a first-in, first-out list. However, there are types of queues in which the ordering is based upon other criteria. Queue is a generic interface that has this declaration: interface Queue



Method	Description
E element( )	Returns the element at the head of the queue. The element is not removed. It throws **NoSuchElementException** if the queue is empty.
boolean offer(E *obj*)	Attempts to add *obj* to the queue. Returns **true** if *obj* was added and **false** otherwise.
E peek( )	Returns the element at the head of the queue. It returns **null** if the queue is empty. The element is not removed.
E poll( )	Returns the element at the head of the queue, removing the element in the process. It returns **null** if the queue is empty.
E remove( )	Removes the element at the head of the queue, returning the element in the process. It throws **NoSuchElementException** if the queue is empty.

Several methods throw a ClassCastException when an object is incompatible with the elements in the queue. A NullPointerException is thrown if an attempt is made to store a null object and null elements are not allowed in the queue. An IllegalArgumentException is thrown if an invalid argument is used. An IllegalStateException is thrown if an attempt is made to add an element to a fixed-length queue that is full. A NoSuchElementException is thrown if an attempt is made to remove an element from an empty queue

4. Explain the constructors of TreeSet class and write a java program to create Tree Set collection and access via iterator.

TreeSet extends AbstractSet and implements the NavigableSet interface. It creates a collection that uses a tree for storage. Objects are stored in sorted, ascending order. Access and retrieval times are quite fast, which makes TreeSet an excellent choice when storing large amounts of sorted information that must be found quickly.
TreeSet is a generic class that has this declaration:
class TreeSet<E>

Here, E specifies the type of objects that the set will hold.
TreeSet has the following constructors:
TreeSet( )
TreeSet(Collection<? extends E> c)
TreeSet(Comparator<? super E> comp)
TreeSet(SortedSet<E> ss)
The first form constructs an empty tree set that will be sorted in ascending order according to the natural order of its elements. The second form builds a tree set that contains the elements of c. The third form constructs an empty tree set that will be sorted according to the comparator specified by comp. (Comparators are described later in this chapter.) The fourth form builds a tree set that contains the elements of ss

```
import java.util.TreeSet;
import java.util.Iterator;

public class Main {
    public static void main(String[] args) {
        TreeSet<String> treeSet = new TreeSet<>();

        // Adding elements to the TreeSet
        treeSet.add("Apple");
        treeSet.add("Banana");
        treeSet.add("Orange");
        treeSet.add("Grapes");

        // Accessing elements using iterator
        System.out.println("Elements in TreeSet:");
        System.out.println("--------------------");
        Iterator<String> iterator = treeSet.iterator();
        while (iterator.hasNext()) {
            String element = iterator.next();
            System.out.println(element);
        }
    }
}
```

5. Explain the following legacy classes with an example. i)Hash Table ii)Vector

Hashtable stores key/value pairs in a hash table. However, neither keys nor values can be null. When using a Hashtable, you specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.
Hashtable was made generic by JDK 5. It is declared like this:
class Hashtable<K, V>
Here, K specifies the type of keys, and V specifies the type of values.
A hash table can only store objects that override the hashCode( ) and equals( ) methods that are defined by Object. The hashCode( ) method must compute and return the hash code for the object. Of course, equals( ) compares two objects. Fortunately, many of Java's built-in classes already implement the hashCode( ) method. For example, the most common type of Hashtable uses a String object as the key. String implements both hashCode( ) and equals( ).
The Hashtable constructors are shown here:
Hashtable( )
Hashtable(int size)
Hashtable(int size, float fillRatio)
Hashtable(Map<? extends K, ? extends V> m)

Method	**Description**
void clear( )	Resets and empties the hash table.
Object clone( )	Returns a duplicate of the invoking object.
boolean contains(Object *value*)	Returns **true** if some value equal to *value* exists within the hash table. Returns **false** if the value isn't found.
boolean containsKey(Object *key*)	Returns **true** if some key equal to *key* exists within the hash table. Returns **false** if the key isn't found.
boolean containsValue(Object *value*)	Returns **true** if some value equal to *value* exists within the hash table. Returns **false** if the value isn't found.
Enumeration<V> elements( )	Returns an enumeration of the values contained in the hash table.
V get(Object *key*)	Returns the object that contains the value associated with *key*. If *key* is not in the hash table, a **null** object is returned.
boolean isEmpty( )	Returns **true** if the hash table is empty; returns **false** if it contains at least one key.
Enumeration<K> keys( )	Returns an enumeration of the keys contained in the hash table.
V put(K *key*, V *value*)	Inserts a key and a value into the hash table. Returns **null** if *key* isn't already in the hash table; returns the previous value associated with *key* if *key* is already in the hash table.
void rehash( )	Increases the size of the hash table and rehashes all of its keys.
V remove(Object *key*)	Removes *key* and its value. Returns the value associated with *key*. If *key* is not in the hash table, a **null** object is returned.
int size( )	Returns the number of entries in the hash table.
String toString( )	Returns the string equivalent of a hash table.

TABLE 17-18 The Legacy Methods Defined by **Hashtable**

```
import java.util.*;
class HTDemo2 {
public static void main(String args[]) {
Hashtable<String, Double> balance = new Hashtable<String, Double>();
String str;
double bal;
balance.put("John Doe", 3434.34);
balance.put("Tom Smith", 123.22);
balance.put("Jane Baker", 1378.00);
balance.put("Tod Hall", 99.22);
balance.put("Ralph Smith", -19.08);
// Show all balances in hashtable.
// First, get a set view of the keys.
Set<String> set = balance.keySet();
// Get an iterator.
Iterator<String> itr = set.iterator();
while(itr.hasNext()) {
str = itr.next();
System.out.println(str + ": " +
balance.get(str));
}
System.out.println();
// Deposit 1,000 into John Doe's account.
bal = balance.get("John Doe");
balance.put("John Doe", bal+1000);
System.out.println("John Doe's new balance: " +
balance.get("John Doe"));
}
}
```

ii)Vector

Vector implements a dynamic array. It is similar to ArrayList, but with two differences: Vector is synchronized, and it contains many legacy methods that are not part of the Collections Framework. With the advent of collections, Vector was reengineered to extend AbstractList and to implement the List interface. With the release of JDK 5, it was retrofitted for generics and reengineered to implement Iterable. This means that Vector is fully compatible with collections,
and a Vector can have its contents iterated by the enhanced for loop.

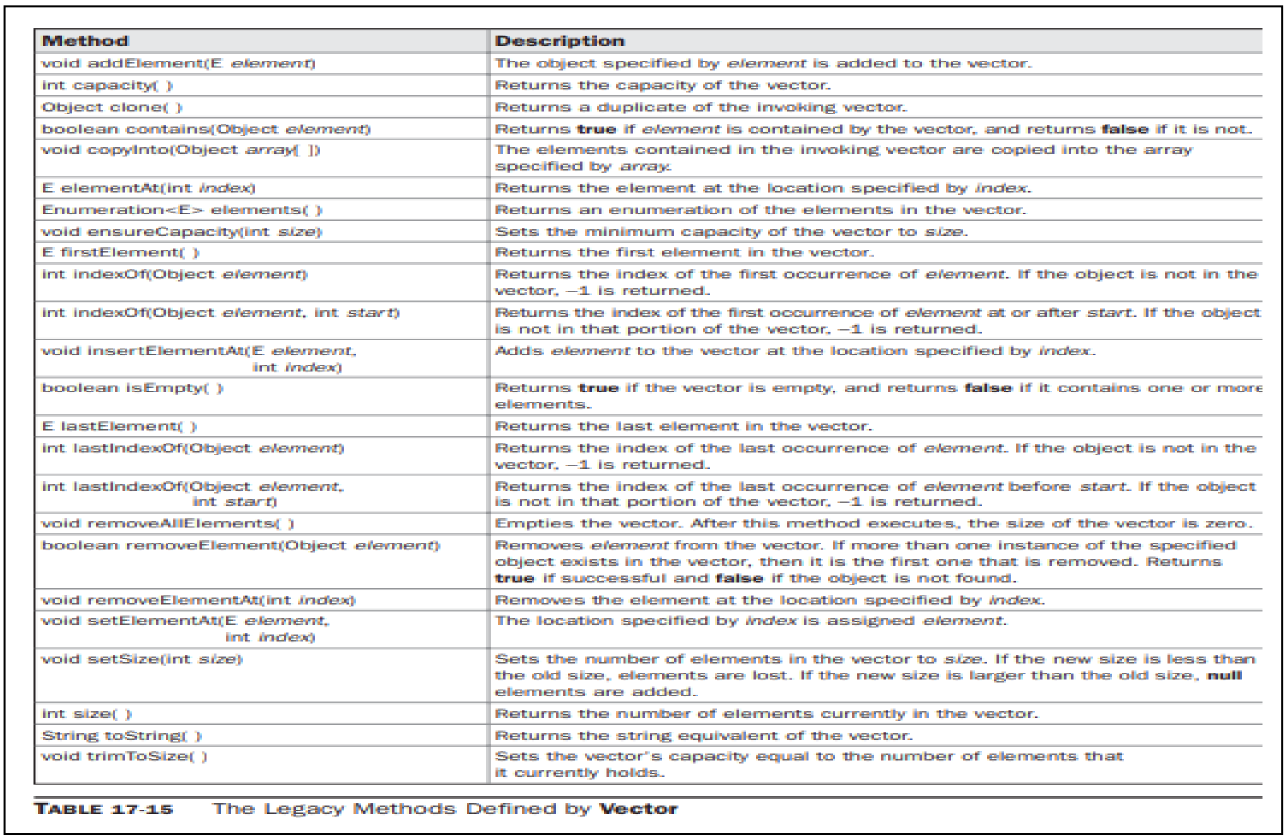
Vector is declared like this:

```
class Vector<E>
```

Here, E specifies the type of element that will be stored.

Here are the Vector constructors:

```
Vector( )
Vector(int size)
Vector(int size, int incr)
Vector(Collection<? extends E> c)
```



Method	Description
void addElement(E *element*)	The object specified by *element* is added to the vector.
int capacity( )	Returns the capacity of the vector.
Object clone( )	Returns a duplicate of the invoking vector.
boolean contains(Object *element*)	Returns **true** if *element* is contained by the vector, and returns **false** if it is not.
void copyInto(Object *array*[ ])	The elements contained in the invoking vector are copied into the array specified by *array*.
E elementAt(int *index*)	Returns the element at the location specified by *index*.
Enumeration<E> elements( )	Returns an enumeration of the elements in the vector.
void ensureCapacity(int *size*)	Sets the minimum capacity of the vector to *size*.
E firstElement( )	Returns the first element in the vector.
int indexOf(Object *element*)	Returns the index of the first occurrence of *element*. If the object is not in the vector, –1 is returned.
int indexOf(Object *element*, int *start*)	Returns the index of the first occurrence of *element* at or after *start*. If the object is not in that portion of the vector, –1 is returned.
void insertElementAt(E *element*, int *index*)	Adds *element* to the vector at the location specified by *index*.
boolean isEmpty( )	Returns **true** if the vector is empty, and returns **false** if it contains one or more elements.
E lastElement( )	Returns the last element in the vector.
int lastIndexOf(Object *element*)	Returns the index of the last occurrence of *element*. If the object is not in the vector, –1 is returned.
int lastIndexOf(Object *element*, int *start*)	Returns the index of the last occurrence of *element* before *start*. If the object is not in that portion of the vector, –1 is returned.
void removeAllElements( )	Empties the vector. After this method executes, the size of the vector is zero.
boolean removeElement(Object *element*)	Removes *element* from the vector. If more than one instance of the specified object exists in the vector, then it is the first one that is removed. Returns **true** if successful and **false** if the object is not found.
void removeElementAt(int *index*)	Removes the element at the location specified by *index*.
void setElementAt(E *element*, int *index*)	The location specified by *index* is assigned *element*.
void setSize(int *size*)	Sets the number of elements in the vector to *size*. If the new size is less than the old size, elements are lost. If the new size is larger than the old size, **null** elements are added.
int size( )	Returns the number of elements currently in the vector.
String toString( )	Returns the string equivalent of the vector.
void trimToSize( )	Sets the vector's capacity equal to the number of elements that it currently holds.

TABLE 17-15 The Legacy Methods Defined by **Vector**

```
// Demonstrate various Vector operations.
import java.util.*;
class VectorDemo {
public static void main(String args[]) {
// initial size is 3, increment is 2
Vector<Integer> v = new Vector<Integer>(3, 2);
System.out.println("Initial size: " + v.size());
System.out.println("Initial capacity: " +
v.capacity());
v.addElement(1);
v.addElement(2);
v.addElement(3);
v.addElement(4);
System.out.println("Capacity after four additions: " +
v.capacity());
v.addElement(5);
System.out.println("Current capacity: " +
v.capacity());
v.addElement(6);
v.addElement(7);
System.out.println("Current capacity: " +
v.capacity());
v.addElement(9);
v.addElement(10);
System.out.println("Current capacity: " +
v.capacity());
v.addElement(11);
v.addElement(12);
System.out.println("First element: " + v.firstElement());
System.out.println("Last element: " + v.lastElement());
if(v.contains(3))
System.out.println("Vector contains 3.");
// Enumerate the elements in the vector.
Enumeration vEnum = v.elements();
System.out.println("\nElements in vector:");
while(vEnum.hasMoreElements())
System.out.print(vEnum.nextElement() + " ");
System.out.println();
}
}
```

6. List and Explain JDBCDriver types.

Type 1 JDBC to ODBC Driver

The JDBC type 1 driver which is also known as a JDBC-ODBC Bridge is a convert JDBC methods into ODBC function calls. Sun provides a JDBC-ODBC Bridge driver by "**sun.jdbc.odbc.JdbcOdbcDriver**".
The driver is a platform dependent because it uses ODBC which is depends on native libraries of the operating system and also the driver needs other installation for example, ODBC must be installed on the computer and the database must support ODBC. Type 1 is the simplest compare to all other driver but it's a platform specific i.e. only on Microsoft platform. The JDBC-ODBC Bridge is use only when there is no PURE-JAVA driver available for a particular database.

Advantages:

(1) Connect to almost any database on any system, for which ODBC driver is installed.
(2) It's an easy for installation as well as easy(simplest) to use as compare the all other driver.

Disadvantages:

(1) The ODBC Driver needs to be installed on the client machine.
(2) It's a not a purely platform independent because its use ODBC which is depends on native libraries of the operating system on client machine.
(3) Not suitable for applets because the ODBC driver needs to be installed on the client machine.

Type 2 Driver: Native-API Driver (Partly Java driver) :-

The JDBC type 2 driver is uses the libraries of the database which is available at client side and this driver converts the JDBC method calls into native calls of the database so this driver is also known as a Native-API driver.

Advantage:

There is no implantation of JDBC-ODBC Bridge so it's faster than a type 1 driver; hence the performance is better as compare the type 1 driver (JDBC-ODBC Bridge).

Disadvantages:

(1) On the client machine require the extra installation because this driver uses the vendor client libraries.
(2) The Client side software needed so cannot use such type of driver in the web-based application.
(3) Not all databases have the client side library.
(4) This driver supports all **JAVA** applications except applets

Type 3 Driver: Network-Protocol Driver (Pure Java driver for database Middleware) :-

The JDBC type 3 driver uses the middle tier(application server) between the calling program and the database and this middle tier converts JDBC method calls into the vendor specific database protocol and the same driver can be used for multiple databases also so it's also known as a Network- Protocol driver as well as a **JAVA** driver for database middleware.

Advantages:

(1) There is no need for the vendor database library on the client machine because the middleware is database independent and it communicates with client.
(2) Type 3 driver can be used in any web application as well as on internet also
(3) A single driver can handle any database at client side so there is no need a separate driver for each database.
(4) The middleware server can also provide the typical services such as connections, auditing, load balancing, logging etc.

Disadvantages:

(1) An Extra layer added, may be time consuming.
(2) At the middleware develop the database specific coding, may be increase complexity.

Type 4 Driver: Native-Protocol Driver (Pure Java driver directly connected to database):

The JDBC type 4 driver converts JDBC method calls directly into the vendor specific database protocol and in between do not need to be converted any other formatted system so this is the fastest way to communicate quires to DBMS and it is completely written in JAVA because of that this is also known as the “direct to database Pure JAVA driver”.

Advantage:

(1) It’s a 100% pure JAVA Driver so it’s a platform independence.
(2) No translation or middleware layers are used so consider as a faster than other drivers.

Disadvantages:

(1) There is a separate driver needed for each database at the client side.
(2) Drivers are Database dependent, as different database vendors use different network protocols.

7. Describe the various steps of JDBC process with code snippets.

Step 0: import the java.sql package

An application that uses the jdbc API must import the java.sql package

```
import java.sql.*;
```

Step 1: Load a JDBC Driver

Prior to JDBC 4.0 it is needed to seperately load the driver and register the driver but in jdbc 4.0 it is no longer needed to register the driver

```
Class.forName(“sun.jdbc.odbc:jdbcodbcDriver”);
```

Step 2: Establishing a connection

Once a driver is loaded we can establish a connection to db

```
Connection con= DriverManager.getconnection(dburl,username,password)
```

DriverManager Connects to given JDBC URL with given user name and password
A Connection represents a session with a specific database.
The connection to the database is established by getConnection(), which requests access to the database from the DBMS.
A Connection object is returned by the getConnection() if access is granted; else getConnection() throws a SQLException.
Sometimes a DBMS requires extra information besides userID & password to grant access to the database.
This additional information is referred as properties and must be associated with Properties or Sometimes DBMS grants access to a database to anyone without using username or password.
Ex: Connection c = DriverManager.getConnection(url) ;

Step 3: Create a statement

A statement object is needed to execute the query and obtain the results produced by it.

```
Statement st= con.createStatement();
```

Step 4: Execute the statement

The db statements can be executed by using methods like executeQuery().
executeQuery() takes querystring as an argument and returns the results as ResultSet object
ResultSet object contains the data returned by the query and the methods for retrieving the data
Ex: ResultSet rs=stmt.executeQuery(“select * from employee”);

Step 5: Process the result

The ResultSet consists of tuples and returns one tuple at a time when the next() is applied.
ResultSet acts as an iterator

```
While(rs.next())
{
System.out.println(rs.getString(1)+” “+rs.getInt(“salary”);
}
```

Getters can be used by referring position/name to retrieve the values

Step 6: Close the statement

```
stmt.close();
```

Step 7: Close the connection

```
Commit()
con.close()
```

8. Discuss in detail the need of prepared Statement with an example program.

The preparedStatement object allows you to execute parameterized queries.A SQL query can be precompiled and executed by using the PreparedStatement object. · Ex: Select * from publishers where pub_id=?

Here a query is created as usual, but a question mark is used as a placeholder for a value· thatis inserted into the query after the query is compiled.

The preparedStatement() method of Connection object is called to return the preparedStatement object.

Ex:

```
import java.sql.*;
public class JdbcDemo {
publicstaticvoidmain(Stringargs[]){ try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");
PreparedStatement pstmt;
pstmt=con.prepareStatement("select*fromemployeewhereUserName=?"); pstmt.setString(1,"khutub");
ResultSetrs1=pstmt.executeQuery(); while(rs1.next()){
System.out.println(rs1.getString(2));
}
} // end of try
catch(Exception e){System.out.println("exception"); }
} //end of main
} // end of class
```

9. Discuss the following map classes with example.
a)Hash Map b)TreeMap

The HashMap class extends AbstractMap and implements the Map interface. It uses a hash table to store the map. This allows the execution time of get( ) and put( ) to remain constant even for large sets.
HashMap is a generic class that has this declaration:
class HashMap<K, V>
Here, K specifies the type of keys, and V specifies the type of values.
The following constructors are defined:
HashMap( )
HashMap(Map<? extends K, ? extends V> m)
HashMap(int capacity)
HashMap(int capacity, float fillRatio)

The first form constructs a default hash map. The second form initializes the hash map by using the elements of m. The third form initializes the capacity of the hash map to capacity. The fourth form initializes both the capacity and fill ratio of the hash map by using its arguments.
The meaning of capacity and fill ratio is the same as for HashSet, described earlier. The default capacity is 16. The default fill ratio is 0.75.
HashMap implements Map and extends AbstractMap. It does not add any methods of its own.
The TreeMap Class
The TreeMap class extends AbstractMap and implements the NavigableMap interface. It creates maps stored in a tree structure. A TreeMap provides an efficient means of storing key/value pairs in sorted order and allows rapid retrieval. You should note that, unlike a hash map, a tree map

guarantees that its elements will be sorted in ascending key order.
TreeMap is a generic class that has this declaration:
class TreeMap<K, V>
Here, K specifies the type of keys, and V specifies the type of values.
The following TreeMap constructors are defined:
TreeMap( )
TreeMap(Comparator<? super K> comp)
TreeMap(Map<? extends K, ? extends V> m)
TreeMap(SortedMap<K, ? extends V> sm)
The first form constructs an empty tree map that will be sorted by using the natural order of its keys. The second form constructs an empty tree-based map that will be sorted by using the Comparator comp. (Comparators are discussed later in this chapter.) The third form initializes a tree map with the entries from m, which will be sorted by using the natural order of the keys. The fourth form initializes a tree map with the entries from sm, which will be sorted in the same order as sm.

```
import java.util.*;
class TreeMapDemo {
public static void main(String args[]) {
// Create a tree map.
TreeMap<String, Double> tm = new TreeMap<String, Double>();
// Put elements to the map.
tm.put("John Doe", new Double(3434.34));
tm.put("Tom Smith", new Double(123.22));
tm.put("Jane Baker", new Double(1378.00));
tm.put("Tod Hall", new Double(99.22));
tm.put("Ralph Smith", new Double(-19.08));
// Get a set of the entries.
Set<Map.Entry<String, Double>> set = tm.entrySet();

// Display the elements.
for(Map.Entry<String, Double> me : set) {
System.out.print(me.getKey() + ": ");
System.out.println(me.getValue());
}
System.out.println();
// Deposit 1000 into John Doe's account.
double balance = tm.get("John Doe");
tm.put("John Doe", balance + 1000);
System.out.println("John Doe's new balance: " +
tm.get("John Doe"));
}
}
```

10. Develop a program to insert following data into music database using prepared Statement object. Table consists of music_id int(5),music_name varchar(20),music_author varchar(20)

```
package jdbcdemo;

import java.io.*;
import java.sql.*;
public class JDBCDemo {
public static void main(String[] args) {
        // TODO code application logic here
                Connection con;
                        PreparedStatement pstmt;
                        Statement stmt;
                        ResultSet rs;
                        String mid, mname,mauthor;
                        Integer marks,count;
                                try{
```

```
                        con=DriverManager.getConnection("jdbc:mysql://127.0.0.1/mca","root","cmrit"); //  type1 access
connection
                                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                          do
                          {
                                System.out.println("\n1. Insert.\n2. Select.\n3. Update.\n4. Delete.\n5. Exit.\nEnter your
choice:");
                                int choice=Integer.parseInt(br.readLine());
                                switch(choice)
                                {
                                        case 1: System.out.print("Enter music id :");
                                                mid=br.readLine();
                                                System.out.print("Enter music name :");
                                                mname=br.readLine();
                                                System.out.print("Enter music author :");
                                                mauthor=br.readLine();
                                                pstmt=con.prepareStatement("insert into music values(?,?,?)");
                                                pstmt.setString(1,mid);
                                                pstmt.setString(2,mname);
                                                pstmt.setString(2,mauthor);
                                                pstmt.execute();
                                                System.out.println("\nRecord Inserted successfully.");
                                        break;
                                        case 2:
                                                stmt=con.createStatement();
                                                rs=stmt.executeQuery("select *from music");
                                                if(rs.next())
                                                {
                                                System.out.println("Mid\t Mname\t Mauthor\n--------------------------------");
                                                do
                                                {
                                                        mid=rs.getString(1);
                                                        mname=rs.getString(2);
                                                        mauthor=rs.getString(3);
                                                        System.out.println(mid+"\t"+mname+"\t"+mauthor);
                                                }while(rs.next());
                                                }
                                                else
                                                        System.out.println("Record(s) are not available in database.");
                                                break;
                                                case 3:
                                                        System.out.println("Enter mid to update :");
                                                        mid=br.readLine();
                                                        System.out.println("Enter new mname:");
                                                        mname=br.readLine();
                                                        stmt=con.createStatement();
                                                        count=stmt.executeUpdate("update student set
mname='"+mname+"'where mid='"+mid+"'");
                                                        System.out.println("\n"+count+" Record Updated.");
                                                break;
                                                case 4: System.out.println("Enter mid to delete record:");
                                                        mid=br.readLine();
                                                        stmt=con.createStatement();
                                                        count=stmt.executeUpdate("delete from music where mid='"+mid+"'");


                                                        if(count!=0)
                                                                System.out.println("\nRecord "+mid+" has deleted.");
                                                        else
                                                                System.out.println("\nInvalid USN, Try again.");
                                                break;

                                                case 5: con.close(); System.exit(0);
                                                default: System.out.println("Invalid choice, Try again.");
                                        }//close of switch
                                        }while(true);
                                        }//close of nested try
                                        catch(SQLException e2)
                                        {
                                                System.out.println(e2);
                                        }
                                        catch(IOException e3)
                                        {
                                                System.out.println(e3);
                                        }
                        }//close of outer try

                }
```