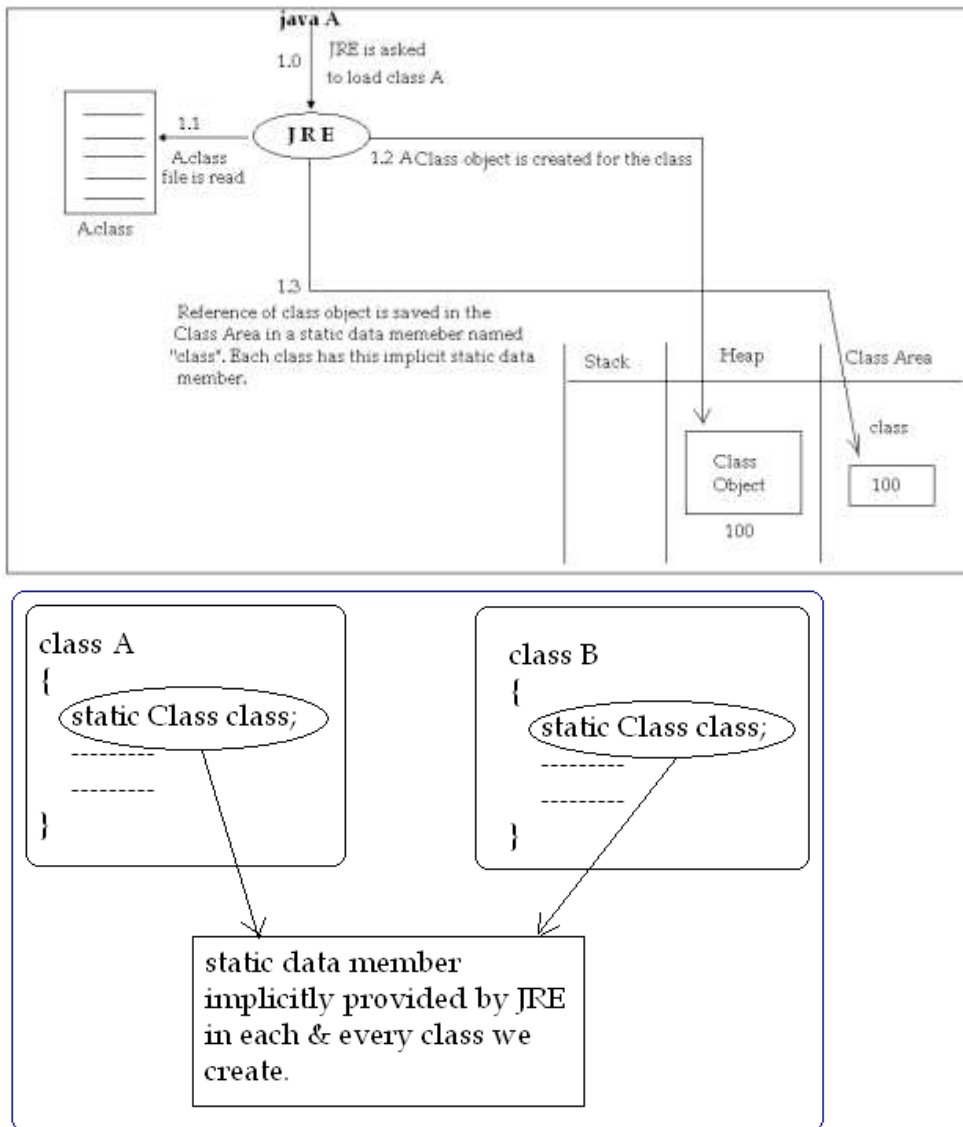


## REFLECTION

- Reflection is the java API that provides the facility of obtaining information of a class at runtime.
- Reflection is facilitated with the help of **java.lang.Class**.
- For each class that is loaded by the JRE an object of type **Class** is created that holds the information of the loaded class.

Fig1



Reference of class object can be obtained in two ways:-

- Using static data member class

Syntax:

**ClassName.class;**

- Using **getClass()** method of object class.

Syntax:

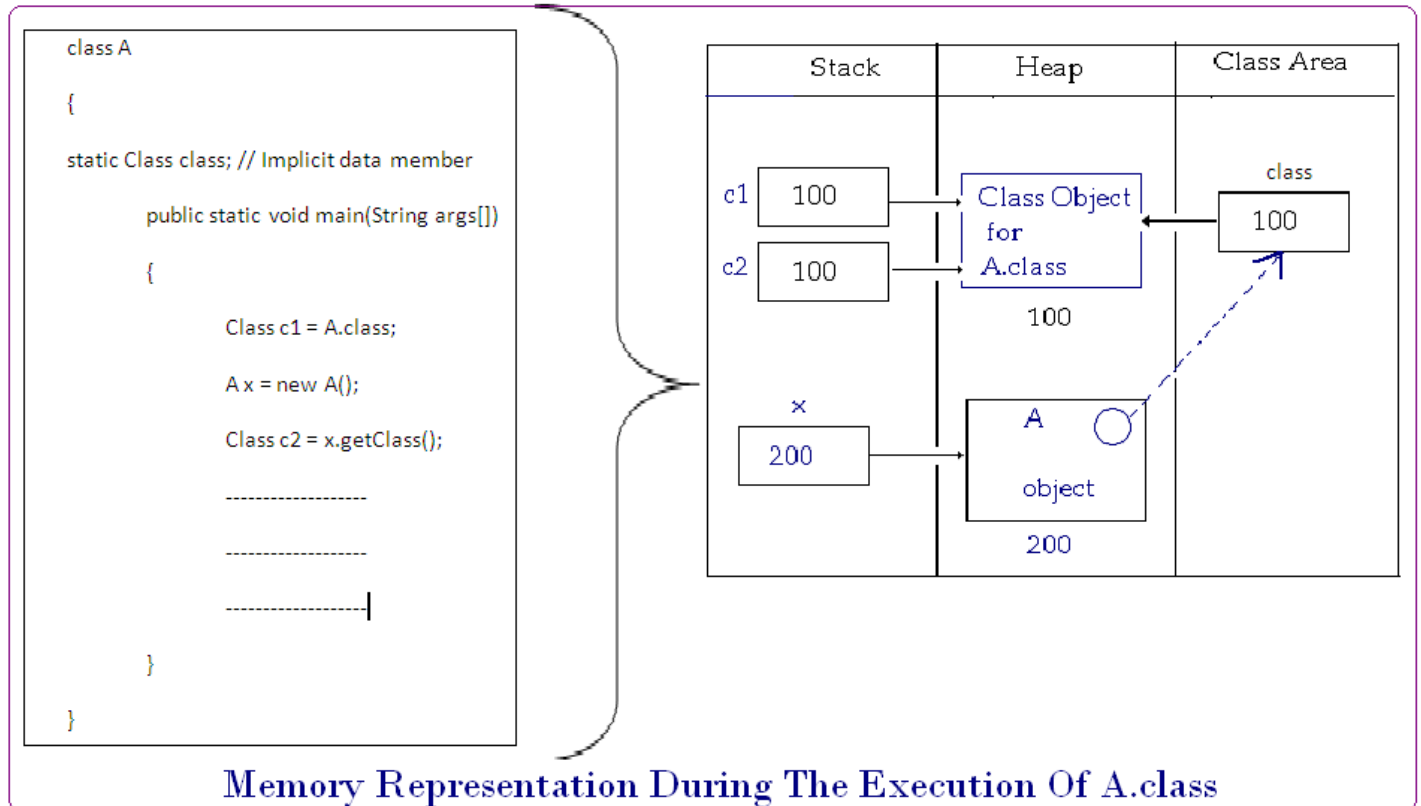
**objName.getClass( );**

```
Class A
{
```

```

public static void main(String arr[ ])
{
    Class c1 = A.class;
    A x = new A();
    Class c2 = x.getClass();
}
    
```

Fig - 2



Commonly used methods of a class:-

1. **forName()** method is used to require dynamic loading of a class.

*Syntax -*

**public static class forName (String className) throws ClassNotFoundException;**

2. **newInstance()** method is used to create an object of a class through its class objects.

*Syntax -*

**public Object newInstance() throws Exception;**

*Example -*

```

class A
{
    public static void main(String args[ ])
    {
        Class c = Class.forName(arr[0]);
        Object o = c.newInstance();
    }
}
    
```

```
        }  
    }  
}
```

3. **getName()** method is used to find out the name of the class.

*Syntax -*

**public String getName();**

4. **isAbstract()** method returns true if class object represents an abstract class.

*Syntax -*

**public boolean isAbstract();**

5. **isInterface()** method returns true if class object represents an Interface.

*Syntax -*

**public Boolean isInterface();**

6. **getFields()** method return an array of **java.lang.reflect.Field** object. Each element of the array describes a data member of the class (declared as well as inherited).

*Syntax -*

**public Fields[] getFields();**

7. **getDeclaredFields()** method returns an array of fields object, each element of the array describes a data member that is defined in the class.

*Syntax -*

**public Field[] getDeclaredFields();**

8. **getMethods()** method returns an array **java.lang.reflect.Method** objects. Each element of the array describes a method of the class (declared as well as inherited).

*Syntax -*

**public Constructor[] getConstructor();**

9. **getDelaredMethods()** method returns an array of Method objects. Each element of the array describes a method that is defined in the class.

*Syntax -*

**public Method[] getDeclaredMethod();**

10. **getConstructor()** method returns an array of **java.lang.reflect.Constructor** objects. Each element of the array describes the constructor of class.

*Syntax -*

**public Method[] getDeclaredMethod();**

// A class that tells the name of the class whose object is provided to it.

## J2EE Notes (By Neeraj Sir)

Class NameFinder

```
{
    public static void findName(Object o)
    {
        Class c = o.getClass();
        System.out.put.line("This is of class:" + c.getName());
    }
}
-----X-----
```

### Example-

class A

```
{}
```

class B

```
{}
```

class C

```
{
    public static void main(String [] arr)
    {
        NameFinder.findName(new A());
        NameFinder.findName(new B());
        NameFinder.findName(new C());
    }
}
```

-----X-----

**//A class that displays data members, constructor & methods of a class whose name is provided as command line argument.**

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
```

class Descriptor

```
{
    public static void main(String args[])
    {
        try
        {
            Class c = Class.forName(args[0]);
            Field f[] = c.getDeclaredFields();
            for(int i=0; i<=f.length; i++)
                System.out.println(f[i]);

            Constructor ctr[] = c.getDeclaredConstructors();
            for (int i = 0; i<ctr.length; i++)
                System.out.println(ctr[i]);

            Method m[] = c.getDeclaredMethods();
            for(int i=0; i<m.length; i++)
```

```
                System.out.println(m[i]);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
```

-----X-----

**// A class that creates the object of another class whose name is provided as command line argument.**

```
package Reflection;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

public class ObjCreator
{
    public static void main(String [] args)
    {
        try
        {
            Class c = Class.forName(args[0]);
            Object o = ctr.newInstance();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

class A2
{
    static
    {
        System.out.println("A is loaded...");
    }

    public A2()
    {
        System.out.println("A is initiated...");
    }

    public void print()
    {
        System.out.println("Display() of A is invoked...");
    }
}

class B2
{
}
```

## J2EE Notes (By Neeraj Sir)

```
static
{
    System.out.println("B2 is loaded...");
}

public B2()
{
    System.out.println("B2 is initiated...");
}

public void display()
{
    System.out.println("print() of B2 is invoked...");
}
}
```

-----X-----

package Reflection;

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
```

```
public class ObjCreator
{
    public static void main(String [] args)
    {
        try
        {
            Class c = Class.forName(args[0]);
            Constructor ctr = c.getDeclaredConstructor(null);

            ctr.setAccessible(true);
            Object o = ctr.newInstance();

            Method m[] = c.getDeclaredMethods();
            for(int i=0; i<m.length; i++)
            {
                m[i].setAccessible(true);
                m[i].invoke(o, null);
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

class A2
{
    static
    {
        System.out.println("A is loaded...");
    }
}
```

## J2EE Notes (By Neeraj Sir)

```
}

public A2()
{
    System.out.println("A is initiated...");
}

public void print()
{
    System.out.println("Display() of A is invoked...");
}
}

class B2
{
    static
    {
        System.out.println("B2 is loaded...");
    }

    public B2()
    {
        System.out.println("B2 is initiated...");
    }

    public void display()
    {
        System.out.println("Print() of B2 is invoked...");
    }
}
```

-----X-----

package Reflection;

import java.applet.Applet;

import java.awt.Graphics;

import javax.swing.JFrame;

class MyViewer

```
{
    public static void main(String[] args)
    {
        try
        {
            if (args.length==0)
                System.out.println("Specify AppletName, width & height as command line arguments..");
            else
            {
                Class c = Class.forName(args[0]);
                Applet a = (Applet) c.newInstance();
            }
        }
    }
}
```

```
        int w = Integer.parseInt(args[1]);
        int h = Integer.parseInt(args[2]);
        JFrame frm = new JFrame("My viewer");
        frm.setSize(w, h);
        frm.add(a);
        frm.setVisible(true);
        frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        a.init();
        a.start();
        Graphics g = a.getGraphics();
        a.paint(g);
    }
}
catch(Exception e)
{
    System.out.println(e);
}
}
```

-----X-----



## COLLECTION FRAMEWORK

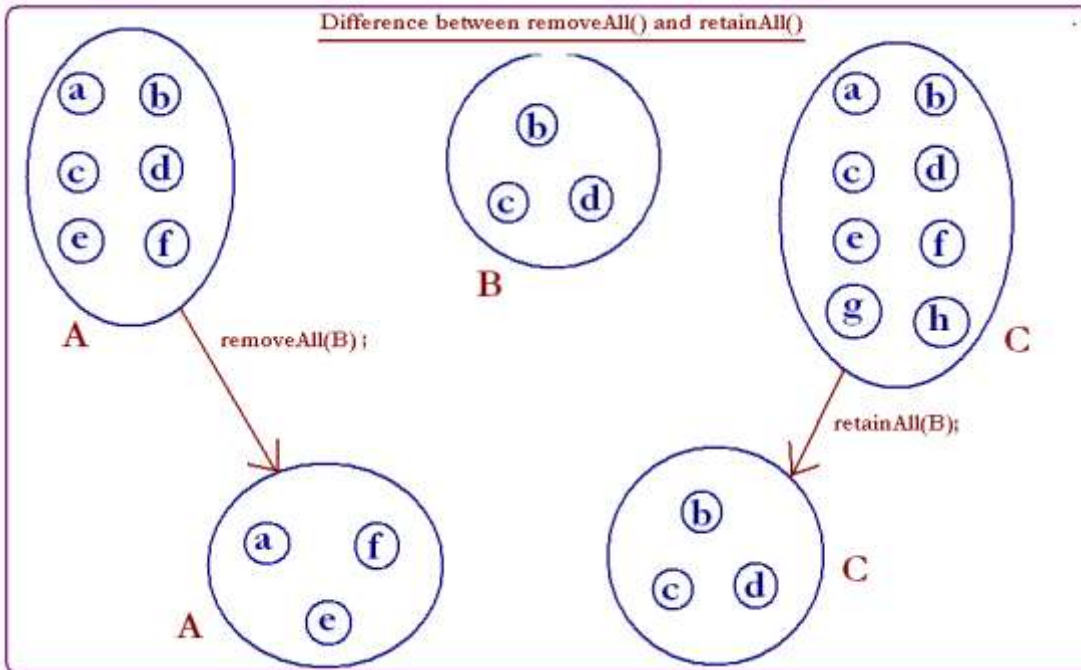
A framework is a collection of classes and interfaces that facilitate development of specific kind of applications by providing an architectural model and reusable components.

Collection Framework provides a unified model to work with different types of data structures such as array, linked list, binary search tree, hash table etc...

At the core of collection framework is an interface named **java.util.Collection**. This interface describes common behavior of all collections.

*Methods of collection interface*

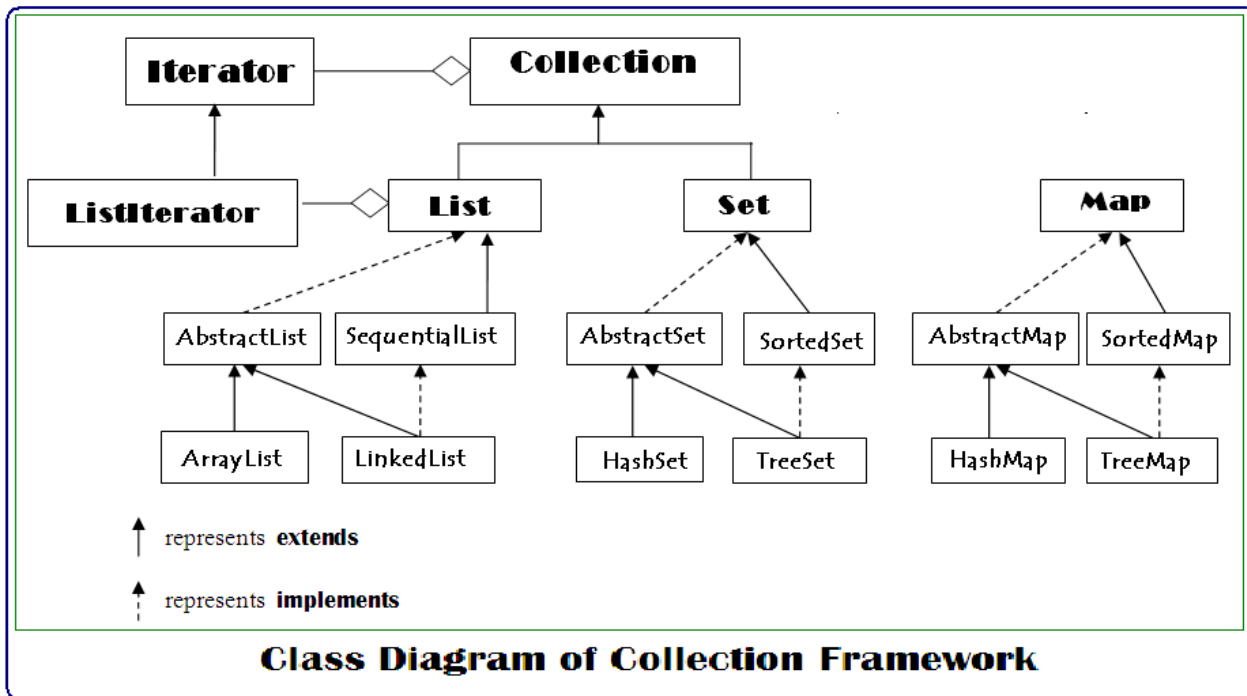
<b>add()</b>	is used to add an element to a collection.
<b>Syntax :</b>	<i>public boolean add(Object element);</i>
<b>addAll()</b>	is used to add all the element of one collection to another.
<b>Syntax :</b>	<i>public boolean addAll(Collection c);</i>
<b>remove():</b>	is used to remove an element from a collection.
<b>Syntax :</b>	<i>public boolean remove(Object o);</i>
<b>removeAll()</b>	is used to remove all the elements of a collection from another collection.
<b>Syntax:</b>	<i>public boolean removeAll(Collection c);</i>
<b>retainAll()</b>	is used to remove all the elements of the invoking collection which are not part of the given collection.
<b>Syntax :</b>	<i>public boolean retainAll(Collection c);</i>
<b>contains()</b>	is used to search an element in a collection.
<b>Syntax :</b>	<i>public boolean contains(Object element);</i>
<b>containsAll()</b>	is used to search all the element of a collection to another collection.
<b>Syntax :</b>	<i>public boolean containsAll(Collection c);</i>
<b>size()</b>	is used to find out number of elements of a collection.
<b>Syntax :</b>	<i>public int size();</i>
<b>clear()</b>	is used to remove all the elements of a collection.
<b>Syntax :</b>	<i>public void clear();</i>
<b>iterator()</b>	is used to obtain an iterator for traversing the elements of a collection.
<b>Syntax :</b>	<i>public Iterator iterator();</i>



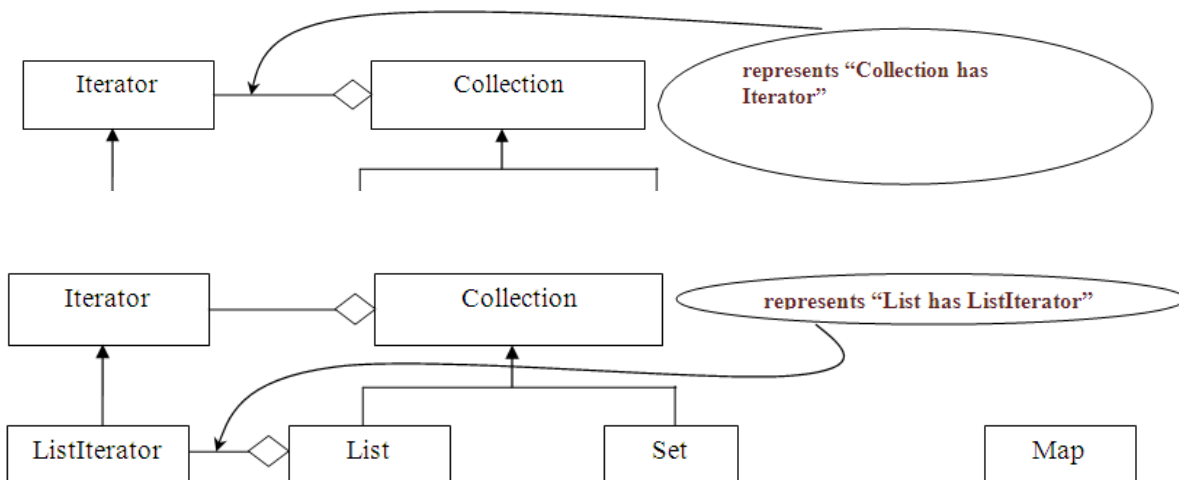
**Iterator** is an interface of collection framework, implementation of which is provided by all collections. This interface is used by application developers for traversing the elements of a collection in implementations independent fashion.

*Methods of Iterator interface*

<b>hasNext()</b>	returns true if there is an element to be traversed.
<b>Syntax :</b>	<code>public boolean hasNext();</code>
<b>next()</b>	is used to obtain the next element from the collection.
<b>Syntax :</b>	<code>public Object next();</code>
<b>remove()</b>	is used to remove last traversed element from the collection.
<b>Syntax:</b>	<code>public void remove();</code>



See the snapshots below for the better understanding of the table above :-



## List

A **List** represents an indexed collection. Functionality of list is represented by **List** interface which extends **Collection** interface and adds following methods:-

<b>add()</b>	is used to insert an element in a list.
<b>Syntax:</b>	<code>public boolean add(int index, Object element);</code>
<b>addAll()</b>	is used to insert all the elements of a collection at the specific position in the list.
<b>Syntax:</b>	<code>public boolean addAll(int index, Collection c);</code>
<b>get()</b>	is used to obtain the reference of an element stored at the given position in a list.
<b>Syntax:</b>	<code>public Object get(int index);</code>

<b>remove()</b>	is used to remove an element stored at the given position in the list.
<b>Syntax:</b>	<i>public boolean remove(int index);</i>
<b>indexOf()</b>	is used to find out a index of an element in a list.
<b>Syntax:</b>	<i>public int indexOf(Object element);</i>
<b>lastIndexOf():-</b>	is used to find out the index of last appearance of an element in a list.
<b>Syntax: →</b>	<i>public int lastIndexOf(Object element);</i>
<b>listIterator()</b>	is used to obtain a list iterator for traversing the element of a list.
<b>Syntax:</b>	<i>public ListIterator listIterator();</i> <i>public ListIterator listIterator(int index);</i>

**ListIterator** facilitate traversing the elements of a list in both directions.

**ListIterator** *extends* **Iterator** and add following methods:-

<b>hasPrevious()</b>	return true if there is an element behind current index.
<b>Syntax:</b>	<i>public boolean hasPrevious();</i>
<b>previous()</b>	returns the reference of previous element.
<b>Syntax:</b>	<i>public Object previous();</i>
<b>add()</b>	is used to insert an element at the current index in the list.
<b>Syntax:</b>	<i>public void add(Object element);</i>

**ArrayList** class provides implementation of **List** interface through **AbstractList** class. An object of **ArrayList** represents a dynamic array.

-----X-----

Program **ListDemo.java**

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

class EmpListDemo
{
    public static void main(String args[])
    {
        ArrayList list = new ArrayList();

        list.add("Java");
        list.add("Spring");
        list.add("Php");
        list.add(1, "J2EE");
        list.add("Hibernate");
    }
}
```

## J2EE Notes (By Neeraj Sir)

```
System.out.println("There are " + list.size() + " elements in the list... ");
System.out.println("Contents of the list are ....");

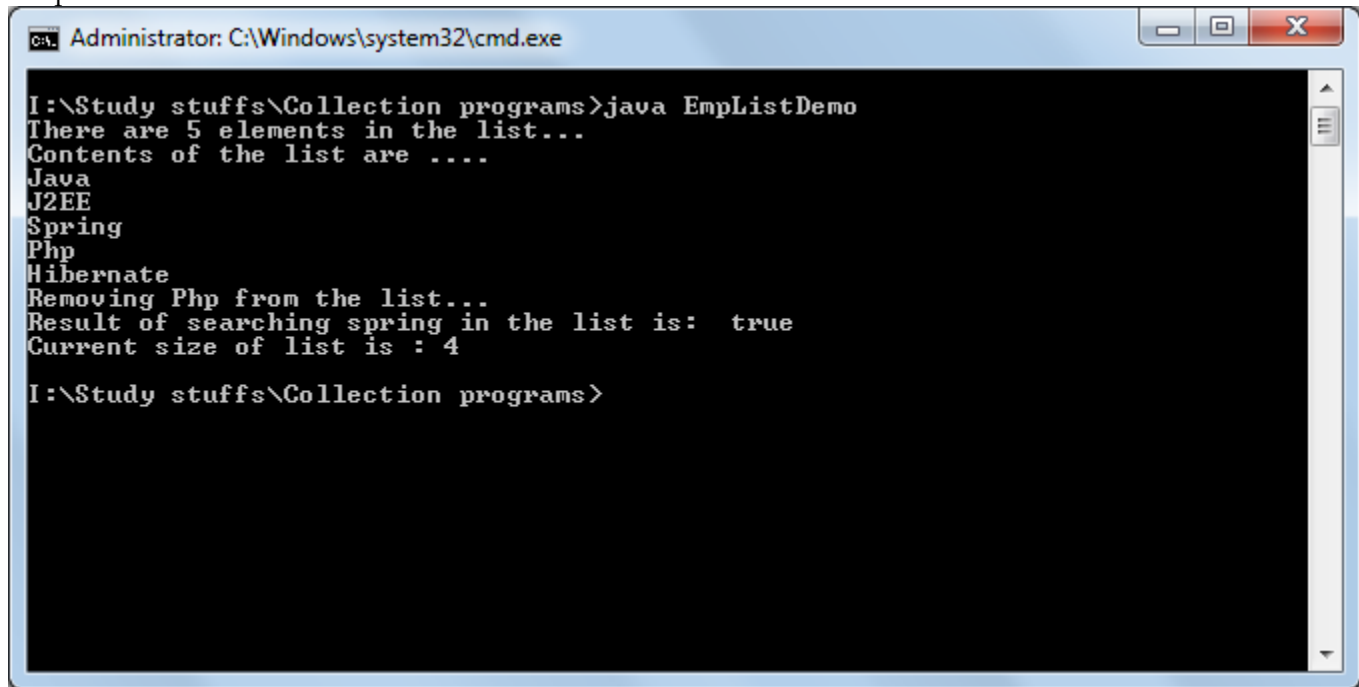
Iterator itr = list.iterator();
while(itr.hasNext())
    System.out.println(itr.next());

System.out.println("Removing Php from the list...");
list.remove("Php");

System.out.println("Result of searching spring in the list is: " + list.contains("Spring"));

System.out.println("Current size of list is : " + list.size());
}
}
```

Output-

A screenshot of a Windows command prompt window titled "Administrator: C:\Windows\system32\cmd.exe". The window shows the execution of a Java program named "EmpListDemo". The output of the program is displayed as follows:

```
I:\Study stuffs\Collection programs>java EmpListDemo
There are 5 elements in the list...
Contents of the list are ....
Java
J2EE
Spring
Php
Hibernate
Removing Php from the list...
Result of searching spring in the list is: true
Current size of list is : 4

I:\Study stuffs\Collection programs>
```

Program ListItrDemo.java

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

class ListItrDemo
{
    public static void main(String args[])
    {
        ArrayList list = new ArrayList();

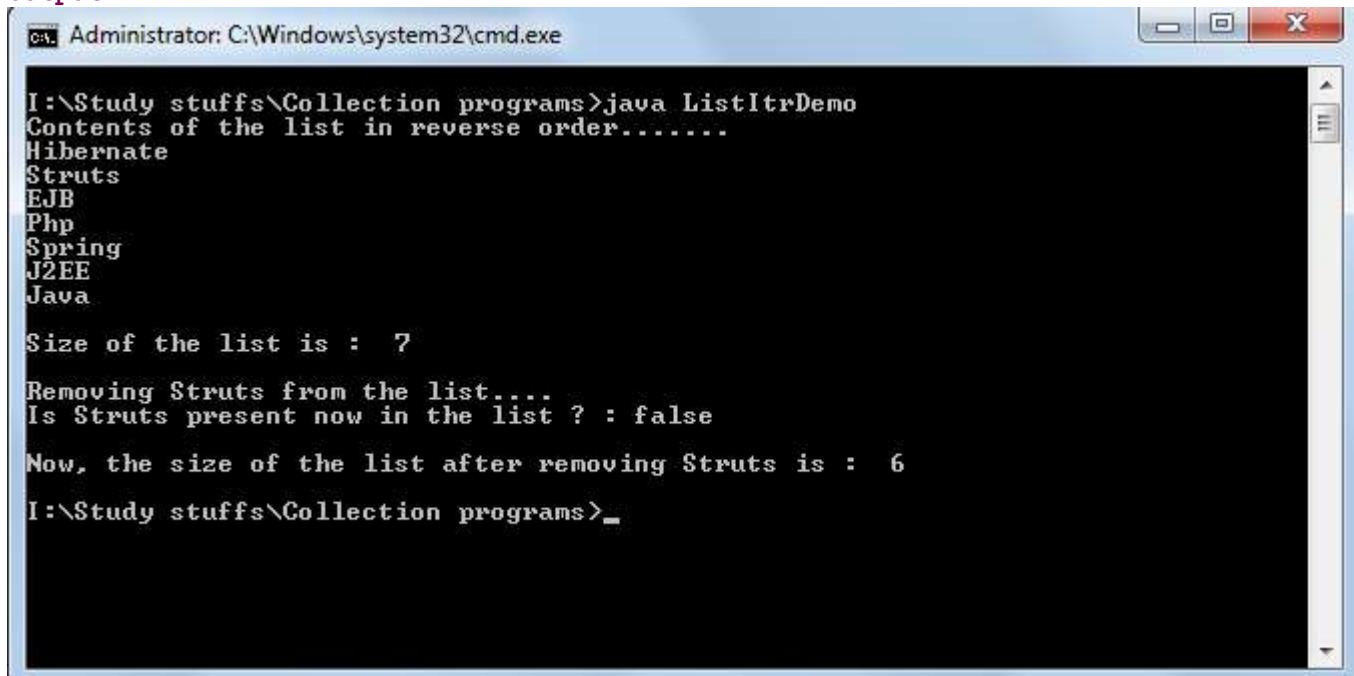
        list.add("Java");
        list.add("Spring");
        list.add("Php");
```

```
list.add(1, "J2EE");
list.add("EJB");
list.add("Struts");
list.add("Hibernate");

System.out.println("Contents of the list in reverse order.....");

ListIterator itr = list.listIterator(list.size());
while (itr.hasPrevious())
    System.out.println(itr.previous());
System.out.println();
System.out.println("Size of the list is : " + list.size());
System.out.println();
System.out.println("Removing Struts from the list....");
list.remove("Struts");
System.out.println("Is Struts present now in the list ? : " +
list.contains("Struts"));
System.out.println();
System.out.println("Now, the size of the list after removing Struts is : " +
list.size());
}
}
```

Output-



**Set** interface represents the functionality of a set. **Set** is a non-indexed collection of unique elements.

Functionality of set is represented by **Set** interface which extends **Collection** interface but does not define any additional methods. It simply specifies that implementation of **add()** method must be provided in such a manner that duplicate elements are not allowed./All implementation of **Set** defines **add()** in such a way that duplicate elements are not added.

**HashSet** & **TreeSet** classes provides implementation of **Set** interface through **AbstractSet** class. Difference between **HashSet** & **TreeSet** is in there internal data structure. A **HashSet** uses 'Hash table' for storing elements and **TreeSet** uses 'binary search tree'.

-----X-----

## J2EE Notes (By Neeraj Sir)

*Example -*

Program **SetTest.java**

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

class SetTest
{
    public static void main(String args[])
    {
        HashSet set = new HashSet();
        System.out.println("Result of adding Java first time in the set is : " + set.add("Java"));
        set.add("Spring");
        set.add("Php");
        set.add("J2EE");
        set.add("EJB");
        set.add("Struts");
        set.add("Hibernate");
        System.out.println("Result of adding Java second time in the set is : " + set.add("Java"));

        System.out.println("Contents of HashSet....");
        Iterator itr = set.iterator();
        while (itr.hasNext())
            System.out.println(itr.next());
        System.out.println();

        TreeSet ts = new TreeSet();
        ts.addAll(set);

        System.out.println("Contents of TreeSet....");
        itr = ts.iterator();
        while (itr.hasNext())
            System.out.println(itr.next());
    }
}
```

*Output-*

The screenshot shows a Windows desktop with two windows. The top-left window is a command prompt titled 'Administration C:\Windows\system32\cmd.exe'. It shows the execution of a Java program:
 

```

C:\Study stuffs\Collection program>java SetTest
Result of adding Java first time in the set is : true
Result of adding Java second time in the set is : false
Contents of HashSet....
{
hibernate
Struts
J2EE
Spring
JSP
Java
}
Contents of TreeSet....
{
hibernate
J2EE
Java
Spring
Struts
}
C:\Study stuffs\Collection program>
  
```

 The top-right window is a Notepad window titled 'SetTest - Notepad'. It contains the source code of the SetTest.java program:
 

```

import java.util.*;

import java.util.ArrayList;
import java.util.ListIterator;

class SetTest
{
    public static void main(String args[])
    {
        HashSet set = new HashSet();
        System.out.println("Result of adding Java first time in the set is : " + set.add("Java"));
        set.add("Spring");
        set.add("PHP");
        set.add("J2EE");
        set.add("JSP");
        set.add("Struts");
        set.add("hibernate");
        System.out.println("Result of adding Java second time in the set is : " + set.add("Java"));

        System.out.println("Contents of HashSet....");
        Iterator itr = set.iterator();
        while (itr.hasNext())
        {
            System.out.println(itr.next());
        }
        System.out.println();

        TreeSet ts = new TreeSet();
        ts.addAll(set);

        System.out.println("Contents of TreeSet....");
        itr = ts.iterator();
        while (itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
  
```

 The bottom window is a small window titled 'Example - Program SetTest.java' showing the same source code as the Notepad window.

## MAP

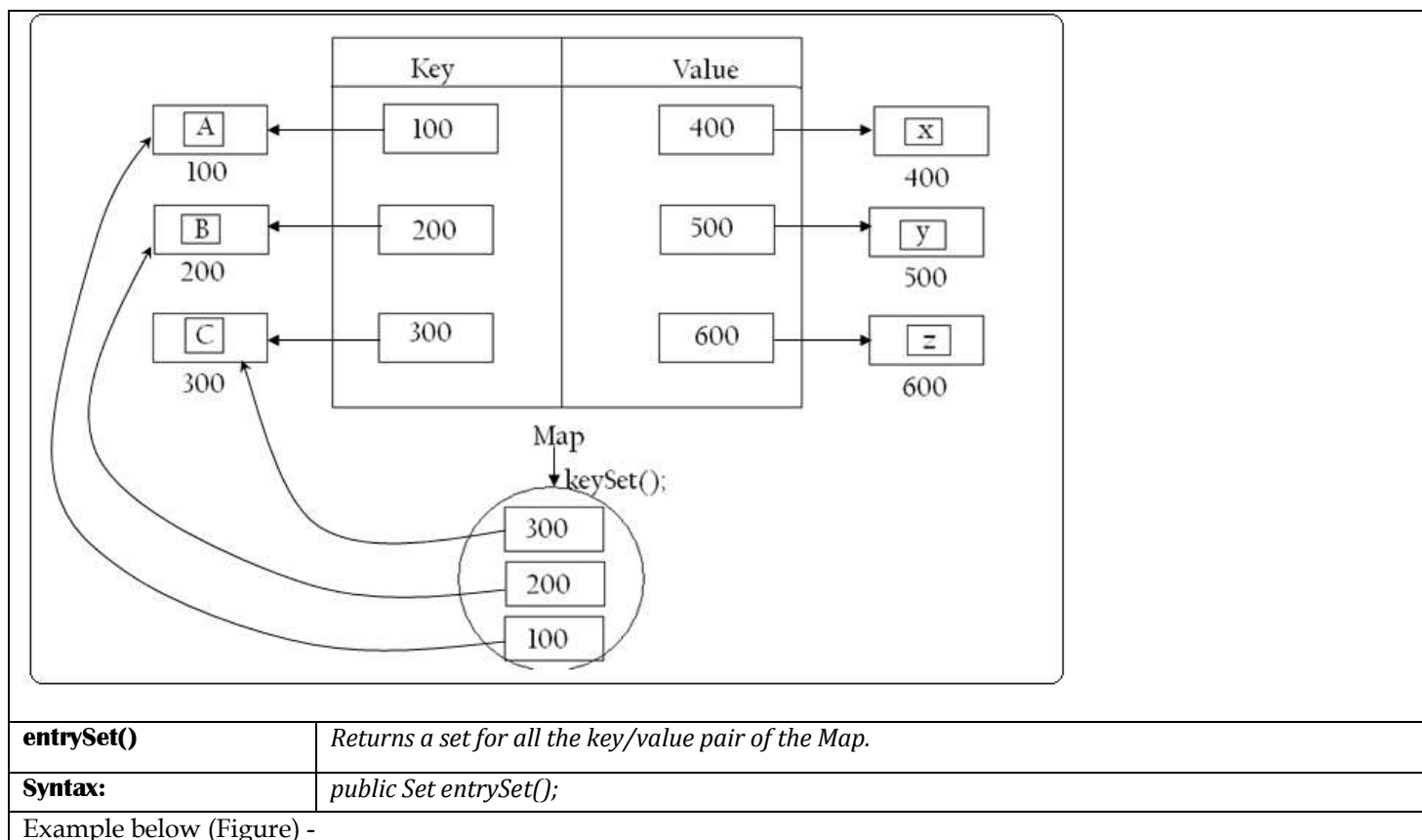
**Map** is used to store key/value pairs. Key is used to randomly search values i.e. key act as index.

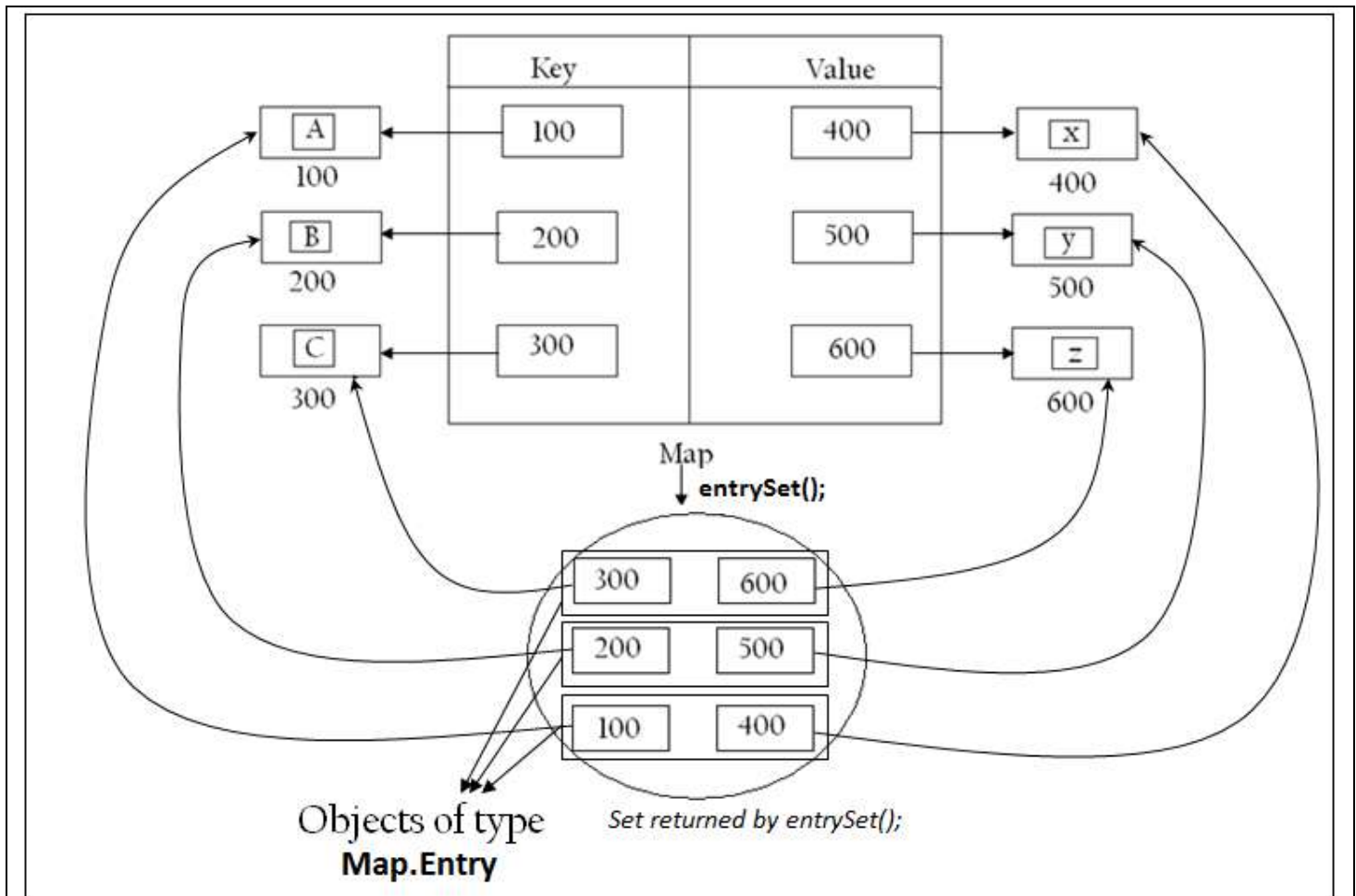
**Map** is a collection in which values are stored by associating them with keys which are used to retrieve values.

Functionality of a **Map** is described by **java.util.Map** interface which contains following methods:-

<b>put()</b>	Is used to store a key/value pair in the map.
<b>Syntax:</b>	<i>public void put(Object key, Object value);</i>
<b>get()</b>	Is used to obtain a value from a map.
<b>Syntax:</b>	<i>public Object get(Object key);</i>
<b>containsKey()</b>	is used to search a key in the map.
<b>Syntax:</b>	<i>public boolean containsKey(Object key);</i>
<b>contains()</b>	Is used to search a value in the map.
<b>Syntax:</b>	<i>public boolean contains(Object value);</i>
<b>size()</b>	Is used to find out the number of elements/entries in a map.
<b>Syntax:</b>	<i>public int size( );</i>
<b>keySet()</b>	Returns a set for traversing the key of a map./Returns a Set view for all the keys of the Map.
<b>Syntax:</b>	<i>public set keySet( );</i>
Example below (Figure) -	







The set that is returned by the entrySet() method contains objects of type Map.Entry.

**Entry** is a nested interface of **Map**. All implementations of **Map** provides implementation of this interface as well.

This interface provides two methods:-

<b>getKey()</b>	
<b>Syntax:</b>	<code>public Object getKey();</code>
<b>getValue()</b>	
<b>Syntax:</b>	<code>public Object getValue();</code>

**HashMap** & **TreeMap** classes provide implementations of **Map** interface through **AbstractMap** class.

Program MapDemo.java

```
import java.util.*;

public class MapDemo2
{
    public static void main(String[] args)
    {
        HashMap map = new HashMap();
        map.put("Java", 6000);
        map.put("J2EE", 6000);
        map.put("Spring & Hibernate", 7000);
        map.put(".Net", 8000);
    }
}
```

## J2EE Notes (By Neeraj Sir)

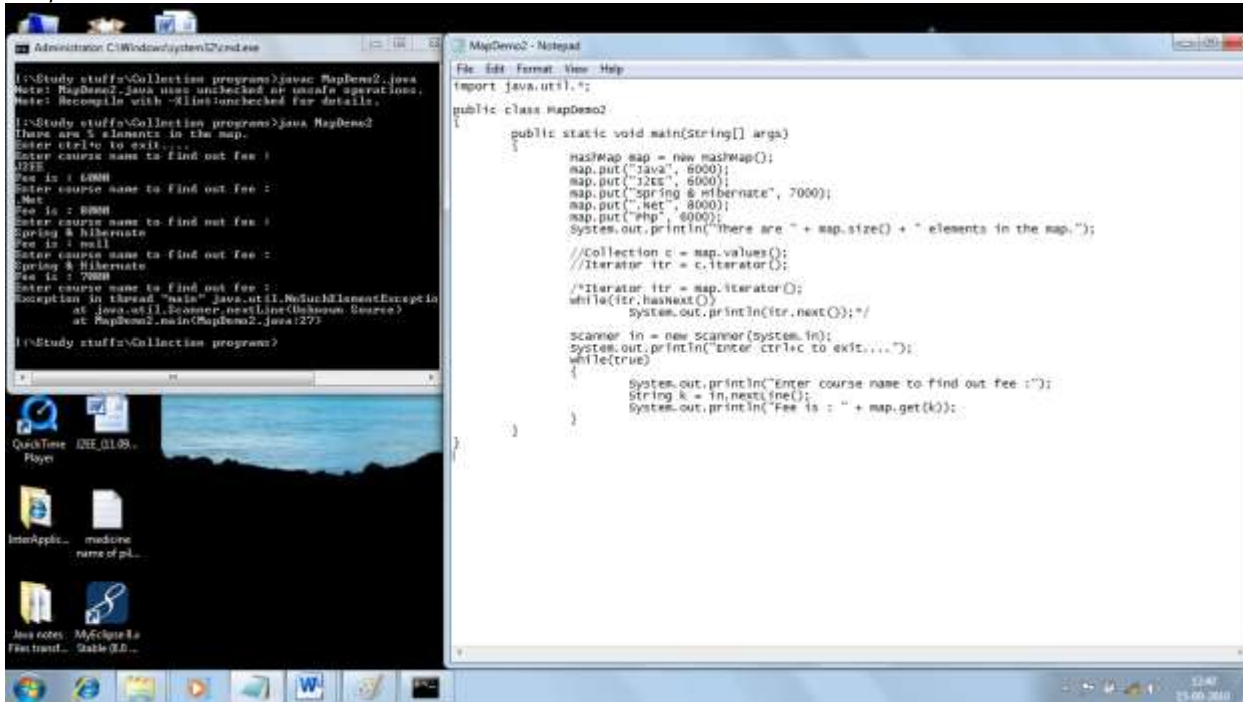
```
map.put("Php", 6000);
System.out.println("There are " + map.size() + " elements in the map.");

//Collection c = map.values();
//Iterator itr = c.iterator();

/*Iterator itr = map.iterator();
while(itr.hasNext())
    System.out.println(itr.next());*/

Scanner in = new Scanner(System.in);
System.out.println("Enter ctrl+c to exit....");
while(true)
{
    System.out.println("Enter course name to find out fee :");
    String k = in.nextLine();
    System.out.println("Fee is : " + map.get(k));
}
}
```

Output -



### Program MapDemo3.java

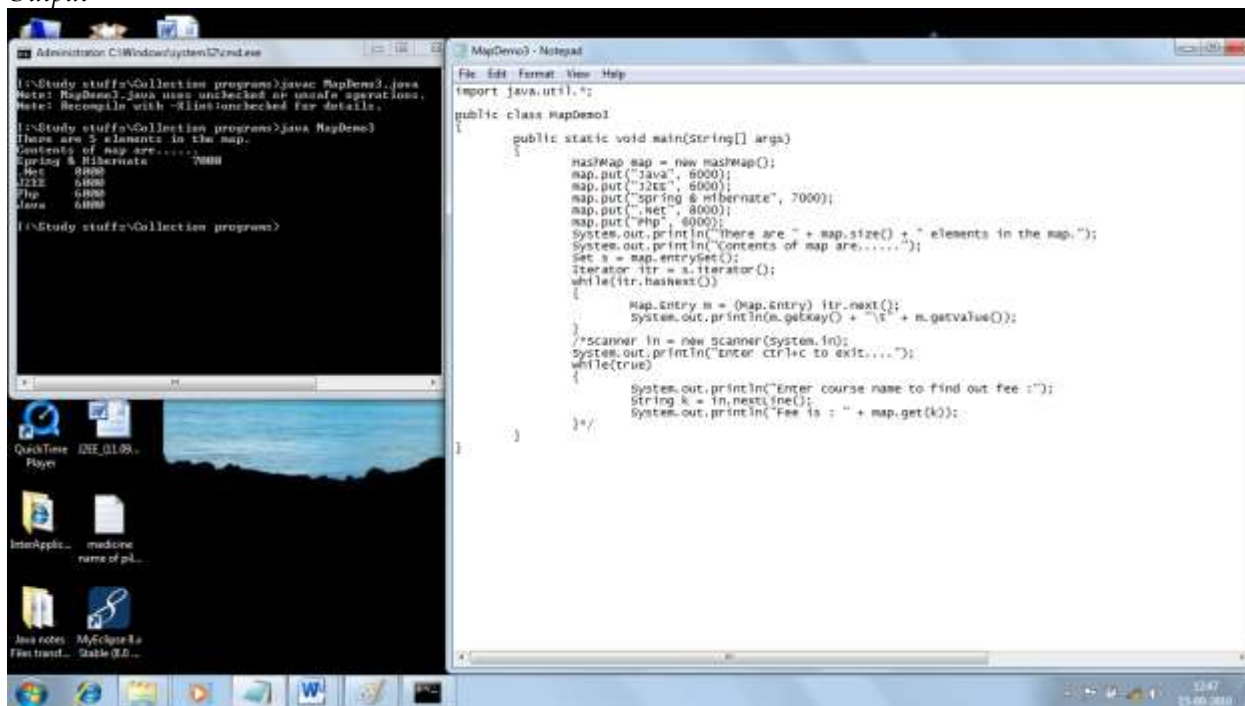
```
import java.util.*;

public class MapDemo3
{
    public static void main(String[] args)
    {
        HashMap map = new HashMap();
        map.put("Java", 6000);
        map.put("J2EE", 6000);
```

## J2EE Notes (By Neeraj Sir)

```
map.put("Spring & Hibernate", 7000);
map.put(".Net", 8000);
map.put("Php", 6000);
System.out.println("There are " + map.size() + " elements in the map.");
System.out.println("Contents of map are.....");
Set s = map.entrySet();
Iterator itr = s.iterator();
while(itr.hasNext())
{
    Map.Entry m = (Map.Entry) itr.next();
    System.out.println(m.getKey() + "\t" + m.getValue());
}
}
```

Output-



## Program TreeMap.java

```
package Collection.Map;

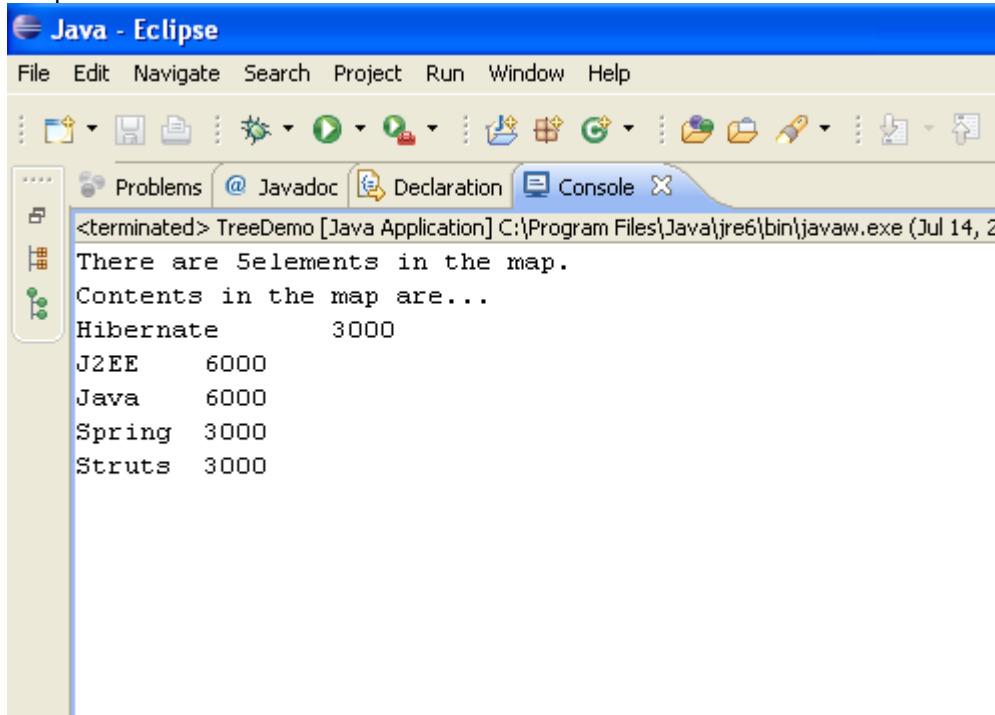
import java.util.*;

public class TreeDemo
{
    public static void main(String[] args)
    {
        TreeMap map = new TreeMap();
        map.put("Java", 6000);
        map.put("J2EE", 6000);
        map.put("Struts", 3000);
        map.put("Spring", 3000);
        map.put("Hibernate", 3000);
        System.out.println("There are " + map.size() + "elements in the map.");
    }
}
```

## J2EE Notes (By Neeraj Sir)

```
System.out.println("Contents in the map are...");
Set s = map.entrySet();
Iterator itr = s.iterator();
while(itr.hasNext())
{
    Map.Entry m = (Map.Entry) itr.next();
    System.out.println(m.getKey() + "\t" + m.getValue());
}
}
```

Output-



### Frequently Asked Questions (FAQ) on Collections –

1. Difference between Vector & ArrayList.
2. Difference between HashMap & Hashtable.
3. Difference between Iterator & Enumeration.
4. Difference between Iterator & ListIterator.
5. Difference between Comparator & Comparable.
6. Difference between Collection & Collections.
7. What is the role of equals() and hashCode() methods of Object class in Collections?
8. What is the role of loadfactor in hashing?
9. What is the complexity of a search operation in worst case in an ArrayList, HashSet & TreeSet?
10. In which case concurrent modification exception is thrown?
11. What is Fail Fast implementation?
12. How can a Collection be made unmodifiable?
13. How can a Collection be made singleton?
14. How can a non-synchronized Collection be made synchronized?
15. How is the size of a Vector & ArrayList be increased dynamically?
16. How the implementation of Iterator & Enumeration is provided by Collections?
17. How can an ArrayList be synchronized?

18. etc.

-----X-----

Date: 03.07.10

```

class Emp
{
    String name, job;

    int salary;

    public Emp(String n, String j, int s)
    {
        name = n;
        job = j;
        salary = s;
    }

    public void display()
    {
        System.out.println(name + "\t" + job + "\t" + salary);
    }
}

import java.util.*;

class EmpListDemo
{
    public static void main(String args[])
    {
        ArrayList list = new ArrayList();
        list.add(new Emp("Raj", "Accountant", 12000));
        list.add(new Emp("Amit", "Manager", 45000));
        list.add(new Emp("Sachin", "Administrator", 20000));
        list.add(new Emp("Vipul", "Director", 75000));
        list.add(new Emp("Nitin", "Clerk", 8000));
        list.add(new Emp("Sachin", "CFO", 65000));
        System.out.println("There are " + list.size() + " elements in the list.");
        System.out.println("Contents of the list....");

        Iterator itr = list.iterator();
        while(itr.hasNext())
        {
            Emp e = (Emp)itr.next();
            e.display();
        }

        System.out.println("Removing following employees from the list...");
        Emp e1 = new Emp("Vipul", "Director", 75000);
        e1.display();
        list.remove(e1);

        System.out.println("Searching following employees in the list...");
    }
}

```

## J2EE Notes (By Neeraj Sir)

```
Emp e2 = new Emp("Amit", "Manager", 45000);
e2.display();

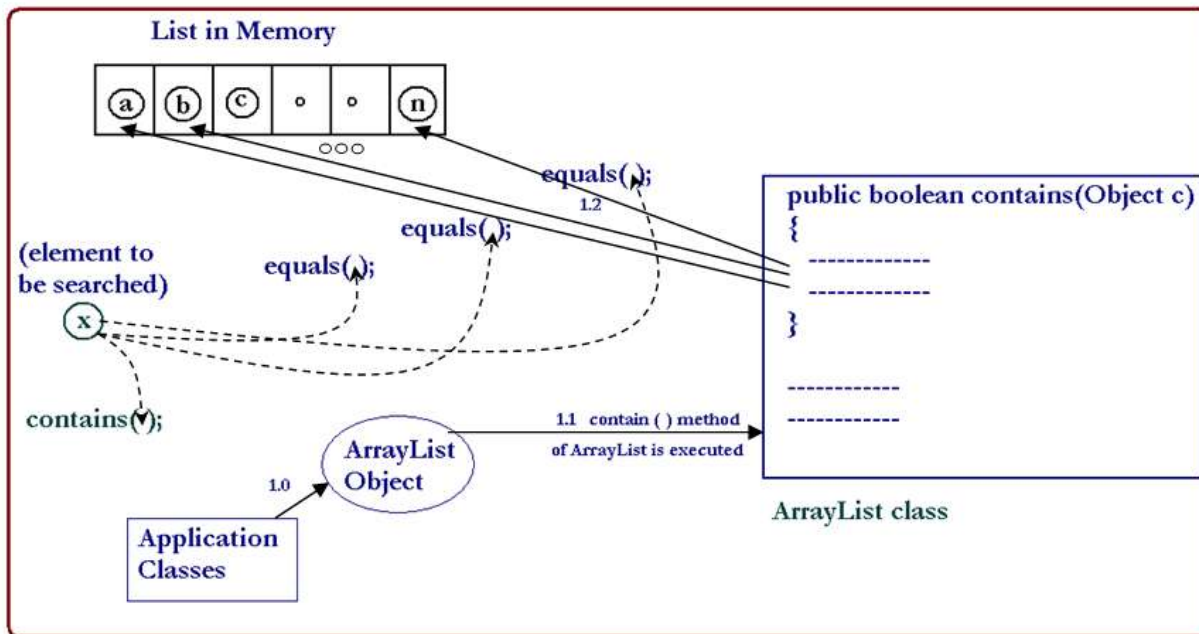
System.out.println("Result of search is : " + list.contains(e2));
System.out.println("Current size of list is Result of search is : " +
list.size());
}
```

### Output-

Element are not be removed & shall not be searched.

### Explanation -

Whenever an element is searched in a list using **contains()** method an exhaustive search is performed using **equals()** method all the elements of **List** are compared with the element to be searched using **equals()** method, i.e. success or failure of a search operation entirely depends on the implementation of **equals()** method. Default implementation of **equals()** in **Object** class compares the value of references not the contents of objects.



**1.2** from the **contains** method of **ArrayList** class, **equals()** method is invoked on list elements to match element to be searched with list elements.

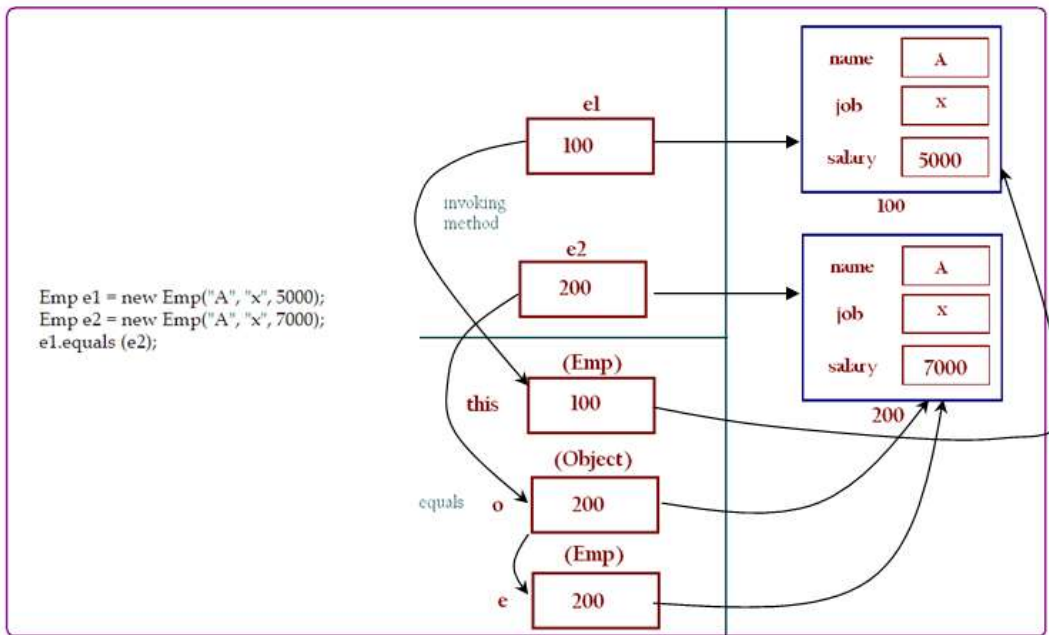
Comparison of objects is performed until a match is found or all the elements are compared.

In order to facilitate successful search of elements in a **List**, **equals()** method must be overridden in the **List** in such a manner that if contents of 2 objects are same, **true** is returned.

OR

In order to successfully search objects of a class in a list, **equals()** method must be overridden in the class to facilitate content wise comparison of objects.

```
Emp e1 = new Emp("A", "x", 5000);
Emp e2 = new Emp("A", "x", 7000);
e1.equals(e2);
```



### Logic applied in equals () method in Emp.java

```
public boolean equals (Object o)
{
    Emp e = (Emp) o;
    return this.name.equals(e.name) && this.job.equals(e.job) && this.salary == e.salary;
}
```

### Now, changes should be done in Emp class

```
class Emp
{
    String name, job;
    int salary;

    public Emp(String n, String j, int s)
    {
        name = n;
        job = j;
        salary = s;
    }

    public void display()
    {
        System.out.println(name + "\t" + job + "\t" + salary);
    }
}
```

**equals()**  
method  
overridden

```
public boolean equals(Object o)
{
    Emp e = (Emp) o;
    return this.name.equals(e.name) && this.job.equals(e.job) && this.salary == e.salary;
}
```

Program after adding **equals ()** method in **Emp1.java**  
import java.util.\*;



```
class Emp1
{
    String name, job;

    int salary;

    public Emp1(String n, String j, int s)
    {
        name = n;
        job = j;
        salary = s;
    }

    public void display()
    {
        System.out.println(name + "\t" + job + "\t" + salary);
    }

    public boolean equals(Object o)
    {
        Emp1 e = (Emp1) o;
        return this.name.equals(e.name) && this.job.equals(e.job) && this.salary == e.salary;
    }
}
```

```
class EmpListDemo1
{
    public static void main(String args[])
    {
        ArrayList list = new ArrayList();
        list.add(new Emp1("Raj", "Accountant", 12000));
        list.add(new Emp1("Amit", "Manager", 45000));
        list.add(new Emp1("Sachin", "Administrator", 20000));
        list.add(new Emp1("Vipul", "Director", 75000));
        list.add(new Emp1("Nitin", "Clerk", 8000));
        list.add(new Emp1("Sachin", "CFO", 65000));
        System.out.println("There are " + list.size() + " elements in the list.");
        System.out.println("Contents of the list....");

        Iterator itr = list.iterator();
        while(itr.hasNext())
        {
            Emp1 e = (Emp1)itr.next();
            e.display();
        }

        System.out.println("Removing following employees from the list...");
        Emp1 e1 = new Emp1("Vipul", "Director", 75000);
        e1.display();
        list.remove(e1);
    }
}
```

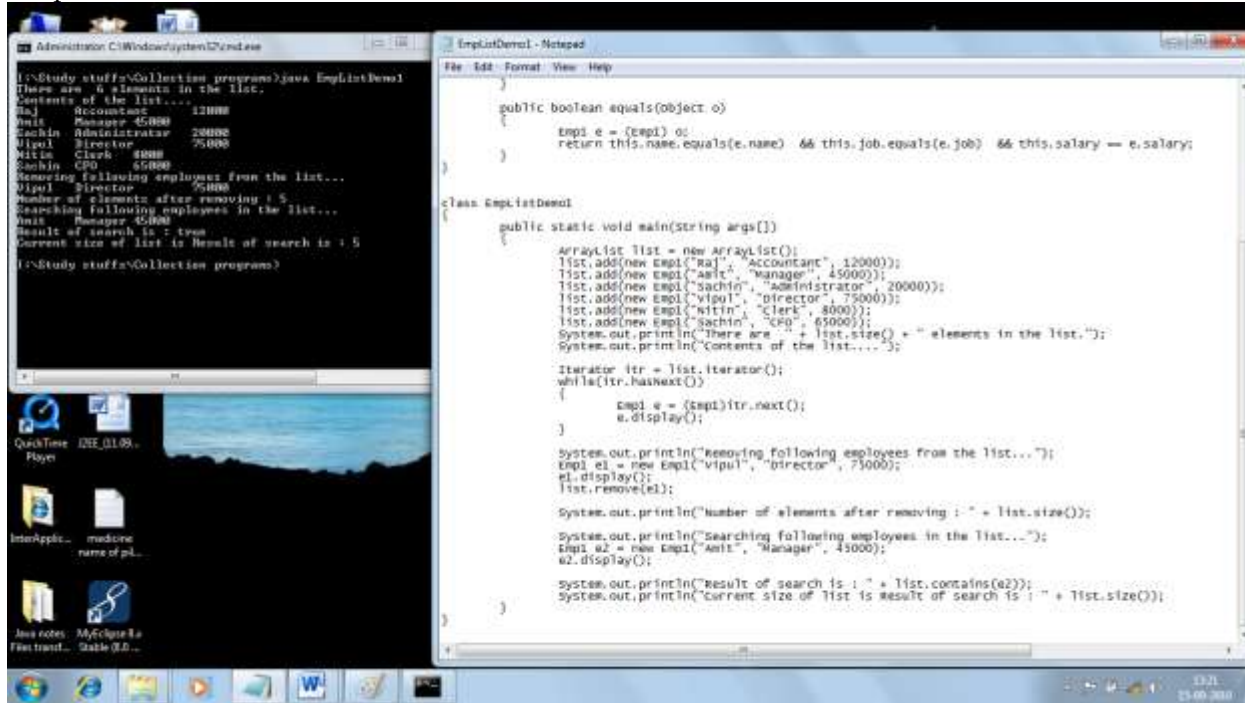
## J2EE Notes (By Neeraj Sir)

```
System.out.println("Number of elements after removing : " + list.size());

System.out.println("Searching following employees in the list...");
Emp1 e2 = new Emp1("Amit", "Manager", 45000);
e2.display();

System.out.println("Result of search is : " + list.contains(e2));
System.out.println("Current size of list is Result of search is : " + list.size());
}
}
```

### Output-



The screenshot shows a Windows desktop with a taskbar at the bottom. On the left, a command prompt window titled "Administrator: C:\Windows\system32\cmd.exe" displays the output of a Java program. The output shows the initial list of 6 employees, the removal of Vipul (Director, 75000), the search for Amit (Manager, 45000), and the final list size of 5. On the right, a Notepad window titled "EmpListDemo1 - Notepad" shows the source code of the program. The code defines an Emp1 class with a display method and a main method that uses an ArrayList to store employee objects, removes one, and searches for another.

```
Administrator: C:\Windows\system32\cmd.exe
C:\Study stuffs\Collection programs>java EmpListDemo1
There are 6 elements in the list.
Contents of the list....
Raj    Accountant    12000
Amit    Manager    45000
Sachin Administrator    20000
Vipul   Director    75000
Nitin   Clerk    80000
Sachin  CFO    65000
Removing following employees from the list...
Vipul   Director    75000
Number of elements after removing : 5
Searching following employees in the list...
Amit    Manager    45000
Result of search is : true
Current size of list is Result of search is : 5
C:\Study stuffs\Collection programs>
```

```
EmpListDemo1 - Notepad
File Edit Format View Help

public boolean equals(Object o)
{
    Emp1 e = (Emp1) o;
    return this.name.equals(e.name) && this.job.equals(e.job) && this.salary == e.salary;
}

class EmpListDemo1
{
    public static void main(String args[])
    {
        ArrayList list = new ArrayList();
        list.add(new Emp1("Raj", "Accountant", 12000));
        list.add(new Emp1("Amit", "Manager", 45000));
        list.add(new Emp1("Sachin", "Administrator", 20000));
        list.add(new Emp1("Vipul", "Director", 75000));
        list.add(new Emp1("Nitin", "Clerk", 80000));
        list.add(new Emp1("Sachin", "CFO", 65000));
        System.out.println("There are " + list.size() + " elements in the list.");
        System.out.println("Contents of the list....");

        Iterator itr = list.iterator();
        while(itr.hasNext())
        {
            Emp1 e = (Emp1)itr.next();
            e.display();
        }

        System.out.println("Removing following employees from the list...");
        Emp1 e1 = new Emp1("Vipul", "Director", 75000);
        e1.display();
        list.remove(e1);

        System.out.println("Number of elements after removing : " + list.size());

        System.out.println("Searching following employees in the list...");
        Emp1 e2 = new Emp1("Amit", "Manager", 45000);
        e2.display();

        System.out.println("Result of search is : " + list.contains(e2));
        System.out.println("Current size of list is Result of search is : " + list.size());
    }
}
```

### Program "EmpHsDemo.java"

```
import java.util.*;

class EmpHSDemo
{
    public static void main(String args[])
    {
        HashSet set = new HashSet();
        set.add(new Emp("Raj", "Accountant", 12000));
        set.add(new Emp("Amit", "Manager", 45000));
        set.add(new Emp("Sachin", "Administrator", 20000));
        set.add(new Emp("Vipul", "Director", 75000));
        set.add(new Emp("Nitin", "Clerk", 80000));
        set.add(new Emp("Sachin", "CFO", 65000));
        System.out.println("There are " + set.size() + " elements in the set.");
        System.out.println("Contents of the set....");
        Iterator itr = set.iterator();
```

```
        while(itr.hasNext())
        {
            Emp e = (Emp)itr.next();
            e.display();
        }

        System.out.println("Removing following emp from the set...");
        Emp e1 = new Emp("Vipul", "Director", 75000);
        e1.display();
        set.remove(e1);

        System.out.println("Searching following emp in the set...");
        Emp e2 = new Emp("Amit", "Manager", 45000);
        e2.display();

        System.out.println("Result of search is : " + set.contains(e2));
        System.out.println("Current size of set is Result of search is : " +
set.size());
    }
}
```

```
class Emp
{
    String name, job;
    int salary;

    public Emp(String n, String j, int s)
    {
        name = n;
        job = j;
        salary = s;
    }

    public void display()
    {
        System.out.println(name + "\t" + job + "\t" + salary);
    }

    public boolean equals(Object o)
    {
        Emp e = (Emp)o;
        return this.name.equals(e.name) && this.job.equals(e.job) && this.salary ==
e.salary;
    }
}
```

Output is not correct.

```

i:\Study stuffs\Collection programs>java EmpSetDemo
There are 6 elements in the set...
Contents of the set...
Rishi Manager 45000
Sachin Administrator 70000
Vipul Director 75000
Sachin CFO 65000
Raj Accountant 12000
Rishi Clerk 80000
Removing following emp from the set...
Vipul Director 75000
Searching following emp in the set...
Rishi Manager 45000
Result of search is : false
Current size of set is Result of search is : 6
i:\Study stuffs\Collection programs>

class Emp {
    String name, job;
    int salary;

    public Emp(String n, String j, int s) {
        name = n;
        job = j;
        salary = s;
    }

    public void display() {
        System.out.println(name + "\t" + job + "\t" + salary);
    }

    public boolean equals(Object o) {
        Emp e = (Emp)o;
        return this.name.equals(e.name) && this.job.equals(e.job) && this.salary == e.salary;
    }
}

while(itr.hasNext()) {
    Emp e = (Emp)itr.next();
    e.display();
}

System.out.println("Removing following emp from the set...");
Emp e1 = new Emp("Vipul", "Director", 75000);
e1.display();
set.remove(e1);

System.out.println("Searching following emp in the set...");
Emp e2 = new Emp("Rishi", "Manager", 45000);
e2.display();

System.out.println("Results of search is : " + set.contains(e2));
System.out.println("Current size of set is Result of search is : " + set.size());
    
```

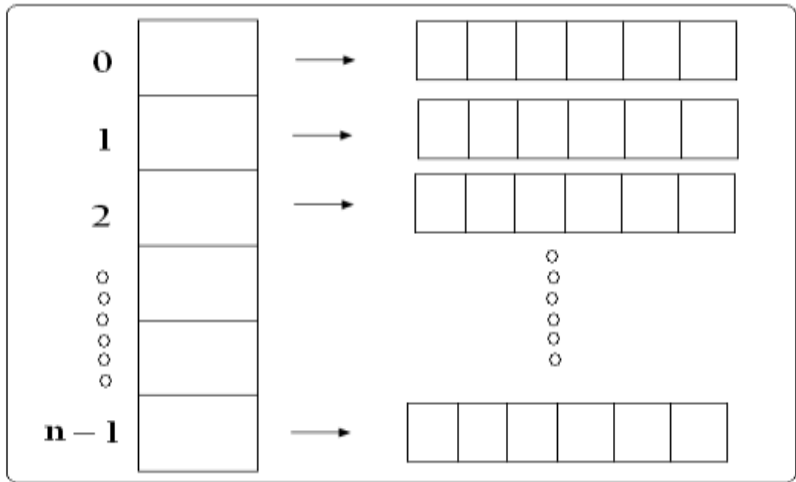
Working of Hashing based collections-

Hashing based collection uses HashTable for storing elements. A HashTable is implemented as an array of fixed size lists. Each list is called bucket.

Commonly used hashing function can be of the following form-  
 $n(e) = e \% n$

**HashTable** is implemented as an array of fixed size list which are called buckets.

See the figure given below for better understanding -



### HashTable

In order to store an element in the hash table hashing function is used which returns index of a bucket in which element is stored. A simple hashing function can be of the following form:-

$$n(e) = e \% n$$

where e is the element to be stored and n is the number of buckets.

Let there be 20 buckets and element 52 is to be stored.

$$\begin{aligned}n(52) &= 52\%20 \\ &= 12\end{aligned}$$

Bucket indexed is 12.

Most of the hashing functions are numeric which implies that only numeric elements can be added to a hash table.

In order to store objects in a hashing based collection objects must be represented by numbers. To facilitate numeric representation of objects hash code method is provided in Object class.

Numeric representation of objects is provided by **hashCode( )** method of **Object** class.

**public int hashCode();**

Implementation of **hashCode( )** method in **Object** class returns a unique number for each object based on the address of object i.e. default implementation does not consider contents of objects while generating hashCode.

Default implementation of **hashCode()** method in Object class generates hash code using the address of objects i.e. hash code of any two objects cannot be same according to default implementation.

Following steps are required to add objects to a hashing based collection:-

1. Hash code of object is obtained.
2. On the hash code of object hashing function is applied.
3. Object is stored in the bucket whose index is obtained by step2.

e.g.:- Let "o" be an object that is to be stored in a hashtable which contains 20 buckets.

- I.  $e = o.\text{hashCode}();$
- II.  $i = e \% n;$
- III. o shall be added to bucket indexed i.

In a hashing based collection when an element is searched using **contains()** method following steps are performed :-

- A. Hash code of object is obtained.
- B. On the hash code of object hashing function is applied.
- C. Object to be searched is compared with the elements of identified bucket using **equals()** method.  
That is in a hashing based collection a selective search is performed.

In a hashing based collection success or failure of a search operation depends on 2 factors:-

- a. Implementation of **hashCode()** method which is use to identify the bucket.
- b. Implementation of **equals()** method which is used to compare element to be searched with the elements of identified bucket.

Changes made in the last program (EmpHsDemo.java) are marked as **bold**:-

```
import java.util.*;

class EmpHSDemoWithHashCodeImplementation
{
    public static void main(String args[])
    {
        HashSet set = new HashSet();
        set.add(new Emp("Raj", "Accountant", 12000));
    }
}
```

## J2EE Notes (By Neeraj Sir)

```
        set.add(new Emp("Amit", "Manager", 45000));
        set.add(new Emp("Sachin", "Administrator", 20000));
        set.add(new Emp("Vipul", "Director", 75000));
        set.add(new Emp("Nitin", "Clerk", 80000));
        set.add(new Emp("Sachin", "CFO", 65000));
        System.out.println("There are " + set.size() + " elements in the set.");
        System.out.println("Contents of the set....");
        Iterator itr = set.iterator();

        while(itr.hasNext())
        {
            Emp e= (Emp) itr.next();
            System.out.println(e.hashCode()+ "\t");
            e.display();
        }

        System.out.println("Removing following emp from the set...");
        Emp e1 = new Emp("Vipul", "Director", 75000);
        System.out.println(e1.hashCode() + "\t");
        e1.display();
        set.remove(e1);

        System.out.println("Searching following emp in the set...");
        Emp e2 = new Emp("Amit", "Manager", 45000);
        System.out.println(e2.hashCode() + "\t");
        e2.display();

        System.out.println("Result of search is : " + set.contains(e2));
        System.out.println("Current size of set is Result of search is : " +
        set.size());
    }
}

class Emp
{
    String name, job;
    int salary;

    public Emp(String n, String j, int s)
    {
        name = n;
        job = j;
        salary = s;
    }

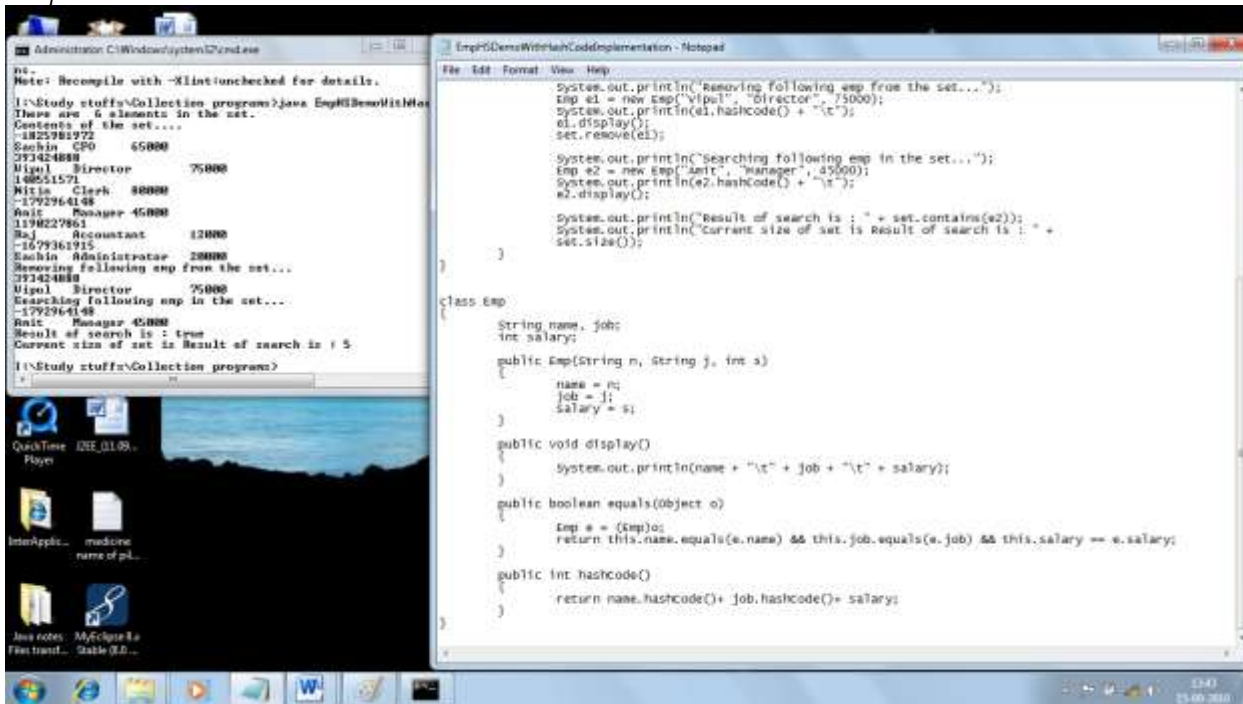
    public void display()
    {
        System.out.println(name + "\t" + job + "\t" + salary);
    }

    public boolean equals(Object o)
    {
        Emp e = (Emp)o;
        return this.name.equals(e.name) && this.job.equals(e.job) && this.salary ==
e.salary;
    }
}
```

## J2EE Notes (By Neeraj Sir)

```
public int hashCode()
{
    return name.hashCode() + job.hashCode() + salary;
}
}
```

Output-



### TreeSet

EmpTsDemo.java

```
import java.util.*;

class EmpTsDemo
{
    public static void main(String args[])
    {
        TreeSet set = new TreeSet();
        set.add(new Emp("Raj", "Accountant", 12000));
        set.add(new Emp("Amit", "Manager", 45000));
        set.add(new Emp("Sachin", "Administrator", 20000));
        set.add(new Emp("Vipul", "Director", 75000));
        set.add(new Emp("Nitin", "Clerk", 80000));
        set.add(new Emp("Sachin", "CFO", 65000));
        System.out.println("There are " + set.size() + " elements in the list.");
        System.out.println("Contents of the list....");
        Iterator itr = set.iterator();

        while(itr.hasNext());
        {
            Emp e = (Emp) itr.next();
            System.out.println(e.hashCode() + "\t");
            e.display();
        }
        System.out.println("Removing following emp from the set...");
    }
}
```

```
Emp e1 = new Emp("Vipul", "Director", 75000);
e1.display();
set.remove(e1);
System.out.println("Searching following emp in the set...");

Emp e2 = new Emp("Amit", "Manager", 45000);
e2.display();
System.out.println("Result of search is : " + set.contains(e2));
System.out.println("Current size of list is Result of search is : " +
set.size());
}
```

In order to add objects of a class in a tree based collection class must contain sorting logic.

Sorting logic can be added to a class in the following 2 ways:-

- a. using **java.lang.Comparable**
- b. using **java.util.Comparator**

**Comparable interface** is used to make a class comparable i.e. if a class is comparable, order of any two objects of the class can be obtained.

This interface provides a single method named **compareTo()**.

**public int compareTo(Object o);**

Thus, we need to add **compareTo()** method in **Emp.java** and also **class Emp** needs to **implements Comparable**

```
class Emp implements Comparable
{
    String name, job;
    int salary;

    public Emp(String n, String j, int s)
    {
        name = n;
        job = j;
        salary = s;
    }

    public void display()
    {
        System.out.println(name + "\t" + job + "\t" + salary);
    }

    public boolean equals(Object o)
    {
        Emp e = (Emp)o;
        return this.name.equals(e.name) && this.job.equals(e.job) && this.salary ==
e.salary;
    }

    public int hashCode()
    {
        return name.hashCode() + job.hashCode() + salary;
    }
}
```



```
public int compareTo(Object o)
{
    Emp e = (Emp) o;
    return this.name.compareTo(e.name); // name wise sorting

    //return this.job.compareTo(e.job); //job wise sorting

    //return this.salary-e.salary; //salary wise sorting
}
```

Using the **Comparable** interface single sorting order can be defined for the objects of a class.

**Comparator** interface is used to define multiple sorting orders for the objects of a class. It contains 2 methods.

```
public int compare(Object o, Object p);
public boolean equals(Object o);
```

-----X-----

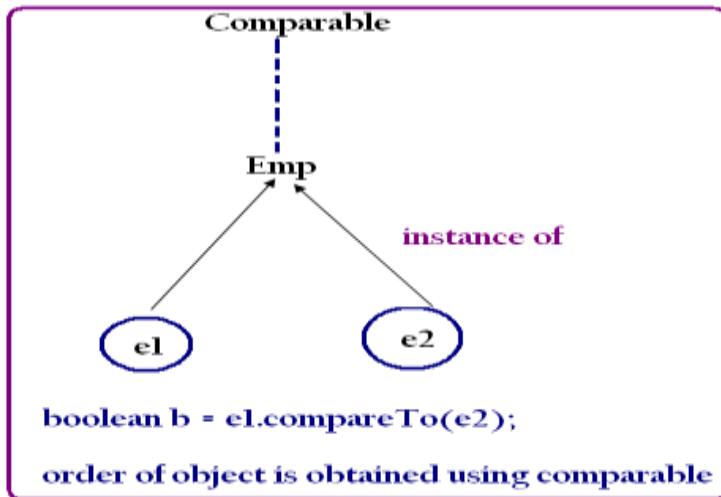
Date: 04.07.10

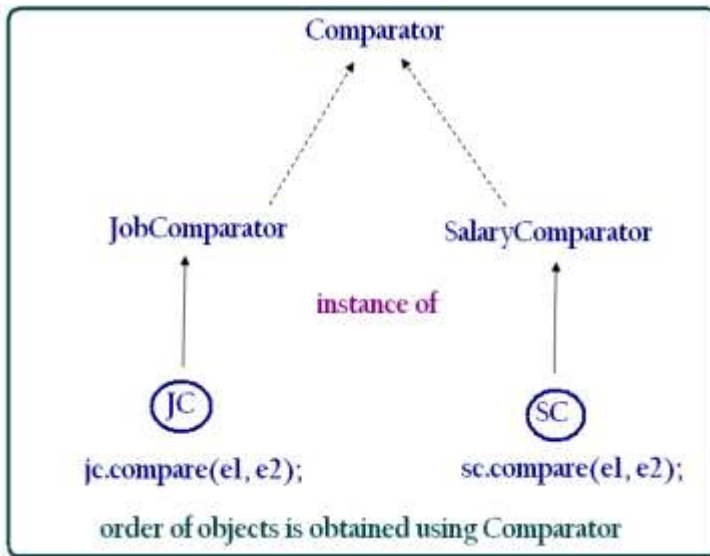
Comparable :-

```
public boolean compareTo(Object o);
```

Comparator:-

```
public int compare(Object o, Object p);
```





Multiple sorting orders can be defined using **Comparator** because **Comparator** interface is implemented in a separate class i.e. for defining multiple sorting orders for the objects of a class, multiple implementations of **Comparator** interface can be provided.

**NOTE:** **Comparable** is implicitly used by Tree based collection whereas **Comparator** needs to be explicitly specified.

Program **JobComparator.java**

```

import java.util.*;

public class JobComparator implements Comparator
{
    public int compare(Object o, Object p)
    {
        Emp e = (Emp)o;
        Emp f = (Emp)p;
        return e.job.compareTo(f.job);
    }
}

```

Program **SalaryComparator.java**

```

import java.util.*;

public class SalaryComparator implements Comparator
{
    public int compare(Object o, Object p)
    {
        Emp e = (Emp)o;
        Emp f = (Emp)p;
        return e.salary - f.salary;
    }
}

```

Program **EmpTsDemo.java**

```

import java.util.*;

```

## J2EE Notes (By Neeraj Sir)

```
class EmpTsDemo
{
    public static void main(String args[])
    {
        TreeSet set = new TreeSet(); //comparable shall be used for ordering
        set.add(new Emp("Raj", "Accountant", 12000));
        System.out.println("First element added.");

        set.add(new Emp("Amit", "Manager", 45000));
        System.out.println("Second element added.");

        set.add(new Emp("Sachin", "Administrator", 20000));
        set.add(new Emp("Vipul", "Director", 75000));
        set.add(new Emp("Nitin", "Clerk", 80000));
        set.add(new Emp("Saurabh", "CFO", 65000));
        System.out.println("There are " + set.size() + " elements in the list.");
        System.out.println("Contents of the set in the ascending order of Name
        ....");

        Iterator itr = set.iterator();

        while(itr.hasNext());
        {
            Emp e= (Emp) itr.next();
            e.display();
        }

        TreeSet ts1 = new TreeSet(new JobComparator()); //JobComparator shall be used
        for ordering
        ts1.addAll(set);
        System.out.println("Contents of the set in the ascending order of Job ....");
        itr = ts1.iterator();
        while(itr.hasNext());
        {
            Emp e= (Emp) itr.next();
            e.display();
        }

        TreeSet ts2 = new TreeSet(new SalaryComparator()); //SalaryComparator shall
        be used for ordering
        ts2.addAll(set);
        System.out.println("Contents of the set in the ascending order of Salary
        ....");
        itr = ts2.iterator();
        while(itr.hasNext());
        {
            Emp e= (Emp) itr.next();
            e.display();
        }
    }
}
```

COLLECTION		COMPLEXITY OF SEARCH OPERATION IN WORST CASE
1. ArrayList		O(n) where n is the size of list.
2. HashMap	HashSet	O(1) at most (k is the size of bucket) comparison shall be made. k is fixed hence

	complexity shall be constant.
3. TreeSet	$O(\log 2n)$ at most comparison shall be equal to the height of BST. For a binary tree of $n$ elements height can be at most $\log 2n$ hence the complexity.

### Q. What is the role of load factor in hashing function?

Load factor is a constant whose value can be between 0 to 1. Value of load factor determines when the size of buckets is to be modified. In a hash table performance of insertion & search operation and space utilization are two inversely proportional objectives i.e. if space utilization is maximized performance is decreased because optimization of space utilization results in Hash collisions which degrades performance.

Optimization of performance results in under utilization of space.

By default value of load factor is .75 which represents that size of buckets is increased when 75% buckets are filled.

### Q. Difference between Vector and ArrayList.

**Vector** – **Vector** is a legacy collection that represents dynamic array. After the introduction of Collection framework **Vector** is modified to make it compatible to the **Collection** framework.

Major difference between Vector and ArrayList is of thread safety.

1. Vector is thread-safe whereas ArrayList is not.
2. Vector contains methods of **Collection** and **List** interface as well as its own legacy methods.
3. Apart from iterator a vector provides **Enumeration** for traversing its elements.

### Legacy methods of vector:-

- addElement( )** :- is used to add an element to a vector.  
**public boolean addElement(Object element);**
- removeElement( )** :- is used to remove an element from the vector.  
**public boolean removeElement (Object element)**
- capacity( )** :- returns the capacity of the vector. **capacity( )** method represents number of elements that can be added to a vector without changing its size.  
**public int capacity( );**
- elements( )** :- returns an enumeration for traversing contents of a vector.  
**public Enumeration elements( );**

- elements( )** :- returns an enumeration for traversing contents of a vector.  
**public Enumeration elements( );**
- Enumeration** is an interface that provides methods which are used for traversing the elements of legacy collection. It provides following methods:-
  - 1) **hasMoreElements( )** :- returns true if there is an element to be traversed.  
**public boolean hasMoreElements( );**
  - 2) **nextElement( )** :- returns the reference of next element.  
**public Object nextElement( );**

Program VectorDemo.java

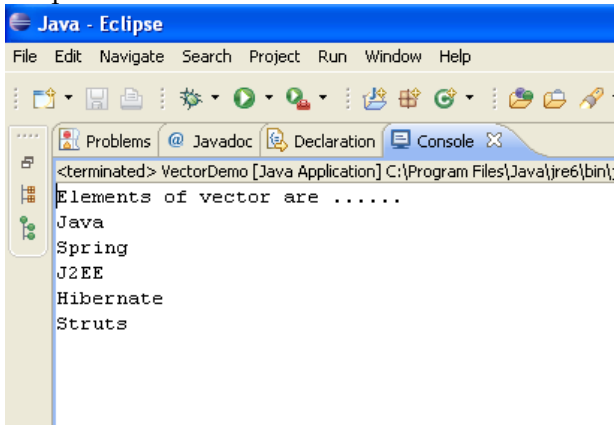
```
import java.util.*;

public class VectorDemo
{
    public static void main(String[] args)
    {
```

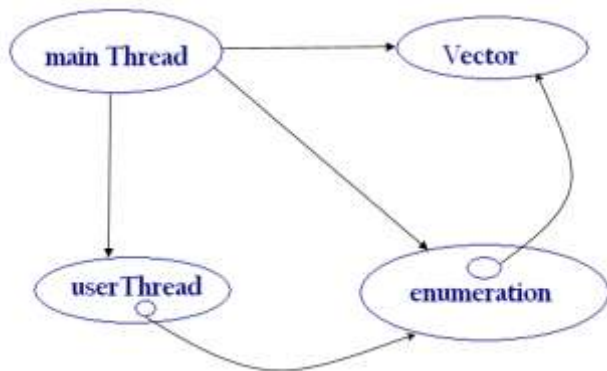
## J2EE Notes (By Neeraj Sir)

```
Vector v = new Vector();
v.addElement("Java");
v.add("J2EE");
v.add(1, "Spring");
v.addElement("Hibernate");
v.add("Struts");
System.out.println("Elements of vector are .....");
Enumeration e = v.elements();
while(e.hasMoreElements())
    System.out.println(e.nextElement());
}
```

Output -



### Difference between Enumeration & Iterator



Program EnumDemo.java

```
import java.util.Enumeration;
import java.util.Vector;

class EnumDemo
{
    static Enumeration e;

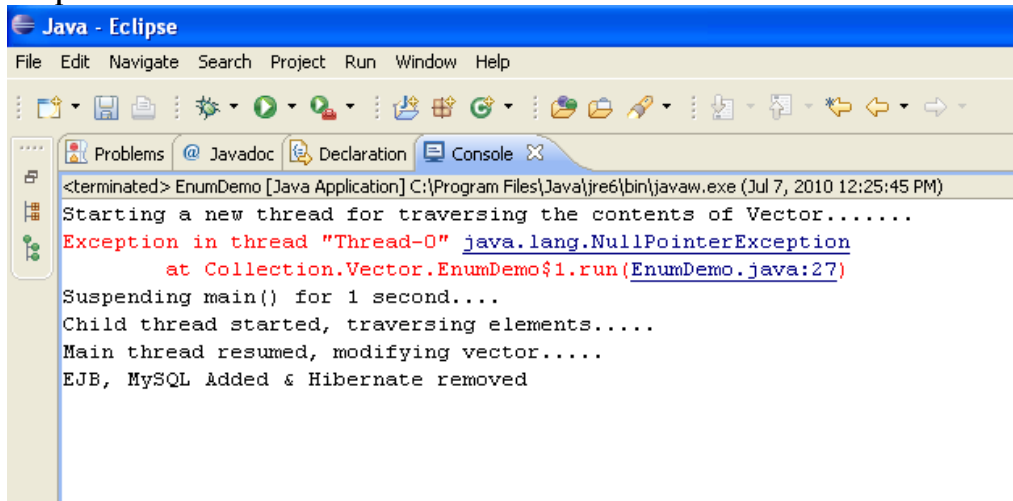
    public static void main(String[] args)
    {
        Vector v = new Vector();
        v.addElement("Java");
        v.add("J2EE");
        v.add(1, "Spring");
```

## J2EE Notes (By Neeraj Sir)

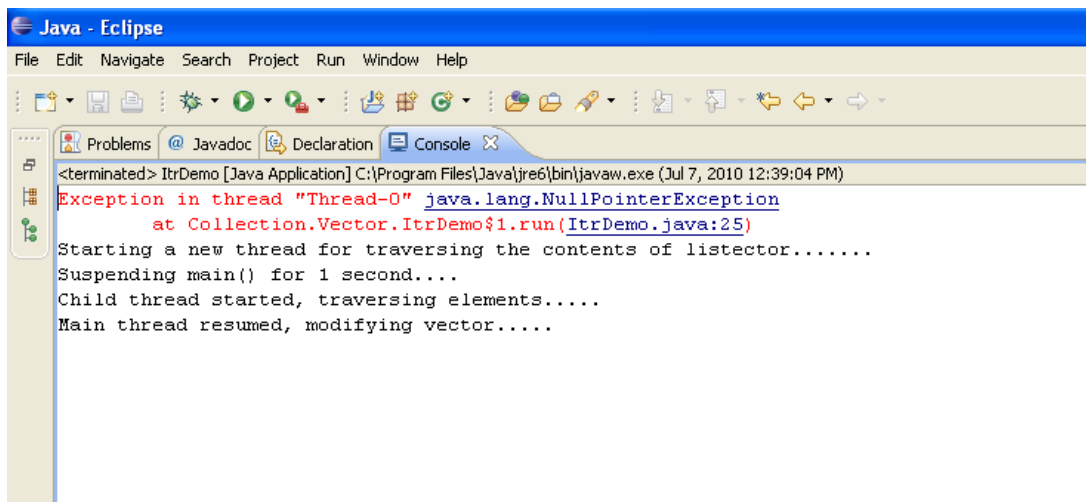
```
v.addElement("Hibernate");
v.add("Struts");
System.out.println("Starting a new thread for traversing the contents of
Vector.....");

Thread t = new Thread(){ // Anonymous inner class that extends thread
    public void run()
    {
        System.out.println("Child thread started, traversing
        elements.....");
        while(e.hasMoreElements())
        {
            System.out.println(e.nextElement());
            try
            {
                Thread.sleep(2000);
            }
            catch(Exception ex){}
        }
    }
};
t.start();
System.out.println("Suspending main() for 1 second....");
try{
    Thread.sleep(1000);
}
catch(Exception ex){}
System.out.println("Main thread resumed, modifying vector.....");
v.add("EJB");
v.add("SQL");
v.remove("Hibernate");
System.out.println("EJB, MySQL Added & Hibernate removed");
}
```

### Output -



### Output -



Major difference between Enumeration & Iterator is in their implementation.

An **Iterator** is designed to fail fast i.e. an iterator fails as soon as the modification to the underlying collection is detected by throwing concurrent modification exception.

The reason behind such an implementation is that iterator is mostly used for traversing the contents of non-thread safe collection in which consistently of modification operation is not guaranteed.

```
class MyVector
{
    Object elements[];
    int size;
    int capacity;

    public MyVector()
    {
        this(10);
    }

    public MyVector(int c)
    {
        elements = new Object[c];
        capacity = c;
        size = 0;
    }

    public int size()
    {
        return size;
    }

    public boolean add()
    {
        if (size < capacity)
        {
            elements[size] = 0;
            size++;
            return true;
        }
        else
            return false;
    }

    public boolean addElement(Object o)
```

```
{
    return add(o);
}

public Enumeration elements()
{
    Enumeration e = new Enumeration() { // anonymous inner class that implements
        Enumeration

        int index;

        public boolean hasMoreElements()
        {
            return (index < size);
        }

        public Object nextElement()
        {
            Object temp = elements[index];
            index++;
            return temp;
        }
    };
    return e;
}
```

-----X-----



## COLLECTIONS

**C**OLLECTIONS is a utility class that represents static methods which represents algorithms common to all *collections*. Such as making a *collection* unmodifiable, making a *collection* singleton, making a *collection* synchronized, sorting the contents of a list, finding out minimum or a maximum in a list etc.

```
package Collections;
```

```
import java.util.*;
```

```
class MyArrayList
```

```
{
    Object elements[];
    int size;
    boolean modified;

    public MyArrayList()
    {
        this(10);
    }

    public MyArrayList(int c)
    {
        elements = new Object[c];
        size = 0;
        modified = false;
    }

    public int size()
    {
        return size;
    }

    public boolean add(Object e)
    {
        if(size<10)
        {
            elements[size]=e;
            size++;
            modified=true;
            return true;
        }
        else
            return false;
    }

    public Iterator iterator()
    {
        Iterator itr = new Iterator()
        {
```

```
        int index = 0;

        public boolean hasNext()
        {
            if(modified)
                throw(new ConcurrentModificationException());
            if (index<size)
                return true;
            else
                return false;
        }

        public Object next()
        {
            if(modified)
                throw(new ConcurrentModificationException());
            Object o = elements[index];
            index++;
            return o;
        }

        public void remove()
        {
        }
    };

    modified = false;
    return itr;
}
}
```

```
import java.util.*;
```

```
public class ItrDemo
{
```

```
    static Iterator itr;
```

```
    public static void main(String[] args)
    {
```

```
        ArrayList list = new ArrayList();
        list.add("Java");
        list.add("J2EE");
        list.add(1, "Spring");
        list.add("Hibernate");
        list.add("Struts");
        System.out.println("Starting a new thread for traversing the contents of listector.....");
```

```
        Thread t = new Thread(){ // Anonymous inner class that extends thread
```

```
            public void run()
            {
```

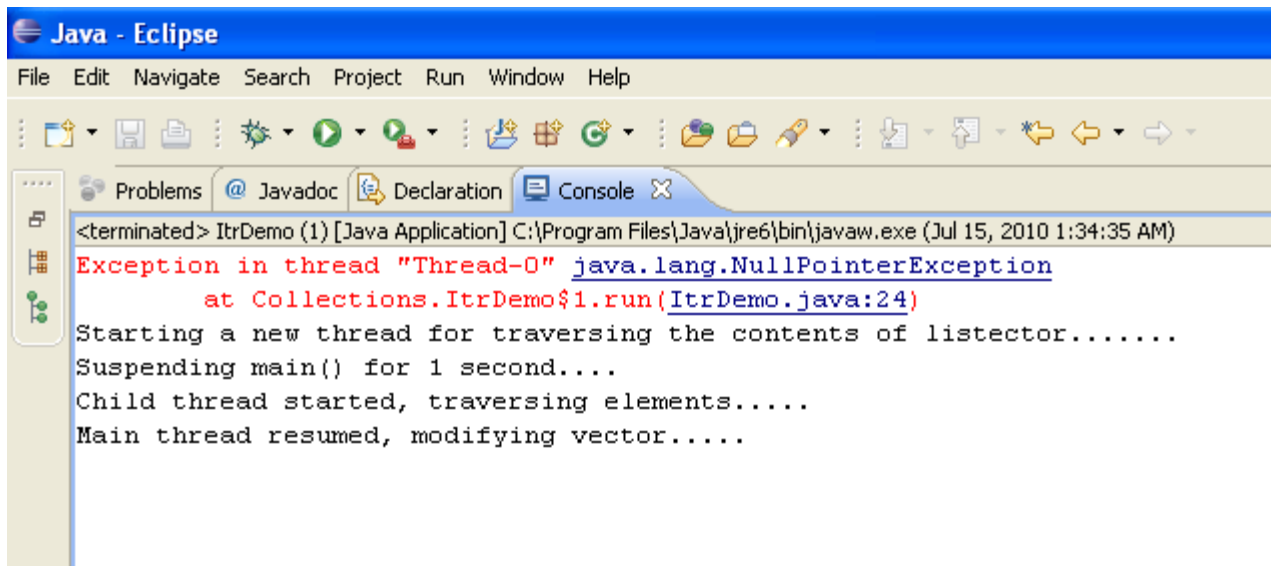
```
                System.out.println("Child thread started, traversing elements.....");
```

```

        while(itr.hasNext())
        {
            System.out.println(itr.next());
            try
            {
                Thread.sleep(2000);
            }
            catch(Exception ex){}
        }
    };
    t.start();
    System.out.println("Suspending main() for 1 second....");
    try{
        Thread.sleep(1000);
    }
    catch(Exception ex){}
    System.out.println("Main thread resumed, modifying vector.....");
    list.add("EJB");
    list.add("SQL");
    list.remove("Hibernate");
}
}

```

Output -



```

Java - Eclipse
File Edit Navigate Search Project Run Window Help
<terminated> ItrDemo (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jul 15, 2010 1:34:35 AM)
Exception in thread "Thread-0" java.lang.NullPointerException
    at Collections.ItrDemo$1.run(ItrDemo.java:24)
Starting a new thread for traversing the contents of listector.....
Suspending main() for 1 second....
Child thread started, traversing elements.....
Main thread resumed, modifying vector.....

```

```

package Collections;

import java.util.*;
import java.util.ArrayList;

class EmpListDemo
{

```

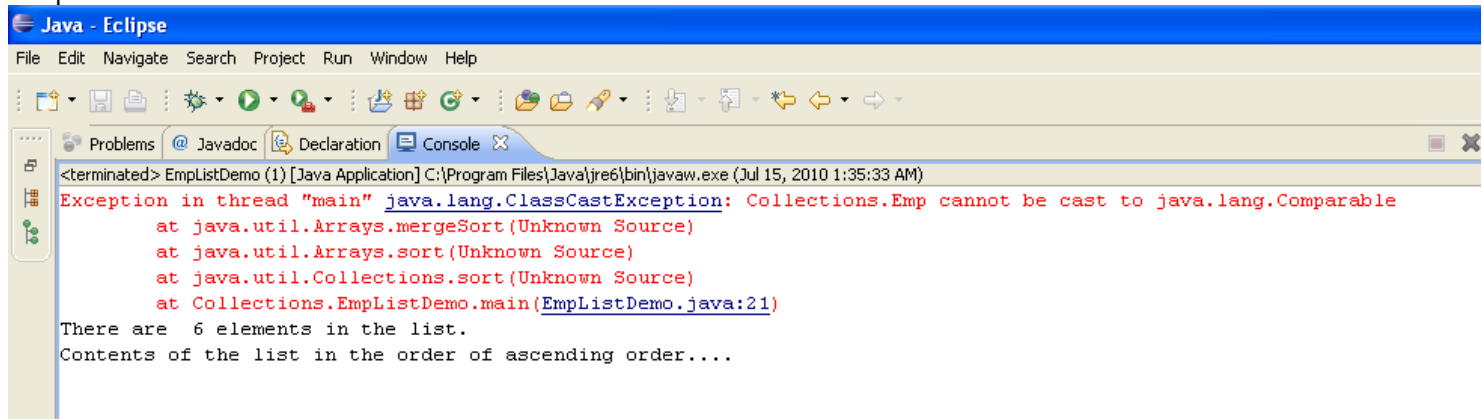
```

public static void main(String args[])
{
    ArrayList list = new ArrayList();
    list.add(new Emp("Raj", "Accountant", 12000));
    list.add(new Emp("Amit", "Manager", 45000));
    list.add(new Emp("Sachin", "Administrator", 20000));
    list.add(new Emp("Vipul", "Director", 75000));
    list.add(new Emp("Nitin", "Clerk", 8000));
    list.add(new Emp("Sachin", "CFO", 65000));
    System.out.println("There are " + list.size() + " elements in the list.");
    System.out.println("Contents of the list in the order of ascending order....");

    Iterator itr = list.iterator();
    Collections.sort(list); // Contents of list are sorted using comparable
    Collections.sort(list, new JobComparator()); // Contents of list are sorted using JobComparable
    while(itr.hasNext());
    {
        Emp e = (Emp)itr.next();
        e.display();
    }
}
}

```

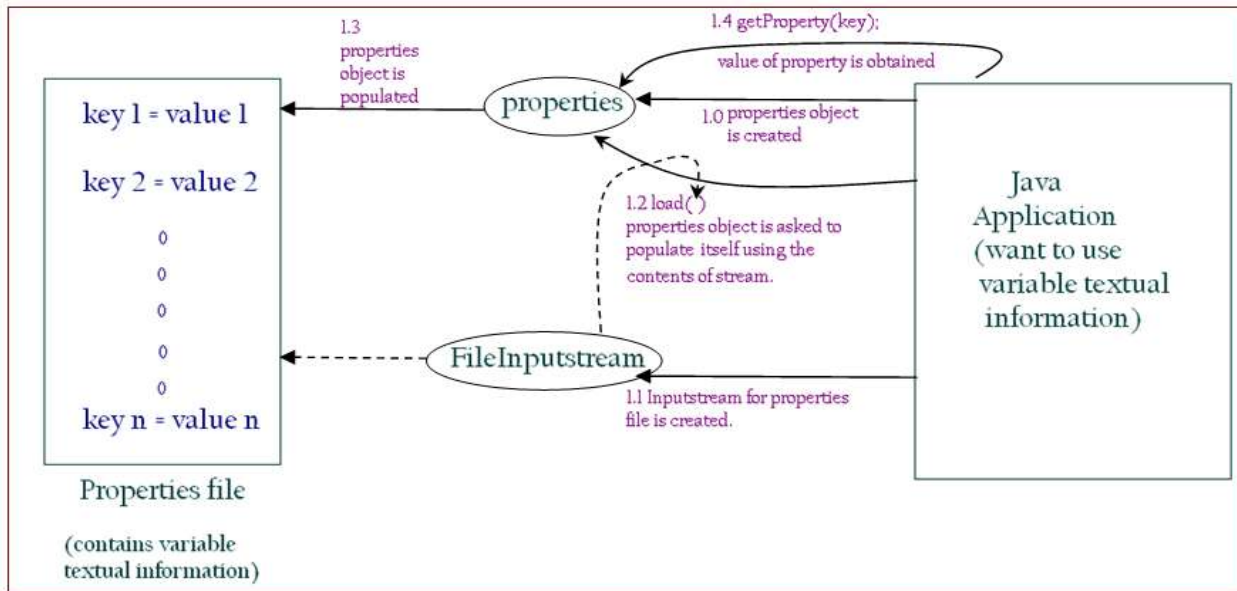
Output -



- **java.util.Properties** is a subclass of **HashTable** which is synchronized version of **HashMap**.

HashTable	HashMap
Legacy	Collection Framework
Synchronized	No-Synchronized

**Properties** class is used to manage text file that contains variable textual information as **key/value** pair. A property file is a text file that contains variable textual information as **key/value** pair.



Commonly used methods of Properties class –

1. **setProperty()** method is used to store a key-value pair in the Properties object.

**public void setProperty(String key, String value);**

2. **getProperty()** method is used to obtain the value of a property.

**public String getProperty(String key);**

3. **load()** method is used to load the contents of a Properties file into your Property object.

**public void load(InputStream stream);**

4. **store()** method is used to store the contents of a Properties object to a Properties file.

**public void store(OutputStream Stream, String comment);**

etc.

Example –

Program PropReader.java

```
import java.io.FileInputStream;
import java.util.Iterator;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class PropReader
{
    public static void main(String[] args)
    {
        try
        {
            Properties p = new Properties();
            FileInputStream f = new FileInputStream("Student.properties");
            p.load(f);
```

## J2EE Notes (By Neeraj Sir)

```
        Set s = p.entrySet();
        Iterator itr = s.iterator();
        System.out.println("Following properties are read....");
        while(itr.hasNext())
        {
            Map.Entry m = (Map.Entry) itr.next();
            System.out.println(m.getKey() + "\t" + m.getValue());
        }
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
```

Student.properties

```
Rajat = Java
Ankit = Spring
Saurabh = .Net
Ravi = Hibernate
Nitin = Struts
```

Output -



```
C:\WINDOWS\system32\cmd.exe
07/07/2010 10:23 AM 1,800 EmphSDemo.class
07/07/2010 10:21 AM 1,107 EmphSDemo.java
07/07/2010 10:54 AM 1,137 EmphSDemo.java
07/12/2010 08:17 PM 2,238 NunFormatter.class
07/12/2010 08:16 PM 1,509 NunFormatter.java
07/15/2010 10:32 AM 677 PropReader.java
07/15/2010 10:33 AM 78 Student.properties
07/06/2010 10:42 AM 316 Test.class
07/06/2010 10:41 AM 556 Test.java
07/06/2010 11:38 AM 1,038 Test1.class
07/06/2010 11:43 AM 987 Test1.java
18 File(s) 14,546 bytes
2 Dir(s) 11,652,812,800 bytes free

E:\NewJava>javac PropReader.java
E:\NewJava>java PropReader
Following properties are read....
Ankit Spring
Ravi Hibernate
Saurabh .Net
Nitin Struts
Rajat Java
E:\NewJava>_
```

Changes made in the last program **PropReader.java**

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Iterator;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

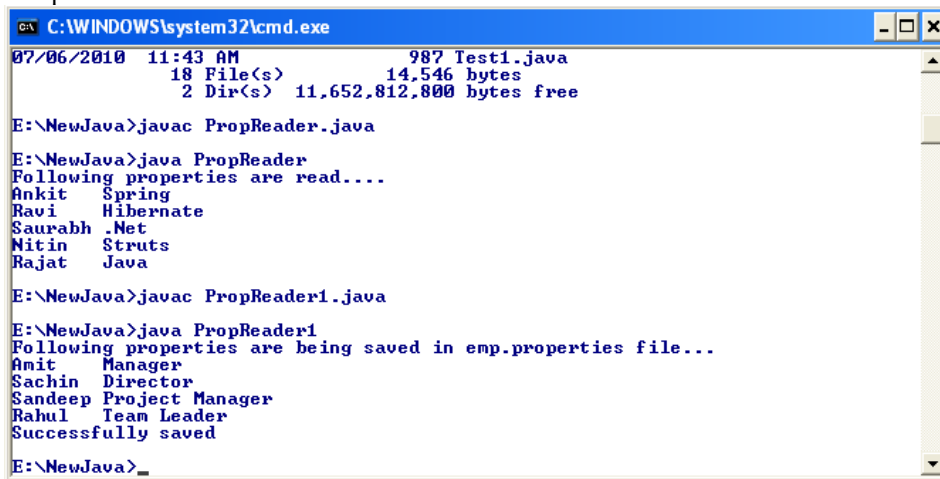
public class PropReader
{
    public static void main(String[] args)
    {
        try
        {
            Properties p = new Properties();
            p.setProperty("Amit", "Manager");
```

## J2EE Notes (By Neeraj Sir)

```
p.put("Rahul", "Team Leader");
p.put("Sandeep", "Project Manager");
p.put("Sachin", "Director");

Set s = p.entrySet();
Iterator itr = s.iterator();
System.out.println("Following properties are being saved in
emp.properties file...");
while(itr.hasNext())
{
    Map.Entry m = (Map.Entry) itr.next();
    System.out.println(m.getKey() + "\t" + m.getValue());
}
FileOutputStream out = new FileOutputStream("emp.properties");
p.store(out, "Emp data");
out.close();
System.out.println("Successfully saved");
}
catch(Exception e)
{
    System.out.println(e);
}
}
```

Output -



```
C:\WINDOWS\system32\cmd.exe
07/06/2010 11:43 AM          987 Test1.java
          18 File(s)      14,546 bytes
          2 Dir(s)  11,652,812,800 bytes free

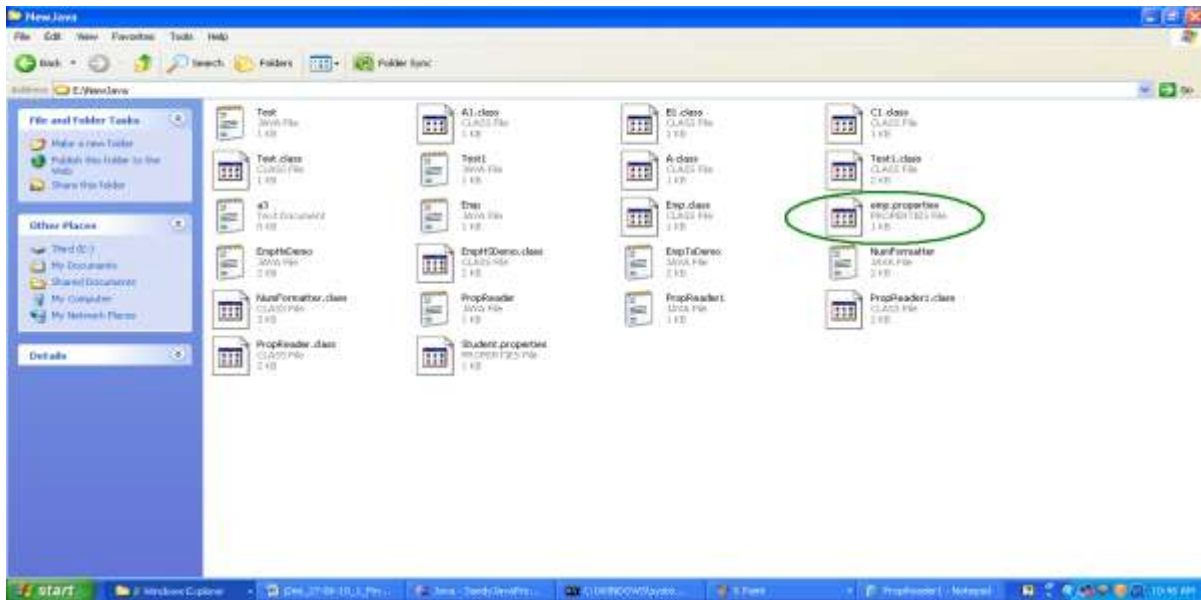
E:\NewJava>javac PropReader.java

E:\NewJava>java PropReader
Following properties are read....
Ankit    Spring
Ravi     Hibernate
Saurabh  .Net
Mitin    Struts
Rajat    Java

E:\NewJava>javac PropReader1.java

E:\NewJava>java PropReader1
Following properties are being saved in emp.properties file...
Amit      Manager
Sachin    Director
Sandeep   Project Manager
Rahul     Team Leader
Successfully saved

E:\NewJava>_
```



**GENERICS** is the facility of defining classes' data members & methods using a generic type whose value is specified at the time of usage.

Generics introduce following advantages over non-generic classes-

1. Type Safety
2. Explicitly type conversions aren't required.

Syntax of defining a Generic class:-

```
class Identifier <Generic Type>
{
    Data Members & method
    definition using Generic type
}
```

Syntax of creating object:-

```
className<Type Replacing Generic Type> reference variable = new className <TypeReplacing
GenericType> (Argument if any)
```

Example -

Program **GenericTest.java**

```
package Generics;

public class GenericTest
{
    public static void main(String[] args)
    {
        A<String> strObj = new A<String> ("It is a generic class");
        A <Integer> intObj = new A<Integer> (5);
        strObj.display();
        System.out.println("Value of Integer instance....");
        intObj.display();
    }
}
```

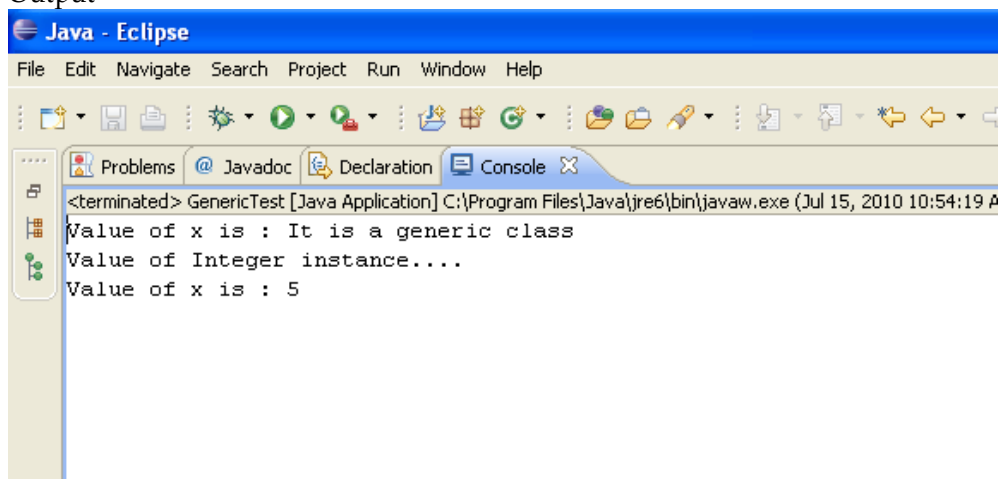


```
}  
}
```

Program A.java

```
package Generics;  
  
class A<T>  
{  
    T x;  
  
    public A(T a)  
    {  
        x=a;  
    }  
  
    public void display()  
    {  
        System.out.println("Value of x is : " + x);  
    }  
}
```

Output -



Now, Same program written without using generics -

Example -

Program NonGenericTest.java

```
package Generics;  
  
public class NonGenericTest  
{  
    public static void main(String[] args)  
    {  
        B strObj = new B ("It is a non generic class");  
        B intObj = new B(5);  
  
        System.out.println("Value of String instance....");  
        strObj.display();  
        System.out.println("Value of Integer instance....");  
        intObj.display();  
    }  
}
```

Program B.java

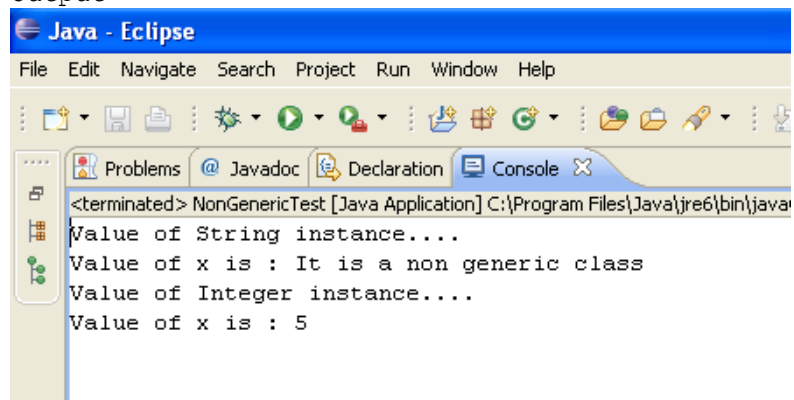
```
package Generics;

class B
{
    Object x;

    public B(Object a)
    {
        x=a;
    }

    public void display()
    {
        System.out.println("Value of x is : " + x);
    }
}
```

Output-



*Same output then why we need Generics when we can get same output by ordinary method?*

*Now let's see –*

*We are doing some changes in the last program(NonGenericTest.java).*

Example –

Program NonGenericTest.java

```

1 package Generics;
2
3 public class NonGenericTest
4 {
5     public static void main(String[] args)
6     {
7         B strObj = new B ("It is a non generic class");
8         B intObj = new B(5);
9
10        System.out.println("Value of String instance....");
11        strObj.display();
12        System.out.println("Value of Integer instance....");
13        intObj.display();
14        intObj = strObj; // Doesn't make sense, must not allowed(Shall compile)
15    }
16 }
17

```

Now, we are doing some changes in the last program(**GenericTest.java**).

```

1 package Generics;
2
3 public class GenericTest
4 {
5     public static void main(String[] args)
6     {
7         A<String> strObj = new A<String> ("It is a generic class");
8         A <Integer> intObj = new A<Integer> (5);
9         strObj.display();
10        System.out.println("Value of Integer instance....");
11        intObj.display();
12        intObj = strObj; //Make sense, must allowed(Shall not compile)
13    }
14 }
15

```

```

1 package Generics;
2
3 public class GenericTest
4 {
5     public static void main(String[] args)
6     {
7         A<String> strObj = new A<String> ("It is a generic class");
8         A <Integer> intObj = new A<Integer> (5);
9         strObj.display();
10        System.out.println("Value of Integer instance....");
11        intObj.display();
12        Type mismatch: cannot convert from A<String> to A<Integer> must allowed(Shall not compile)
13    }
14 }
15

```

**Type safety** is the feature of detecting incompatible assignments at compilation time. In case of Generics each reference variable is given a type at the time of declaration hence, incompatible assignments are detected whereas in case of non-generic classes incompatible assignments are not detected usually when references of parent type are used to hold child objects.

So, Now add **getValue()** method in **B.java**

```

1 package Generics;
2
3 class B
4 {
5     Object x;
6
7     public Object getValue()
8     {
9         return x;
10    }
11
12
13    public B(Object a)
14    {
15        x=a;
16    }
17
18    public void display()
19    {
20        System.out.println("Value of x is : " + x);
21    }
22 }

```

Add the following line in **NonGenericTest.java**

```

1 package Generics;
2
3 public class NonGenericTest
4 {
5     public static void main(String[] args)
6     {
7         B strObj = new B ("It is a non generic class");
8         B intObj = new B(5);
9
10        System.out.println("Value of String instance...");
11        strObj.display();
12        System.out.println("Value of Integer instance...");
13        intObj.display();
14        String s = strObj.getValue(); //make sense must be allowed. (Shall not compile, explicit conversion is required.)
15    }
16 }
17
18 package Generics;
19
20 public class NonGenericTest
21 {
22     public static void main(String[] args)
23     {
24         B strObj = new B ("It is a non generic class");
25         B intObj = new B(5);
26
27        System.out.println("Value of String instance...");
28        strObj.display();
29        System.out.println("Value of Integer instance...");
30        intObj.display();
31        String s = strObj.getValue(); //make sense must be allowed. (Shall not compile, explicit conversion is required.)
32    }
33 }
34

```

Add **getValue()** method in **A.java**

```

1 package Generics;
2
3 class A<T>
4 {
5     T x;
6
7     public A(T a)
8     {
9         x=a;
10    }
11
12    public void display()
13    {
14        System.out.println("Value of x is : " + x);
15    }
16
17    public T getValue()
18    {
19        return x;
20    }
21 }
22

```

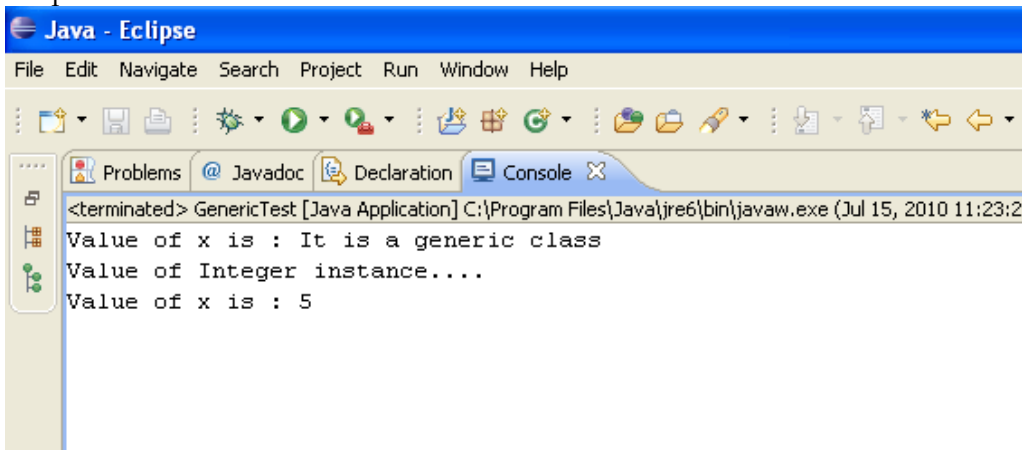
Add this line in GenericTest.java

```

1 package Generics;
2
3 public class GenericTest
4 {
5     public static void main(String[] args)
6     {
7         A<String> strObj = new A<String> ("It is a generic class");
8         A<Integer> intObj = new A<Integer> (5);
9         strObj.display();
10        System.out.println("Value of Integer instance....");
11        intObj.display();
12        String s = strObj.getValue(); // make sense, must be allowed(Shall compile)
13    }
14 }
15

```

Output -



```

Java - Eclipse
File Edit Navigate Search Project Run Window Help

<terminated> GenericTest [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jul 15, 2010 11:23:2
Value of x is : It is a generic class
Value of Integer instance....
Value of x is : 5

```

```
package GenericsDemo;

class One
{
    public void display()
    {
        System.out.println("In one..");
    }
}

class Two extends One
{
    public void display()
    {
        System.out.println("In Two..");
    }
}

class Three extends One
{
    public void display()
    {
        System.out.println("In Three..");
    }
}

class GenericForOneFamily<T>
{
    T x;

    public GenericForOneFamily(T t)
    {
        x=t;
    }

    public void describe()
    {
        x.display();
    }
}

class FamilyTest
{
    public static void main(String args[])
    {
        GenericForOneFamily <One> oneObj = new GenericForOneFamily<One> (new One());
        GenericForOneFamily <Two> twoObj = new GenericForOneFamily<Two> (new Two());
        GenericForOneFamily <Three> threeObj = new GenericForOneFamily<Three> (new
        Three());
        oneObj.describe();
        twoObj.describe();
        threeObj.describe();
    }
}
```

## J2EE Notes (By Neeraj Sir)

}  
This program will not compile. Why? To get the reason see the figure below.

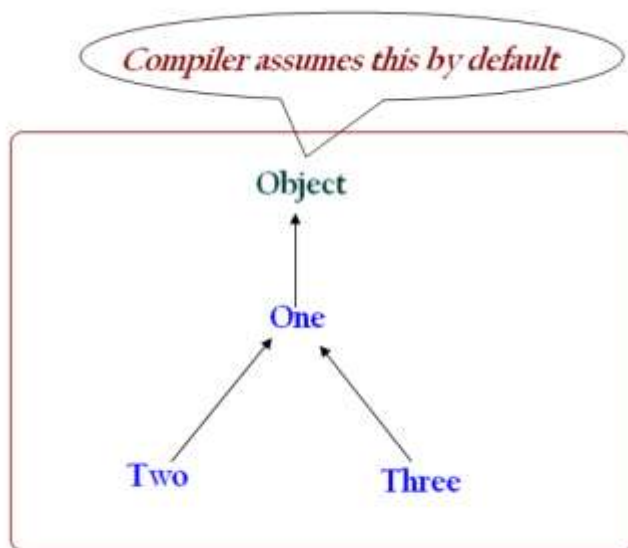
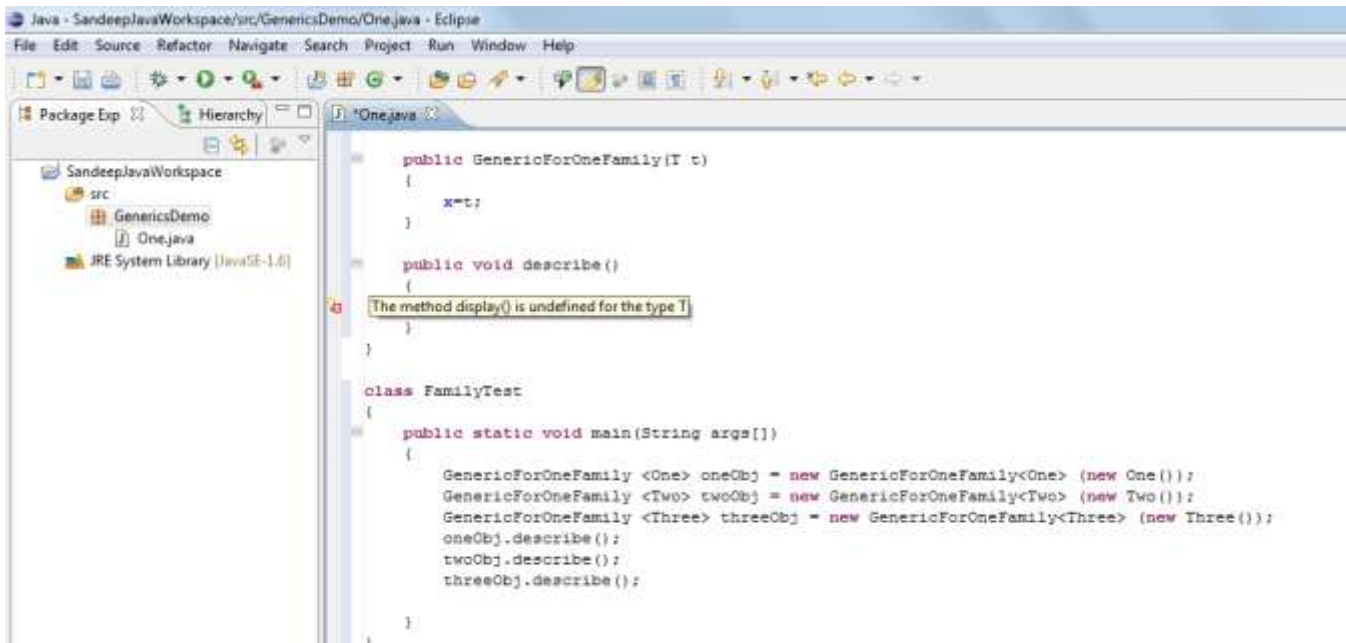
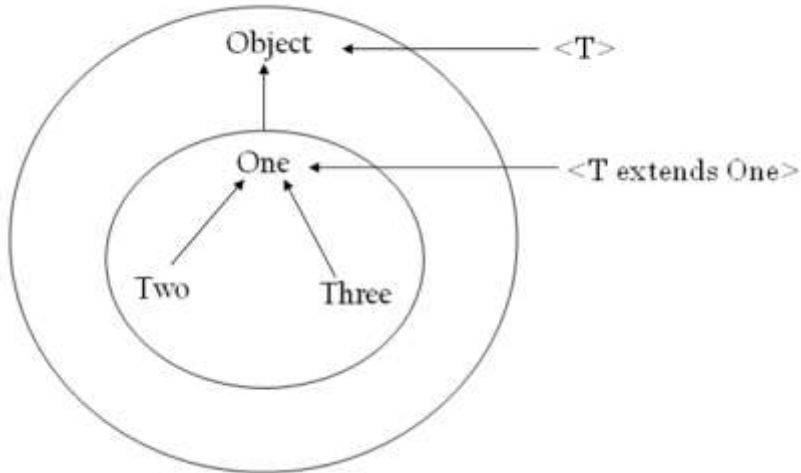


Figure given below is the actual working internally.



Program **FamilyTest.java**

```
package GenericsDemo;
```

```
class One
{
    public void display()
    {
        System.out.println("In one..");
    }
}
```

```
class Two extends One
{
    public void display()
    {
        System.out.println("In Two..");
    }
}
```

```
class Three extends One
{
    public void display()
    {
        System.out.println("In Three..");
    }
}
```

```
class GenericForOneFamily<T extends One>
{
    T x;

    public GenericForOneFamily(T t)
    {
        x=t;
    }

    public void describe()
    {
```



## J2EE Notes (By Neeraj Sir)

```
        x.display();
    }
}

class FamilyTest
{
    public static void main(String args[])
    {
        GenericForOneFamily <One> oneObj = new GenericForOneFamily<One> (new One());
        GenericForOneFamily <Two> twoObj = new GenericForOneFamily<Two> (new Two());
        GenericForOneFamily <Three> threeObj = new GenericForOneFamily<Three> (new
Three());
        oneObj.describe();
        twoObj.describe();
        threeObj.describe();
    }
}
```

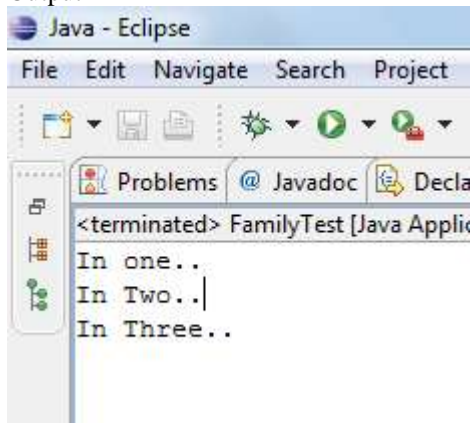
```
class Three extends One
{
    public void display()
    {
        System.out.println("In Three..");
    }
}

class GenericForOneFamily<T extends One>
{
    T x;

    public GenericForOneFamily(T t)
    {
        x=t;
    }
}
```

← changes made here

Output-



### NOTE:

By default a generic type is assumed to be **Object** by the compiler i.e. in the generic class using the generic type only features of **Object** class can be referred. To specify a **non-Object** upper limit for a generic type following syntax is used:-

<GenericType extends SuperType>

-----X-----

*Q. Convert all simple collection programs into Generic one:*

**HINT :** how to make changes in converting all programs into Generics:

*Changes should be done in all the previous solved programs in the following lines indicated below -*

`ArrayList list = new ArrayList();` ☒

`ArrayList<String> list = new ArrayList<String> ();` ☒

`Iterator itr = list.iterator();` ☒

`Iterator<String> itr = list.iterator ();` ☒

`while (itr.hasNext ())`

`{`

`String s = (String) itr.next();` ☒

`String s = itr.next ();` ☒

`-----`

`-----`

`}`

**End Of Collection**

## INTERNATIONALIZATION

**I18N:** Internationalization

**L10N:** Localization

**Internationalization** is the process of developing such applications which can adapt themselves according to the preferences of users of different countries.

Usually following preferences change from one country to another:-

1. Language
2. Number & currency format.
3. Date and time format.

### **1) public Locale(String LangCode);**

Two alphabet codes are used to represent languages:-

<u>Language</u>	<u>Code</u>
Hindi	hi
English	en
French	fr
Arabic	ar
etc.	

**2) public Locale(String langCode, String countryCode);**

Two alphabet codes are used to represent countries:-

<u>Country</u>	<u>Code</u>
India	IN
USA	US
Britain	UK
France	FR
etc.	

**3) public Locale (String langCode, String countryCode, String variant);**

Commonly used methods of class:-

1. **getAvailableLocales()** is used to find out the Locales supported by the native operating system.

**public static Locale[] getAvailableLocales();**

2. **getLanguage()** returns the language code represented by the Locale object.

**public String getLanguage();**

3. **getDisplayLanguage()** returns the language name represented by the Locale object.

**public String getDisplayLanguage();**

4. **getCountry()** returns country code.

**public String getCountry();**

5. **getDisplayCountry()** returns country name

**public String getDisplayCountry();**

6. **getDisplayName()** returns language and country name for the Locale object.

**public String getDisplayName();**

Program LocaleDemo.java

```
package Internationalization;

import java.util.Locale;

public class LocaleDemo
{
    public static void main(String[] args)
    {
        System.out.println("Obtaining locales from the O/S.....");
        Locale[] locales = Locale.getAvailableLocales();
        System.out.println("Following are supported by the O/S...");
        for(int i=0; i<locales.length; i++)
        {
            System.out.println(locales[i]);
            //System.out.println(locales[i].getDisplayLanguage());
            //System.out.println(locales[i].getCountry());
        }
    }
}
```

```
        //System.out.println(locales[i].getDisplayLanguage());
        //System.out.println(locales[i].getCountry());
    }
}
```

In Java class library, there are various locale sensitive classes. A locale sensitive its behaviour according to the given locale.

**java.text.NumberFormat** & **java.Text.DateFormat** are two locale sensitive classes which are used to apply Locale specific formatting on numbers, currency and date/time respectively.

#### 1. **java.text.NumberFormat**

- a. **getNumberInstance()** method of NumberFormat class is used to obtain a NumberFormat object to apply locale specific number formatting on a number.

**public static NumberFormat getNumberInstance(Locale l);**

- b. **getCurrencyInstance()** method is used to obtain a NumberFormat object for applying locale specific currency formatting on a number.

**public static NumberFormat getCurrencyInstance(Locale l);**

- c. **format()** method of NumberFormat class is used to apply locale specific number or currency formatting.

**public String format(Long number);**  
**public String format(Double number);**

```
package Internationalization;

import java.util.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import java.text.*;

class NumFormatter extends JFrame implements ActionListener
{
    JTextField txt1, txt2, txt3, txt4;
    JButton btn1, btn2;

    public NumFormatter (String title)
    {
        super(title);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.add(new JLabel("Language:"));
        c.add(txt1 = new JTextField(15));
        c.add(new JLabel("Country:"));
        c.add(txt2 = new JTextField(15));
        c.add(new JLabel("Number:"));
        c.add(txt3 = new JTextField(15));
        c.add(new JLabel("Formatted:"));
    }
}
```

## J2EE Notes (By Neeraj Sir)

```
c.add(txt4 = new JTextField(15));

c.add(btn1 = new JButton("Number Format"));
c.add(btn2 = new JButton("Number Format"));

btn1.addActionListener(this);
btn2.addActionListener(this);

txt4.setEditable(false);
setSize(250, 350);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void actionPerformed (ActionEvent e)
{
    String l = txt1.getText();
    String c = txt2.getText();

    double d = Double.parseDouble(txt3.getText());
    Locale locale = new Locale(l, c);
    NumberFormat nf;
    if(e.getSource()==btn1)
        nf = NumberFormat.getNumberInstance(locale);
    else
        nf = NumberFormat.getCurrencyInstance(locale);
    txt4.setText(nf.format(d));
}

public static void main(String[] args)
{
    new NumFormatter("Number & Currency Formatter");
}
}
```

**DateFormat** class is used to apply locale specific date & time formatting.

- a. **getDateInstance()** method is used to obtain a **DateFormat** object that is used to apply date format for the given locale.

**public static DateFormat getDateInstance(int style, Locale locale);**

*to specify style following static final int constants are used:-*

```
DateFormat.SHORT
DateFormat.LONG
DateFormat.DEFAULT
etc.
```

- b. **getTimeInstance()** method is used to obtain a **DateFormat** object that can be used to apply time formatting for the given locale.

**public static DateFormat getTimeInstance(int style, Locale locale);**

- c. **format()** method is used to apply locale specific date or time formatting.

**public String format(Date d);**

## J2EE Notes (By Neeraj Sir)

### *Example -*

```
package Internationalization;

import java.util.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import java.text.*;

class DateFormatter extends JFrame implements ActionListener
{
    JTextField txt1, txt2, txt3, txt4;
    JButton btn;

    public DateFormatter (String title)
    {
        super(title);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.add(new JLabel("Language:"));
        c.add(txt1 = new JTextField(15));
        c.add(new JLabel("Country:"));
        c.add(txt2 = new JTextField(15));
        c.add(new JLabel("Date:"));
        c.add(txt3 = new JTextField(15));
        c.add(new JLabel("Formatted:"));
        c.add(txt4 = new JTextField(15));

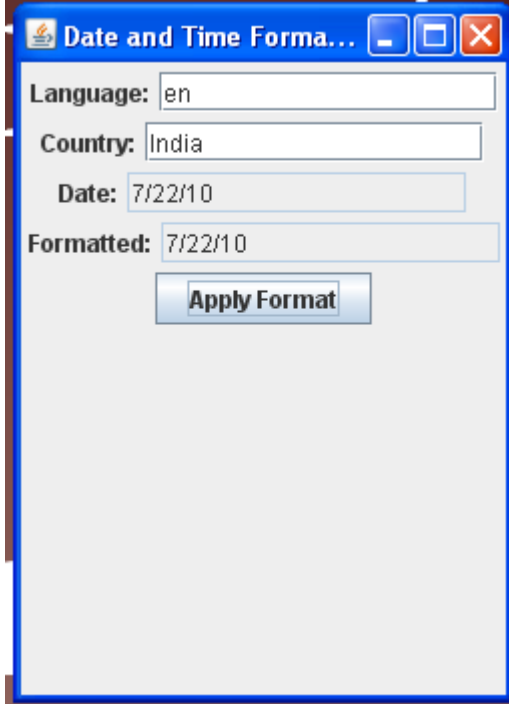
        c.add(btn = new JButton("Apply Format"));
        btn.addActionListener(this);
        txt3.setEditable(false);
        txt4.setEditable(false);
        setSize(250, 350);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed (ActionEvent e)
    {
        String l = txt1.getText();
        String c = txt2.getText();
        Date d = new Date();
        Locale locale = new Locale(l, c);
        DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT, locale);
        DateFormat tf = DateFormat.getTimeInstance(DateFormat.SHORT, locale);
        txt3.setText(df.format(d));
        txt4.setText(tf.format(d));
    }

    public static void main(String[] args)
    {
        new DateFormatter("Date and Time Formatter");
    }
}
```

}

Output –



**Dated : 17.07.10**

*To internationalize textual contents following steps are required:-*

1. Create properties files for each locale to be supported by the application.  
Each property files have contents, locale sensitive text in the form of key value pair. Keys of all property files remain same.
2. Use **java.util.ResourceBundle** class to automatically load property file according to the given locale.  
**ResourceBundle** is a utility class that manages properties objects for the application. In order to use the **ResourceBundle** properties files must be named according to the following convention.

**bundleName\_langCode.properties**

or

**bundleName\_langCode\_CountryCode.properties**

*Example –*

Let label, captions and titles of a GUI application are to be internationalized. Only langcode is used to represent user preference.

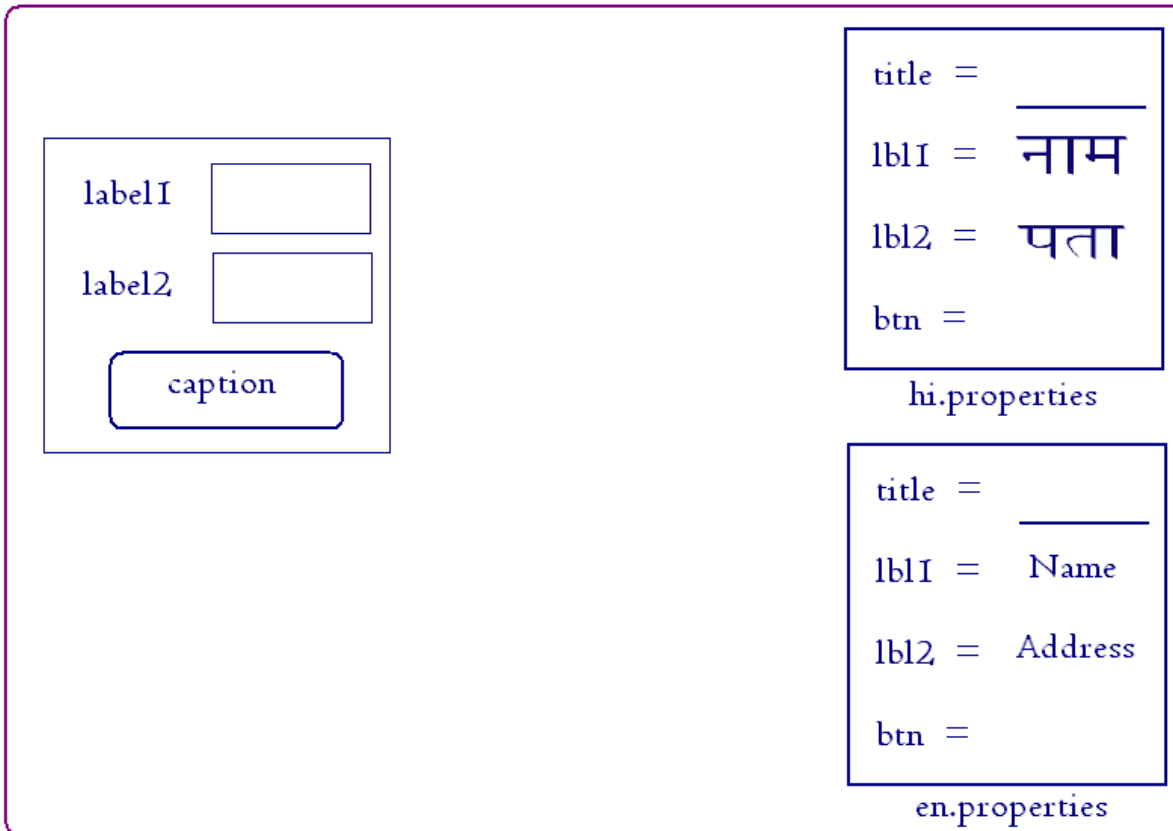
If Hindi, English, French and Arabic languages are to be supported then properties file can be Named as:-

**MyBundle\_en.properties**

**MyBundle\_hi.properties**

**MyBundle\_ar.properties**

**etc.**



- 1) **getBundle()** method of **ResourceBundle** class is used to load a locale properties file.

*Syntax*

**public static ResourceBundle getBundle(String baseNameOfPropertiesFile, Locale locale);**

- 2) **getString()** method of **ResourceBundle** class is used to obtain the value of a property.

*Syntax*

**public String getString(String key);**

*Example –*



Type this whole matter in the notepad and save the file as **"MyBundle\_en.properties"**

title = Registration Form  
lbl1 = Name  
lbl2 = Address  
btn = Register

"MyBundle\_en.properties"

Type this whole matter in the notepad and save the file as **"MyBundle\_hi.properties"**

title = **Panjikaran karen**  
lbl1 = **Naam**  
lbl2 = **Pata**  
btn = **Panjikarat Karen**

"MyBundle\_hi.properties"

*Program* **I18nDemo.java**

```
package Internationalization;

import java.awt.Container;
import java.awt.FlowLayout;
import java.util.Locale;
import java.util.ResourceBundle;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

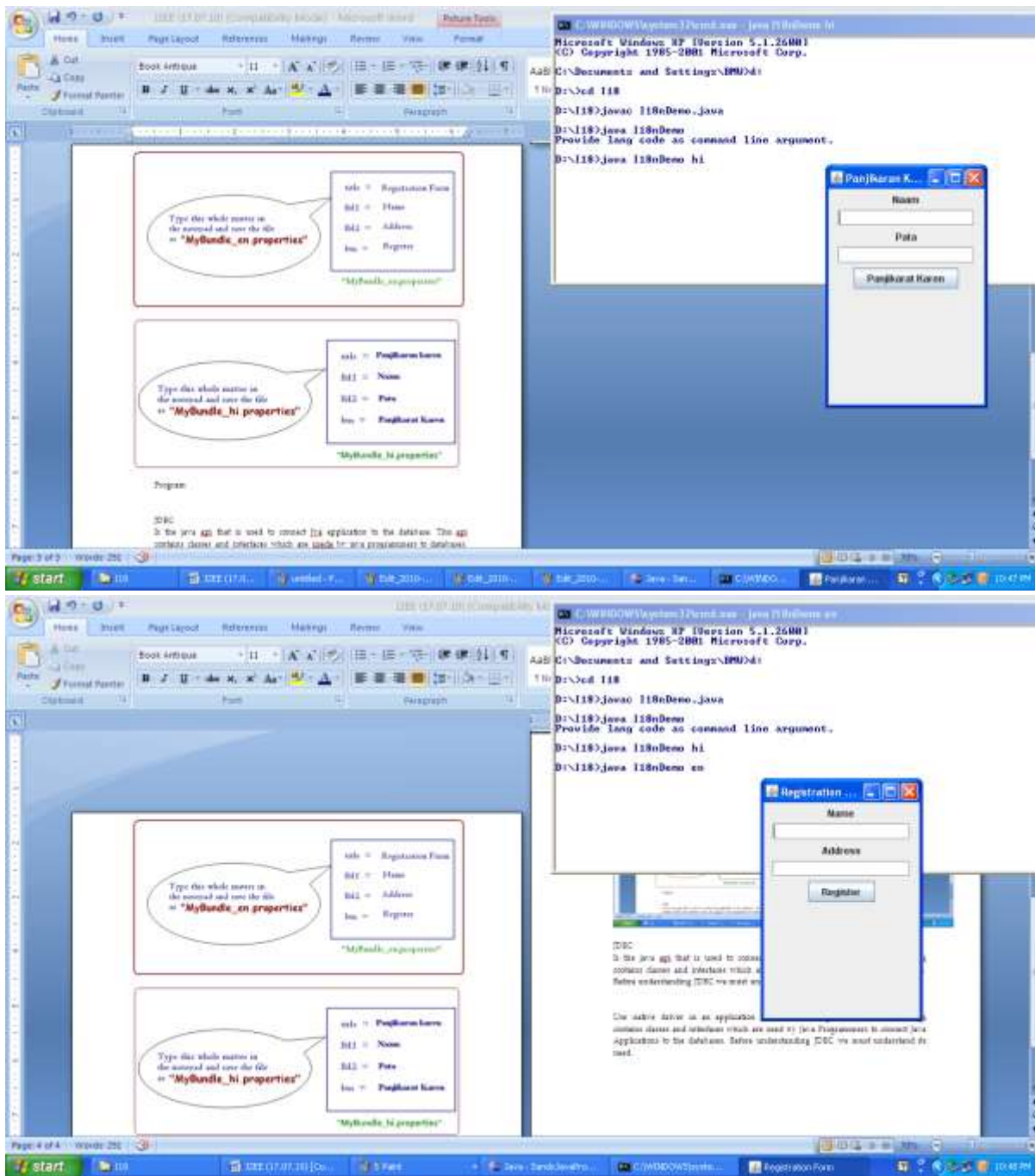
class I18nDemo extends JFrame
{
    JTextField txt1, txt2;
    JLabel lbl1, lbl2;
```

## J2EE Notes (By Neeraj Sir)

```
JButton btn;
static Locale locale;

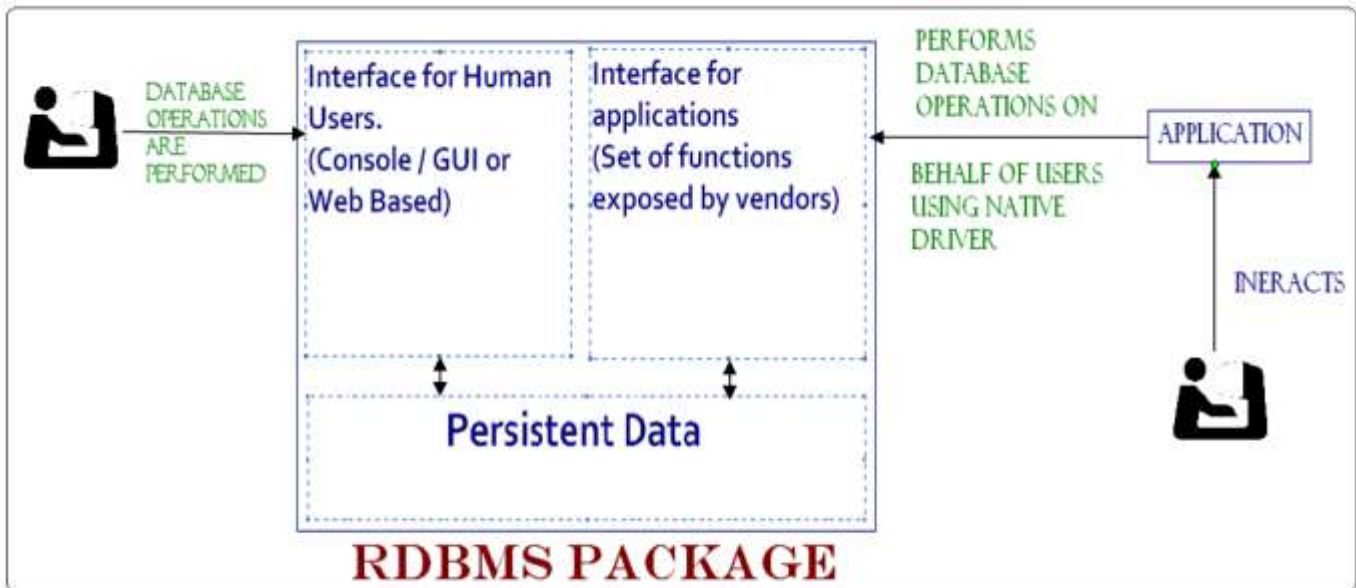
public static void main(String[] args)
{
    if (args.length==0)
        System.out.println("Provide lang code as command line argument.");
    else
    {
        locale = new Locale(args[0]);
        new I18nDemo();
    }
}

public I18nDemo()
{
    ResourceBundle rb = ResourceBundle.getBundle("MyBundle", locale);
    lbl1 = new JLabel(rb.getString("lbl1"));
    lbl2 = new JLabel(rb.getString("lbl2"));
    btn = new JButton(rb.getString("btn"));
    Container c = getContentPane();
    c.setLayout(new FlowLayout());
    c.add(lbl1);
    c.add(new JTextField(15));
    c.add(lbl2);
    c.add(new JTextField(15));
    c.add(btn);
    setSize(200, 300);
    setTitle(rb.getString("title"));
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
}
}
```



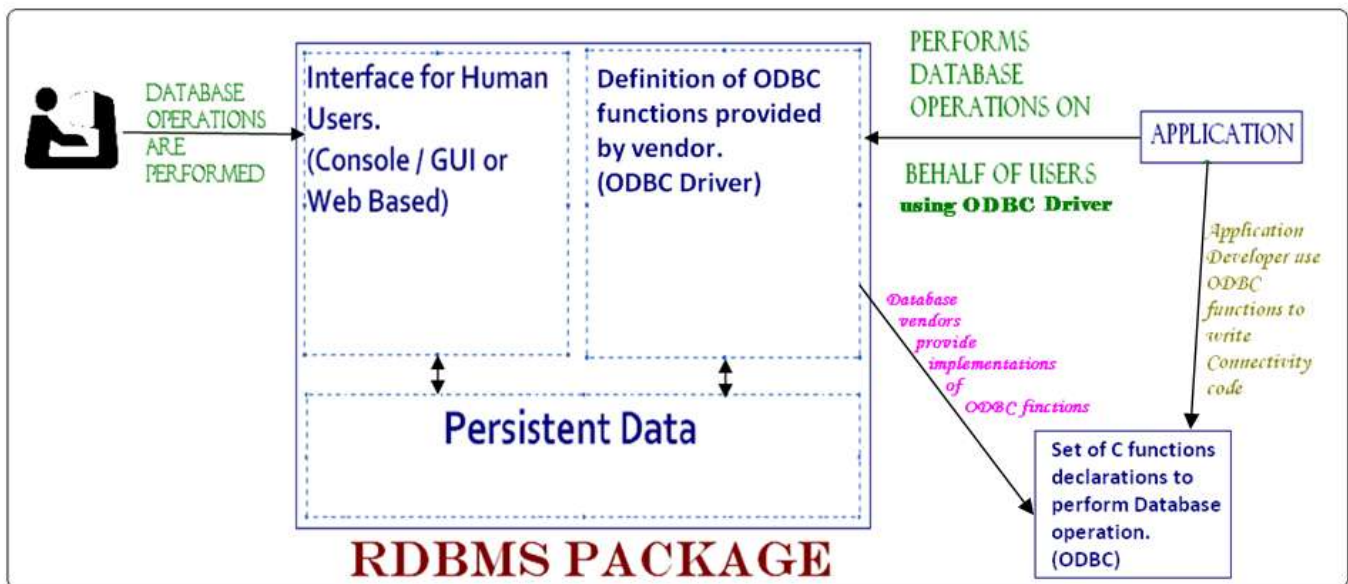
## JDBC

Is the java API that is used to connect Java Application to the database. This API contains classes and interfaces which are used by Java Programmers to connect Java Applications to the databases. Before understanding JDBC we must understand its need.

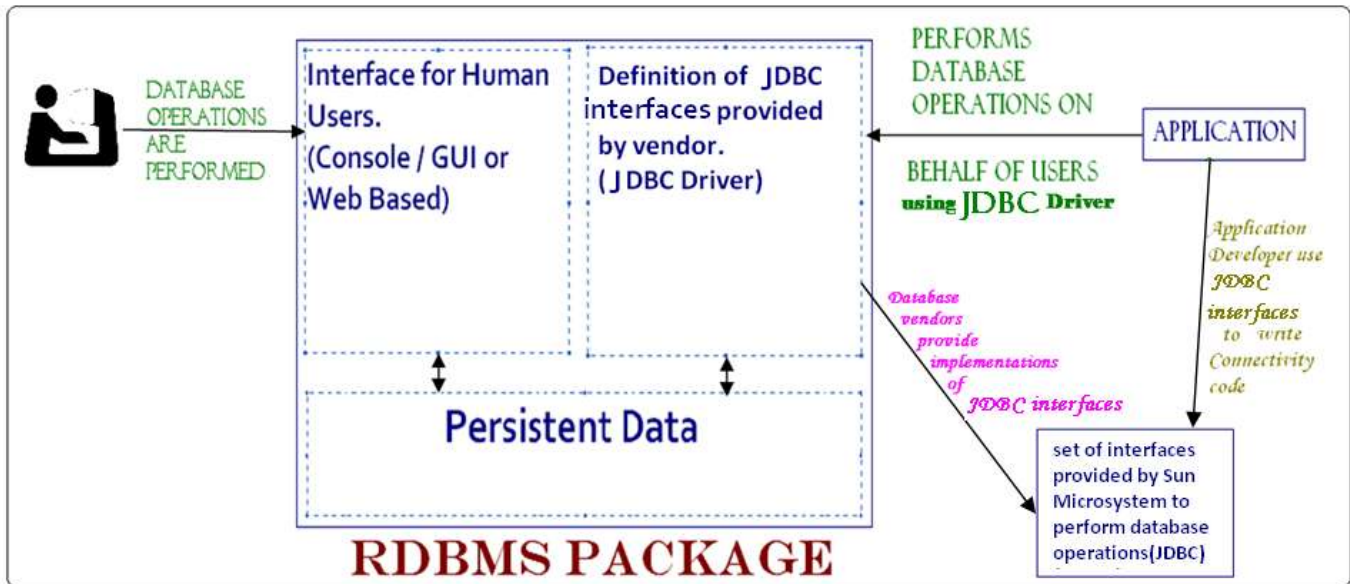


- Using native driver in an application for communicating to a database had following problems:-
1. Application Programmers were to learn different API's for different database packages.
  2. Each time database package was changed in an application was required to be modified.

ODBC was introduced to solve above mentioned problems.

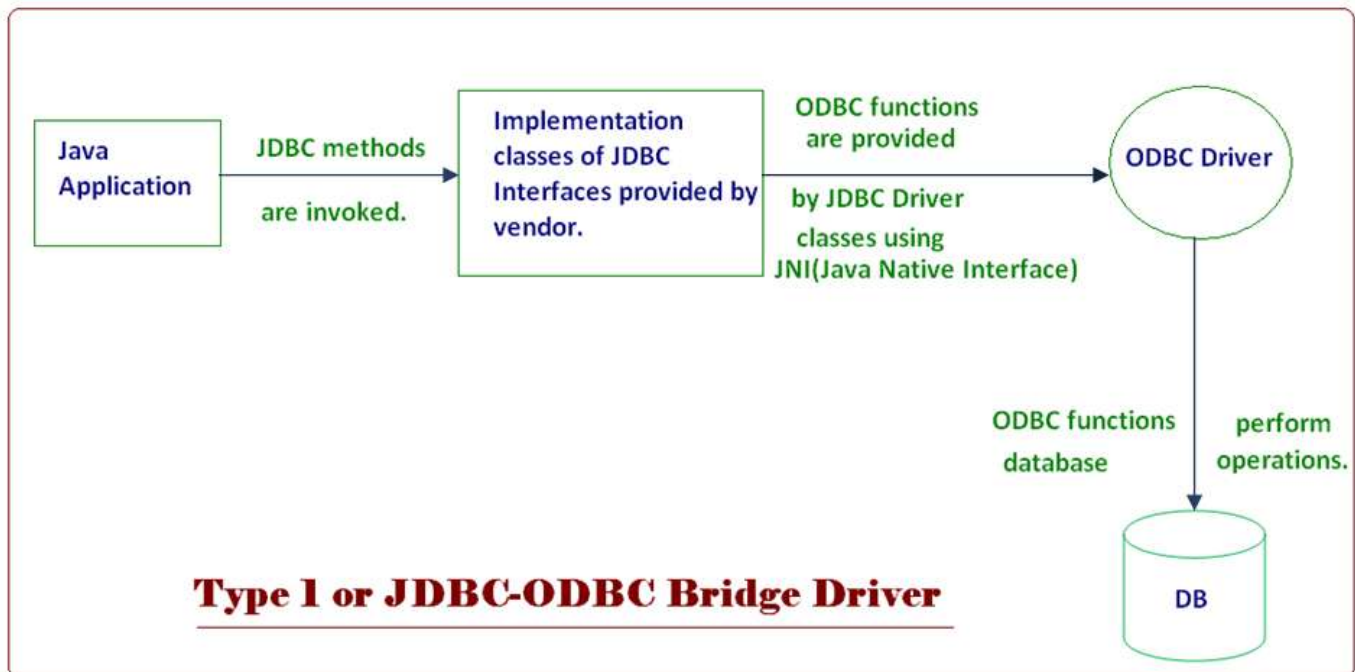


**Disadvantage of using ODBC** was that Application Programmers need to invoke C functions from their applications.



Different database vendors implemented JDBC interfaces in different ways. Depending upon the implementation we have 4 types of JDBC drivers.

1. **Type 1 or JBC-ODBC Bridge Driver** - In Type 1 JDBC driver, driver classes provided by database vendors invokes ODBC functions using JNI.



**Advantage of Type 1 driver -**

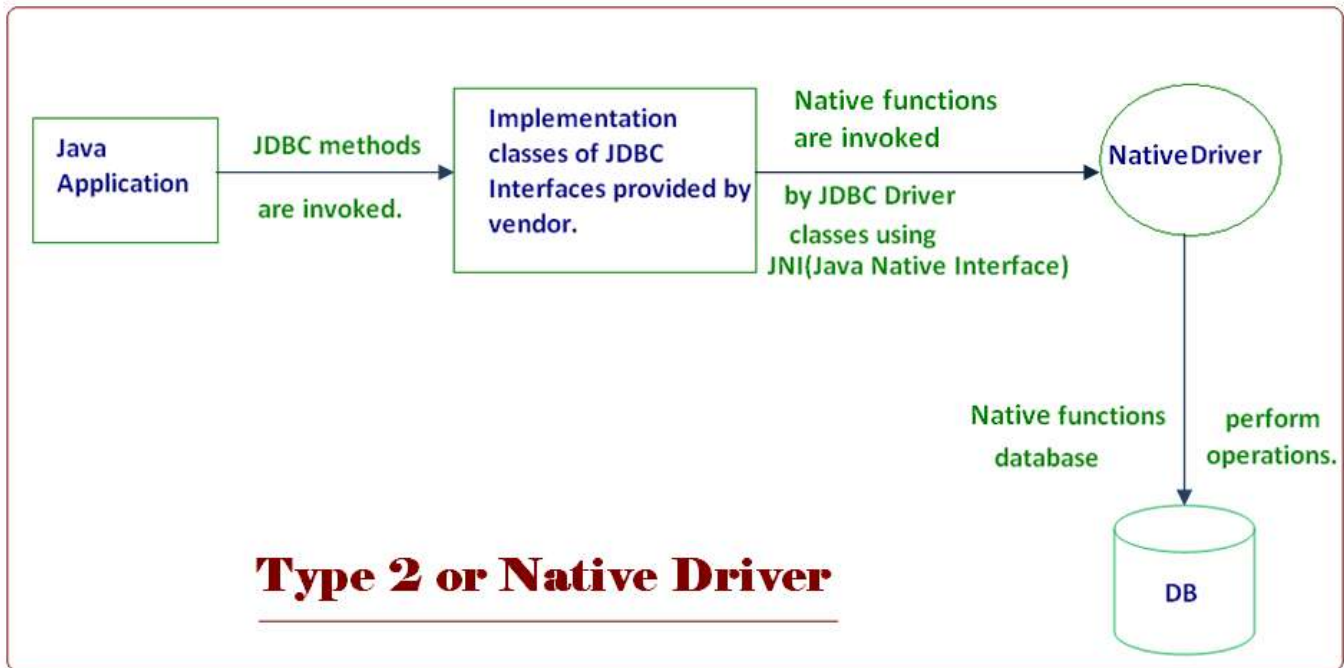
- a. This is the simplest driver from implementation Sun Microsystems provides implementation of Type 1 JDBC driver with Core Java library.
- b. Single implementation of Type1 driver can be used with all databases.



Disadvantage of Type 1 driver -

- a. Major disadvantage of Type1 JDBC Driver is the degradation of performance because each database operation requires multiple calls and conversion.
- b. ODBC Driver needs to be installed on each machine on which application is to be executed.

**2. Type 2 or Native Driver**



In Type 2 Driver, driver classes provided by Vendor act as Java Wrapper of Native Driver.

Advantage of Type 2 driver -

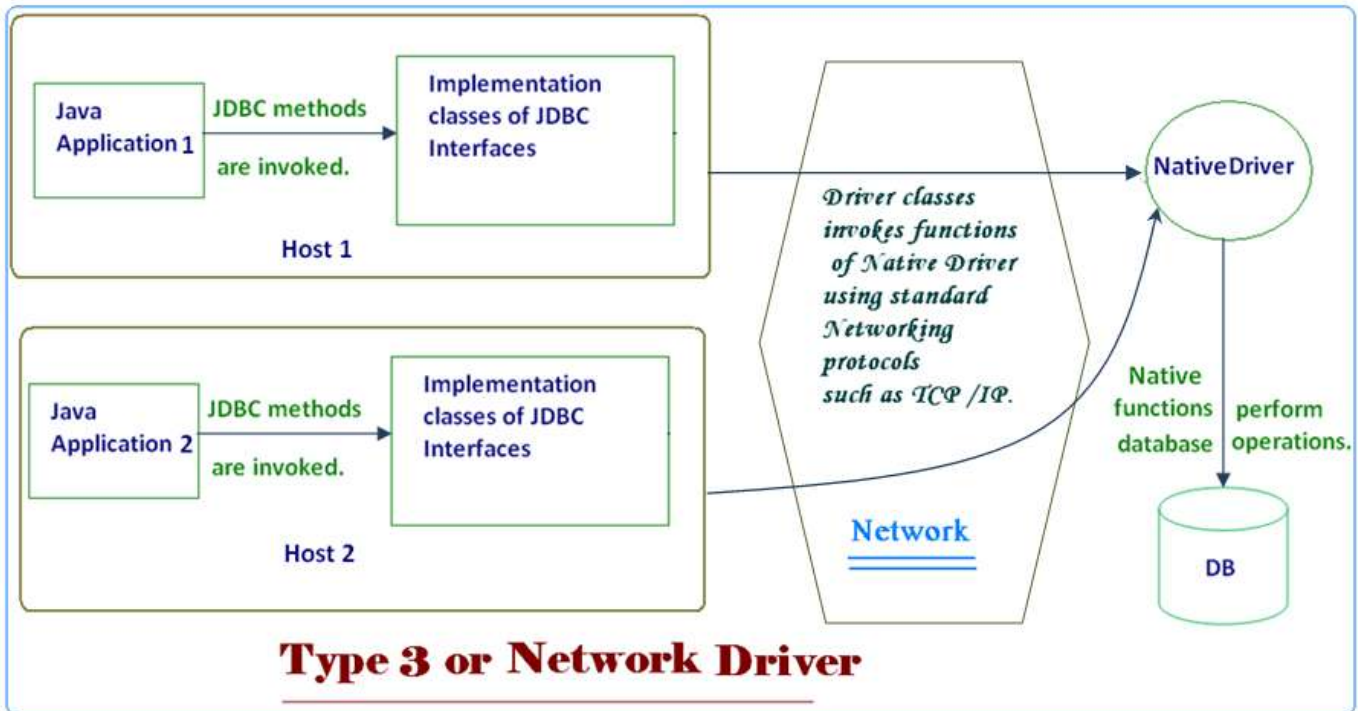
- a. ODBC driver is not required.
- b. Better performance is obtained as compared to Type 1 Driver.

Disadvantage of Type 2 driver -

- a. Native Driver needs to be installed on each machine on which application is to be executed.
- b. For each database different driver implementation is required.

**3. Type 3 or Network Driver.**

- a.



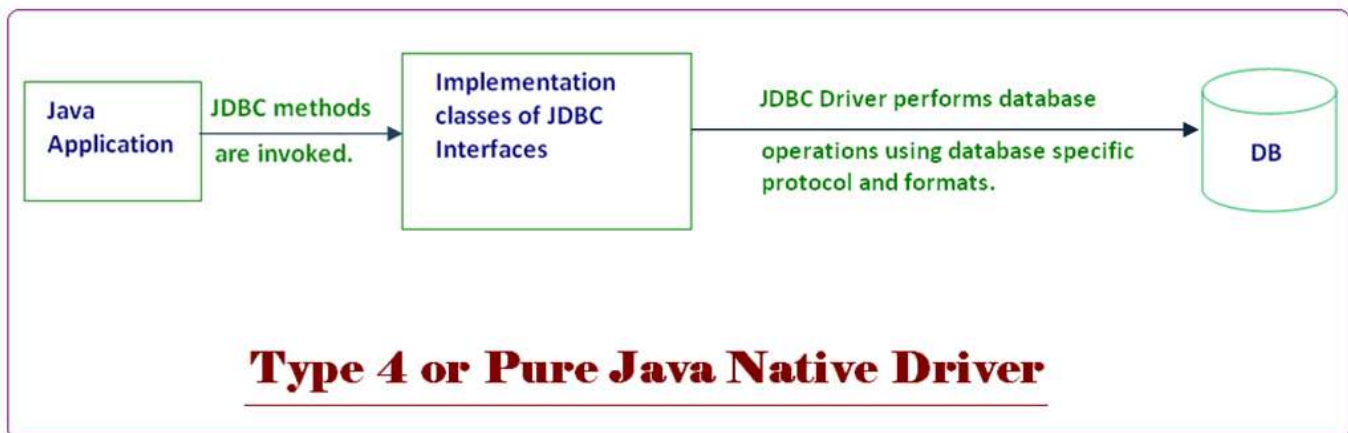
Advantage of Type 3 driver -

Native Driver need to be installed only on a single machine on the network.

Disadvantage of Type 3 driver -

Performance is degraded because of additional networking overhead.

**4. Type 4 or Pure Java Native Driver.**



Advantage of Type 4 driver -

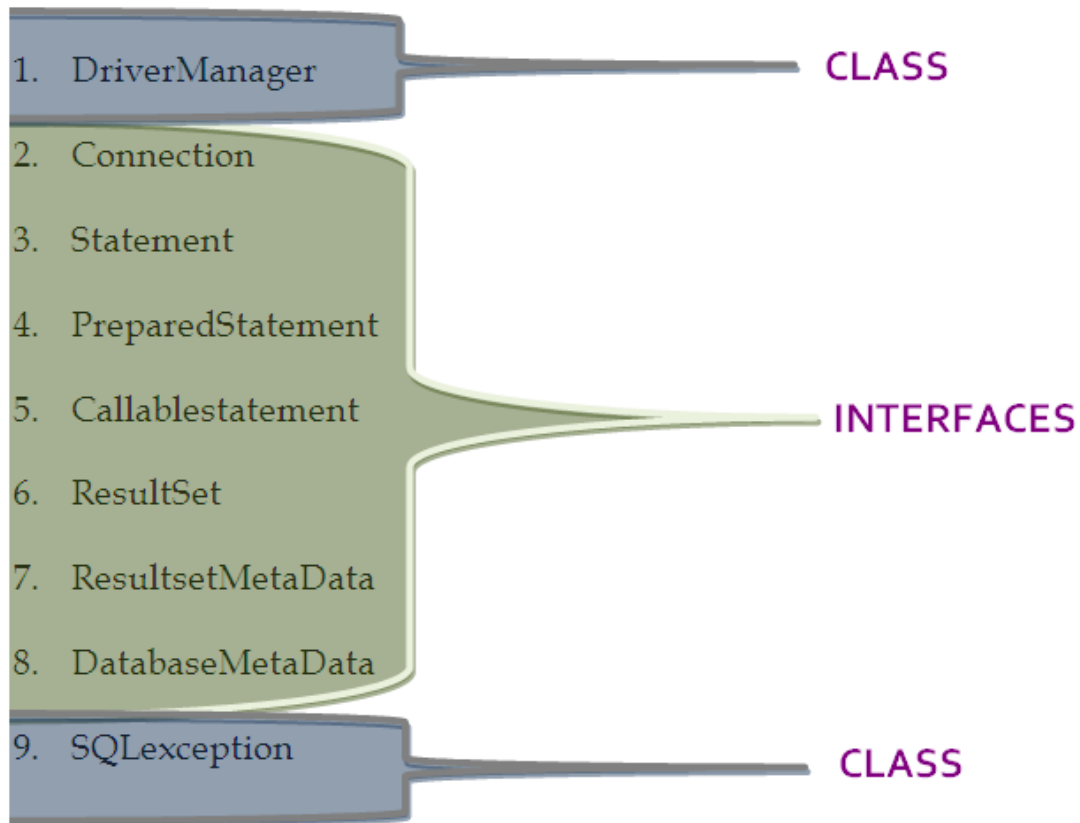
- a. ODBC & Native Driver is not required.
- b. Better performance is obtained as compared to other drivers.

Disadvantage of Type 4 driver -

For each database different implementation of the driver is required.

java.Sql package contains classes & interfaces of JDBC API.

### COMMONLY USED CLASSES & INTERFACES ARE -



- + **DriverManager** is a utility class that acts as a factory of connections.
- + **Connection Interface** provides the abstraction of a database connection and act as a factory of statements.
- + **Statement** provides the facility of executing sql queries and act as a factory of Resultset.
- + **PreparedStatement** provides the facility of executing parameterized query.
- + **CallableStatement** provides the facility of invoking stored procedures & functions.
- + **ResultSet** is used to store the result of a **SELECT QUERY** & act as a factory of Resultset Metadata.
- + **ResultSetMetaData** provides the facility of obtaining information about the result contained in ResultSet.
- + **DatabaseMetaData** is used to obtain information about the database.
- + **SQLException** is the superclass of all database related exception.

**Factory** is a creational designed pattern that is used to control the creation of objects. This design pattern is implemented in the form of a factory class.

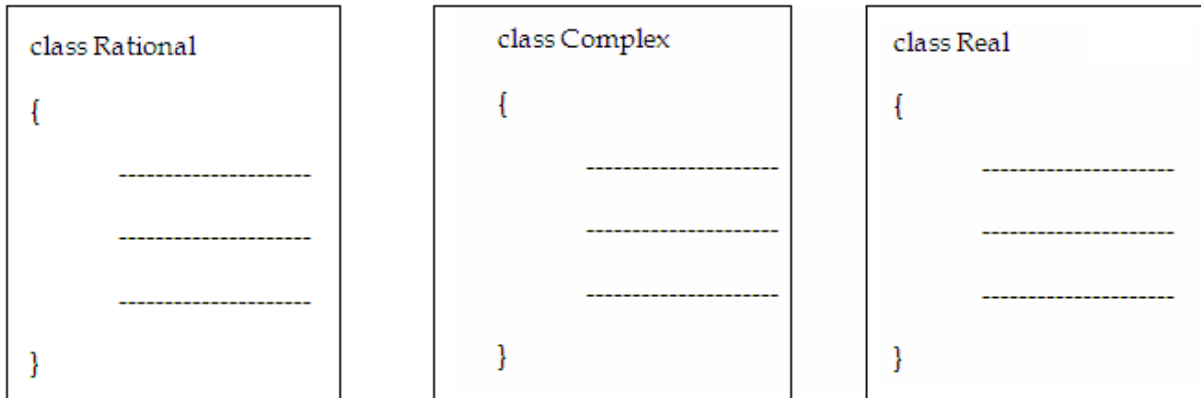
A factory class is a class that contains factory methods.

A factory method is a method that creates & returns objects.



For example –

*Define a class that creates & uses objects of any of these classes. User class must not be modified to switch form one type of Number to Another.*



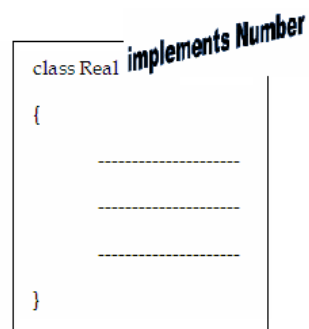
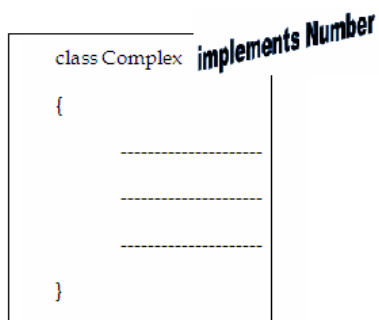
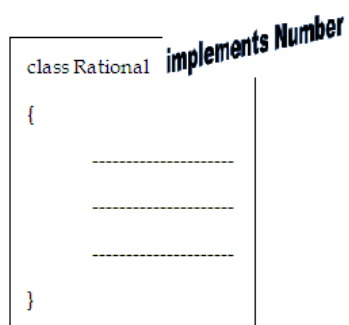
Now, one way we have –

We will create an interface “**Number.java**”.

```
interface Number
{
    void display();
    Number add (Number);
    -----
    -----
}
```

class User

```
{
    public static void main(String args[])
    {
        Rational r1 = new Rational(5, 4); // traditional method of creating objects
        Number r1 = NumberFactory.getNumber(i); // creation of objects using factory methods
        Rational r2 = new Rational(6, 7); // traditional method of creating objects
        Number r2 = NumberFactory.getNumber(i); // creation of objects using factory methods
    }
}
```



```
class NumberFactory
{
    public static Number getNumber(int type)
    {
        if(type == 1)
            return new Rational();
        else if (type == 2)
            return new Complex();
        else
            return new Real();
    }
}
```

Now, let's suppose we assume 2 different companies **Microsoft** and **Oracle** provides the implementation for Connection class as follows:-

```
class SQLConnection implements Connection
{
    public Statement createStatement()
    {
        return new SQLStatement();
    }
    -----
    -----
    -----
}
```

**Microsoft's Implementation**

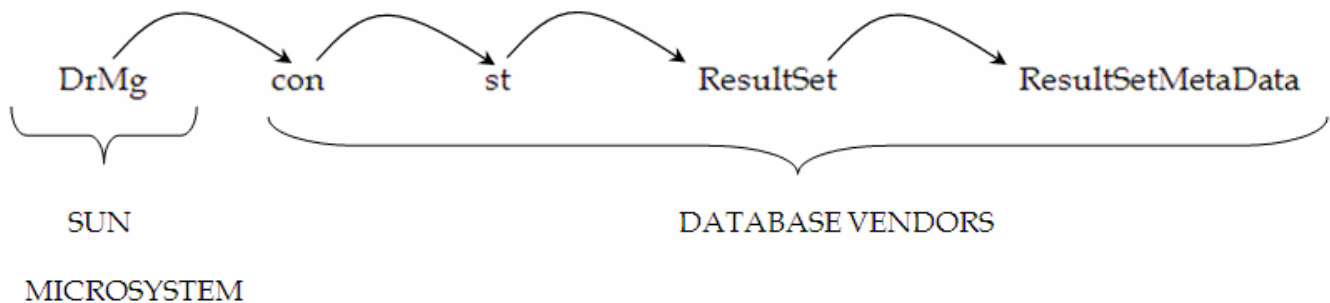
```
class OracleConnection implements Connection
{
    public Statement createStatement()
    {
        return new OracleStatement();
    }
    -----
    -----
    -----
}
```

**Oracle's implementation**

Let *con* be a connection Object.

Statement can be created in implementation independent manner as:-

*Statement stmt = con.createStatement();*



Following steps are required to connect a Java application to a database:-

1. Driver class is registered with the **DriverManager**.
2. Using the **DriverManager**, **Connection** object is created.
3. Using the **Connection** object, **Statement** object is created.
4. Using the statement queries executed.
5. **Connection** is closed.

Implementation of steps:-

Each driver implementation provides a **MetaData** class that describes connection implementation for the driver. This class needs to be registered with the DriverManager.

All Driver classes contain registration code in their static block i.e. in order to register the driver class it simply needs to be loaded.

In case of **Type 1 driver sun.jdbc.odbc.JdbcOdbcDriver** class need to be loaded.

*Example -*

### 1<sup>st</sup> Step

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

### 2<sup>nd</sup> step

getConnection() method of DriverManager class is used to create a Connection object.

*Syntax-*

```
public static Connection getConnection(String url) throws SQLException;  
public static Connection getConnection(String url, String username, String password) throws  
SQLException;
```

URL represents the information that is used by the driver to establish a database connection. Different drivers require different information in different formats.

*In case of Type1, url has following format:-*

**"jdbc:odbc:DatasourceName"**

Let a DSN named myDb be created.

Then, connection con = DriverManager.getConnection("jdbc:odbc:myDb");

### 3<sup>rd</sup> Step-

Connection Interface provide createStatement () factory method for creating statement object.

*Syntax -*

```
public Statement createStatement();
```

*Example:-*

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

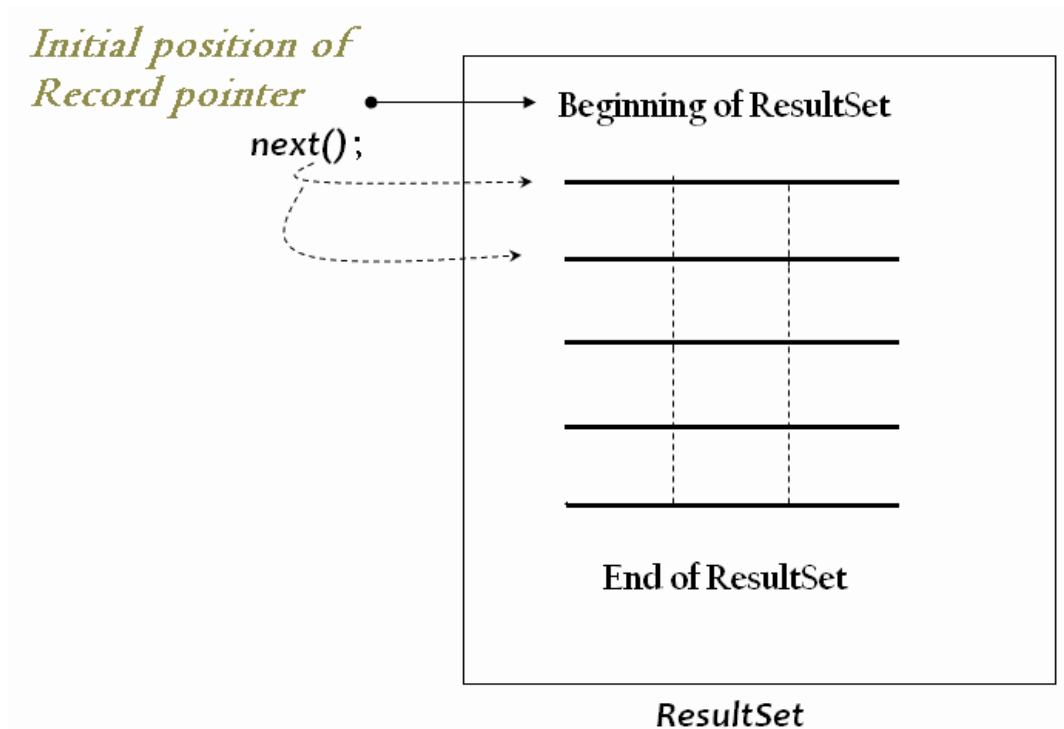
### 4<sup>th</sup> Step -

Statement interface provides following methods for executing queries:-

```
public ResultSet executeQuery(String selectQuery) throws SQLException;  
public int executeUpdate(String NonSelectDMLQuery) throws SQLException;  
public void execute (String NonDMLQuery) throws SQLException
```

If select query is executed data is to be obtained from the ResultSet. Obtaining data from the ResultSet is a 2 step process:-

1. Record pointer is to be placed on the desired record.
2. Value of individual fields of the record is read.



**next()** method of Resultset is used to advance the record pointer by one record.

**public boolean next();**

Resultset interface provides various methods to read the value of individual fields of current record.

General signature of these methods is:-

public type getType(int fieldindex) throws SQLException;

Actual Methods:-

**public String getString(int index) throws SQLException;**

**public int getInt(int index) throws SQLException;**

**public float getFloat(int index) throws SQLException;**

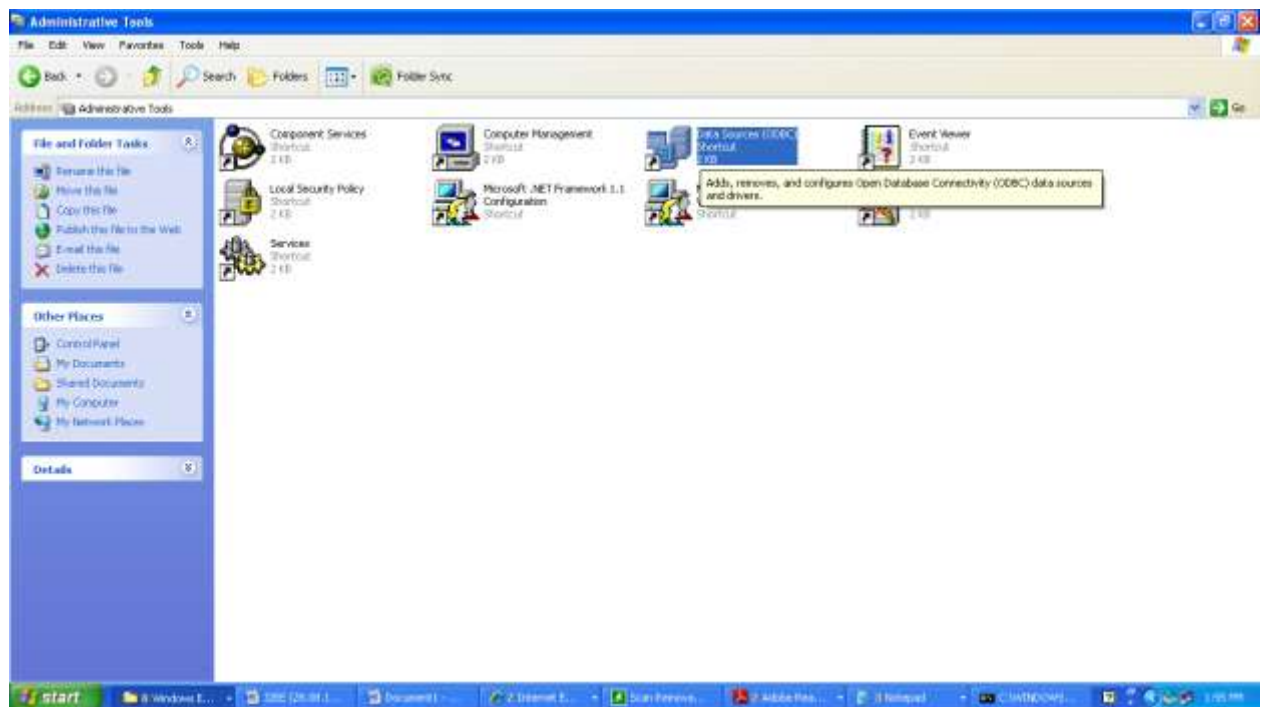
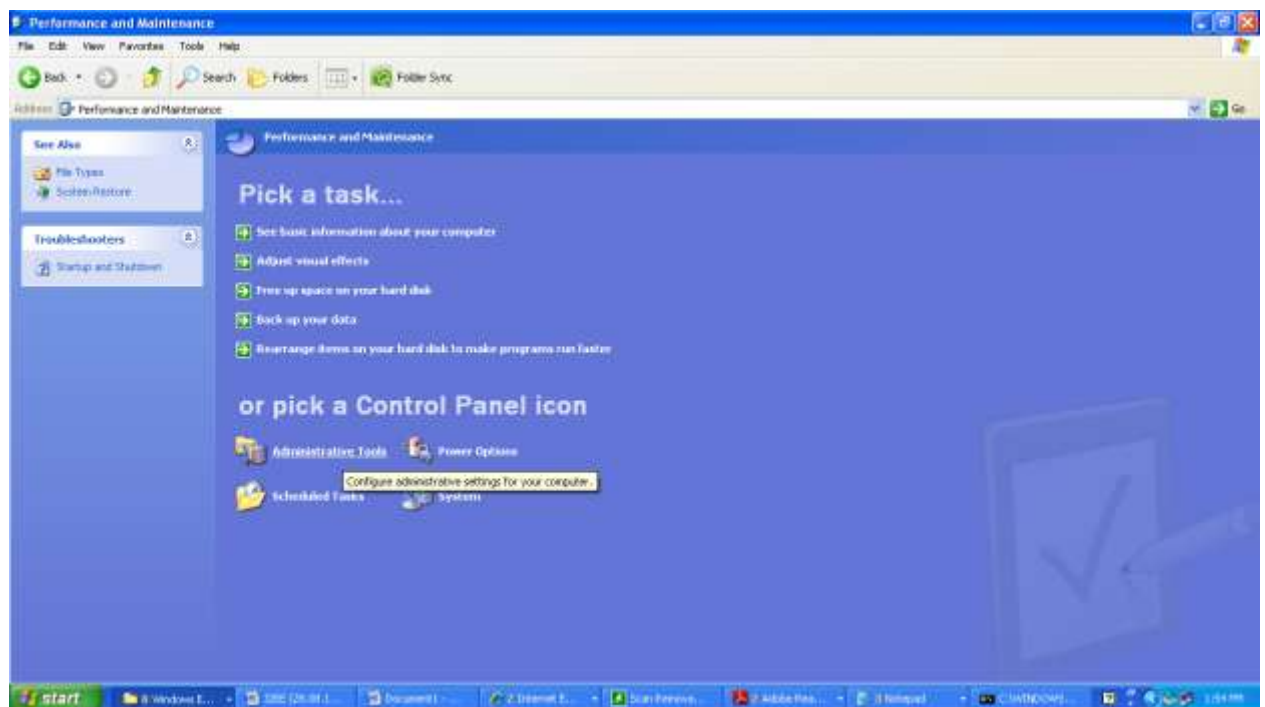
etc.

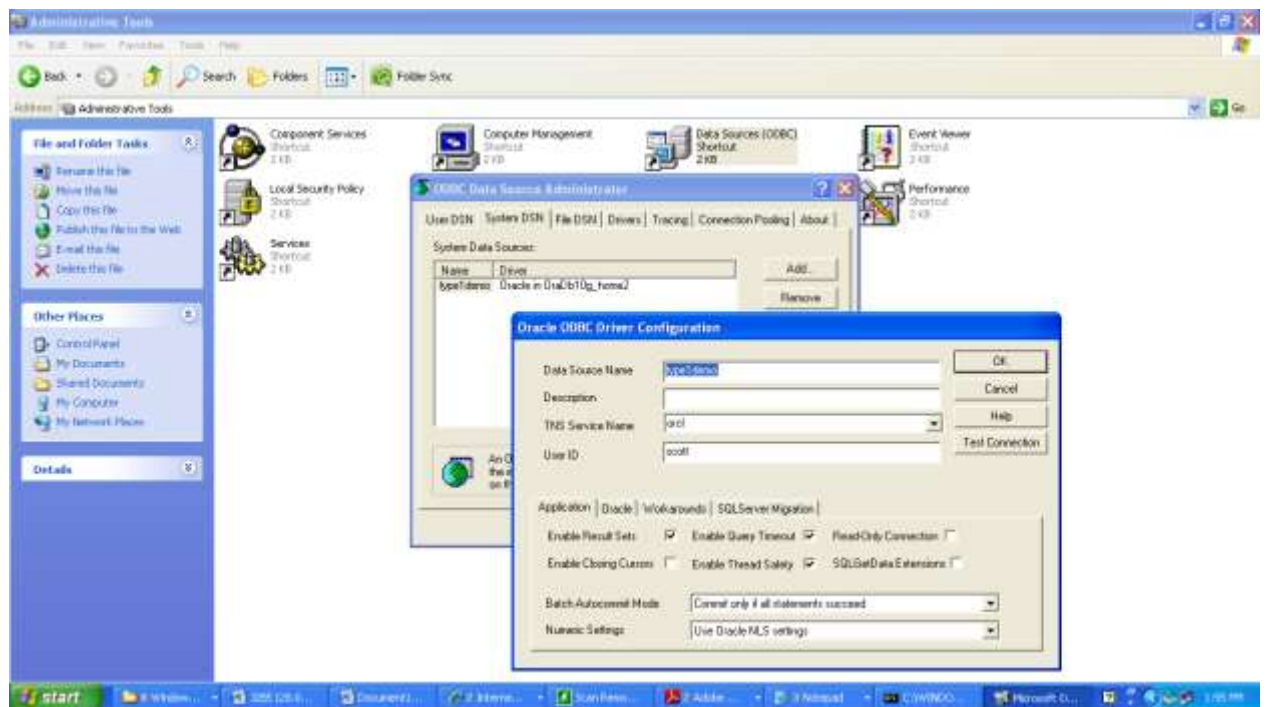
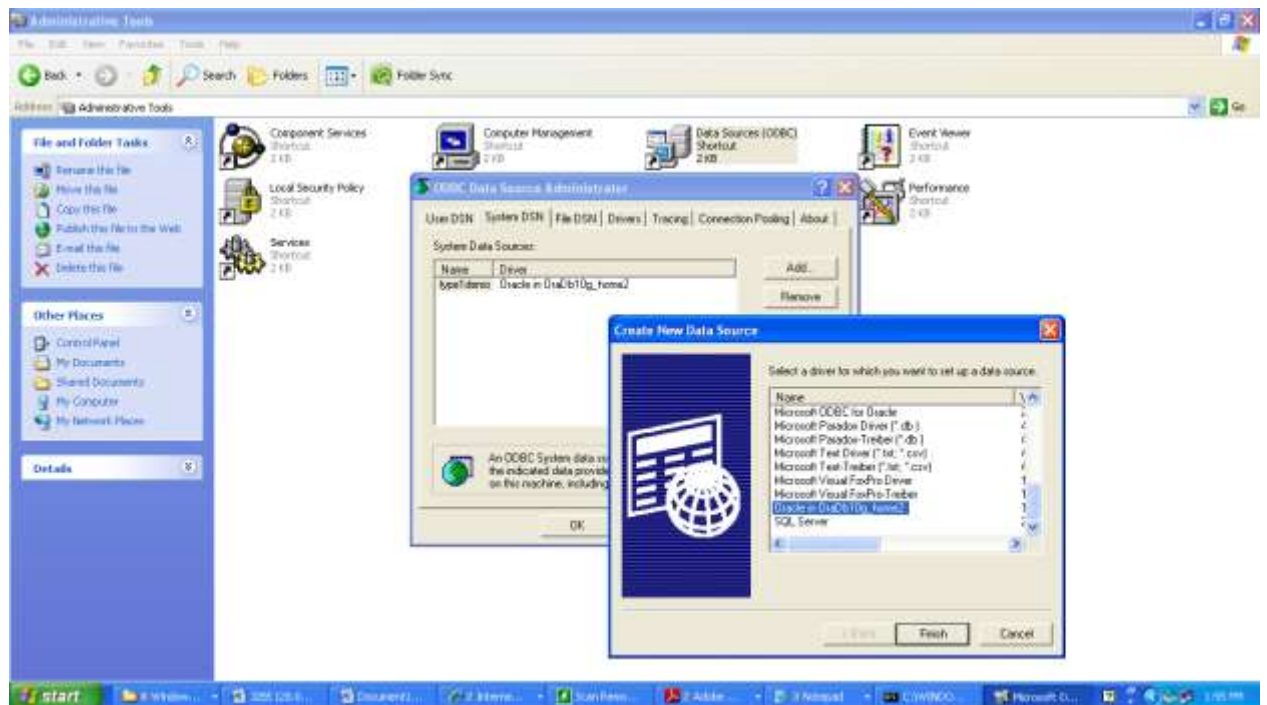
**Last step -**

Close() method of Connection interface is used to close the connection.

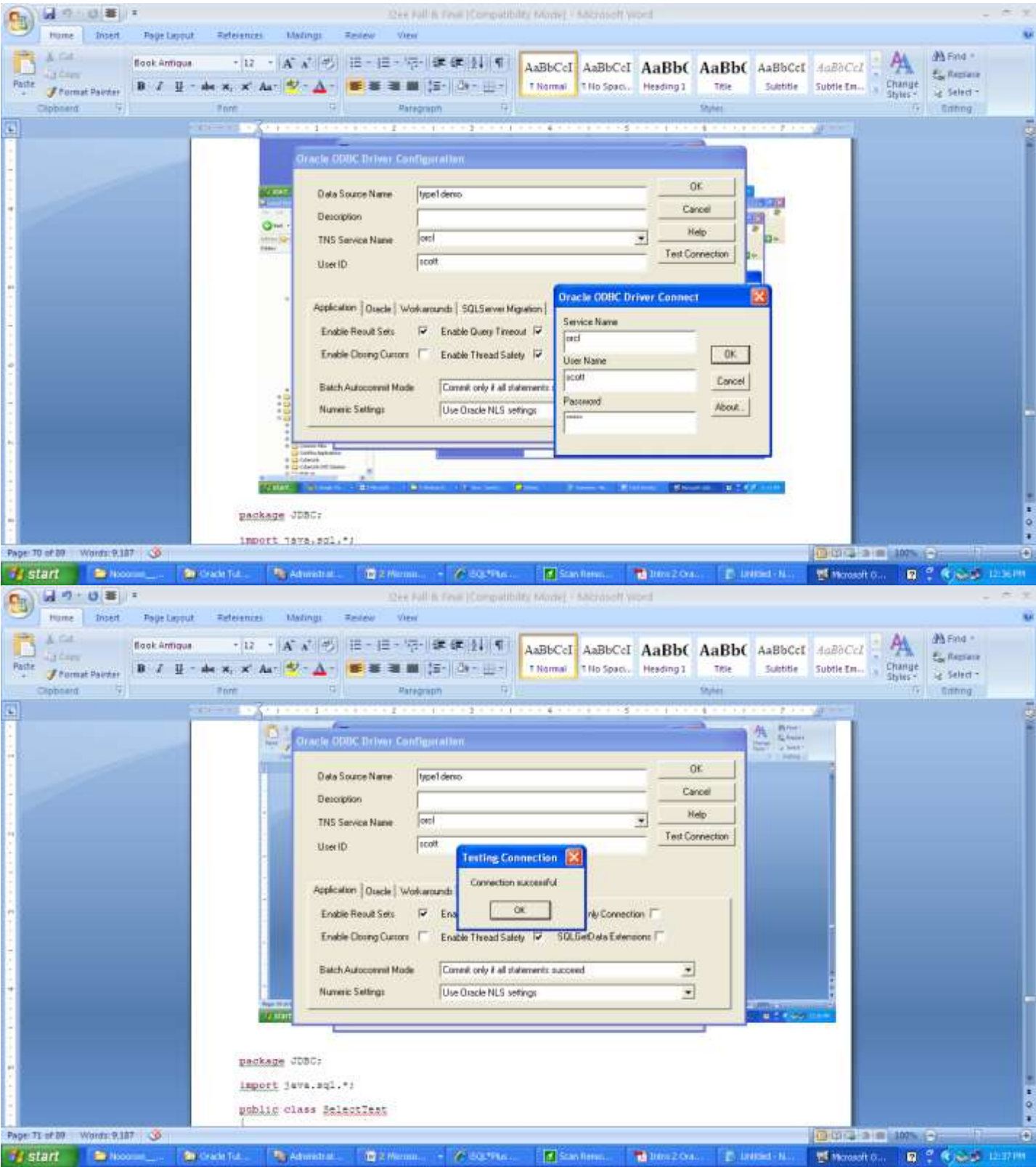
public void close() throws SQLException;

- **How to use Type1 driver**









Example of type1



The screenshot shows a Java IDE with a file named `Type1JdbcDemo.java`. The code is as follows:

```
import java.sql.*;

class Type1JdbcDemo
{
    public static void main(String args[])
    {
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection("jdbc:odbc:type1demo", "scott", "tiger");
            Statement stat = con.createStatement();
            ResultSet rset = stat.executeQuery("Select * from jdbcdemo");
            System.out.println("Following records are selected....");
            while(rset.next())
            {
                System.out.println(rset.getString(1) + "\t" + rset.getInt(2) + "\t" + rset.getInt(3) +
                    "\t" + rset.getString(4));
            }
            con.close();
        } catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

An overlaid command prompt window shows the execution of the program:

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>g:

G:\>javac Type1JdbcDemo.java

G:\>java Type1JdbcDemo
Following records are selected....
Rohit    21      8      sector-16, f-12
Mohit    23      10     sector-15, g-21

G:\>_
```

Below the command prompt, a web browser window displays the ISQL\*Plus interface. The workspace shows the SQL statement `select * from jdbcdemo` and the resulting data table:

NAME	AGE	CLASS	ADDRESS
Rohit	21	8	sector-16, f-12
Mohit	23	10	sector-15, g-21

`package` JDBC;

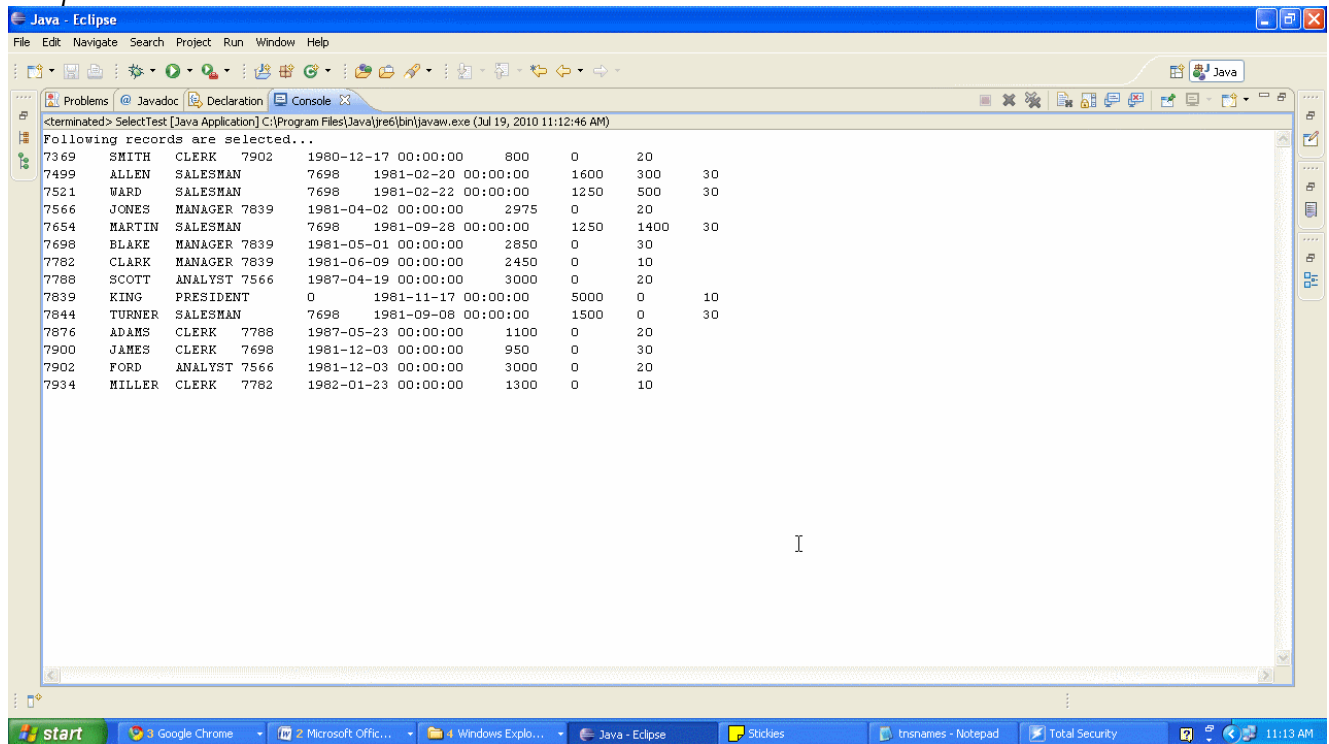
`import` java.sql.\*;

## J2EE Notes (By Neeraj Sir)

```
public class SelectTest
{
    public static void main(String[] args)
    {
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection("jdbc:odbc:myDb", "scott",
"tiger");

            Statement stmt = con.createStatement();
            ResultSet rset = stmt.executeQuery("select * from emp");
            System.out.println("Following records are selected...");
            while(rset.next())
            {
                System.out.println(rset.getInt(1)+ "\t" + rset.getString(2)+ "\t"
+ rset.getString(3)+ "\t" + rset.getInt(4)+ "\t" + rset.getString(5)+ "\t" +
rset.getInt(6)+ "\t" + rset.getInt(7)+ "\t" + rset.getInt(8));
            }
            con.close();
        } catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

### Output –



Using Type4 JDBC Driver for Oracle oracle.jdbc.driver.OracleDriver class need to be loaded.

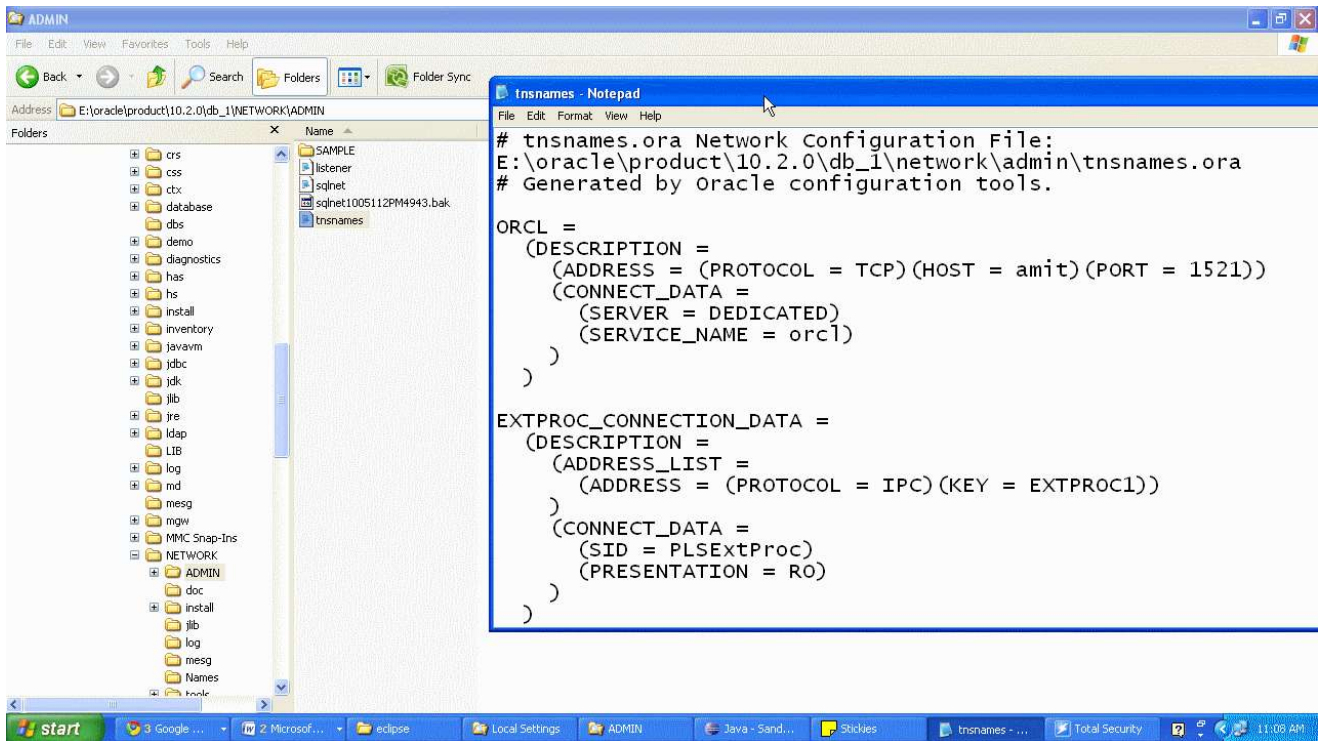
url has the following format :-

Variable

ServiceName is saved



## J2EE Notes (By Neeraj Sir)



*In order to use Type 4 JDBC Driver for Oracle jar file containing implementation classes of driver need to be available in the classpath.*

ojdbcversionNo.jar

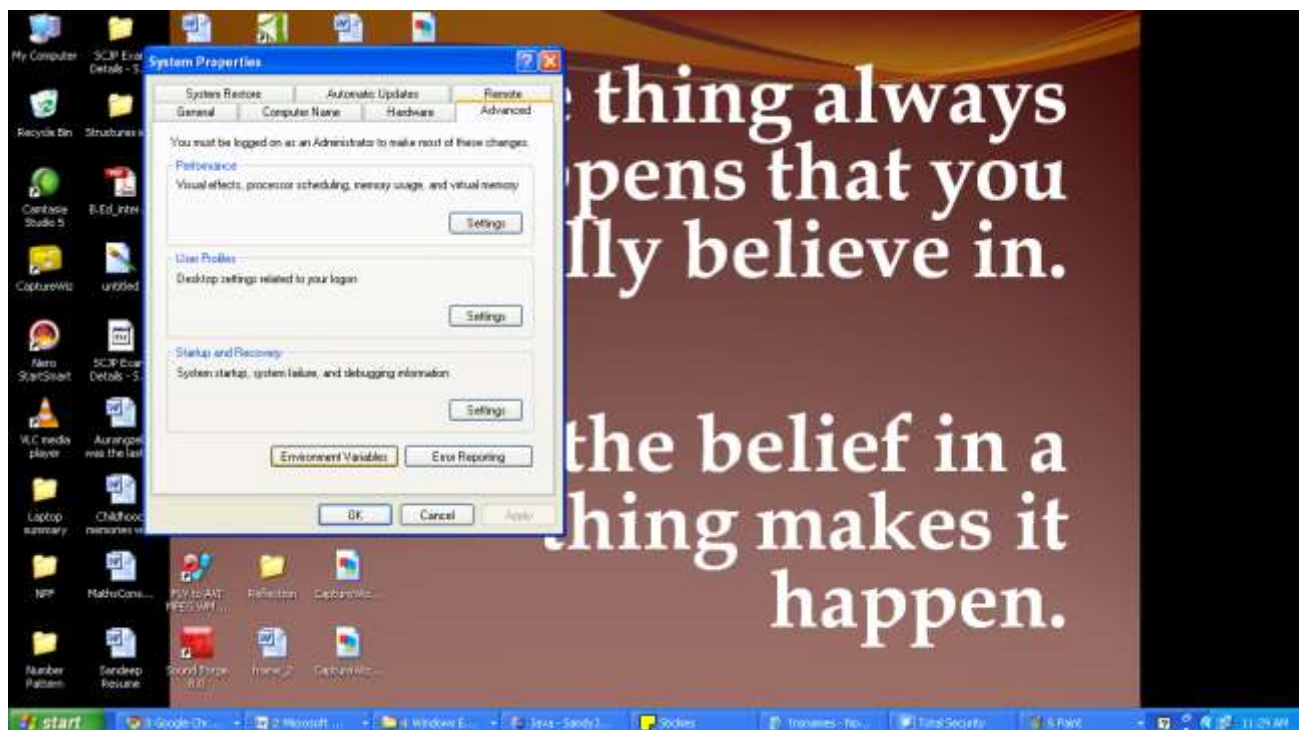
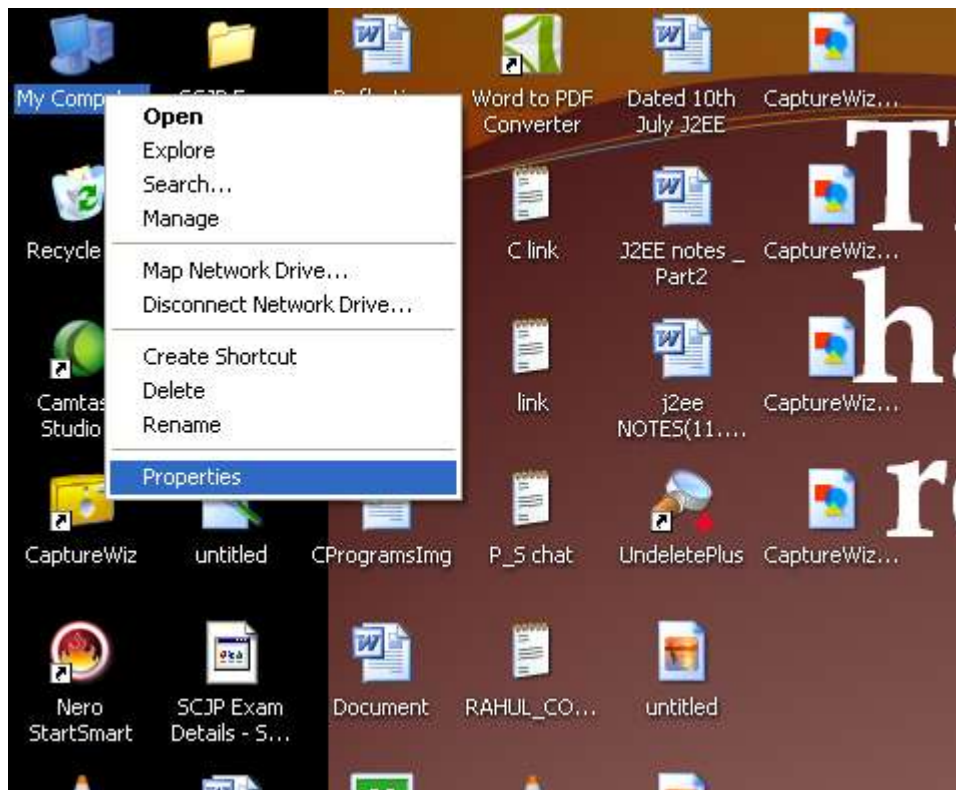
ojdbc14.jar

ojdbc10.jar

ojdbc15.jar

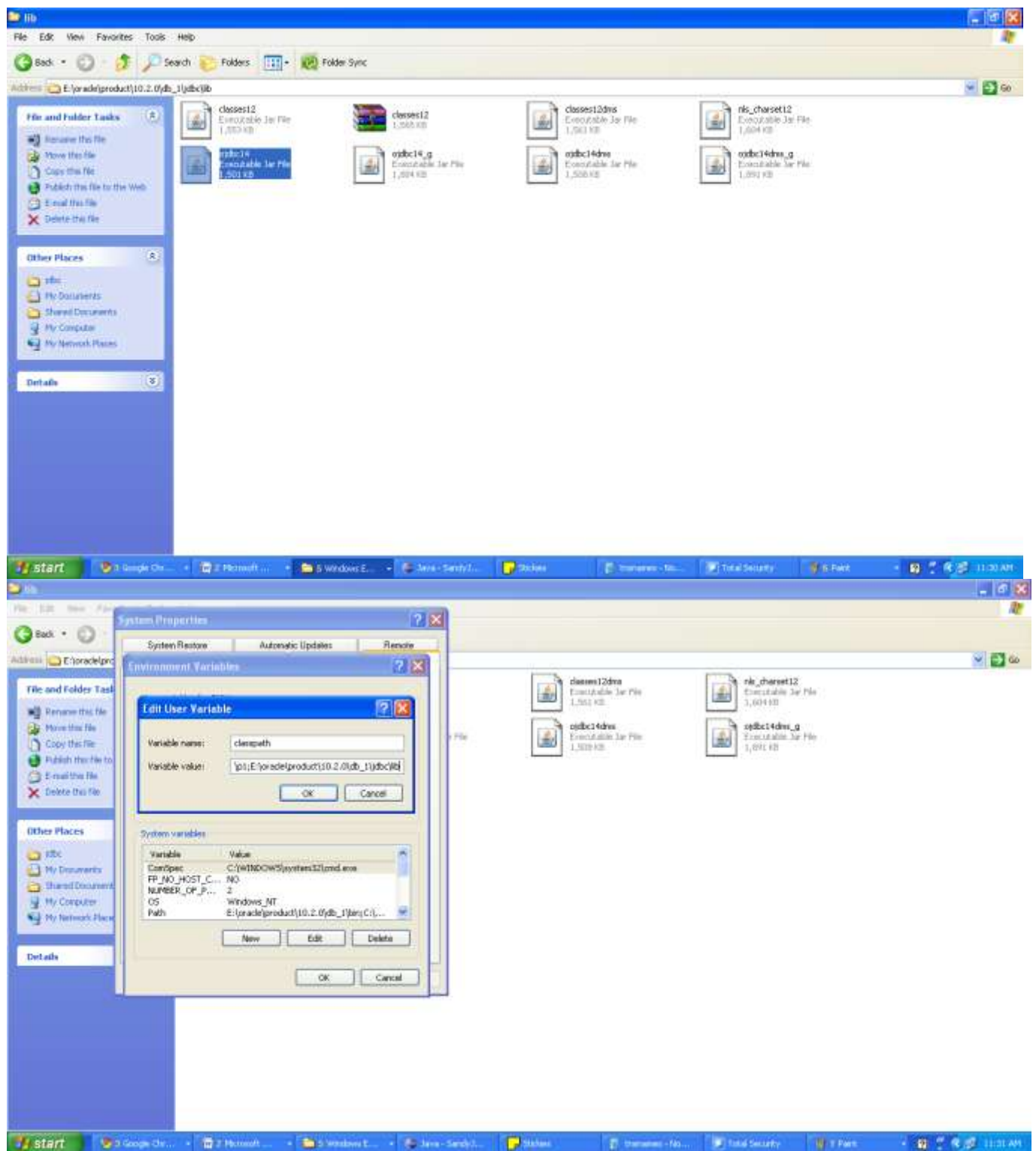
etc.

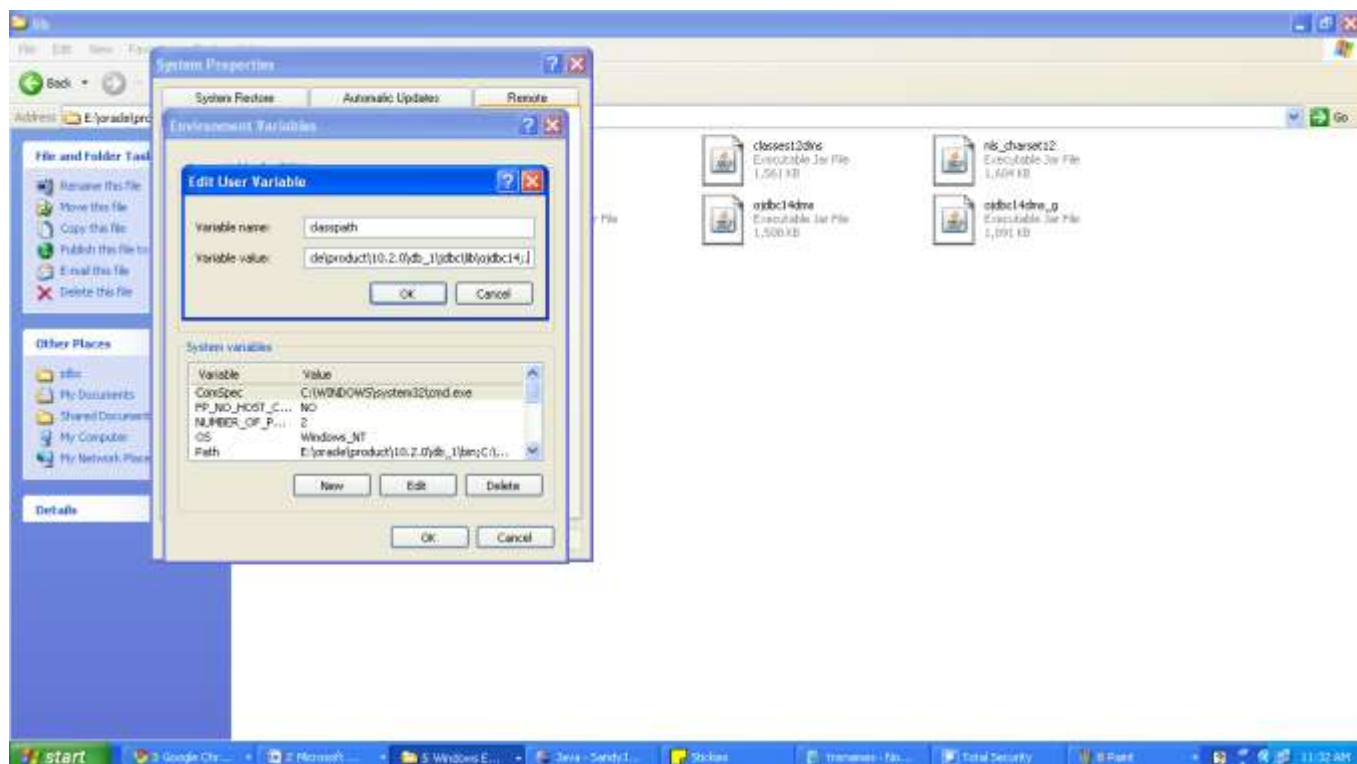
*(See the figure below to figure out the whole process)*





## J2EE Notes (By Neeraj Sir)



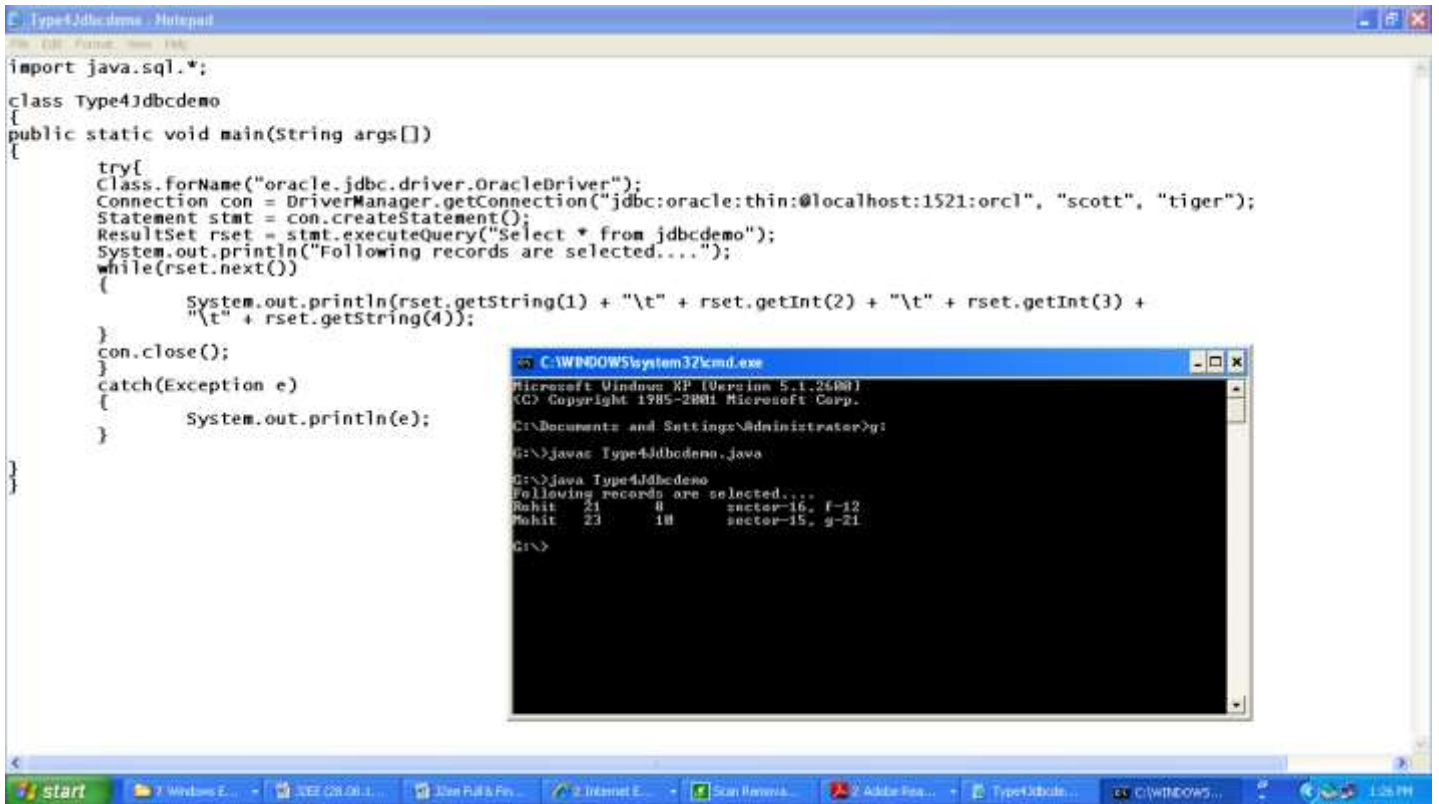


**Note :** Each time we change database, we need to change URL in our program. So instead of following that pattern we will modify the same program using properties file.

package JDBC;

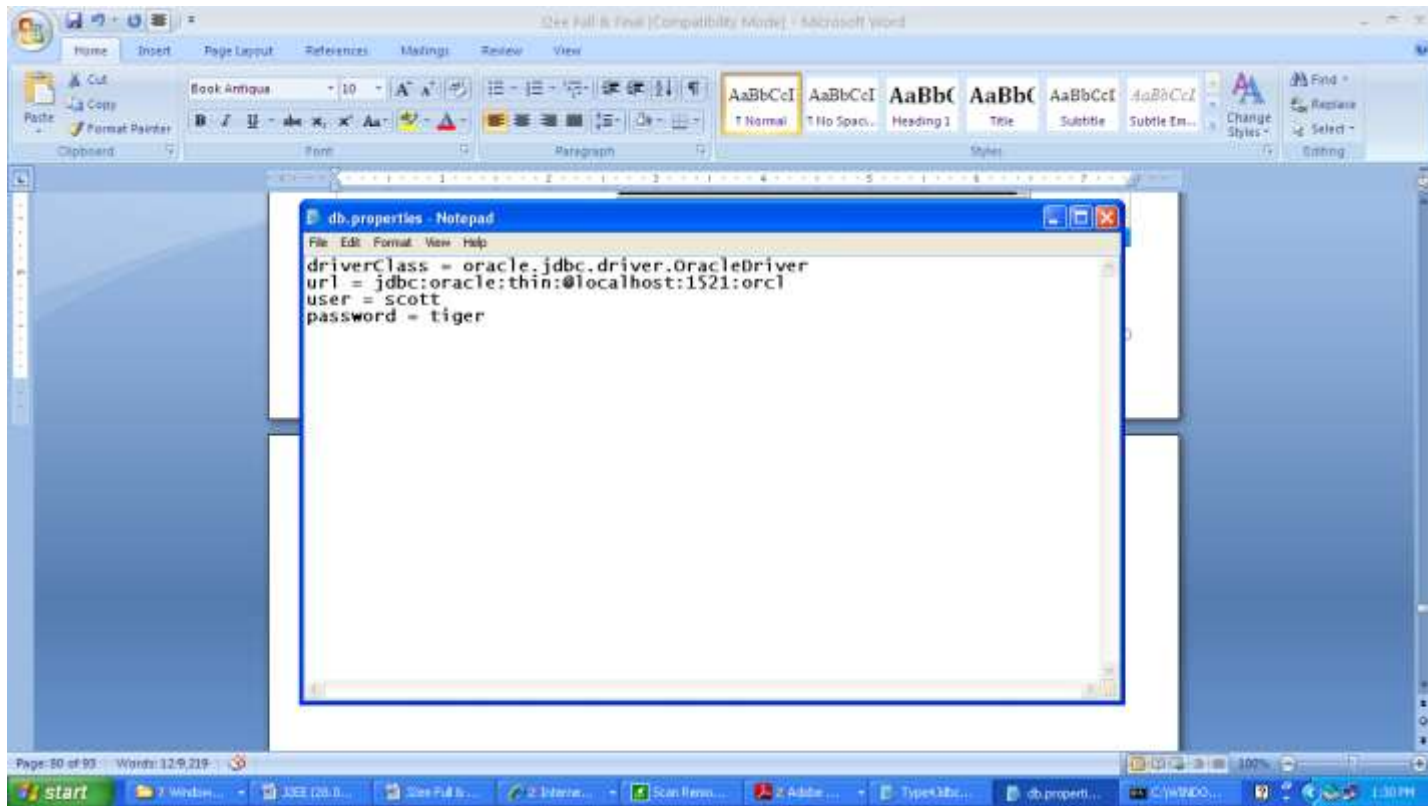
#### Example using type4 driver

1. To set path of ojdbc.jar



2. Now using properties file(type4 driver)





#### db.properties

```
driverClass = oracle.jdbc.driver.OracleDriver
url = jdbc:oracle:thin:@localhost:1521:orcl
user = scott
password = tiger
```

First we will create connection class using properties file

#### ConnectionProvider.java

```
import java.util.*;
import java.io.*;
import java.sql.*;

public class ConnectionProvider
{
    public static Connection getConnection()
    {
        Connection con = null;
        try
        {
            Properties p = new Properties();
            p.load(new FileInputStream("db.properties"));
            Class.forName(p.getProperty("driverClass"));
            con = DriverManager.getConnection(p.getProperty("url"),
                p.getProperty("user"), p.getProperty("password"));
        } catch (Exception e)
        {
        }
    }
}
```

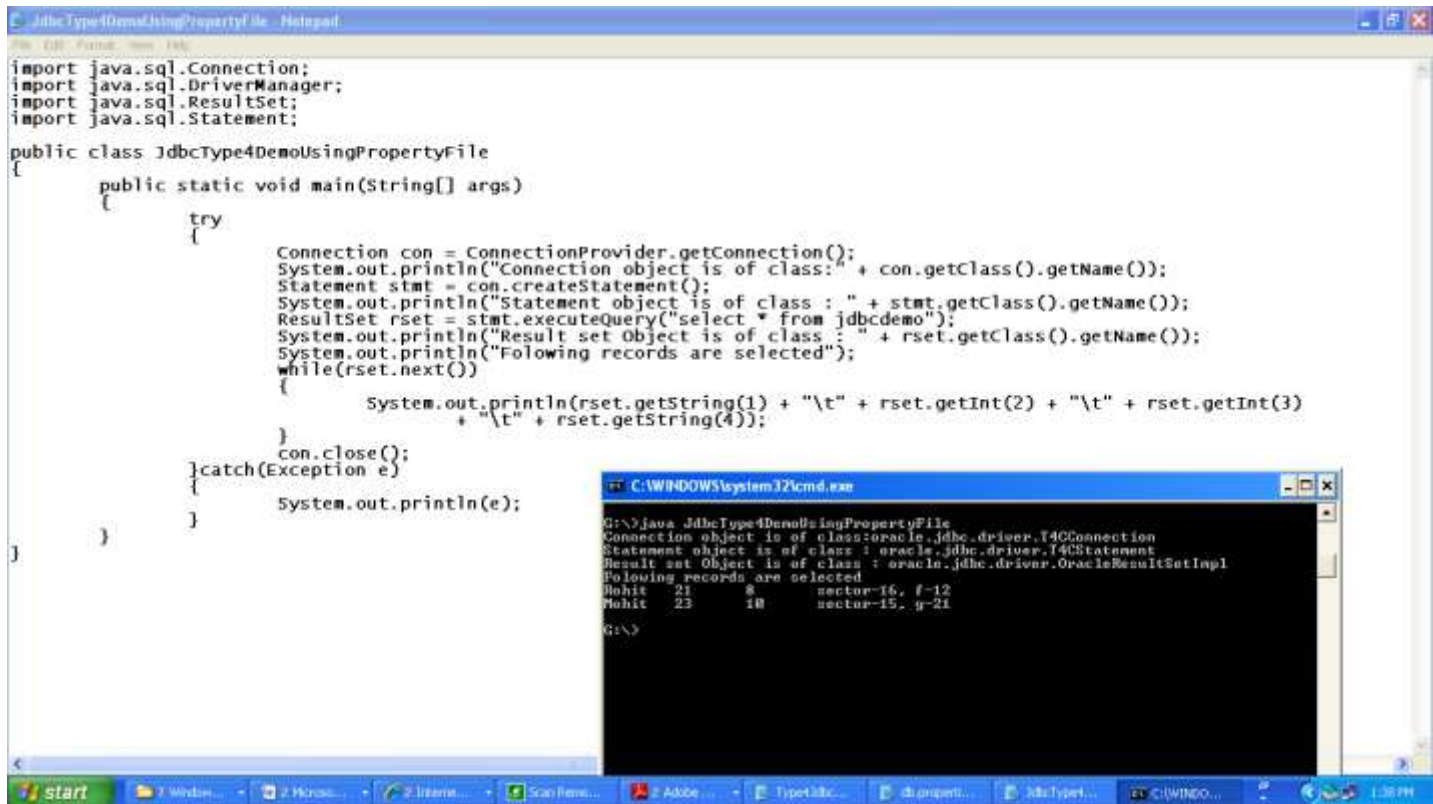
```
        System.out.println(e);
    }
    return con;
}
}
```

Now, **JdbcType4DemoUsingPropertyFile.java**

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcType4DemoUsingPropertyFile
{
    public static void main(String[ ] args)
    {
        try
        {
            Connection con = ConnectionProvider.getConnection();
            System.out.println("Connection object is of class:" + con.getClass().getName());
            Statement stmt = con.createStatement();
            System.out.println("Statement object is of class : " + stmt.getClass().getName());
            ResultSet rset = stmt.executeQuery("select * from jdbcdemo");
            System.out.println("Result set Object is of class : " + rset.getClass().getName());
            System.out.println("Folowing records are selected");
            while(rset.next())
            {
                System.out.println(rset.getString(1) + "\t" + rset.getInt(2) + "\t" +
                    rset.getInt(3) + "\t" + rset.getString(4));
            }
            con.close();
        }catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Output-



The screenshot displays a Windows desktop environment. In the background, a Notepad window titled 'JdbcType4DemoUsingPropertyFile - Notepad' contains the following Java code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

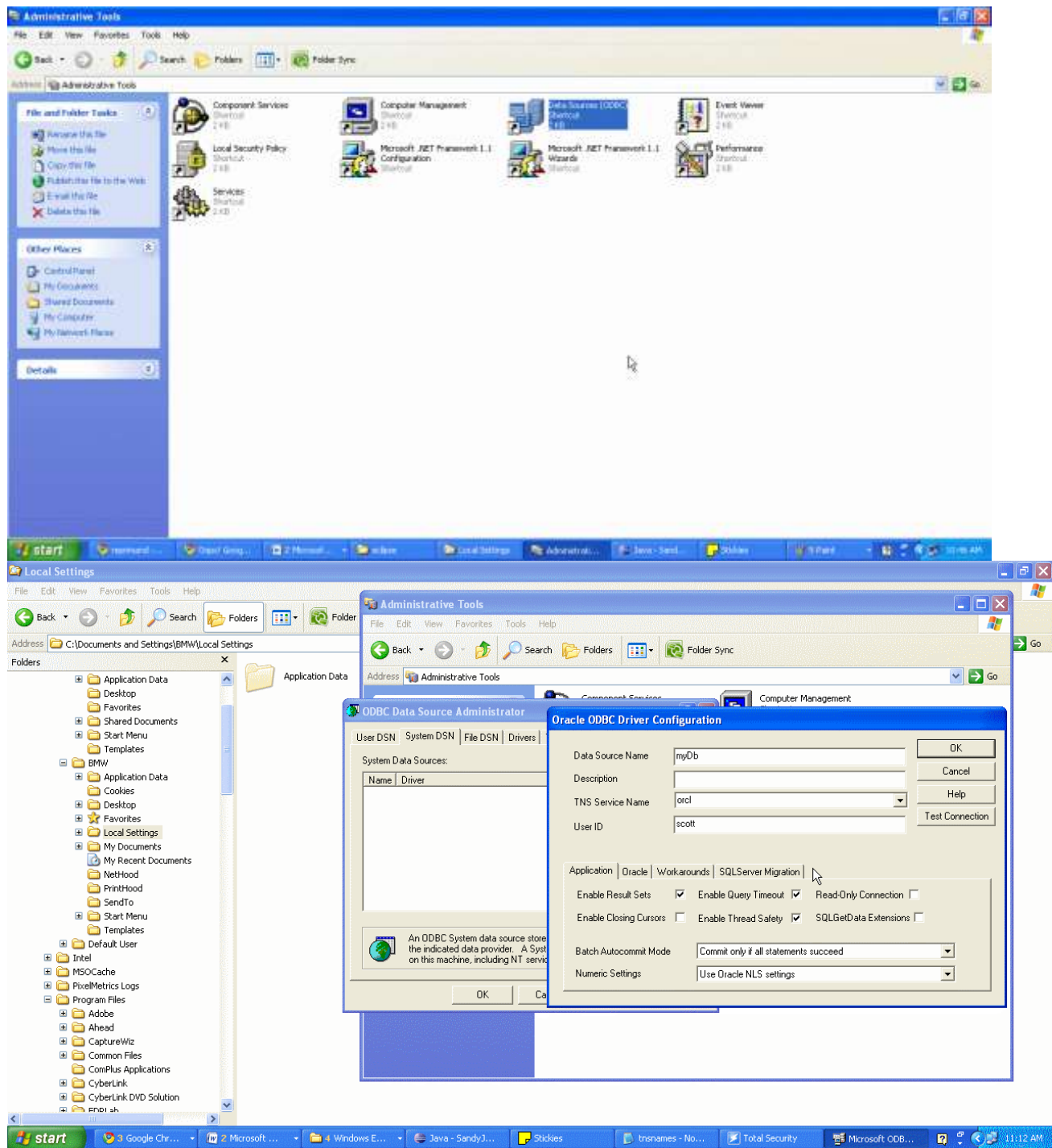
public class JdbcType4DemoUsingPropertyFile
{
    public static void main(String[] args)
    {
        try
        {
            Connection con = ConnectionProvider.getConnection();
            System.out.println("Connection object is of class:" + con.getClass().getName());
            Statement stmt = con.createStatement();
            System.out.println("Statement object is of class : " + stmt.getClass().getName());
            ResultSet rset = stmt.executeQuery("select * from jdbcdemo");
            System.out.println("Result set Object is of class : " + rset.getClass().getName());
            System.out.println("Following records are selected");
            while(rset.next())
            {
                System.out.println(rset.getString(1) + "\t" + rset.getInt(2) + "\t" + rset.getInt(3)
                    + "\t" + rset.getString(4));
            }
            con.close();
        } catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

In the foreground, a command prompt window titled 'C:\WINDOWS\system32\cmd.exe' shows the execution of the program. The output is as follows:

```
G:\>java JdbcType4DemoUsingPropertyFile
Connection object is of class:oracle.jdbc.driver.T4CConnection
Statement object is of class : oracle.jdbc.driver.T4CStatement
Result set Object is of class : oracle.jdbc.driver.OracleResultSetImpl
Following records are selected
Mehit 21 8 sector-16, f-12
Mehit 23 10 sector-15, g-21
G:\>
```

The Windows taskbar at the bottom shows the Start button and several open applications: '7 WinBox', '7 WinBox', '7 IZams...', 'Scan Remo...', '2 Adobe...', 'Type4Jbc...', 'JdbcType...', and 'C:\WINDO...'.

## J2EE Notes (By Neeraj Sir)



```
package JDBC;

import java.sql.*;

public class SelectTest
{
    public static void main(String[] args)
    {
        try
```

## J2EE Notes (By Neeraj Sir)

```
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection("jdbc:odbc:myDb", "scott",
"tiger");

    Statement stmt = con.createStatement();
    ResultSet rset = stmt.executeQuery("select * from emp");
    System.out.println("Following records are selected...");
    while(rset.next())
    {
        System.out.println(rset.getInt(1)+ "\t" + rset.getString(2)+ "\t"
+ rset.getString(3)+ "\t" + rset.getInt(4)+ "\t" + rset.getString(5)+ "\t" +
rset.getInt(6)+ "\t" + rset.getInt(7)+ "\t" + rset.getInt(8));
    }
    con.close();
} catch (Exception e)
{
    System.out.println(e);
}
}
```

### Output -

Employee ID	Name	Job	Salary	Commission	Hire Date	Department Name	Manager Name
7369	SMITH	CLERK	7902	0	1980-12-17 00:00:00	800	0
7499	ALLEN	SALESMAN	7698	0.3	1981-02-20 00:00:00	1600	300
7521	WARD	SALESMAN	7698	0.3	1981-02-22 00:00:00	1250	500
7566	JONES	MANAGER	7839	0.1	1981-04-02 00:00:00	2975	0
7654	MARTIN	SALESMAN	7698	0.3	1981-09-28 00:00:00	1250	1400
7698	BLAKE	MANAGER	7839	0.1	1981-05-01 00:00:00	2850	0
7782	CLARK	MANAGER	7839	0.1	1981-06-09 00:00:00	2450	0
7788	SCOTT	ANALYST	7566	0.1	1987-04-19 00:00:00	3000	0
7839	KING	PRESIDENT	0	0.1	1981-11-17 00:00:00	5000	0
7844	TURNER	SALESMAN	7698	0.3	1981-09-08 00:00:00	1500	0
7876	ADAMS	CLERK	7788	0	1987-05-23 00:00:00	1100	0
7900	JAMES	CLERK	7698	0	1981-12-03 00:00:00	950	0
7902	FORD	ANALYST	7566	0	1981-12-03 00:00:00	3000	0
7934	MILLER	CLERK	7782	0	1982-01-23 00:00:00	1300	0

Using Type4 JDBC Driver for Oracle oracle.jdbc.driver.OracleDriver class need to be loaded.

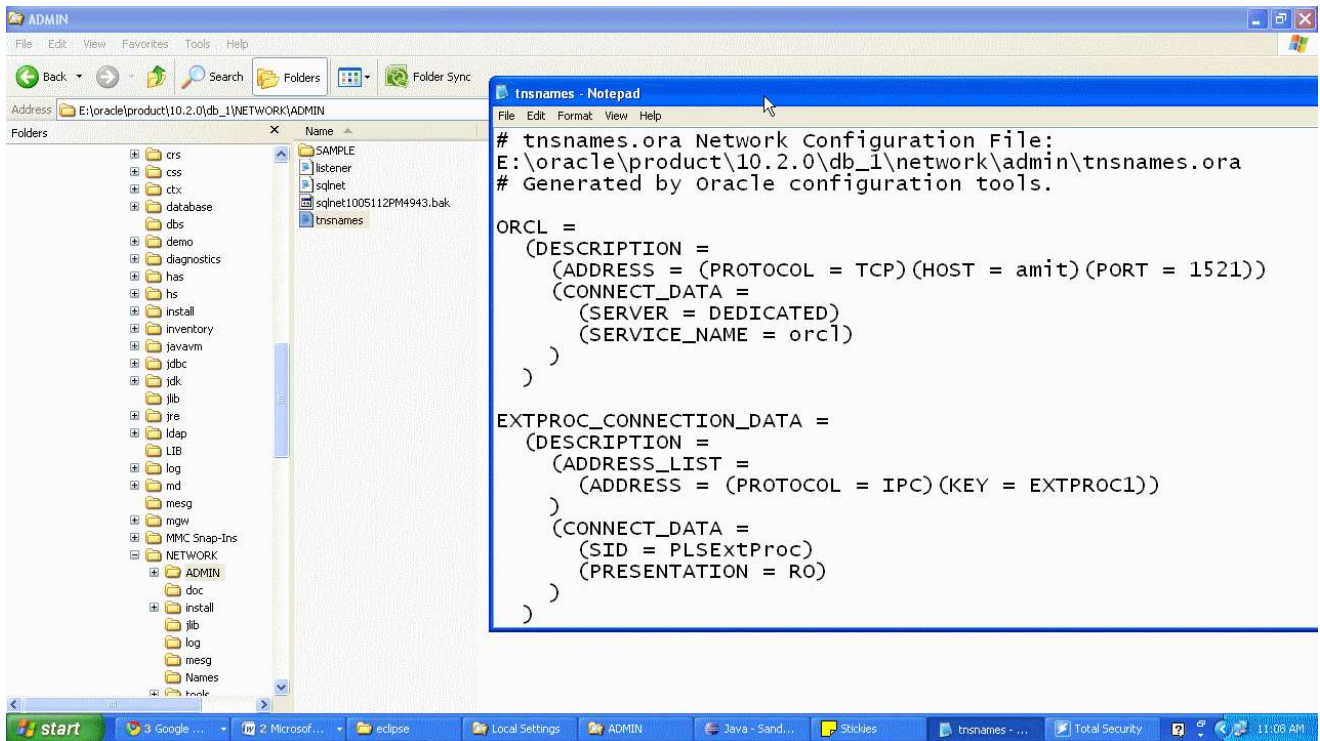
url has the following format :-

Variable

ServiceName is saved



## J2EE Notes (By Neeraj Sir)



*In order to use Type 4 JDBC Driver for Oracle jar file containing implementation classes of driver need to be available in the classpath.*

ojdbcversionNo.jar

ojdbc14.jar

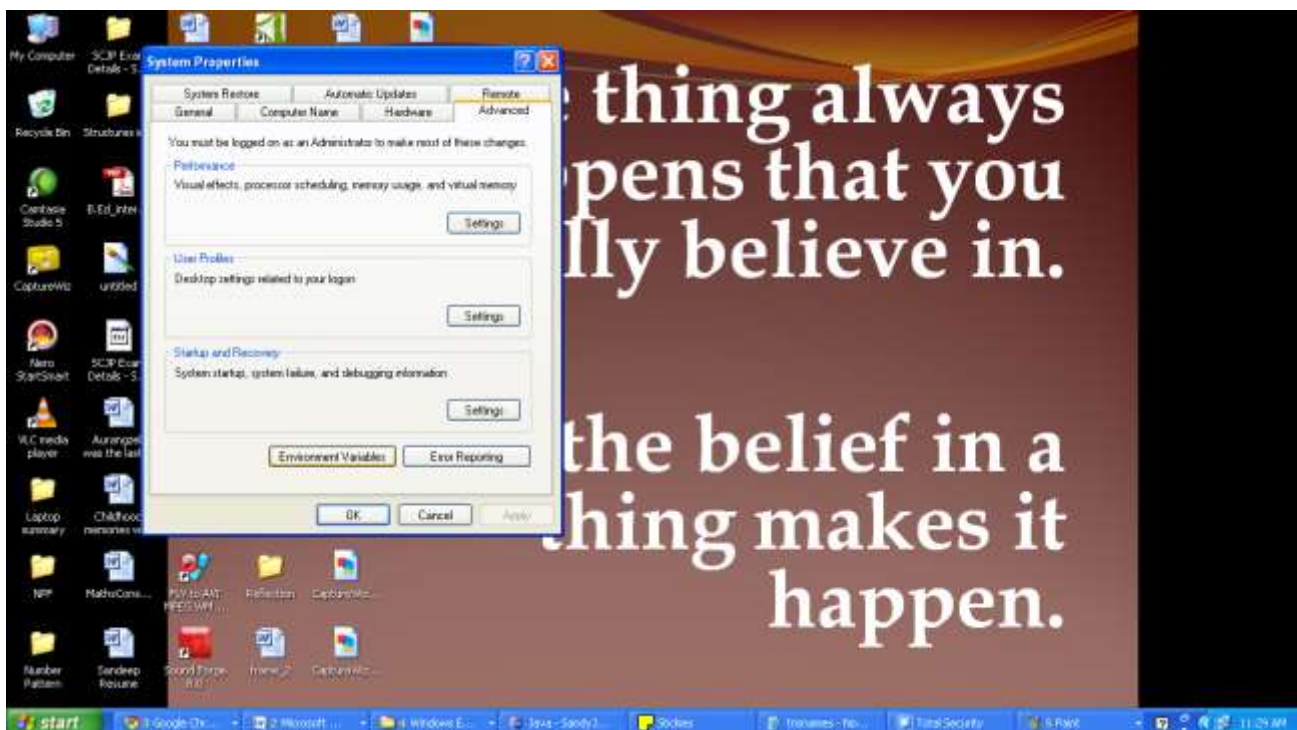
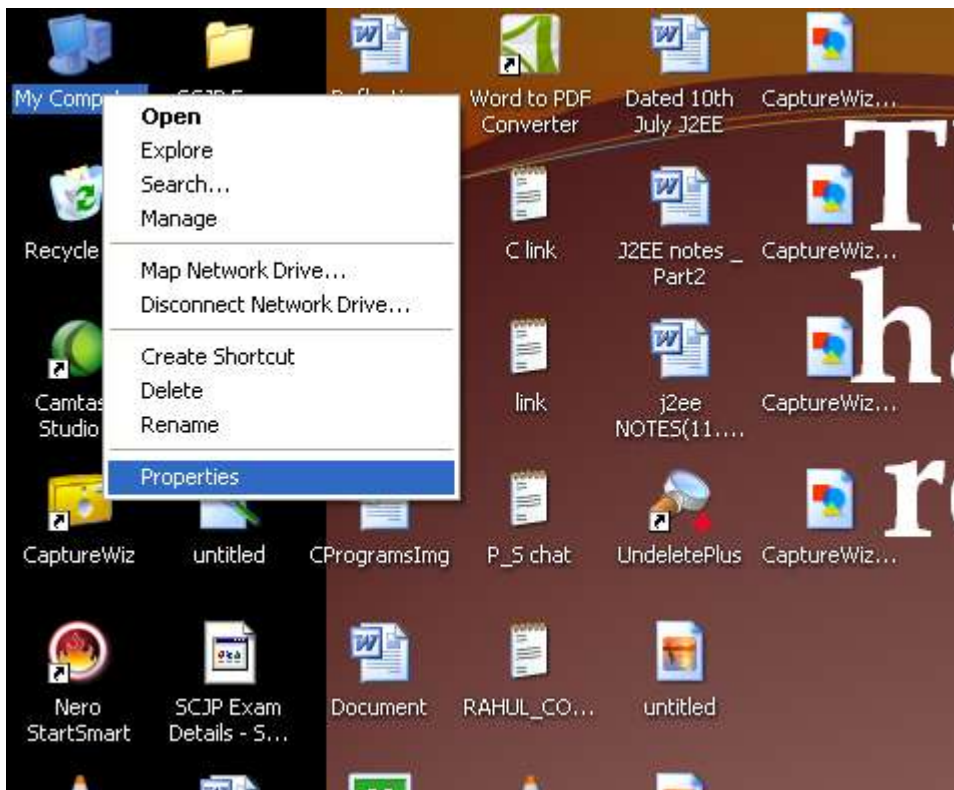
ojdbc10.jar

ojdbc15.jar

etc.

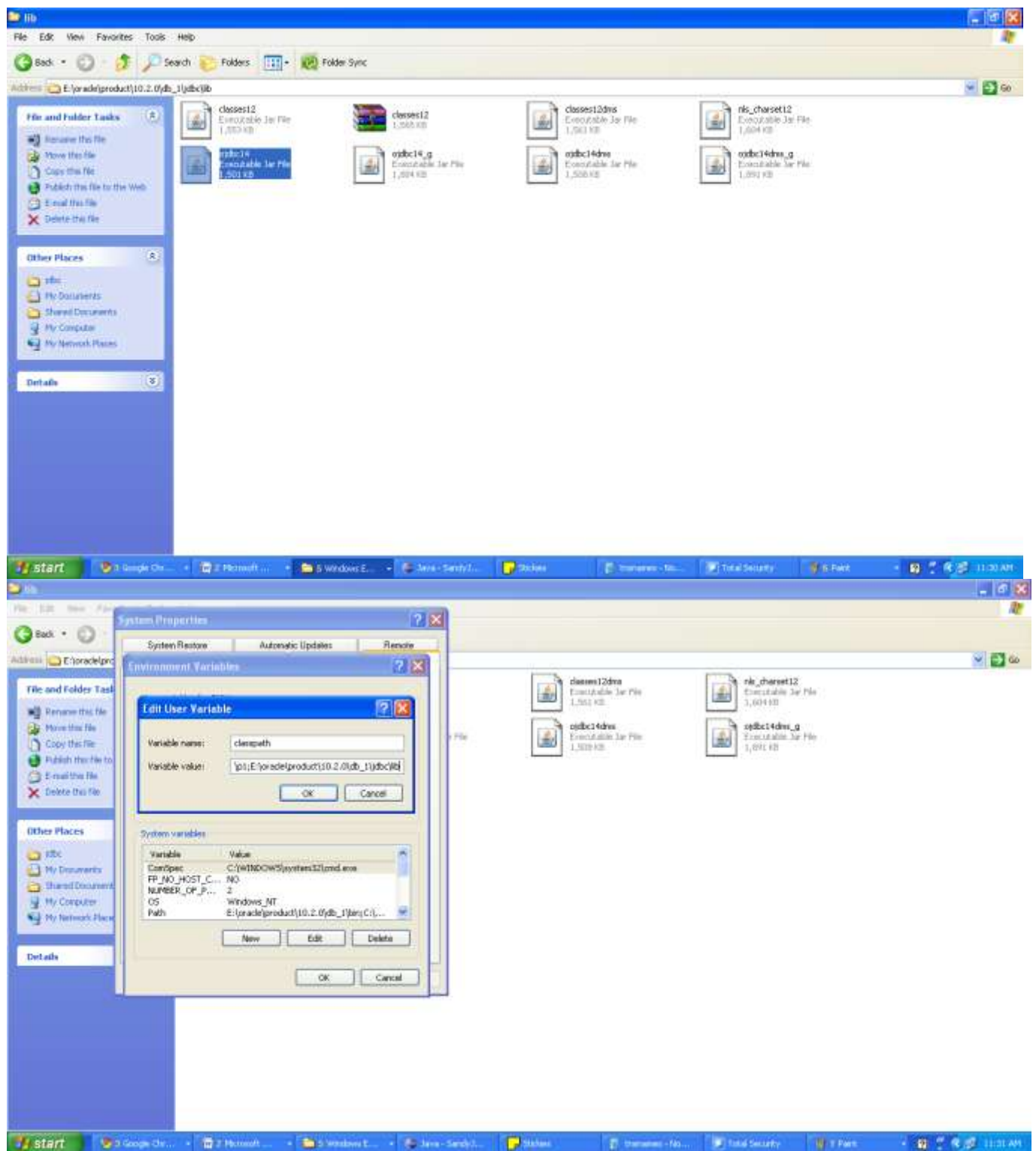
*(See the figure below to figure out the whole process - )*

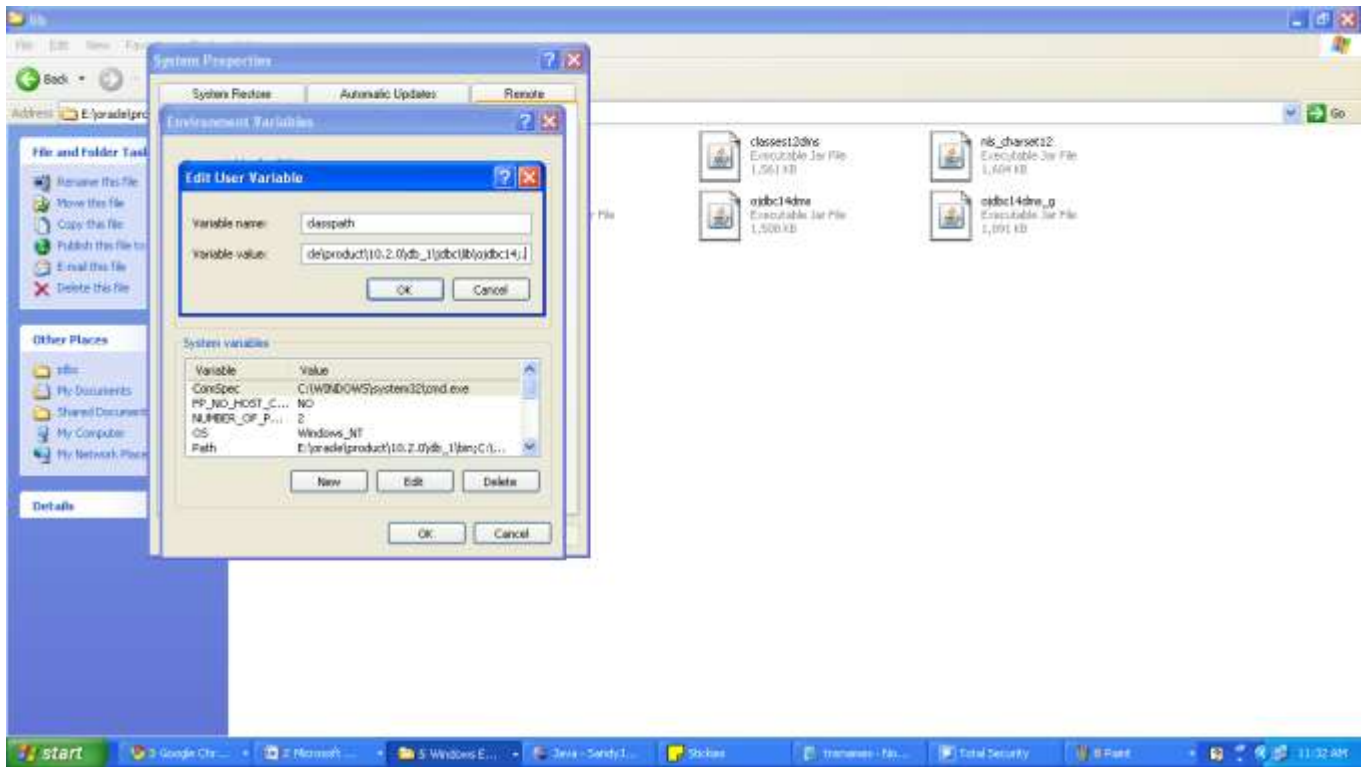






## J2EE Notes (By Neeraj Sir)





**Note :** Each time we change database, we need to change URL in our program. So instead of following that pattern we will modify the same program using properties file.

package JDBC;

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
```

```
public class SelectTest2
{
    public static void main(String[ ] args)
    {
        try
        {
            Connection con = ConnectionProvider.getConnection();
            System.out.println("Connection object is of class:" + con.getClass().getName());
            Statement stmt = con.createStatement();
            System.out.println("Statement object is of class : " + stmt.getClass().getName());
            ResultSet rset = stmt.executeQuery("select * from emp");
            System.out.println("Result set Object is of class : " + rset.getClass().getName());
            System.out.println("Following records are selected");
            con.close();
        }
    }
}
```

```
        }catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

package JDBC;
import java.util.*;
import java.io.*;
import java.sql.*;

public class ConnectionProvider
{
    public static Connection getConnection()
    {
        Connection con = null;
        try
        {
            Properties p = new Properties();
            p.load(new FileInputStream("db.properties"));
            Class.forName(p.getProperty("driverClass"));
            con = DriverManager.getConnection(p.getProperty("url"),
                p.getProperty("user"),p.getProperty("password"));

        }catch(Exception e)
        {
            System.out.println(e);
        }
        return con;
    }
}
```

**Problem** : Whenever we change the Query, we need to compile the code again.

**Dated: 25.07.10**

❖ **Difference between PL/SQL procedures & functions**

PL/SQL function explicitly returns a value whereas PL/SQL procedure doesn't.

PL/SQL procedure uses "OUT" type parameters for returning values.

In PL/SQL, parameters can be of 3 types-

Parameter Type	Purpose
IN	Used to provide input to procedures and functions.
OUT	Used to receive result from procedures.
IN OUT	Used to provide input as well as receive output from procedures.

❖ In 'C'

```
int a = 5, b;
```

```
b = square(a);
```

---

```
int square (int x)
```

```
{
```

```
    return x * x;
```

```
}
```

Here x is used to receive input. Such parameters are of type **IN** in PL/SQL.

❖

```
int a = 5, b;
```

```
b = square(a, &b);
```

---

```
void square (int x, int y)
```

```
{
```

```
    &y = x * x;
```

```
}
```

Here y is used to indirectly return result. Such parameters are of type **OUT** in PL/SQL.



```
int a = 5;

square(&a);

void square (int *p)
{
    *p = *p * *p;
}
```

Here p is used to receive input as well as return result indirectly. Such parameters are of type **IN OUT** in PL/SQL.

**Syntax of defining PL/SQL procedures:**

```
create or replace procedure procedureName(ParameterName [IN, OUT , IN OUT]
DataType, ..... )
is
local variable declaration
begin
    body of procedure
end;
```

**Syntax of defining PL/SQL functions:**

```
create or replace function functionName(Parameter IN DataType, ..... )
return ReturnType
is
local variable declaration
begin
    body of function
end;
```

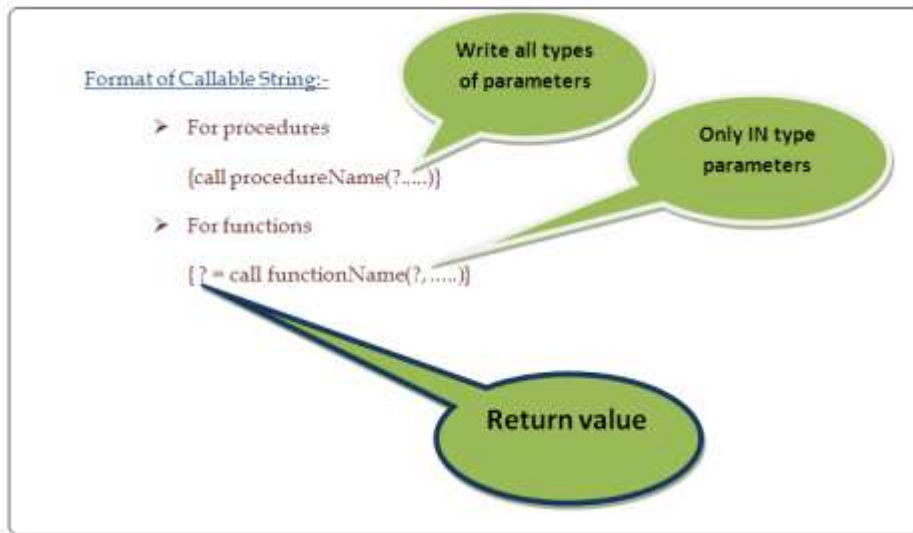
- ❖ Callable Statement is used to invoke procedures and functions.

*Invoking a procedure & function from a Java Application requires following step-*

**1.** CallableStatement object is created.

**prepareCall()** method of Connection interface is used to obtain a **CallableStatement** object.

**public CallableStatement prepareCall (String CallableString) throws SQLException;**



2. value of all IN type parameters is set.

setter methods exposed by **PreparedStatement** interface is used for setting the value of IN type parameters.

3. Type of value expected from each OUT type parameter is specified.

**registerOutParameter()** method of **CallableStatement** interface is used to specify the type of OUT parameters.

public void **registerOutParameter(int index, int type)** throws SQLException;

**java.sql.Types** class defines **static final int** constants which are used to specify type

Example -

**Types.INTEGER**  
**Types.VARCHAR**  
**Types.BYTE**  
etc.

4. Execute the procedure or function.

**execute()** method is used for this purpose.

5. Read the value of OUT type parameters.

**CallableStatement** interface provides various methods to read the value of OUT type parameters.

❖ General signature of these methods

**public Type getType (int parameterIndex) throws SQLException;**

○ Actual Methods

**public int getInt (int parameterIndex) throws SQLException;**

**public String getString(int parameterIndex) throws SQLException;**

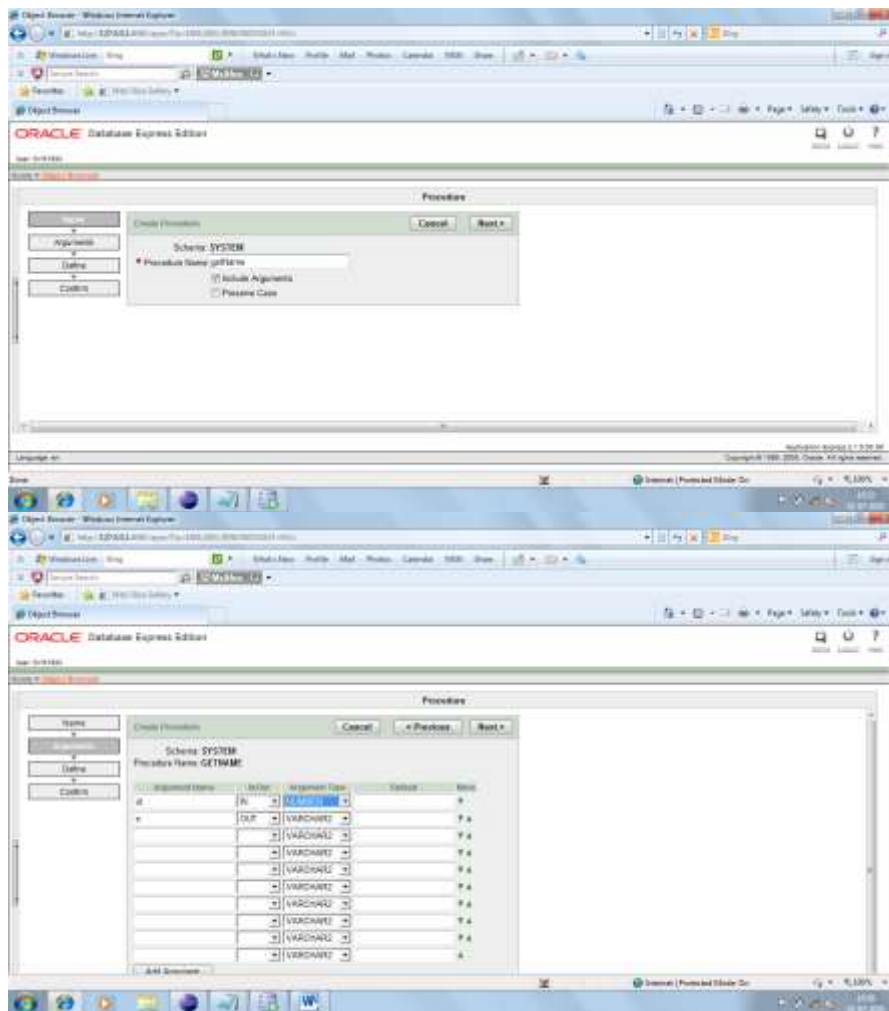
**public Date getDate(int parameterIndex) throws SQLException;**

etc.

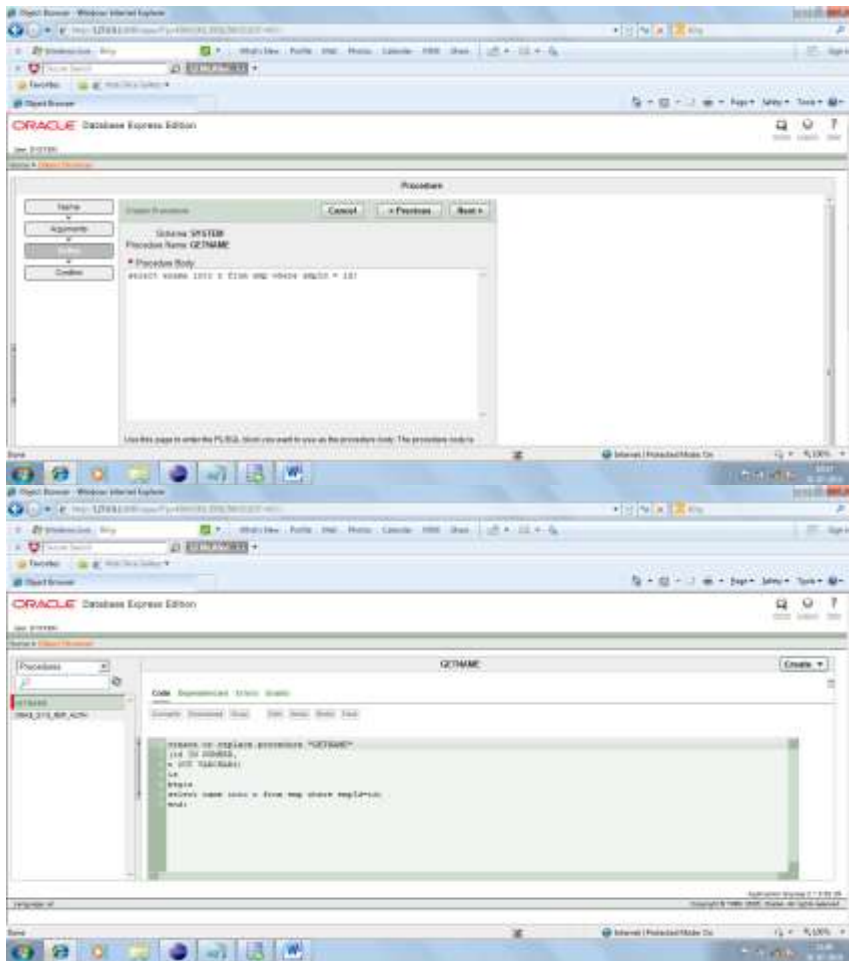
❖ Example-

We want to give ID as input, and want to get Name from emp table.

select ename into n from emp where empId=id;







**package** JDBC;

**import** java.io.BufferedReader;  
**import** java.sql.CallableStatement;  
**import** java.sql.Connection;  
**import** java.sql.Types;

**class** ProcTest

```

{
    public static void main(String[] args)
    {
        try
        {
            Connection con = ConnectionProvider.getConnection();
            CallableStatement stmt = con.prepareCall("{call getName(?, ?)}");
            BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter Id : ");
            int id = Integer.parseInt(b.readLine());
            stmt.setInt(1, id);
            stmt.registerOutParameter(2, Types.VARCHAR);
            stmt.execute();
            System.out.println("Name is : " + stmt.getString(2));
            con.close();
        } catch (Exception e)
        {

```

```
        System.out.println(e);
    }
}
}
```



```
package JDBC;
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Types;
```

```
class FuncTest
```

```
{
    public static void main(String[] args)
    {
        try
        {
            Connection con = ConnectionProvider.getConnection();
            CallableStatement stmt = con.prepareCall("{?= call getJob(?)}");
            BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter Id : ");
            int id = Integer.parseInt(b.readLine());
            stmt.setInt(2, id);
            stmt.registerOutParameter(1, Types.VARCHAR);
            stmt.execute();
            System.out.println("Job is : " + stmt.getString(1));
            con.close();
        } catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```



Creating a SQL file (myproc.sql)

```
create or replace procedure getDetails(id IN NUMBER, n OUT VARCHAR2, j OUT
VARCHAR2 , s OUT NUMBER)
is
begin
select name, job, salary into n, j, s from emp where empId = id;
end;
```

## PROGRAM

```
package JDBC;
```

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.Types;

class ProcCreator
{
    public static void main(String[ ] args)
    {
        try
        {
            Connection con = ConnectionProvider.getConnection();
            Statement stmt = con.createStatement();
            FileInputStream f = new FileInputStream(args[0]);
            byte a[ ] = new Byte[f.available( )];
            f.read(a);
            stmt.execute(new String(a));
            con.close();
            System.out.println("Successfully created.");
        } catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

- ❖ **ResultSetMetaData** provides methods for obtaining information about the result contained in a **ResultSet**.

**getMetaData()** method of **ResultSet** interface is used to obtain a **ResultSetMetaData** object.

**public ResultSetMetaData getMetaData();**

Commonly used methods of **ResultSetMetaData** interface –

- **getColumnCount()** method returns number of columns in the **ResultSet**.

**public int getColumnCount();**

- **getColumnName()** method returns the name of the specified column.

**public String getColumnName(int ColumnIndex);**

- **getColumnTypeName()** method returns the name of the datatype of the specified column.

**public String getColumnTypeCount(int index);**

- **getTableName()** method returns table name for the specified column.

**public String getTableName(int index);**

- etc.

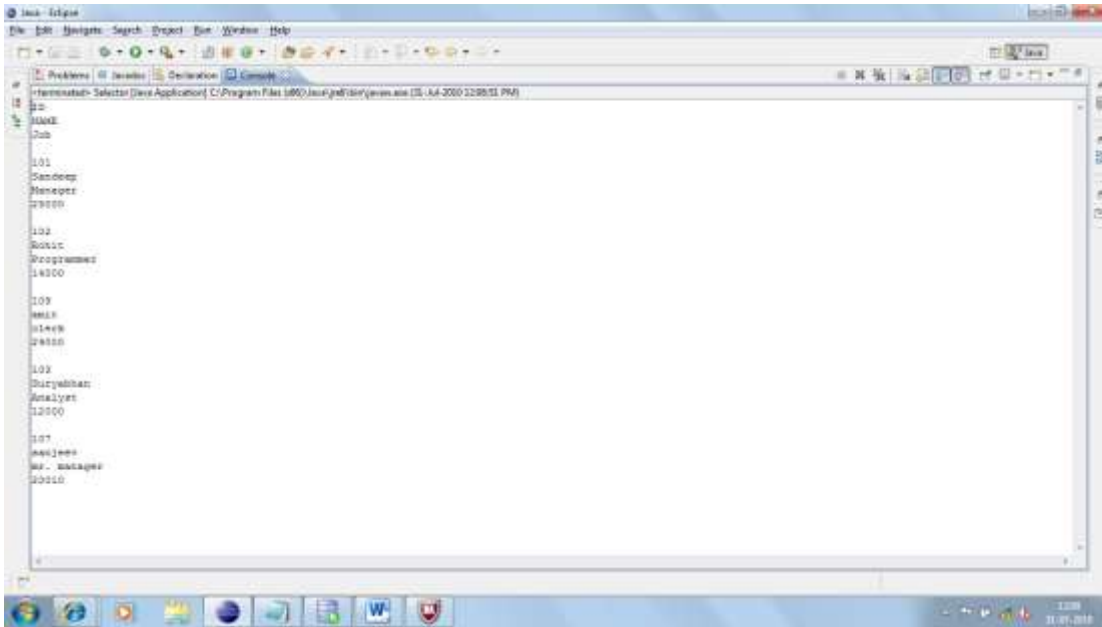
- ❖ *Define a class name Selector that receives a Table name as command-line argument & displays its record along with column header.*

```
package JDBC;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.sql.Types;

class Selector
{
    public static void main(String[] args)
    {
        try
        {
            Connection con = ConnectionProvider.getConnection();
            Statement stmt = con.createStatement();
            String query = "select * from " + args[0];
            ResultSet rset = stmt.executeQuery(query);
            ResultSetMetaData rmd = rset.getMetaData();
            int c = rmd.getColumnCount();
            for(int i=1; i<=c; i++)
                System.out.println(rmd.getColumnName(i)+"\t");
            while(rset.next())
            {
                System.out.println();
                for(int i=1; i<=c; i++)
                    System.out.println(rset.getString(i)+"\t");
            }
            con.close();
        } catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

OUTPUT-



- ❖ **DatabaseMetaData** interface provides methods for obtaining information about the database.

**getMetaData()** method of Connection interface is used to obtain a **DatabaseMetaData** object.

```
public DatabaseMetaData getMetaData();
```

- Commonly used methods of **DatabaseMetaData** interface
  - **getDatabaseProductName()** method returns the name of the RDBMS package.

```
public String getDatabaseProductName();
```

- **getDriverName()** method returns the details of the driver.

```
public String getDriverName();
```

- **getTables()** method is used to fetch the details of tables.

```
public ResultSet getTables(String catalog, String schema, String pattern, String[] criteria)  
throws SQLException
```


- ✚ This method returns a **ResultSet** in which 3<sup>rd</sup> field of records contains table name.
- ✚ This method receives 4 parameters, out of which first 3 are optional.

- ✚ **Last parameter** specifies the criterion that is used to identify **MetaData** table on which select queries executed. To fetch the details of data table, criteria must be **TABLE**.

Example

let dmd be a DatabaseMetaData object.

```
ResultSet rset = dmd.getTables(null, null, null, new String[]{"TABLE"});
```



Following query shall be executed in ORACLE.


```
select * from tab;
```

- ✚ **Second Last parameter** specifies the pattern that is used to filter records returned by the **select** query.

Example

let dmd be a DatabaseMetaData object.

```
ResultSet rset = dmd.getTables(null, null, "a%", new String[]{"TABLE"});
```



Following query shall be executed in ORACLE.

```
select * from tab where Table_Name like 'a%';  
(Only those tables whose name starts with "a" shall be selected)
```

- ✚ **Second parameter** is used to specify the schema. Schema represents user and database objects and privileges owned by the user.

## Example

let dmd be a DatabaseMetaData object.

```
ResultSet rset = dmd.getTables(null, "SYSTEM", "a%",
new String[]{"TABLE"});
```



Following query shall be executed in  
ORACLE.

(Only those tables which are owned by SYSTEM user  
and whose name starts with  
"a" shall be selected.)

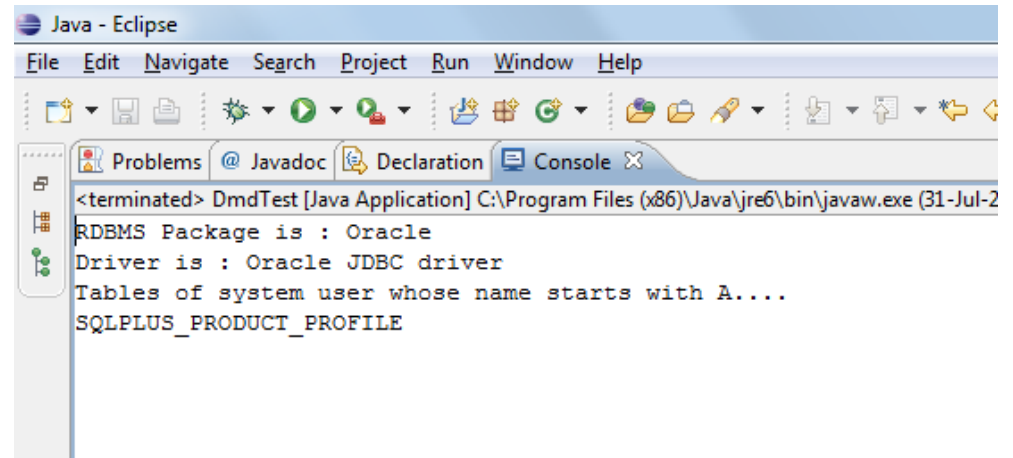
✚ First parameter represents group of schemas.

```
package JDBC;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

class DmdTest
{
    public static void main(String[] args)
    {
        try
        {
            Connection con =
            ConnectionProvider.getConnection();
            DatabaseMetaData dmd =con.getMetaData();
            System.out.println("RDBMS Package is : " +
            dmd.getDatabaseProductName());
            System.out.println("Driver is : " +
            dmd.getDriverName());
            System.out.println("Tables of system user whose
            name starts with A...");
            ResultSet rset = dmd.getTables(null, "SYSTEM",
            "S%", new String[]{"TABLE"});
            while(rset.next())
            {
                System.out.println(rset.getString(3)+"\t");
            }
            con.close();
        } catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

OUTPUT-



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. Below the menu bar is a toolbar with various icons. The main workspace area is divided into several tabs: Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the following output:

```
<terminated> DmdTest [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (31-Jul-2011 10:10:10 AM)
RDBMS Package is : Oracle
Driver is : Oracle JDBC driver
Tables of system user whose name starts with A....
SQLPLUS_PRODUCT_PROFILE
```