Java-Collection Frame Work

Collections in java is a framework that provides an architecture to store and manipulate the group of java objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (Array-List, LinkedList, Vector, Priority-Queue, HashSet, Linked-HashSet, Tree-Set etc).

What is Collection in java

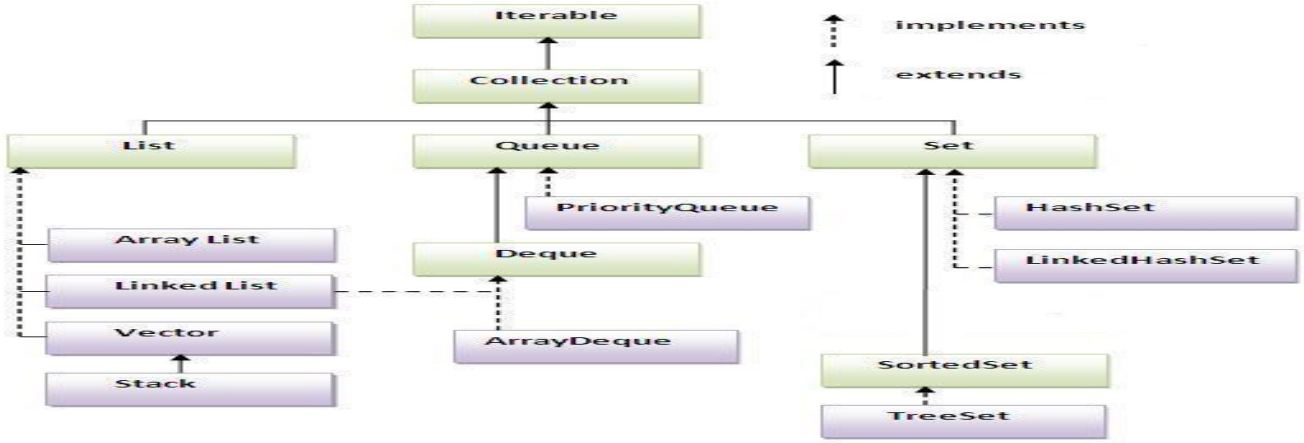
Collection represents a single unit of objects i.e. a group.

What is framework in java

provides readymade architecture. represents set of classes and interface.

Hierarchy of Collection Framework

Let us see the hierarchy of collection frame work. The java. util package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

No.	Method	Description
1.	public boolean add(Object element)	is used to insert an element in this collection.
2.	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3.	public boolean remove(Object element)	is used to delete an element from this collection.
4.	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5.	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6.	public int size()	return the total number of elements in the collection.
7.	public void clear()	removes the total no of element from the collection.
8.	Public boolean contains(Object element)	is used to search an element.
9.	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10.	public Iterator iterator()	returns an iterator.
11.	public Object [] toArray()	converts collection into array.
12.	public boolean isEmpty()	checks if collection is empty.
13.	public boolean equals(Object element)	matches two collection.
14.	public int hashCode()	returns the hashcode number for collection.
15.	public Object get(int index)	Return element at index

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

public boolean hasNext() it returns true if iterator has more elements.
public object next() it returns the element and moves the cursor pointer to the next element.
public void remove() it removes the last elements returned by the iterator. It is rarely used.

Java Array List

Java Array List class uses a dynamic array for storing the elements. It extends Abstract List class and implements List interface. Java Array List class can contain duplicate elements .Java Array List class maintains insertion order. Java Array List class is non synchronized.
Java Array List allows random access because array works at the index basis. In Java Array List class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic. Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.
Array List al = new Array List();//creating old non-generic Array List

non-generic will take any object :
we can add any object to array list:
al.add(employee);
al.add(student);
al.add(customer);
Let's see the new generic example of creating java collection.
Array List <String> al = new Array List <String> ();//creating new generic array-list

In generic collection, we specify the type in angular braces. Now Array List is forced to have only specified type of objects in it. If you try to add another type of object, it gives compile time error.

Example of Java ArrayList class

```
package com.nrit.mnrao.test;
import java.util.ArrayList;
import java.util.Iterator;
public class Test {
	public static void main(String[] args) {
		ArrayList < Integer > al = new ArrayList< Integer > ();
		al.add(new Integer(10));
		al.add(20);  // auto boxing , basic converting into object
		al.add(30);
		al.add(40);
		al.add(50);
		al.add(60);
		al.add(70);
		// reading arryalist elements	//1st way
		System.out.println("array elements "+al);
		System.out.println();
		//2nd way
		for(int i=0; i<al.size() ; i++)  {
			System.out.print(al.get(i)+"\t");
		}
		System.out.println();
		//3rd way
		for (Integer i : al) {
			System.out.print(i+"\t");
		}
		System.out.println();
		//4th way
		Iterator <Integer> iterator = al.iterator();
		while(iterator.hasNext())  {
			Integer nextInteger = iterator.next();
			System.out.print(nextInteger+"\t");
		} } }
package com.nrit.mnrao.test;
import java.util.ArrayList;
import java.util.Iterator;
public class Test {
	public static void main(String[] args) {
		ArrayList <String> al  = new	ArrayList <String> ();
		System.out.println(" no of elements : "+al.size());
		if(al.isEmpty())  {
			System.out.println("empty");
		}
```

```
		else {
			System.out.println("Not empty");
		}
		al.add("java");
		al.add("hadoop");
		System.out.println(" no of elements : "+al.size());
		System.out.println("1st element "+al.get(0));
		System.out.println("last element "+al.get(al.size()-1));
		// to insert element at required index.
		al.add(0, "oracle");
		System.out.println(al);
		al.add(2, "html");
		System.out.println(al);
		String str = al.remove(0);// physically removes
		System.out.println(str);
		System.out.println(al);
		String str1 = al.get(0);// read only, will not remove physically
		System.out.println(str1);
		System.out.println(al);
		System.out.println(al.size());
		if(al.isEmpty()){
			System.out.println("empty ");
		}
		else {
			System.out.println(" not empty ");
		}
		al.clear();
		if(al.isEmpty()){
			System.out.println("empty ");
		}
		else{
			System.out.println(" not empty ");
		} } }
```

Another example

```
package com.nrit.mnrao.test;
import java.util.ArrayList;
import java.util.Iterator;
public class Test {
	public static void main(String[] args) {
		ArrayList <String> al1 = new ArrayList <String> ();
		al1.add("hive");
		al1.add("sqoop");
		al1.add("pig");
		al1.add("hbase");
		al1.add("mapper");
		ArrayList <String> al2 = new ArrayList <String> ();
		al2.add("java");
		al2.add("jsp");
		al2.add("servlet");
		al2.add("html");
		al2.add("spring");
		al2.addAll(al1);
		System.out.println(al2);
		if(al2.containsAll(al1)){
			System.out.println("contains all");
		}
		else{
			System.out.println("no contains");
		}
		al2.remove("hive");
		if(al2.containsAll(al1)){
			System.out.println("contains all");
		}
		else{
			System.out.println("not contain");
```

```
		}
		if( al2.contains("java") && al2.contains("hadoop")){
			System.out.println("Yes");
		}
		else{
			System.out.println("no");
		}
		ArrayList <String> al3 = (ArrayList <String> )al2.clone();
		System.out.println(al2);
		System.out.println(al3);
	}  }
import java.util.ArrayList;
import java.util.Iterator;
public class TestCollection{
	public static void main(String []  args){
		ArrayList<String> al = new ArrayList<String>();// creating arraylist
		al.add("Ravi");			// adding object in arraylist
		al.add("Vijay");
		al.add("Ravi");
		al.add("Ajay");
		Iterator itr = al.iterator();// getting Iterator from arraylist to
		// traverse elements
		while (itr.hasNext()) {
			System.out.println (itr.next());
		} } }
```

O/p:

Ravi Vijay
Ravi Ajay

Iterating the elements of Collection by for-each loop

```
 import java.util.ArrayList;
public class TestCollection {
	public static void main(String []  args) {
		ArrayList<String> al = new ArrayList<String>();
		al.add("Ravi");
		al.add("Vijay");
		al.add("Ravi");
		al.add("Ajay");
		for (String obj : al)
	System.out.println (obj);
	}   }
```

O/p:

Ravi Vijay
Ravi Ajay

User-defined class objects in Java ArrayList

```
package com.durga.mnrao.xyz;
public class Employee {
	private int empNum;
	private String empName;
	private String empJob;
	private double empSalary;
	private String empDeptName;
	private String empGender;
	private int empAge;
	public int getEmpNum() {
		return empNum;
	}
	public void setEmpNum(int empNum) {
		this.empNum = empNum;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
```

```
	public String getEmpJob() {
		return empJob;
	}
	public void setEmpJob(String empJob) {
		this.empJob = empJob;
	}
	public double getEmpSalary() {
		return empSalary;
	}
	public void setEmpSalary(double empSalary) {
		this.empSalary = empSalary;
	}
	public String getEmpDeptName() {
		return empDeptName;
	}
	public void setEmpDeptName(String empDeptName) {
		this.empDeptName = empDeptName;
	}
	public String getEmpGender() {
		return empGender;
	}
	public void setEmpGender(String empGender) {
		this.empGender = empGender;
	}
	public int getEmpAge() {
		return empAge;
	}
	public void setEmpAge(int empAge) {
		this.empAge = empAge;
	}   }
package com.durga.mnrao.xyz;
import java.util.ArrayList;
import java.util.Iterator;
public class Test {
	public static void main(String[] args)  {
		String [] records =  {
	"1001,ajay,manager,account,45000,male,38",
	"1002,aiswrya,clerk,admin,25000,female,30",
	"1003,varun,manager,sales,50000,male,35",
	"1004,amit,manager,account,47000,male,40",
	"1005,kareena,executive,sales,15000,female,24",
	"1006,deepak,clerk,admin,23000,male,30",
	"1007,sunil,accountant,sales,13000,male,29",
	"1008,satvik,director,purchase,80000,male,45"		};
	ArrayList<Employee> list = new ArrayList<Employee>();
	for(String record: records) {
			String[] fields = record.split(",");
			Employee employee = new Employee();
			employee.setEmpNum(Integer.parseInt(fields[0]));
			employee.setEmpName(fields[1]);
			employee.setEmpJob(fields[2]);
			employee.setEmpDeptName(fields[3]);
			employee.setEmpSalary(Double.parseDouble(fields[4]));
			employee.setEmpGender(fields[5]);
			employee.setEmpAge(Integer.parseInt(fields[6]));
			list.add(employee);
		}
		Iterator<Employee> iterator = list.iterator();
		while(iterator.hasNext()){
			Employee emp = iterator.next();
			int empNum = emp.getEmpNum();
			String empName = emp.getEmpName();
			String empJob = emp.getEmpJob();
			double empSalary = emp.getEmpSalary();
			String empDeptName = emp.getEmpDeptName();
```

```
                String empGender = emp.getEmpGender();
                int empAge = emp.getEmpAge();
System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"\t"+empAge);
        } } }
```

Java LinkedList class

Java LinkedList class uses doubly linked list to store the elements. It extends the Abstract List class and implements List and Deque interfaces. Java LinkedList class can contain duplicate elements. Java LinkedList class maintains insertion order. Java LinkedList class is non synchronized. In Java LinkedList class, manipulation is fast because no shifting needs to be occurred. Java LinkedList class can be used as list, stack or queue.

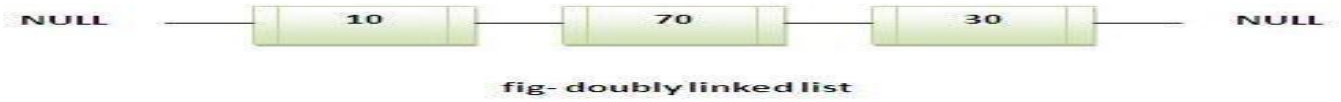


fig- doubly linked list

Java LinkedList Example

```
import java.util.*;
public class Test  {
 public static void main(String []  args)  {
        LinkedList<String> al=new LinkedList<String>();
        al.add("Ravi");
        al.add("Vijay");
       al.add("Ravi");
       al.add("Ajay");
       Iterator<String> itr=al.iterator();
        while(itr.hasNext()) {
                System.out.println (itr.next());
       } }  }
```

Output:

Ravi Vijay Ravi Ajay

Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes. But there are many differences between Array-List and LinkedList classes that are given below.

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

Example of ArrayList and LinkedList in Java

Let's see a simple example where we are using ArrayList and LinkedList both.

```
import java.util.LinkedList;
import java.util.List;
import java.util.ArrayList;
public class Test  {
        public static void main(String []  args) {
        List<String> al = new ArrayList<String>();
                al.add("Ravi");
                al.add("Vijay");
                al.add("Ravi");
                al.add("Ajay");
                List<String> al2 = new LinkedList<String>();
                al2.add("James");
                al2.add("Serena");
                al2.add("Swati");
                al2.add("Junaid");
                System.out.println ("arraylist: " + al);
                System.out.println ("linkedlist: " + al2);
        }  }  Output:
arraylist: [Ravi,Vijay,Ravi,Ajay]
linkedlist: [James,Serena,Swati,Junaid]
```

Vector:

The **java.util.Vector** class implements a growable array of objects. Similar to an Array, it contains components that can be accessed using an integer index. The size of a Vector can grow or shrink as needed to accommodate adding and removing items.

Vector is synchronized.

. Class declaration

Following is the declaration for **java.util.Vector** class:

```
public class Vector<E>
  extends AbstractList<E>
  implements List<E>, RandomAccess, Cloneable, Serializable
```

Constructor & Description

Vector()

This constructor is used to create an empty vector so that its internal data array has size 10 and its standard capacity increment is zero.

Vector(Collection<? extends E> c)

This constructor is used to create a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Vector(int initialCapacity)

This constructor is used to create an empty vector with the specified initial capacity and with its capacity increment equal to zero.

Vector(int initialCapacity, int capacityIncrement)

This constructor is used to create an empty vector with the specified initial capacity and capacity increment.

Method & Description

Method & Description
boolean add(E e)
This method appends the specified element to the end of this Vector.
void add(int index, E element)

This method inserts the specified element at the specified position in this Vector

boolean addAll(Collection<? extends E> c)

This method appends all of the elements in the specified Collection to the end of this Vector.

boolean addAll(int index, Collection<? extends E> c)

This method inserts all of the elements in the specified Collection into this Vector at the specified position.

void addElement(E obj)

This method adds the specified component to the end of this vector, increasing its size by one.

int capacity()

This method returns the current capacity of this vector.

void clear()

This method removes all of the elements from this vector.

Clone clone()

This method returns a clone of this vector.

boolean contains(Object o)
This method returns true if this vector contains the specified element.

boolean containsAll(Collection<?> c)
This method returns true if this Vector contains all of the elements in the specified Collection.

void copyInto(Object[ ] anArray)
This method copies the components of this vector into the specified array.
E elementAt(int index)
This method returns the component at the specified index.
Enumeration<E> elements()
This method returns an enumeration of the components of this vector.
void ensureCapacity(int minCapacity)

This method increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

boolean equals(Object o)

This method compares the specified Object with this Vector for equality.

E firstElement()

This method returns the first component (the item at index 0) of this vector.

E get(int index)

This method returns the element at the specified position in this Vector.

int hashCode()
This method returns the hash code value for this Vector.
int indexOf(Object o)

This method returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

int indexOf(Object o, int index)

This method returns the index of the first occurrence of the specified element in this vector, searching forwards from index, or returns -1 if the element is not found.

void insertElementAt(E obj, int index)

This method inserts the specified object as a component in this vector at the specified index.

boolean isEmpty()

This method tests if this vector has no components.

E lastElement()

This method returns the last component of the vector.

int lastIndexOf(Object o)

This method returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

int lastIndexOf(Object o, int index)

This method returns the index of the last occurrence of the specified element in this vector, searching backwards from index, or returns -1 if the element is not found.

E remove(int index)

This method removes the element at the specified position in this Vector.

boolean remove(Object o)
This method removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.
boolean removeAll(Collection<?> c)
This method removes from this Vector all of its elements that are contained in the specified Collection.
void removeAllElements()

This method removes all components from this vector and sets its size to zero.

boolean removeElement(Object obj)

This method removes the first occurrence of the argument from this vector.

void removeElementAt(int index)

This method deletes the component at the specified index.

boolean retainAll(Collection<?> c)

This method retains only the elements in this Vector that are contained in the specified Collection.

E set(int index, E element)

This method replaces the element at the specified position in this Vector with the specified element.

void setElementAt(E obj, int index)

This method sets the component at the specified index of this vector to be the specified object.

void setSize(int newSize)

This method sets the size of this vector.

i**nt size()**

This method returns the number of components in this vector.

List <E> subList(int fromIndex, int toIndex)

This method returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.

object[ ] toArray()

This method returns an array containing all of the elements in this Vector in the correct order.

<T> T[ ] toArray(T[ ] a)

This method returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array.

String toString()

This method returns a string representation of this Vector, containing the String representation of each element.

void trimToSize()
This method trims the capacity of this vector to be the vector's current size.

Program: Basic Vector Operations.

```
package com.nrit.mnrao.test;
import java.util.Vector;
public class BasicVectorOperations {
        public static void main(String a[]) {
                Vector<String> vct = new Vector<String>();
                vct.add("First");
                vct.add("Second");
                vct.add("Third");
                System.out.println(vct);
                // adding element at specified index
                vct.add(2, "Random");
                System.out.println(vct);
                // getting elements by index
                System.out.println("Element at index 3 is: " + vct.get(3));
                // getting first element
                System.out.println("The first element of this vector is: " +vct.firstElement());
                // getting last element
                System.out.println("The last element of this vector is: " + vct.lastElement());
                // how to check vector is empty or not
                System.out.println("Is this vector empty? " + vct.isEmpty());
        }   }
```

Output:
[First, Second, Third]
[First, Second, Random, Third]
Element at index 3 is: Third
The first element of this vector is: First
The last element of this vector is: Third
Is this vector empty? False

Program: How to read all elements in vector by using iterator?

```
package com.nrit.mnrao.test;
import java.util.Iterator;
import java.util.Vector;
public class VectorIterator {
        public static void main(String a[]) {
                Vector<String> vct = new Vector<String>();
                // adding elements to the end
                vct.add("First");
                vct.add("Second");
                vct.add("Third");
                vct.add("Random");
                Iterator<String> itr = vct.iterator();
                while (itr.hasNext()) {
        System.out.println(itr.next());
                } } }
```

Output:
First Second
Third Random
Program: How to read all elements in vector by using Enumeration?

```
package com.nrit.mnrao.test;
```

```
import java.util.Enumeration;
import java.util.Vector;
public class VectorEnnumaratio {
public static void main(String a[]) {
        Vector<String> vct = new Vector<String>();
                // adding elements to the end
                vct.add("First");
                vct.add("Second");
                vct.add("Third");
                vct.add("Random");
                Enumeration<String> enm = vct.elements();
                while (enm.hasMoreElements()) {
                        System.out.println(enm.nextElement());
                } } }
```

Program: How to copy or clone a vector?

```
package com.nrit.mnrao.test;
import java.util.Vector;
public class MyVectorClone {
  public static void main(String a[]){
    Vector<String> vct = new Vector<String>();
    //adding elements to the end
    vct.add("First");
    vct.add("Second");
    vct.add("Third");
    vct.add("Random");
    System.out.println("Actual vector:"+vct);
    Vector<String> copy = (Vector<String>) vct.clone();
    System.out.println("Cloned vector:"+copy);
  } }
```

Program: How to add all elements of a list to vector?

```
package com.nrit.mnrao.test;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
public class MyVectorNewCollection {
        public static void main(String a[]) {
                Vector<String> vct = new Vector<String>();
                // adding elements to the end
                vct.add("First");
                vct.add("Second");
                vct.add("Third");
                vct.add("Random");
                System.out.println("Actual vector:" + vct);
                List<String> list = new ArrayList<String>();
                list.add("one");
                list.add("two");
                vct.addAll(list);
                System.out.println("After Copy: " + vct);
        } }
```

Program: How to delete all elements from my vector?

```
package com.nrit.mnrao.test;
import java.util.Vector;
public class ClearMyVector {
  public static void main(String a[]){
    Vector<String> vct = new Vector<String>();
    //adding elements to the end
    vct.add("First");
    vct.add("Second");
    vct.add("Third");
    vct.add("Random");
    System.out.println("Actual vector:"+vct);
```

```
    vct.clear();
    System.out.println("After clear vector:"+vct);
  }  }
```

Program: How to find does vector contains all list elements or not?

```
package com.nrit.mnrao.test;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
public class MyElementCheck {
        public static void main(String a[]) {
                Vector<String> vct = new Vector<String>();
                vct.add("First");
                vct.add("Second");
                vct.add("Third");
                vct.add("Random");
                List<String> list = new ArrayList<String>();
                list.add("Second");
                list.add("Random");
        System.out.println("Does vector contains all list elements?: " + vct.containsAll(list));
                list.add("one");
        System.out.println("Does vector contains all list elements?: " + vct.containsAll(list));
        }   }
```

Program: How to copy vector to array?

```
package com.nrit.mnrao.test;
import java.util.Vector;
public class MyVectorArrayCopy {
        public static void main(String a[]) {
                Vector<String> vct = new Vector<String>();
                vct.add("First");
                vct.add("Second");
                vct.add("Third");
                vct.add("Random");
                System.out.println("Actual vector:" + vct);
                String  []copyArr = new String[vct.size()];
                vct.copyInto(copyArr);
                System.out.println("Copied Array content:");
                for (String str : copyArr) {
                System.out.println(str);
                } } }
```

Program: How to get sub list from vector?

```
package com.nrit.mnrao.test;
import java.util.List;
import java.util.Vector;
public class MyVectorSubRange {
        public static void main(String a[]) {
                Vector<String> vct = new Vector<String>();
                // adding elements to the end
                vct.add("First");
                vct.add("Second");
                vct.add("Third");
                vct.add("Random");
                vct.add("Click");
                System.out.println("Actual vector:" + vct);
                List<String> list = vct.subList(2, 4);
                System.out.println("Sub List: " + list);
        }   }
```

Difference between ArrayList and Vector:

ArrayList and Vector both implements List interface and maintains insertion order.

S.No.	ArrayList	Vector
1.	ArrayList is not synchronized.	Vector is synchronized.

2.	ArrayList increments 50% of current array size if number of element exceeds from its capacity.	Vector increments 100% means doubles the array size if total number of element exceeds than its capacity.
3.	ArrayList is not a legacy class, it is introduced in JDK 1.2.	Vector is a legacy class.
4.	ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.
5.	ArrayList uses Iterator interface to traverse the elements.	Vector uses Enumeration interface to traverse the elements. But it can use Iterator also.

Stack Class

Stack is a subclass of **Vector** that implements a standard last-in, first-out stack. **Stack** only defines the default constructor, which creates an empty stack. **Stack** includes all the methods defined by **Vector**, and adds several of its own.

To put an object on the top of the stack, call **push()**. To remove and return the top element, call **pop()**. An **EmptyStackException** is thrown if you call **pop( )** when the invoking stack is empty. You can use **peek( )** to return, but not remove, the top object.

The **empty()** method returns **true** if nothing is on the stack. The **search()** method determines whether an object exists on the stack, and returns the number of pops that are required to bring it to the top of the stack. Here is an example that creates a stack, pushes several **Integer** objects onto it, and then pops them off again:

The **java.util.Stack** class represents a last-in-first-out (LIFO) stack of objects.

- When a stack is first created, it contains no items.
- In this class, the last element inserted is accessed first.

Class declaration

Method & description boolean empty()

This method tests if this stack is empty.

peek()

This method looks at the object at the top of this stack without removing it from the stack.

pop()

This method removes the object at the top of this stack and returns that object as the value of this function. It stack is empty it throws **"EmptyStackException"**

push(E item)

This method pushes an item onto the top of this stack.

int search(Object o)
This method returns the 1-based position where an object is on this stack.

Following is the declaration for **java.util.Stack** class:

```
public class
Stack<E>
  extends Vector<E>
```

Constructor & Description

Stack()

This constructor creates an empty stack.

```
package com.nrit.mnrao.test;
import java.util.Stack;
public class Test {
	public static void main(String[] args) {
		Stack <String> bookRack = new Stack<String>();
		bookRack.push("HADOOP");
		bookRack.push("JAVA");
		bookRack.push("ORACLE");
		bookRack.push("C");
		bookRack.push("LINUX");
		System.out.println("books in the Rack "+bookRack);
		System.out.println("Top book : "+bookRack.peek());
		String book = bookRack.pop();
		System.out.println("Removed Book  "+book);
		 book = bookRack.pop();
		System.out.println("Removed Book  "+book);
		System.out.println("Current books in the Rack "+bookRack);		} }
```

Program is to remove stack elements one by one.

```
package com.nrit.mnrao.test;
import java.util.Stack;
public class Test {
	public static void main(String[] args) {
		Stack<String> bookRack = new Stack<String>();
		bookRack.push("HADOOP");
		bookRack.push("JAVA");
		bookRack.push("ORACLE");
		bookRack.push("C");
		bookRack.push("LINUX");
		while( !  bookRack.isEmpty() ){
			String book = bookRack.pop();
			System.out.println(book);
		} } }
```

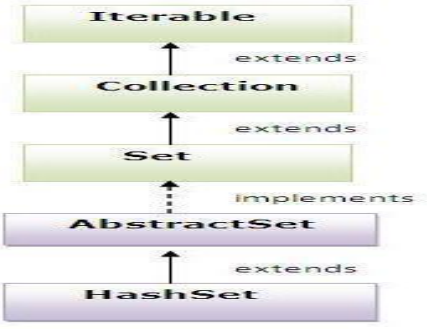
Java HashSet class

uses hash table to store the elements. It extends Abstract Set class and implements Set interface. contains unique elements only.

Difference between List and Set:

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class:



Example of HashSet class:

```
import java.util.Iterator;
import java.util.HashSet;
public class Test{
	public static void main(String []  args){
		HashSet<String> al = new HashSet<String>();
		al.add("Ravi");
		al.add("Vijay");
		al.add("Ravi");
		al.add("Ajay");
		Iterator<String> itr = al.iterator();
		while (itr.hasNext()) {
			System.out.println (itr.next());
		} } }
```

```
Vijay	Ajay
Ravi
```

TreeSet class:

TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order. Contains unique elements only like HashSet.

Access and retrieval times are quiet fast. Maintains ascending order.

```
import java.util.Iterator;
import java.util.TreeSet;
public class Test{
	public static void main(String []  args){
```

```
// Creating and adding elements
        TreeSet<String> al = new TreeSet<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");
        // Traversing elements
        Iterator<String> itr = al.iterator();
        while (itr.hasNext()){
                System.out.println (itr.next());
        } } }
```

Java Map Interface

A map contains values based on the key i.e. key and value pair. Each pair is known as an entry. Map contains only unique elements.

Commonly used methods of Map interface:

➢ **public Object put(object key,Object value):**	is used to insert an entry in this map.
➢ **public void putAll(Map map):**	is used to insert the specified map in this map.
➢ **public Object remove(object key):**	is used to delete an entry for the specified key.
➢ **public Object get(Object key):**	is used to return the value for the specified key.
➢ **public boolean containsKey(Object key):**	is used to search the specified key from this map.
➢ **public boolean containsValue(Object value):**	is used to search the specified value from this map.
➢ **public Set keySet():**	is returns the Set view containing all the keys.
➢ **public Set entrySet():**	is returns the Set view containing all the keys and values.

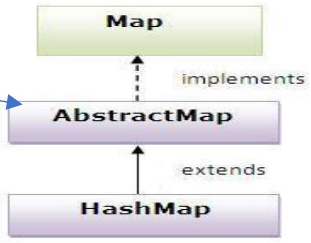
Java HashMap class

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

Difference between HashSet and HashMap

HashSet contains only values whereas HashMap contains entry(key and value).

Hierarchy of HashMap class:



Non generic Example for HashMap (any key and any value)

```
import java.util.HashMap;
import java.util.Stack;
public class Test {
        public static void main(String[] args) {
                HashMap  hmap = new HashMap();
                hmap.put(1, "java");
                hmap.put("hello", 1);
                hmap.put(2, new Employee());
                System.out.println(hmap);    }  }
```

Generic HashMap Example:

```
import java.util.HashMap;
import java.util.Stack;
public class Test {
        public static void main(String[] args) {
                HashMap<Integer, String> hmap = new HashMap<Integer, String> ();
                hmap.put(1, "java");
                hmap.put(2,"hadoop");
                hmap.put(3, "oracle");
                hmap.put(4, "unix");
                System.out.println(hmap);
        } }
```

Using keyset iterator :

HashMap:

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;
public class Test {
        public static void main(String[] args)  {
                HashMap<String, String> hmap = new HashMap<String, String>();
                hmap.put("j", "java");
                hmap.put("h", "hadoop");
                hmap.put("p", "python");
                hmap.put("o", "oracle");
                hmap.put("u", "unix");
```

```
		hmap.put("l", "linux");
		System.out.println(hmap);
		Set<String> keySet = hmap.keySet();
		Iterator<String> iterator = keySet.iterator();
		while( iterator.hasNext()) {
			String key = iterator.next();
			String value = hmap.get(key);
			System.out.println(key+"\t"+value);
		} } }
```

Hashtable:

```
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
public class Sample {
	public static void main(String[] args) {
		Hashtable<String, String>  htable = new Hashtable<String, String>();
		htable.put("j", "java");
		htable.put("h", "hadoop");
		htable.put("p", "python");
		htable.put("o", "oracle");
		htable.put("u", "unix");
		htable.put("l", "linux");
		System.out.println(htable);
		Set<String> keySet = htable.keySet();
		Iterator<String> iterator = keySet.iterator();
		while( iterator.hasNext() ) {
			String key = iterator.next();
			String value = htable.get(key);
			System.out.println(key+"\t"+value);
		}  } }
```

Entrset

HashMap:

```
package com.durga.mnrao.xyz;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.Set;
public class Test {
	public static void main(String[] args)  {
		HashMap<String, String> hmap = new HashMap<String, String>();
		hmap.put("j", "java");
		hmap.put("h", "hadoop");
		hmap.put("p", "python");
		hmap.put("o", "oracle");
		hmap.put("u", "unix");
		hmap.put("l", "linux");
		System.out.println(hmap);
		Set<Entry<String, String>> entrySet = hmap.entrySet();
		Iterator<Entry<String, String>> iterator = entrySet.iterator();
		while( iterator.hasNext() )  {
			Entry<String, String> entry = iterator.next();
			String key = entry.getKey();
			String value = entry.getValue();
			System.out.println(key+"\t"+value);
		} } } }
```

Hashtable

```
package com.durga.mnrao.xyz;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
import java.util.Map.Entry;
public class Sample {
	public static void main(String[] args) {
	Hashtable<String, String>  htable = new Hashtable<String, String>();
		htable.put("j", "java");
```

```
htable.put("h", "hadoop");
htable.put("p", "python");
htable.put("o", "oracle");
htable.put("u", "unix");
htable.put("l", "linux");
System.out.println(htable);
Set<Entry<String, String>> entrySet = htable.entrySet();
Iterator<Entry<String, String>> iterator = entrySet.iterator();
while( iterator.hasNext() )  {
        Entry<String, String> entry = iterator.next();
        String key = entry.getKey();
        String value = entry.getValue();
        System.out.println(key+"\t"+value);
} } }
```

Replacing element value

```
package com.nrit.mnrao.test;
import java.util.HashMap;
public class Test {
    public static void main(String[] args) {
        HashMap<Integer, String> hmap = new HashMap<Integer, String>();
        hmap.put(1, "java");
        hmap.put(2, "hadoop");
        hmap.put(3, "oracle");
        hmap.put(4, "unix");
        hmap.put(5, "linux");
        System.out.println(hmap);
        if(hmap.containsKey(4) )  {
            if(hmap.replace(4, "unix", "linux")) {
                System.out.println("successfully replaced");
            }
            else {
                System.out.println("failed to  replace");
            } }
        System.out.println(hmap);
    }   }
```

Database properties

```
import java.util.HashMap;
public class Test{
    public static void main(String[] args) {
HashMap<String,String> dataBaseproperties =new HashMap<String,String>();
         dataBaseproperties.put("user", "demo");
        dataBaseproperties.put("path", "/home/demo");
        dataBaseproperties.put("host", "160.10.20.3");
        dataBaseproperties.put("dbname", "student");
        dataBaseproperties.put("uid", "root");
        dataBaseproperties.put("pwd", "admin123");
        String uidValue = dataBaseproperties.get("uid");
        String pwdValue = dataBaseproperties.get("pwd");
        String pathValue = dataBaseproperties.get("path");
        String userValue = dataBaseproperties.get("user");
        System.out.println (uidValue);
        System.out.println (pwdValue);
        System.out.println (pathValue);
        System.out.println (userValue);
        System.out.println (dataBaseproperties.toString());
    } }
```

Hashtable class:

Hashtable class implements a Map interface, which maps keys to values. It inherits Dictionary class and implements the Map interface.

A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hash table contains values based on the key.

It contains only unique elements. It may have not have any null key or value. It is synchronized

```
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;
```

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

Difference between HashMap and Hashtable:

S.No	HashMap	Hashtable
1.	HashMap is non synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is synchronized. It is thread-safe and can be shared with many threads.
2.	HashMap allows one null key and multiple null values.	Hashtable doesn't allow any null key or value.
3.	HashMap is a new class introduced in JDK 1.2.	Hashtable is a legacy class.
4.	HashMap is fast.	Hashtable is slow.
5.	We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6.	HashMap is traversed by Iterator.	Hashtable is traversed by Enumerator and Iterator.
7.	Iterator in HashMap is fail-fast.	Enumerator in Hashtable is not fail-fast.
8.	HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.

HashMap	HashTable
Key and value	Key and value
Key is unique value	Key is unique value
Value can be duplicate	Value can be duplicate
Null key allowed but only one null allowed	Key should not be null
Any no of null values allowed	Value also should not null
Non Synchronized	Synchronized
Not a thread safe	Thread safe

Collections class:

collection class is used exclusively with static methods that operate on or return collections. It inherits Object class.
Java Collection class supports the polymorphic algorithms that operate on collections.
Java Collection class throws a NullPointerException if the collections or class objects provided to them are null.

Collections class declaration

public class Collections extends Object

Methods of Java Collections class

Method	Description
static <T> boolean addAll(Collection<? super T> c, T... elements)	It is used to add all of the specified elements to the specified collection.
static <T> Queue<T> asLifoQueue(Deque<T> deque)	It is used to return a view of a Deque as a Last-In-First-Out (LIFO) Queue.
static <T> int binarySearch(List<? extends T> list, T key, Comparator<? super T< c)	It is used to search the specified list for the specified object using the binary search algorithm.
static <E> List<E> checkedList(List<E> list, Class<E> type)	It is used to return a dynamically typesafe view of the specified list.
static <E> Set<E> checkedSet(Set<E> s, Class<E> type)	It is used to return a dynamically typesafe view of the specified set.
static<E> SortedSet<E>checkedSortedSet(SortedSet<E> s, Class<E> type)	It is used to return a dynamically typesafe view of the specified sorted set
static void reverse(List<?> list)	It is used to reverse the order of the elements in the specified list.
static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp)	It is used to return the maximum element of the given collection, according to the

	order induced by the specified comparator.
static <T extends Object & Comparable<? super T>>T min(Collection<? extends T> coll)	It is used to return the minimum element of the given collection, according to the natural ordering of its elements.
static boolean replaceAll(List list, T oldVal, T newVal)	It is used to replace all occurrences of one specified value in a list with another.

Collections Example:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Test{
        public static void main(String a[]) {
                List<String> list = new ArrayList<String>();
                list.add("C");
                list.add("Core Java");
                list.add("Advance Java");
                System.out.println ("Initial collection value:" + list);
                Collections.addAll(list, "Servlet", "JSP");
System.out.println ("After adding elements collection value:" + list);
                String[] strArr = { "C#", ".Net" };
                Collections.addAll(list, strArr);
System.out.println ("After adding array collection value:" + list);
        }    }
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Test {
        public static void main(String a[]) {
                List<Integer> list = new ArrayList<Integer>();
                list.add(46);
                list.add(67);
                list.add(24);
                list.add(16);
                list.add(8);
                list.add(12);
System.out.println ("Value of maximum element from the collection: " + Collections.max(list));
        }
}
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Test{
        public static void main(String a[]){
          List<Integer> list = new ArrayList<Integer>();
          list.add(46);
          list.add(67);
          list.add(24);
          list.add(16);
          list.add(8);
          list.add(12);
 System.out.println ("Value of minimum element from the collection: "+Collections.min(list));
  }    }
```

Sorting in Collection:

We can sort the elements of:

String objects Wrapper class objects User-defined class objects Collections class provides static methods for sorting the elements of collection. If collection elements are of Set type, we can use TreeSet. But We cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

Method of Collections class for sorting List elements:

public void sort(List list): is used to sort the elements of List.List elements must be of Comparable type.

Example of Sorting the elements of List that contains string objects:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
public class Test{
        public static void main(String []  args) {
```

```
ArrayList<String> al = new ArrayList<String>();
al.add("Viru");
al.add("Saurav");
al.add("Mukesh");
al.add("Tahir");
Collections.sort(al);
Iterator itr = al.iterator();
while (itr.hasNext()) {
        System.out.println (itr.next());
}   }  }
```

Example of Sorting the elements of List that contains Wrapper class objects:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
public class Test{
    public static void main(String []  args) {
        ArrayList  al = new ArrayList();
        al.add(Integer.valueOf(201));
        al.add(Integer.valueOf(101));
        al.add(230);// internally will be converted into objects as
        // Integer.valueOf(230)
        Collections.sort(al);
        Iterator itr = al.iterator();
        while (itr.hasNext()) {
            System.out.println (itr.next());
        }   } }
```

Properties class in Java

The properties object contains key and value pair both as a string. It is the subclass of Hashtable.

It can be used to get property value based on the property key. The Properties class provides methods to get data from properties file and store data into properties file. Moreover, it can be used to get properties of system.

Advantage of properties file

Easy Maintenance: If any information is changed from the properties file, you don't need to recompile the java class. It is mainly used to contain variable information i.e. to be changed.

Methods of Properties class

The commonly used methods of Properties class are given below.

Method	Description
public void load(Reader r)	loads data from the Reader object.
public void load(InputStream is)	loads data from the InputStream object
public String getProperty(String key)	returns value based on the key.
public void setProperty(String key,String value)	sets the property in the properties object.
public void store(Writer w, String comment)	writers the properties in the writer object.
public void store(OutputStream os, String comment)	writes the properties in the OutputStream object.
storeToXML(OutputStream os, String comment)	writers the properties in the writer object for generating xml document.
public void storeToXML(Writer w, String comment, String encoding)	writers the properties in the writer object for generating xml document with specified encoding.

Example of Properties class to get information from properties file

To get information from the properties file, create the properties file first.

db.properties file

```
==========
  user=system
  password=oracle
```

Test.java

```
  import java.util.*;
  import java.io.*;
  public class Test   {
  public static void main(String[] args)throws Exception  {
        FileReader reader=new FileReader("db.properties");
        Properties   p=new Properties();
         p.load(reader);
        System.out.println (p.getProperty("user"));
```

```
        System.out.println (p.getProperty("password"));
      }
  }
```

Now if you change the value of the properties file, you don't need to compile the java class again. That means no maintenance problem.

Input and Output streams

Java I/O (Input and Output) is used to process the input and produce the output based on the input. Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
We can perform file handling in java by java IO API.

Stream:

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow.

In java, 3 streams are created for us automatically. All these streams are attached with console.

1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream (System errors i.e JVM error message)

Let's see the code to print output and error message to the console.

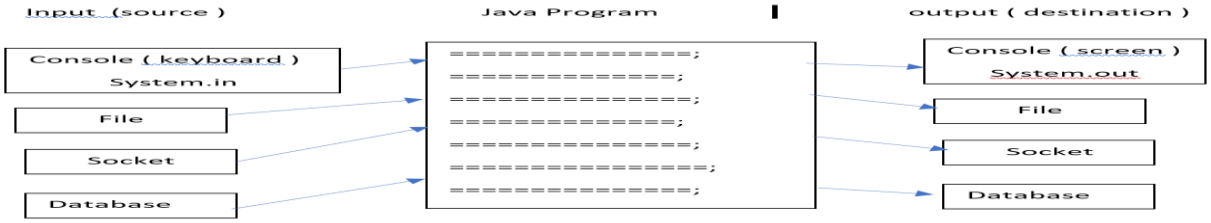
```
System.out.println ("simple message");
System.err.println ("error message");
int i=System.in.read();//returns ASCII code of 1st character .
System.out.println ((char)i);//will print the character .
```

InputStream (Source of data)

Java application uses an input stream to read data from a source, it may be a file, an array, peripheral device or socket.

OutputStream (Destination of Data)

Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.



Character Stream Vs Byte Stream in Java

Byte Streams

A byte stream access the file byte by byte. Java programs use byte streams to perform input and output of 8-bit bytes. It is suitable for any kind of file, however not quite appropriate for text files. For example, if the file is using a unicode encoding and a character is represented with two bytes, the byte stream will treat these separately and you will need to do the conversion yourself. Byte oriented streams do not use any encoding scheme while Character oriented streams use character encoding scheme(UNICODE). All byte stream classes are descended from InputStream and OutputStream .

Character Streams

A character stream will read a file character by character. Character Stream is a higher level concept than Byte Stream . A Character Stream is, effectively, a Byte Stream that has been wrapped with logic that allows it to output characters from a specific encoding . That means, a character stream needs to be given the file's encoding in order to work properly. Character stream can support all types of character sets ASCII, Unicode, UTF-8, UTF-16 etc. All character stream classes are descended from Reader and Writer .

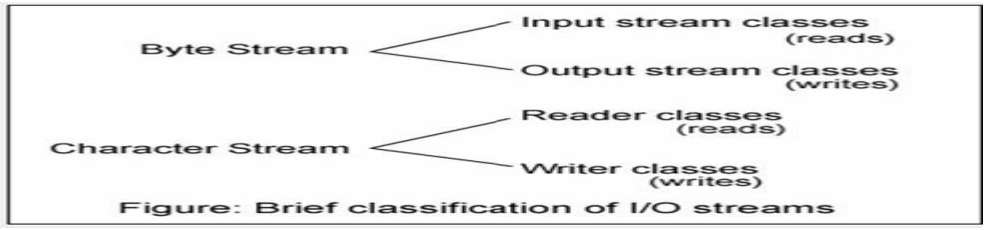
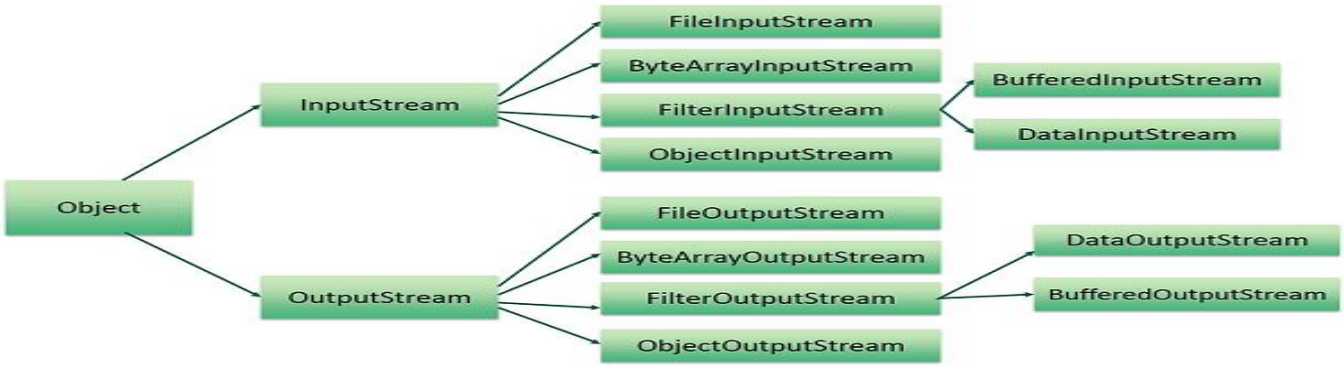


Figure: Brief classification of I/O streams

Hierarchy of Byte Stream classes.



OutputStream class

OutputStream class is an abstract class.It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Commonly used methods of OutputStream class

1) public void write(int) throws IOException: is used to write a byte to the required output stream.

2) public void write(byte []) throws IOException: is used to write an array of byte to the required output stream.

3) public void flush() throws IOException: flushes the required output stream.
4) public void close()throws IOException: is used to close the required output stream.

InputStream class

InputStream class is an abstract class.It is the superclass of all classes representing an input stream of bytes.
1) public abstract int read() throws IOException:
reads the next byte of data from the input stream. It returns -1 at the end of file

2) public int available()throws IOException
returns an estimate of the number of bytes that can be read from the required input stream.

3) public void close()throws IOException:
is used to close the required input stream.

File Class :

It is class from java.io package to get information about file attributes/properties.
File (String filePath) : constructor to create file object
File file = new File("C:\\project\\test\\file1"); windows
File file = new File("/home/nrit/file1"); linux
Since java is an independent of plat form we should write common code for both OS.
File file = new File("c:/project/test/file1");

Use / as path separator for any OS.
Eg:
File file = new File("C:/project/test/EMPLOYEE.dat");
file.exists() : checking for existence of the file.
file.isFile() : checking for file or not
file.isDirectory() : checking for direcoty or not
file.canRead() : checking for read permission
file.canWrite() : checking for write permission
file.canExecute() : checking for execute permission
file.length() : to get size of the file in bytes.
file.getParent : to get parent name
file.getName() : it return name of the file
file.renameTo(dest) : to rename a file
file.pathSeparator
file.pathSeparatorChar : OS file path separator.
file.lastModified() : last modified time.

Program for the all above methods.

```
package com.durga.mnrao.xyz;
import java.io.File;
import java.util.Date;
public class Test {
        public static void main(String[] args) {
                if( args.length != 1 ){
```

```
                System.out.println("Missing input file Path");
                System.exit(0);
        }
        File  filePath = new File( args[0] );
        if( ! filePath.exists() ){
                System.out.println(args[0]+" does not exist");
                System.exit(0);
        }
        if( filePath.isFile() ){
                System.out.println(args[0]+" is a file ");
        }
        if( filePath.isDirectory() ){
                System.out.println(args[0]+" is dir ");
        }
        if( filePath.canRead() ){
                System.out.println("it is readable file");
        }
        if( filePath.canWrite() ){
                System.out.println("it can be writable file");
        }
        if( filePath.canExecute() ){
                System.out.println("this file can be opened");
        }
        String path = filePath.getAbsolutePath();
        System.out.println("file location = "+path);
        String parent = filePath.getParent();
        System.out.println("parent = "+parent);
        String name = filePath.getName();
        System.out.println("file name = "+name);
        long size = filePath.length();
        System.out.println("file size = "+size+"  bytes");
        System.out.println("file size = "+(size/1024.0)+"  KB");
        System.out.println("file size = "+(size/(1024.0*1024.0))+"  MB");
        System.out.println("file size = "+(size/(1024.0*1024.0*1024))+"  GB");
        long time = filePath.lastModified();
        Date date = new Date(time);
        System.out.println("time of last modification =  "+date);
}       }
```

Pass file path as command line params and run the above program
runAs → RunConfiguration → argumets → "D:\MNRAO-Java-material\CoreJava\latest.pdf"
to create new file

```
File  filePath  = new File("D:\\NRIT_Java-material\\CoreJava\\emp.dat");
(or)
File  filePath  = new File(args[0]);
        if(filePath.createNewFile()){
                System.out.println("created");
        }
        else  {
                System.out.println("failed");
        }
```

To delete file:

```
File filePath = new File(args[0]);
        filePath.deleteOnExit();
```

FileInputStream and FileOutputStream (File Handling)

In Java, FileInputStream and FileOutputStream classes are used for file handling in java.
Java FileOutputStream class
Java FileOutputStream is an output stream for writing data to a file.
If you have to write primitive values then use FileOutputStream.Instead, for character-oriented data, prefer FileWriter.But you can write byte-oriented as well as character-oriented data.

Program to create file using commandline params

D:\test>java Create file1
It should take only one parameter.
It has to take data from keyboard
i/p:

at the end shoud be ctrl+z
step1:
D:\test>notepad Create.java

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
public class Create {
	public static void main(String[] args) throws IOException {
		if(args.length!=1){
		System.out.println("invalid syntax, usage: java Create <file_name>");
		System.exit(0);
		}
		File filePath = new File(args[0]);
		if(filePath.exists()){
		System.out.println(args[0]+" already exist, can not create");
		System.exit(0);
		}
		FileOutputStream  fos=null;
		try{
		 fos = new FileOutputStream(filePath);
		char ch = (char )System.in.read();
		while(ch!=(char)-1){
			fos.write(ch);
			ch = (char )System.in.read();
		}
		fos.flush();
		fos.close();
		System.out.println("Successfully created");
		}
		catch(Exception e){
		e.printStackTrace();
		}
		finally{
		if(fos!=null){
		 fos.close();
		}  } } }
```

D:\test> javac Create.java
D:\test> java Create file1
Hello this mnrao
Welcome to hyderbad
^z (ctrl + z)
Output:
File1 Successfullly created
Program to display file data :
D:\test> notepad Display.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class Display {
	public static void main(String[] args) throws IOException {
		if(args.length!=1){
		System.out.println("invalid syntax, usage: java Display <file_name>");
			System.exit(0);
		}
		File inputFile = new File(args[0]);
		if( !inputFile.exists() ){
			System.out.println(args[0]+" file not found ");
			System.exit(0);
		}
		if( ! inputFile.isFile() ){
			System.out.println(args[0]+"not a file ");
			System.exit(0);
		}
		FileInputStream  fis = null;
		try {
```

```
                        fis = new FileInputStream(inputFile);
                        char ch = (char)fis.read();
                        while(ch!=(char)-1){
                                System.out.print(ch);
                                ch = (char)fis.read();
                        }
                        fis.close();
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
                finally{
                        if(fis!=null){
                                fis.close();
                        } } } }
$javac Display.java
$java Display  file1
```

Copying files.

```
D:\test>java  MyCopy  file1  file2
D:\test>notepad  MyCopy.java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class MyCopy {
        public static void main(String[] args) throws IOException {
                if( args.length != 2 ){
                System.out.println("Invalid Syntax, Usage : java Mycopy <src_file> <dest_file>");
                        System.exit(0);
                }
                File srcFilePath = new File(args[0]);
                File destFilePath = new File(args[1]);
                if(! srcFilePath.exists()){
                        System.out.println(args[0]+" source does not exist, can not copy");
                        System.exit(0);
                }
                if( ! srcFilePath.isFile()){
                        System.out.println(args[0]+" exist but not a file");
                        System.exit(0);
                }
                if( destFilePath.exists() ){
System.out.println(args[1]+" destination already exist can not copy ");
        System.exit(0);
                }
                FileInputStream fis = null;
                FileOutputStream fos = null;
                try{
                        fis = new FileInputStream(srcFilePath);
                        fos = new FileOutputStream(destFilePath);
                        char ch  =(char) fis.read();
                        while( ch!=(char)-1 )  {
                                fos.write(ch);
                                ch  =(char) fis.read();
                        }
                        fis.close();
                        fos.close();
                        System.out.println("Copied ");
                }
                catch(Exception e){
                        e.printStackTrace();
                }
                finally {
                        if(fis!=null){
                                fis.close();
                        }
```

```
                    if(fos!=null){
                            fos.close();
                    } } } }
```

```
$ javac MyCopy.java
$ java MyCopy file1 file2
```

Reading and writing lines:

DataInputStream provides readLine() method to read a line

```
FileInputStream  fis= null;
DataInputStream dis = null;
fis= new FileInputStream("employee.dat");
dis  = new DataInputStream(fis);
String line = dis.readLine();  --> returns null when reach to EOF.
```

Program to read emplyeee records line by line

"employee.dat" **file contains following data.**

```
1001:ajay:manager:account:45000:male:38
1002:aiswrya:clerk:account:25000:female:30
1003:varun:manager:sales:50000:male:35
1004:amit:manager:account:47000:male:40
1005:kareena:executive:sales:15000:female:24
1006:deepak:clerk:sales:23000:male:30
1007:sunil:accountant:sales:13000:male:29
1008:satvik:director:purchase:80000:male:45
```

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class ReadLineByLine {
        public static void main(String[] args) throws IOException {
                if(args.length!=1){
                        System.out.println("Input file missing");
                        System.exit(0);
                }
                File filePath = new File(args[0]);
                if( ! filePath.exists() ){
                        System.out.println(args[0]+" not found ");
                        System.exit(0);
                }
                FileInputStream fis = null;
                DataInputStream dis = null;
                try  {
                fis = new FileInputStream(filePath);
                        dis = new DataInputStream(fis);
                String line = dis.readLine();
        while( line!=null ){
                                System.out.println(line);
                                line = dis.readLine();
                        }
                dis.close();
                        fis.close();
                }
                catch(Exception e){
                        e.printStackTrace();
                }
                finally {
                        if(dis!=null)
                        {
                                dis.close();
                        }
                        if(fis!=null)
                        {
                                fis.close();
                        } } } }
```

RunAs RunConfiguration Java Application (double click) ReadLineByLine arguments "C:\project\mnrao\employee.dat" run

program to convert above format into csv file (employee.dat to employee.csv)

Note : employee.csv should also be generated at same location of "employee.dat"

"employee.dat" **file contains following data.**

1001:ajay:manager:account:45000:male:38
1002:aiswrya:clerk:account:25000:female:30
1003:varun:manager:sales:50000:male:35
1004:amit:manager:account:47000:male:40
1005:kareena:executive:sales:15000:female:24
1006:deepak:clerk:sales:23000:male:30
1007:sunil:accountant:sales:13000:male:29
1008:satvik:director:purchase:80000:male:45

program:

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class GenerateCSVFile {
	public static void main(String[] args) throws IOException {
		if(args.length!=2){
		System.out.println("Missing input files, submit source and destination files ");
			System.exit(0);
		}
		File srcFilePath = new File(args[0]);
		File destFilePath = new File(args[1]);
		if(!srcFilePath.exists()){
			System.out.println(args[0]+" not found ");
			System.exit(0);
		}
		FileInputStream fis = null;
		DataInputStream dis = null;
		FileOutputStream fos = null;
		DataOutputStream dos = null;
		try{
			fis = new FileInputStream(srcFilePath);
			dis = new DataInputStream(fis);
			fos = new FileOutputStream(destFilePath);
			dos = new DataOutputStream(fos);
			String line = dis.readLine();
			while( line!=null ){

				String result = line.replaceAll(":", ",");
				dos.writeBytes(result+"\n");
				line = dis.readLine();
			}
			dis.close();
			fis.close();
			dos.close();
			fos.close();
			System.out.println("Successfully data converted ");
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if(dis!=null){
				dis.close();
			}
			if(fis!=null){
				fis.close();
			}
			if(dos!=null){
				dos.close();
			}
```

```
                    if(fos!=null){
                        fos.close();
                    } } } }
```

RunAs → RunConfiguration → Java Application (double click) → GenerateCSVFile → arguments → "C:\project\mnrao\employee.dat" "C:\project\mnrao\employee.csv" → run

o/p file employee.csv as follows:

```
1001,ajay,manager,account,45000,male,38
1002,aiswrya,clerk,account,25000,female,30
1003,varun,manager,sales,50000,male,35
1004,amit,manager,account,47000,male,40
1005,kareena,executive,sales,15000,female,24
1006,deepak,clerk,sales,23000,male,30
1007,sunil,accountant,sales,13000,male,29
1008,satvik,director,purchase,80000,male,45
```

Program for selected fields for the below data :

```
1001,ajay,manager,account,45000,male,38
1002,aiswrya,clerk,account,25000,female,30
1003,varun,manager,sales,50000,male,35
1004,amit,manager,account,47000,male,40
1005,kareena,executive,sales,15000,female,24
1006,deepak,clerk,sales,23000,male,30
1007,sunil,accountant,sales,13000,male,29
1008,satvik,director,purchase,80000,male,45
```

```
package com.durga.mnrao.files;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class EmpDataSelectedFilelds {
    public static void main(String[] args) throws IOException {
        if( args.length!=1 ){
            System.out.println("Missing input file ");
            System.exit(0);
        }
        File filePath = new File(args[0]);
        if(!filePath.exists()){
            System.out.println(args[0]+" not found");
            System.exit(0);
        }
        FileInputStream fis = null;
        DataInputStream dis = null;
        try{
            fis = new FileInputStream(filePath);
            dis = new DataInputStream(fis);
            String record =  dis.readLine();
            while( record != null && ! record.isEmpty() ){
                String[] fileds = record.split(",");
                System.out.println(fileds[0]+"\t"+fileds[1]+"\t"+fileds[3]+"\t"+fileds[5]);
    record =  dis.readLine();
            }
            dis.close();
            fis.close();
        }
        catch(Exception e){
            e.printStackTrace();
        }
        finally{
            if(dis!=null){
                dis.close();
            }
            if(fis!=null){
                fis.close();
            } } } }
```

Based conditions:

For the below data :

1001,ajay,manager,account,45000,male,38
1002,aiswrya,clerk,account,25000,female,30
1003,varun,manager,sales,50000,male,35
1004,amit,manager,account,47000,male,40
1005,kareena,executive,sales,15000,female,24
1006,deepak,clerk,sales,23000,male,30
1007,sunil,accountant,sales,13000,male,29
1008,satvik,director,purchase,80000,male,45

Only Managers :

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class EmpDataManagers {
	public static void main(String[] args) throws IOException {
		if( args.length!=1 ){
			System.out.println("Missing input file ");
			System.exit(0);
		}
		File filePath = new File(args[0]);
		if(!filePath.exists()){
			System.out.println(args[0]+" not found");
			System.exit(0);
		}
		FileInputStream fis = null;
		DataInputStream dis = null;
		try{
			fis = new FileInputStream(filePath);
			dis = new DataInputStream(fis);
			String record =  dis.readLine();
			while( record != null && ! record.isEmpty())  {
				String[] fields = record.split(",");
				if(fields[2].equals("manager")){
		System.out.println(record);
				}
				record =  dis.readLine();
			}
			dis.close();
			fis.close();
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if(dis!=null){
				dis.close();
			}
			if(fis!=null){
				fis.close();
			} } } }
```

Only Male Managers:

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class EmpDataMaleManagers {
	public static void main(String[] args) throws IOException {
		if( args.length!=1 ){
			System.out.println("Missing input file ");
			System.exit(0);
		}
		File filePath = new File(args[0]);
		if(!filePath.exists()){
			System.out.println(args[0]+" not found");
			System.exit(0);
```

```
		}
		FileInputStream fis = null;
		DataInputStream dis = null;
		try{
			fis = new FileInputStream(filePath);
			dis = new DataInputStream(fis);
			String record =  dis.readLine();
			while( record != null && ! record.isEmpty()){
				String[] fields = record.split(",");
				if(fields[2].equals("manager") && fields[5].equals("male")){
					System.out.println(record);
				}
				record =  dis.readLine();
			}
			dis.close();
			fis.close();
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if(dis!=null){
				dis.close();
			}
			if(fis!=null){
				fis.close();
			}  } } }
```

Only sales dept, Male Managers

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class EmpDataMaleManagersDept10 {
	public static void main(String[] args) throws IOException {
		if( args.length!=1 ){
			System.out.println("Missing input file ");
			System.exit(0);
		}
		File filePath = new File(args[0]);
		if(!filePath.exists()){
			System.out.println(args[0]+" not found");
			System.exit(0);
		}
	FileInputStream fis = null;
		DataInputStream dis = null;
		try{
			fis = new FileInputStream(filePath);
			dis = new DataInputStream(fis);
			String record =  dis.readLine();
			while( record != null && ! record.isEmpty()){
String[] fields = record.split(",");
	if(fields[2].equals("manager") && fields[3].equals("sales")  && fields[5].equals("male") ){
					System.out.println(record);
				}
				record =  dis.readLine();
			}
			dis.close();
			fis.close();
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally {
			if(dis!=null){
				dis.close();
```

```
                }
                if(fis!=null){
                        fis.close();
                } } } }
```

Only sales dept , Male Managers with Salary Condition :

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class EmpDataMaleManagersDeptSalary {
        public static void main(String[] args) throws IOException {
                if( args.length!=1 ){
                        System.out.println("Missing input file ");
                        System.exit(0);
                }
                File filePath = new File(args[0]);
                if(!filePath.exists()){
                        System.out.println(args[0]+" not found");
                        System.exit(0);
                }
                FileInputStream fis = null;
                DataInputStream dis = null;
                try{
                        fis = new FileInputStream(filePath);
                        dis = new DataInputStream(fis);
                        String record =  dis.readLine();
                        while( record != null && ! record.isEmpty()){
                        String[] fields = record.split(",");
if(fields[2].equals("manager") && fields[3].equals("sales")  && fields[5].equals("male")
&&Double.parseDouble(fields[4])>=45000){
                                        System.out.println(record);
                                }
                                record =  dis.readLine();
                        }
                        dis.close();
                        fis.close();
                        }
                catch(Exception e){
                        e.printStackTrace();
                }
                finally{
                        if(dis!=null){
                                dis.close();
                        }
                        if(fis!=null){
                                fis.close();
                        }  } } }
```

Program to read records from text file, convert into Employee object and store into ArrayList and display from ArrayList

Data :

```
1001,ajay,manager,account,45000,male,38
1002,aiswrya,clerk,account,25000,female,30
1003,varun,manager,sales,50000,male,35
1004,amit,manager,account,47000,male,40
1005,kareena,executive,sales,15000,female,24
1006,deepak,clerk,sales,23000,male,30
1007,sunil,accountant,sales,13000,male,29
1008,satvik,director,purchase,80000,male,45
```

POJO class for Employye record

```
package com.durga.mnrao.list;
public class Employee {
        private int empNum;
        private String empName;
        private String empJob;
        private String empDeptName;
```

```
	private double empSalary;
	private String empGender;
	private int empAge;
	public int getEmpNum() {
		return empNum;
	}
	public void setEmpNum(int empNum) {
		this.empNum = empNum;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public String getEmpJob() {
		return empJob;
	}
	public void setEmpJob(String empJob) {
		this.empJob = empJob;
	}
	public String getEmpDeptName() {
		return empDeptName;
	}
	public void setEmpDeptName(String empDeptName) {
		this.empDeptName = empDeptName;
	}
	public double getEmpSalary() {
		return empSalary;
	}
	public void setEmpSalary(double empSalary) {
		this.empSalary = empSalary;
	}
	public String getEmpGender() {
		return empGender;
	}
	public void setEmpGender(String empGender) {
		this.empGender = empGender;
	}
	public int getEmpAge() {
		return empAge;
	}
	public void setEmpAge(int empAge) {
		this.empAge = empAge;
	}  }
package com.durga.mnrao.list;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
public class EmpDataLoadArrayList {
	public static void main(String[] args) throws IOException {
		if( args.length != 1 ){
			System.out.println("Missing input data file ");
			System.exit(0);
		}
		File filePath = new File(args[0]);
		if( ! filePath.exists() ){
			System.out.println(args[0]+" does not exist ");
			System.exit(0);
		}
		FileInputStream fis = null;
		DataInputStream dis = null ;
```

```
		try{
			ArrayList<Employee> empDataList  = new ArrayList<Employee>();
			fis = new FileInputStream(filePath);
			dis = new DataInputStream( fis );
			String record = dis.readLine();
			while( record != null    &&    ! record.isEmpty()){
				String[] fields = record.split(",");
				Employee employee = new Employee();
				employee.setEmpNum( Integer.parseInt(fields[0]) );
				employee.setEmpName(fields[1]);
				employee.setEmpJob(fields[2]);
				employee.setEmpDeptName(fields[3]);
				employee.setEmpSalary(Double.parseDouble(fields[4]));
				employee.setEmpGender(fields[5]);
				employee.setEmpAge(Integer.parseInt(fields[6]));
				empDataList.add(employee);
				record = dis.readLine();
			}
			dis.close();
			fis.close();
			System.out.println("Data loaded into array list");
			Iterator<Employee> iterator = empDataList.iterator();
			while( iterator.hasNext() ){
				Employee emp = iterator.next();
				int empNum = emp.getEmpNum();
				String empName = emp.getEmpName();
				String empJob = emp.getEmpJob();
				String empDeptName = emp.getEmpDeptName();
				double empSalary = emp.getEmpSalary();
				String empGender = emp.getEmpGender();
				int empAge = emp.getEmpAge();
	System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empDeptName+"\t"+empSalary+"\t"+empGender+
"\t"+empAge);
			}
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if( dis != null ) {
				dis.close();
			}
			if( fis != null){
				fis.close();
		} } } }
```

Program to find dept numbers of Employees data.

Data as follows "employee.csv"

```
1001,ajay,manager,10,45000,male,38
1002,aiswrya,clerk,20,25000,female,30
1003,varun,manager,30,50000,male,35
1004,amit,manager,10,47000,male,40
1005,kareena,executive,30,15000,female,24
1006,deepak,clerk,20,23000,male,30
1007,sunil,accountant,40,13000,male,29
1008,satvik,director,20,80000,male,45
1009,vijay,manager,30,40000,male,35
1010,sandeep,executive,10,50000,male,31
```

```
package com.durga.mnrao.list;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.TreeSet;
public class EmpDataDeptList {
```

```
	public static void main(String[] args) throws IOException {
		if(args.length!=1){
			System.out.println("Missing input files");
			System.exit(0);
		}
		File srcFilePath = new File(args[0]);
		if(!srcFilePath.exists()){
			System.out.println(args[0]+" not found ");
			System.exit(0);
		}
		FileInputStream fis = null;
		DataInputStream dis = null;
		try{
			fis = new FileInputStream(srcFilePath);
			dis = new DataInputStream(fis);
			TreeSet<Integer>  empDeptList = new TreeSet<Integer>();
			String record = dis.readLine();
			while( record != null && ! record.isEmpty() ){
				String [] fileds = record.split(",");
				int deptNum = Integer.parseInt(fileds[3]);
				empDeptList.add(deptNum);
				record = dis.readLine();
			}
			dis.close();
			fis.close();
			System.out.println("Data loaded into list");
			Iterator<Integer> iterator = empDeptList.iterator();
			while(iterator.hasNext()){
				Integer num = iterator.next();
				System.out.println(num);
			}
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if(dis!=null){
				dis.close();
			}
			if(fis!=null){
				fis.close();
			} } } }
```

To display records order by deptnum ;

```
package com.durga.mnrao.list;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;
public class EmpDataOrderByDeptNum {
	public static void main(String[] args) throws IOException {
		if(args.length!=1){
			System.out.println("Missing input files");
			System.exit(0);
		}
		File srcFilePath = new File(args[0]);
		if(!srcFilePath.exists()){
			System.out.println(args[0]+" not found ");
			System.exit(0);
		}
		FileInputStream fis = null;
		DataInputStream dis = null;
		try{
```

```
                fis = new FileInputStream(srcFilePath);
                dis = new DataInputStream(fis);
                TreeSet<Integer>  empDeptList = new TreeSet<Integer>();
                ArrayList<String> empRecordsList = new ArrayList<String>();
                String record = dis.readLine();
                while( record != null && ! record.isEmpty() ){
                        String [] fileds = record.split(",");
                        int deptNum = Integer.parseInt(fileds[3]);
                        empDeptList.add(deptNum);
                        empRecordsList.add(record);
                        record = dis.readLine();
                }
                dis.close();
                fis.close();
                System.out.println("Data loaded into list");
                Iterator<Integer> deptIterator = empDeptList.iterator();
                while(deptIterator.hasNext()) {
                        Integer deptNum = deptIterator.next();
        Iterator<String> empRecordsIterator = empRecordsList.iterator();
    while(empRecordsIterator.hasNext()) {
                String empRecord = empRecordsIterator.next();
                        String[] fileds = empRecord.split(",");
    if( deptNum == Integer.parseInt(fileds[3]) ){
                                        System.out.println(empRecord);
                                } } } }

        catch(Exception e){
                e.printStackTrace();
        }
        finally{
                if(dis!=null){
                        dis.close();
                }
                if(fis!=null){
                        fis.close();
                } } } }
EmpDataOrderByGender;
package com.durga.mnrao.list;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;
public class EmpDataOrderByGender {
    public static void main(String[] args) throws IOException {
        if(args.length!=1){
                System.out.println("Missing input files");
                System.exit(0);
        }
        File srcFilePath = new File(args[0]);
        if(!srcFilePath.exists()){
                System.out.println(args[0]+" not found ");
                System.exit(0);
        }
        FileInputStream fis = null;
        DataInputStream dis = null;
        try{
                fis = new FileInputStream(srcFilePath);
                dis = new DataInputStream(fis);
                TreeSet<String>  empGenderList = new TreeSet<String>();
                ArrayList<String> empRecordsList = new ArrayList<String>();
                String record = dis.readLine();
                while( record != null && ! record.isEmpty() ){
```

```
                        String [] fileds = record.split(",");
                        empGenderList.add(fileds[5]);
        empRecordsList.add(record);
                        record = dis.readLine();
                        }
                        dis.close();
                        fis.close();
                        System.out.println("Data loaded into list");
                        Iterator<String> GenderIterator = empGenderList.iterator();
                        while(GenderIterator.hasNext()){
                                String gender = GenderIterator.next();
Iterator<String> empRecordsIterator = empRecordsList.iterator();
                        while(empRecordsIterator.hasNext()) {
                                String empRecord = empRecordsIterator.next();
                                String[] fileds = empRecord.split(",");
                                if( gender.equals(fileds[5])){
                                        System.out.println(empRecord);
                                } } } }
                catch(Exception e){
                        e.printStackTrace();
                }
                finally{
                        if(dis!=null){
                                dis.close();
                        }
                        if(fis!=null){
                                fis.close();
                        } } } }
```

EmpDataOrderByJob:

```
package com.durga.mnrao.list;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;
public class EmpDataOrderByJob {
        public static void main(String[] args) throws IOException {
                        if(args.length!=1){
                        System.out.println("Missing input files");
                        System.exit(0);
                }
                File srcFilePath = new File(args[0]);
                if(!srcFilePath.exists()){
                        System.out.println(args[0]+" not found ");
                        System.exit(0);
                }
                FileInputStream fis = null;
                DataInputStream dis = null;
                try{
                        fis = new FileInputStream(srcFilePath);
                        dis = new DataInputStream(fis);
                        TreeSet<String>  empJobList = new TreeSet<String>();
                        ArrayList<String> empRecordsList = new ArrayList<String>();
                        String record = dis.readLine();
                         while( record != null && ! record.isEmpty() )
                                String [] fileds = record.split(",");
                                empJobList.add(fileds[2]);
        empRecordsList.add(record);
                                record = dis.readLine();
                        }
                        dis.close();
                        fis.close();
                                System.out.println("Data loaded into list");
```

```
                    Iterator<String> JobIterator = empJobList.iterator();
                        while(JobIterator.hasNext()){
                        String job = JobIterator.next();
                    Iterator<String> empRecordsIterator = empRecordsList.iterator();
                    while(empRecordsIterator.hasNext()){
                        String empRecord = empRecordsIterator.next();
                            String[] fileds = empRecord.split(",");
                            if( job.equals(fileds[2])){
                                System.out.println(empRecord);
                             } } } }
        catch(Exception e){
                    e.printStackTrace();
            }
            finally{
                    if(dis!=null){
                        dis.close();
                    }
                    if(fis!=null){
                        fis.close();
                    } } } }
```

Java SequenceInputStream Example

Example that reads the data from two files and writes screen

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
public class Test{
        public static void main(String[] args) throws IOException{
                File file1 = new File("D:\\testin1.txt");
                File file2 = new File("D:\\testin2.txt");
                 FileInputStream fis1= new FileInputStream(file1);
                 FileInputStream fis2=new FileInputStream(file2);
                  SequenceInputStream inst = new SequenceInputStream(fis1, fis2);
                  char ch=(char) inst.read();
                  while(ch!=(char)-1) {
                    System.out.print(ch);
                            ch=(char)inst.read();
                 }
                 inst.close();
                 fis1.close();
                 fis2.close();
        }  }
```

Example that reads the data from two files and writes into another file

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
public class Test{
        public static void main(String[] args) throws IOException
                File file1 = new File("D:\\testin1.txt");
                File file2 = new File("D:\\testin2.txt");
                File file3 = new File("D:\\testout.txt");
                 FileInputStream input1= new FileInputStream(file1);
                 FileInputStream input2=new FileInputStream(file2);
                 FileOutputStream output=new FileOutputStream(file3);
                SequenceInputStream inst=new SequenceInputStream(input1, input2);
            char ch=(char)inst.read();
                  while(ch!=(char)-1) {
                        output.write(ch);
                        ch=(char)inst.read();
                 }
                 inst.close();
                 input1.close();
                 input2.close();
```

output.close(); } }

Java Console Class:

The Java Console class is be used to get input from console. It provides methods to read texts and passwords. If you read password using Console class, it will not be displayed to the user.

The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

Let's see a simple example to read text from console.

public final class Console extends Object implements Flushable

Method	Description
Reader reader()	It is used to retrieve the reader object associated with the console
String readLine()	It is used to read a single line of text from the console.
String readLine(String fmt, Object... args)	It provides a formatted prompt then reads the single line of text from the console.
char[] readPassword()	It is used to read password that is not being displayed on the console.
char[] readPassword(String fmt, Object... args)	It provides a formatted prompt then reads the password that is not being displayed on the console.
Console format(String fmt, Object... args)	It is used to write a formatted string to the console output stream.
Console printf(String format, Object... args)	It is used to write a string to the console output stream.
PrintWriter writer()	It is used to retrieve the PrintWriter object associated with the console.
void flush()	It is used to flushes the console.

To read data from console:

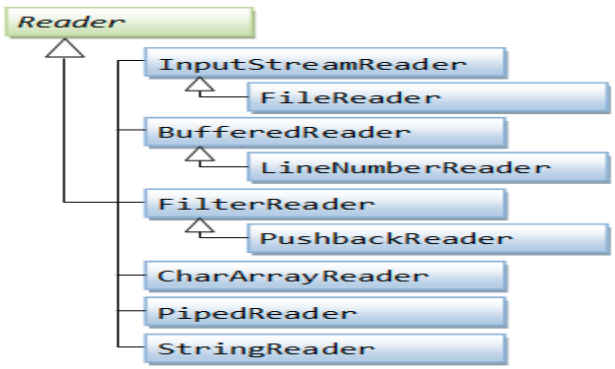
```
import java.io.Console;
public class ConsoleTest{
        public static void main(String[] args) {
                Console c = System.console();
                System.out.println ("Enter your name: ");
                String  name= c.readLine();
                System.out.println ("Welcome "+name);
        }
}
```

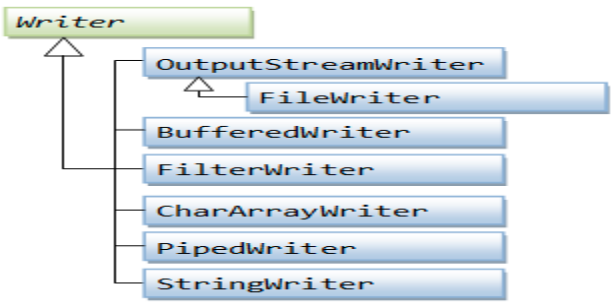
Password example:

```
import java.io.Console;
public class ConsoleTest{
        public static void main(String[] args)  {
                Console c = System.console();
                System.out.println ("Enter your name: ");
                String name = c.readLine();
                System.out.println ("Enter your passwd: ");
                char [] ch =  c.readPassword();
                String  pass = String.valueOf(ch);
                if(name.equals("mnrao") && pass.equals("java")){
                        System.out.println ("Valid User");
                }
                else{
                        System.out.println ("Invalid user");
                }   } }
```

Character Streams

FileWriter class to create a file





==========

```
import java.io.File;
import java.io.FileWriter;
```

```
import java.io.IOException;
public class Create {
	public static void main(String[] args) throws IOException {
			if( args.length!=1){
			System.out.println("Invalid syntax, Usage : java Create <file_name> ");
				System.exit(0);
			}
			File filePath = new File(args[0]);
			if( filePath.exists() ){
				System.out.println(args[0]+" already exist ");

				System.exit(0);
			}
			FileWriter fw = null;
			try{
				fw = new FileWriter(filePath);
				char ch = (char) System.in.read();
				while( ch != (char) -1 ){
					fw.write( ch );
					ch = (char) System.in.read();
				}
				fw.close();
				System.out.println("successfully created ");
			}
			catch(Exception e){
				e.printStackTrace();
			}
			finally {
				if( fw != null ){
					fw.close();
				} } } }
```

FileReader Example:

```
package com.durga.mnrao.cstream;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
public class Display {
	public static void main(String[] args) throws IOException {
		if( args.length!=1){
			System.out.println("Invalid syntax, Usage : java Display <file_name> ");
			System.exit(0);
		}
		File filePath = new File(args[0]);
		if( ! filePath.exists() ){
			System.out.println(args[0]+" not found can not display ");
			System.exit(0);
		}
		FileReader fr =null;
		try{
			fr = new FileReader(filePath);
			char ch = (char) fr.read();
			while( ch!=(char)-1 ){
				System.out.print(ch);
				ch = (char) fr.read();
			}
			fr.close();
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if(fr!=null){
				fr.close();
			}  } } }
```

Java BufferedReader class:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
public class ReadLineByLine {
	public static void main(String[] args) throws IOException {
		if( args.length!=1){
			System.out.println("Invalid syntax, Usage : java Display <file_name> ");
			System.exit(0);
		}
		File filePath = new File(args[0]);
		if( ! filePath.exists() ){
			System.out.println(args[0]+" not found can not display ");
			System.exit(0);
		}
		FileReader fr =null;
		BufferedReader br = null;
		try{
			fr = new FileReader(filePath);
			br = new BufferedReader(fr);
			String line = br.readLine();
			while( line !=null  ){
				System.out.println(line);
				line = br.readLine();
			}
			br.close();
			fr.close();
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if(br!=null){
				br.close();
			}
			if(fr!=null){
				fr.close();
			} } } }
```

BufferedWriter Example:
program to convert belowe format into csv file (employee.dat to employee.csv)
Note : employee.csv should also be generated at same location of "employee.dat"

"employee.dat" **file contains following data.**

```
1001:ajay:manager:account:45000:male:38
1002:aiswrya:clerk:account:25000:female:30
1003:varun:manager:sales:50000:male:35
1004:amit:manager:account:47000:male:40
1005:kareena:executive:sales:15000:female:24
1006:deepak:clerk:sales:23000:male:30
1007:sunil:accountant:sales:13000:male:29
1008:satvik:director:purchase:80000:male:45
```

Convert into employee.csv Format as below

```
1001,ajay,manager,account,45000,male,38
1002,aiswrya,clerk,account,25000,female,30
1003,varun,manager,sales,50000,male,35
1004,amit,manager,account,47000,male,40
1005,kareena,executive,sales,15000,female,24
1006,deepak,clerk,sales,23000,male,30
1007,sunil,accountant,sales,13000,male,29
1008,satvik,director,purchase,80000,male,45
```

```
package com.durga.mnrao.cstream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
```

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class GenerateCSVFile {
	public static void main(String[] args) throws IOException {
		if(args.length!=2){
		System.out.println("Missing input files, submit source and destination files ");
			System.exit(0);
		}
		File srcFilePath = new File(args[0]);
		File destFilePath = new File(args[1]);
		if(!srcFilePath.exists()){
			System.out.println(args[0]+" not found ");
			System.exit(0);
		}
		FileReader fr = null;
		BufferedReader br = null;
		FileWriter fw = null;
		BufferedWriter bw = null;
		try{
			fr = new FileReader(srcFilePath);
			br  =  new BufferedReader(fr);
			fw = new FileWriter(destFilePath);
			bw = new BufferedWriter(fw);
			String line = br.readLine();
			while( line!=null ){
				String result = line.replaceAll(":", ",");
				bw.write(result+"\n");
				line = br.readLine();
			}
			br.close();
			fr.close();
			bw.close();
			fw.close();
			System.out.println("Successfully data converted ");
		}
		catch(Exception e){
			e.printStackTrace();
		}
		finally{
			if(br!=null){
				br.close();
			}
			if(fr!=null){
				fr.close();
			}
			if(bw!=null){
				bw.close();
			}
			if(fw!=null){
				fw.close();
			} } } }
```

Above program submit as below

Runas Runconfiguration arguments

C:\project\mnrao\employee.dat C:\project\mnrao\employee.csv

Reading Line from console by InputStreamReader and BufferedReader

```
import java.io.*;
public class BufferedReaderExample{
	public static void main(String []  args) throws Exception{
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader  br =  new BufferedReader( isr );
		System.out.println ("Enter your name");
		String name = br.readLine();
		System.out.println ("Welcome " + name);
	}
```

}

PrintStream class Example:

it provides print() and println () over loaded methods for different data types.

```
import java.io.FileOutputStream;
import java.io.PrintStream;
public class Test{
        public static void main(String [] args)throws Exception {
         FileOutputStream fout=new FileOutputStream("D:\\testout.txt ");
                PrintStream pout = new PrintStream(fout);
                pout.println (2016);
                pout.println (2000.50);
                pout.println ("Welcome to NR IT Solutions");
                pout.close();
                fout.close();
               System.out.println ("Success?");
        }
}
```

PrintWriter class Example:

Example of writing the data on a console and in a text file testout.txt using Java PrintWriter class.

```
import java.io.File;
import java.io.PrintWriter;
public class Test{
        public static void main(String[] args) throws Exception{
                // Data to write on Console using PrintWriter
                PrintWriter writer = new PrintWriter(System.out);
                writer.write("Welcome to NR IT Solutions");
                writer.flush();
                writer.close();
                // Data to write in File using PrintWriter
                PrintWriter writer1 = null;
                writer1 = new PrintWriter(new File("D:\\testout.txt"));
                writer1.write("Training on Java, Spring, Hibernate");
                writer1.flush();
                writer1.close();
        }
}
```

System.out.printf():

==========

output formatting:

```
public static void main(String [] args) throws Exception {
                System.out.printf("%5d  %10s",10, "hello");
        }
```

Loading Properties file data

Properties class provide following methods

1. Object setProperty(String key, String value) ⍰ it takes key and value pair
2. String getProperty(String key)

Properties file data as below:

```
Path = C:\project\DatabaseProperties.prop
dbDriver=com.mysql.cj.jdbc.Driver
dbHost=localhost
dbPort=3305
dbUid=root
dbPasswd=root
dbName=mnrao
```

```
package com.durga.mnrao.prop;
import java.io.File;
import java.io.FileReader;
import java.util.Properties;
public class LoadDatabaseProperties {
        public static void main(String[] args) {
                if(args.length !=1 ){
                        System.out.println("Missing properties file");
                        System.exit(0);
                }
```

```
		File filePath = new  File(args[0]);
		if( ! filePath.exists() ){
			System.out.println(args[0]+" does not exist ");
			System.exit(0);
		}
		FileReader freader = null;
		try{
			freader = new FileReader(filePath);
			Properties dbProp = new Properties();
			dbProp.load(freader);
			freader.close();
			System.out.println("Properties loaded ");
			String driver = dbProp.getProperty("dbDriver");
			String host = dbProp.getProperty("dbHost");
			String port = dbProp.getProperty("dbPort");
			String uid = dbProp.getProperty("dbUid");
			String pwd = dbProp.getProperty("dbPasswd");
			String name = dbProp.getProperty("dbName");
			System.out.println("db driver = "+driver);
			System.out.println("db host = "+host);
			System.out.println("db prot num = "+port);
			System.out.println("db uid = "+uid);
			System.out.println("db password = "+pwd);
			System.out.println("db name = "+name);
		}
		catch(Exception e){
			e.printStackTrace();
		} } }
```

runAs → **RunConfiguartion** → **JavaApplication** → LoadDatabaseProperties → arguments
C:\project\DatabaseProperties.prop
Apply → run

Excel Data Parsing

1001	scott	50000.5	admin	male	28
1002	blake	30000.2	sales	male	30
1003	dona	60000.2	accounts	female	29
1004	henry	45000	marketing	female	31
1005	martin	35000	finance	female	31
1006	david	75000	it	male	40
1007	rathod	25000	accounts	female	22
1008	jones	55000	it	male	30
1009	sandeep	45000	admin	male	28
1010	vijay	35000	sales	male	31
1011	Dona	25000	accounts	female	24
1012	Gourav	35000	it	male	23
1013	Keerthana	23000	sales	female	25

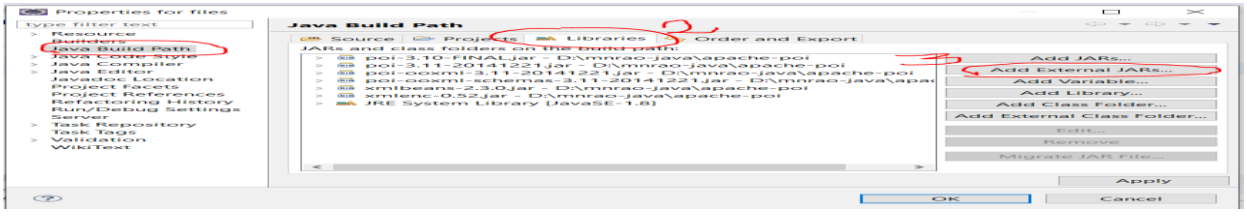
Pojo class

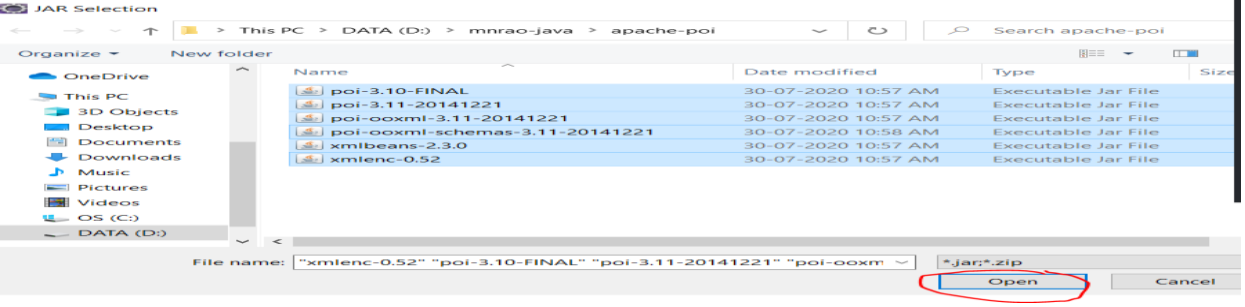
```
package com.durga.mnrao.exel;
public class Employee {
	private int empNum;
	private String empName;
	private double empSalary;
	private String empDeptName;
	private String empGender;
	private int empAge;
	public int getEmpNum() {
		return empNum;
	}
	public void setEmpNum(int empNum) {
```

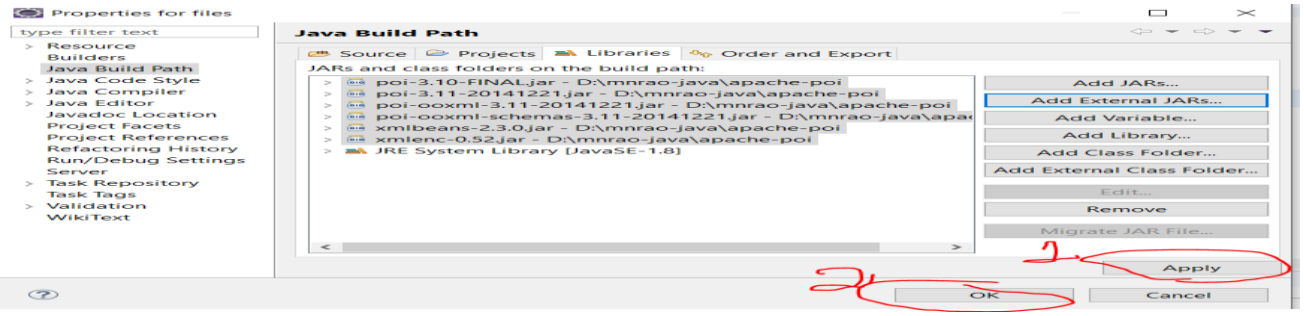
```
		this.empNum = empNum;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSalary() {
		return empSalary;
	}
	public void setEmpSalary(double empSalary) {
		this.empSalary = empSalary;
	}
	public String getEmpDeptName() {
		return empDeptName;
	}
	public void setEmpDeptName(String empDeptName) {
		this.empDeptName = empDeptName;
	}
	public String getEmpGender() {
		return empGender;
	}
	public void setEmpGender(String empGender) {
		this.empGender = empGender;
	}
	public int getEmpAge() {
		return empAge;
	}
	public void setEmpAge(int empAge) {
		this.empAge = empAge;
	}  }
package com.durga.mnrao.exel;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class ExcelParser {
	public List<Employee> parseExceldata(String filePath){
		File file = new File(filePath);
		FileInputStream fis = null;
		DataInputStream dis = null;
		List<Employee> list = null;
		try{
			fis = new FileInputStream(file);
			dis = new DataInputStream(fis);
			list  = new ArrayList<Employee>();
			XSSFWorkbook workBook = new XSSFWorkbook(dis);
			XSSFSheet sheet = workBook.getSheetAt(0);
			Iterator<Row> rowIterator = sheet.rowIterator();
			while( rowIterator.hasNext() ){
				Row row = rowIterator.next();
				Cell cell = row.getCell(0);
				int empNum = (int) cell.getNumericCellValue();
				cell = row.getCell(1);
				String empName = cell.getStringCellValue();
				cell = row.getCell(2);
				double  empSalary =  cell.getNumericCellValue();
				cell = row.getCell(3);
```

```
                        String empDeptName = cell.getStringCellValue();
                    cell = row.getCell(4);
                        String empGender = cell.getStringCellValue();
                        cell = row.getCell(5);
                        int empAge = (int) cell.getNumericCellValue();
                        Employee employee = new Employee();
                        employee.setEmpNum(empNum);
                        employee.setEmpName(empName);
                        employee.setEmpSalary(empSalary);
                        employee.setEmpDeptName(empDeptName);
                        employee.setEmpGender(empGender);
                        employee.setEmpAge(empAge);
                        list.add(employee);
                }
                        workBook.close();
                        dis.close();
                        fis.close();
                }
                catch(Exception e ){
                        e.printStackTrace();
                }
                return list;
        } }
package com.durga.mnrao.exel;
import java.util.Iterator;
import java.util.List;
public class ExcelFileReader {
        public static void main(String[] args) {
                if(args.length!=1){
                        System.out.println(args[0]+" missing excel file");
                        System.exit(0);
                }
                String filePath = args[0];
                ExcelParser parser = new ExcelParser();
                List<Employee> empRecordsList = parser.parseExceldata(filePath);
                Iterator<Employee> iterator = empRecordsList.iterator();
                while(iterator.hasNext()){
                        Employee emp = iterator.next();
                        int empNum = emp.getEmpNum();
                        String empName = emp.getEmpName();
                        double empSalary = emp.getEmpSalary();
                        String empDeptName = emp.getEmpDeptName();
                        String empGender = emp.getEmpGender();
                        int empAge = emp.getEmpAge();
        System.out.println(empNum+"\t"+empName+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender);
                } } }
```

In eclipse :
Right click on main () → RunAs → RunConfiguration → JavaApplication →
Arguments tab:
Windows:
C:\project\employee.xlsx
Adding external Apache poi jars
Right click on project → properties → java buildpath





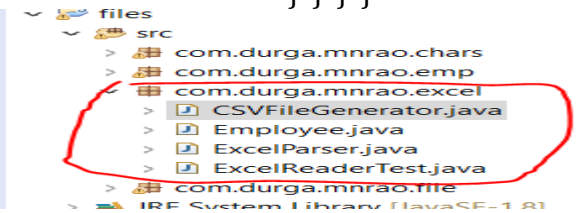


Same above example Generating CSV File
ExcelParser and Employee Classes are same
(anthor way main)

```
package com.durga.mnrao.excel;
import java.io.IOException;
import java.util.List;
public class ExcelReaderTest {
	public static void main(String[] args) throws IOException{
		if(args.length!=2){
			System.out.println("Missing input and output parameters ");
			System.exit(0);
		}
		String excelFilePath = args[0];
		String csvFilePath = args[1];
		List<Employee> empRecordsList = null;
		ExcelParser parser = new ExcelParser();
		empRecordsList =  parser.parseExcelData(excelFilePath);
		CSVFileGenerator  generator = new CSVFileGenerator();
		generator.generateCSVFile(csvFilePath, empRecordsList);
		System.out.println("Successfully Converted from excel to CSV ");
} }
package com.durga.mnrao.excel;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
public class CSVFileGenerator {
	public void generateCSVFile(String csvFilePath, List<Employee> empRecordsList) throws IOException{
		File filePath = new File(csvFilePath);
		FileOutputStream fos = null;
		DataOutputStream dos = null;
		try{
			fos = new FileOutputStream(filePath);
			dos = new DataOutputStream(fos);
			Iterator<Employee> iterator = empRecordsList.iterator();
			while(iterator.hasNext()){
				Employee emp = iterator.next();
				int empNum = emp.getEmpNum();
				String empName = emp.getEmpName();
				double empSalary = emp.getEmpSalary();
				String empDeptName = emp.getEmpDeptName();
				String empGender = emp.getEmpGender();
				int empAge = emp.getEmpAge();
	String record  empNum+","+empName+","+empSalary+","+empDeptName+","+empGender+","+empAge;
```

```
                    dos.writeBytes(record+"\n");
                }
                dos.close();
                fos.close();
            }
            catch(Exception e){
                e.printStackTrace();
            }
            finally{

                if(dos!=null){
                    dos.close();
                }
                if(fos!=null){
                    fos.close();
                } } } }
```



XML Parser

Using DOM Parser:

Sample Data:

Employee.xml:

==========

```
<employees>
  <employee>
        <id>111</id>
          <firstName>Lokesh</firstName>
          <lastName>Gupta</lastName>
          <location>India</location>
  </employee>
  <employee>
          <id>222</id>
          <firstName>Alex</firstName>
          <lastName>Gussin</lastName>
          <location>Russia</location>
  </employee>
  <employee>
          <id>333</id>
          <firstName>David</firstName>
          <lastName>Feezor</lastName>
          <location>USA</location>
  </employee>
  <employee>
          <id>444</id>
          <firstName>Russ</firstName>
          <lastName>Dawson</lastName>
          <location>USA</location>
  </employee>
</employees>
```

Result file emp.csv:

```
111,Lokesh,Gupta,India
222,Alex,Gussin,Russia
333,David,Feezor,USA
444,Russ,Dawson,USA
```

```
package com.nrit.mnrao.xml;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
```

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class XMLParser{
	public String parseXMLData(String xmlFilePath, String keyElement)
				throws IOException, InterruptedException{
			File file = new File(xmlFilePath);
			FileInputStream fis = new FileInputStream(file);
			DataInputStream dis = new DataInputStream(fis);
			// Get Document Builder
			DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
			DocumentBuilder builder = null;
			Document document = null;
			try{
				builder = factory.newDocumentBuilder();
			} catch (ParserConfigurationException e1){
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
			// Build Document
			try{
				document = builder.parse(dis);
			} catch (SAXException e)
			{
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
			// /Normalize the XML Structure; It's just too important !!
			document.getDocumentElement().normalize();
			// Here comes the root node
			Element root = document.getDocumentElement();
			System.out.println (root.getNodeName());
			// Get all employees
			NodeList nList = document.getElementsByTagName(keyElement);
			StringBuffer buffer = new StringBuffer();
			System.out.println (nList.getLength());
			for (int temp = 0; temp < nList.getLength(); temp++){
				Node node = nList.item(temp);
				System.out.println (""); // Just a separator
				if (node.getNodeType() == Node.ELEMENT_NODE){
					// Print each employee's detail
					Element eElement = (Element) node;
					// buffer.append(eElement.getAttribute("id")+",");
buffer.append(eElement.getElementsByTagName("id").item(0).getTextContent()+",");
buffer.append(eElement.getElementsByTagName("firstName").item(0).getTextContent( +",");
buffer.append(eElement.getElementsByTagName("lastName").item(0).getTextContent()+",");
buffer.append(eElement.getElementsByTagName("location").item(0).getTextContent());
	buffer.append("\n");
				}
			}
			String data = buffer.toString();
			System.out.println (data);
			return data;
	}   }
package com.nrit.mnrao.xml;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
public class CSVFile {
public void generateCSVFile(String [] records, String targetParth) throws IOException{
```

```
                File file = new File(targetParth);
                FileOutputStream fos = null;
                DataOutputStream dos = null;
                try{
                        fos = new FileOutputStream(file);
                        dos = new DataOutputStream(fos);
                        for (String string : records) {
                                System.out.println (string);
                                dos.writeBytes(string+"\n");
                        }
                        dos.close();
                        fos.close();
                }
                catch(Exception e){
                }
                finally{
                        if(dos!=null){
                                dos.close();
                        }
                        if(fos!=null){
                                fos.close();
                        } } } }
package com.visix.mnrao.xml;
import java.io.IOException;
public class Test {
        public static void main(String[] args) throws IOException, InterruptedException{
                String sourceFilePath = args[0];
                String keyElement=args[1];
                String targetFilePath = args[2];
                System.out.println (sourceFilePath);
                XMLParser parser = new XMLParser();
                String xmlData = parser.parseXMLData(sourceFilePath, keyElement);
                String [] records =  xmlData.split("\n");
                CSVFile  csvFile = new CSVFile();
                csvFile.generateCSVFile(records, targetFilePath);
                System.out.println("Successfully");
        }   }
```

In eclipse :
Right click on main () → RunAs → RunConfiguration→JavaApplication →
Arguments tab:
Windows:
D:\\hadoop\\employee.xml   employee   D:\\hadoop\\emp.csv
Linux:
/home/demo/hadoop/ employee.xml   employee   /home/demo/hadoop/ employee.csv
(source path of data   searchKey DestinationPath of Data)

Serialization

Serialization in java is a mechanism of *writing the state of an object into a byte stream*.

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called *deserialization* (Recollecting Object)

Advantage of Java Serialization

It is mainly used to travel object's state on the network (known as marshaling) or to store object into the file.

java.io.Serializable interface

Serializable is a marker interface (has no data member and method).

It is used to "mark" java classes so that objects of these classes may get certain capability.

The Cloneable and Remote are also marker interfaces.

The String class and all the wrapper classes implements *java.io.Serializable* interface by default.

Persistance permanent storage

Object Persistance : it is a stroring an Object into the file perminanently

For object persistence serialization required.

By implementing Serializable interface we can make class object as Serializable object

```
package com.durga.mnrao.serde;
import java.io.Serializable;
public class Employee implements Serializable{
		private int empNum;
		private String empName;
		private String empJob;
		private double empSalary;
		private String empDeptName;
		private String empGender;
		private int empAge;
		public int getEmpNum() {
			return empNum;
		}
		public void setEmpNum(int empNum) {
			this.empNum = empNum;
		}
		public String getEmpName() {
			return empName;
		}
		public void setEmpName(String empName) {
			this.empName = empName;
		}
		public double getEmpSalary() {
			return empSalary;
		}
		public void setEmpSalary(double empSalary) {
			this.empSalary = empSalary;
		}
		public String getEmpDeptName() {
			return empDeptName;
		}
		public void setEmpDeptName(String empDeptName) {
			this.empDeptName = empDeptName;
		}
		public String getEmpGender() {
			return empGender;
		}
		public void setEmpGender(String empGender) {
			this.empGender = empGender;
		}
		public int getEmpAge() {
			return empAge;
		}
		public void setEmpAge(int empAge) {
			this.empAge = empAge;
		}
		public String getEmpJob() {
			return empJob;
		}
		public void setEmpJob(String empJob) {
			this.empJob = empJob;
		}
	}
```

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Constructor

public ObjectOutputStream(OutputStream out) throws IOException

creates an ObjectOutputStream that writes to the specified OutputStream.

Methods

1) public final void writeObject(Object obj) throws IOException writes the specified object to the ObjectOutputStream.

2) public void flush() throws IOException flushes the current output stream.

3) public void close() throws IOException closes the current output stream.

```
package com.durga.mnrao.serde;
import java.io.File;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
public class ObjectPersistTest {
    public static void main(String[] args) {
    if(args.length!=1){
    System.out.println("Missing outytput file ");
                System.exit(0);
            }
            File filePath = new File(args[0]);
            FileOutputStream fos = null;
            ObjectOutputStream oos = null;
            try{
                fos = new FileOutputStream(filePath);
                oos = new ObjectOutputStream(fos);
                Employee employee = new Employee();
                employee.setEmpNum(1001);
                employee.setEmpName("mnrao");
                employee.setEmpJob("Architect");
    employee.setEmpSalary(50505.50);
                employee.setEmpDeptName("it");
                employee.setEmpGender("male");
                employee.setEmpAge(35);
                oos.writeObject(employee);
                employee = new Employee();
                employee.setEmpNum(1002);
                employee.setEmpName("scott");
                employee.setEmpJob("Manager");
                employee.setEmpSalary(60505.50);
                employee.setEmpDeptName("admin");
                employee.setEmpGender("female");
                employee.setEmpAge(40);
                oos.writeObject(employee);
                employee = new Employee();
                employee.setEmpNum(1003);
                employee.setEmpName("flag");
                employee.setEmpJob("clerk");
                employee.setEmpSalary(30505.50);
                employee.setEmpDeptName("sales");
                employee.setEmpGender("male");
                employee.setEmpAge(28);
                oos.writeObject(employee);
                oos.close();
                fos.close();
            System.out.println("Successfuly saved to file");
            }
            catch(Exception e){
```

```
                    e.printStackTrace();
                } } }
```

Deserialization

Deserialization is the process of reconstructing the object from the serialized state.It is the reverse operation of serialization.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

public ObjectInputStream(InputStream in) throws IOException
creates an ObjectInputStream that reads from the specified InputStream.

1) public final Object readObject() throws IOException, ClassNotFoundException

☒ reads an object from the input stream.

2) public void close() throws IOException ? closes **ObjectInputStream**

```
package com.durga.mnrao.serde;
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class ObjectDePersistTest {
    public static void main(String[] args) {
        if(args.length!=1){
            System.out.println("Missing input file ");
            System.exit(0);
        }
        File filePath = new File(args[0]);
        FileInputStream fis = null;
        ObjectInputStream ois = null;
        try{
            fis = new FileInputStream(filePath);
            ois = new ObjectInputStream(fis);
            Object obj = ois.readObject();
            if( obj instanceof Employee){
        Employee emp = (Employee) obj;
        int empNum = emp.getEmpNum();
        String empName = emp.getEmpName();
        String empJob = emp.getEmpJob();
        double empSalary = emp.getEmpSalary();
        String empDeptName = emp.getEmpDeptName();
        String empGender = emp.getEmpGender();
        int empAge = emp.getEmpAge();
    System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+emp
DeptName+"\t"+empGender+"\t"+empAge);
            }
            obj = ois.readObject();
            if( obj instanceof Employee){
            Employee emp = (Employee) obj;
        int empNum = emp.getEmpNum();
        String empName = emp.getEmpName();
         String empJob = emp.getEmpJob();
        double empSalary = emp.getEmpSalary();
        String empDeptName = emp.getEmpDeptName();
        String empGender = emp.getEmpGender();
                int empAge = emp.getEmpAge();
    System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+emp
DeptName+"\t"+empGender+"\t"+empAge);
            }
            obj = ois.readObject();
```

```
                    if( obj instanceof Employee){
                Employee emp = (Employee) obj;
                    int empNum = emp.getEmpNum();
                        String empName = emp.getEmpName();
                        String empJob = emp.getEmpJob();
                    double empSalary = emp.getEmpSalary();
                    String empDeptName = emp.getEmpDeptName();
                    String empGender = emp.getEmpGender();
                    int empAge = emp.getEmpAge();
System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"\t"+em
pAge);          }
                        ois.close();
                        fis.close();
                    }
                    catch(Exception e){
                        e.printStackTrace();
                    } } }
```

Java Serialization with static data member

If there is any static data member in a class, it will not be serialized because static is the part of class but not part of an object.

```
package com.durga.mnrao.serde;
import java.io.Serializable;
public class Employee implements Serializable{
        private int empNum;
        static private String empName;
        private String empJob;
        private double empSalary;
        private String empDeptName;
        private String empGender;
        static private int empAge;
        public int getEmpNum() {
            return empNum;
        }
        public void setEmpNum(int empNum) {
            this.empNum = empNum;
        }
        public String getEmpName() {
            return empName;
        }
        public void setEmpName(String empName) {
            this.empName = empName;
        }
        public double getEmpSalary() {
            return empSalary;
        }
        public void setEmpSalary(double empSalary) {
            this.empSalary = empSalary;
        }
        public String getEmpDeptName() {
            return empDeptName;
        }
        public void setEmpDeptName(String empDeptName) {
            this.empDeptName = empDeptName;
        }
        public String getEmpGender() {
            return empGender;
        }
        public void setEmpGender(String empGender) {
            this.empGender = empGender;
        }
        public int getEmpAge() {
            return empAge;
        }
        public void setEmpAge(int empAge) {
            this.empAge = empAge;
```

```
}
public String getEmpJob() {
        return empJob;
}
public void setEmpJob(String empJob) {
        this.empJob = empJob;
}  }
```

Java Serialization with array or collection

Rule:

In case of array or collection, all the objects of array or collection must be serializable. If any object is not serialiizable, serialization will be failed.

Java Transient Keyword

If you don't want to serialize any data member of a class, you can mark it as transient.

```
package com.durga.mnrao.serde;
import java.io.Serializable;
public class Employee implements Serializable{
        private int empNum;
        private String empName;
        private String empJob;
        transient private double empSalary;
        private String empDeptName;
        transient private String empGender;
        static private int empAge;
        public int getEmpNum() {
                return empNum;
        }
        public void setEmpNum(int empNum) {
                this.empNum = empNum;
        }
        public String getEmpName() {
                return empName;
        }
        public void setEmpName(String empName) {
                this.empName = empName;
        }
        public double getEmpSalary() {
                return empSalary;
        }
        public void setEmpSalary(double empSalary) {
                this.empSalary = empSalary;
        }
        public String getEmpDeptName() {
                return empDeptName;
        }
        public void setEmpDeptName(String empDeptName) {
                this.empDeptName = empDeptName;
        }
        public String getEmpGender() {
                return empGender;
        }
        public void setEmpGender(String empGender) {
                this.empGender = empGender;
        }
        public int getEmpAge() {
                return empAge;
        }
        public void setEmpAge(int empAge) {
                this.empAge = empAge;
        }
        public String getEmpJob() {
```

```
		return empJob;
	}
	public void setEmpJob(String empJob) {
		this.empJob = empJob;
	}
}
```

Reflection API

Java Reflection is a process of examining or modifying the run time behaviour of a class at run time.

The java.lang.Class class provides many methods that can be used to get metadata, examine and change the run time behaviour of a class.

Metadata: Data about data (information about other information)
The java.lang and java.lang.reflect packages provide classes for java reflection.

Where it is used

The Reflection API is mainly used in:

IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
Debugger, Test Tools etc.

Commonly used methods of Class class:

Method	Description
1) public String getName()	returns the class name
2) public static Class forName(String className)throws ClassNotFoundException	loads the class and returns the reference of Class class.
3) public Object newInstance()throws InstantiationException,IllegalAccessException	creates new instance.
4) public boolean isInterface()	checks if it is interface.
5) public boolean isArray()	checks if it is array.
6) public boolean isPrimitive()	checks if it is primitive.
7) public Class getSuperclass()	returns the superclass class reference.
8) public Field[] getDeclaredFields()throws SecurityException	returns the total number of fields of this class.
9) public Method[] getDeclaredMethods()throws SecurityException	returns the total number of methods of this class.
10) public Constructor[] getDeclaredConstructors()throws SecurityException	returns the total number of constructors of this class.
11) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException	returns the method class instance.

How to get the object of Class class?

There are 3 ways to get the instance of Class class. They are as follows:

forName() method of Class class
getClass() method of Object class

1) forName() method of Class class
is used to load the class dynamically.
returns the instance of Class class.
It should be used if you know the fully qualified name of class.
This cannot be used for primitive types only for the classes.

Let's see the simple example of forName() method.

```
package com.nrit.mnrao.test;
public class Employee {
	private int empNum;
	private String empName;
	private double empSalary;
	private String empDept;

	public Employee(){
	}
	public Employee(int empNum,String empName,double empSalary,String empDept ){
	}
	public int getEmpNum() {
		return empNum;
```

```
	}
	public void setEmpNum(int empNum) {
		this.empNum = empNum;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSalary() {
		return empSalary;
	}
	public void setEmpSalary(double empSalary) {
		this.empSalary = empSalary;
	}
	public String getEmpDept() {
		return empDept;
	}
	public void setEmpDept(String empDept) {
		this.empDept = empDept;
	}
	@Override
	public String toString() {
return "Employee [empNum=" + empNum + ", empName=" + empName + ", empSalary=" + empSalary + ", empDept="+
empDept + "]";
	}
}
package com.nrit.mnrao.sample;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
public class Test {
		public static void main(String[] args) {
			try	{
			Class  c = Class.forName("com.nrit.mnrao.test.Employee");
			System.out.println("Successfully loaded");
			System.out.println("class name : "+c.getName());
			Constructor [] conNames = c.getConstructors();
			System.out.println("It provides following constructors");
			for (Constructor conName : conNames) {
				System.out.println(conName);
			}
			Field [] fieldNames = c.getDeclaredFields();
			System.out.println("It provides following Instance Variables");
			for (Field fieldName : fieldNames) {
				System.out.println(fieldName);
			}
			Method [] methodNames = c.getDeclaredMethods();
			System.out.println("It provides following Methods");
			for (Method methodName : methodNames) {
				System.out.println(methodName);
			}
			Class superclass = c.getSuperclass();
			System.out.println("superclass  : "+superclass);
		}
		catch(ClassNotFoundException e){
			e.printStackTrace();
		} } }
```

Creating object by the user inplace of Class.forName()

```
package com.nrit.mnrao.sample;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import com.nrit.mnrao.test.Employee;
```

```
public class Test {
public static void main(String[] args) {
                try{
                        Employee  emp = new Employee();
                        Class<? extends Employee> c = emp.getClass();
        System.out.println("class name : "+c.getName());
                        Constructor [] conNames = c.getConstructors();
                        System.out.println("It provides following constructors");
                        for (Constructor conName : conNames) {
                                System.out.println(conName);
                        }
                        Field [] fieldNames = c.getDeclaredFields();
                        System.out.println("It provides following Data members");
                        for (Field fieldName : fieldNames) {
                                System.out.println(fieldName);
                        }
                        Method [] methodNames = c.getDeclaredMethods();
                        System.out.println("It provides following Methods");
                        for (Method methodName : methodNames) {
                                System.out.println(methodName);
                        }
                        Class superclass = c.getSuperclass();
                        System.out.println("superclass  : "+superclass);
                }
                catch(Exception e) {
                        e.printStackTrace();
                } } }
```

Multi Threading

Mutli Tasking:

It is a running multiple jobs simultaneously.

Multi tasking can be in two ways.

1) Process based Multi Tasking

2) Thread based Muti Tasking

Process based Multi Tasking:

In this for every job separate process creates and loads the job into the process. These processes are created at Operating System. OS has to manage all the processes.

If jobs are java program, then JVM loads into every process to execute those programs. Two JVMs can not share the data. Every one runs in its processes. Two JVM can not communicate each other.

RMI is the way to communicate between Two JVM. For example there are four jobs. Then OS creates four childs and loads four JVMs into that processes.

Process based job scheduling can be

1) Roud robin scheduling (Circular)

2) Priority based scheduling (four ground job will be given more priority and next is its back-ground

Advantage:

1.Process based is a safe one as no chance of crashing.

2.No chance of corrupting data as every process has it’s own memory space.

Dis-Advantage :

1.OS has to manage many processes. Hence heavy load on the OS. It is an heavy weight process.

2.Performance is less.

Eg: UNIX OS

Thread based Muti Tasking:

In this concept, only one child process creates for any no of threads and all threads will share same memory space.

Advantage: It is light weight technology and more performance.

Dis-Advantage : Chance of crashing threads.

Developer should expert in writing Multi thread coding.

Thread based Multi Tasking can be a round robin or priority based. For process based recommended one is round robin based

For thread based recommended one priority based.

Threading:

Thread is a part of an application. Thread is an independent path of execution.

Def: Execution of different parts of same application at the same time is called as multi-threading.

For any no of threads only one child process will be created and that can be shared by all threads of application.

Adv:

maximum utilization of CPU Time, so that we can get more performance.

What is a Thread ?

Ans: it is an Independent Path of execution. it is a part of application.

What is Mutli Threading ?

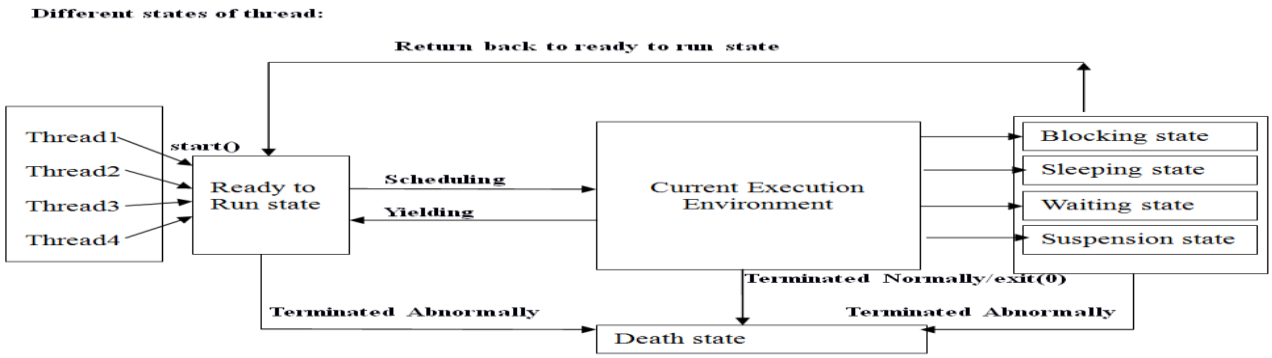
Ans: Multiple Parts of Same application running in parallel is called as Multi Threading

What is purpose of Multi Threading?

ans: to Utilize Maximum CPU Time.

What is Advantage of Multi Threading.

Ans: to Improve the performance of application.



JVM provides **Thread scheduler**, which manages different states of the thread.

1) Ready to run state:

It is a state of waiting for CPU time. A Thread comes to ready to run state, When start() method is called on thread.

2) Current execution state :

It is a state of assigning CPU time to thread

3) Blocking state:

Thread goes to blocking state when performing I/O operations with external resources, such as keyboard, database server and other servers.

4) Sleeping state:

It is a state of waiting for shared resources for specific amount of time. A thread goes to sleeping state when sleep() is called on thread.

5) waiting state

It is state of waiting for the shared resources until getting the resources. Thread goes to waiting state when wait() method called on thread.

6) suspend state:

If any child thread is going to perform any illegal transactions on the system resources, then main thread will suspend the child thread. If any thread is in suspend state it should be revoked by some other thread.

7) Death state:

A thread goes to death state, when terminated normally or abnormally (or) called exit(0) method on thread.

Once thread goes to death state it can not be invoked.

8) Scheduling : thread scheduler schedules the threads for execution. Thread scheduling can be done any one of the two ways.

1) priority scheduling (or) primitive scheduling

2) Round robin process.

9) yielding :

If any thread enters into ready to run state (new or old thread) thread scheduler checks priority of current execution thread with threads which are waiting for CPU time. If waiting thread is having more priority than the current execution thread, then thread scheduler forcibly brings the current execution thread into ready to run state.

Creation of a thread :

A thread can be created in two ways:

1) by extending from Thread class

2) by implementing Runnable interface

Recommended one is implements Runnable.

Thread class and Runnable interfaces are from java.lang package.

Runnable interface;

```
public interface Runnable  {

        public void  run();

}
```

run() should be implemented in the child class, which is acting as thread. run() contains thread coding.

Thread class:

This class implements the Runnable interface. public class Thread implements Runnable

```
{

}
```

Constructors of Thread class:

1) Thread()

        Thread t = new Thread (); default name of the thread is "child"

2) Thread(String name)

        Thread t = new Thread ("chid_name");

3) Thread(Runnable r, String name)

        Thread t = new Thread(this, "child_name");

Methods of Thread class:

1) String name = t.getName() returns name of the thread

2) String parent = t.getParent(); return parent name of the thread.

3) t.start() to take thread into ready to run state.

4) public static void sleep(int milliSeconds) to take thread into sleeping state.

1 sec = 1000 milli sec

Creation of a thread by extending from Thread class.

```
public class MyThread extends Thread  {
        public MyThread()  {
                super("child");
        }
        public void run()  {
                System.out.println ("Child thread started");
                try  {
                        for(int i=0;i<5;i++)            {
                                System.out.println ("child : "+i);
                                Thread.sleep(200);
                        }
                }
                catch(InterruptedException e)  {
                        System.out.println ("Child Interrupted");
                }
                System.out.println ("end of child thread");
        }
}
public class SingleThreadTest {
```

```
        public static void main(String[] args) {
                System.out.println ("parent started");
                Thread t = new MyThread();
                System.out.println ("child created");
                t.start();
                try     {
                        for (int i = 0; i < 5; i++)     {
                                System.out.println ("Parent : " + i);
                                Thread.sleep(200);
                        }
                }
             catch (InterruptedException e) {
                System.out.println ("Parent Interrupted");
            }
                System.out.println ("end of Parent thread");
        }
}
```

o/p:

```
parent started
child created
Parent : 0
Child thread started
child : 0
Parent : 1
child : 1
Parent : 2
child : 2
Parent : 3
child : 3
Parent : 4
child : 4
end of Parent thread
end of child thread
```

Creation of thread by implementing Runnable interface.

```
public class MyThread implements Runnable{
        Thread t;
        public MyThread(){
                t=new Thread(this,"child");
                t.start();
        }
        public void run()   {
                System.out.println ("Child thread started");
                try{
                        for(int i=0;i<5;i++){
                                System.out.println ("child : "+i);
                                Thread.sleep(200);
                        }
                }
                catch(InterruptedException e)           {
                        System.out.println ("Child Interrupted");
                }
                System.out.println ("end of child thread");
        }
}
public class ThreadTest{
        public static void main(String[] args)  {
                System.out.println ("parent started");
                new MyThread();
                try         {
                        for (int i = 0; i < 5; i++)         {
                                System.out.println ("Parent : " + i);
                                Thread.sleep(200);
                        }
```

```
                }
                catch (InterruptedException e){
                        System.out.println ("Parent Interrupted");
                }
                System.out.println ("end of Parent thread");
        }  }
```

o/p:

```
parent started
child created
Parent : 0
Child thread started
child : 0
Parent : 1
child : 1
Parent : 2
child : 2
Parent : 3
child : 3
Parent : 4
child : 4

end of child thread

end of Parent thread
```

Creating Multiple threads:

```
public class MyThread implements Runnable {
        Thread t;
        public MyThread(String name)  {
                t=new Thread(this,name);
                t.start();
        }
        public void run()  {
                String name = t.getName();
                System.out.println ("Child Name : "+name);
                        try  {
                        for(int i=0;i<5;i++){
                                System.out.println (name+"  : "+i);
                                Thread.sleep(200);
                        }
                }
                catch(InterruptedException e){
                        System.out.println ("Child Interrupted name : "+name);
                }
                System.out.println ("end of child thread name : "+name);
                }
}
public class ThreadTest{
        public static void main(String[] args)  {
                System.out.println ("parent started");
                String [] threadNames = {"one","two","three","four","five"};
                try         {
                    for (int i = 0; i < threadNames.length; i++) {
                                new MyThread(threadNames[i]);
                    }
                        Thread.sleep(8000);
                }
                catch (InterruptedException e){
                        System.out.println ("Parent Interrupted");
                }
                System.out.println ("end of Parent thread");
        }
}
```

Thread priority:

Priority is Ranging from 1 to 10
Thread.***MAX_PRIORITY*** =10;
Thread.***MIN_PRIORITY*** = 1;
Thread.***NORM_PRIORITY*** = 5; it is a default priority.
Methods
public void setPriority(int priority) : to set priority of the thread.
public int getPriority( ) : to get priority of the thread.

```
public class MyThread implements Runnable{
        Thread t;
        public MyThread(String name, int priority){
                t=new Thread(this, name);
                t.setPriority(priority);
                t.start();
        }
        public void run(){
                System.out.println ("Child Name : "+t.getName());
                System.out.println ("Priority : "+t.getPriority());
                System.out.println ("End ");
        }
}
public class ThreadTest{
        public static void main(String[] args) {
        String [] threadNames={"One","Two","Three","Four","Five","Six","Seven","eight","nine"};
                Thread currentThread = Thread.currentThread();
                currentThread.setPriority(10);
                for (int i = 0; i < threadNames.length; i++){
                        new MyThread(threadNames[i], i+1);
                }
                System.out.println ("Childs are created");
                System.out.println ("End of parent");
        } }
```

In the above example parent thread dies first and then child . Hence child threads becomes orphaned threads.
In this scenario we can use join() to wait for the child threads to complete.

Methods:

1) public boolean isAlive(); return true if thread is alive otherwise false.
2) public void join(); to wait for the child thread.

```
public class MyThread implements Runnable{
        Thread t;
        public MyThread(String name){
                t = new Thread(this, name);
                t.start();
        }
        public void run(){
                System.out.println ("Child Name : "+t.getName());
                System.out.println ("End ");
        }
}
public class ThreadTest{
        public static void main(String[] args) {
                MyThread t1 = new MyThread("One");
                MyThread t2 = new MyThread("Two");
                MyThread t3 = new MyThread("Three");
                System.out.println ("One is in Alive "+t1.t.isAlive());
                System.out.println ("Two is in Alive "+t2.t.isAlive());
                System.out.println ("Three is in Alive "+t3.t.isAlive());
                try{
                        t1.t.join();
                        t2.t.join();
                        t3.t.join();
                }
            catch (InterruptedException e){
                        System.out.println (e.getMessage());
                }
                System.out.println ("One is in Alive "+t1.t.isAlive());
                System.out.println ("Two is in Alive "+t2.t.isAlive());
```

```
                System.out.println ("Three is in Alive "+t3.t.isAlive());
                System.out.println ("End of parent");
        } }
```

Thread synchronization:

When Multiple threads are trying to access shared resources, there is a chance of crashing threads. To avoid thread crashing, developer has to write synchronization coding.

Methods to synchronize a thread.

1) public void wait()
   this method takes the thread into waiting state. It throws InterruptedException.
2) public void notify()
   this method is to invoke the first waiting thread in the waiting queue.
3) public void notifyAll()
   this method is to invoke all waiting threads, in wating queue
   the above three methods are from Object class, not from thread class. Hence these can be called direcly.

```
public class Queue{
        int data=0;
        boolean state = false;
        public synchronized void pop(){
                if( ! state )           {
                        try{
                                wait();
                        }
                        catch(InterruptedException  e){
                          System.out.println ("Consumer Interrupted");
                        }
                }
        System.out.println ("consumed Data "+data);
                state=false;
                notify();
        }

        public synchronized void push(int num){
                        if(state)  {
                        try        {
                                wait();
                        }
                        catch(InterruptedException  e)                  {
                          System.out.println ("Consumer Interrupted");
                        }
                }
                data=num;
                System.out.println ("Produced Data "+data);
                state=true;
                notify();
        }
}
public class Producer  implements Runnable{
        Thread t;
        Queue q1;
        public Producer(Queue q)    {           q1=q;
                t=new Thread(this,"producer");
                t.start();
        }
        public void run()  {
                System.out.println ("Producer started");
                int i=0;
                while(true)                     {
                        i++;
                        q1.push(i);
                } } }
public class Consumer implements Runnable{
        Thread t;
        Queue q1;
        public Consumer(Queue q)  {
                q1=q;
```

```
                t=new Thread(this,"consumer");
                t.start();
        }
        public void run()   {
                System.out.println ("consumer started");
                while(true)                     {
                        q1.pop();
                } } }
public class ProdConsumer{
        public static void main(String[] args)  {
                Queue q = new Queue();
                Consumer c1 = new Consumer(q);
                Producer p1 = new Producer(q);
                System.out.println ("Press Ctrl + C  for End ");
        }
}
```

Java Versions, Features and History

- **Released on 23 January 1996, JDK 1.0 version**

- **Released on 19 February 1997 JDK 1.1 version.**
  New features in JDK 1.1
  1. JDBC (Java Database Connectivity)
  2. Inner Classes
  3. Java Beans
  4. RMI (Remote Method Invocation)
  5. Reflection (introspection only)

- **Released on 8 December 1998 J2SE 1.2 version.**

  New features in J2SE 1.2

  Collections framework.

  1. Java String memory map for constants.
  2. Just In Time (JIT) compiler.
  3. Jar Signer for signing Java ARchive (JAR) files.
  4. Policy Tool for granting access to system resources.
  5. Java Foundation Classes (JFC) which consists of Swing 1.0, Drag and Drop, and Java 2D class libraries.
  6. Java Plug-in
  7. Scrollable result sets, BLOB, CLOB, batch update, user-defined types in JDBC.
  8. Audio support in Applets.

- **Released on 8 May 2000 J2SE 1.3 version.**
  New features in J2SE 1.3
  1. Java Sound
  2. Jar Indexing
  3. A huge list of enhancements in almost all the java area.

- **Released on 6 February 2002 J2SE 1.4 version.**
  New features in J2SE 1.4

1. XML Processing	2. Assertions
3. Java Print Service	4. Preferences API
5. Logging API	6. Chained Exception
7. Java Web Start	8. IPv6 Support
9. JDBC 3.0 API	10. Regular Expressions
	11. Image I/O API

- **Released on 30 September 2004 J2SE 1.5 version.**
  New features in J2SE 1.5
  1. Generics
  2. Enhanced for Loop
  3. Autoboxing/Unboxing
  4. Enum
  5. Varargs

6. Static Import
7. Metadata (Annotations)
8. Instrumentation

- **Released on 11 December 2006 J2SE 1.6 version.**
  New features in J2SE 1.6
  1. Scripting Language Support
  2. JDBC 4.0 API
  3. Java Compiler API
  4. Pluggable Annotations
  5. Native PKI, Java GSS, Kerberos and LDAP support.
  6. Integrated Web Services.
  7. Lot more enhancements.

- **Released on 28 July 2011 J2SE 1.7 version.**
  New features in J2SE 1.7
  1. Strings in switch Statement
  2. Type Inference for Generic Instance Creation
  3. Multiple Exception Handling
  4. Support for Dynamic Languages
  5. Try with Resources
  6. Java io Package
  7. Binary Literals, underscore in literals
  8. Diamond Syntax
  9. Automatic null Handling

Released on 18th march 2014 JDK 1.8 version.
New features in JDK 1.8

1. Default and Static methods in Interface
2. Lambda Expressions
3. Optional
4. Streams
5. Method References
6. Data Time API
7. Na shorn Java-script Engine
8. Parallel Arrays

Some of the important java 9 features are;

1. Java 9 REPL (JShell)	2. Private methods in Interfaces
3. Java 9 Module System	4. Process API Improvements
5. Try With Resources Improvement	6. CompletableFuture API Improvements
7. Reactive Streams	8. Diamond Operator for Anonymous Inner Class
9. Optional Class Improvements	10. Stream API Improvements
11. Enhanced @Deprecated annotation	12. HTTP 2 Client
13. Private methods in Interfaces	14. Multi-Resolution Image API
15. Miscellaneous Java 9 Features	16. Factory Methods for Immutable List, Set, Map and Map .Entry

----------End----------