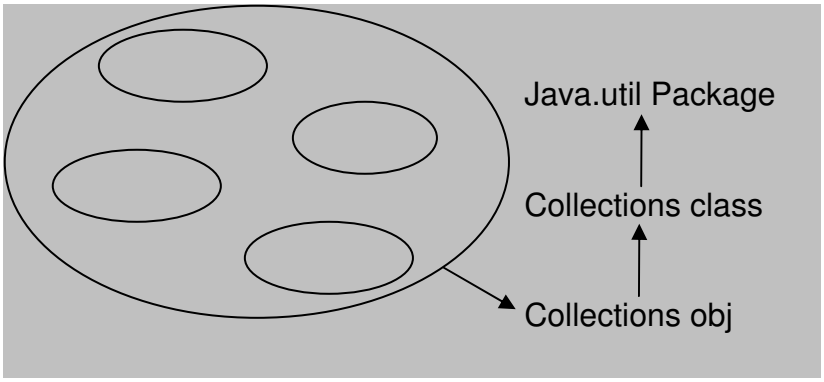
# ADVANCED JAVA

- A group of elements are handled by representing with an array.
- A group of objects are also handled with array.

To store objects in Array

```
Employee arr[ ] = new Employee[100];
  for(int i = 0; i<100; i++)
  {
    arr[ i ] = new Employee(……);
  }
```




**Collections object (or) Container Object: -** It is an object that stores a group other objects.

**Collections class (or) Container class: -** It is a class whose object can store other objects.

All collection classes have defined in java.util package.

Frame work means group of classes.

- **What is collections frame work? *******

**Ans: -** Collections frame work is a class library to handle groups of objects. Collection frame work is defined in java.util package.

- **What is the difference between collection and Collections? *******

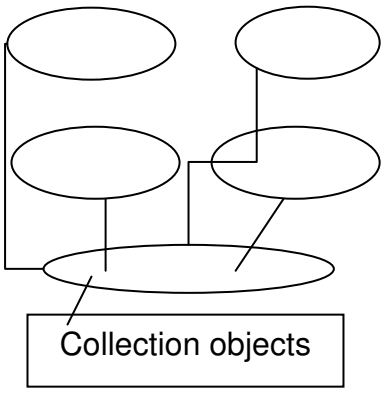
**Ans: -** Collection is an interface.

Collections is a class.

 rambabumandalapu@yahoo.com

Collections objects will not act upon primitive datatypes. (or) collections objects will not store primitive datatypes.

Collections objects stores references of other objects. All collection classes are defined into 3 types.




**1. <u>Sets</u>: -** A set represents a group of objects is called as Set.

**Ex: -** HashSet, LinkedHashSet, TreeSet, etc…

**2. <u>Lists</u>: -** A list represents the group of elements or objects. List is also similar to set. It will also store elements or objects. Sets will not allow the duplicate values but Lists will duplicate values. This is the difference between Lists and Sets.

**Ex: -** ArrayList, Vector, LinkedList etc…

❖ **Which of the collection class will not allow duplicate values? *******

**Ans: -** Sets of the collection class will not allow the duplicate values. Set will not stores the object in ordered/sorted order.

Lists of the collection class will allow the duplicate values. List will stores the object in ordered/sorted order.

❖ Sorted means iterating through a collection in a natural sorted order.

**3. Maps: -** A map stores elements in the form of the keys and value pairs.

**Ex: -** Hashmap, Hashtable etc…

**List of interfaces and classes in java.util package (or) Collection framework:**

| S. No. | Interfaces | Collection Classes |
|---|---|---|
| 1. | Set | ArrayList |
| 2. | List | Vector |
| 3. | Map | LinedList |
| 4. | SortedSet | HashSet |
| 5. | SortedMap | TreeSet |
| 6. | Iterator | Collections |
| 7. | ListIterator | LinkedHashSet |
| 8. | Enumeration | HashMap |

 rambabumandalapu@yahoo.com

| | | |
|---|---|---|
| **9.** | | TreeMap |
| **10.** | | HashTable |
| **11.** | | LinkedHashMap |

**ArrayList: -** It is dynamically growing array that stores objects. It is not synchronized[1].

**1.** To create an ArrayList

```
ArrayList arl = new ArrayList();
ArrayList arl = new ArrayList(100);
```

**2.** To add objects use add() method.

```
arl.add("Apple");
arl.add(2, "Apple");
```

**3.** To remove objects use remove() method.

```
arl.remove("Apple");
arl.remove(2);
```

**4.** To know number of objects use size() method.

```
int n = arl.size();
```

**5.** To covert ArrayList into an array use toArray() method.

```
object x[ ] = arl.toArray();
```

- Object is the super class of other classes including user defined classes also.
- When we are retreiveing the elements from ArrayList, it will maintains and gives the same order what we have added the elements to the ArrayList.
- ArrayList doesn't supports the null values.
- ArrayList supports sort method by using the below code

```
Collections.sort(al);  //al is ArrayList Class object reference.
```

**Ex: -**

```
//ArrayList creation with string objects
import java.util.*;
class ArrayListDemo
{
  public static void main(String args[ ])
  {
    //Create an ArrayList
```

[1] **Synchronized: -** It means only one process will allow to act on one object.

```
ArrayList arl = new ArrayList();
//Add elements to arl
arl.add("Apple");          //Θ Apple, Grapes, Guava, Banana …are objects
arl.add("Grapes");
arl.add("Guava");
arl.add("Banana");
arl.add("Mango");
//Display the content of ArrayList (or) in arl
System.out.println("Array list = " +arl);
//Remove elements
arl.remove("Apple");
arl.remove(2);
//Display the content of ArrayList (or) in arl
System.out.println("Array list = " +arl);
//Find number of elements in arl
System.out.println("Size of Array list = " +arl.size());
//Use Terator to retrieve elements
Iterator it = arl.iterator();
while(it.hasNext())
System.out.println(it.next());
}
}
```

There are three types of interfaces are available, which are useful to retrieve the objects or elements one by one from array list. They are

1. Iterator
2. ListIterator
3. Enumeration

| **What are the difference between Iterator and ListIterator: -** | |
|---|---|
| **Iterator** | **ListIterator** |
| **1.** Iterator supports only hasNext(), next() and remove() methods. | **1.** Iterator supports add(), set(), next(), hasNext(), previous(), hasPrevious(), nextIndex(), previousIndex() and |

| | |
|---|---|
| | remove() methods. |
| **2.** Access the collections in the forward direction only. | **2.** Access the collections in forward and backward directions. |
| **3.** Iterator is a super interface | **3.** ListIterator is the sub interface of Iterator super interface. |

| **What are the difference between Iterator and Enumeration: -** | |
|---|---|
| **Iterator** | **Enumeration** |
| **1.** Iterator supports a remove() method. | **1.** Enumeration doesn't supports a remove() method. |
| **2.** It is not synchronized. | **2.** It is synchronized. |
| **3.** Iterator supports ArrayList, Vector, HashMap, HashTable. | **3.** Enumeration doesn't supports ArrayList, HashMap. |
| **4.** It doesn't supports legacy methods. | **4.** It supports legacy methods like as hasMoreElements(), nextElement(). |

**Vector: -** It is a dynamically growing array that stores objects. But it is synchronized.

When the object is synchronized then we will get reliable results or values. Vectors are suitable objects.

**1.** To create a vector

```
Vector v = new Vector();
Vector v = new Vector (100);
```

**2.** To know the size of Vector, use size() method.

```
int n = v.size();
```

**3.** To add elements use add() method.

```
v.add(obj);
v.add(2, obj);
```

**4.** To retrieve elements use get() method

```
v.get(2);
```

**5.** To remove elements use remove() method

```
v.remove(2);
```

To remove all elements

 rambabumandalapu@yahoo.com

```
v.clear();
```

**6.** To know the current capacity, use capacity() method

```
int n = v.capacity();
```

**7.** To search for last occurrence of an element use indexOf() method

```
int n = v . intdexOf(obj);
```

**8.** To search for last occurrence of an element use lastIndexOf() method

```
int n = v . lastIndexOf(obj);
```

**9.** 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.

```
void ensureCapacity(int minCapacity);
```

- ArrayList is not synchronized by default, but by using some methods we can synchronized the ArrayList.
- List myList = Collections.synchronizedList(myList);
- Vector is synchronized default.
- When we are retreiveing the elements from Vector, it will maintains and gives the same order what we have added the elements to the Vector.
- Vector doesn't supports the null values.

**Ex: -**

```
//A Vector of int values
import java.util.*;
class VectorDemo
{
  public static void main(String args[ ])
  {
    //Create an empty Vector
    Vector v = new Vector ();
    //Take (an one dimensional) int type array
    int x [ ] = {10, 22, 33, 44, 60, 100};
    //Read int values from x and store into v
    for(int i = 0; i< x.lenght(); i++)
      v.add(new Integer(x[ i ]));
    //Retrieve and display the elements
    for(int i = 0; i< v.size(); i++)
    System.out.println(v.get(i));
    //Retrieve elements using ListIterator
```

```
        ListIterator lit = v. listIterator();
    //Θ In the above statement ListIterator is an Interface, listIterator() is a method
        System.out.println("In forward direction:  " );
        while(lit.hasNext())
        System.out.print (lit.next() +"\t");
        System.out.println ("\n In reverse direction: ");
        while(lit.previous())
        System.out.print(lit.previous() + "\t");
       }
    }
```

| **What are differences between ArrayList and Vector: -** | |
|---|---|
| **ArrayList** | **Vector** |
| **1.** An ArrayList is dynamically growing array that stores objects. | **1.** Vector is dynamically growing array that stores objects. |
| **2.** It is not synchronized. | **2.** It is synchronized. |
| **3.** It is efficient than Vector, due to ArrayList is fast iterartion and fast random access. | **3.** It is not efficient than ArrayList, it is slower than ArrayList due to its synchronized methods. |
| **4.** ArrayList supports only Iterator interface. | **4.** Vector supports Iterator and enumeration interfaces. |
| **5.** ArrayList doesn't supports the legacy methods. | **5.** Vector supports the legacy methods like hasMoreElements(), nextElement(). |
| **6.** ArrayList doesn't supports the capacity(). Its default capacity is 10. | **6.** Vector supports the capacity(). The default capacity of vector is 10. |

**Maps: -** A map represents storage of elements in the form of the key and value pairs. Keys must be unique and keys cannot allow duplicate values.

**HashTable: -** Hashtable stores object in the form of keys and value pairs. It is synchronized.

**1.** To create a HashTable

```
HashTable ht = new HashTable();          //Initial
Capacity = 11, load factor = 0.75
HashTable ht = new HashTable(100);
```

**2.** To store key-Value pair the HashTable

```
ht.put("Sachin", "Cricket Player");
```

**3.** To get the value when is given

```
ht.get("Sachin");
```

**4.** To remove the key (and its corresponding value)

```
ht.remove("Sachin");
```

**5.** To know the number of keys in the HashTable.

```
int n = ht.size();
```

**6.** To clear all the keys.

```
ht.clear();
```

❖ **What is load factor? *****

**Ans: -** Load factor determines the point at which the capacity of HashTable or HashMap will be automatically doubled.

**Ex: -** For HashTable initial capacity (11) X load factor (0.75) = 8

i.e. After storing 8th pair the capacity of the HashTable will be doubled i.e. becomes 22 (11 x 2).

Initial capacity = 11, this value depends on the version.

**Ex: -**

```
//HashTable with Cricket Scores
import java.util.*;
import java.io.*;
class HashtableDemo
{
  public static void main(String args[ ])
  throws IOException
  {
    //Create an empty hash table
   HashTable ht = new HashTable();
   //Store Player Name, Score
  //Θ Here Player Name is Key and Score is a Value
  ht.put("Amarnadh", new Integer(50));
  ht.put("Sachin", new Integer(150));
  ht.put("Dhoni", new Integer(125));
  ht.put("Kapil", new Integer(175));
```

 rambabumandalapu@yahoo.com

```
        ht.put("Ganguly", new Integer(86));
        //Retrieve all keys
        Enumeration e = ht.keys();
        System.out.println("Player Name: ");
        While(e.hasMoreElements())
        System.out.println(e.nextElement());
        //Ask for Player Name from Keyboard
        BufferedReader br = new BufferedReader(InputStreamReader(System.in));
         //Θ In the above statement (System.in) represents Keyboard
        System.out.println("Enter Player Name: ");
        String name = br.readLine();
        //Find number of runs of this player
        Integer score = (Integer) ht.get(name);
        if(score!=null)
        {
          int runs = score.intValue();
          System.out.println(Name+"Scored runs = "+runs);
         }
         else
         System.out.println("Player not found");
        }
       }
```

**Ex 2: -**

```
import java.util.*;
public class DemoHashTable {
public static void main(String[] args) {
Hashtable ht=new Hashtable();
ht.put("1", "Babu");
ht.put("2", "Anand");
ht.put("3", "Mogili");
ht.put("0", "@Marlabs");
System.out.println("---Retreiveing the elements--- = "+ht);
System.out.println(ht.get("1"));
System.out.println(ht.get("2"));
```

```
System.out.println(ht.get("3"));
System.out.println(ht.get("0"));
Enumeration e=ht.elements();
while(e.hasMoreElements()){
System.out.println("---Retreving the elements using enumeration from HashTable = "+e.nextElement());
}
Enumeration e1=ht.keys();
while(e1.hasMoreElements()){
        System.out.println("---Retreving the keys using enumeration from HashTable = "+e1.nextElement());
}
}
}
```

**O/P: -**

```
---Retreiveing the elements--- = {3=Mogili, 2=Anand, 1=Babu, 0=@Marlabs}
Babu
Anand
Mogili
@Marlabs
---Retreving the elements using enumeration from HashTable = Mogili
---Retreving the elements using enumeration from HashTable = Anand
---Retreving the elements using enumeration from HashTable = Babu
---Retreving the elements using enumeration from HashTable = @Marlabs
---Retreving the keys using enumeration from HashTable = 3
---Retreving the keys using enumeration from HashTable = 2
---Retreving the keys using enumeration from HashTable = 1
---Retreving the keys using enumeration from HashTable = 0
```

**Ex 3: -**

```
import java.util.*;
public class DemoHashTable {
public static void main(String[] args) {
Hashtable ht=new Hashtable();
ht.put("1", "Babu");
```

```
ht.put(null, "Anand"); (or) ht.put("2", null);
ht.put("3", "Mogili");
ht.put("0", "@Marlabs");
System.out.println("---Retreiveing the elements--- = "+ht);
System.out.println(ht.get("1"));
System.out.println(ht.get("2"));
System.out.println(ht.get("3"));
System.out.println(ht.get("0"));
Enumeration e=ht.elements();
while(e.hasMoreElements()){
        System.out.println("---Retreving the elements using enumeration from HashTable = "+e.nextElement());
}
Enumeration e1=ht.keys();
while(e1.hasMoreElements()){
        System.out.println("---Retreving the keys using enumeration from HashTable = "+e1.nextElement());
}
}
}
```

**O/P: -**

```
Exception in thread "main" java.lang.NullPointerException
        at java.util.Hashtable.put(Unknown Source)
        at com.marlabs.vara.DemoHashTable.main(DemoHashTable.java:7
```

- Enumeration will not maintain objects in the same order.
- IOException may caused by readLine() method. IO stands for input/output.
- Playernames are all keys and scores are all values.
- When we retreving the elements or keys using HashTable it will gives the elements or keys in revrse order only.
- In HashTable duplicate keys are not allowed but duplicate values are allowed.

**HashMap: -** HashMap stores objects in the form of keys and value pairs. It is not synchronized.

**1.** To create a HashMap

```
HashMap hm = new HashMap();  //Initial
Capacity = 16, load factor = 0.75
HashMap hm = new HashMap(101);
```

**2.** To store key-Value pair the HashMap

```
hm.put("Sachin", "Cricket Player");
```

**3.** To get the value when is given

```
hm.get("Sachin");
```

**4.** To remove the key (and its corresponding value)

```
hm.remove("Sachin");
```

**5.** To know the number of key-value pairs in the HashMap.

```
int n = hm.size();
```

**6.** To clear all the keys.

```
hm.clear();
```

- HashMap is similar to HashTable but the difference between these two is HashTable is synchronized but HashMap is not synchronized
- When we retreving the elements or keys using HashMap it will gives the elements or keys irregular order, not even the same order that we have added keys and elements.
- In HashMap duplicate keys are not allowed but duplicate values are allowed.
- Map myMap = Collections.synchronizedMap(myMap);

**Ex: -**

```
//Telephone entry book
import java.util.*;
import java.io.*;
class Tele
{
  public static void main(String args[ ])
  throws IOException
  {
    //Vars
   HashMap hm = new HashMap();
   String name, str;
   Long phno;
   BufferedReader br = new BufferedReader(InputStreamReader(System.in));
  //Menu
```

 rambabumandalapu@yahoo.com

```
while(true)   //It is an infinite loop
{
  System.out.println("1 Enter entries into Phone Book");
  System.out.println("2 Lookup for a Phone Number");
  System.out.println("3 Exit");
  int n = Integer.parseInt(br.readLine());
  //Depending on n value, perform a task
  switch(n)
  {
    case 1:
    System.out.print("Enter Person Name: ");
    Name = br.readLine();
    System.out.print("Enter Phone No. ");
    str = br.readLine();
    //Convert str into Long obj
    phno = new Long(str);
    //Store name, phone in hm
    break;
    case 2:
    System.out.print("Enter Person Name");
    name = br.readLine();
    //Pass name to hm and get the Phone No.
    phno = (Long)hm.get(name);
    System.out.println("Phone No: "+phno);
    break;
    default:
    return;
    }
  }
}
}
```

**Ex 1: -**

```
import java.util.*;
public class DemoHashmap {
```

```
        public static void main(String args[]){
        HashMap hm=new HashMap();
        System.out.println("---HasMap initial size---"+hm.size());
        hm.put("1", "Vara");
        hm.put("2", "anji");
        hm.put("3", "anand");
        hm.put("4", "bujji");
                System.out.println("---HasMap size---"+hm.size());
        System.out.println("---Retrieving the elements HasMap---"+hm);
        System.out.println(hm.get("4"));
        System.out.println(hm.get("1"));
        System.out.println(hm.get("3"));
        System.out.println(hm.get("2"));
        Set s=hm.keySet();
        Iterator it=s.iterator();
        while(it.hasNext())
        System.out.println("Retreiveing the Keys = "+it.next());
        }
}
```

**O/P: -**

```
---HasMap initial size---0
---HasMap size---4
---Retrieving the elements HasMap---{3=anand, 2=anji, 4=bujji, 1=Vara}
bujji
Vara
anand
anji
Retreiveing the Keys = 3
Retreiveing the Keys = 2
Retreiveing the Keys = 4
Retreiveing the Keys = 1
```

**Ex 2: -**

```
        import java.util.*;
        public class DemoHashmap {
```

```
        public static void main(String args[]){
        HashMap hm=new HashMap();
        System.out.println("---HasMap initial size---"+hm.size());
        hm.put("1", "Vara");
        hm.put(null, null);
        hm.put("3", null);
        hm.put("4", "bujji");
                System.out.println("---HasMap size---"+hm.size());
        System.out.println("---Retrieving the elements HasMap---"+hm);
        System.out.println(hm.get("4"));
        System.out.println(hm.get("1"));
        System.out.println(hm.get("3"));
        System.out.println(hm.get("2"));
        Set s=hm.keySet();
        Iterator it=s.iterator();
        while(it.hasNext())
        System.out.println("Retreiveing the Keys = "+it.next());
        }
}
```

**O/P: -**

```
---HasMap initial size---0
---HasMap size---4
---Retrieving the elements HasMap---{null=null, 3=null, 4=bujji, 1=Vara}
bujji
Vara
null
null
Retreiveing the Keys = null
Retreiveing the Keys = 3
Retreiveing the Keys = 4
Retreiveing the Keys = 1
```

**What are differences between HashMap and HashTable: -**

| HashMap | HashTable |
|---|---|
| **1.** It stores the objects in the form of key and value pairs. | **1.** It stores the objects in the form of key and value pairs. |
| **2.** It is not synchronized. | **2.** It is synchronized. |
| **3.** It will makes fastest updates (key/value pairs) so it is efficient than HashTable. | **3.** It is not efficient than HashMap due to its synchronized methods. |
| **4.** It supports only Iterator interface. | **4.** HashTable supports Iterator and enumeration interfaces. |
| **5.** It doesn't supports legacy methods. | **5.** It supports legacy methods like hasMoreElement(), nextElement(). |
| **6.** HashMap initial capacity is 16 and the capacity of HashMap depends on version of Java. | **6.** HashTable initial capacity is 11 and the capacity of HashTable depends on version of Java. |
| **7.** HashMap doesn't maintained the orderd of elements, not even the keys and the elements added to it. | **7.** HashTable maintained the orderd of elements in reverse. |
| **8.** HapMap takes only one null key and many null values. | **8.** HashTable doesn't takes null keys and null values. |

**StringTokenizer: -** The StringTokenizer class is useful to break a string into small pieces, called tokens.

**1.** To create an object to StringTokenizer

StringTokenizer st = new StringTokenizer(str, "delimiter"); (or)

StringTokenizer st = new StringTokenizer(str, ","); (or)

StringTokenizer st = new StringTokenizer(str, ", :");

Θ Here , : are called as delimiters

**2.** To find the next piece in the string.

String piece = st.nextToken();

**3.** To know if more pieces are remaining.

booleab x = st.hasMoreTokens();

**4.** To know how many number of pieces are there.

int no = st.countTokens();

- Token means piece of string.

**Ex: -**

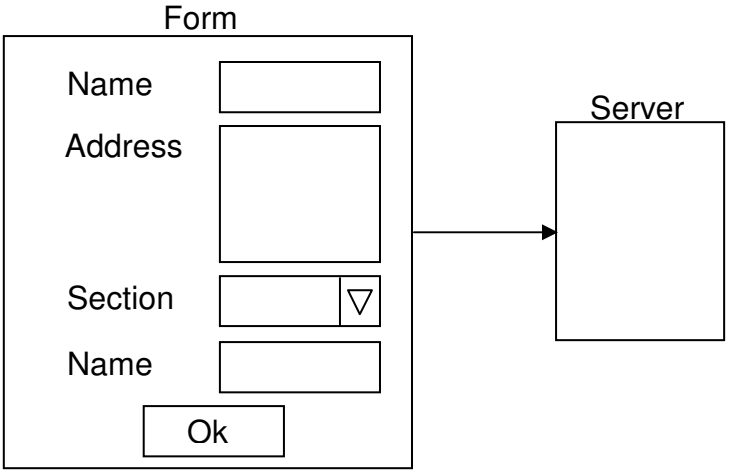
```
//Cutting the string into pieces
import java.util.*;
class STDemo
{
  public static void main(String args[ ])
   {
    //Take a string
    String str = "It is our capital city called New Delhi";
    //brake the string into species
    StringTokenizer st = new StringTokenizer(str, " ");
    //retrieve the pieces and display
    System.out.println("The token are: ");
    while(st.hasMoreTokens())
    {
     String s1 = st.nextToken();
     System.out.println(s1);
    }
   }
}
```

**(or)**

```
//Cutting the string into pieces
import java.util.*;
class STDemo
{
  public static void main(String args[ ])
   {
    //Take a string
    String str = "It, is our: capital city, called New: Delhi";
    //brake the string into species
    StringTokenizer st = new StringTokenizer(str, ", : ");
    //retrieve the pieces and display
   System.out.println("The token are: ");
   while(st.hasMoreTokens())
   {
```

 rambabumandalapu@yahoo.com

```
            String s1 = st.nextToken();
            System.out.println(s1);
          }
        }
      }
```




**Calendar: -** This class is useful to handle, date & time.

**1.** To create an object to calendar class:

Calendar cl = Calendar.getInstance();

**2.** Use get() method to retrieve date or time from calendar object. This method returns an integer.

cl.get(constant);

**Note: -**

**Constants: -**

Calendar.DATE

Calendar.MONTH

Calendar.YEAR

Calendar.HOUR

Calendar.MINUTE

Calendar.SECOND

**3.** Use set() to get the date or time in the calendar object.

cl.set(calendar.MONTH,10); //Default counting of January month starts with 0

booleab x = st.hasMoreTokens();

**4.** To convert a date into string, use toString(). This method returns a string.

cl.toString();

Gregorian calendar is another type class like as calendar class.

**Ex: -**
```
//System date and time
import java.util.*;
class Cal
{
   public static void main(String args[ ])
   {
    //Create an obj to calendar class
    Calendar cl = Calendar.getInstance();
    //retrieve date details
    int dd = cl.get(Calendar.DATE);
    int mm = cl.get(Calendar.MONTH);
    int yy = cl.get(Calendar.YEAR);
    ++mm;
    System.out.println("System date: ");
    System.out.println("dd+ "/" +mm+ "/" +yy);
    //retrieve time details
    int h = cl.get(Calendar.HOUR);
            int m = cl.get(Calendar.MINUTE);
            int s = cl.get(Calendar.SECOND);
           System.out.println("System time: ");
   System.out.println(h+ ":" +m+ ":" +s));
   }
}
```

**Date Class: -** Date class is class useful to handle date and time.

**1.** To create an object to Date class.

```
Date d = new Date();
```

**2.** Format the date and times using getDateInstance() or getDateTimeInstance() methods of DateFormat class. This is in java.txt package.

**Syntax: -** DateFormat fmt = DateFormat.getDateInstance(formatconst, region);

Θ Here region is the place/the country.

**Ex:-**DateFormat fmt=DateFormat.getDateInstance(DateFormat.Medium, Locale.UK);

**Syntax: -** DateFormat fmt = DateFormat.getDateInstance(formatconst, formatconst region);

**Ex: -** DateFormat fmt = DateFormat.getDateInstance(DateFormat.Medium,

```
                    DateFormat.SHORT, Locale.US);
```

**Note: -**

| formatconst | Example (region = LocaleUK) |
|---|---|
| DateFormat.LONG | 03 September 2004 19:43:14 GMT + 5.30 |
| DateFormat.FULL | 03 September 2004 19:43:14 °clock GMT + 5.30 |
| DateFormat.MEDIUM | 03-Sep-04 19:43:14 |
| DateFormat.LONG | 03/09/04 7.43 pm |

**3.** Applying format to Date object is done by format () method.

```
String str = fmtt.format(d);
```

**Ex: -**

```
//To display system date and time
import java.util.*;
import java.text.*;
class MyDate
{
  public static void main(String args[ ])
  {
    //Create an obj to Date class
    Date d = new Date();
    //Store the format in DateFormat obj
DateFormat fmt = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
                    DateFormat.SHORT, Locale.UK);
    //Applying the format to d
    String s = fmt.format(d);
    //Display the formatted date and time
    System.out.println("System date and time: "+s);
    }
}
```

Θ Here d represents the system date & time already consists after creating an object.

**H.W.**

**1.** Create an Employee class with an employee's id, name, and address store objects of this class in an ArrayList. When an employee's ID is given, display his name and address.

**2.** Create a vector with a group of strings. Sort them and display them in a ascending order. Display them in reverse order also?

**3.** Create HashTable with some students hall tickets no.'s & their results. Type a hall ticket number and display his result?

**4.** Cut the string into pairs of pieces: "India=>Delhi, America=>Washington, Britain=>London, France=>Paris"

Now display the token as given below.

| City | Capital |
|---|---|
| Delhi | India |
| Washington | America |
| London | Britain |
| Paris | France |

❖ **What is the difference between Observable and Observer? *****

**Ans: -** Observable is a class is used to create subclasses that other parts of our program can observe. When an object of such a subclass undergoes a change, Observing classes are notified. Observing classes must implement the Observer interface. It will have more methods than Observer interface.

An Observer is an interface, which is useful to observe an observable object, and must implement the observer interface. This interface will defines only one method i.e. void update(Observable observOb, Object arg). Here observOb is the object being observed, and arg is the value passed by notifyObservers(). The update() method is called when a change in the observed object take place.

**STREAMS: -** A stream represents flow of data from one place to another place. They are two types of streams,

**1. Input Streams: -** It receives or reads the data to output stream.

**2. Output Stream: -** It sends or writes the data to some other place.

❖ Streams are represented by classes in java.io package.

Streams (java.io): - A stream is a sequence of bytes, or characters traveling from source to a destination.

When the bytes passing then it is called as byte stream and when the characters are passing then it is called as character stream.

To handle data in the form of 'bytes', the abstract class: InputStream and OutputStream are used.

```
                    InputStream
                        |
        ------------------------------------------------------------
        |                       |                       |
FileInputStream         FilterInputStream       ObjectInputStream
                                |
                    ----------------------------------------
                    |                                      |
            BufferedInputStream                    DataInputStream

                    OutputStream
                        |
        ------------------------------------------------------------
        |                       |                       |
FileOutputStream        FilterOutputStream      ObjectOutputStream
                                |
                    ----------------------------------------
                    |                                      |
            BufferedOutputStream                   DataOutputStream
```

**a) FileInputStream/FileOutputStream: -** They handle data to be read or written to disk files.

**b) FilterInputStream/FilterOutputStream: -** They read data from one stream and write it another stream.

**c) ObjectInputStream/ObjectOutputStream: -** They handle storage of objects and primitive data.

- Storing objects in a file called serialization.
- Retrieving the data from files is called de-serialization.

```
                    Reader
                        |
    ------------------------------------------------------------------------------------
    |                       |                       |                       |
BufferedReader          CharArrayReader         IntputStreamReader      PrintReader
                                                        |
                                                    FileReader

                    Writer
                        |
```

```
        ----------------------------------------------------------------------------------------
        |                          |                          |                          |
BufferedWriter          CharArrayWriter          IntputStreamWriter          PrintWriter
                                                          |
                                                      FileWriter
```

❖ ByteStream stores the data in the form of bytes, CharacterStream stores the data in the form of characters.

❖ Buffered means a block of memory.

**a) BufferedReader/BufferedWriter: -** Handles character (text) by buffering them. They provide efficiency.

**b) CharArrayReader/CharArrayWriter: -** Handles array of characters.

**c) InputStreamReader/OutputStreamWriter: -** They are bridge between byte streams and character streams. Reader reads bytes and then decode them into 16-bit Unicode character, write decodes character into bytes and then write.

**d) PrinterReader/PrinterWriter: -** Handle printing of characters on the screen.

❖ A file is an organized collection of data.

**How can you improve Java I/O performance:**

Java applications that utilise Input/Output are excellent candidates for performance tuning. Profiling of Java applications that handle significant volumes of data will show significant time spent in I/O operations. This means substantial gains can be had from I/O performance tuning. Therefore, I/O efficiency should be a high priority for developers looking to optimally increase performance. The basic rules for speeding up I/O performance are:

- Minimise accessing the hard disk.
- Minimise accessing the underlying operating system.
- Minimise processing bytes and characters individually.

Let us look at some of the techniques to improve I/O performance.

- Use **buffering** to minimise disk access and underlying operating system. As shown below, with buffering

large chunks of a file are read from a disk and then accessed a byte or character at a time.

**Without buffering :** inefficient code

```
try{
File f = new File("myFile.txt");
FileInputStream fis = new FileInputStream(f);
```

```
int count = 0;
int b = ;
while((b = fis.read()) != -1){
if(b== '\n') {
count++;
}
}
// fis should be closed in a finally block.
fis.close() ;
}
catch(IOException io){}
```

**Note:** fis.read() is a native method call to the underlying system.

**With Buffering: yields better performance**

```
try{
File f = new File("myFile.txt");
FileInputStream fis = new FileInputStream(f);
BufferedInputStream bis = new BufferedInputStream(fis);
int count = 0;
int b = ;
while((b = bis.read()) != -1){
if(b== '\n') {
count++;
}
}
//bis should be closed in a finally block.
bis.close() ;
}
catch(IOException io){}
```

**Note:** bis.read() takes the next byte from the input buffer and only rarely access the underlying operating system.

Instead of reading a character or a byte at a time, the above code with buffering can be improved further by reading one line at a time as shown below:

```
FileReader fr = new FileReader(f);
BufferedReader br = new BufferedReader(fr);
```

```
While (br.readLine() != null) count++;
```

By default the System.out is line buffered, which means that the output buffer is flushed when a new line character is encountered. This is required for any interactivity between an input prompt and display of output.

The line buffering can be disabled for faster I/O operation as follows:

```
FileOutputStream fos = new FileOutputStream(file);
BufferedOutputStream bos = new BufferedOutputStream(fos, 1024);
PrintStream ps = new PrintStream(bos,false);
System.setOut(ps);
while (someConditionIsTrue)
System.out.println("blah…blah…");
}
```

**Uses of files: -**

**1.** We can store the data permanently into the hard disk. (When we are strong the data in HashTable, vector etc… the data will store temporarily on the RAM).

**2.** Once we stored the data in the form of file we can share that data in different programs.

The above two are main advantages of file.

System.in –it is a InputStream object

Here System is a class java.io. package.

in is a field.

System.in – InputStream obj – Keyboard

System.out – PrintSream obj – Monitor

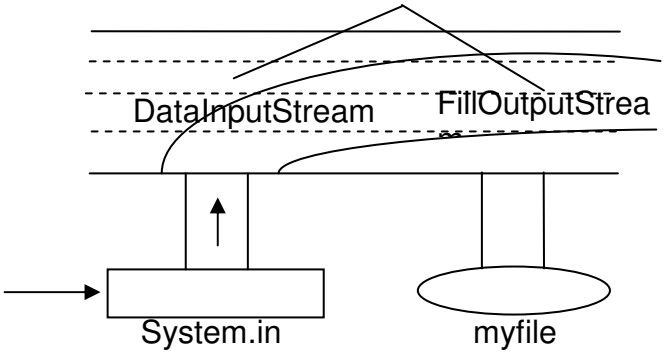
System.err – PrintSream obj – Monitor

System.out will displays normal messages on the monitor, System.err will displays error messages on the monitor.

- **What is use of Stream? *****

**Ans: -** Stream will handle input/output devices. The achieving the hardware independent of java programs we are using stream.

 rambabumandalapu@yahoo.com




**Ex: -** //Creating a file

```
import java.io.*;
class Create1
{
  public static void main(String args[ ])
  throws IOException
  {
    //Attach the keyboard to DataInputStream
    DataInputStream dis = new DataInputStream(System.in);
    //Connect file to FileOutputStream
    FileOutputStream fout = new FileOutputStream("myfile");
//reading data from DataInputStream and write that data into FileOutputStream
    char ch;
    System.out.println("Enter data (@at end): ");
    while((ch = char) dis.read()) != '@')
    fout.write(ch);
   //close the file
   fout.close();
  }
}
```

After executing & running the program, we can also open file using command "type" (i.e. a Ms-Dos Command). Every time executing & running the program old data will be remove/overwrite and new data will stored. To overcome this problem or to appending the data we have to use 'true' in the following statement.

```
FileOutputStream fout = new FileOutputStream("myfile", true);
```

**Ex: -** //Creating a file

```
import java.io.*;
class Create1
{
  public static void main(String args[ ])
  throws IOException
  {
    //Attach the keyboard to DataInputStream
    DataInputStream dis = new DataInputStream(System.in);
    //Connect file to FileOutputStream
    FileOutputStream fout = new FileOutputStream("myfile", true);
//reading data from DataInputStream and write that data into FileOutputStream
    char ch;
    System.out.println("Enter data (@at end): ");
    while((ch = char) dis.read()) != '@')
    fout.write(ch);
   //close the file
   fout.close();
  }
}
```

To improve the efficiency or the speed of execution of program we to use Buffered class.

```
BufferedOutputStream bos = new BufferedOutputStream(fout, 1024);
```

- Default size used by any Buffered class is 512 bytes.

In the place of fout.write(); we have to use bos.write(); and in the place of fout.close(); we have to use bos.close();

**Ex: -**

```
//Creating a file
import java.io.*;
class Create1
{
  public static void main(String args[ ])
  throws IOException
  {
    //Attach the keyboard to DataInputStream
    DataInputStream dis = new DataInputStream(System.in);
```

```
        //Connect file to FileOutputStream
        FileOutputStream fout = new FileOutputStream("myfile", true);
        BufferedOutputStream bos = new BufferedOutputStream(fout, 1024);
//reading data from DataInputStream and write that data into FileOutputStream
        char ch;
        System.out.println("Enter data (@at end): ");
        while((ch = char) dis.read()) != '@')
        bos.write(ch);
        //close the file
        bos.close();
      }
    }
```

**Ex: -**

```
//Reading data from a text file
import java.io.*;
class Read1
{
  public static void main(String args[ ])
  throws IOException
  {
    //Attach the file to FileInputStream
    FileInputStream fin = new FileInputStream("myfile");
    //now read from FileInputStream and display
    int ch;
    while((ch = fin.read()) != -1)
    System.out.println((char)ch);
    //close the file
    fin.close();
  }
}
```

**Ex: -**

```
//Reading data from a text file
import java.io.*;
class Read1
{
  public static void main(String args[ ])
```

```
    throws IOException
    {
      BufferedInputStream bin = new BufferedInputStream(fin);
      //Attach the file to FileInputStream
      FileInputStream fin = new FileInputStream("myfile");
      //now read from FileInputStream and display
      int ch;
      while((ch = bin.read()) != -1)
      System.out.println((char)ch);
      //close the file
      bin.close();
    }
  }
```

**Ex: -**

```
//Reading data from a text file
import java.io.*;
class Read1
{
  public static void main(String args[ ])
  throws IOException
  {
    try{
        //to enter filename from keyboard
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.ot.print("Enter file name: ");
        String fname = readLine();
        //Attach the file to FileInputStream
        FileInputStream fin = new FileInputStream(fname);
        //now read from FileInputStream and display
        int ch;
        while((ch = bin.read()) != -1)
        System.out.println((char)ch);
        //close the file
        bin.close();
        }
```

 rambabumandalapu@yahoo.com

```
        catch(FileNotFoundException fe)
        {
          System.out.println("File not found");
        }
      }
```

**Ex: -**

```
//Creating a file
import java.io.*;
class Create2
{
  public static void main(String args[ ])
  throws IOException
  {
    //to write data into file
    FileWriter fw = new FileWriter("myfile1.txt");
    //take string
    String str = "This is an institute" + "\nIam a student here";
    //read char by char from str and write into fw
    for(int i = 0; i<str.length(); i++)
    fw.write(str.charAt(i));
    //close the file
    fw.close();
  }
}
```

**Ex: -**

```
//Creating a file
import java.io.*;
class Create3
{
  public static void main(String args[ ])
  throws IOException
  {
    //to write data into file
    FileWriter fw = new FileWriter("myfile1.txt");
    BufferedWriter bw = new BufferedWriter(fw, 1024);
    //take string
```

```
    String str = "This is an institute" + "\nIam a student here";
    //read char by char from str and write into fw
    for(int i = 0; i<str.length(); i++)
    bw.write(str.charAt(i));
    //close the file
    bw.close();
  }
}
```

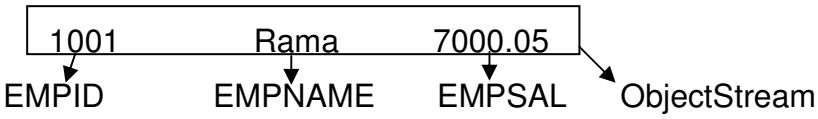
**Ex: -**

```
//Reading data from the file
import java.io.*;
class Read2
{
  public static void main(String args[ ])
  throws IOException
  {
    //attach file to FileReader
    FileReader fr = new FileReader("myfile1.txt");
    //now read data from fr and dispaly
    int ch;
    while((ch = fr.read()) != -1)
    System.ot.println((char)ch);
    //close the file
    fr.close();
  }
}
```

**Ex: -**

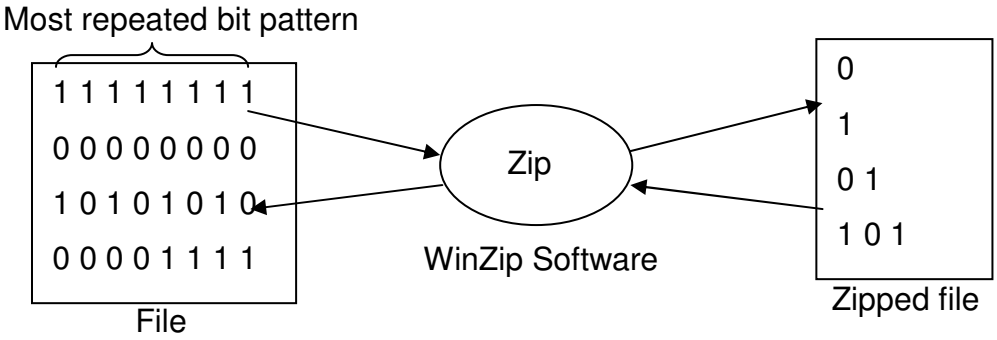
```
//Creating a file
import java.io.*;
class Read3
{
  public static void main(String args[ ])
  throws IOException
  {
    //attach file to FileReader
    FileReader fr = new FileReader("myfile1.txt");
```

```
    BufferedReader br = new BufferedReader(fr, 512);
    //now read data from fr and dispaly
    int ch;
    while((ch = br.read()) != -1)
    System.ot.println((char)ch);
    //close the file
    br.close();
  }
}
```




❖ JDBC is more convenience to handle or storing and retrieve the data from the table form.

**Zipping and Unzipping the file: -**




**1.** File converts to compressed.

**2.** File format is changed.

In java language to zip and unzip a file we will use below two classes.

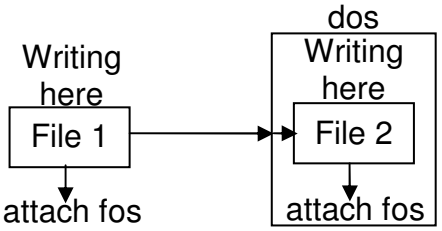
**1.** DeflaterOutputStream: - It is used to zip the file.

**2.** InflaterOutputStream: - It is used to unzip the file.

These above two classes are in java.util.zip packagae.

**Ex: -**

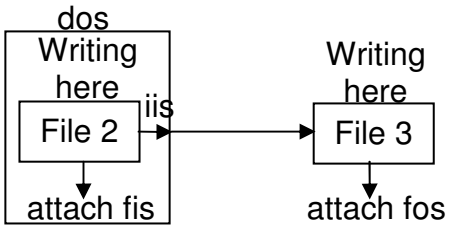
```
//Zipping
import java.util.*;
import java.util.zip.*;
class Compress
{
  public static void main(String args)
  throws Exception
```



```
  {
    //attach file1 to FileInputStream
    FileInputStream fis = new FileInputStream("file1");
    //attach file2 to FileOutputStream
    FileOutputStream fos = new FileOutputStream("file2");
    //attach fos to DeflaterOutputStream
    DeflaterOutputStream dos = new DeflaterOutputStream(fos);
    //read data from fis and write into dos
    int data;
    while((data = fis.read()) != -1)
    dos.write(data);
    //close the files
    fis.close();
    dos.close();
  }
}
```

Here we have to create file1 using Ms-Dos Edit command save it and then execute the java file.

**Ex: -**




```
//UnZip
import java.util.*;
import java.util.zip.*;
class UnCompress
{
  public static void main(String args)
  throws Exception
  {
    //attach file2 to FileInputStream
    FileInputStream fis = new FileInputStream("file2");
    //attach file3 to FileOutputStream
    FileOutputStream fos = new FileOutputStream("file3");
    //attach fis to InflaterInputStream (iis)
    InflaterInputStream iis = new InflaterInputStream (fis);
    //read data from iis and write into fos
    int data;
```

 rambabumandalapu@yahoo.com

```
        while((data = iis.read()) != -1)
         fos.write();
        //close the files
        iis.close();
        fos.close();
      }
    }
```

**H. W.**

**5.** Copy a file content into another file (new file)

**6.** Append the contents of one file to another file existing file.

**7.** Count the no. of characters, words, and lines in a text file.

**<u>Network</u>: -**

- Inter connections of computers is called Network.
- A client is a machine that sends a request for some service.
- Server is a machine that provides service to the clients.
- Internet is a network of all the computers existing on the earth.
- ISP – Internet Service Provider.
- The software which is available on the server is called Web Server.

  Ex: - Apache, Tomcat, Web Logic, Web Sphere, etc…
- Protocol represented by a set of rules, to be followed by every computer in network.

  Ex: - TCP/IP – Transmission Control Protocol/Internet Protocol
- HTTP is most widely used protocol on internet.
- FTP (File Transfer Protocol) is used when we are downloading files.
- SMTP (Short Mail Transfer Protocol) is used for sending mails.
- POP (Post Office Protocol) is used to receiving mails.
- Internet protocol address is a Unique ID number is given to every computer in network.

**Ex: -** 192.45.50.01 ----- It is a IP address of server.

192.45.50.01 ---------→ is a equal to like www.yahoo.com

- DNS – Domain Naming System, It will convert the name into the IP address.
- The current version of IP address is 6, which uses 16 bytes.

| | | | |
|---|---|---|---|
| Class A | 0 | Network 7bits | Local Address 24bits |
| | | | 16,777,216 Hosts |


rambabumandalapu@yahoo.com

Class B 10 Network 14bits Local Address 16bits

65,536 Hosts

| Class C | 110 | Network 21bits | Local Address 8bits |
|---|---|---|---|

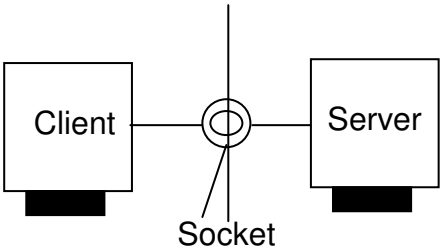
256 Hosts

Class D
Class E } Research/Experimental works

- Classes are of IP address.
- Host means Server.

**<u>Socket</u>: -** A socket is a point of connection to connect a client and server.

Every socket has an ID No. that is Port No. An ID number is allotted to a socket is called Port number.




- **When you will change socket number? *******

Ans: - 1. When we use new socket we use a new port number.

2. On the same socket if the service changes.

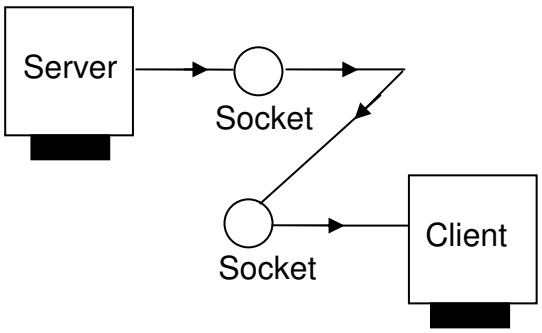
**Some allotted port numbers: -**

| S. No. | Port Number | Application |
|---|---|---|
| 1. | 13 | Date and time services. |
| 2. | 21 | FTP, which transfer files. |
| 3. | 23 | Telnet, which provides remote login. |
| 4. | 25 | SMTP, which delivers mail messages. |
| 5. | 67 | BOOTP, which provides configuration at boot time. |
| 6. | 80 | HTTP, which transfer webpage |
| 7. | 109 | POP, this enables users to access mail boxes on remote system. |

- java.net package contains classes to create a socket for server and client.
- Socket is a class to create a Socket client side.
- ServerSocket is a class to create a Socket server side.
- These two classes are in java.net package.

**Ex: -**

```
//Create a server that sends messages
import java.io.*;
import java.net.*;
class Server
{
  public static void main(String args)
  throws Exception
  {
    //Create Server Socket
    ServerSocket ss= new ServerSocket(777);
    //make that Socket accepts Client connection
    Socket s = ss.accept();
    System.out.println("Connection Established");
    //attach OutputStream to Socket
    OutputStream obj = s.getOutputStream();
    //to send data to the Socket
    PrintStream ps = new PrintStream(obj);
    //now send data
    String str = "Hello";
    ps.println(str);
    ps.println("Bye");
    //Disconnect the Server
    s.close();
    ps.close();
  }
}
```




Now compile this program and don't run it, then we have to create socket for client side.

**[** To display IP address we have to use command igcong

Syntax: -C:\rnr>ipconfig↵ **]**

**Ex: -**

```
//A Client that receives data
import java.io.*;
import java.net.*;
```

```
class Client1
{
  public static void main(String args)
  throws Exception
  {
    //Create Client Socket
    Socket s= new Socket(ipaddress/"loacalhost", 777);
    //attach InputStream to Socket
    InputStream obj = s.getInputStream();
    //To receive the data to this Socket
    BufferedReader br = new BufferedReader(new InputStreamReader(obj));
    //Accept data coming from Server
    String str;
    while(str=br.readLine()) !=null)
    System.out.println(str);
    //Disconnect the Server
    s.close();
    br.close();
  }
}
```

- We can run as many JVM's at a time.

**Communicating from the Server: -**

**1.** Create a ServerSocket

ServerSocket ss = new ServerSocket(Port No.);

**2.** Accept any Client

Socket s = ss.accept(); ------→ It returns Socket object.

**3.** To send data, connect the OutputStream to the Socket

OutputStream obj = s.getOutputStream();

**4.** To receive data from the Client, connect InputStream to the Socket

InputStream obj = s.getInputStream();

**5.** Send data to the Client using PrintStream

**Ex: -** PrintStream ps = new PrintStream(obj);

ps.print(str);

ps.println(str);

**6.** Read data coming from the Client using BufferedReader.

```
BufferedReader br = new BufferedReader(new InputStreamReader(obj);
ch = br.read();
str = br.readLine();
```

**Communicating from Client: -**

**1.** Create a Client Socket

```
Socket s= new Socket(ipaddress/"loacalhost", 777);
```

**2.** To send data connect the OutputStream to the Socket

```
OutputStream obj = s.getOutputStream();
```

**3.** To receive data fro the Server, connect InputStream to the Socket

```
InputStream obj = s.getInputStream();
```

**4.** Send data to the Server using DataOutputStream.

**Ex: -**

```
DataOutputStream dos = new DataOutputStream(obj);
Dos.writebyte(str);
```

**5.** Read data coming from the Server using BufferedReader

```
BufferedReader br = new BufferedReader(new InputStreamReader(obj);
ch = br.read();
str = br.readLine();
```

**3. Closing Communication: -**

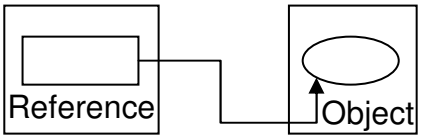
Close all Streams and Sockets

```
ps.close();
br.close();
dos.close();
ss.close();
s.close();
```

- **RMI (Remote Method Invocation)** It is a technology to call and use remote methods of remote objects.

**Ex: -**

```
//Chat Server
import java.io.*;
import java.net.*;
class Server2
{
  public static void main(String args)
```




```
        throws Exception
        {
          //Create Server Socket
          ServerSocket ss= new ServerSocket(888);
          //make this Socket wait for Client connection
          Socket s = ss.accept();
          System.out.println("Connection Established");
          //to send data to the Client
          PrintStream ps = new PrintStream(s.getOutputStream());
          //to receive data from Client
BufferedReader br = new BufferedReader(newInputStreamReader (s.getInputStream()));
           //to read data from Keyboard
BufferedReader kb=new BufferedReader(new InputStreamReader(System.in()));
          //now communicate
          while(true)                    // Server runs continuously
          {
            String str, str1;
            while(str = br.readLine() != null)
            {
              System.out.println(str);
              str1 = kb.readLine();
              ps.println(str1);
            }
          //Disconnect the Server
          ss.close();
          s.close();
          ps.close();
          br.close();
          kb.close();
          System.exit(0);
         }
        }
       }
```

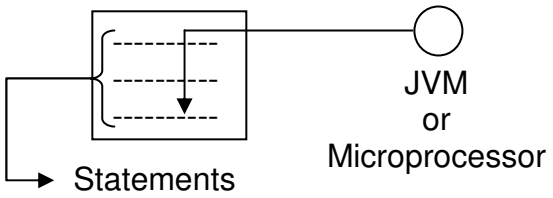
**Ex: -** //Chat Client

```
import java.io.*;
import java.net.*;
class Client2
{
  public static void main(String args)
  throws Exception
  {
    //Create Client Socket
    Socket s= new Socket("Localhost", 888);
    //to send data to the Server
    DataOutputStream dos = new DataOutputStream(s.getOutputStream());
    //to receive data from Server
BufferedReader br=new BufferedReader(new InputStream(s.getInputStream()));
    //to read data from Keyboard
BufferedReader kb=new BufferedReader(new InputStreamReader(System.in()));
    //now start communicate
    String str, str1;
    while(str = br.readLine() .equals("exit"))
    //As long as the String are not typing exit.
    {
       dos.writeBytes(str+"\n");
       str1 = br.readLine();
       System.out.println(str1);
     }
    //Disconnect the Server
    s.close();
    dos.close();
    br.close();
    kb.close();
    }
  }
```

**H. W.**

**8.** Develop a server that sends to system data and time to the client display that data and time at client side (Hint: Use Port No.13)

**9.** Write client server programs so that the client sends the name of a file to the server and the server sends the file content to client display the file contents at client side (the file should present in the server).




**Thread: -** A thread represents a process of execution (or) executing the set of statements is called a thread.

- Every java program is executed by using one thread i.e. is called main thread.
- JVM will execute statements step by step.

**Ex: -**

```
//Every java program has a thread
class Current
{
  public static void main(String args)
  {
    System.out.println("This is first line");
    Thread t = Thread.currentThread();
    System.out.println("Present thread = "+t);
    System.out.println("Its name = "+t.getName());
  }
}
```

C:\rnr>javac Current.java↵

C:\rnr>java Current ↵

**O/P: -** This is first line

Present Thread = Thread[main, 5, main]

Here main is a thread name.

5 is the Priority Number.

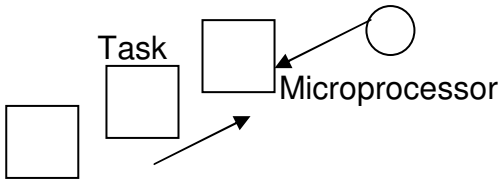
[main, 5, main] is called as Thread group.

From the above thread name is main. Thread has priority number. Priority number will varies from 1 to 10.

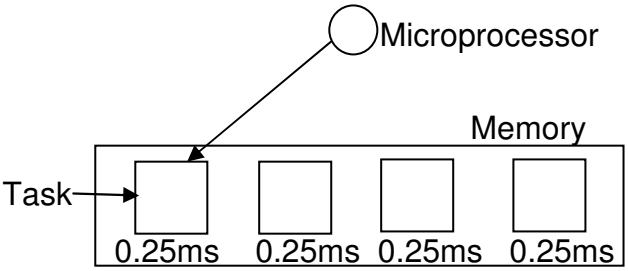
| | | |
|---|---|---|
| Priority No. 1 | -------→ | Minimum Priority |
| Priority No. 5 | -------→ | Normal Priority |
| Priority No. 10 | -------→ | Maximum Priority |

- Main thread priority number is 5.
- We can provide more threads to the microprocessor. Microprocessor will execute the 10billions of machine code instructions per second.
- Executing the tasks (one or more statements is two types.

**1. <u>Single tasking</u>: -** Executing one task at a time is called single tasking. In a Single tasking lot of processor time wasted.




**<u>Time Slice</u>: -** Time slice is the time allotted by the Microprocessor to execute each task.




Microprocessor will allot equal timings (0.25milli seconds) for each task.

**<u>Round Robin Method</u>: -** Microprocessor uses round robin method to execute several tasks simultaneously.

Executing first task after last task is called round robin method.

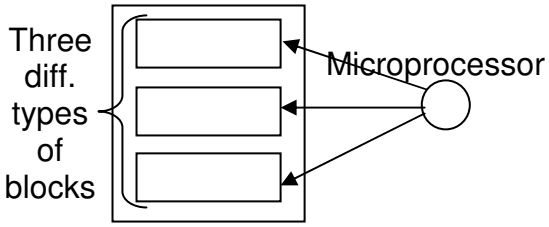
**Ex: -** Robin is a bird which will comes down by making rounds and it will jump up by making rounds that's why they have compared the microprocessor like that above.

**<u>2. Multitasking</u>: -** Executing several tasks simultaneously is called multitasking.

**a) <u>Process based multitasking</u>: -** Executing several programs at a time is called process based multitasking.

Time slice, Round robin methods are process based multitasking.

**b) <u>Thread based multitasking</u>: -** Executing several parts of a program simultaneously is called thread based multitasking.




Using more than one thread is called multithreading to perform multiple tasks simultaneously at a time.

**<u>Create a thread</u>: -**

**1.** Write a class that extends thread class or implements runnable interface.

Thread class, runnable implements will helpful to create a thread. These two are available in java.lang package.

**2.** Write public void run in the class thread will execute only this method.

public void run()

Thread can't act upon any method, but default it works only on run() method.

**3.** Create an object to the class.

**4.** Attach a thread to this object.

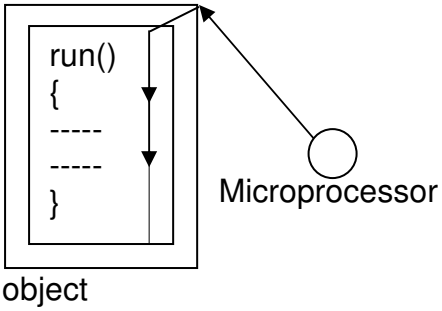
**5.** Start thread, then will act upon that object.

**Ex: -**

```
//Creating thread
Class MyThread extend Thread
{
  public void run()
  {
    for(int i = 1; i<100000; i++)
    {
      System.out.println(i);
    }
  }
}
class TDemo
{
  public static void main(String args[ ])
  {
    MyThread obj = new MyThread();
    Thread t1 = new Thread();
    t1.start();
  }
}
```

- stop() method is used to stop the thread in old versions. But stop() method is deprecated in new versions.
- Forcibly to terminate the program we have to use "control + c" (Ctrl+c keys in the keyboard).
- To stop the thread also we have to use control + c keys.

**Ex: -** //Creating thread and smoothly terminate/stop the program or thread




```
Import java.io.*;
Class MyThread extend Thread
{
   boolean stop = false;
   public void run()
   {
     for(int i = 1; i<100000; i++)
     {
       System.out.println(i);
       if(stop) return;
     }
   }
}
class TDemo
{
  public static void main(String args[ ])
  {
     MyThread obj = new MyThread();
     Thread t1 = new Thread();
     t1.start();
     System.in.read();
     obj.stop = true;
  }
}
```

This above program is for stopping the thread smoothly.

❖ **How can you stop the thread which is running? *******

**Ans: - 1.** Declare a boolean variable and initialize it as false

**Ex: -** boolean stop = false;

**2.** When ever the thread should be stopped store true into this variable

**Ex: -** obj.stop = true;

**3.** If the variable becomes true using return statement come out run() method

**Ex: -** if(stop) return;

 rambabumandalapu@yahoo.com

❖ **What is the difference between extends and implement Runnable? Which one is advantage? *******

**Ans: -** There is no difference between these two. Implement Runnable is advantage because interfaces is always better than classes.

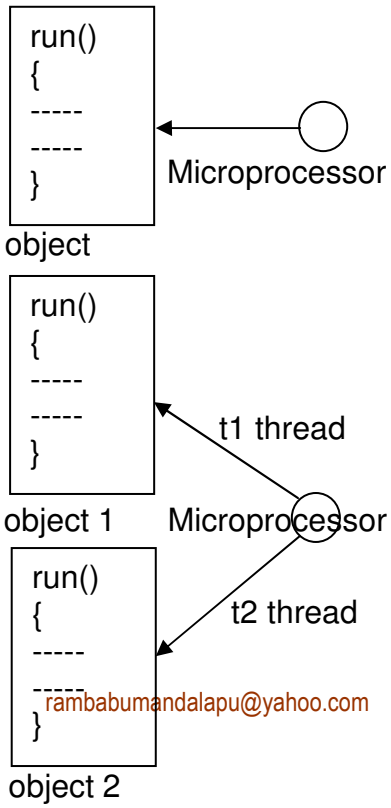
If we extend thread class, there is no scope any another class, this is the limitation of extends.

If we implement Runnable interface still there is a scope to extend other class.

❖ The main advantage of multitasking is utilizing a processor time in an optimum way.

**Ex: -**

```
//Theatre Example
class MyThread implements Runnable
{
  String str;
  MyThread(String str)
  {
    this.str = str;
  }
  public void run()
  {
    for(int i = 1; i<=10; i++)
    {
      System.out.println(str+ " : " +i);
      try{
          Thread.sleep(2000);
          }catch(InterruptedException ie)
          {
            ie.printStackTrace();
          }
      }
    }
}
class Theatre
```

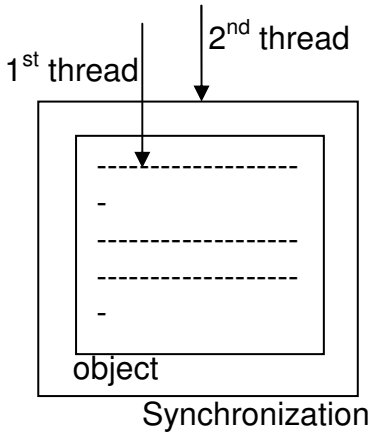


```
{
  public static void main(String args[ ])
  {
    //create objects to MyThread class
   MyThread obj1=new MyThread("Cut ticket");
   MyThread obj2=new MyThread("Show ticket");
   //Create two threads and attach them to          //these two objects
  Thread t1 = new Thread(obj1);
  Thread t2 = new Thread(obj2);
  //start the thread
  t1.start();
  t2.start();
 }
}
```

❖ When multiple threads are acting on the one or same object we will get sometimes unreliable results.

**Ex: -** //Two threads acting on one object

```
class Reserve implements Runnable
{
  int available = 1;
  int wanted;
  Reserve (int i)
  {
    wanted = i;
  }
  public void run()
  {
   System.out.println("Number of berths available = "+available);
   if(available >= wanted)
   {
     String name = Thread.currentThread.getName();
     System.out.println(wanted + "berths reserved for"+name);
     try{
          Thread.sleep(2000);
```



```
            available = available – wanted;
          }
          catch(Interrupted Exception ie){ }
        }
        else
        System.out.println("Sorry, no berths to reserve");
      }
    }
    class Syn
    {
      public static void main(String args[ ])
      {
        //create an obj to Reserve class with 1 berth
        Reserve obj = new Reserve(1);
        //create 2 threads and attach them to obj
        Thread  t1 = new Thread(obj);
        Thread  t2 = new Thread(obj);
        //give names to threads
        t1.setName("First Person");
        t2.setName("Second Person");
        //run the threads
        t1.start();
        t2.start();
        }
    }
```

This above program will allot tickets for both persons. For this permanent solution we have to block the 2nd thread till the completion of the 1st thread. So we have to use synchronization method to block the 2nd thread.

**Ex: -**

```
//Two threads acting on one object
class Reserve implements Runnable
{
  int available = 1;
  int wanted;
  Reserve (int i)
```

```
    {
      wanted = i;
    }
    public void run()
    synchronized(obj)
    {
     System.out.println("Number of berths available = "+available);
     if(available >= wanted)
     {
       String name = Thread.currentThread.getName();
       System.out.println(wanted + "berths reserved for"+name);
       try{
            Thread.sleep(2000);
            available = available – wanted;
          }
          catch(Interrupted Exception ie){ }
       }
       else
       System.out.println("Sorry, no berths to reserve");
      }
    }
  }
  class Syn
  {
    public static void main(String args[ ])
    {
     //create an obj to Reserve class with 1 berth
     Reserve obj = new Reserve(1);
     //create 2 threads and attach them to obj
     Thread  t1 = new Thread(obj);
     Thread  t2 = new Thread(obj);
     //give names to threads
     t1.setName("First Person");
     t2.setName("Second Person");
```

```
    //run the threads
    t1.start();
    t2.start();
    }
}
```

**<u>Thread Synchronization (or) Synchronization</u>: -** Synchronization is locking the object, so that when a thread is processing object any other thread will not be allowed to act upon the object. Synchronized object is also called as Mutex i.e. Mutually Exclusive lock.

An object can be synchronized in two ways

**1<u>. Synchronized block()</u>: -**

```
synchronized(obj)
{
  statements;
}
```

For group of statements we can use this synchronized block.

**Ex: -** class **BlockLevel** {

```
//shared among threads
SharedResource x, y ;
//dummy objects for locking
Object xLock = new Object(), yLock = new Object();
pubic void method1() {
synchronized(xLock){
//access x here. thread safe
}
//do something here but don't use
SharedResource x, y ;
synchronized(xLock) {
synchronized(yLock) {
//access x,y here. thread safe
}
}
//do something here but don't use
SharedResource x, y ;
}
```

```
}
```

**2.** By making a method as synchronized method

```
synchronized void method()
{
  statements;
}
```

For entire method to synchronized we will use this method.

**Ex: -**

```
class MethodLevel {
//shared among threads
SharedResource x, y ;
pubic void synchronized
method1() {
//multiple threads can't access
}
pubic void synchronized
method2() {
//multiple threads can't access
}
public void method3() {
//not synchronized
//multiple threads can access
}
}
```

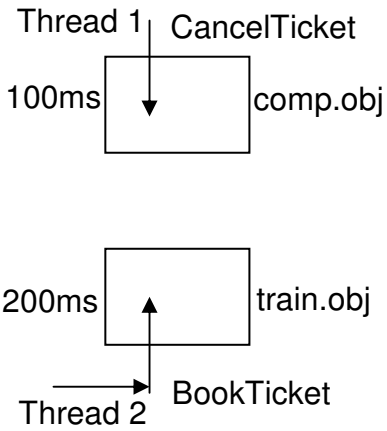
**<u>Synchronization important</u>: -** Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often causes dirty data and leads to significant errors.

**Disadvantage of synchronization** is that it can cause deadlocks when two threads are waiting on each other to do something. Also synchronized code has the overhead of acquiring lock, which can adversely the performance.

- Synchronization is also known as thread safe, unsynchronized is also known as thread unsafe.
- Locking means it will not allow another thread still the completion of one task of one thread.

**Ex: -**

```
//To cancel the ticket
Class CancelTicket extends Thread
```




```
{
  object train, comp;
  CancelTicket(object train, object comp)
 {
    this.train = train;
    this.comp = comp;
  }
    public void run()
    {
      synchronized(comp)
      {
        System.out.println("CancelTicket locked the compartment");
        try{
             sleep(100);
            }catch(InterruptedException ie){ }
        System.out.println("CancelTicket wants to lock on train");
        synchronized(train)
        {
          System.out.println("CancelTicket now locked train");
        }
      }
   }
}
class BookTicket extends Thread
{
  object train, comp;
  BookTicket(object train, object comp)
  {
    this.train = train;
    this.comp = comp;
  }
  public void run()
  {
    synchronized(train)
```

 rambabumandalapu@yahoo.com

```
        {
          System.out.println("BookTicket locked the train");
          try{
              sleep(200);
              }catch(InterruptedException ie){ }
         System.out.println("BookTicket wants to lock on compartmet");
         synchronized(comp)
         {
           System.out.println("BookTicket now locked compartment");
          }
        }
      }
    }
    class DeadLock
    {
      public static void main(String args[ ])
      throws Exception
      {
        //take train and compartment as objects
        object  train = new object();
        object compartment = new object();
        //create objects to CancelTicket, BookTicket
        CancelTicket obj1 = new CancelTicket(train, compartment);
        BookTicket obj1 = new BookTicket (train, compartment);
        //create 2 threads and attach them to these objects
        Thread t1 = new Thread(obj1);
        Thread t2 = new Thread(obj2);
        //run the threads
        t1.start();
        t2.start();
      }
    }
```

Save it as DeadLock.java compile & run the program.

**Output: -**

CancelTicket locked the Compartment

BookTicket locked the Train

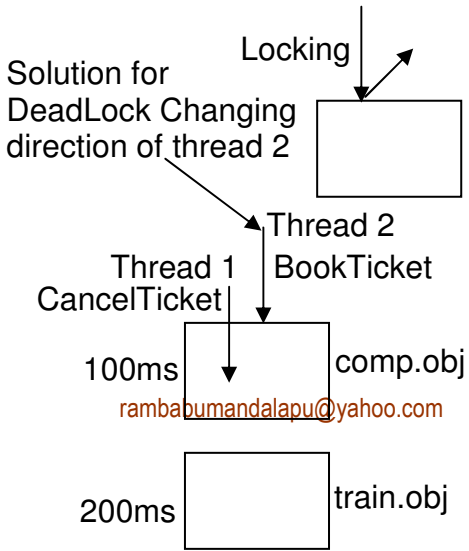
CancelTicket wants to lock on train

BookTicket wants to lock the Compartment

Press Ctrl+c for forcibly to terminate the program, because to solve this program in another way.

**Ex: -** //To cancel the ticket Solution for dead lock

```
Class CancelTicket extends Thread
{
  object train, comp;
  CancelTicket(object train, object comp);
 {
    this.train = train;
    this.comp = comp;
  }
    public void run()
    {
      synchronized(comp)
      {
        System.out.println("CancelTicket locked the compartment");
        try{
             sleep(100);
           }catch(InterruptedException ie){ }
        System.out.println("CancelTicket wants to lock on train");
        synchronized(train)
        {
          System.out.println("CancelTicket now locked train");
        }
      }
   }
}
class BookTicket extends Thread
{
```



```
    object train, comp;
    BookTicket(object train, object comp);
    {
      this.train = train;
      this.comp = comp;
    }
    public void run()
    {
      synchronized(comp)
      {
        System.out.println("BookTicket locked the Compartment");
        try{
            sleep(200);
            }catch(InterruptedException ie){ }
        System.out.println("BookTicket wants to lock on compartmet");
        synchronized(train)
        {
          System.out.println("BookTicket now locked train");
         }
       }
     }
  }
  class DeadLock
  {
    public static void main(String args[ ])
    throws Exception
    {
      take train and compartment as objects
      object  train = new object();
      object compartment = new object();
      //create objects to CancelTicket, BookTicket
      CancelTicket obj1 = new CancelTicket(train, compartment);
      BookTicket obj1 = new BookTicket (train, compartment);
      //create 2 threads and attach them to these objects
```

```
        Thread t1 = new Thread(obj1);
        Thread t2 = new Thread(obj2);
        //run the threads
        t1.start();
        t2.start();
      }
    }
```

**DeadLock of thread: -** When a thread locked an object and waiting for another object which has been already locked by another thread and the other thread is also waiting for the first object it leads to DeadLock situation.

In DeadLock both the threads mutually keep under waiting forever and further processing is canceled.

A programmer should avoid DeadLock situations in his program by properly planning designing the program.

**Thread Class Methods: -**

**1.** To know the currently running thread.

```
Thread t = Thread.currentThread();
```

**2.** To start thread

```
t.start();
```

**3.** To stop execution of a thread for a specified time

```
Thread.sleep(milliseconds);
```

**4.** To get the name of a thread

```
String name = t.getName();
```

**5.** To set a new to a thread

```
t.setName("name");
```

**6.** To get the priority of a thread

```
int priority_no = t.getPriority();
```

**7.** To set the priority of a thread

```
t.setPriority(int priority_no);
```

**Note: -** The priority number constants are as given below

Thread.MAX_PRIORITY value is 10

Thread.MIN_PRIORITY value is 0

Thread.NORM_PRIORITY value is 5

**8.** To test if a thread is still alive

t.isAlive(); returns true/false

**9.** To wait till a thread dies

t.join();

**10.** To send a notification to awaiting thread

obj.notify();

**11.** To send notification to all waiting threads

obj.notifyAll();

**12.** To wait till the obj is released (till notification is sent)

obj.wait();

- notify() method will sends a notification to a thread, i.e. some object is available to the thread.
- The above three methods (10, 11, 12) are belong to object class.

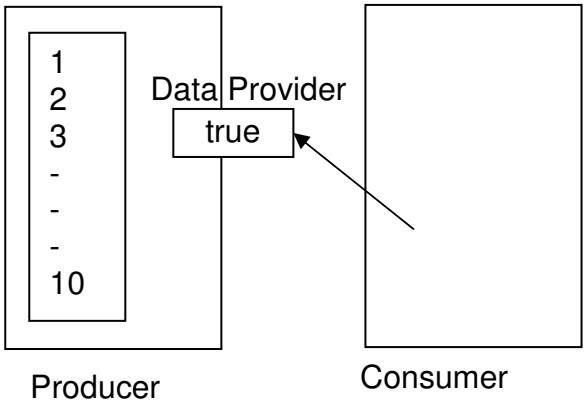
- **What is the difference between Green thread and Nativethread? *******

**Ans: -** A program thread uses two types of operating system threads, they are Green Thread model and Native Thread model. Green Thread model will provide only one thread for a program thread, where as Native Thread provides a separate thread for each program thread.

**Ex: -**

```
//Thread Communication
Class Communicate
{
  public static void main(String args[ ])
  {
    //create Producer, Consumer, objects
    Producer  obj1 = new Producer();
    Consumer obj2 = new Consumer(obj1);
   //create 2 threads and attach them obj1, obj2
   Thread t1 = new Thread(obj1);
   Thread t2 = new Thread(obj2);
   //run the threads
   t1.start();
   t2.start();
  }
}
```

```
class Producer extends Thread
{
  Boolean dataprodover = false;    //Θ dataprodove is a data production over
  SrtingBuffer sb;
  producer()
  {
    sb = new StringBuffer();
  }
  public void run()
  {
    for(int i = 1; i<=10; i++)
    {
      try
          {
            sb.append(i+":");
            sleep(100);
             System.out.println("appending");
          }
          catch(Exception e){ }
      }
    dataprodover = true;
  }
}
class Consumer extends Thread
{
  Producer prod;
  Consumer(Producer prod)
   {
      this.prod = prod;
    }
public void run()
 {
    try{
          while(!prod.dataprodover)
```



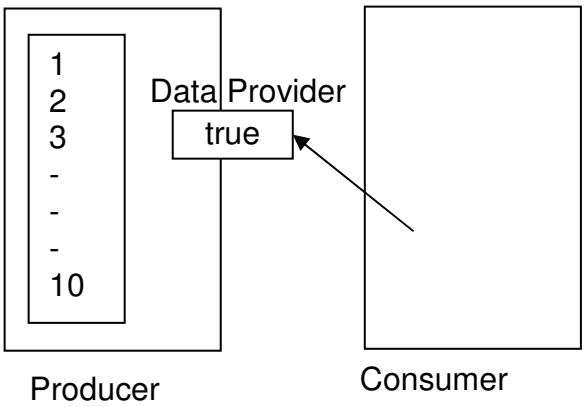
```
            {
              sleep(10);
             }
            }catch(Exception e){ }
        System.out.println(prob.sb);
      }
    }
```

Save it as communication.java. This above program/method is not efficient way to communication between threads.

If we want communicate efficient way we have to use notify() method.

**Ex: -** //Thread communication in efficient way

```
Class Communicate
{
  public static void main(String args[ ])
  {
   //create Producer, Consumer, objects
   Producer  obj1 = new Producer();
   Consumer obj2 = new Consumer(obj1);
  //create 2 threads and attach them obj1, obj2
  Thread t1 = new Thread(obj1);
  Thread t2 = new Thread(obj2);
  //run the threads
  t1.start();
  t2.start();
 }
}
class Producer extends Thread
{
  SrtingBuffer sb;
  producer()
  {
    sb = new StringBuffer();
  }
```



```
  public void run()
  {
    synchronized()
    {
      for(int i = 1; i<=10; i++)
      {
        try
          {
            sb.append(i+":");
            sleep(100);
             System.out.println("appending");
          }
         catch(Exception e){ }
      }
    sn.notify(); // or we can use sb.notifyAll();
   }
 }
}
class Consumer extends Thread
{
  Producer prod;
  Consumer(Producer prod)
   {
     this.prod = prod;
   }
public void run()
 {
   synchronized(prod.sb)
   {
     try{
          prod.wait();
         }catch(Exception e){ }
     System.out.println(prod.sb);
   }
```

 rambabumandalapu@yahoo.com

```
    }
  }
```

❖ **What is the difference between sleep() method and wait() method? *****

**Ans: -** Both methods will make the thread wait for some time. When the thread comes out of sleep() method the object may be still lock. When the threads comes out of the wait() method the object is automatically unlocked.

But both methods will wait temporarily.

Ex: -

```
synchronized(obj)                    synchronized(obj)
{                                    {
  ----------------------}locked       ----------------------}locked
  ----------------------}              ----------------------}
  sleep(2000);                         wait(2000);
  ----------------------}locked       ----------------------}Unlocked
  ----------------------}              ----------------------}
}                                    }
```

**What are different types of threads: -**

**1.** User thread or Main thread.

**2.** Daemon thread

**3.** GUI thread

❖ **What is Daemon thread? *****

**Ans: -** A Daemon thread is a thread that continuously provides services to other threads i.e. Daemon thread are used for background services.

Ex: - To start mysql

F:\rnr>mysqld↵ Θ Here d is the Daemon thread.

F:\rnr>

Now onwards Daemon thread makes the mysql database running continuously.

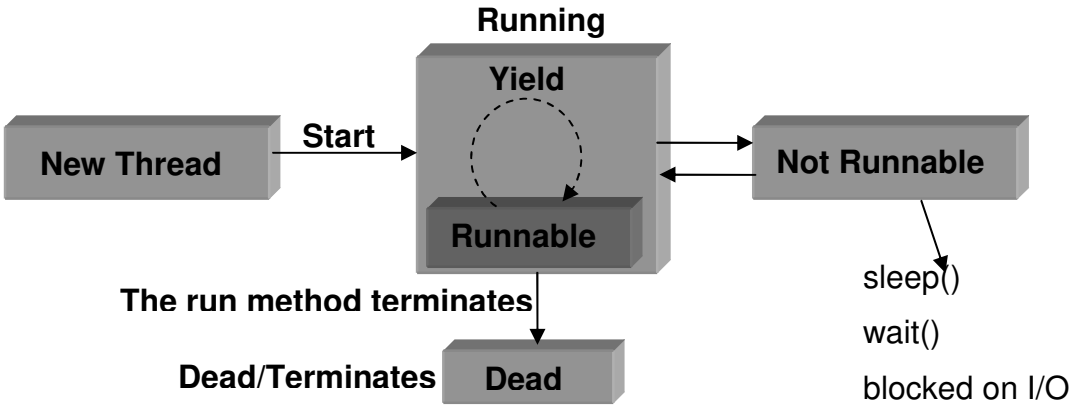
Daemon threads are generally used background services.

❖ **What is thread life cycle? *****

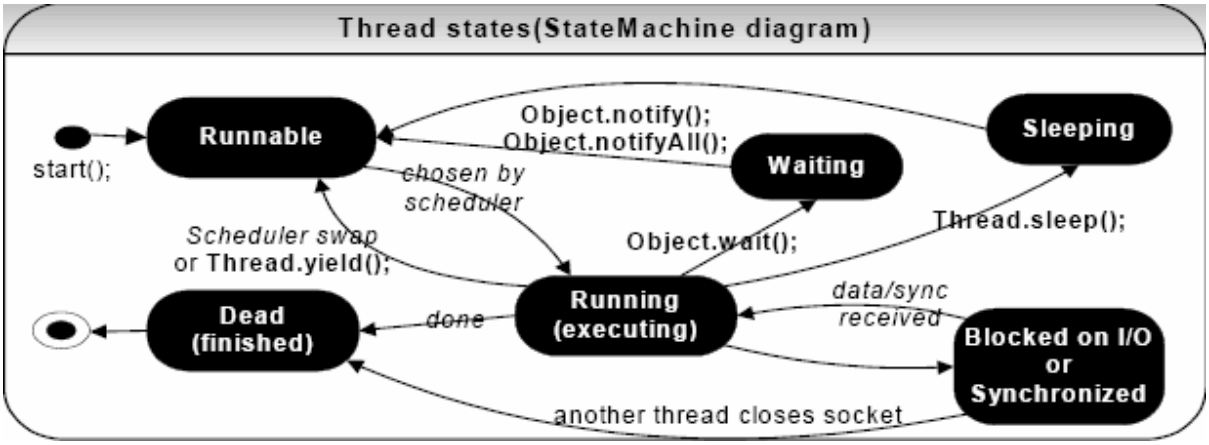
**Ans: -** Life cycle of thread means from the creation of thread till its termination. The states assumed by the thread are called life cycle of a thread.

Start → run → wait (or) blocked state → Destroy State

**Life Cycle of Thread**




**(or)**




Runnable means it will executes public void run methods yield makes pausing the thread.

Not Runnable means thread will stop runnable.

Afetr executing the all methods by run() method then the thread will be terminated.

**Runnable: -** Waiting for its turn to be picked for execution by the thread schedular based on thread priorities.

**Running:** - The processor is actively executing the thread code. It runs until it becomes blocked, or voluntarily gives up its turn with this static method *Thread.yield().* Because of context switching overhead, *yield()* should not be used very frequently.

**Waiting:** - A thread is in a **blocked state** while it waits for some external processing such as file I/O to finish.

**Sleeping:** - Java threads are forcibly put to sleep (suspended) with this overloaded method: Thread.sleep(milliseconds), Thread.sleep(milliseconds, nanoseconds);

**Blocked on I/O**: Will move to runnable after I/O condition like reading bytes of data etc changes.

**Blocked on synchronization**: Will move to Runnable when a **lock is acquired.**

**Dead**: The thread is finished working.

❖ **What is the difference between yield and sleeping? *******

**Ans: -** When a task invokes yield(), it changes from running state to runnable state. When a task invokes sleep(), it changes from running state to waiting/sleeping state.

❖ **What are the states of thread? *******

**Ans: -** New thread state → Runnable → Running → wait (or) blocked state → Destroy State

**Thread Group: -** A group of threads as single unit is called ThreadGroup.

If we applied certain methods on the ThreadGroup it will effect all other threads. Controlling all the threadsx by giving a single command or method is possible, because of ThreadGroup.

ThreadGroup tg = new ThreadGroup("group name");

Thread t1 = new Thread(tg, obj, "threadname");

Θ Here obj is target object

Thread t2 = new Thread(tg, obj1, "threadname");

We can also add one ThreadGroup to another ThreadGroup.

ThreadGroup tg1 = new ThreadGroup(tg, "group name");

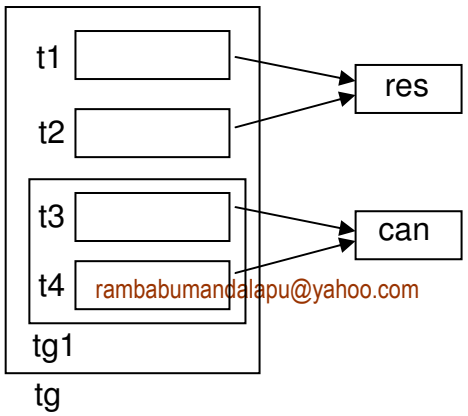
**Daemon Thread: -** Daemon threads are service providers fro other threads or objects. A daemon thread executes continuously. It provides generally a background processing.

**1.** To make a thread as Daemon thread

t.setDaemon(true);

**2.** To know if a thread is Daemon or not

Boolean x = t.isDaemon();



**Ex: -**

```
/*Using thread groups*/
class WhyTGroups
{
  public static void main(String args[ ])
  throws Exception
  {
   Reservation res = new Reservation();
   Cancellation can = new Cancellation();
   //create a ThreadGroup with name
   ThreadGroup tg = new ThreadGroup("Reservation Group");
   //create 2 threads and add them to ThreadGroup
   Thread t1 = new Thread(tg, res, "First thread");
   Thread t2 = new Thread(tg, res, "Second thread");
   //create another ThreadGroup as a child to tg
   ThreadGroup tg1 = new ThreadGroup(tg, "Cancellation Group");
   Thread t3 = new Thread(tg1, res, "Third thread");
   Thread t4 = new Thread(tg1, res, "Fourth thread");
   System.out.println('Number of threads in a group = "+tg.activeCount());
   //Find parent group of tg1
   System.out.println('Parent of tg1 = "+tg1.getParent());
   //set maximum priority to tg1 as 7
   tg1.setMaxPriority (7);
    //know the ThreadGroup of t1 and t2
    System.out.println('ThreadGroup of t1 = "+t1.getThreadGroup());
    System.out.println('ThreadGroup of t2 = "+t2.getThreadGroup());
    //start the threads
    t1.start();
    t2.start();
    t3.start();
    t4.start();
  }
}
class Reservation extends Thread
{
```

 rambabumandalapu@yahoo.com

```
  public void run()
  {
    System.out.println('I am Reservation thread");
  }
}
class Cancellation extends Thread
{
  public void run()
  {
    System.out.println('I am Cancellation thread");
  }
}
```

**H.W.**

**10.** Create a thread that display the time on the screen continuously, till enter button is pressed.

**11.** Create a parallel server using 3 or 4 threads that can serve several clients simultaneously.

User can interact with application in two ways.

**1. CUI (Character User Interface): -** In CUI the user can interact with the application by typing the characters. The user cannot remember all commands every time in CUI.

**2. GUI (Graphic User Interface): -** If the user interacts with the application through graphics (Pictures/Images/…..).

**Advantages of GUI: -**

**1.** GUI is user friendly.

**2.** GUI makes an application more attractive.

**3.** Using GUI we can simulate real objects.

Components means Push Buttons, Radio Buttons, Check boxes,…

**java.awt package: -** AWT Standards for Abstract Window Toolkit. It is a package that provides a set of classes and interfaces to create GUI components.

```
- - - label
        |
- - - Button
        |
```

```
                          - - - Checkbox
                                  |
                          - - - Choice
                                  |
Object- - -Component- - - - - - - List
   |                              |
CheckBoxGroup             - - - Canvas
                                  |
                          - - - Scrollbar            - - - TextFiled
                                  |                  |
                          - - - TextComponent - - -  |
                                  |                  |
                          - - - Container            - - - TextArea
                                    |
                              |-------------|
                              |             |
                          Panel         Window
                              |             |
                          Applet        Frame
```

- Layout manager is an interface that arranges the components on the screen.
- Layout manager is implemented in the following classes.

```
                 - - - FlowLayout
                 |         |
                 - - - BorderLayout
                 |         |
Object - - - - - - - - GridLayout
                 |         |
                 - - - CardLayout
                 |         |
                 - - - GridBagLayout
```

- A frame is a top level window that is not contained in another window.
- A window represents the some rectangular part on the screen.

- **What is the difference between window and frame? *****

**Ans: -** A window represents a rectangular part of the screen.

A frame is a window with title bar and some button to minimize, maximize or close i.e. is called frame.

❖ Pixel is a short form of a picture each dot (.) is a pixel. Pixel means picture element.

**Creating a frame: -**

**1.** Create an object to frame class.

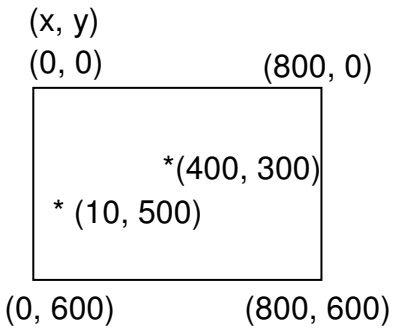
**Ex: -** Frame f = new Frame();

**2.** Write a class that extends a frame and then creates an object to that class.

**Ex: -** class MyFrame extends Frame

MyFrame f = new MyFrame();

This is advantageous than 1st method.




**Ex: -**

```
//creating a frame
import java.awt.*;
class MyFrame
{
  public static void main(String args[ ])
  {
   //create an object to Frame class
   Frame f = new Frame();
   //increase the size of frame
    f.setSize(400,300);
    //display the frame
    f.setVisible(true); (or) f.show();
   }
}
```

❖ Components are to display but not to perform any task.

**Ex: -**

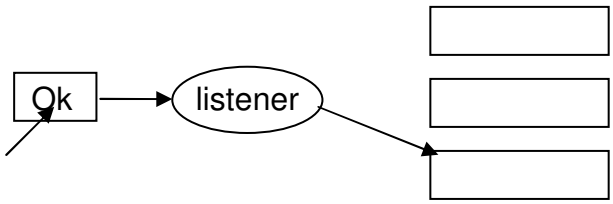
```
//creating a frame –version-2.0.
import java.awt.*;
class MyFrame extends Frame
{
  public static void main(String args[ ])
  {
```

```
        //create an object to frame class
        MyFrame f = new MyFrame();
       //increase the size of frame
       f.setSize(400,300);
       //display the title
       f.steTitle("My AWT Frame");
       //display the frame
       f.setVisible(true); (or) f.show();
      }
    }
```

**Event: -** User interaction with the component is called Event.

**Ex: -** Clicking, Double Clicking, Pressing a key, moving the mouse and dragging the item.

**Listener: -** Listener is an interface that will added to the component. It is listen to the event.




❖ **What is Event Delegation model? *******

**Ans: -** Events are generated by user to perform a task. The component will delegate the event to a listener. The listener will delegate the event to the method and finally the method will handle the event. This is called Event Delegation model.

❖ Delegation means handing over the work to another.

❖ Event delegation model separate/isolated presentation logic and business logic. A programmer can modify one part without affecting the other part. Each logic can be developed using different technologies.

To close the frame: -

1. Attach a WindowListener to the frame. A Listener is an interface.
2. Implement all the methods of WindowListener interface.
3. When an event is generated a relevant method is called and executed by the Listener.

**Ex: -**
```
//creation of frame and closing the frame
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
  public static void main(String args[ ])
  {
   //create an obj to Frame class
   MyFrame f = new MyFrame();
   //increase the size of frame
   f.setSize(400,300);
   //display the title
   f.steTitle("My AWT Frame");
   //display the frame
   f.setVisible(true); (or) f.show();
   //attach a WindowListener to the frame
   //f.addWindowListener(WindowListener obj);
   f.addWindowListener(new Myclass());
  }
}
class Myclass implements WindowListener
{
   public void windowActivated(windowEvent e){ }
   public void windowClosed(windowEvent e){ }
   public void windowClosing(windowEvent e){ }
  {
    System.exit(0);
  }
   public void windowDeactivated(windowEvent e){ }
   public void windowDeiconified(windowEvent e){ }
   public void windowIconified(windowEvent e){ }
   public void windowOpended(windowEvent e){ }
}        (or)
```

**Ex: -**
```
//creation of frame and closing the frame
```

```
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
  public static void main(String args[ ])
  {
   //create an obj to Frame class
   MyFrame f = new MyFrame();
   //increase the size of frame
   f.setSize(400,300);
   //display the title
   f.steTitle("My AWT Frame");
   //display the frame
   f.setVisible(true); (or) f.show();
   //attach a WindowListener to the frame
   f.addWindowListener(new WindowAdaptor()
  {
    public void windowClosing(windowEvent we)
    {
      System.exit(0);
    }
   });
  }
}
class Myclass implements WindowAdaptor
{
   public void windowClosing(windowEvent e){ }
  {
    System.exit(0);
  }
}
```

- **What is adaptor class? *****

**Ans: -** An adaptor class is similar to an interface, which contains empty implements of all the methods of the interface. WindowAdaptor is the adaptor class interface.

❖ **What is anonymous inner class? *******

**Ans: -** It is an inner class whose name is hidden to outer class and for only which is created.

A Frame is used to display following things

**1.** Messages

**2.** To display images/photos/icons/.gif files

**3.** To display components (Radio buttons, push buttons, check buttons etc…)

**Displaying the messages in the Frame: -** For this purpose we should use drawString() method of graphics class.

**Ex: -** g.drawString("Message", x, y);

Here x, y are coordinates of pixels.

**Ex: -**

```
//Displaying a message in the frame
import java.awt.*;
import java.awt.event*;
class Message extends Frame
{
  //vars
  //Default Constructor
  Message()
  {
    //write code to close the frame
    this.addWindowListener(new windowAdaptor()
    {
      public void windowClosing(windowEvent e)
      {
        System.exit(0);
      }
    });
  } //end of constructor
```

```
 public void paint(Graphics g)
{
  //set a background color for frame
  this.setBackground(new color(100,20,20);
  //set a text color
  set.Color(Color.green);
  //set a font for text
  Font f = new Font("Helvetica", Font.BOLD+Font.ITALIC, 35);
  g.setFont(f);
  //now display the messages
  g.drawString("Hello Students", 20, 100);
}
public static void main(String args[ ])
{
  //create the frame
  Message m = new Message();
  //set the size of frame
  m.setSize(500,400);
  //set a title to the frame
  m.steTitle("My message");
  //display the frame
  m.setVisible(true); (or) m.show();
 }
} (or)
```

**Ex: -**

```
//Displaying a message in the frame
import java.awt.*;
import java.awt.event.*;
class Message extends Frame
{
 //vars
 //Default Constructor
 Message()
 {
  //write code to close the frame
```

```
    addWindowListener(new windowAdaptor()
    {
      public void windowClosing(windowEvent e)
      {
        System.exit(0);
      }
    });
  } //end of constructor
  public void paint(Graphics g)
  {
    //set a background color for frame
    setBackground(new color(100,20,20);
    //set a text color
    set.Color(Color.green);
    //set a font for text
    Font f = new Font("Helvetica", Font.BOLD+Font.ITALIC, 35);
    g.setFont(f);
    //now display the messages
    g.drawString("Hello Students", 20, 100);
  }
  public static void main(String args[ ])
  {
    //create the frame
    Message m = new Message();
    //set the size of frame
    m.setSize(500,400);
    //set a title to the frame
    m.steTitle("My message");
    //display the frame
    m.setVisible(true); (or) m.show();
  }
}
```

**<u>To specify a color in awt</u>: -**

**1.** We can directly specify a color name from color class.

**Ex: -** Color.yellow, Color.red, Color.gray etc…

Here Color is a class, yellow is variable.

**2.** By creating color class object with a combination of red, green, blue values

**Ex: -** Color c = new Color(r,g,b);

R,G,B, values are varies from 0 – 255.

**Ex: -** Color c = new Color(255,0,0);

Color c = new Color(25,0,0);

Color c = new Color(0,0,0); It is a black color

Color c = new Color(255,255,255); It is pure white color

Color c = new Color(100,20,20); it is snuff color

In font there will be three types they are

1. Font.BOLD 2. Font.ITALIC 3. Font.PLAIN

In the above program public void paint(Graphics g)

paint() method belongs to component class.

Graphics is an Abstract class.

g is an object.

**GIF –** Graphic image format

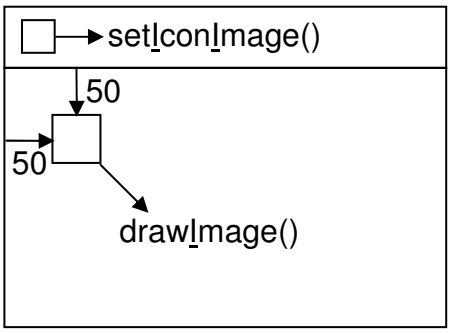
**JPEG –** Joint photographer's expert group

**MPEG –** Motion pictures expert group. (or) Moving pictures expert group

- To display image in a frame for this purpose we should use drawImage of graphic class.

g.drawImage(image, x, y, Image observer obj);

- Image observer containing the address of the image and history of the image.




**Ex: -**

```
//To display an image
import java.awt.*;
import java.awt.event*;
class Message extends Frame
{
  //vars
  static Image img;   //Θ Image is a class name & img is a variable.
  //Default Constructor
 Images()
 {
   //Load an image into img
```

```
        img = Toolkit.getDefaultToolkit().getImage("twist.gif");
        //To keep the processor with till image is loaded into image.
    MediaTracker track = new MediaTracker(this);  //Θ 'this' is a current class object.
        //now add image to track
        track.addImage(img, 0);  //Θ  Here '0' is Image ID No.
        //wait till the image is loaded
        try{
             track.waitForID(0);
            }catch(InterrputedException ie){ }
        //close the frame
        addWindowListener(new windowAdaptor()
         {
           public void windowClosing(windowEvent e)
           {
             System.exit(0);
           }
         });
       } //close constructor
       public void paint(Graphics g)
       {
         //Display the image in frame
         g.drawImage(img, 50, 50, null);
        }
        public static void main(String[ ] args)
        {
        //create the frame
        Image i = new Image();
        //set the size of frame
        i.setSize(500,400);
        //set a title to the frame
        i.steTitle("My Image");
        //display the image in the title bar
        i.setIconImage(img);
        //display the frame
```

```
        i.setVisible (true); (or) i.show();
       }
      }        (or)
```

**Ex: -**

```
//To display an image
import java.awt.*;
import java.awt.event.*;
class Message extends Frame
{
  //vars
  static Image img;   //Θ Image is a class name & img is a variable.
  //Default Constructor
 Images()
 {
   //Load an image into img
   img = Toolkit.getDefaultToolkit().getImage("twist.gif");
   //To keep the processor with till image is loaded into image.
MediaTracker track = new MediaTracker(this);  //Θ 'this' is a current class object.
   //now add image to track
   track.addImage(img, 0);  //Θ  Here '0' is Image ID No.
   //wait till the image is loaded
   try{
        track.waitForID(0);
       }catch(InterrputedException ie){ }
   //close the frame
   addWindowListener(new windowAdaptor()
    {
      public void windowClosing(windowEvent we)
      {
        System.exit(0);
      }
    });
  } //close constructor
  public void paint(Graphics g)
  {
```

 rambabumandalapu@yahoo.com

```
 //Display the image in frame
 g.drawImage(img, 50, 50, 300, 350, null); //Θ 300, 350 are x,y, pixels
}
public static void main(String[ ] args)
{
//create the frame
Image i = new Image();
//set the size of frame
i.setSize(500,400);
//set a title to the frame
i.steTitle("My Image");
//display the image in the title bar
i.setIconImage(img);
//display the frame
i.setVisible(true); (or) i.show();
}
}
```

**Button: -** Button class is useful to create push buttons. A push button triggers a series of events.

**1.** To create a push button with a label

Button b = new Button("label"); // Θ label -The text will displayed on the button.

**2.** To get the label of the button

String l = b.getLabel();

**3.** To set the label of the button:

b.setLabel("label");

**4.** To get the label of the button clicked:

String s = ae.getActionCommand(); //Θ Here ae is object of ActionEvent.

**Listeners: -** A Listener is an interface that listens to an event from component. Listeners are in java.awt.event package.

| S. No. | Component | Listener | Listener Methods |
|---|---|---|---|
| 1. | Button | ActionListener | actionPerformed(ActionEvent e) |
| 2. | CheckBox | ItemListener | itemStateChanged(ItemEvent e) |

| | | | |
|---|---|---|---|
| 3. | CheckBoxGroup | ItemListener | |
| 4. | Label | - - - | |
| 5. | TextField | ActionListener (or) FocusListener | focusGained(FocusEvent e) |
| 6. | TextArea | ActionListener (or) FocusListener | |
| 7. | Choice | ItemListener (or) ActionListener | |
| 8. | List | ItemListener (or) ActionListener | |
| 9. | Scrollbar | AdjustmentListener(or) MouseMotionListener | adjustmentValueChanged(adjustement e) mouseDragged(MouseEvent e) mouseMoved(MouseEvent e) |

**Note 1: -** The above all Listener methods are all 'public void' methods.

**Note 2: -** A Listener can be added to a component using addxxxListener() method.

**Ex: -** addActionListener();

**Note 3: -** A Listener can be removed from a component using removexxxListener() method.

**Ex: -** removeActionListener();

**Note 4: -** A Listener takes an object of xxxEvent class.

**Ex: -** actionPerformed(ActionEvent e);

**Ex: -**

```
//Push Buttons
import java.awt.*;
import java.awt.event.*;
class MyButton extends Frame implenets ActionListener
{
  //vars
  Button b1, b2, b3;
  //Default Constructor
 MyButton()
 {
```

```
//donot set any layout manager
setLayput(new FlowLayout());   // (or) setLayout(null); Here null means not setting any Layout.
//Create 3 push buttons
b1 = new Button("Yellow");
b2 = new Button("Blue");
b3 = new Button("Pink");
//set position of these buttons
b1 = setBounds(50, 100, 75, 40); /*Here 50, 100 are x, y coordinates where the buttons should place in the frame., 75 is the width of button, 40 is the height of the button.*/
b2 = setBounds(50, 150, 75, 40);
b3 = setBounds(50, 200, 75, 40);
//add these buttons to the frame
add(b1);
add(b2);
add(b3);
//add action listener to the frame
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
//close the frame
addWindowListener(new windowAdaptor()
 {
   public void windowClosing(windowEvent we)
   {
     System.exit(0);
   }
 });
} //close constructor
public void actionPerformed(ActionEvent ae)
{
 //Know which button is clicked
 String str = ae.getActionCommand();
```

```
    if(str.equals("Yellow"))
      setBackground(Color.yellow);
    if(str.equals("Blue"))
      setBackground(Color.blue);
    if(str.equals("Pink"))
      setBackground(Color.pink);
  }
  public static void main(String args[ ])
  {
  //create the frame
  MyButton mb = new MyButton();
  //set the size of frame
  mb.setSize(500,400);
  //set a title to the frame
  mb.steTitle("My Push Buttons");
  //display the frame
  mb.setVisible(true); (or) mb.show();
  }
}
```

❖ **What is default layout manager in frame? *******

**Ans: -** BorderLayout manger

❖ **What is default layout manager in Applets? *******

**Ans: -** FlowLayout manager

❖ **What is default layout manager in Panel? *******

**Ans: -** FlowLayout manager

**Frame: -** Frame is also a component. A frame is a top level window that oes not contain in another window. It is a container to place the components.

**1.** To create a Frame, extend your class to Frame class. Then create an object to our class.

**2.** To set a title for the frame, use setTitle()

**3.** To set a size for the frame, use setSize()

**4.** To display the frame, use show() (or) setVisible()

**Refreshing the contents of a Frame: -**

**5.** The component class got a method paint() that paints (refreshes) the area in a frame.

**Displays text in a Frame: -**

**6.** Use Graphics class method: drawString()

**Displaying images in a Frame: -**

**7.** Use Graphics class method: drawImage()

**About components (Component class methods): -**

**8.** An component can be added to a frame using add()

**9.** A component can be removed from a frame using remove()

**10.** All components can removed from frame using removeAll()

**11.** Any component can be displayed using setVisible(true)

**12.** A component can be disappeared using setVisible(false)

**13.** A component's colors can be set using

setBackground()

setForeground()

**14.** Font for the displayed text on the component can be set with setFont();.

**15.** A component can be placed in a particular location in the Frame with setBounds();.

**Creating Font: -**

Font f = new Font();

setFont(f); //Component class methods also in Graphics class.

**Creating Color: -**

**Ex 1: -** setColor(Color.yellow);

**Ex 2: -** Color c = new Color(255, 0, 0);

//Θ (255, 0, 0) are R, G, B, Values respectively

setColor(c);

**Maximum size of screen in pixels: -**

1024 x 768, (or) 800 x 600 (or) 640 x 480

**Check box: -** A check box is a square shape box, which provides a set of options to the user.

**1.** To create a check box:

```
Checkbox cb = new Checkbox("label");
```

**2.** To create a checked checkbox:

```
Checkbox cb = new Checkbox("label", null, true);
```

Here 'null' is checkbox group object.

**3.** To get the state of a checkbox

```
boolean b = cb.getState();
```

**4.** To get the state of Checkbox

```
cb.setState(true);
```

**5.** To get the label of a checkbox

```
String s = cb.getLabel();
```

**6.** To get the selected checkbox label into an array we can use getSelectedObjects(); method. This method returns an array size 1 only.

```
Object x[ ] = cb.getSelectedObjects();
```

**Ex: -**

```
//CheckBoxes
import java.awt.*;
import java.awt.event.*;
class MyCheckbox extends Frame implements ItemListener
{
  //Variables
  Checkbox c1, c2, c3;
  String msg = "  ";
  MyCheckbox()
  {
    //Set flow layout manager
    setLayout(new FlowLayout());
    //Create 3 check boxes
    c1 = new Checkbox("Bold", null, true);
    c1 = new Checkbox("Italic");
    c1 = new Checkbox("Underline");
    //Add checkboxes to frame
    add(c1);
```

```
add(c2);
add(c3);
//Add item listeners to checkboxes
c1.addItemListener(this);
c2.addItemListener(this);
c3.addItemListener(this);
//close the frame
addWindowListener(new windowAdaptor()
{
  public void windowClosing(windowEvent we)
  {
   System.exit(0);
  }
 });
}
public void itemStateChanged(ItemEvent ie)
{
  repaint();  // (or) call paint();
}
public void paint(Graphics g)
 {
    //Display the status of check boxes
    g.drawString("Status of Checkboxes", 200, 100);
    msg = "Bold: "+e1.getState();
    g.drawString(msg, 20, 120);          //Θ 20, 120 are x, coordinates
    msg = "Italic: "+e2.getState();
    g.drawString(msg, 20, 140);
    msg = "Underline: "+e3.getState();
    g.drawString(msg, 20, 160);
 }
 public static void main(String args[ ])
 {
   //Create Frame
   MyCheckbox mc = new MyCheckbox();
```

```
            //Set size and title
            mc.setSize(500, 400);
            mc.setTitle("My Check Boxes);
            //Display the frame
            mc.setVisible(true);
          }
      }
```

Save it as MyCheckbox.java.

> ❖ **Which method is called by repaint() method? *****
>
> Ans: - repaint(); method calls update() method. Then update() method will calls the paint() method.

> ❖ **How can reduce in graphics or awt or frames? *****
>
> **Ans: -** By overridingUpadte() method in specifying background color in update() method.

**Choice menu (or) Choice Box: -** Choice is a popup list of items. Only one item can be selected.

**1.** To create a choice menu

```
Choice ch = new Choice();
```

**2.** To ad items to the choice menu

```
ch.add("text");
```

**3.** To know the name of the item selected from the choice menu

```
String s = ch.getSelectedItem();
```

**4.** To know the index of the currently selected item

```
int i = ch.getSelectedIndex();
```

This method returns -1 if nothing is selected.

**Ex: -**

```
//Choice Box
import java.awt.*;
import java.awt.event.*;
class MyChoice extends Frame implements ItemListener
{
```

```
//Variables
String msg;
Choice ch;
MyChoice()
{
  //Set flow layout manager
  setLayout(new FlowLayout());
  ch = new Choice();
  //add items to choice box
  ch.add("Idly");
  ch.add("Dosee");
  ch.add("Chapathi");
  ch.add("Veg Biryani");
  ch.add("Parata");
  //Add the Choice box to the frame
  add(ch);
  //ad item listener to the choice box
  ch.addItemListener(this);
  //Window closing
  addWindowListener(new windowAdaptor()
  {
    public void windowClosing(windowEvent we)
    {
      System.exit(0);
    }
  });
}
public void itemStateChanged(ItemEvent ie)
{
 //call the paint()
  repaint();  // (or) call paint();
}
public void paint(Graphics g)
 {
```

 rambabumandalapu@yahoo.com

```
        //know the user selected item
        msg = ch.getSelectedItem();
        g.drawString("Selected item: ", 10,150);
        g.drawString(msg, 10,170);
       }
      public static void main(String args[ ])
      {
        //Create Frame
        MyChoice mc = new MyChoice();
        //Set size and title
        mc.setSize(500, 400);
        mc.setTitle("My Choice menu");
        //Display the frame
        mc.setVisible(true);
      }
}
```

**(or)**

Save the above code as MyChoice.java

```
//Choice Box
import java.awt.*;
import java.awt.event.*;
class MyChoice extends Frame implements ItemListener
{
  //Variables
  String msg[ ];
  List ch;
  MyChoice()
  {
    //Set flow layout manager
    setLayout(new FlowLayout());
    ch = new List(3, true);
    //add items to choice box
    ch.add("Idly");
    ch.add("Dosee");
    ch.add("Chapathi");
```

```
ch.add("Veg Biryani");
ch.add("Parata");
//Add the Choice box to the frame
add(ch);
//ad item listener to the choice box
ch.addItemListener(this);
//Window closing
addWindowListener(new windowAdaptor()
{
  public void windowClosing(windowEvent we)
  {
    System.exit(0);
  }
});
}
public void itemStateChanged(ItemEvent ie)
{
 //call the paint()
  repaint();  // (or) call paint();
}
public void paint(Graphics g)
 {
  //know the user selected item
  msg = ch.getSelectedItems();
  g.drawString("Selected items: ", 10,150);
  for(int i = 0; i<msg.length; i++)
  {
    g.drawString(msg[ i ], 10,170 + i * 20);
  }
 }
 public static void main(String args[ ])
 {
  //Create Frame
  MyChoice mc = new MyChoice();
```

 rambabumandalapu@yahoo.com

```
            //Set size and title
            mc.setSize(500, 400);
            mc.setTitle("My Choice menu");
            //Display the frame
            mc.setVisible(true);
          }
        }
```

**List Box: -** A list box is a similar to a choice box, but it allows the user to select multiple items.

**1.** To create a list box

```
List lst = new List(); //Only 1 item can be selected.
List lst = new List(4, true);  //Multiple items can be selected.
//and 4 items are initially visible
```

From the above true means we can select multiple items, if we write false, it means we can select only single item.

**2.** To add items to list box

```
lst.add("text");
```

**3.** To get the selected items

```
String x[ ] = lst.getSelectedItems();
```

**4.** To get the selected indexes

```
int x[ ] = lst.getSelectedIndexes();
```

**Ex: -**

```
//List Box
import java.awt.*;
import java.awt.event.*;
class MyChoice extends Frame implements ItemListener
{
  //Variables
  String msg[ ];
  List ch;
  MyChoice()
  {
    //Set flow layout manager
    setLayout(new FlowLayout());
    ch = new List(3, true);
```

```
//add items to choice box
ch.add("Idly");
ch.add("Dosee");
ch.add("Chapathi");
ch.add("Veg Biryani");
ch.add("Parata");
//Add the Choice box to the frame
add(ch);
//ad item listener to the choice box
ch.addItemListener(this);
//Window closing
addWindowListener(new windowAdaptor()
{
  public void windowClosing(windowEvent we)
  {
    System.exit(0);
  }
});
}
public void itemStateChanged(ItemEvent ie)
{
 //call the paint()
  repaint();  // (or) call paint();
}
public void paint(Graphics g)
 {
   //know the user selected item
   msg = ch.getSelectedItems();
   g.drawString("Selected items: ", 10,150);
   for(int i = 0; i<msg.length; i++)
   {
     g.drawString(msg[ i ], 10,170 + i * 20);
   }
  }
```

 rambabumandalapu@yahoo.com

```
        public static void main(String args[ ])
        {
          //Create Frame
          MyChoice mc = new MyChoice();
          //Set size and title
          mc.setSize(500, 400);
          mc.setTitle("My Choice menu");
          //Display the frame
          mc.setVisible(true);
        }
    }
```

**Radio Buttons: -** A radio button represents a round shaped button, such that only one can be selected from a panel/group. Radio button can be treated using CheckboxGroup class and Checkbox classes.

**1.** To create a radio button

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox cb = new Checkbox("label", cbg, true);
```

**2.** To know the selected checkbox

```
Checkbox cb = cbg.getSelectedCheckbox();
```

**3.** To know the selected checkbox label

```
String label = cbg.getSelectedCheckbox().getLabel();
```

**Ex: -**

```
//CheckboxGroup emo
import java.awt.*;
import java.awt.event.*;
class Myradio extends Frame implements ItemListener
{
  //Variables
  String msg ="  ";
  CheckboxGroup cbg;
  Choicebox y,n;
  Myradio ()
  {
    //Set flow layout manager
    setLayout(new FlowLayout());
```

```
cbg = new CheckboxGroup();
y = new Checkbox("Yes",cbg,true);
n = new Checkbox("No",cbg,false);
add(y);
add(n);
y.addItemListener(this);
n.addItemListener(this);
//Window closing
addWindowListener(new windowAdaptor()
{
  public void windowClosing(windowEvent we)
  {
    System.exit(0);
  }
});
}    //end of constructor
public void itemStateChanged(ItemEvent ie)
{
 //call the paint()
  repaint();  // (or) call paint();
}
//Display the selected checkbox name
public void paint(Graphics g)
{
    msg = "Current Selection:   ";
    msg = cbg.getSelectedCheckbox().getLabel();
    g.drawString(msg, 10, 100);
  }
 public static void main(String args[ ])
 {
   //Create Frame
   Myradio mr = new Myradio();
   //Set size and title
   mr.setSize(500, 400);
```

 rambabumandalapu@yahoo.com

```
            mr.setTitle("My Radio Buttons");
            //Display the frame
            mr.setVisible(true);     //(or) mr.show();
          }
        }
```

**TextField: -** TextFiled allows a user to enter a single line of text.

**1.** To create a TextFiled

TextField tf = new TextField(25); (Θ Here 25 is width of characters)

**2.** To get the text from a TextField

TextField tf = new TextField("default text", 25);

**3.** To set the text into a TextFiled

tf.setText ("text");

**4.** To hide the text begin typed into the TextField, by a character.

tf.setEchoChar ('char');  (or)

**Ex: -** tf.setEchoChar ('*');

**TextArea: -** TextArea is similar to a TextField, but it accepts more than one line of text from the user.

**1.** To create a TextArea

TextArea ta = new TextArea();

TextArea ta = new TextArea(rows, cols);

**Note: -** TextArea supports getText(); and setText();

**Label: -** A Label is a constant text that is displayed with a TextField.

**1.** To create a label

Label l = new Label("text", alignmentconstant);

**Note: -** alignmentconstant: Label.RIGHT, Label.LEFT, Label.CENTER.

**Ex: -**

```
//Labels and TextField
import java.awt.*;
import java.awt.event.*;
class MyText extends Frame implements ActionListener
{
  //Variables
  TextField name, pass;
  CheckboxGroup cbg;
  Label n, p;
```

 rambabumandalapu@yahoo.com

```
MyText()
{
  //Set flow layout manager
  setLayout(new FlowLayout());
  //create 2 labels
  n = new Label("Enter Name:   ", Label.RIGHT);
  p = new Label("Password:   ", Label.RIGHT);
  //create 2 textfields
 name = new TextField(20);
 pass = new TextField(15);
 //Hide the text filed in the pass filed
 pass.setEchoChar('*');
 //add the components to the Frame
 add(n);
 add(name);
 add(p);
 add(pass);
 //add actionListener to text fields
 name.addActionListener(this);
 pass.addActionListener(this);
 //Closing for the Frame
 addWindowListener(new windowAdaptor()
 {
   public void windowClosing(windowEvent we)
   {
     System.exit(0);
  }
 });
}    //end of constructor
public void actionPerformed(ActionEvent ae)
{
 //call the paint()
  repaint();          // (or) call paint();
}
```

 rambabumandalapu@yahoo.com

```
    //Display the selected checkbox name
    public void paint(Graphics g)
    {
      g.drawString("Name:   "+name.getText(),20,100);
      g.drawString("Password:   "+pass.getText(),20,120);
    }
    public static void main(String[ ] args)
     {
      //Create Frame
      MyText mt = new MyText();
      //Set size and title
      mt.setSize(500, 400);
      mt.setTitle("My Text Fields");
      //Display the frame
      mt.setVisible(true);     //(or) mr.show();
     }
}
```

**(or)**

```
//Labels and TextField
import java.awt.*;
import java.awt.event.*;
class MyText extends Frame implements ActionListener
{
  //Variables
  TextField name, pass;
  CheckboxGroup cbg;
  Label n, p;
  MyText()
  {
    //Set flow layout manager
    setLayout(new FlowLayout());
    //create 2 labels
    n = new Label("Enter Name:   ", Label.RIGHT);
    p = new Label("Password:   ", Label.RIGHT);
    //create 2 textfields
```

 rambabumandalapu@yahoo.com

```
name = new TextField(20);
pass = new TextField(15);
//set colors and font for name
name.setBackground(Color.yellow);
name.setForeground(Color.red);
Font f = new Font("Arial", Font.PLAIN, 20);
Name.setFont(f);
//Hide the text filed in the pass filed
pass.setEchoChar('*');
//add the components to the Frame
add(n);
add(name);
add(p);
add(pass);
//add actionListener to text fields
name.addActionListener(this);
pass.addActionListener(this);
//Closing for the Frame
addWindowListener(new windowAdaptor()
{
   public void windowClosing(windowEvent we)
   {
     System.exit(0);
  }
 });
}    //end of constructor
public void actionPerformed(ActionEvent ae)
{
 //call the paint()
  repaint();          // (or) call paint();
}
//Display the selected checkbox name
public void paint(Graphics g)
{
```

```
        g.drawString("Name:   "+name.getText(),20,100);
        g.drawString("Password:   "+pass.getText(),20,120);
      }
      public static void main(String[ ] args)
       {
        //Create Frame
        MyText mt = new MyText();
        //Set size and title
        mt.setSize(500, 400);
        mt.setTitle("My Text Fields");
        //Display the frame
        mt.setVisible(true);     //(or) mr.show();
       }
    }
```

**Ex: -**

```
    //Moving from one frame to another frame
    import java.awt.*;
    import java.awt.event.*;
    class Frame1 extends Frame implements ActionListener
    {
      //Variables
      Button b;
      //Constructor
      Frame1()
      {
      //Closing for the Frame
        addWindowListener(new windowAdaptor()
        {
          public void windowClosing(windowEvent we)
          {
            System.exit(0);
          }
        });
      }     //end of constructor
      public void actionPerformed(ActionEvent ae)
```

 rambabumandalapu@yahoo.com

```
    {
      //Go to the next frame
      Frame2 f2 = new Frame2();
      f2.setSize(300,300);
      f2.setTitle("My Next Window");
      f2.setVisible(true);        //(or) f2.show();
    }
    public static void main(String args[ ])
      {
        //Create first Frame
        Frame1 f1 = new Frame1();
        //Set size and title
        f1.setSize(500, 400);
        f1.setTitle("My Text Fields");
        //Display the frame
        f1.setVisible(true);      //(or) f1.show();
      }
  }
```

Save the above code as Frame1.java, compile it but not run the above code, we have to write below code.

```
//This is Second Frame
import java.awt.*;
import java.awt.event.*;
class Frame2 extends Frame implements ActionListener
{
  //Variables
  Button b;
  //Constructor
  Frame2()
  {
    //Set flow layout manager
    setLayout(new FlowLayout());
    b = new Button("Back");
    add(b);
```

```
        b.addActionListener(this);
      }
      public void actionPerformed(ActionEvent ae)
      {
        //Terminate the present frame
        this.dispose();
       }
    }
```

Save the above code as Frame2.java, compile it but not run the above code, initially we have to run the Frame1.java program. **(or)**

```
//Moving from one frame to another frame
import java.awt.*;
import java.awt.event.*;
class Frame1 extends Frame implements ActionListener
{
  //Variables
  Button b;
  String str = "Hello";
  //Constructor
  Frame1()
  {
    //Set flow layout manager
    setLayout(new FlowLayout());
    b = new Button("Next");
    add(b);
    b.addActionListener(this);
    //Closing for the Frame
    addWindowListener(new windowAdaptor()
    {
      public void windowClosing(windowEvent we)
      {
        System.exit(0);
     }
    });
```

```
    }     //end of constructor
  public void actionPerformed(ActionEvent ae)
  {
    //Go to the next frame
    Frame2 f2 = new Frame2(str);
    f2.setSize(300,300);
    f2.setTitle("My Next Window");
    f2.setVisible(true);        //(or) f2.show();
  }
  public static void main(String args[ ])
    {
      //Create first Frame
      Frame1 f1 = new Frame1();
      //Set size and title
      f1.setSize(500, 400);
      f1.setTitle("My Text Fields");
      //Display the frame
      f1.setVisible(true);      //(or) f1.show();
    }
}
```

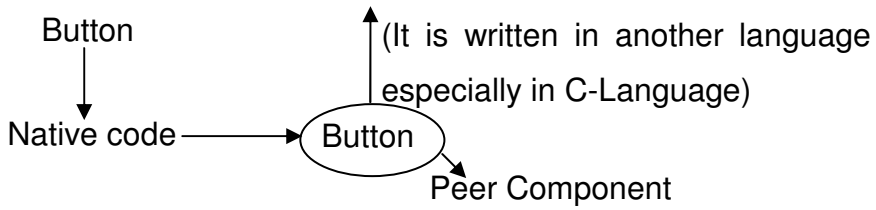
Save the above code as Frame1.java, compile it but not run the above code, we have to write below code.

```
//This is Second Frame
import java.awt.*;
import java.awt.event.*;
class Frame2 extends Frame implements ActionListener
{
  //Variables
  Button b;
  String str;
  //Constructor
  Frame2(String str)
  {
    this.str = str;
```

```
        //Set flow layout manager
        setLayout(new FlowLayout());
        b = new Button("Back");
        add(b);
        b.addActionListener(this);
      }
      public void actionPerformed(ActionEvent ae)
      {
        //Terminate the present frame
        this.dispose();
      }
      public void paint(Graphics g)
      {
        g.drawString(str, 20, 150);
      }
    }
```

Save the above code as Frame2.java, compile it but not run the above code, initially we have to run the Frame1.java program.

AWT is peer component based model




AWT depends on native code. Native code internally creates the peer component. Native code after creating the peer component it will returns the awt.

AWT depends on the Operating System.

All the AWT components are heavy weight components because they take (or) use more resources of the system i.e. they take more memory and take more time to process by the processor.

AWT

- Peer Component based model
- The look and feel of components is changing.
- Heavy weight components

Due to the above disadvantages Javasoft people developed **JFC (Java Foundation Classes)** is a group of classes which are extended from AWT. JFC is written in pure Java language. In JFC we have 5 packages.

**JFC: -** JFC is an extension of the original AWT. It contains libraries that are completely portable.

**1.** JFC components are light-weight. They take less system resources.

**2.** JFC components will have same look and feel or all platforms (or) Operating Systems.

**3.** The programmer can change the look and feel as suited for a platform.

**4.** JFC offers a rich set of components with lots of features. This is the main difference of JFC with AWT.

**5.** JFC does not replace AWT. JFC is an extension of AWT.

**Major packages in JFC: -**

**1. javax.swing: -** To develop components like buttons, menus.

**2. javax.swing.plaf: -** To provide a native look and feel to swing components. It is pluggable. (Θ In javax, x stands for extended package)

**3. java.awt.dnd: -** To drag and drop the components and data from one place to another place. (Θ Where dnd stands for drag & drop the data (or) component).

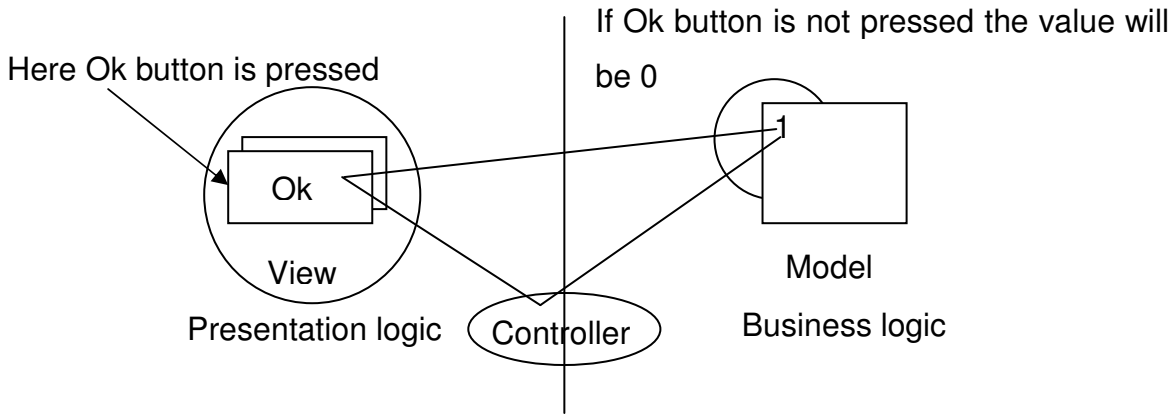
**4. java.accessibility: -** To provide accessibility of applications to disabled persons.

**5. java.awt.geom: -** To draw 2D graphics, filling rotating the shapes, etc.. (Θ Where geom stands for geometrical shapes).

**javax.swing: -**

**1.** All components in swing follow a **Model View Controller (MVC)** architecture.

- **"Model"** stores the data that defines the state of a button (pushed in or not) or text present in a text field.
- **"View"** creates the visual representation of the component from the data in the model.
- **"Controller"** deals with user interaction with the component and modifies the model or the view in response to the user action.




State of the component is represented by a model. Controller works like a mediator. It will inform the component changes view controlling on operator supporting model same data model. We can display different views now display that the data has pie chart i.e. another views.

Different views we can reviews some same data different systems.

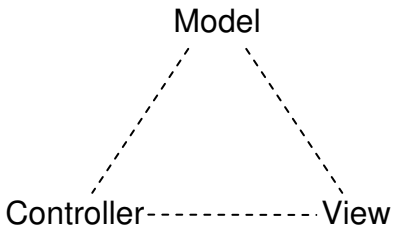
**1.** MVC architecture is useful to creating plof components pluggable and lockable components.

**2.** In MVC the Presentation logic and Business logic have been separated

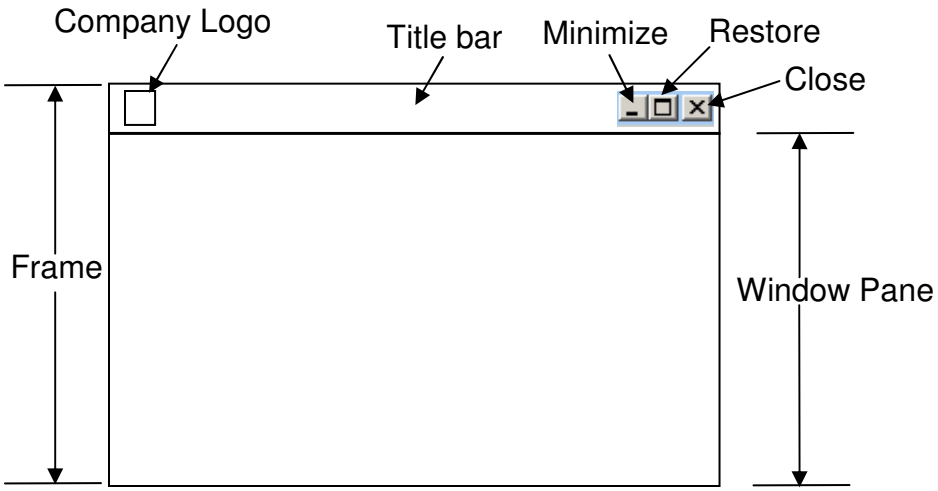
**Advantages: -**

**a)** We can develop Presentation logic and Business logic using different languages.

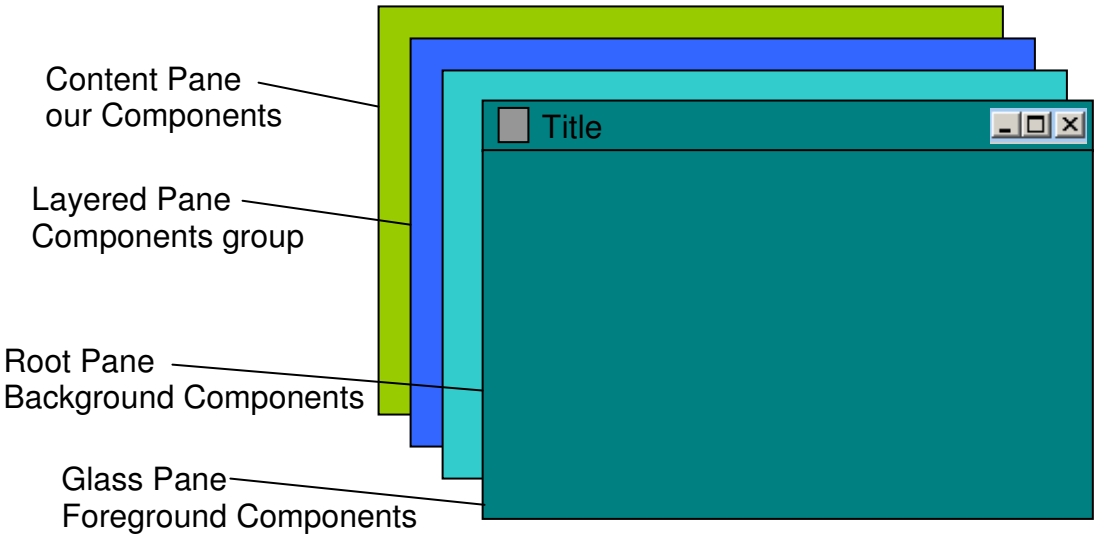
**b)** Any changes to one part can be made without disturbing the other part.




**Window Panes: -** A window pane represents an area of a window. It is an object that can contain other components.




**Window Panes**




**JFrame**

- All individual components are attached to Content pane.
- Groups of components are attached to Layered pane.
- To give background we have to use Root pane.
- To give Foreground we have to use Glass pane.

**JFrame class methods: -**

**1.** getContentPane() : - Returns an object of Container class

**2.** getLayeredPane() : - Returns an object of JLayeredPane class

**3.** getRootPane() : - Returns an object of JRootPane class

**4.** getGlassPane() : - Returns an object of Component class

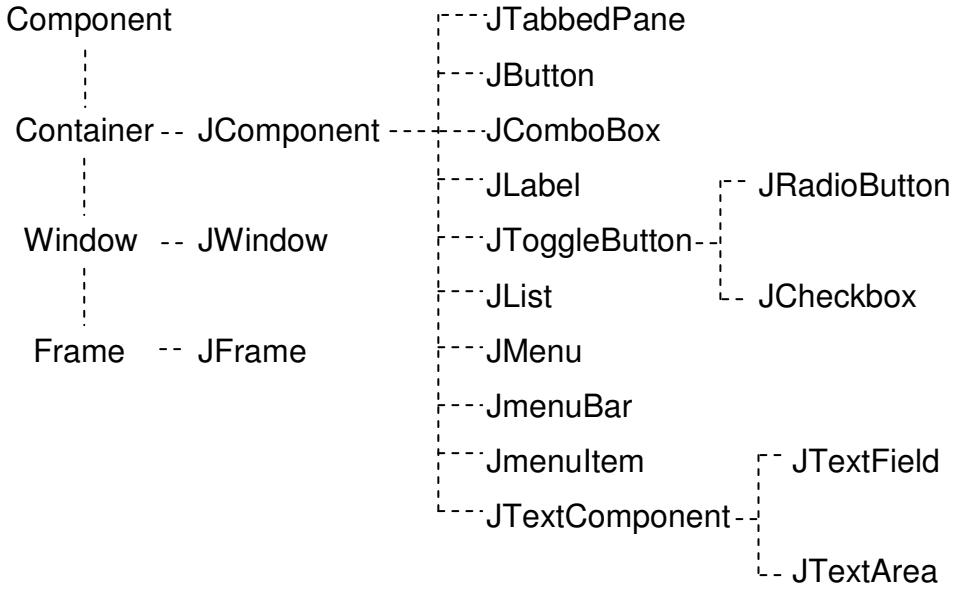
**Ex: -** Going to Content Pane

```
JFrame jf = new JFrame();
```

Container c = jf.getContentPane();

c.add(button);  (Θ Here c is a Container Object).

**Hierarchy of classes in Swing: -**




Component, Container, Window, Frame these four classes are belongs to awt package.

**Creating a Frame: -**

**1.** We can create a Frame b directly creating an object to JFrame

JFrame obj = new JFrame("Title");

**2.** Write a class that extends JFrame then create an object to it.

class MyClass extends JFrame

MyClass obj = new MyClass();

(Θ Here obj represents the JFrame)

**Ex: -**

```
//Creating a Frame in Swing
import java.swing.*;
class MyFrame
{
  public static void main(String args[ ])
  {
    //create the frame
   JFrame obj = new JFrame("My Swing Frame");
   //increase the size of the frame
   obj.setSize(400,350);
```

 rambabumandalapu@yahoo.com

```
    //display the frame
    obj.setVisible(true);
    //close the frame
    obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

**Ex: -**

```
//Creating a Frame in Swing v2.0
import java.awt.*;
import java.swing.*;
class MyFrame extends JFrame
{
  public static void main(String args[ ])
  {
    //create the frame
    MyFrame obj = new MyFrame();
    //set the title for frame
    obj.setTitle("Swing Frame");
    //increase the size of the frame
    obj.setSize(400,350);
    //reach the content pane
    Container c = obj.getContentPane();
    //Set background color to c (Θ Where c is Content pane)
    c.setBackground(Color.green);
    //display the frame
    obj.setVisible(true);
    //close the frame
    obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

**1. Adding Components: -**

To add a component to JFrame, we should add it to container as shown below.

```
Container c = jf.getContentPane();
```

c.add(component);

(or) jf.get.getContentPane().add(component);

**2. Removing Components: -**

To remove a component

c.remove(component);

**3. Setting colors for components: -**

componet.setBackground(color);

componet.setForeground(color);

**4. Setting line Borders: -**

javax.swing.border.BorderFactory class is helpful to set different line borders.

Border bd = BorderFactory.createLineBorder(color.black,3);

Border bd = BorderFactory.createEtchedBorder();

Border bd = BorderFactory.createBevelBorder(BevelBorder.LOWERED);

Border bd = BorderFactory.createBevelBorder(BevelBorder.RAISED);

Border bd = BorderFactory.createMatteBorder(top,left,bottom,right,color.black);

component.setBorder(bd);

**5. Setting tooltip to a component: -**

component.setToolTipText("This is a Swing Component");

**6. Setting a shotcut key: -**

A shortcut key (mnemonic) is an underlined character used in labels, menus and buttons. It is used with ALT key to select an option.

componet.setMnemonic(chart 'c');

**7. Setting a Layout Manager: -**

To set a LayoutManager for the Container

c.setLayout(LayoutManager obj);

**Ex: -**

```
//A push button with all attributes
import java.awt.*;
import java.swing.*;
import java.swing.border.*;
class MyButtton extends JFrame
{
  //variables
 JButton b;
 //default constructor
```

```
MyButtton()
{
  //go to content pane
  Container c = this.getContentPane();
  //set a layout to contentpane
  c.setLayout(new FlowLayout());
  //Create ImageIcon object
  ImageIcon ii = new ImageIcon("twist.gif");
  //create push button with image on it
  b = new JButton("Click me", ii);
  //set colors to b
  b.setBackground(Color.yellow);
  b.setForeground(Color.red);
  //set a font to b
  Font f = new Font("Arial", Font.BOLD,25);
  b.setFont(f);
  //set bevel border around b
  Border bd = BorderFactory.createBevelBorder(BevelBorder.RAISED);
  b.setBorder(bd);
  //set tooltip text for b
  b.setToolTipText("I am a lazy button");
  //set a shortcut key
  b.setMnemonic('c');
  //add button to conten pane
 c.add(b);
 //close the frame
 this.setDefaultOperation(JFrame.EXIT_ON_CLOSE);
 }
  public static void main(String args[ ]);
  {
    //create the frame
    MyButton mb = new MyButton();
    //set size, title
    mb.setTitle("My Push Button");
```

 rambabumandalapu@yahoo.com

```
        mb.setSize(500,400);
        //display the frame
        mb.setVisible(true);
      }
    }
```

**Ex: -**

```
//A push button with all attributes
import java.awt.*;
import java.awt.event.*;
import java.swing.*;
import java.swing.border.*;
class MyButtton extends JFrame implement ActionListener
{
  //variables
 JButton b;
 JLabel lbl;
 //default constructor
  MyButtton()
  {
    //go to content pane
    Container c = this.getContentPane();
    //set a layout to contentpane
    c.setLayout(new FlowLayout());
    //Create ImageIcon object
    ImageIcon ii = new ImageIcon("twist.gif");
    //create push button with image on it
    b = new JButton("Click me", ii);
    //set colors to b
    b.setBackground(Color.yellow);
    b.setForeground(Color.red);
    //set a font to b
    Font f = new Font("Arial", Font.BOLD,25);
    b.setFont(f);
    //set bevel border around b
    Border bd = BorderFactory.createBevelBorder(BevelBorder.RAISED);
```

```
b.setBorder(bd);
//set tooltip text for b
b.setToolTipText("I am a lazy button");
//set a shortcut key
b.setMnemonic('c');
//add button to conten pane
c.add(b);
//create an empty label and add to c
lbl = new Label();
lbl.setFont("Impact", Font.BOLD,25);
c.add(lbl):
//add ActionListener to the button
b.addActionListener(this);
//close the frame
this.setDefaultOperation(JFrame.EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent ae)
{
  lbl.setText("Hello Students");
}
public static void main(String args[ ]);
{
  //create the frame
  MyButton mb = new MyButton();
  //set size, title
  mb.setTitle("My Push Button");
  mb.setSize(500,400);
  //display the frame
  mb.setVisible(true);
 }
}
```

**To create a checkbox and radio buttons**

```
JCheckBox cb = new JCheckBox("label", true);
JRadioButton rb = new JRadioButton("label", true);
```

```
ButtonGroup bg = new ButtonGroup();
bg,ad(rb1);
bg,ad(rb2);
bg,ad(rb3);
JTextField tf = new JTextField(20);
JTextArea ta = new JTextArea(5,20);
```

(Θ Where 5 is No.of rows and 20 is No. of Columns)

- A label represents constant text i.e. displayed in the content pane.

```
JLabel lbl = new JLabel("text", JLabel.RIGHT/JLabel.LEFT/JLabel.CENTER);
```

**Ex: -**

```
//Checkboxes and Radio buttons
import java.awt.*;
import java.awt.event.*;
import java.swing.*;
class CheckRadio extends JFrame implement ActionListener
{
  //variables
 JCheckBox cb1,cb2;
 JRadioButton rb1,rb2;
 JTextArea ta;
 ButtonGroup bg;
 String str = " ";
 //default constructor
  CheckRadio()
  {
    //reach the content pane
    Container c = getContentPane(); (or)  Container c = this.getContentPane();
    //set a layout to contentpane
    c.setLayout(new FlowLayout());
    //create 2 check boxes
    cb1 = new JCheckBox("J2SE", true);
    cb2 = new JCheckBox("J2EE");
    //create 2 check boxes
    rb1 = new JRadioButton("Male", true);
    rb2 = new JRadioButton("Female");
```

```
//Specify these 2 RadioButtons belong to one group
bg = new ButtonGroup();
bg.add(rb1);
bg.add(rb2);
//create a text area
ta = new JTextArea(5,20);
//add all these components to c
c.add(cb1);
c.add(cb2);
c.add(rb1);
c.add(rb2);
c.add(ta);
//add ActionListener to the Checkboxes and Radio Buttons
cb1.addActionListener(this);
cb2.addActionListener(this);
rb1.addActionListener(this);
rb2.addActionListener(this);
//close the frame
this.setDefaultOperation(JFrame.EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent ae)
{
 //To know the user selection
 if(cb1.getModel().isSelected());
 str +="J2SE";
 if(cb2.getModel().isSelected());
 str +="J2EE";
 if(rb1.getModel().isSelected());
 str +="Male";
 if(rb2.getModel().isSelected());
 str +="Female";
 //send user selection to text area
 ta.setText(str);
 str=" ";
```

```
    }
    public static void main(String[ ] args);
    {
     //create the frame
     CheckRadio cr = new CheckRadio();
     //set size, title
     cr.setTitle("My CheckBoxes and Radio Buttons");
     cr.setSize(500,400);
     //display the frame
     cr.setVisible(true);
    }
}
```

**Ex: -**

```
//Checkboxes and Radio buttons
import java.awt.*;
import java.awt.event.*;
import java.swing.*;
class CheckRadio extends JFrame implement ActionListener
{
  //variables
 JCheckBox cb1,cb2;
 JRadioButton rb1,rb2;
 JTextArea ta;
 ButtonGroup bg;
 String str = " ";
 //default constructor
  CheckRadio()
  {
    //reach the content pane
    Container c = getContentPane(); (or)  Container c = this.getContentPane();
    //set a layout to contentpane
    c.setLayout(new FlowLayout());
    //create 2 check boxes
    cb1 = new JCheckBox("J2SE", true);
    cb2 = new JCheckBox("J2EE");
```

```
//set colors
cb1.setBackground(Color.yellow);
cb1.setForeground(Color.red);
cb1.setFont(new Font("Sanserif", Font_ITALIC,20);
//create 2 check boxes
rb1 = new JRadioButton("Male", true);
rb2 = new JRadioButton("Female");
//Specify these 2 RadioButtons belong to one group
bg = new ButtonGroup();
bg.add(rb1);
bg.add(rb2);
//create a text area
ta = new JTextArea(5,20);
//add all these components to c
c.add(cb1);
c.add(cb2);
c.add(rb1);
c.add(rb2);
c.add(ta);
//add ActionListener to the Checkboxes and Radio Buttons
cb1.addActionListener(this);
cb2.addActionListener(this);
rb1.addActionListener(this);
rb2.addActionListener(this);
//close the frame
this.setDefaultOperation(JFrame.EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent ae)
{
 //To know the user selection
 if(cb1.getModel().isSelected());
 str +="J2SE";
 if(cb2.getModel().isSelected());
 str +="J2EE";
```

```
        if(rb1.getModel().isSelected());
        str +="Male";
        if(rb2.getModel().isSelected());
        str +="Female";
        //send user selection to text area
        ta.setText(str);
        str=" ";
      }
      public static void main(String[ ] args);
      {
        //create the frame
        CheckRadio cr = new CheckRadio();
        //set size, title
        cr.setTitle("My CheckBoxes and Radio Buttons");
        cr.setSize(500,400);
        //display the frame
        cr.setVisible(true);
      }
    }
```

- A table represents several rows and columns of information.

**JTable: -** It represents data in the form of a table. The table can have rows of data, and columns headings.

JTable Header →

String arr[4][3]

| | | |
|---|---|---|
| | | |
| | | |
| | data | |
| | | |
| | | |

By using String arr[ ] [ ] we can store only one type of data, if we want to store different type of data it is better to use Vector.

**1. To create a JTable**

JTable tab = new JTable(data, column names);

Here data and column names can be a 2D array or both can be Vector of Vectors,

**2. To create a row using a Vector**

```
Vector row = new Vector();
row.add(object);      //Θ Here object represents a column.
row.add(object);
row.add(object);
```

**3. To create a Table heading, we use getTableHeader()** method of JTable class.

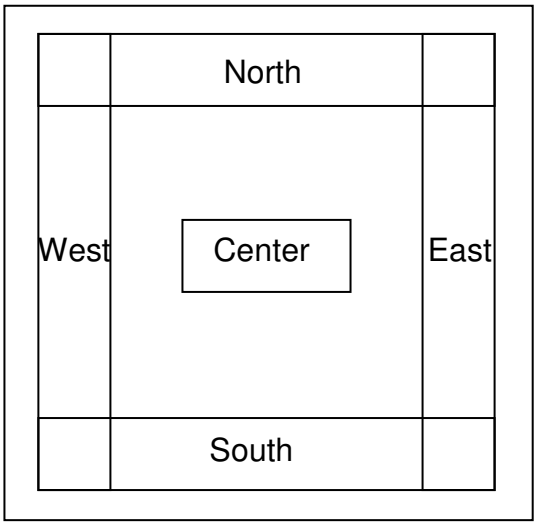
```
JTableHeader head = tab.getTableHeader();
```

**Note: -** JTableHeader class is defined in javax.swing.table package.

**Ex: -**

```
//Creating Employee Table
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.table.*;
import java.util.*;
class Mytable extends JFrame
{
  //Constructor
  Mytable()
  {
    //take data of table as Vector object
    Vector data = new Vector();
    //Create first row
    Vector row = new Vector();
    //store column data in row
    row.add("Subba Rao");
    row.add("System Analyst");
    row.add("12,500.50"); (or) row.add(new Double(12,500.50));
    //add this row to data table
    data.add(row);
    //Create Second row add to data
    row = new Vector();
    row.add("Srinivasa Rao");
```

```
row.add("Sr. Programmer");
row.add("16,000.00");
//add this row to data table
data.add(row);
//Create third row add to data
row = new Vector();
row.add("Sudha Sree");
row.add("Receptionist");
row.add("35,000.75");
//add this row to data table
data.add(row);
//create a row with column names
Vector cols = new Vector();
cols.add("Employee Name");
cols.add("Designation");
cols.add("Salary");
//create the table now
JTable tab = new JTable(data, cols);
//set a border to table
tab.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
//display table heading
JTableHeader head = tab.getTableHeader();
//Go to the content pane
Container c = getContenPane();
//set border layout to c
c.setLayout(new BorderLayout());
//add head and tab to c
c.ad("nort", head);
c.add("Center", tab)'
//close the frame
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[ ])
 {
```



```
    Mytable nt = new Mytable();
      mt.setSize(500,400);
      mt.setVisible(true);
    }
  }
```

**JTabbedPane: -** Pane means a window. TabbedPane means a winow with tab sheets.

JTabbedPane is a container to add multiple components on every tab. The user can choose a component from a tab.

**1. To create a JTabbedPane**

```
JTabbedPane jtp = new JTabbedPane();
```

**2. To add tabs**

```
jtp.addTab("Title", object);
```

Here, the object can be an object of panel.

**3. To create a panel containing some components**

```
class MyPanel extends JPanel
```

Now pass 'MyPanel' class's object to addTab();

**4. To remove a tab (and its components) from the tabbed pane.**

```
jtp.removeTabAt(int index);
```

**5. To remove all the tabs and their corresponding components**

```
jtp.removeAll();
```

- JPanel always represents group of components.

**Ex: -**

```
//Creating Employee Table
import java.awt.*;
import javax.swing.*;
class JTabbeddPaneDemo extends JFrame
{
  //Constructor
  JTabbeddPaneDemo()
  {
    //Create tabbed pane
    JTabbeddPane jtp = new JTabbeddPane();
    //add 2 tab sheets
    jtp.addTab("Countries", new CountriesPanel());
```

```
    jtp.addTab("Capitals", new CapitalsPanel());
   //attach jtp to Content pane
   Container c = getContenPane();
   c.add(jtp);
  }
  public static void main(String args[ ])
  {
   //Create a frame
   JTabbeddPaneDemo demo = new JTabbeddPaneDemo();
  demo.setSize(500,400);
  demo.setVisible(true);
  }
}
class CountriesPanel entends JPanel
{
  CountriesPanel()
  {
    JButton b1, b2, b3;
    b1 = new JButton("India");
    b2 = new JButton("USA");
    b3 = new JButton("Japan");
    add(b1);
    add(b2);
    add(b3);
   }
}
class CapitalsPanel entends JPanel
{
  CapitalsPanel()
  {
    JCheckBox c1 = new JCheckBox("New Delhi");
    JCheckBox c1 = new JCheckBox("Wasington");
    JCheckBox c1 = new JCheckBox("Tokyo");
    add(c1);
```

```
        add(c2);
        add(c3);
      }
    }
```

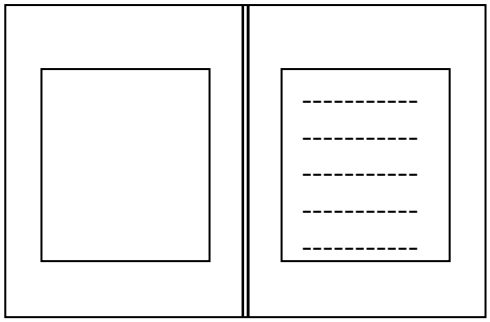
**JSplitPane: -** JSplitPane is used to divide two (and only two) components.

**1. Creating a JSplitPane**

JSplitPane sp = new JSplitPane(orientation, component1, component2);

Here, orientation is JSplitPane.HORINZONTAL_SPLIT to align the componets from left to right.

JSplitPane.VERTICAL_SPLIT to align the componets from top to bottom.




HORINZONTAL_SPLIT

VERTICAL_SPLIT

**2. Setting the divider location between the components**

sp.setDividerLocation(int pixels);

**3. Getting the divider location**

int n = sp.getDividerLocation();

**4. To get the top or left side component**

Component obj = sp.getTopComponent();

**5. To get the bottom or right side component**

Component obj = sp.getBottomComponent();

**6. To remove a component fromsplit pane**

sp.remove(Component obj);

**Ex: -**

```
//Split pane demo
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class JSplitPaneDemo extends JFrame implements ActionListener
```

```
{
  //Variables
   String str = "This is my text being displayed in the text area" + "And this
                    String will wrapped accorindly in the text area";
  JSplitPane sp;
  JButton b;
  JTextArea ta;
  //Constructor
  JSplitPaneDemo()
  {
    //Go to the content pane
   Container c = getContenPane();
   //set border layout to c
   c.setLayout(new BorderLayout());
   //create a push button
   b = new JButton("My Button");
   //create the text area
   ta = new JTextArea();
   //wrap the text in ta
   ta.setLineWrap(true);
   //create the split pane with b, ta
   sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, ta);
   //set the divider location
   sp.setDividerLocation(300);
   //add split pane to c
   c.add("Center", sp);
   //add action listener to b
   b.addActionListener(this);
   //Close the frame
   setDefaultCloseOperaton(JFrame.EXIT_ON_CLOSE);
  }
  public void actionPerformed(ActionEvent ae)
  {
    //send str to ta
```

```
    ta.setText(str);
  }
  public static void main(String args[ ])
  {
    //Create a frame
   JSplitPaneDemo jsp = new JSplitPaneDemo();
   jsp.setSize(500,400);
   jsp.setTitle("My Split Pane");
   jsp.setVisible(true);
 }
}
```

**JTree: -** This component displays a set of hierachical data as an outline.

```
C:\      -------->Root Noe (or) Root Directory
└──── Java Progs ----------> Node ──────── DefaultMutableTreeNode
      ├──── JButton.java ---------->Leaf Node
      ├──── JCheck.java
      └──── JRadioButton.java
```

**1. Create a node of the tree (root node, or node, or leaf node) using DefaultMutableTreeNode class.**

DefaultMutableTreeNode node = new DefaultMutableTreeNode("Java Programs");

**2. Add the tree by supplying root node**

JTree tree = new JTree(root);

**3. To find the path of selected of selected item**

TreePath tp = tse.getNewLoadSelectionPath();

Θ Where tse – tree seletion event

**4.** To find the selected item in the tree

Object comp = tp.getLastPathComponent();

**5. To know the path number (this represents the level)**

int n = tp.getPathCount();

**Ex: -**

```
//Creating a tree with directory names and file names
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;        //TreeSelectionListener
import javax.swing.tree.*;         //TreePath
```

```
class JTreeDemo extends JFrame implements TreeSelectionListener
{
  //Variables
  JTree tree;
  DefaultMutableTreeNode root;
  //Constructor
  JTreeDemo()
  {
    //Go to the content pane
   Container c = getContenPane();
   //set border layout to c
   c.setLayout(new BorderLayout());
   //Create root, node, leaf nodes
   root = new DefaultMutableTreeNode("C:\\");
DefaultMutableTreeNode dir1 = new DefaultMutableTreeNode("Java Progs");
DefaultMutableTreeNode file1 = new DefaultMutableTreeNode("JButton.java");
DefaultMutableTreeNode file2 = new DefaultMutableTreeNode("JCheckBox.java");
DefaultMutableTreeNode file3 = new DefaultMutableTreeNode("JRadioBut.java");
    //add dir1 to root
    root.add(dir1);
   //add file1, file2, file3 to dir1
   dir1.add(file1);
   dir1.add(file2);
   dir1.add(file3);
   //now create the tree
   tree = new JTree(root);
   //add this tree to c
   c.add("North", true);
   //add TreeSelectionListener to tree
    tree.addTreeSelectionListener(this);
    //close the frame
    setDefaultCloseOperaton(JFrame.EXIT_ON_CLOSE);
   }
   public void valueChanged(TreeSelectionListener tse)
```

```
  {
    //Know the path of item selected
    TreePath tp = tse.getNewLeadSelectionPath();
    System.out.println("Path of selected item = " +tp);
    //Know the item selected
    Object comp = tp.getLastPathComponent();
    System.out.println("Item selected = " +comp);
    //Know the level of item selected
    int n = tp.getPathCount();
    System.out.println("Level of item = " +n);
  }
  public static void main(String args[ ])
  {
    //Create a frame
   JTreeDemo demo = new JTreeDemo();
   demo.setSize(500,400);
   demo.setVisible(true);
 }
}
```

**(or)**

**Ex: -**

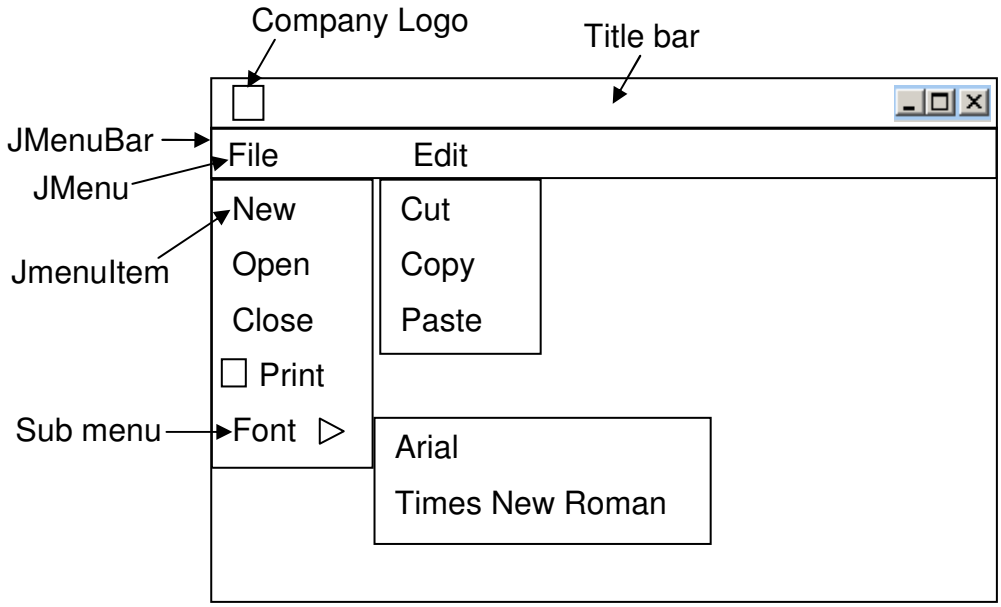
```
//Creating a tree with directory names and file names
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;          //TreeSelectionListener
import javax.swing.tree.*;           //TreePath
class JTreeDemo extends JFrame implements TreeSelectionListener
{
  //Variables
  JTree tree;
  DefaultMutableTreeNode root;
  //Constructor
  JTreeDemo()
  {
    //Go to the content pane
   Container c = getContenPane();
```

 rambabumandalapu@yahoo.com

```
//set border layout to c
c.setLayout(new BorderLayout());
//Create root, node, leaf nodes
root = new DefaultMutableTreeNode("C:\\");
DefaultMutableTreeNode dir1 = new DefaultMutableTreeNode("Java Progs");
DefaultMutableTreeNode file1 = new DefaultMutableTreeNode("JButton.java");
DefaultMutableTreeNode file2 = new DefaultMutableTreeNode("JCheckBox.java");
DefaultMutableTreeNode file3 = new DefaultMutableTreeNode("JRadioBut.java");
//add dir1 to root
root.add(dir1);
//add file1, file2, file3 to dir1
dir1.file1.add(file3);
//now create the tree
tree = new JTree(root);
//add this tree to c
c.add("North", true);
//add TreeSelectionListener to tree
tree.addTreeSelectionListener(this);
//close the frame
setDefaultCloseOperaton(JFrame.EXIT_ON_CLOSE);
}
public void valueChanged(TreeSelectionListener tse)
{
//Know the path of item selected
TreePath tp = tse.getNewLeadSelectionPath();
System.out.println("Path of selected item = " +tp);
//Know the item selected
Object comp = tp.getLastPathComponent();
System.out.println("Item selected = " +comp);
//Know the level of item selected
int n = tp.getPathCount();
System.out.println("Level of item = " +n);
}
public static void main(String args[ ])
```

```
    {
     //Create a frame
    JTreeDemo demo = new JTreeDemo();
    demo.setSize(500,400);
    demo.setVisible(true);
   }
  }
```

**Menu: -** A menu represents a group of options for the user to select.




**Creating a menu: -**

**Steps: -**

**1.** Create menu bar using JMenuBar class

**2.** Add menu bar to the content pane**.**

**3.** Create menus using JMenu class.

**4.** Create menu items separately and add them to menus.

**Creating a Sub menu: -** Sub menu is a menu inside another menu.

**Steps: -**

**1.** Create a menu and add it to another menu.

**2.** Create menu items and add them to sub menu.

**Creating a menu: -**

**1. Create a MenuBar**

```
JMenuBar mb = new JMenuBar();
```

**2. Attach this MenuBar to the container/Container Pane**

```
c.add(mb);
```

**3. Create separate menu to attach to the MenuBar**

```
JMenu file = new JMenu("File");
```

**Note: -** Here, "File" is title for the menu which appears in the MenuBar.

**4. Attach this menu to the MenuBar**

```
mb.add(file);
```

**5. Create menu items using JMenuItem or JCheckBoxMenuItem or JRadioButtonMenuItem classes**

```
JMenuItem nw = new JMenuItem("New");
```

**6. Attach this menu item to the menu**

```
file.add(new);
```

**Creating a sub menu: -**

**7. Create a menu: -**

```
JMenu font = new JMenu();
```

**8. Now attach it to a menu**

```
file.add(font);
```

**9. Attach menu items to the font sub menu**

```
font.add(obj);
```

**Note: -** obj can be a MenuItem or JCheckBoxMenuItem or JRadioButtonMenuItem.

**Ex: -**

```
//Creating a menu
import java.awt.*;                  //It is for Container
import javax.swing.*;               //It is for JMenu, MenuBar etc…
import javax.swing.event.*;         //All Listeners in this package.
class MyMenu extends JFrame implements ActionListener
{
  //Variables
  JMenuBar mb;
  JMenu file, edit, font;
  JMenuItem op, cl, cp, pt;
  JCheckBoxMenuItem pr;
  //Constructor
  MyMenu()
  {
    //Go to the content pane
```

```
Container c = getContenPane();
//set border layout to c
c.setLayout(new BorderLayout());
//create menu bar
mb = new JMenuBar();
//add menu bar to c
c.add("North", mb);
//Create file, edit menus
file = new JMenu("File");
edit = new JMenu("File");
//add file, edit menus to mb
mb.add(file);
mb.add(edit);
//create menu items
op = new JMenuItem("Open");
cl = new JMenuItem("Close");
cp = new JMenuItem("Copy");
pt = new JMenuItem("Paste");
//add op, cl to file menu
file.add(op);
file.add(cl);
//add cp, pt to edit menu
edit.add(cp);
edit.add(pt);
//disable close item
cl.setEnabled(false);
//Create Print check box as a menu item
pr = new JCheckBoxMenuItem("Print");
//add pr to file menu
file.add(pr);
 //add a line
 file.addSeparator();
 //Create a font as a sub menu
 font = new JMenu("Font");
```

```
  //add font to file menu
  file.add(font);
   //set a font for text
   Font f = new Font("Arail");
   Font f1 = new Font("Times New Roman");
   //add menu items to font
   font.add(f1);
   font.add(f2);
   //add listeners to menu items
   op.addActionListener(this);
   cl.addActionListener(this);
   cp.addActionListener(this);
   pt.addActionListener(this);
   pr.addActionListener(this);
   //Clsoe the frame
   setDefaultCloseOperaton(JFrame.EXIT_ON_CLOSE);
 }
 public void actionPerformed(ActionEvent ae)
{
  //Know which item is selected by user
  if(op.isArmed())
  System.out.println("Open Selected");
  if(cl.isArmed())
  System.out.println("Close Selected");
  if(cp.isArmed())
  System.out.println("Copy Selected");
  if(pt.isArmed())
  System.out.println("Paste Selected");
  if(pt.getModel.isSelected())
  System.out.println("Printer On");
}
public static void main(String args[ ])
{
  //Create a frame
```

```
    MyMenu mm = new MyMenu();
    mm.setSize(500,400);
    mm.setVisible(true);
  }
}
```

**(or)**

**Ex: -**

```
//Creating a menu
import java.awt.*;                  //It is for Container
import javax.swing.*;               //It is for JMenu, MenuBar etc…
import javax.swing.event.*;         //All Listeners in this package.
class MyMenu extends JFrame implements ActionListener
{
  //Variables
  JMenuBar mb;
  JMenu file, edit, font;
  JMenuItem op, cl, cp, pt;
  JCheckBoxMenuItem pr;
  //Constructor
  MyMenu()
  {
    //Go to the content pane
    Container c = getContenPane();
    //set border layout to c
    c.setLayout(new BorderLayout());
    //create menu bar
    mb = new JMenuBar();
    //add menu bar to c
    c.add("North", mb);
    //Create file, edit menus
    file = new JMenu("File");
    edit = new JMenu("File");
    //add file, edit menus to mb
    mb.add(file);
    mb.add(edit);
    //create menu items
```

 rambabumandalapu@yahoo.com

```
op = new JMenuItem("Open");
cl = new JMenuItem("Close");
cp = new JMenuItem("Copy");
pt = new JMenuItem("Paste");
//add op, cl to file menu
file.add(op);
file.add(cl);
//add cp, pt to edit menu
edit.add(cp);
edit.add(pt);
//disable close item
cl.setEnabled(false);
//Create Print check box as a menu item
pr = new JCheckBoxMenuItem("Print");
//add pr to file menu
file.add(pr);
 //add a line
 file.addSeparator();
 //Create a font as a sub menu
 font = new JMenu("Font");
 //add font to file menu
 file.add(font);
  //set a font for text
  Font f = new Font("Arail");
  Font f1 = new Font("Times New Roman");
  //add menu items to font
  font.add(f1);
  font.add(f2);
  //add listeners to menu items
  op.addActionListener(this);
  cl.addActionListener(this);
  cp.addActionListener(this);
  pt.addActionListener(this);
  pr.addActionListener(this);
```

```
    //Clsoe the frame
    setDefaultCloseOperaton(JFrame.EXIT_ON_CLOSE);
  }
  public void actionPerformed(ActionEvent ae)
 {
   //Know which item is selected by user
   if(op.isArmed())
   this.openFile();
   if(cl.isArmed())
   System.out.println("Close Selected");
   if(cp.isArmed())
   System.out.println("Copy Selected");
   if(pt.isArmed())
   System.out.println("Paste Selected");
   if(pt.getModel.isSelected())
   System.out.println("Printer On");
 }
 public static void main(String args[ ])
 {
   //Create a frame
  MyMenu mm = new MyMenu();
  mm.setSize(500,400);
  mm.setVisible(true);
}
void openFile();
{
  //display file Selection box
  JFileChooser fc = new JFileChooser();
  int i = fc.showOpenDialog(this);
  //See the file is selected or not
  if(i == JFileChooser.APPROVE _ OPTION)
  System.out.println("U Select: "+fc.getSelectedFile().getName());
 }
}
```

 rambabumandalapu@yahoo.com

- In java internally all constant numbers are represents as integer numbers.

**LayoutManagers: -** A layout manager arranges the components in a container.

**GridLayout: -** It arranges the components in a 2D grid.

GridLayout()

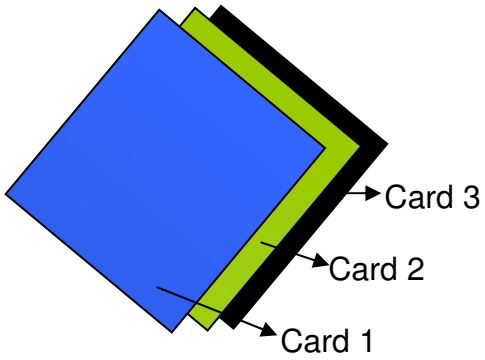
GridLayout(int rows, int cols)

GridLayout(int rows, int cols, int hgap, int vgap)

**CardLayout: -** It arranges the components in the form of deck of cards.

CardLayout()

CardLayout(int hgap, int vgap)




**1. To retrieve the cards**

void first(container)

void last(container)

void next(container)

void previous(container)

void show(container, "Card Name");

**2. To add the component: -**

add("cardname", component);

**Ex: -**

```
//GridLayout Demo
import java.awt.*;
import javax.swing.*;
class GridLayoutDemo extends JFrame
{
  //Constructor
  GridLayoutDemo()
  {
    //Go to the content pane
   Container c = getContenPane();
   JButton b1, b2, b3, b4;
   GridLayout grid = new GridLayout();
   c.setLayout(grid);
    b1 = new JButton("Button1");
    b2 = new JButton("Button2");
    b3 = new JButton("Button3");
    b4 = new JButton("Button4");
```

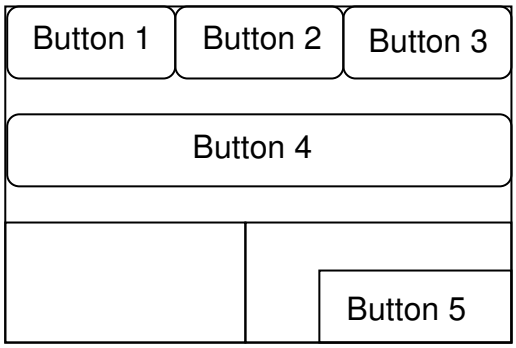
```
    c.add(b1);
    c.add(b2);
    c.add(b3);
    c.add(b4);
   }
   public static void main(String args[ ])
  {
    //Create a frame
   GridLayoutDemo demo = new GridLayoutDemo();
   demo.setSize(500,400);
   demo.setVisible(true);
  }
}
```

**Ex: -**

```
//CardLayout Demo
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class CardLayoutDemo extends JFrame implements ActionListener
{
  Container c;
  CardLayout card;
  //Constructor
  CardLayoutDemo()
  {
    //Go to the content pane
   c = getContenPane();
   card = new CardLayout();
   c.setLayout(card);
   JButton b1, b2, b3;
   b1 = new JButton("Button1");
   b2 = new JButton("Button2");
   b3 = new JButton("Button3");
   c.add("Fisrt Card", b1);
   c.add("Second Card",b2);
```

```
    c.add("Third Card",b3);
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
  }
  public void actionPerformed(ActionEvent ae);
  {
    card.next(c);
  }
  public static void main(String args[ ])
  {
    //Create a frame
   CardLayoutDemo demo = new CardLayoutDemo();
   demo.setSize(500,400);
   demo.setVisible(true);
  }
 }
```

**GridBagLayout: -** It specifies the components in a grid like fashion, in which some components span more than one row or column.



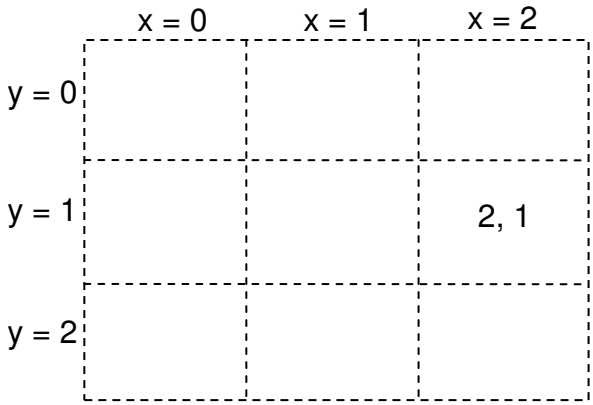

GridBagLayout()

**1.** It specify constraints for positioning of the components.

We can use GridBagConstraints class object.

GridBagConstriants()

**gridx, gridy: -** Specify the row and column at the upper left of the component.




**gridwidth, gridheight: -** Specify the number of colums (for gridwidth) or rows (for gridheight) in the components display area. The efault value is 1.

 rambabumandalapu@yahoo.com

**weightx, weighty: -** To specify whether the component can be stretched horizontally or vertically, when resized. Generally weights are specified between 0.0 and 1.0 (0.0 tells that the components size will not change when resized.

**anchor: -** It is used when the componet is smaller than its display area to determine where (within the area) to place the component. Valid values are shown below.

| | | |
|---|---|---|
| FIRST_START | PAGE_START | FIRST_LINE_END |
| LINE_START | CENTER | LINE_END |
| LAST_LINE_END | PAGE_END | LAST_LINE_END |

**fill: -** It is used the components display area is larger than the component's size to determine whether and how to resize the componet.

**Ex: -** None(default); HORIZONTAL, VERTICAL, BOTH.

**inset: -** Space around the component (external padding) in four corners.

**Ex: -** insets(0,0,0,0) //default

**ipadx, ipady: -** To specify internal padding (widthwise or heightwise) to be added to the component. Default is 0.

**Ex: -**

```
//GridBagLayout Demo
import java.awt.*;
import javax.swing.*;
class GridBagLayoutDemo extends JFrame
{
  GridBagLayout gbag;
  GridBagConstraints cons;
  //Constructor
  GridBagLayoutDemo()
  {
    //Go to the content pane
   Container c = getContenPane();
   gbag = new GridBagLayout();
   c.setLayout(gbag);
   JButton b1 = new JButton("Button1");
   JButton b2 = new JButton("Button2");
   JButton b3 = new JButton("Button3");
   JButton b4 = new JButton("Button4");
   JButton b5 = new JButton("Button5");
```

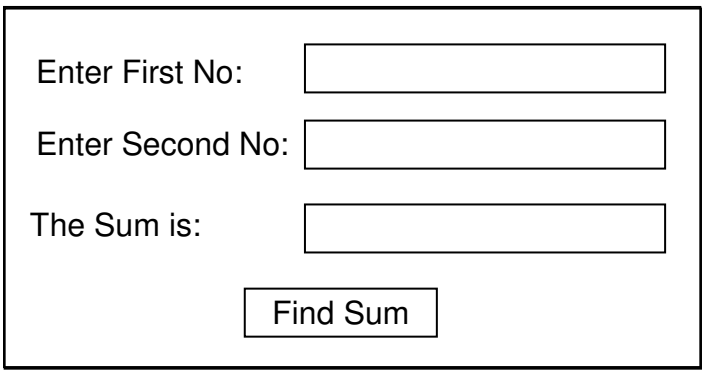
```
    cons.fill = GridBagConstraints.HORIZONTAL;
    cons.gridx = 0;
    cons.gridy = 0;
    cons.weightx = 0.7;        //cons.weightx = 0.0;
    gbag.setConstraints(b1, cons);
    c.add(b1);
    cons.gridx = 1;
    cons.gridy = 0;
    gbag.setConstraints(b2, cons);
    c.add(b2);
    cons.gridx = 1;
    cons.gridy = 0;
    gbag.setConstraints(b3, cons);
    c.add(b3);
    cons.gridx = 1;
    cons.gridy = 0;
    cons.ipady = 100; //make this component tall
    cons.weightx = 0.0;
    cons.gridwidth = 3;
    gbag.setConstraints(b4, cons);
    c.add(b4);
    cons.gridx = 1;            //aligned with button2
    cons.gridy = 2;            //third row
    cons.ipady = 0;            //reset to default
    cons.weighty = 1.0;        //request any extra vertical space
    //button of space
    cons.anchor = GridBagConstraints.PAGE_END;
    //top padding
    cons.insets = new Insets(10, 0, 0, 0);
    //2 columns wide
    cons.gridWidth = 2;
    gbag.setConstraints(b5, cons);
    c.add(b5);
  }
```

 rambabumandalapu@yahoo.com

```
public static void main(String args[ ])
{
  //Create a frame
 GridBagLayoutDemo demo = new GridBagLayoutDemo();
 demo.setSize(500,400);
 demo.setVisible(true);      //(or) demo.show();
}
}
```

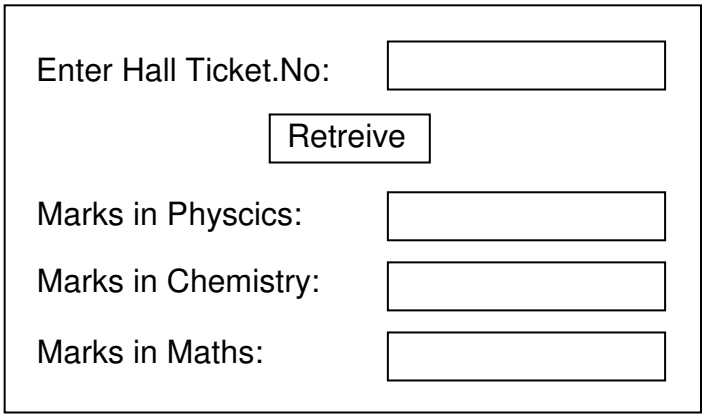
**H.W: -**

**12.** Create 2 text filelds to enter 2 numbers into them. When the push button is clicked, display their sum in the third textfiled.




**13.** Create a calculator using textfield and pushbuttons also ass the functionality to it.

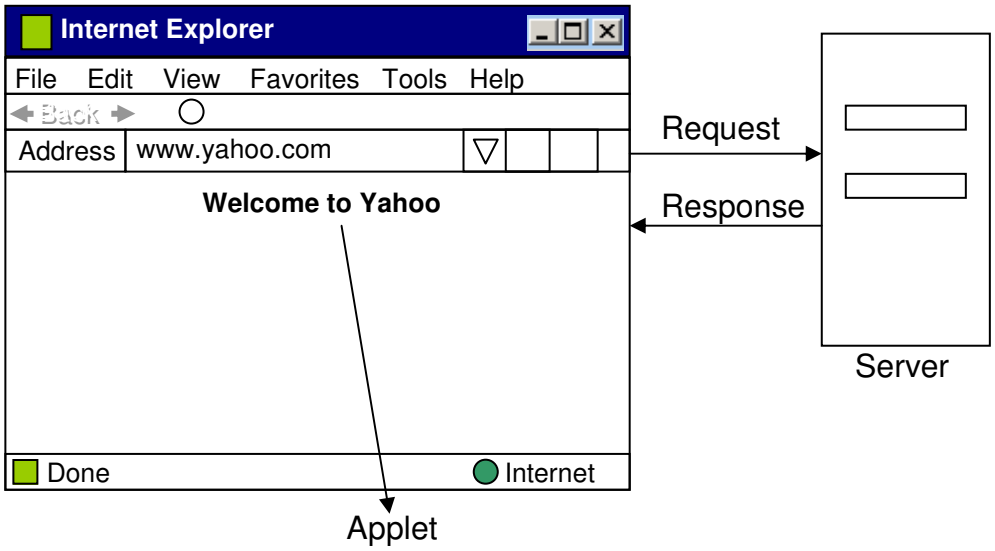
**14.** Take a student class with H.No. Name and Marks in the Physcics, Maths and Chemistry. Create the following screen which accepts H.No. and ddisplay the rest of details in text fileld.




**<u>Applet</u>: -** An applet is a program which comes from server travels on the internet and executes in the client machine Applet is also java program.

Applet = Java Code + HTML Page

An Applet is a java program embded in HTML page. To create Applets in java we can use Applet class of java.applet package or JApplet class of javax.swing package.




**Methods available in Applet Class: -** In an Applet we will have following methods.

**1. public void init(): -** It is useful to initialize variables and any parameters. It is also useful to create components and attach to frame.

init() or this method is executed immediately when an Applet is loaded into memory. This method is executed only once.

**2. public void start(): -** start() method is executed after init() method. This method is executed as long as an applet receives focus. Coe related to opening files, connecting to databases and processing the data is written inside the start() method.

**3. public void stop(): -** This method is executed when the applet loses the focus (When the application is minimized) code related closing the files, disconnecting from databases, and any cleanup operations is written in stop() method.

start(), stop() methods are repeatedly executed when the application (maximize, minimized) receives focuse and loses the focus.

**4. public void destroy(): -** It will terminate the applet from memory this method executed when the web page is closed.

**Note: -** Always the stop() method is executed before the destroy() method is called.

**Note: -** Executed the above methods in that order is called **"Applet Life Cycle".**

❖ **What is the Applet Life Cycle? *******

**Ans: -** init() ⟶ start() ⟶ stop() ⟶ destroy() and explain the above steps clearly.

**Note: -** public void paint(Graphics g) we can also use awt swing componets

❖ **Where are the Applet executed? *****

Ans: - Applets are executed in the web browser software of the client. In web browser (software will contain) a small JVM called applet engine will executes the applets. Applet will execute in limited memory or in a sand memory.

❖ **What is the advantage using of Apple? *****

Ans: - Applets make the webpages to dynamically interact with the users.

**Ex: -**

```
//A sample Applet
import java.awt.*;
import javax.applet.*;
public class MyApp extends Applet
{
  String str = " ";
  public void init()
  {
    setBackground(Color.yellow);
    setForeground(Color.red);
    Font f = new Font("Dialog", Font.BOLD, 20);
    setFont(f);
    str +="init";
  }
  public void init()
  {
    str +="start";
  }
  public void paint(Graphics g)
  {
    str +="paint";
    g.drawString(str, 20, 50);
  }
```

```
    public void stop()
    {
      str +="stop";
    }
    public void destroy()
    {     }
  }
```

Save the above code as MyApp.java and compile it but don't run it, because we have to embded with the HTML of this byte code.

```
<! This HTML page embdeds MyApp applet>
<html>
<applet code = "MyApp.class" width =500, height =400>
</applet>
</html>
```

Save it as x.html.

Open IntenetExplorer page.

Applet viewer is a tool to test the applet. It is developed by the (Software people) Sum Microsystem.

F:\rnr>appletviewer x.html↵ (Θ Where as appletviewer is a tool in J2SDK)

**IIS –** Internet Informat Server.

- Hyper text means the text that uses hyperlink to directly jump to any piece of information.
- <html> tag specify the text, that is hyper link.
- <applet> tag is useful to embded (or) insert applet code into a HTML page.
- Attributes provides additional information of the tag.

**Ex: -**

```
//Creating a shooping form
import java.awt.*;
import java.awt.event*;
import javax.swing.*;
import java.applet.*;
public class MyForm extends JApplet implements ActionListener
{
  String str = "  "; str1 = "  "; str2 = "  ";
```

```
JLabel n, a, l, lbl;
JTextField name;
JTextArea addr;
JList lst;
object x[ ];
JButton b1, b2;
Container c;
public void init()
{
  //Create the frame and goto contentpane
  JFrame jf = new JFrame();
  c = jf.getContentPane();
  c.setLayout(null);
  c.setBackground(Color.yellow);
   jf.setSize(500,400);
   jf.setTitle("My Form");
   jf.setVisible(true);
   //Heading
   lbl = new JLabel();
   lbl.setText("INetSolv Online Shopping Form");
   lbl.setFont(new Font("Dailog", Font.BOLD, 30););
   lbl.setForeground(Color.red);
   lbl.setBouns(200, 10, 500, 40);
   c.add(lbl);
   //name label and text field
   n = new JLabel("Enter Name", JLabel.RIGHT);
   name = new JTextField(20);
   n.setBounds(50, 50, 100, 40);
   name.setBounds(200, 50, 200, 40);
   c.add(n);
   c.add(name);
   //add label an text area
   a = new JLabel("Enter Address", JLabel.RIGHT);
   addr = new JTextArea(6, 30);
```

```
  a.setBounds(50, 50, 100, 40);
  addr.setBounds(200, 100, 200, 150);
  c.add(a);
  c.add(addr);
  //items label and list box
  i = new JLabel("Select Items", JLabel.RIGHT);
  String data[ ] = {"T- Shirts", "Sarees", "Punjabees", "Shorts"};
  lst = new JList(data);
  i.setBounds(50, 260, 100, 40);
  lst.setBounds(200, 260, 200, 150);
  c.add(i);
  c.add(lst);
  //Create 2 push buttons
  b1 = new JButtons("Ok");
  b2 = new JButtons("Cancel");
  b1.setBounds(200, 420, 75, 40);
  b2.setBounds(300, 420, 75, 40);
  c.add(b1);
  c.add(b2);
  //add action Listeners to buttons
  b1.addActionListener(this);
  b2.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
  //Know which button is clicked
  str = ae.getActionCommand();
  if(str.equals("Ok");
  {
    str1 += name.getText() + "\n";
    str1 += addr.getText() + "\n";
    x = lst.getSelectedValues();
    for(int i = 0; i<x.length; i++)
    str2 += (String)x[ ] + "\n";
```

```
            //display user selection in addr box
             addr.setText(str1 + str2);
             str1 = " ";
             str2 = " ";
            }
            else{
                  //Clear the form data
                  name.setText("  ");
                  addr.setText("  ");
                  lst.clearSelection();
                 }
          }
        }
```

Save it MyForm.java. We can run this program by JVM. We can compile it. Now create Applet

```
<! This is HTML page that contains MyForm>
<html>
<applet code = "MyForm.class" width = 500 height = 400>
</applet>
<html>
```

Save it as MyForm.html

F:\rnr>appletviewer MyForm.html↵

**Animation: -** Moving components/objects from one place to another is called animation.

**Ex: -**

```
//Animation in applets
import java.awt.*;
import java.applet.*;
public class Animate extends Applet
{
  public void init()
  { }
  public void paint(Graphics g)
  {
```

```
        Image i = getImage(getDocumentBase(), "plane.gif");
        for(int x =0; x<800; x++);
        {
          g.drawImage(i, x, o, this);
          try{
               Thread.sleep(20);
             }
             catch(InterruptedException ie){ }
         }
       }
     }
```

Save the above code as Animate.java and compile it but don't run it, because we have to embded with the HTML of this byte code.

```
<! HTML that contains Animate.class>
<html>
<applet code = "Animate.class" width =500, height =400>
</applet>
</html>
```

Save it as Animate.html.

Then we have create plane.gif using Flash, DreamWeaver, Adobe PhotoShop. Save it as plane.gif in the same irectory "rnr". But we will draw using Ms-Paint/Ms-PowerPonit.

```
F:\rnr>appletviewer Animate.html↵
```

In the above code (Applet.java)

```
Image i = getImage(getDocumentBase(), "plane.gif");
```

Where getImage will loads the image in image object

getDocumentBase() method will gives URL of the image (or) current directory path.

g.drawImage(i, x, o, this); this is image present object.

**H.W.**

**15.** Create an applet that animates two objects simulataneously on the screen using two threads.

**Generic types are parameteize types: -** This is introduced in Java 1.5. version.

Generic types represents classes or interfaces, which are type-safe. Generic classes and interfaces Generic methods handle any type of data.

Generic types or declared as sub types of object class so they act upon any class object. Generic types can't act up on primitive types.

**Ex: -**

```
//A generic class – to store any types of data
class MyClass
{
  this.obj= obj;
}
integer.getObj()
{
  return obj;
}
}
class Gent
{
  public static void main(String args[ ])
  {
    Integer i = new Integer(100);
    MyClass obj = new MyClass();
    System.out.println("U Stored" +obj.getObj());
```

**Generic class: -**

```
 class MyClass
{
  T obj;
   MyClass()
   {
     this.obj = obj;
   }
   T.getObj()
  {
  return obj;
  }
```

```
class Gen1
{
  public static void main(String args[ ])
  {
    Integer i = new Integer(100);
    MyClass< Integer> obj = new MyClass<Integer>();
    System.out.println("U Stored" +obj.getObj());
  }
}
```

**Ex: -**

```
//A generic method – to ispaly any type of array elements
class MyClass
{
  static void display(int[ ] arr)
  {
    for(int i:arr)   //this is for loop
    System.out,println(i);
   }
 }
 class Gen2
{
  public static void main(String args[ ])
  {
    int arr1[ ] = {1, 2, 3, 4, 5};
    MyClass.display(arr1);
  }
}
```

**2.**

```
class MyClass
{
  static <GT>void display(GT[ ] arr)
  {
    for(GT i:arr)   //this is for loop
    System.out,println(i);
   }
 }
```

```
 class Gen2
{
  public static void main(String args[ ])
  {
    integer arr1[ ] = {1, 2, 3, 4, 5};
    MyClass.display(arr1);
    Double arr2[ ] = {1.1, 2.2, 3.3, 4.4, 5.5};
    MyClass.display(arr2);
    String arr3[ ] = {“Prasad”, “Siva”, “Srinu”, “Guru”};
    MyClass.display(arr3);
  }
}
```

**Ex: -**

```
//Hashtable OS rewritten in JDK 1.5 as Hashtable <KV>
import java.util.*;
class HT
{
  public static void main(String args[ ])
  {
    Hashtable ht = new Hashtable();
    ht.put(“Ajay”, new Integer(50));
    ht.put(“Sachin”, new Integer(100));;
    String s =”Sachin”;
    Integer score = (Integer) ht.get(s);
    System.out.println(“Score = ” +score);
   }
}
import java.util.*;
class HT
{
    Hashtable<String, Integer> ht = new Hashtable<String, Integer>();
    ht.put(“Ajay”, 50);
    ht.put(“Sachin”, 100);;
    String s =”Sachin”;
    Integer score = (Integer) ht.get(s);
```

```
        System.out.println("Score = " +score);
      }
    }
```

**J2SE 1.5: -**

To grasp the magnitude of the changes that J2SE 5 made to Java, consider the following list of its major few features:

- Genrics
- Metadata
- Autoboxing and auto-unboxing
- Enumerations
- Enchanced, for-each style for loop
- Variable-length arguments (varagrs)
- Static import
- Formatted I/O
- Concurrency utilities
- Upgrades to the API

**Generic class: -**

**1.** A generic class or generic interface represents a class or an interface which is type-safe.

**2.** A generic class, generic interface or a generic method can handle any type of data.

**3.** Generic types are sub types of class object.

**4.** So they can act on objects of any class.

**5.** They cannot act on primitive data type.

**6.** Java compiler creates a non-generic versions using the specified data type from a generic version.

**7.** When generic type is used, we can elimainate casting in any class.

**8.** The class of java.util have been rewritten using generic types.

**9.** We cannot create an object to generic type.

**Ex: -**
```
class MyClass <T>  //Invalid
T obj = new T();   //Invalid
```

**H.W.**

**16.** Write a generic metho swaps any two data types.

**The advantages of a 3-tiered or n-tiered application:** 3-tier or multi-tier architectures force separation among presentation logic, business logic and database logic. Let us look at some of the key benefits:

- Manageability: Each tier can be monitored, tuned and upgraded independently and different people can have clearly defined responsibilities.
- Scalability: More hardware can be added and allows clustering (i.e. horizontal scaling).
- Maintainability: Changes and upgrades can be performed without affecting other components.
- Availability: Clustering and load balancing can provide availability.
- Extensibility: Additional features can be easily added.

*******END*******

 rambabumandalapu@yahoo.com