SYLLABUS JAVA PROGRAMMING

Paper Code: ETCS-307	**L**	**T/P**	**C**
Paper: Java Programming	**3**	**1**	**4**

INSTRUCTIONS TO PAPER SETTERS: **MAXIMUM MARKS: 75**

Objective: To learn object oriented concepts and enhancing programming skills.

UNIT I

Overview and characteristics of Java, Java program Compilation and Execution Process Organization of the Java Virtual Machine, JVM as an interpreter and emulator, Instruction Set, class File Format, Verification, Class Area, Java Stack, Heap, Garbage Collection. Security Promises of the JVM, Security Architecture and Security Policy. Class loaders and security aspects, sandbox model [T1,R2][No. of Hrs.: 11]

UNIT II

Java Fundamentals, Data Types & Literals Variables, Wrapper Classes, Arrays, Arithmetic Operators, Logical Operators, Control of Flow, Classes and Instances, Class Member Modifiers Anonymous Inner Class Interfaces and Abstract Classes, inheritance, throw and throws clauses, user defined Exceptions, The String Buffer Class, tokenizer, applets, Life cycle of applet and Security concerns. [T1,T2][No. of Hrs.: 12]

UNIT III

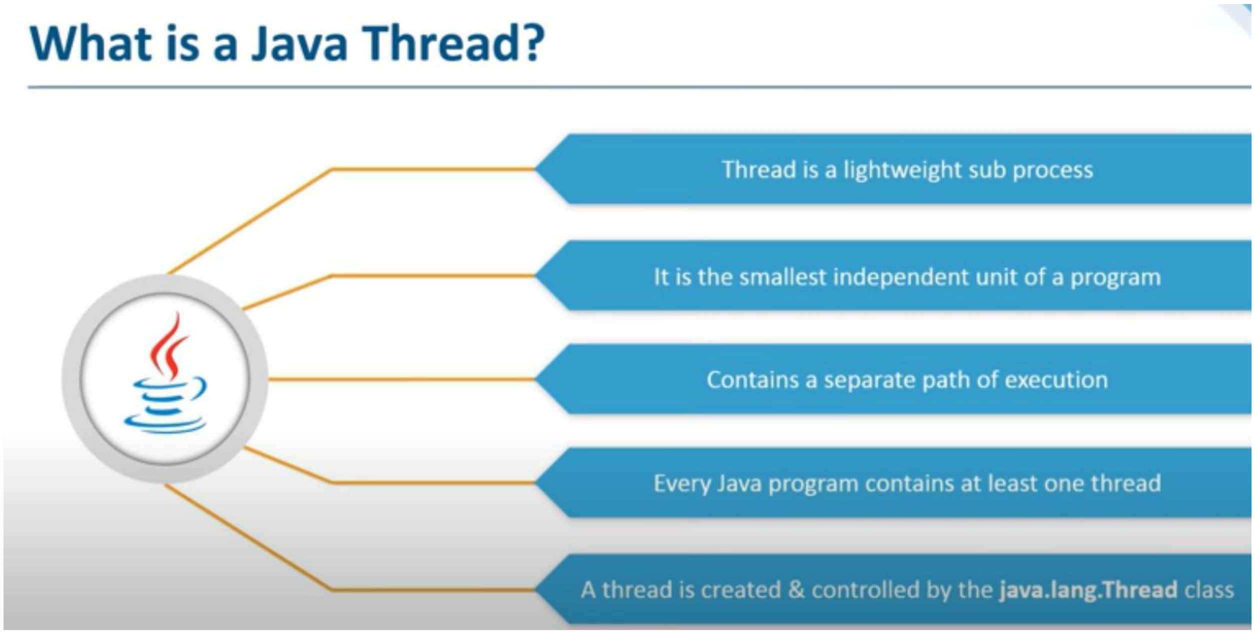
Threads: Creating Threads, Thread Priority, Blocked States, Extending Thread Class, Runnable Interface, Starting Threads, Thread Synchronization, Synchronize Threads, Sync Code Block, Overriding Synced Methods, Thread Communication, wait, notify and notify all.AWT Components, Component Class, Container Class, Layout Manager Interface Default Layouts, Insets and Dimensions, Border Layout, Flow Layout, Grid Layout, Card Layout Grid Bag Layout AWT Events, Event Models, Listeners, Class Listener, Adapters, Action Event Methods Focus Event Key Event, Mouse Events, Window Event [T2][No. of Hrs.: 11]

UNIT IV

Input/Output Stream, Stream Filters, Buffered Streams, Data input and Output Stream, Print Stream Random Access File, JDBC (Database connectivity with MS-Access, Oracle, MS-SQL Server), Object serialization, Sockets, development of client Server applications, design of multithreaded server. Remote Method invocation, Java Native interfaces, Development of a JNI based application. Collection API Interfaces, Vector, stack, Hashtable classes, enumerations, set, List, Map, Iterators. [T1][R1][No. of Hrs.: 10]

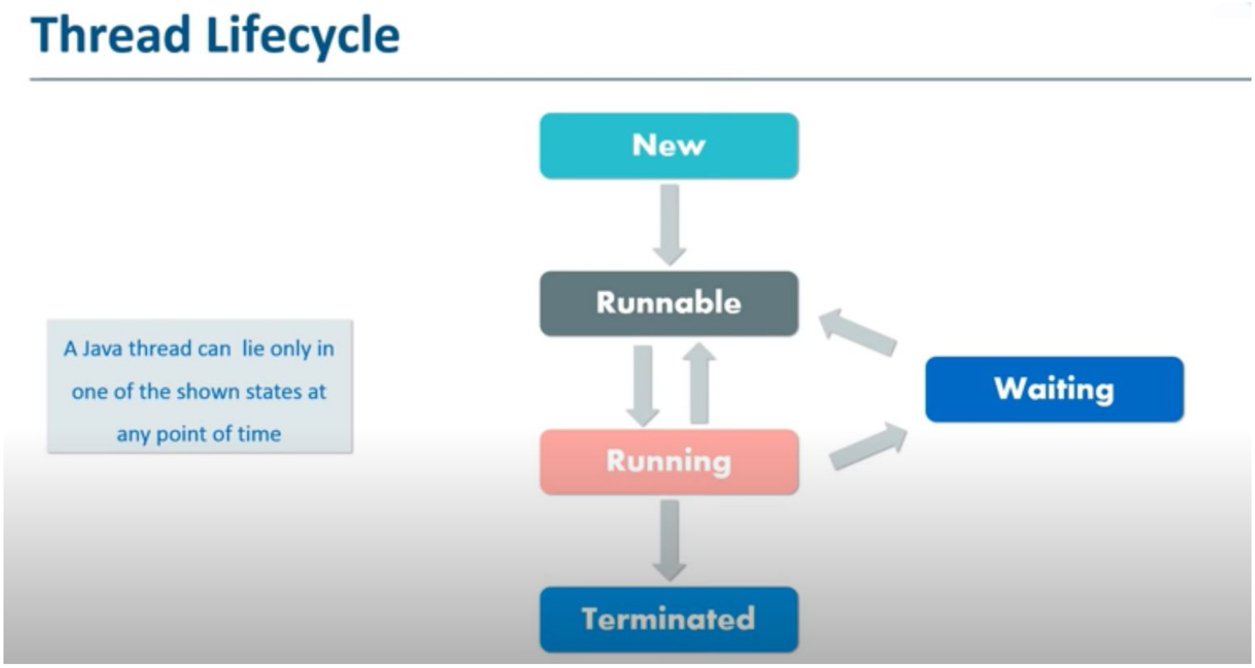
Thread:

A thread is actually a lightweight process. Unlike many other computer languages, Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called thread and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.



What is a Java Thread?

- Thread is a lightweight sub process
- It is the smallest independent unit of a program
- Contains a separate path of execution
- Every Java program contains at least one thread
- A thread is created & controlled by the **java.lang.Thread** class



Thread Lifecycle

New

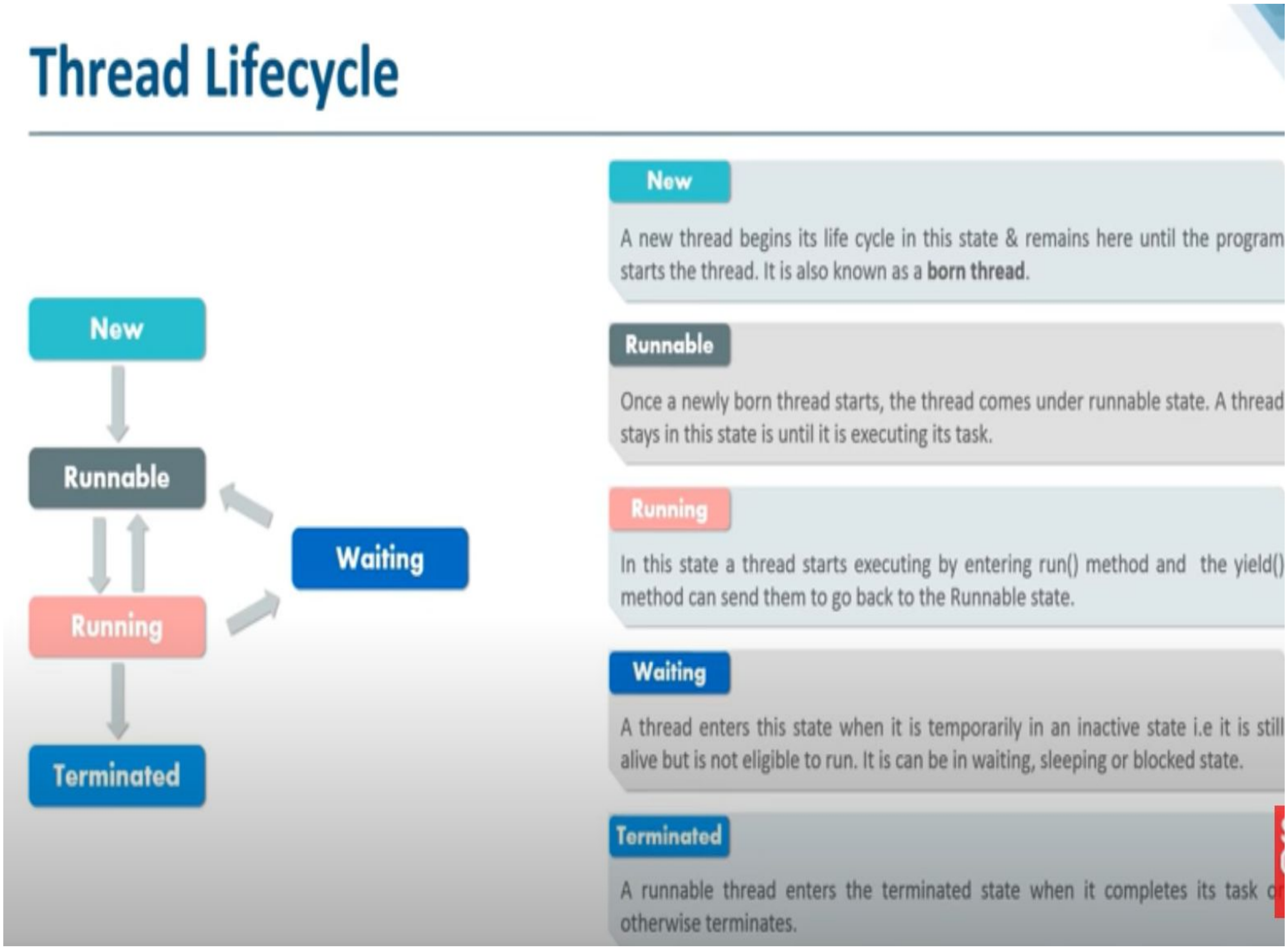
Runnable

Running

Waiting

Terminated

A Java thread can lie only in one of the shown states at any point of time



Thread Lifecycle

New

A new thread begins its life cycle in this state & remains here until the program starts the thread. It is also known as a **born thread**.

Runnable

Once a newly born thread starts, the thread comes under runnable state. A thread stays in this state is until it is executing its task.

Running

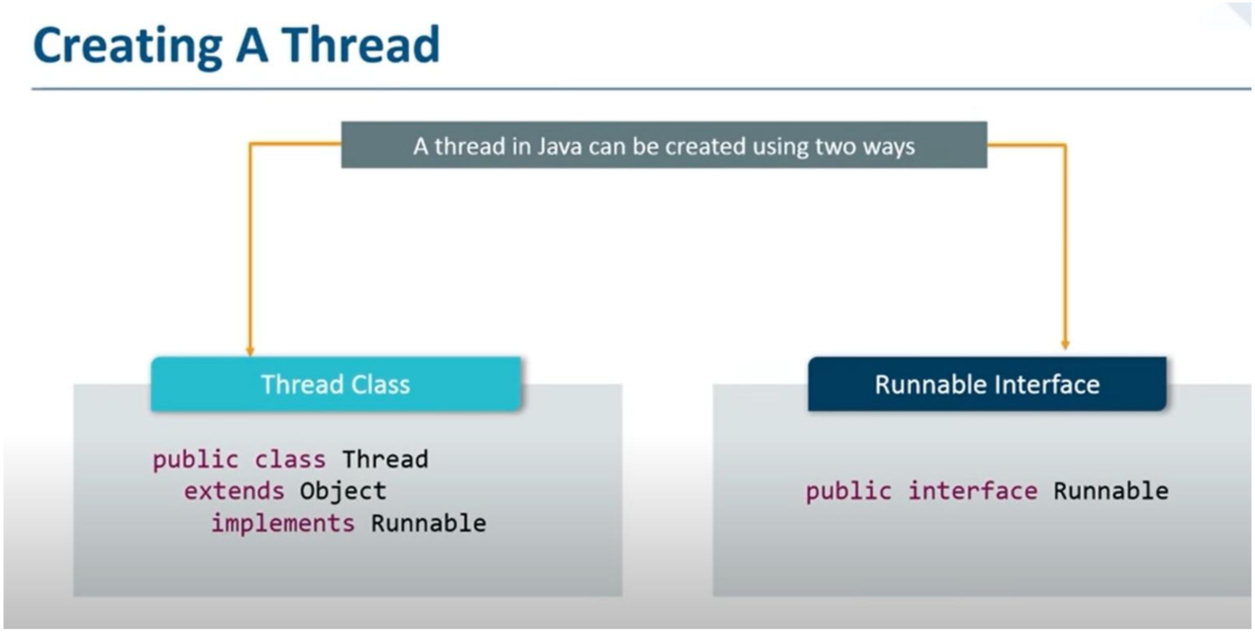
In this state a thread starts executing by entering run() method and the yield() method can send them to go back to the Runnable state.

Waiting

A thread enters this state when it is temporarily in an inactive state i.e it is still alive but is not eligible to run. It is can be in waiting, sleeping or blocked state.

Terminated

A runnable thread enters the terminated state when it completes its task o otherwise terminates.



Creating A Thread

A thread in Java can be created using two ways

Thread Class

```
public class Thread
  extends Object
    implements Runnable
```

Runnable Interface

```
public interface Runnable
```

The Java Thread Model-Why use Threads in Java?

The Java run-time system depends on threads for many things. Threads reduce inefficiency by preventing the waste of CPU cycles.

Threads exist in several states. Following are those states:

- New – When we create an instance of Thread class, a thread is in a new state.
- Runnable – The Java thread is in running state.
- Suspended – A running thread can be suspended, which temporarily suspends its activity. A suspended thread can then be resumed, allowing it to pick up where it left off.
- Blocked – A java thread can be blocked when waiting for a resource.
- Terminated – A thread can be terminated, which halts its execution immediately at any given time. Once a thread is terminated, it cannot be resumed.

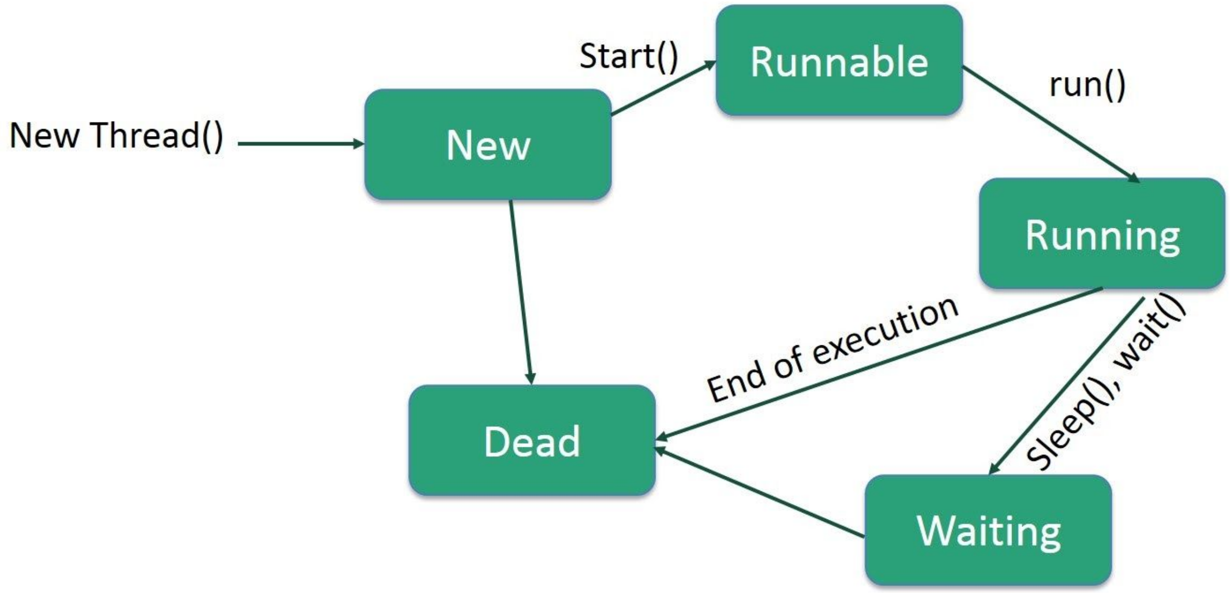
Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle −

- New − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- Runnable − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- Waiting − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- Timed Waiting − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- Terminated (Dead) − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Priorities

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

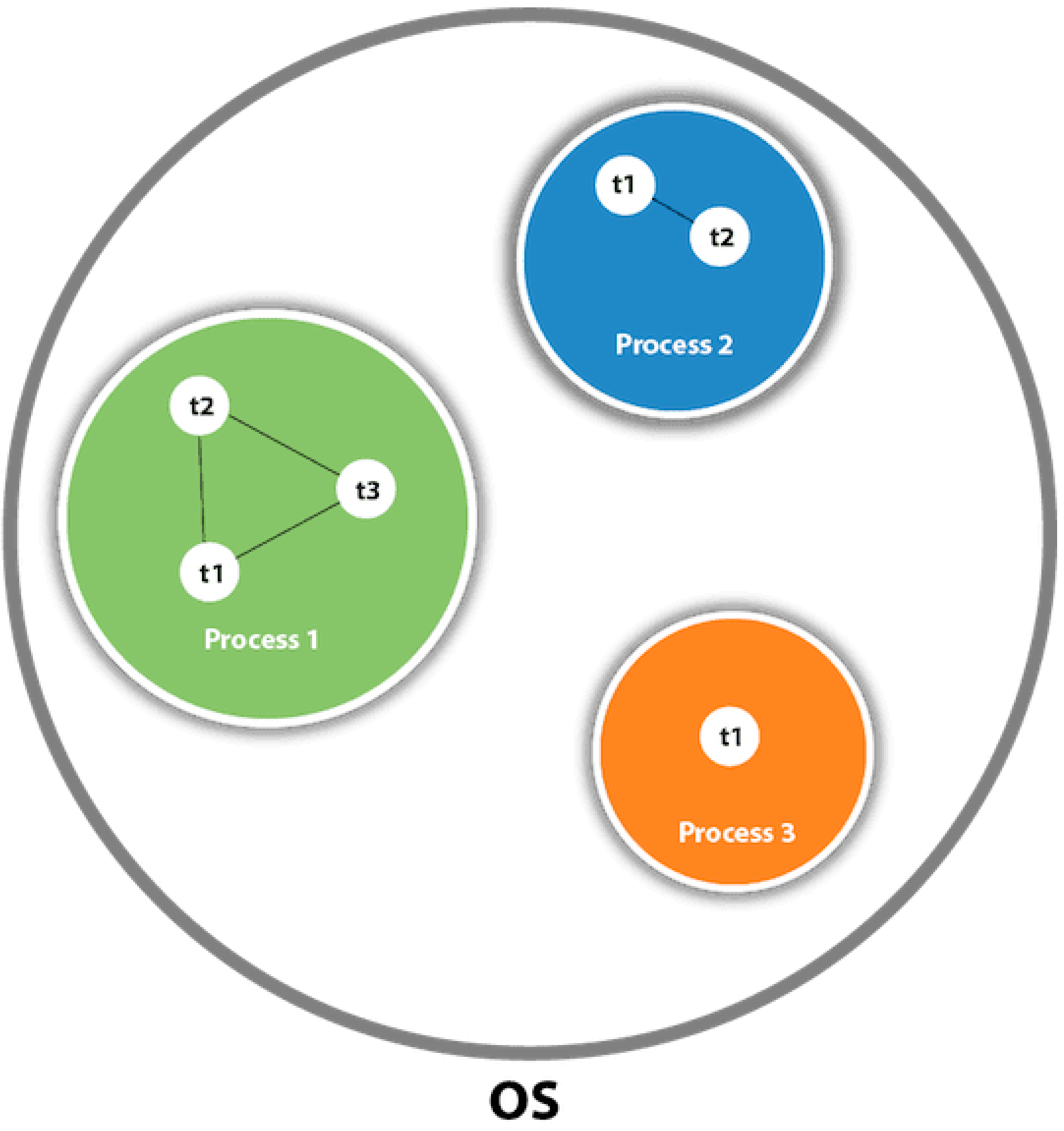
- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

> Note: At least one process is required for each thread.

What is Thread in java

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

> Note: At a time one thread is executed only.

Java Thread class

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

How to create thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class:

We have used few of these methods in the example below.

- getName(): It is used for Obtaining a thread's name
- getPriority(): Obtain a thread's priority
- isAlive(): Determine if a thread is still running
- join(): Wait for a thread to terminate
- run(): Entry point for the thread
- sleep(): suspend a thread for a period of time
- start(): start a thread by calling its run() method

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

---

1) Java Thread Example by extending Thread class

```java
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output:thread is running...

---

2) Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

Output:thread is running...

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitly create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

You can also create an anonymous subclass of Thread like this:

```
Thread thread = new Thread(){
   public void run(){
     System.out.println("Thread Running");
   }
 }
 thread.start();
```

This example will print out the text "Thread running" once the run() method is executed by the new thread.

Main Thread :

```
class MyTask{
    void executeTask(){
        for(int doc=1;doc<=10;doc++)
        {
            System.out.println("@@Printing documents #" +doc+ " -Printer2");
        }
    }
}


public class HelloWorld{

     public static void main(String []args){

         //Job-1
        System.out.println("===Application Started===");
        //job-2
        MyTask task= new MyTask();
        task.executeTask();

        //Job-3
        for(int doc=1;doc<=10;doc++)
        {
            System.out.println("^^Printing documents #" +doc+ " -Printer1");
        }
        //job-4
         System.out.println("===Application Finished===");
    }
}
```

Output:

```
===Application Started===
@@Printing documents #1 -Printer2
@@Printing documents #2 -Printer2
@@Printing documents #3 -Printer2
@@Printing documents #4 -Printer2
@@Printing documents #5 -Printer2
@@Printing documents #6 -Printer2
@@Printing documents #7 -Printer2
@@Printing documents #8 -Printer2
@@Printing documents #9 -Printer2
@@Printing documents #10 -Printer2
^^Printing documents #1 -Printer1
^^Printing documents #2 -Printer1
^^Printing documents #3 -Printer1
^^Printing documents #4 -Printer1
^^Printing documents #5 -Printer1
^^Printing documents #6 -Printer1
^^Printing documents #7 -Printer1
^^Printing documents #8 -Printer1
^^Printing documents #9 -Printer1
^^Printing documents #10 -Printer1
 ===Application Finished===
```

Case 2: With thread:

```
class MyTask extends Thread{

         public void run(){
```

```
        for(int doc=1;doc<=10;doc++)
        {
            System.out.println("@@Printing documents #" +doc+ " -Printer2");
        }
    }

}

public class HelloWorld{

     public static void main(String []args){

         //Job-1
        System.out.println("===Application Started===");

        //job-2
        MyTask task= new MyTask();
       Thread t1=new Thread(task);
        //task.executeTask();
        task.start();

        //Job-3
        for(int doc=1;doc<=10;doc++)
        {
            System.out.println("^^Printing documents #" +doc+ " -Printer1");
        }
        //job-4
         System.out.println("===Application Finished===");
     }
}
```

Output-1;

```
===Application Started===
^^Printing documents #1 -Printer1
^^Printing documents #2 -Printer1
^^Printing documents #3 -Printer1
^^Printing documents #4 -Printer1
^^Printing documents #5 -Printer1
```

```
@@Printing documents #1 -Printer2
@@Printing documents #2 -Printer2
@@Printing documents #3 -Printer2
@@Printing documents #4 -Printer2
@@Printing documents #5 -Printer2
@@Printing documents #6 -Printer2
@@Printing documents #7 -Printer2
@@Printing documents #8 -Printer2
@@Printing documents #9 -Printer2
@@Printing documents #10 -Printer2
^^Printing documents #6 -Printer1
^^Printing documents #7 -Printer1
^^Printing documents #8 -Printer1
^^Printing documents #9 -Printer1
^^Printing documents #10 -Printer1
===Application Finished===
```

Output-2:

```
===Application Started===
^^Printing documents #1 -Printer1
^^Printing documents #2 -Printer1
^^Printing documents #3 -Printer1
^^Printing documents #4 -Printer1
^^Printing documents #5 -Printer1
^^Printing documents #6 -Printer1
^^Printing documents #7 -Printer1
^^Printing documents #8 -Printer1
^^Printing documents #9 -Printer1
^^Printing documents #10 -Printer1
===Application Finished===
@@Printing documents #1 -Printer2
@@Printing documents #2 -Printer2
@@Printing documents #3 -Printer2
@@Printing documents #4 -Printer2
@@Printing documents #5 -Printer2
@@Printing documents #6 -Printer2
@@Printing documents #7 -Printer2
@@Printing documents #8 -Printer2
@@Printing documents #9 -Printer2
```

```
@@Printing documents #10 -Printer2
```

Output-3:

```
===Application Started===
^^Printing documents #1 -Printer1
^^Printing documents #2 -Printer1
^^Printing documents #3 -Printer1
@@Printing documents #1 -Printer2
^^Printing documents #4 -Printer1
^^Printing documents #5 -Printer1
@@Printing documents #2 -Printer2
@@Printing documents #3 -Printer2
@@Printing documents #4 -Printer2
@@Printing documents #5 -Printer2
@@Printing documents #6 -Printer2
^^Printing documents #6 -Printer1
^^Printing documents #7 -Printer1
@@Printing documents #7 -Printer2
^^Printing documents #8 -Printer1
^^Printing documents #9 -Printer1
@@Printing documents #8 -Printer2
^^Printing documents #10 -Printer1
===Application Finished===
@@Printing documents #9 -Printer2
@@Printing documents #10 -Printer2
```

Through Runnable:

```
class MyTask implements Runnable{

        public void run(){

       for(int doc=1;doc<=10;doc++)
       {
           System.out.println("@@Printing documents #" +doc+ " -Printer2");
       }
   }

}
```

```
public class HelloWorld{

     public static void main(String []args){

         //Job-1
        System.out.println("===Application Started===");

        //job-2
        Runnable obj=new MyTask();
        Thread t1=new Thread(obj);
        //MyTask task= new MyTask();
        //task.executeTask();
        //task.start();
        t1.start();

        //Job-3
        for(int doc=1;doc<=10;doc++)
        {
            System.out.println("^^Printing documents #" +doc+ " -Printer1");
        }
        //job-4
         System.out.println("===Application Finished===");
     }
}
```

Example 2:

```
package threads;

class Hi extends Thread
{
    public void run()
    {
        for(int i=1;i<=10;i++)
        System.out.println("Hi");
        try {Thread.sleep(1000);} catch(Exception e) {}
    }
}
```

```
class Hello extends Thread
{
    public void run()
    {
        for(int i=1;i<=10;i++)
        System.out.println("Hello");
        try {Thread.sleep(1000);} catch(Exception e) {}
    }
}
public class ThreadDemo
{
    public static void main(String[] args)
    {
        Hi ob1= new Hi();
        Hello ob2= new Hello();
        ob1.start();
        ob2.start();
    }
}
```

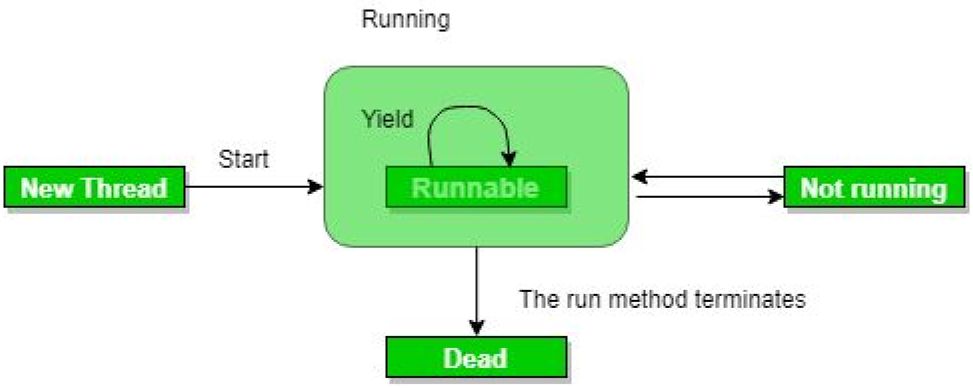
Output:

```
Hi
Hi
Hi
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hi
Hi
Hi
Hi
Hi
Hi
```

Hi

Using Thread Methods:

1. **yield():** Suppose there are three threads t1, t2, and t3. Thread t1 gets the processor and starts its execution and thread t2 and t3 are in Ready/Runnable state. Completion time for thread t1 is 5 hour and completion time for t2 is 5 minutes. Since t1 will complete its execution after 5 hours, t2 has to wait for 5 hours to just finish 5 minutes job. In such scenarios where one thread is taking too much time to complete its execution, we need a way to prevent execution of a thread in between if something important is pending. yeild() helps us in doing so.
   yield() basically means that the thread is not doing anything particularly important and if any other threads or processes need to be run, they should run. Otherwise, the current thread will continue to run.



Use of yield method:

- Whenever a thread calls java.lang.Thread.yield method, it gives hint to the thread scheduler that it is ready to **pause its execution.** Thread scheduler is free to ignore this hint.
- **If any thread executes yield method , thread scheduler checks if there is any thread with same or high priority than this thread. If processor finds any thread with higher or same priority then it will move the current thread to Ready/Runnable state and give processor to other thread and if not – current thread will keep executing.**

Note:

- Once a thread has executed yield method and there are many threads with same priority is waiting for processor, then we can't specify which thread will get execution chance first.
- The thread which executes the yield method will enter in the **Runnable state from Running state.**
- Once a thread pauses its execution, we can't specify when it will get chance again it depends on thread scheduler.
- Underlying platform must provide support for preemptive scheduling if we are using yield method.

2. **sleep():** This method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

Note:

a. Based on the requirement we can make a thread to be in sleeping state for a specified period of time
b. Sleep() causes the thread to definitely **stop executing for a given amount of time;** if no other thread or process needs to be run, the CPU will be idle (and probably enter a power saving mode).

yield() vs sleep()

he major difference between yield and sleep in Java **is that yield() method pauses the currently executing thread temporarily for giving a chance to the remaining waiting threads of the same priority to execute. If there is no waiting thread or all the waiting threads have a lower priority then the same thread will continue its execution.**

yield:() indicates that the thread is not doing anything particularly important and if any other threads or processes need to be run, they can. Otherwise, the **current thread will continue to**

run.

sleep(): causes the thread to definitely stop executing for a given amount of time; if no other thread or process needs to be run, the CPU will be idle (and probably enter a power saving mode).

3. **join():** The join() method of a Thread instance is used to join the start of a thread's execution to end of other thread's execution such that a **thread does not start running until another thread ends**. If join() is called on a Thread instance, the currently running thread will block until the Thread instance has finished executing.
   The join() method waits at most this much milliseconds for this thread to die. A timeout of 0 means to wait forever

Note:

- If any executing thread t1 calls join() on t2 i.e; t2.join() immediately t1 will enter into waiting state until t2 completes its execution.
- Giving a timeout within join(), will make the join() effect to be nullified after the specific timeout.

4. The **stop()** method of thread class terminates the thread execution. Once a thread is stopped, it cannot be restarted by start() method.

Program:

```
class A extends Thread
{
   public void run()
   {
      for (int i=1;i<=5;i++)
      {
         if(i==1) yield();
         System.out.println("\t From thread A: i= " +i);
      }
      System.out.println("exit from A ");
   }
}
```

```
class B extends Thread
{
    public void run()
    {
        for (int j=1;j<=5;j++)
        {
            System.out.println("\t From thread B: j= " +j);
            if(j==3) stop();
        }
        System.out.println("exit from B");
    }
}
class C extends Thread
{
    public void run()
    {
        for (int k=1;k<=5;k++)
        {
            System.out.println("\t From thread C: k= " +k);
            if(k==1)
            {
                try
                {
                    sleep(1000);
                }
                catch(Exception e)
                {

                }
            }
        }
        System.out.println("exit from C");
    }
}

public class ThreadMethods
{
    public static void main(String args[])
    {
        A threadA =new A();
        B threadB =new B();
        C threadC =new C();
        System.out.println("start thread A");
        threadA.start();
        System.out.println("start thread B");
        threadB.start();
        System.out.println("start thread C");
        threadC.start();
        System.out.println("end of main thread");

    }
}
```

Output:

```
start thread A
start thread B
            From thread A: i= 1
            From thread A: i= 2
            From thread A: i= 3
            From thread A: i= 4
            From thread A: i= 5
exit from A
start thread C
            From thread B: j= 1
            From thread B: j= 2
            From thread B: j= 3
end of main thread
            From thread C: k= 1
            From thread C: k= 2
            From thread C: k= 3
            From thread C: k= 4
            From thread C: k= 5
 exit from C
```

Thread Priorities:

```
class TA extends Thread
{
    public void run()
    {
        System.out.println("\tthread A started ");
        for (int i=1;i<=5;i++)
        {
            System.out.println("\t From thread A: i= " +i);
        }
        System.out.println("exit from A ");
    }
}
class TB extends Thread
{
    public void run()
    {
        System.out.println("\tthread B started ");
        for (int j=1;j<=5;j++)
        {
            System.out.println("\t From thread B: j= " +j);
        }
        System.out.println("exit from B");
    }
}
class TC extends Thread
{
    public void run()
    {
        System.out.println("\tthread C started ");
        for (int k=1;k<=5;k++)
        {
            System.out.println("\t From thread C: k= " +k);

        }
        System.out.println("exit from C");
    }
}

public class ThreadPriority
{
    public static void main(String args[])
    {
        TA threadA =new TA();
        TB threadB =new TB();
        TC threadC =new TC();
        threadC.setPriority(Thread.MAX_PRIORITY);
        threadB.setPriority(threadA.getPriority()+1);
        threadA.setPriority(Thread.MIN_PRIORITY);

        System.out.println("start thread A");
        threadA.start();
        System.out.println("start thread B");
        threadB.start();
        System.out.println("start thread C");
        threadC.start();
        System.out.println("end of main thread");

    }
}
```

Output:

```
start thread A
start thread B
start thread C
end of main thread
          thread C started
           From thread C: k= 1
           From thread C: k= 2
           From thread C: k= 3
           From thread C: k= 4
           From thread C: k= 5
exit from C
          thread B started
           From thread B: j= 1
           From thread B: j= 2
           From thread B: j= 3
           From thread B: j= 4
           From thread B: j= 5
exit from B
          thread A started
           From thread A: i= 1
           From thread A: i= 2
           From thread A: i= 3
           From thread A: i= 4
           From thread A: i= 5
exit from A
```

Synchronization in Java

Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

If you declare any method as synchronized, it is known as **synchronized method.**

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Program Without Synchronization

```
package threads;

class One

{

    public void increment()

    {
}
        int count=0;
        for(int i=1;i<10;i++)
        {
            count++;
            System.out.println("count  is =" + count);
        }
    }
   }
class Two extends Thread
{
    One o;
    Two(One o)
    {
        this.o=o;
    }
    public void run()
    {
        o.increment();
    }
}
```

```
public class SynDemo
{
    public static void main(String[] args)
    {
        One obj =new One();
        Two tw1 =new Two(obj);
        Two tw2 =new Two(obj);
        tw1.start();
        tw2.start();
    }
}
```

Output:

```
count is =1
count is =1
count is =2
count is =3
count is =2
count is =4
count is =3
count is =5
count is =4

count is =6

count is =7

count is =8

count is =9

count is =5

count is =6

count is =7

count is =8

count is =9
```

With Synchronization;

```
package threads;

class AA

{

    public synchronized void increment()

    {

        int count=0;

        for(int i=1;i<10;i++)

        {
            count++;
            System.out.println("count  is =" + count);
        }
    }

}
class BB extends Thread
{
    AA o;
    BB(AA o)
    {
        this.o=o;
    }
    public void run()
    {
        o.increment();
    }
}
public class WithSynch
{
    public static void main(String[] args)
    {
        AA obj =new AA();
        BB tw1 =new BB(obj);
        BB tw2 =new BB(obj);
        tw1.start();
        tw2.start();
    }
}
```

Output:

```
count  is =1
count  is =2
count  is =3
count  is =4
count  is =5
count  is =6
count  is =7
count  is =8
count  is =9
count  is =1

count  is =2

count  is =3

count  is =4

count  is =5

count  is =6

count  is =7

count  is =8

count  is =9
```

Synchronized block:

```
// A Java program to demonstrate working of  synchronized block

import java.io.*;

import java.util.*;

// A Class used to send a message

class Sender

{

        public void send(String msg)

        {
```

```
        System.out.println("Sending\t" + msg );
        try
        {
            Thread.sleep(1000);
        }
        catch (Exception e)
        {
            System.out.println("Thread interrupted.");
        }
        System.out.println("\n" + msg + "Sent");
    }
}
// Class for send a message using Threads
class ThreadedSend extends Thread
{
    private String msg;
    Sender sender;
    // Receives a message object and a string message to be sent
    ThreadedSend(String m, Sender obj)
    {
        msg = m;
        sender = obj;
    }
    public void run()
    {
        // Only one thread can send a message at a time.
        synchronized(sender)
        {
            // synchronizing the snd object
            sender.send(msg);
        }
    }
}
// Driver class
public class SyncDemo
{
    public static void main(String args[])
    {
        Sender snd = new Sender();
        ThreadedSend S1 = new ThreadedSend( " Hi " , snd );
        ThreadedSend S2 = new ThreadedSend( " Bye " , snd );

        // Start two threads of ThreadedSend type

        S1.start();

        S2.start();

        // wait for threads to end

        try

        {

            S1.join();

            S2.join();

        }

        catch(Exception e)

        {

            System.out.println("Interrupted");

        }

    }

}
```

Output:

```
Sending         Hi

 Hi Sent

Sending         Bye


  Bye Sent
```

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

```java
class Table{
void printTable(int n){//method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }
 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
```

```java
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

```
Output: 5
       100
       10
       200
       15
       300
       20
       400
       25
       500
```

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```java
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

```

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

```
Output: 5
       10
       15
       20
       25
       100
       200
       300
       400
       500
```

Java Thread isAlive() method

The **isAlive()** method of thread class tests if the thread is alive. A thread is considered alive when the start() method of thread class has been called and the thread is not yet dead. This method returns true if the thread is still running and not finished.

Syntax

```
public final boolean isAlive()
```

Return

This method will return true if the thread is alive otherwise returns false.

Example

```
public class JavaIsAliveExp extends Thread
{
    public void run()
    {
        try
        {
            Thread.sleep(300);
            System.out.println("is run() method isAlive "+Thread.currentThread().isAlive());
        }
        catch (InterruptedException ie) {
        }
    }
    public static void main(String[] args)
    {
        JavaIsAliveExp t1 = new JavaIsAliveExp();
        System.out.println("before starting thread isAlive: "+t1.isAlive());  //false
        t1.start();
        System.out.println("after starting thread isAlive: "+t1.isAlive()); //true
    }
}
```

Output:

```
before starting thread isAlive: false
after starting thread isAlive: true
is run() method isAlive true
```

Inter-thread communication in Java

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait(): Running to Suspended
- notify() : Suspended to Running: For one thread
- notifyAll()

---

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed. (Running -> Suspended state)

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	**Description**
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

---

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

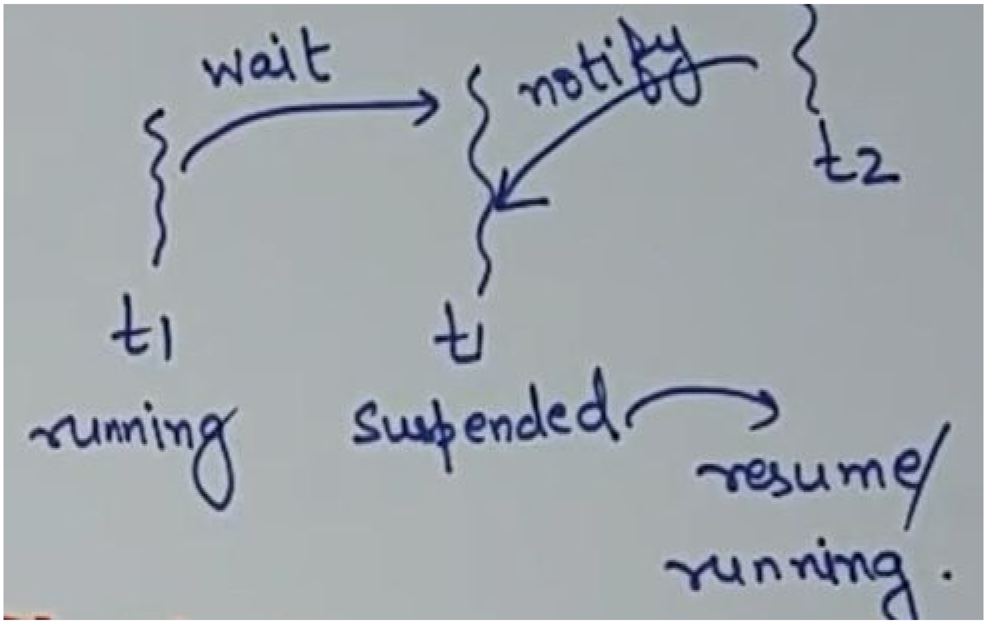
public final void notify()

(Suspended -> Running/Resume)

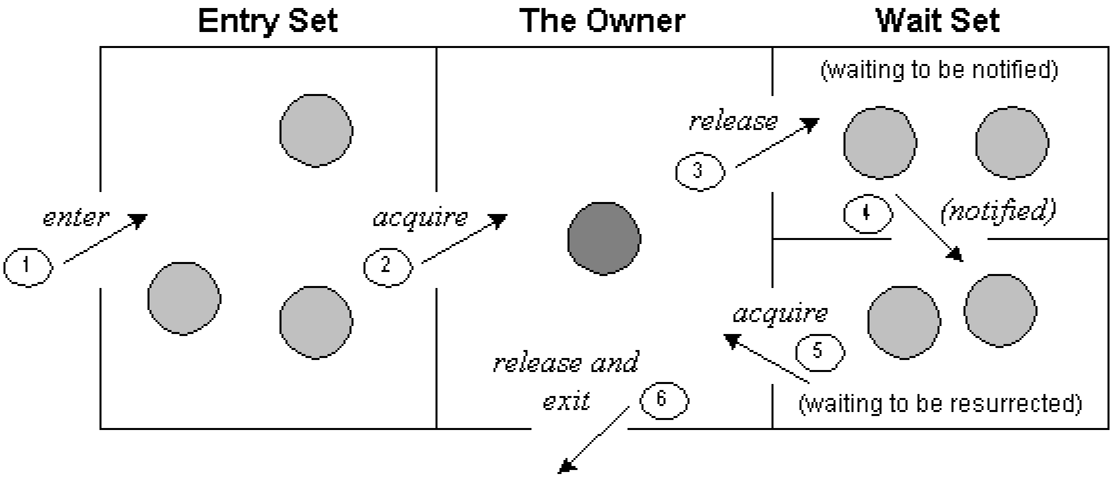
3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()



Understanding the process of inter-thread communication



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

Difference between wait and sleep?

Let's see the important differences between wait and sleep methods.

wait()	**sleep()**
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	is the method of Thread class
is the non-static method	is the static method
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

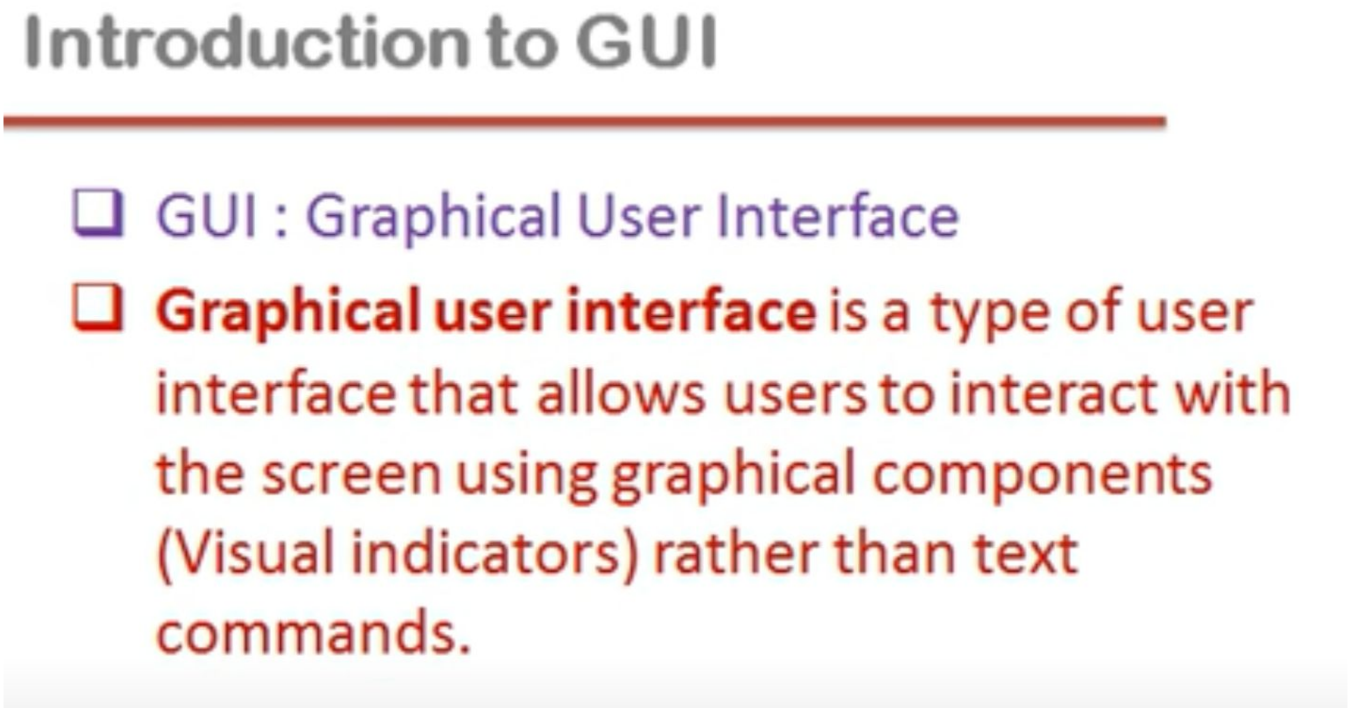
Example of inter thread communication in java

Let's see the simple example of inter thread communication.

```
class Customer{
int amount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){  //10000<15000
System.out.println("Less balance; waiting for deposit..." );
try{wait();}catch(Exception e){}
}
this.amount-=amount; //20000-15000
System.out.println("...amount= " + this.amount);//5000
System.out.println("withdraw completed...");
}
synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;  //10000+10000
System.out.println("deposit completed...amount= " + this.amount);  //20000
notify();
}
}
public class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread()
{
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();
}}
```

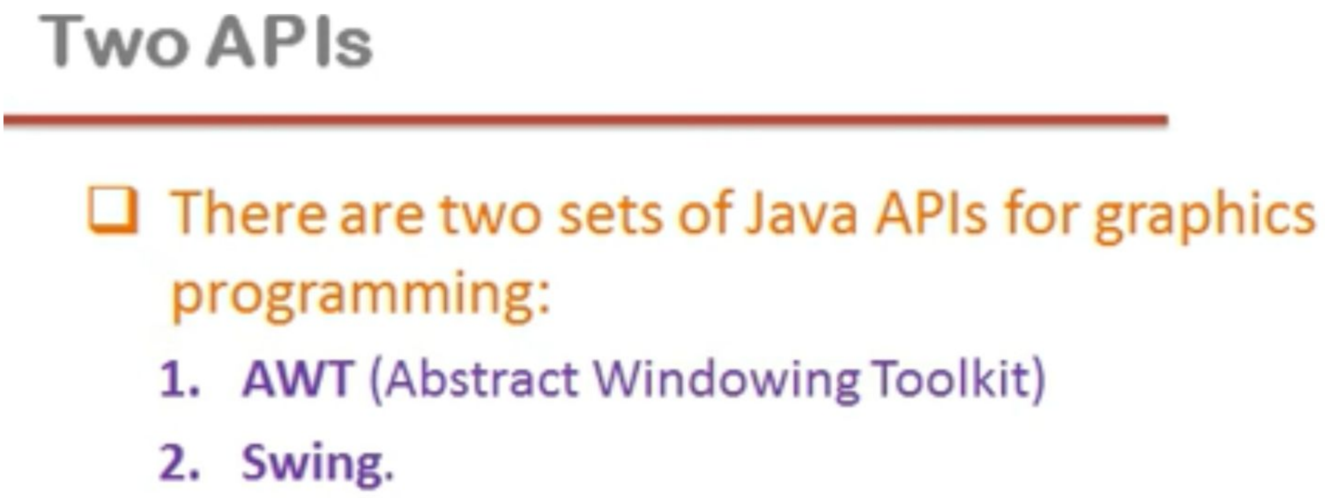
```
Output: going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...amount= 20000
...amount= 5000
withdraw completed...
```

JAVA GUI



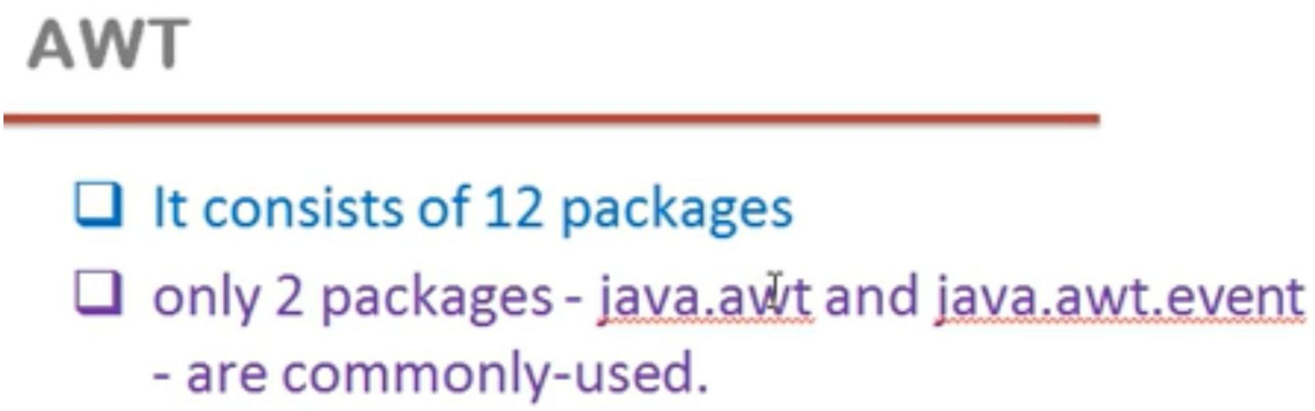
Introduction to GUI

- □ GUI : Graphical User Interface
- □ **Graphical user interface** is a type of user interface that allows users to interact with the screen using graphical components (Visual indicators) rather than text commands.



Two APIs

- □ There are two sets of Java APIs for graphics programming:
  1. **AWT** (Abstract Windowing Toolkit)
  2. **Swing.**



AWT

- ❑ It consists of 12 packages
- ❑ only 2 packages - java.awt and java.awt.event - are commonly-used.

Java AWT Tutorial

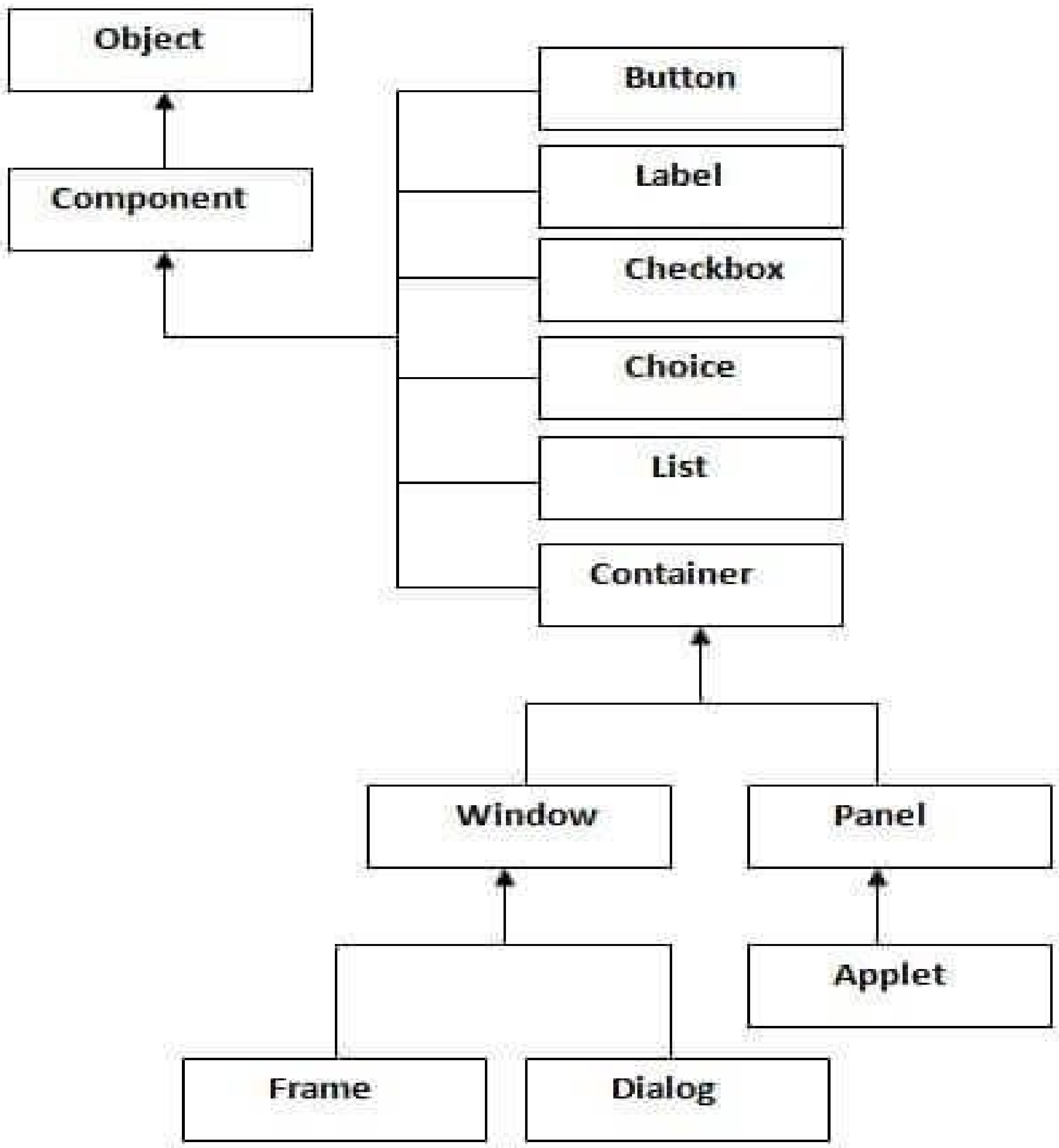
Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

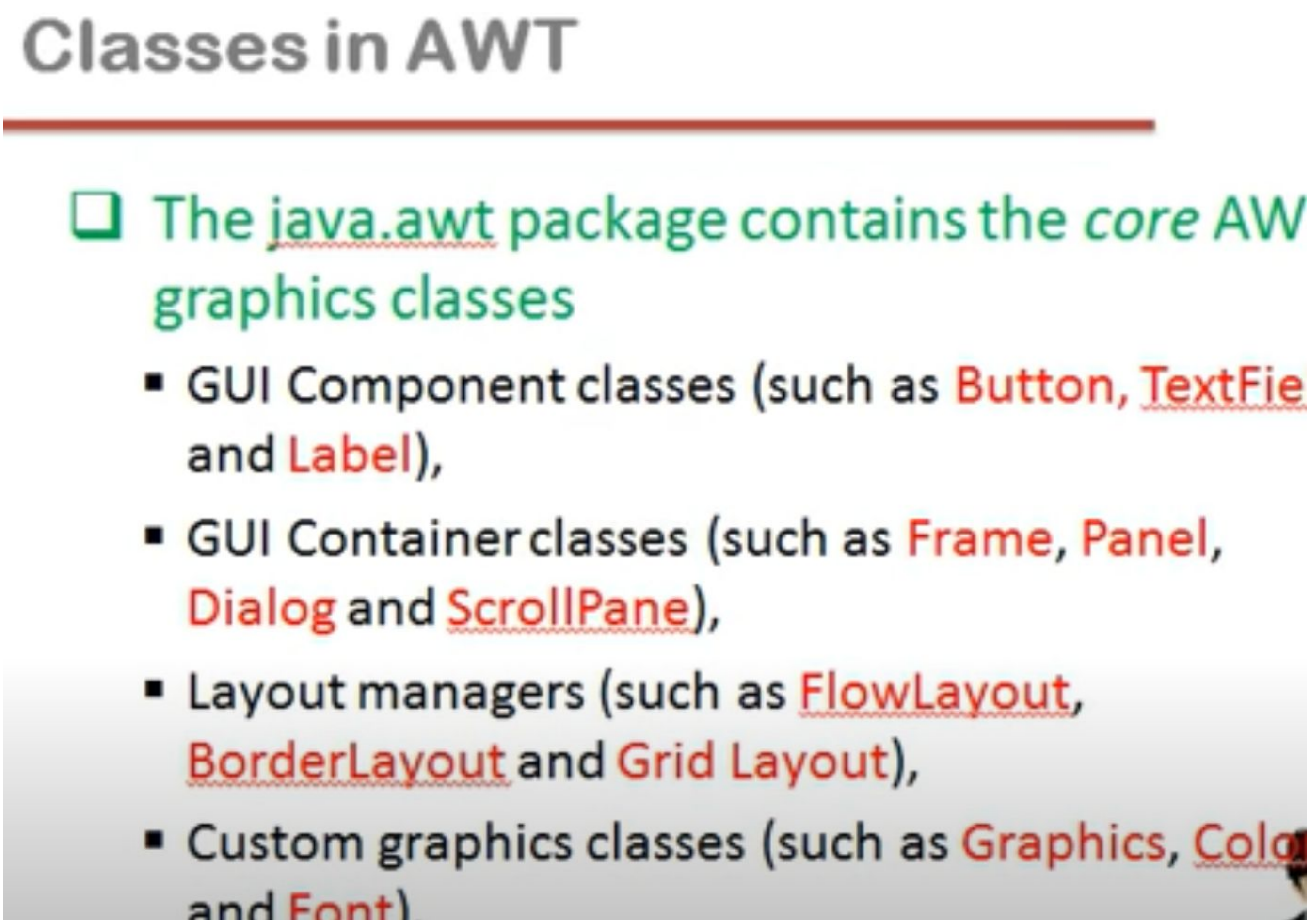
Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as **TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.**

Java AWT Hierarchy

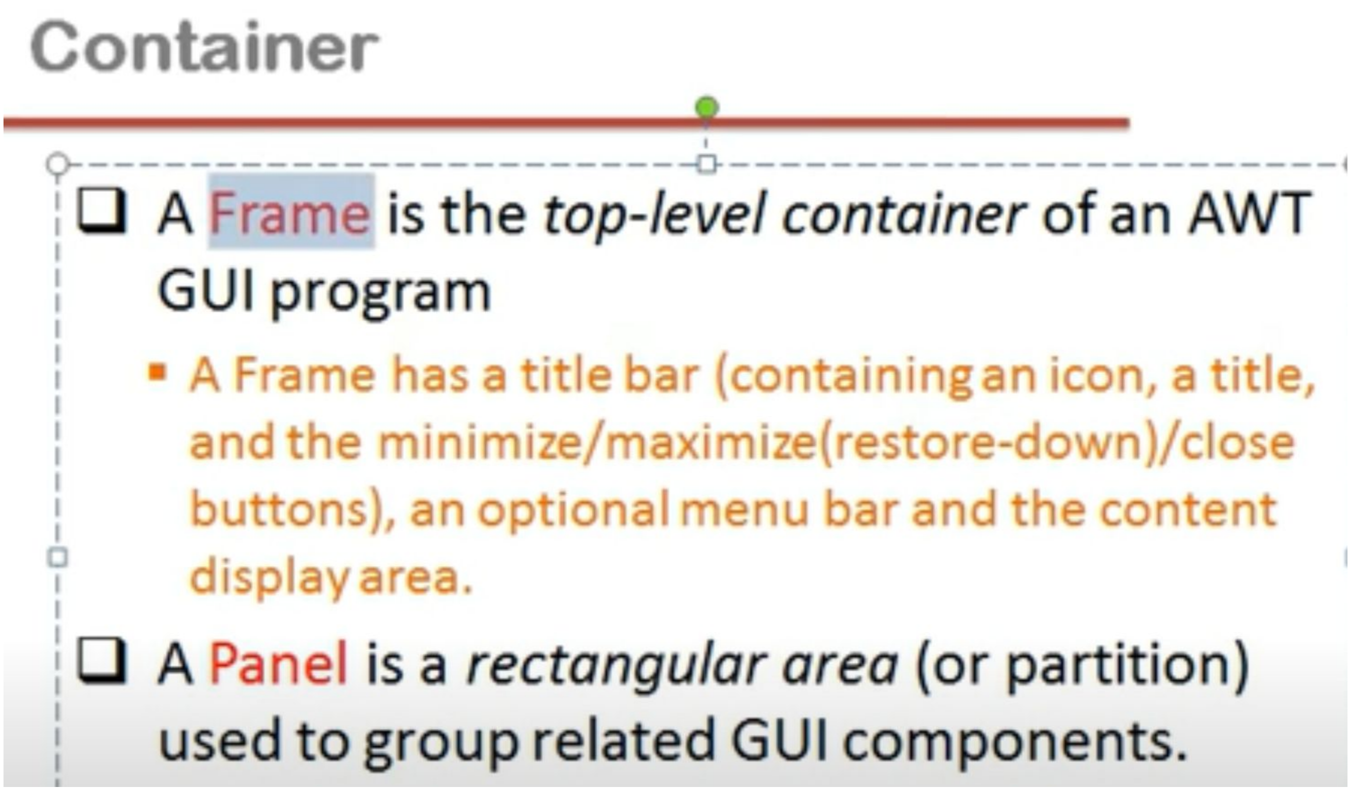
The hierarchy of Java AWT classes are given below.


Object
Component
Button
Label
Checkbox
Choice
List
Container
Window
Panel
Applet
Frame
Dialog



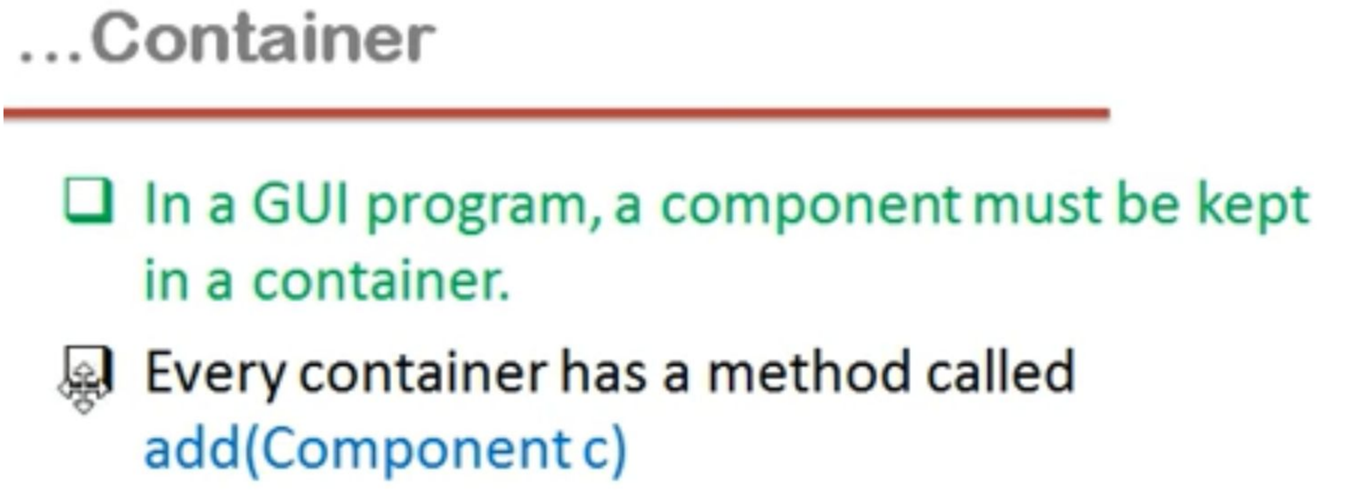
Classes in AWT

- ❑ The java.awt package contains the *core* AW graphics classes
  - GUI Component classes (such as Button, TextFie and Label),
  - GUI Container classes (such as Frame, Panel, Dialog and ScrollPane),
  - Layout managers (such as FlowLayout, BorderLayout and Grid Layout),
  - Custom graphics classes (such as Graphics, Colo and Font)



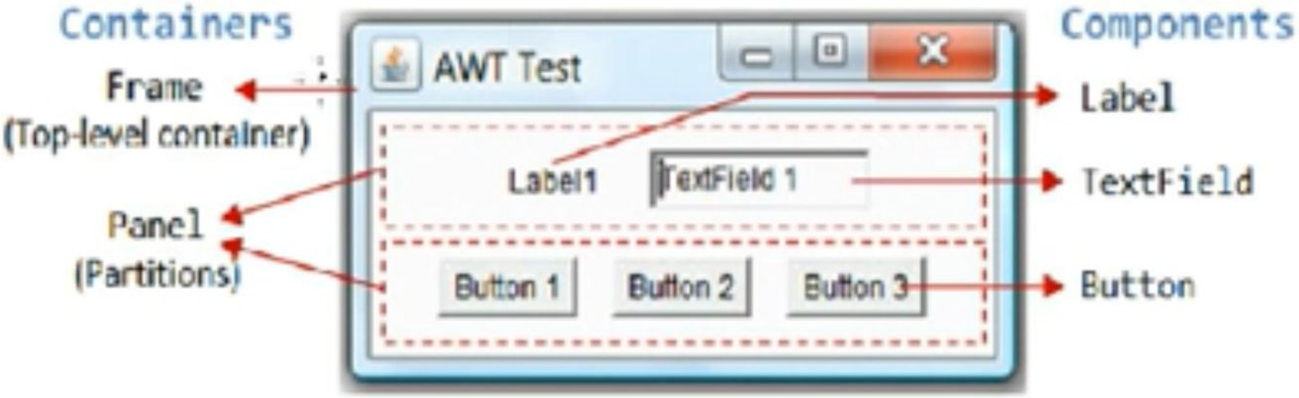
Container

- A Frame is the *top-level container* of an AWT GUI program
  - A Frame has a title bar (containing an icon, a title, and the minimize/maximize(restore-down)/close buttons), an optional menu bar and the content display area.
- A Panel is a *rectangular area* (or partition) used to group related GUI components.



...Container

- In a GUI program, a component must be kept in a container.
- Every container has a method called add(Component c)



Useful Methods of Component class

Method	**Description**
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

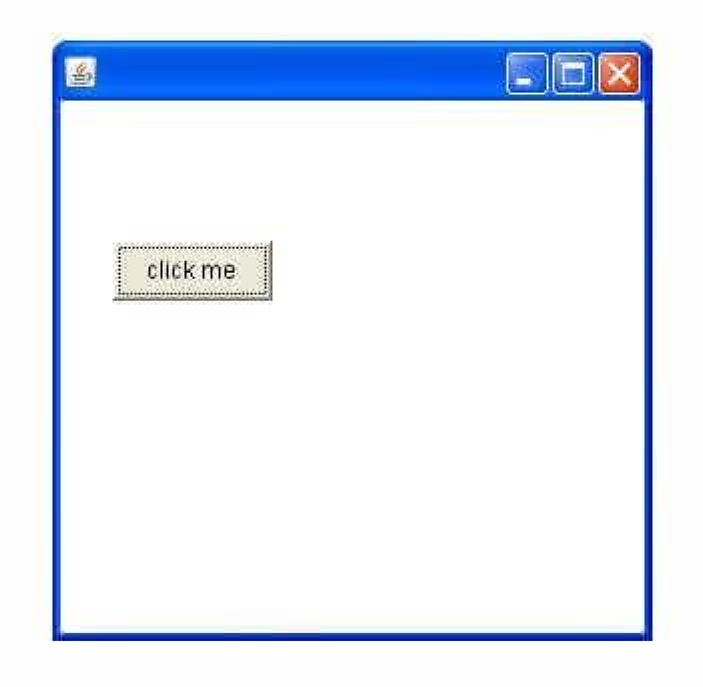
- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.



AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
14.}}
```

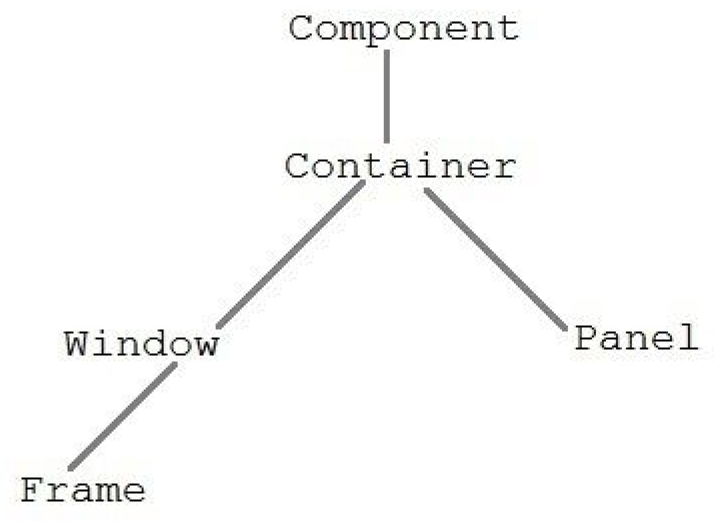
Java Abstract Window Toolkit(AWT)

Java AWT is an API that contains large number of classes and methods to create and manage graphical user interface (GUI) applications. The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms. The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform. This is an advantage of using AWT. But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform that means AWT component are platform dependent.

AWT is the foundation upon which Swing is made i.e Swing is a improved GUI API that extends the AWT. But now a days AWT is merely used because most GUI Java programs are implemented using Swing because of its rich implementation of GUI controls and light-weighted nature.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below, all the classes are available in **java.awt** package.



Component class

Component class is at the top of AWT hierarchy. It is an abstract class that encapsulates all the attributes of visual component. A component object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container

Container is a component in AWT that contains another component like button, text field, tables etc. **Container** is a subclass of component class. **Container** class keeps track of components that are added to another component.

Panel

Panel class is a concrete subclass of **Container**. Panel does not contain title bar, menu bar or border. It is container that is used for holding components.

Window class

Window class creates a top level window. Window does not have borders and menubar.

Frame

Frame is a subclass of **Window** and have resizing canvas. It is a container that contain several different components like button, title bar, textfield, label etc. In Java, most of the AWT applications are created using **Frame** window. Frame class has two different constructors,

```
Frame() throws HeadlessException
```

```
Frame(String title) throws HeadlessException
```

Creating a Frame

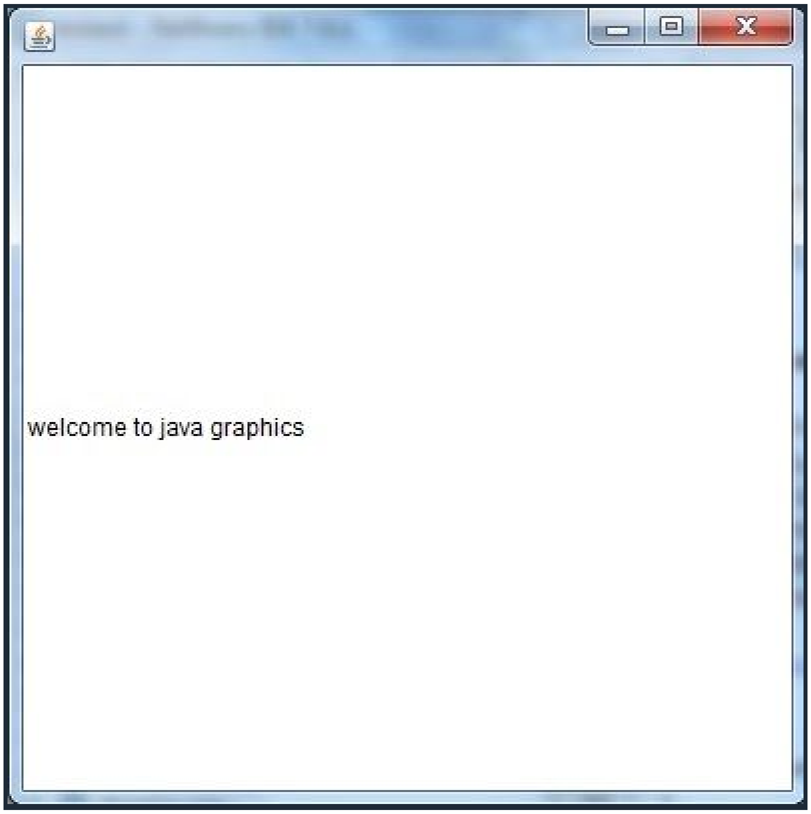
There are two ways to create a Frame. They are,

1. By Instantiating Frame class

2. By extending Frame class

Creating Frame Window by Instantiating Frame class

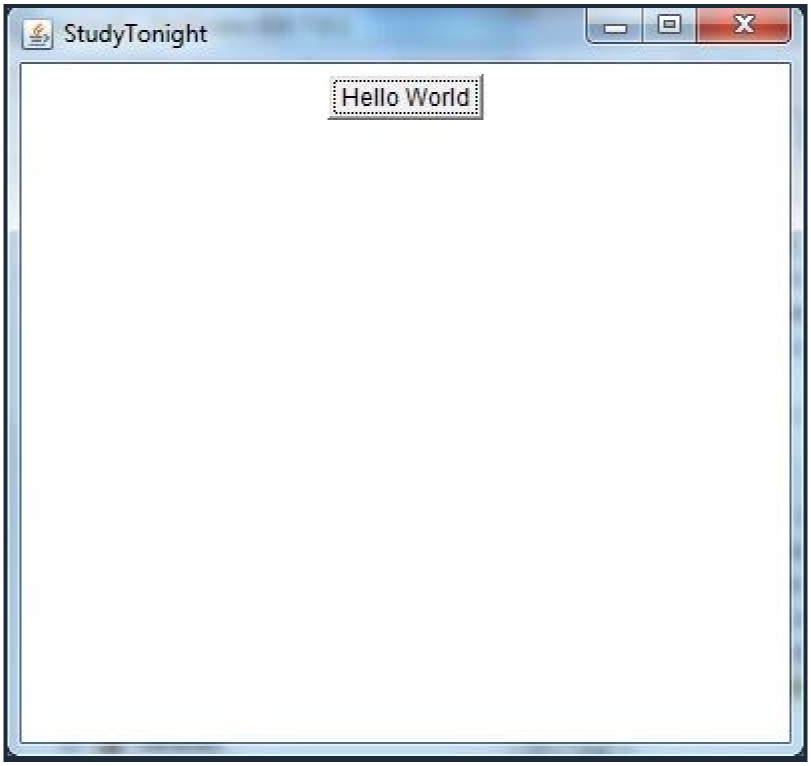
```
import java.awt.*;
public class Testawt
{
  Testawt()
  {
    Frame fm=new Frame();    //Creating a frame
    Label lb = new Label("welcome to java graphics");   //Creating a label
    fm.add(lb);                  //adding label to the frame
    fm.setSize(300, 300);   //setting frame size.
    fm.setVisible(true);     //set frame visibilty true
  }
  public static void main(String args[])
  {
    new Testawt();
  }
}
```



welcome to java graphics

Creating Frame window by extending Frame class

```
import java.awt.*;
import java.awt.event.*;

public class Testawt1 extends Frame
{
   public Testawt1()
   {
      Button btn=new Button("Click Me");
      add(btn);            //adding a new Button.
      setSize(400, 500);        //setting size.
      setTitle("Adding Button");  //setting title.
      setLayout(new FlowLayout());          //set default layout for frame.
      setVisible(true);           //set frame visibilty true.
   }

   public static void main (String[] args)
   {
      new Testawt1();   //creating a frame.
   }

}
```



Points to Remember:

1. While creating a frame (either by instantiating or extending Frame class), Following two attributes are must for visibility of the frame:
   - **setSize(int width, int height);**
   - **setVisible(true);**
2. When you create other components like Buttons, TextFields, etc. Then you need to add it to the frame by using the method - **add(Component's Object);**
3. You can add the following method also for resizing the frame - **setResizable(true);**

AWT Button

In Java, AWT contains a Button Class. It is used for creating a labelled button which can perform an action.

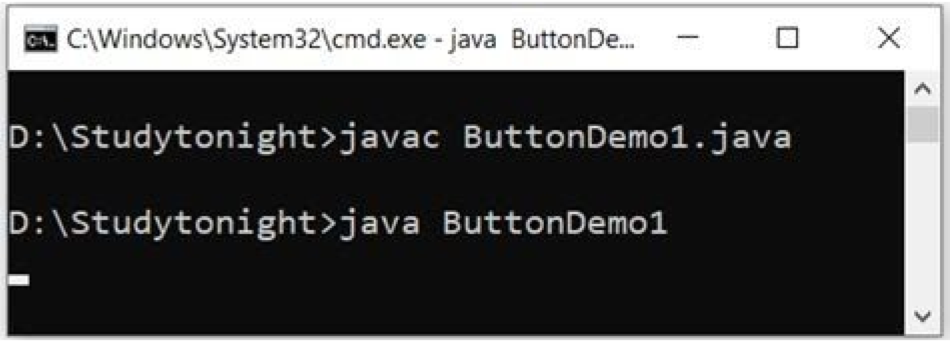
AWT Button Class Declaration:

public class Button extends Component implements Accessible

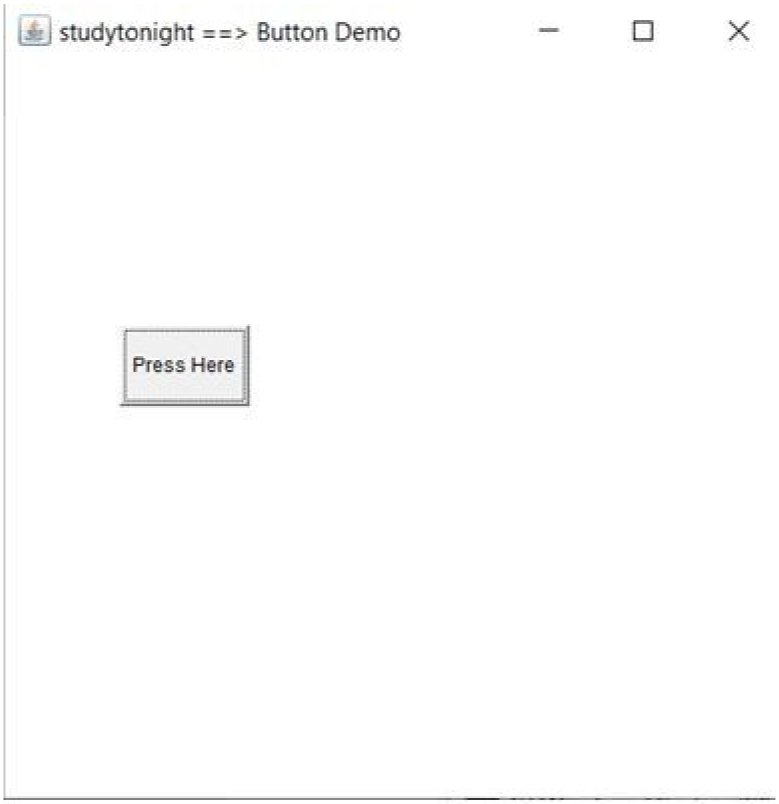
Example:

Lets take an example to create a button and it to the frame by providing coordinates.

```
import java.awt.*;
public class ButtonDemo1
{
public static void main(String[] args)
{
    Frame f1=new Frame("==> Button Demo");
    Button b1=new Button("Press Here");
    b1.setBounds(80,200,80,50);
    f1.add(b1);
    f1.setSize(500,500);
    f1.setLayout(null);
    f1.setVisible(true);
}
}
```



```
D:\Studytonight>javac ButtonDemo1.java

D:\Studytonight>java ButtonDemo1
```



AWT Label

In Java, AWT contains a Label Class. It is used for placing text in a container. Only Single line text is allowed and the text can not be changed directly.

Label Declaration:

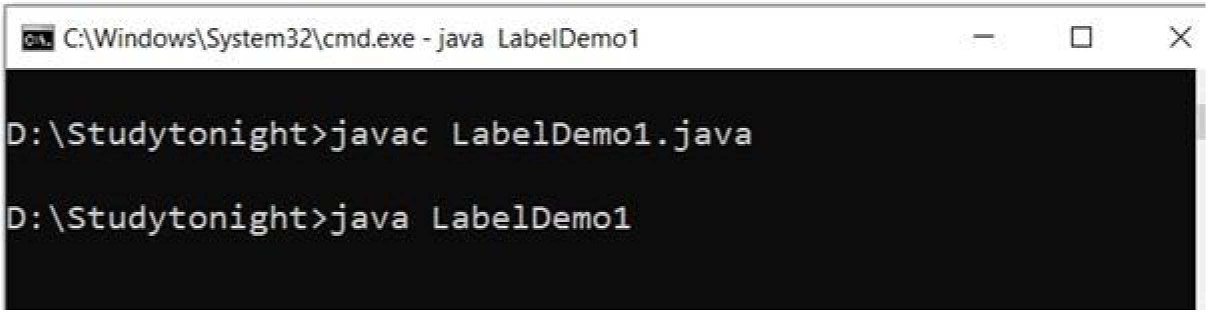
public class Label extends Component implements Accessible

Example:

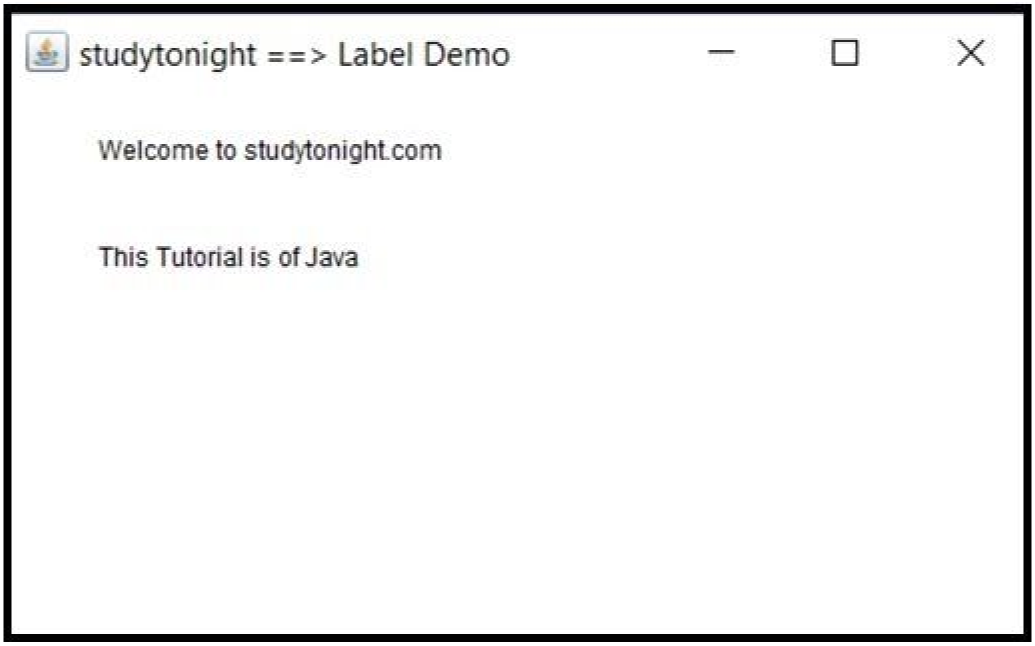
In this example, we are creating two labels to display text to the frame.

```
import java.awt.*;
class LabelDemo1
```

```
{
 public static void main(String args[])
  {
    Frame l_Frame= new Frame("studytonight ==> Label Demo");
    Label lab1,lab2;
    lab1=new Label("Welcome to studytonight.com");
    lab1.setBounds(50,50,200,30);
    lab2=new Label("This Tutorial is of Java");
    lab2.setBounds(50,100,200,30);
l_Frame.add(lab1);
l_Frame.add(lab2);
l_Frame.setSize(500,500);
l_Frame.setLayout(null);
l_Frame.setVisible(true);
}
}
```



```
D:\Studytonight>javac LabelDemo1.java

D:\Studytonight>java LabelDemo1
```



Welcome to studytonight.com

This Tutorial is of Java

AWT TextField

In Java, AWT contains aTextField Class. It is used for displaying single line text.

TextField Declaration:

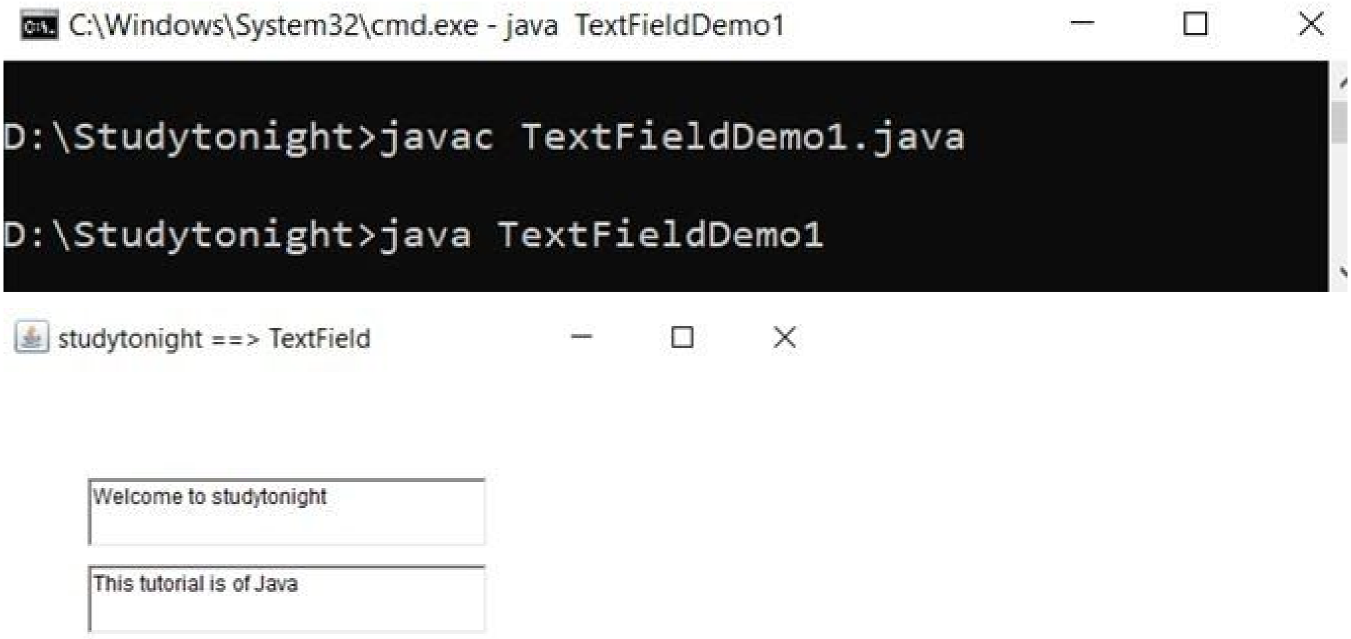
public class TextField extends TextComponent

Example:

We are creating two textfields to display single line text string. This text is editable in nature, see the below example.

```
import java.awt.*;
class TextFieldDemo1{
public static void main(String args[]){
    Frame TextF_f= new Frame("studytonight ==>TextField");
TextField text1,text2;
    text1=new TextField("Welcome to studytonight");
    text1.setBounds(60,100, 230,40);
    text2=new TextField("This tutorial is of Java");
```

```
   text2.setBounds(60,150, 230,40);
TextF_f.add(text1);
TextF_f.add(text2);
TextF_f.setSize(500,500);
TextF_f.setLayout(null);
TextF_f.setVisible(true);
}
}
```


C:\Windows\System32\cmd.exe - java TextFieldDemo1
studytonight ==> TextField

```
D:\Studytonight>javac TextFieldDemo1.java

D:\Studytonight>java TextFieldDemo1
```

Welcome to studytonight

This tutorial is of Java

AWT TextArea

In Java, AWT contains aTextArea Class. It is used for displaying multiple-line text.

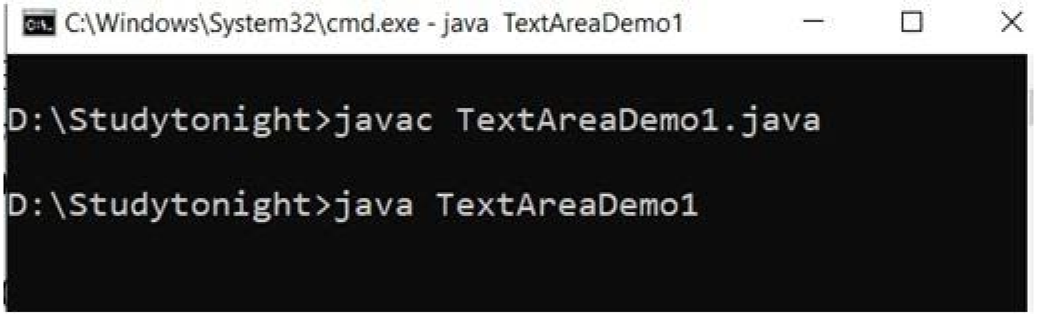
TextArea Declaration:

public class TextArea extends TextComponent

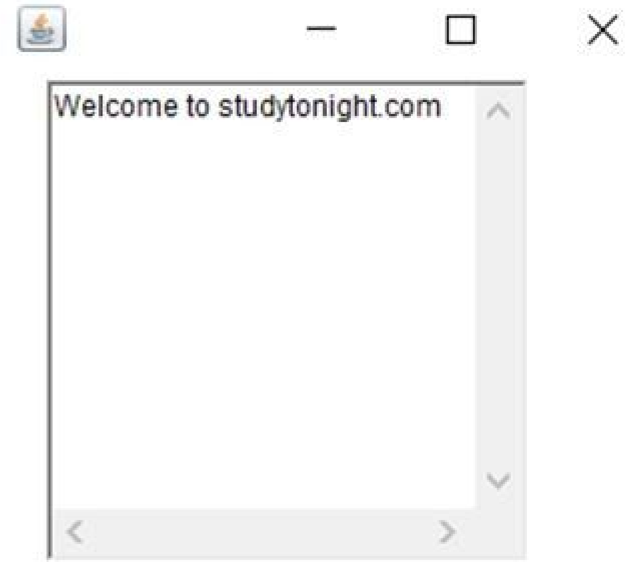
Example:

In this example, we are creating a TextArea that is used to display multiple-line text string and allows text editing as well.

```
import java.awt.*;
public class TextAreaDemo1
{
 TextAreaDemo1()
 {
  Frame textArea_f= new Frame();
  TextArea area=new TextArea("Welcome to studytonight.com");
  area.setBounds(30,40, 200,200);
  textArea_f.add(area);
  textArea_f.setSize(300,300);
  textArea_f.setLayout(null);
  textArea_f.setVisible(true);
 }
 public static void main(String args[])
 {
  new TextAreaDemo1();
 }
}
```



```
D:\Studytonight>javac TextAreaDemo1.java

D:\Studytonight>java TextAreaDemo1
```



```
Welcome to studytonight.com
```

AWT Checkbox

In Java, AWT contains a Checkbox Class. It is used when we want to select only one option i.e true or false. When the checkbox is checked then its state is "on" (true) else it is "off"(false).

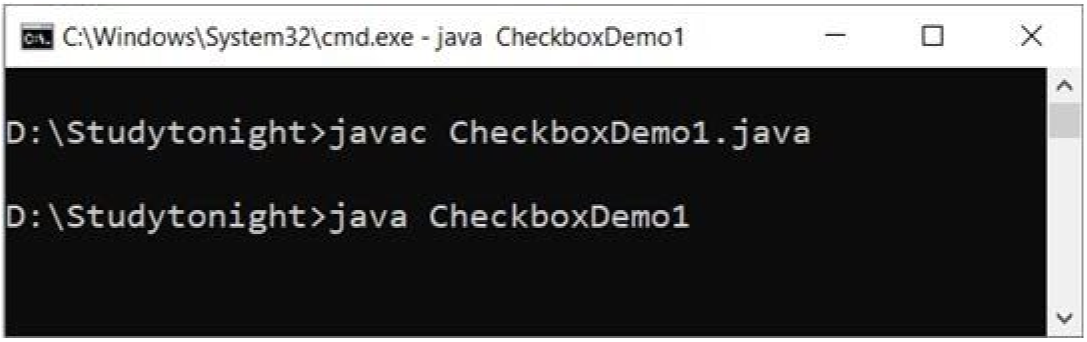
Checkbox Syntax

public class Checkbox extends Component implements ItemSelectable, Accessible

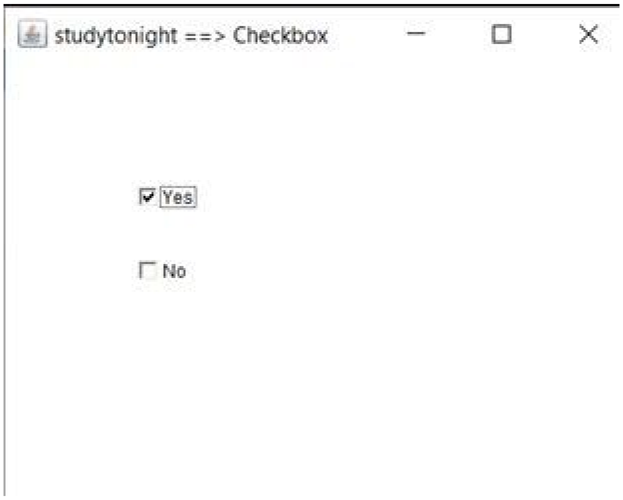
Example:

In this example, we are creating checkbox that are used to get user input. If checkbox is checked it returns true else returns false.

```
import java.awt.*;
public class CheckboxDemo1
{
  CheckboxDemo1(){
    Frame checkB_f= new Frame("studytonight ==>Checkbox Example");
    Checkbox ckbox1 = new Checkbox("Yes", true);
    ckbox1.setBounds(100,100, 60,60);
    Checkbox ckbox2 = new Checkbox("No");
    ckbox2.setBounds(100,150, 60,60);
    checkB_f.add(ckbox1);
    checkB_f.add(ckbox2);
    checkB_f.setSize(400,400);
    checkB_f.setLayout(null);
    checkB_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new CheckboxDemo1();
  }
}
```



```
D:\Studytonight>javac CheckboxDemo1.java

D:\Studytonight>java CheckboxDemo1
```



☑ Yes

□ No

AWT CheckboxGroup

In Java, AWT contains aCheckboxGroup Class. It is used to group a set of Checkbox. When Checkboxes are grouped then only one box can be checked at a time.

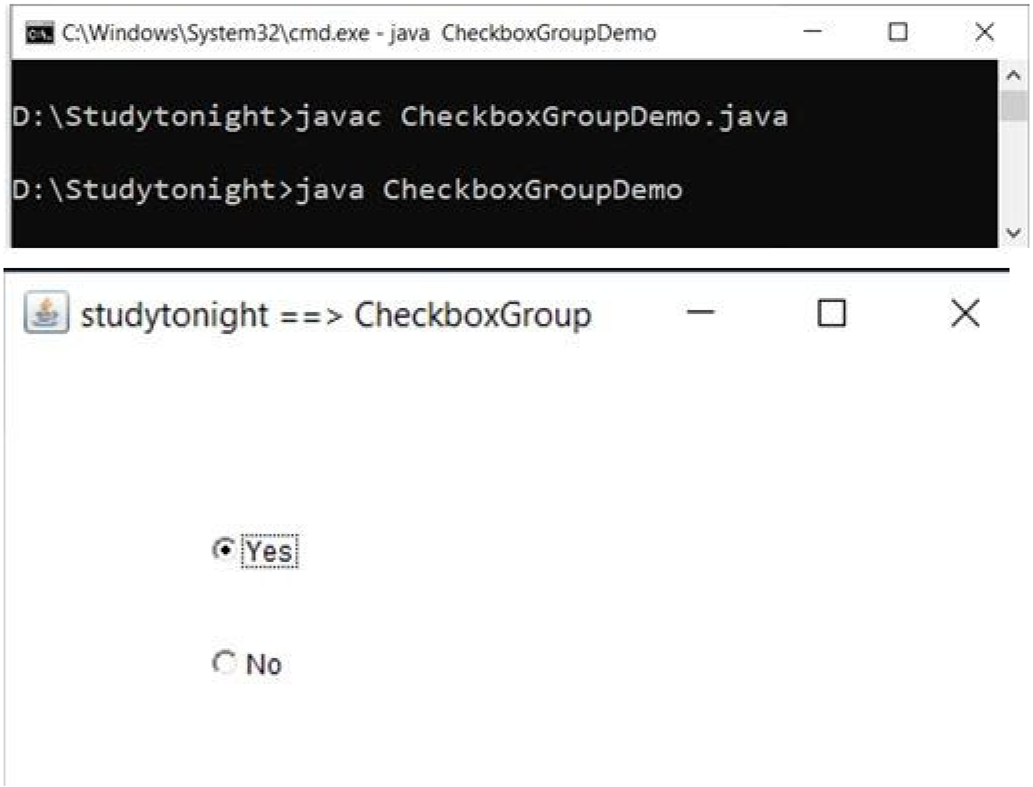
CheckboxGroup Declaration:

public class CheckboxGroup extends Object implements Serializable

Example:

This example creates a checkboxgroup that is used to group multiple checkbox in a single unit. It is helpful when we have to select single choice among the multiples.

```
import java.awt.*;
public class CheckboxGroupDemo
{
  CheckboxGroupDemo(){
    Frame ck_groupf= new Frame("studytonight ==>CheckboxGroup");
    CheckboxGroupobj = new CheckboxGroup();
    Checkbox ckBox1 = new Checkbox("Yes", obj, true);
    ckBox1.setBounds(100,100, 50,50);
    Checkbox ckBox2 = new Checkbox("No", obj, false);
    ckBox2.setBounds(100,150, 50,50);
    ck_groupf.add(ckBox1);
    ck_groupf.add(ckBox2);
    ck_groupf.setSize(400,400);
    ck_groupf.setLayout(null);
    ck_groupf.setVisible(true);
  }
  public static void main(String args[])
  {
    new CheckboxGroupDemo();
  }
}
```



C:\Windows\System32\cmd.exe - java CheckboxGroupDemo

```
D:\Studytonight>javac CheckboxGroupDemo.java

D:\Studytonight>java CheckboxGroupDemo
```

studytonight ==> CheckboxGroup

● Yes

○ No

AWT Choice

In Java, AWT contains a Choice Class. It is used for creating a drop-down menu of choices. When a user selects a particular item from the drop-down then it is shown on the top of the menu.

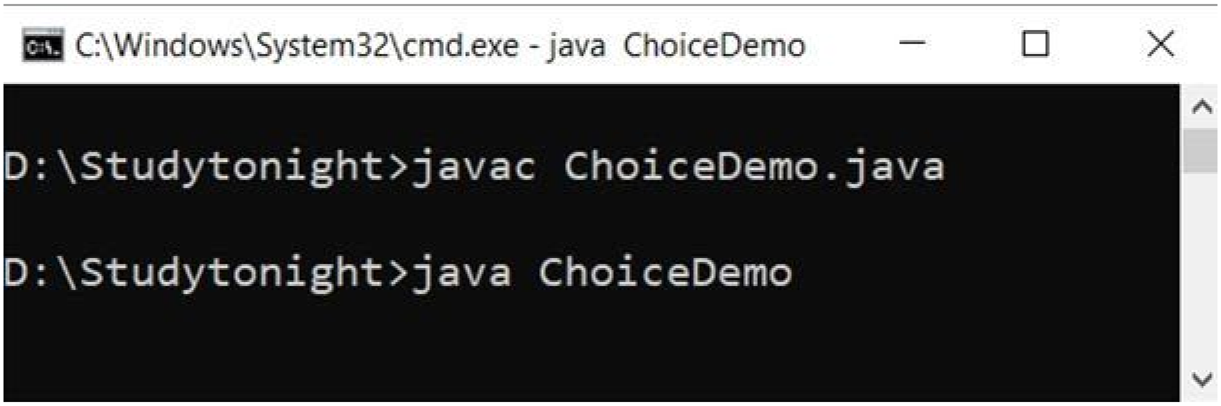
Choice Declaration:

public class Choice extends Component implements ItemSelectable, Accessible

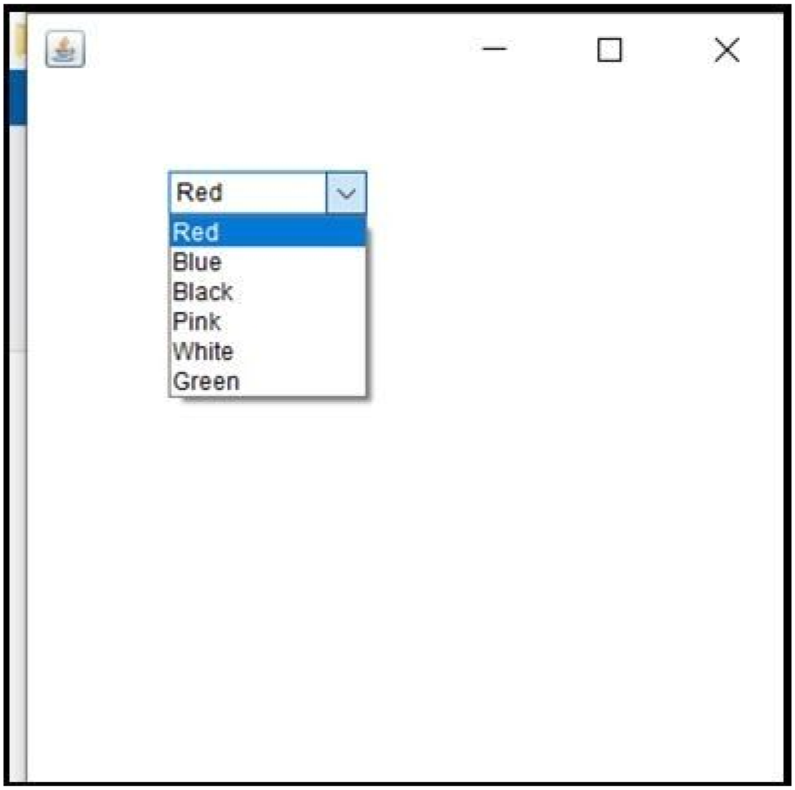
Example:

In this example, we are creating drop-down menu that is used to get user choice from multiple choices.

```
import java.awt.*;
public class ChoiceDemo
{
  ChoiceDemo()
  {
    Frame choice_f= new Frame();
    Choice obj=new Choice();
    obj.setBounds(80,80, 100,100);
    obj.add("Red");
    obj.add("Blue");
    obj.add("Black");
    obj.add("Pink");
    obj.add("White");
    obj.add("Green");
    choice_f.add(obj);
    choice_f.setSize(400,400);
    choice_f.setLayout(null);
    choice_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new ChoiceDemo();
  }
}
```



```
C:\Windows\System32\cmd.exe - java  ChoiceDemo

D:\Studytonight>javac ChoiceDemo.java

D:\Studytonight>java ChoiceDemo
```



AWT List

In Java, AWT contains a List Class. It is used to represent a list of items together. One or more than one item can be selected from the list.

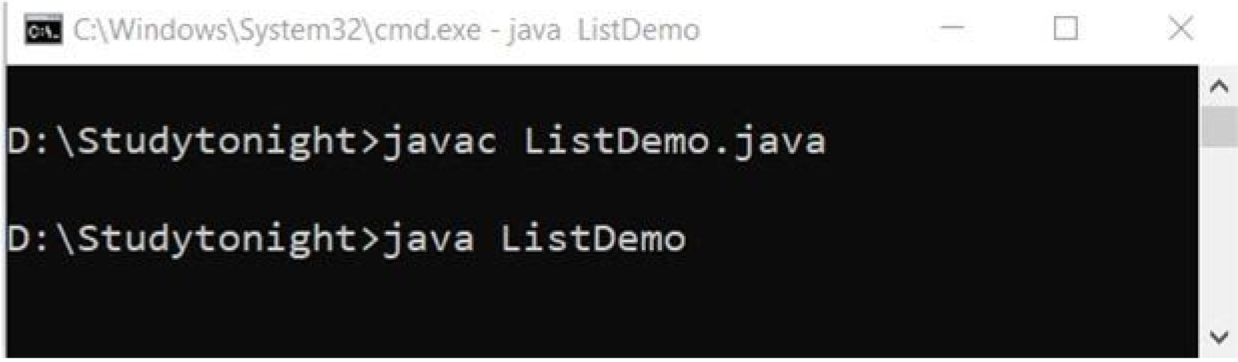
List Declaration:

public class List extends Component implements ItemSelectable, Accessible

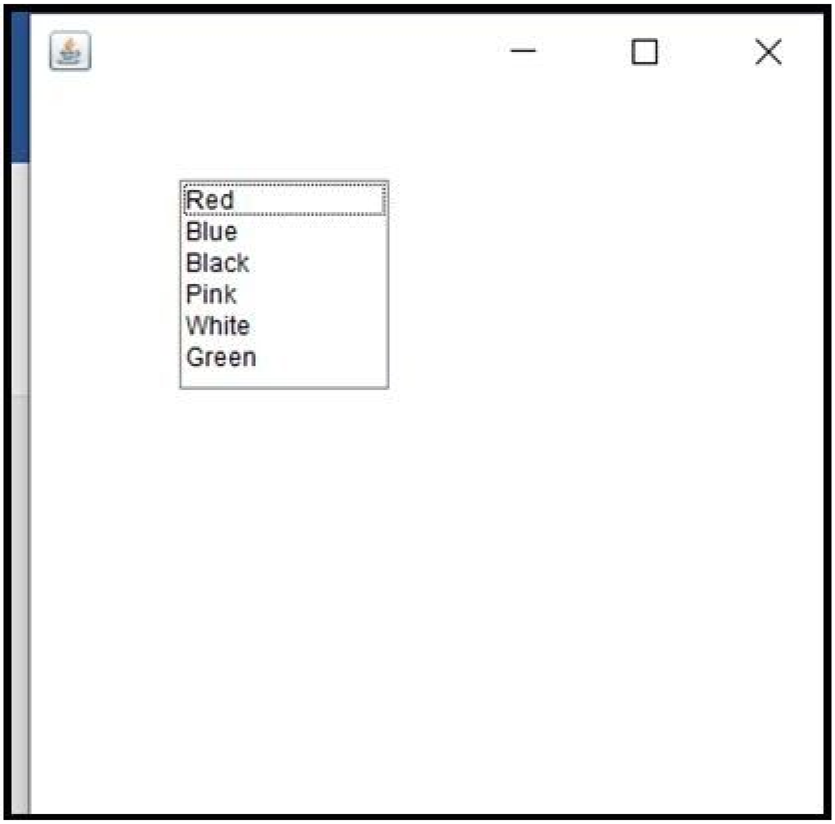
Example:

In this example, we are creating a list that is used to list out the items.

```
import java.awt.*;
public class ListDemo
{
  ListDemo()
  {
    Frame list_f= new Frame();
    List obj=new List(6);
    obj.setBounds(80,80, 100,100);
    obj.add("Red");
    obj.add("Blue");
    obj.add("Black");
    obj.add("Pink");
    obj.add("White");
    obj.add("Green");
    list_f.add(obj);
    list_f.setSize(400,400);
    list_f.setLayout(null);
    list_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new ListDemo();
  }
}
```



```
D:\Studytonight>javac ListDemo.java

D:\Studytonight>java ListDemo
```



```
Red
Blue
Black
Pink
White
Green
```

AWT Canvas

In Java, AWT contains a Canvas Class. A blank rectangular area is provided. It is used when a user wants to draw on the screen.

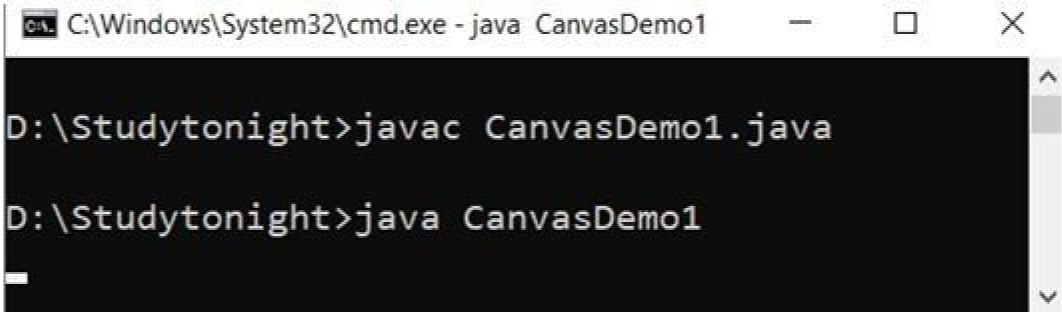
Declaration:

public class Canvas extends Component implements Accessible

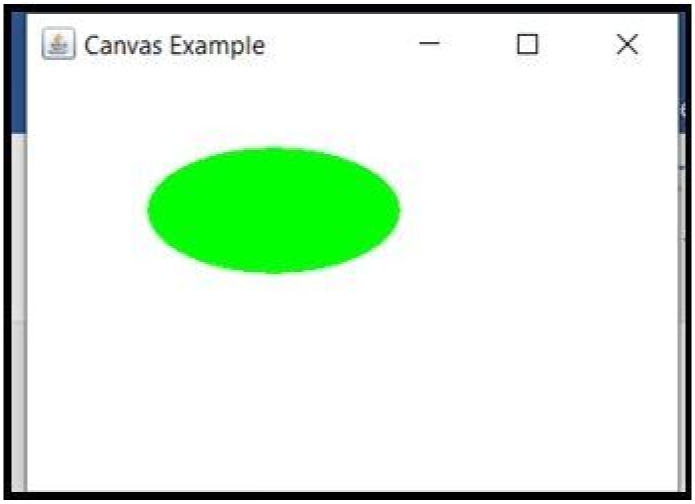
Example:

The canvas is used to provide a place to draw using mouse pointer. We can used it to get user architectural user input.

```
import java.awt.*;
public class CanvasDemo1
{
 public CanvasDemo1()
 {
  Frame canvas_f= new Frame("studytonight ==> Canvas");
  canvas_f.add(new CanvasDemo());
  canvas_f.setLayout(null);
  canvas_f.setSize(500, 500);
  canvas_f.setVisible(true);
 }
 public static void main(String args[])
 {
  new CanvasDemo1();
 }
}
class CanvasDemo extends Canvas
{
 public CanvasDemo() {
  setBackground (Color.WHITE);
  setSize(300, 200);
 }
 public void paint(Graphics g)
 {
  g.setColor(Color.green);
  g.fillOval(80, 80, 150, 75);
 }
}
```



```
C:\Windows\System32\cmd.exe - java  CanvasDemo1

D:\Studytonight>javac CanvasDemo1.java

D:\Studytonight>java CanvasDemo1
```



Java AWT Panel

The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.

It doesn't have title bar.

AWT Panel class declaration

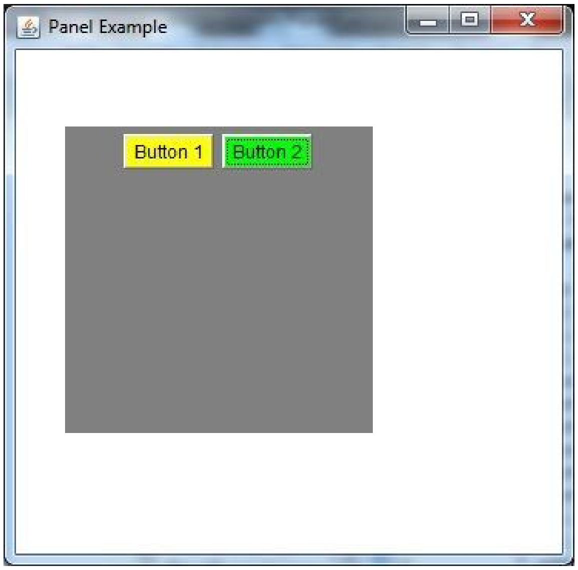
```
public class Panel extends Container implements Accessible
```

Java AWT Panel Example

```
import java.awt.*;
public class PanelExample {
     PanelExample()
        {
        Frame f= new Frame("Panel Example");
        Panel panel=new Panel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        Button b1=new Button("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        Button b2=new Button("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        }
        public static void main(String args[])
        {
```

```
        new PanelExample();
        }
}
```

Output:



Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

```
public class MenuItem extends MenuComponent implements Accessible
```

AWT Menu class declaration

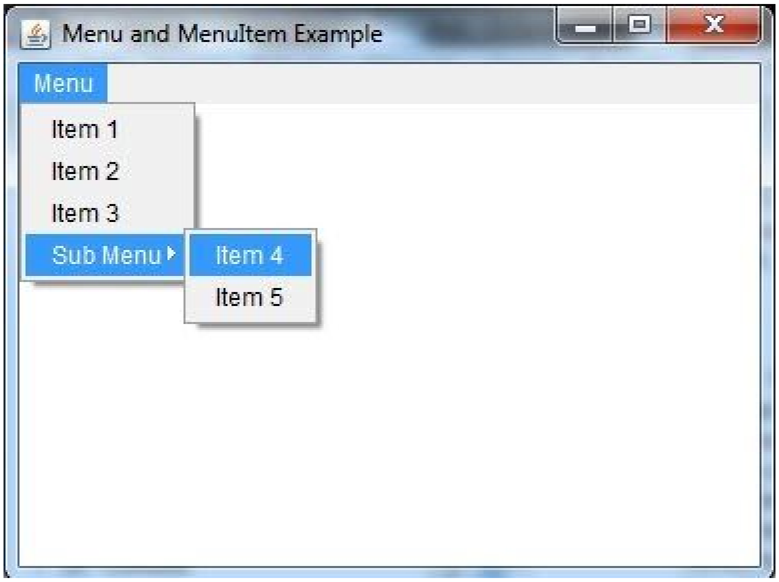
```
public class Menu extends MenuItem implements MenuContainer, Accessible
```

Java AWT MenuItem and Menu Example

```
import java.awt.*;
class MenuExample
```

```
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("File");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```

Output:



Java AWT Dialog

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Unlike Frame, it doesn't have maximize and minimize buttons.

Frame vs Dialog

Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have.

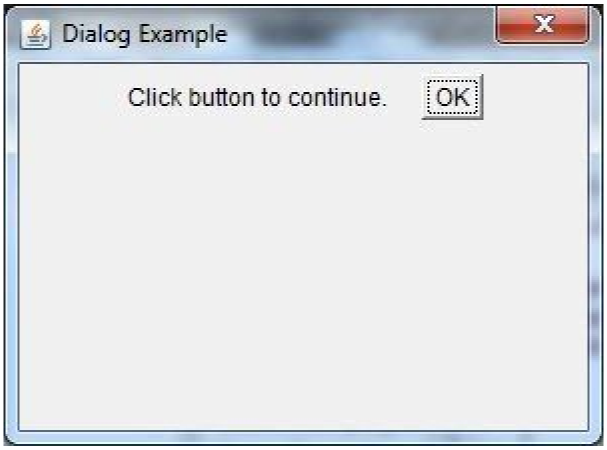
AWT Dialog class declaration

```
public class Dialog extends Window
```

Java AWT Dialog Example

```
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
    private static Dialog d;
    DialogExample() {
        Frame f= new Frame();
        d = new Dialog(f , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        Button b = new Button ("OK");
        d.add( new Label ("Click button to continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
    }
    public static void main(String args[])
    {
        new DialogExample();
    }
}
```

Output:



Java LayoutManagers

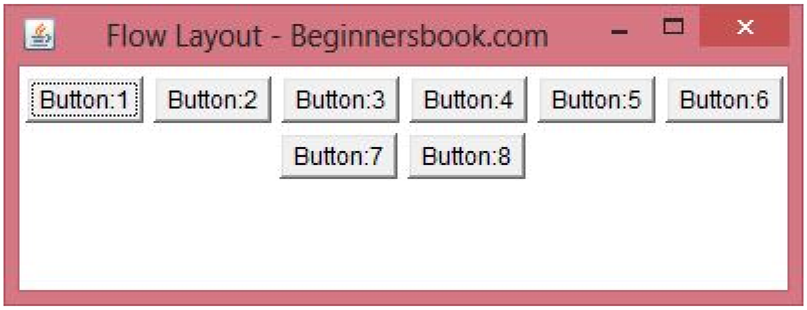
The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout

Java – FlowLayout in AWT

Flow layout is the default layout, which means if you don't set any layout in your code then layout would be set to Flow by default. Flow layout puts components (such as text fields, buttons, labels etc) **in a row**, if horizontal space is not enough to hold all components then Flow layout adds them in a next row and so on.

Example: Here is the image of a Frame where eight buttons have been added to a Frame under Flow layout. As you can see buttons 7 & 8 are in second row because first six buttons consumed all horizontal space.



Points to Note:

- All rows in Flow layout are **center aligned by default**. As you can see in the above image that buttons 7 & 8 are in center. However we can set the alignment to left or right, we will learn about it later in this post.
- The default horizontal and vertical gap between components is **5 pixels**.

- **By default the components Orientation is left to right**, which means the components would be added from left to right, however we can change it to right to left as well, we will see that later in this post.

Simple Flow Layout Example

The image shown above is the output of this code. Here we are adding 8 buttons to a Frame and layout is being set to FlowLayout.

```
import java.awt.*;
public class FlowLayoutDemo extends Frame{
    // constructor
      public FlowLayoutDemo(String title)
      {
       /* It would create the Frame by calling Frame(String title)
        * the constructor of Frame class.
        */
       super(title);

       //Setting up Flow Layout
      setLayout(new FlowLayout());

      //Creating a button and adding it to the frame
      Button b1 = new Button("Button:1");
      add(b1);

      /* Adding other components to the Frame
      */
      Button b2 = new Button("Button:2");
      add(b2);

      Button b3 = new Button("Button:3");
      add(b3);

      Button b4 = new Button("Button:4");
      add(b4);

      Button b5 = new Button("Button:5");
      add(b5);

      Button b6 = new Button("Button:6");
      add(b6);

      Button b7 = new Button("Button:7");
      add(b7);

      Button b8 = new Button("Button:8");
      add(b8);
      }
      public static void main(String[] args)
      {   FlowLayoutDemo screen =  new FlowLayoutDemo("Flow Layout -
Beginnersbook.com");
      screen.setSize(400,150);
      screen.setVisible(true);
      }
}
```

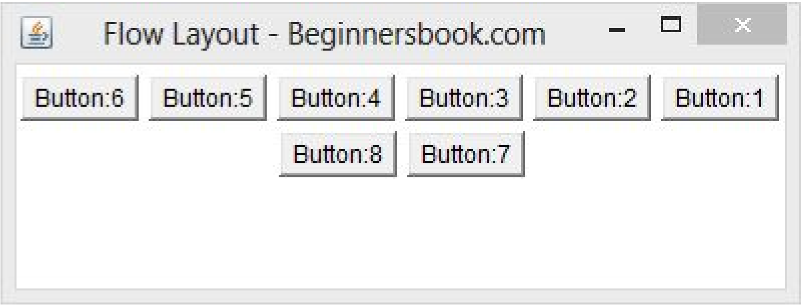
Flow Layout where Orientation is right to left

The default Orientation for flow layout is left to right, however we can set it to right to left if want.

```
import java.awt.*;
public class FlowLayoutDemo extends Frame{
    // constructor
      public FlowLayoutDemo(String title)
      {
       /* It would create the Frame by calling
        * the constructor of Frame class.
        */
       super(title);

       //Setting up Flow Layout
      setLayout(new FlowLayout());

      //Creating a button and adding it to the frame
      Button b1 = new Button("Button:1");
      add(b1);

      /* Adding other components to the Frame
      */
      Button b2 = new Button("Button:2");
      add(b2);

      Button b3 = new Button("Button:3");
      add(b3);
```

```
        Button b4 = new Button("Button:4");
        add(b4);

        Button b5 = new Button("Button:5");
        add(b5);

        Button b6 = new Button("Button:6");
        add(b6);

        Button b7 = new Button("Button:7");
        add(b7);

        Button b8 = new Button("Button:8");
        add(b8);
        /* This would set the Orientation to
        *  RightToLeft
        */
        setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
        }
        public static void main(String[] args)
        {   FlowLayoutDemo screen = new FlowLayoutDemo("Flow Layout -
Beginnersbook.com");
        screen.setSize(400,150);
        screen.setVisible(true);
        }
}
```

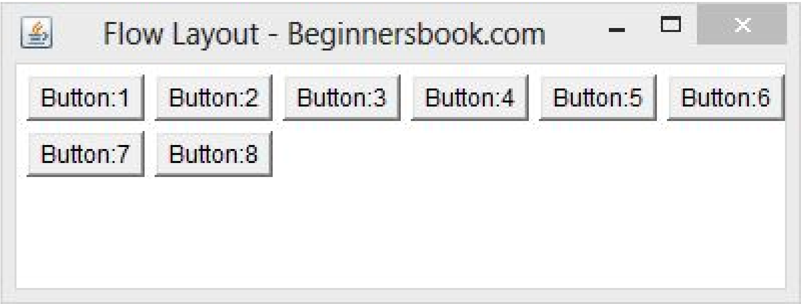
Output:



Left alignment of components in FlowLayout

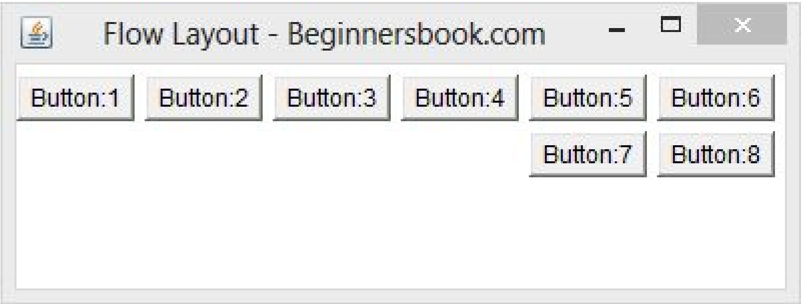
As we have seen in the examples above that the rows are center aligned (look at the buttons 7 & 8 in the image above). However we can change the alignment by passing the parameters to the constructor of flow layout while setting up the layout.

To change the alignment to Left, replace the statement setLayout(new FlowLayout()); with this one: setLayout(new FlowLayout(FlowLayout.LEFT)); in the example 1. The output would look like the image below –



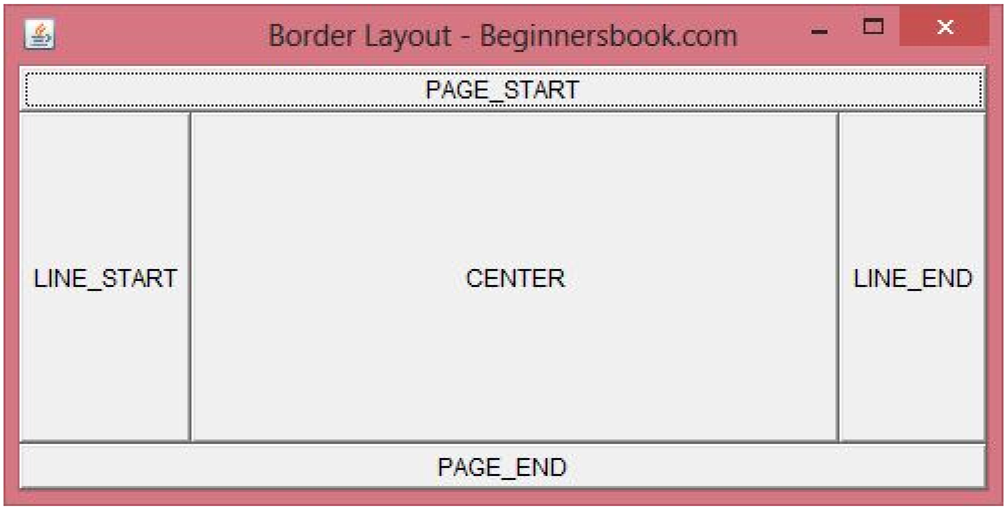
Right alignment of components in FlowLayout

To change the alignment to Right, replace the statement setLayout(new FlowLayout()); with this one: setLayout(new FlowLayout(FlowLayout.RIGHT)); in the example 1. The output would look like the image below –



Java – Border Layout in AWT

In Border layout we can add components (such as text fields, buttons, labels etc) to the five specific regions. These regions are called **PAGE_START, LINE_START, CENTER, LINE_END, PAGE_END.** Refer the diagram below to understand their location on a Frame.



The diagram above is the output of below code, where I have added five buttons (which have same name as regions in which they have been placed) to the five regions of Border Layout. You can add any component of your choice in a similar manner.

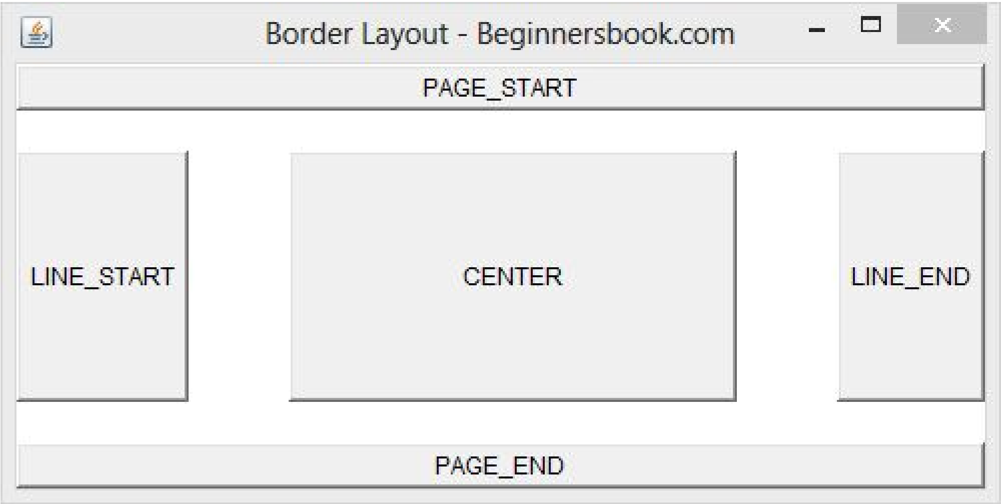
```
import java.awt.*;
public class BorderDemo extends Frame{
    // constructor
     public BorderDemo(String title)
     {
      /* It would create the Frame by calling the constructor of Frame
class.*/
      super(title);
      //Setting up Border Layout
     setLayout(new BorderLayout());
     //Creating a button and adding it to PAGE_START area
     Button b1 = new Button("PAGE_START");
     add(b1, BorderLayout.PAGE_START);

     /* Similarly creating 4 other buttons and adding
     * them to other 4 areas of Border Layout    */
     Button b2= new Button("CENTER");
     add(b2, BorderLayout.CENTER);
     Button b3= new Button("LINE_START");
     add(b3, BorderLayout.LINE_START);

     Button b4= new Button("PAGE_END");
     add(b4, BorderLayout.PAGE_END);

     Button b5= new Button("LINE_END");
     add(b5, BorderLayout.LINE_END);
     }
     public static void main(String[] args)
     {   BorderDemo screen = new BorderDemo("Border Layout");
     screen.setSize(500,250);
     screen.setVisible(true);
     }
 }
```

Note: The name of buttons in the example above are intentionally set same as region names, this is just for educational purpose, you can name them as per your wish and requirement.

What if you want spaces among regions?

In the example above, we do not have any space among regions; however we can have horizontal as well as vertical space between regions. There are two ways to do it –

1) Notice the statement setLayout(new BorderLayout( )); in the example above, if you change it to this: **setLayout(new BorderLayout(50,20));** then the output Frame would look like the image below. Here **50 is horizontal gap and 20 is vertical gap**.



Method details:

public BorderLayout(int hgap, int vgap)

Constructs a border layout with the specified gaps between components. The horizontal gap is specified by hgap and the vertical gap is specified by vgap.

Parameters:

hgap – the horizontal gap.

vgap – the vertical gap.

2) You can also do it by using setHgap(int hgap) method for horizontal gap between components and setVgap(int vgap) method for vertical gap.

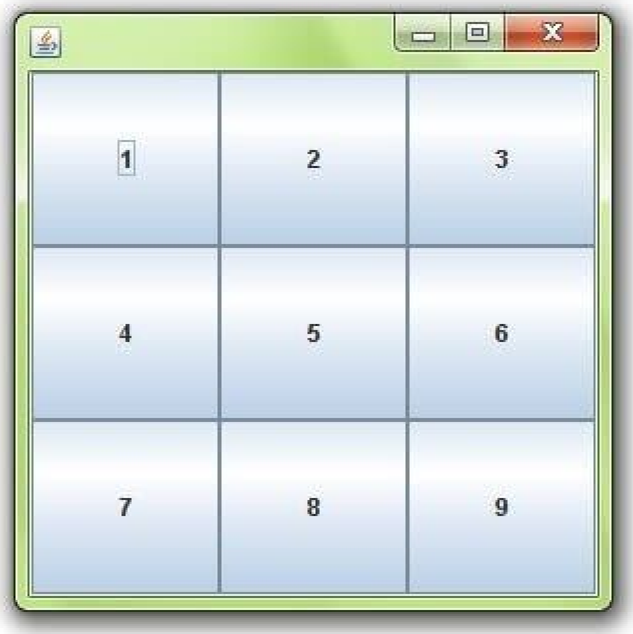
Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with **one column** per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class



Program:

```
package AWTDemo;
import java.awt.*;
public class MyGridLayout{
          Frame f;
MyGridLayout(){
   f=new Frame();
   Button b1=new Button("1");
   Button b2=new Button("2");
   Button b3=new Button("3");
   Button b4=new Button("4");
   Button b5=new Button("5");
   Button b6=new Button("6");
   Button b7=new Button("7");
   Button b8=new Button("8");
   Button b9=new Button("9");

   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
   f.add(b6);f.add(b7);f.add(b8);f.add(b9);

   f.setLayout(new GridLayout(3,3));
   //setting grid layout of 3 rows and 3 columns

   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args) {
   new MyGridLayout();
}
}
```

Event Handling in AWT JAVA

What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of **source**. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through a keyboard, selecting an item from a list, scrolling the page are the activities that cause an event to happen.

Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through the keyboard,selecting an item from a list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of the end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, and operation completion are the example of background events.

What is Event Handling?

Event Handling is the mechanism that controls the **event** and decides what should happen if an event occurs. **This mechanism has the code which is known as an event handler that is executed when an event occurs.** Java Uses the **Delegation Event Model** to handle the events. This model defines the standard mechanism to generate and handle the events.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provides classes for source objects.
- **Listener** - It is also known as **event handler**.Listener is responsible for generating response to an event. From a Java implementation point of view the listener is also an object. **Listener waits until it receives an event. Once the event is received , the listener processes the event and then returns.**

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of the concerned event class is created automatically and information about the source and the event get populated within the same object.
- Event object is forwarded to the method of registered listener class.
- the method now gets executed and returns.

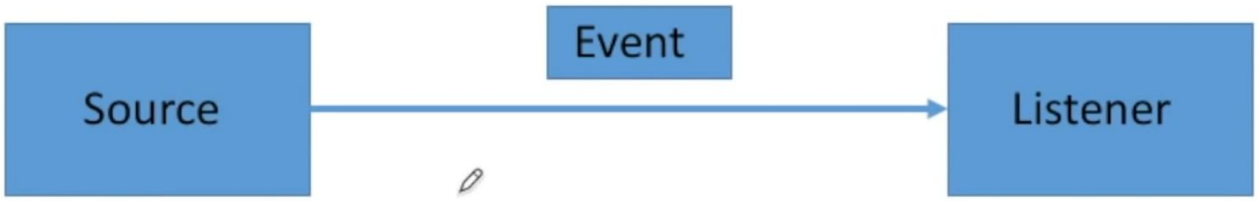
Points to remember about listener

- In order to design a **listener class** we have to develop some **listener interfaces.**These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.
- If you do not implement any of the predefined interfaces then your class can not act as a listener class for a source object.

Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represents an event method. In response to an event java jre will fire callback method. All such callback methods are provided in listener interfaces.

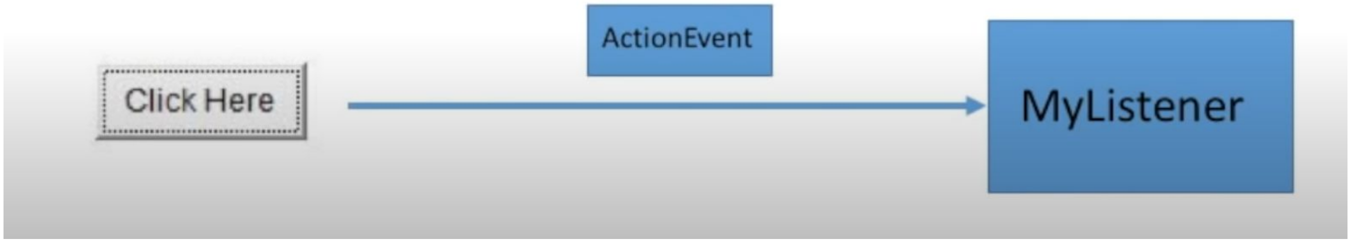
If a component wants some listener will listen to it's events the the source must register itself to the listener.



Event

Source

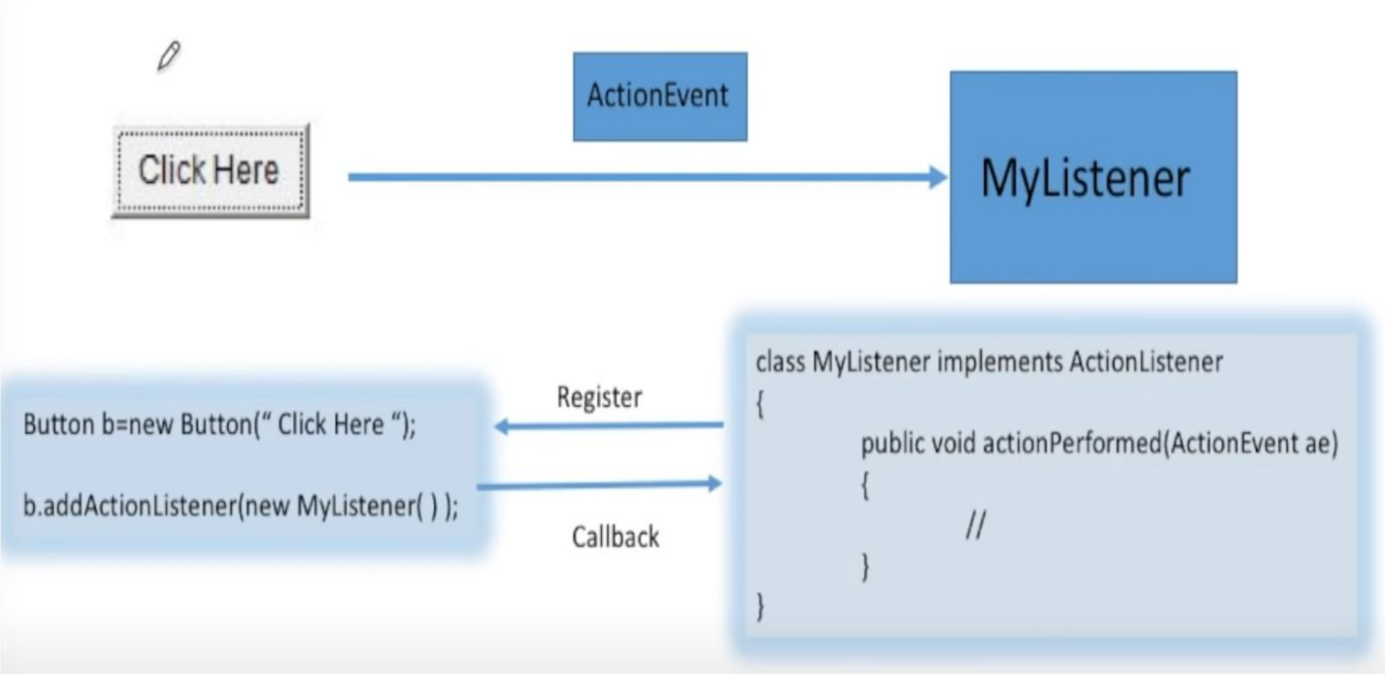
Listener



ActionEvent

Click Here

MyListener



ActionEvent

Click Here

MyListener

Register

Callback

```
Button b=new Button(" Click Here ");

b.addActionListener(new MyListener( ) );
```

```
class MyListener implements ActionListener
{
            public void actionPerformed(ActionEvent ae)
            {
                        //
            }
}
```

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	**Listener Interfaces**
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Components of Event Handling

Event handling has three main components,

- **Events :** An event is a change in state of an object.
- **Events Source :** Event source is an object that generates an event.
- **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs

How Events are handled?

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Important Event Classes and Interface

Event Classes	**Description**	**Listener Interface**
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener

MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener

AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

---

Steps to handle events:

1. Implement appropriate interface in the class.
2. Register the component with the listener.

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
    - public void addActionListener(ActionListener a){}
- **MenuItem**
    - public void addActionListener(ActionListener a){}
- **TextField**
    - public void addActionListener(ActionListener a){}
    - public void addTextListener(TextListener a){}

- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

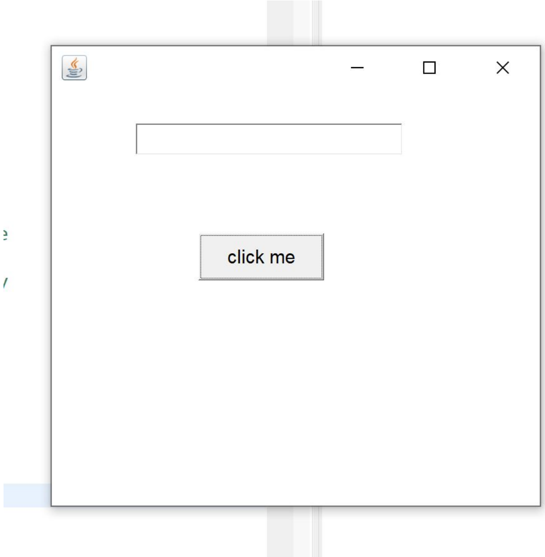
Java Event Handling Code

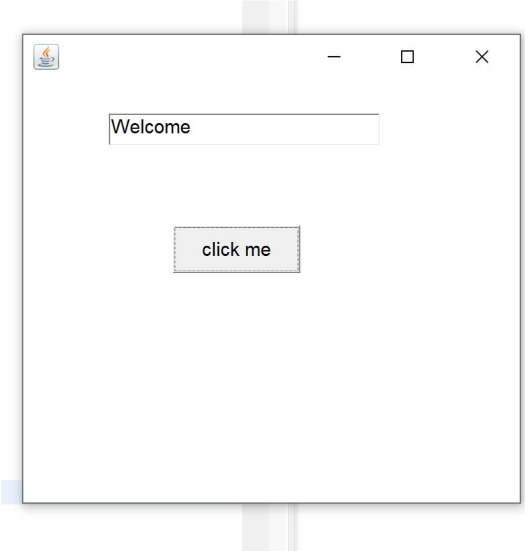
We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

1. Java event handling by implementing ActionListener within Class

```
package AWTDemo;
import java.awt.*;
import java.awt.event.*;
class ActionDemo extends Frame implements ActionListener{
TextField tf;
ActionDemo(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new ActionDemo();
}
}
```




Welcome

2) Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
TextField tf;
AEvent2(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
```

```
import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}
```

3) Java event handling by anonymous class

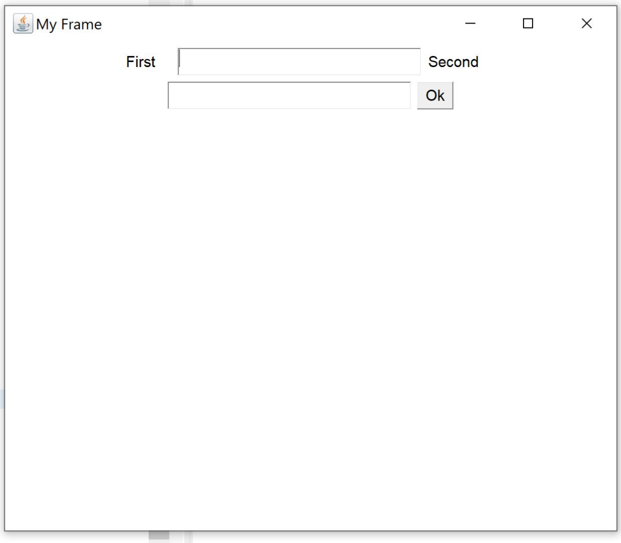
```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
TextField tf;
AEvent3(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener(){
public void actionPerformed(){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent3();
}
}
```

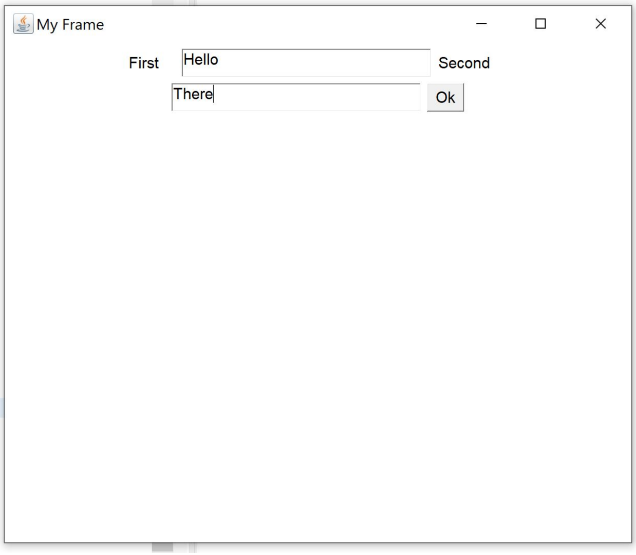
Example-2:

package AWTDemo;

```
import java.awt.*;
import java.awt.event.*;

public class ActionListenDemo {
        public static void main(String[] args)
        {
                Frame f=new Frame("My Frame");
                Label l1=new Label("First");
                Label l2=new Label("Second");

                TextField t1=new TextField(10);
                TextField t2=new TextField(10);

                Button b=new Button("Ok");
                f.add(l1);
                f.add(t1);
                f.add(l2);
                f.add(t2);
                f.add(b);
                f.setSize(500,500);
                f.setLayout(new FlowLayout());
                f.setVisible(true);
                b.addActionListener(new ActionListener()
                        {
                          public void actionPerformed(ActionEvent ae)
                          {
                                  String temp=t1.getText();
                                  t1.setText(t2.getText());
                                  t2.setText(temp);
                          }
                        }       );
        }
}
```

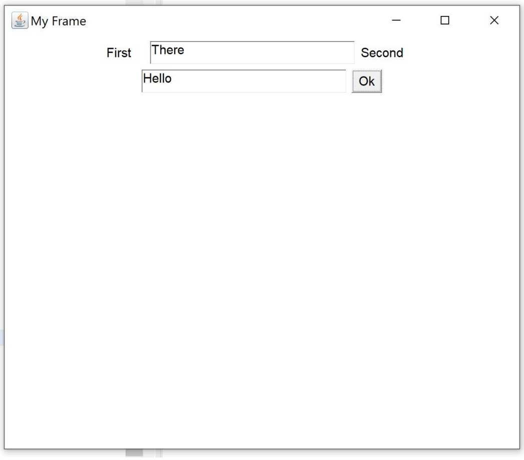
Output:





Hello

There



There

Hello

Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

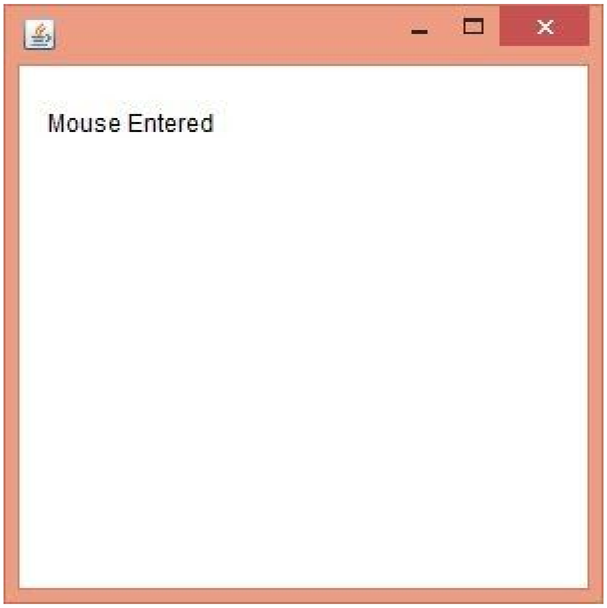
1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);

```
public abstract void mousePressed(MouseEvent e);
public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener{
    Label l;
    MouseListenerExample(){
        addMouseListener(this);
        l=new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
        l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
        l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
        l.setText("Mouse Released");
    }
public static void main(String[] args) {
    new MouseListenerExample();
}
}
```

Output:



```
Mouse Entered
```

WindowEvent and WindowListener

An event of type **WindowEvent** is generated in such situations -

- When a window is activated for the first time.
- When a window is minimized.
- When a window is brought up back from minimized state.
- When the close button **(x)** of window is clicked to close it.

Some methods of WindowEvent class

public Window getWindow()	Returns the window which triggered the **WindowEvent.**
public int getNewState()	Returns the new state of the window.

A class to listen & respond to a WindowEvent must perform the next two steps:

- It should implement **WindowListener** interface, by implementing its all next seven methods -

-

public void windowOpened(WindowEvent e)	This method is called when a window is opened for the first time.
public void windowActivated(WindowEvent e)	This method is called when a window shows up on screen.
public void windowDeactivated(WindowEvent e)	This method is called is no longer the window in use or active.
public void windowIconified(WindowEvent e)	This method is called when a window is changed from a normal to a minimized state.
public void windowDeiconified(WindowEvent e)	This method is called when a window is brought up on the screen from a minimized state.
public void windowClosing(WindowEvent ke)	This method is called a user clicks on the (x) icon to close the window.
public void windowClosed(WindowEvent e)	This method is called when a window has been closed.

- A **WindowEvent** event source is a window in which such event is generated, must call its method -

public void addWindowListener(ItemListener *object*)	where *object* is an object of the class that has implemented **Windowistener** interface. Doing this, registers the class to listen and respond to **WindowEvent**.

Handling an WindowEvent by implementing WindowListener interface

In the upcoming code, we are going to create a class that will listen to **WindowEvent**, by implementing **WindowListener** interface. In this code, WindowEvent is generated when a key is pressed and released within a textfield.

```
import java.awt.*;
import java.awt.event.*;
public class WindowEx1 implements WindowListener
{
Label label1;
Frame frame;
WindowEx1()
{
frame = new Frame("Handling KeyEvent");
label1= new Label("-See window events -", Label.CENTER);
frame.setLayout(new FlowLayout());
frame.add(label1);
//Registering class WindowEx1 to catch and respond to window events
frame.addWindowListener(this);
frame.setSize(340,200);
frame.setVisible(true);
}
```

```
public void windowActivated(WindowEvent we)

{

System.out.println("Window Activated");

}

public void windowClosed(WindowEvent we)

{

System.out.println("Window Closed");

}

public void windowClosing(WindowEvent we)

{

frame.dispose();

System.out.println("Window Closing");

}

public void windowDeactivated(WindowEvent we)

{

System.out.println("Window Deactivated");

}

public void windowDeiconified(WindowEvent we)

{

System.out.println("Window Deiconified");

}


public void windowIconified(WindowEvent we)

{

System.out.println("Window Iconified/minimized");

}


public void windowOpened(WindowEvent e)
```

```
{
System.out.println("Window Opened for the first time");
}

public static void main(String... ar)
{
new WindowEx1();
}
}
```

When you run the code, you are presented a window shown in the window below -:

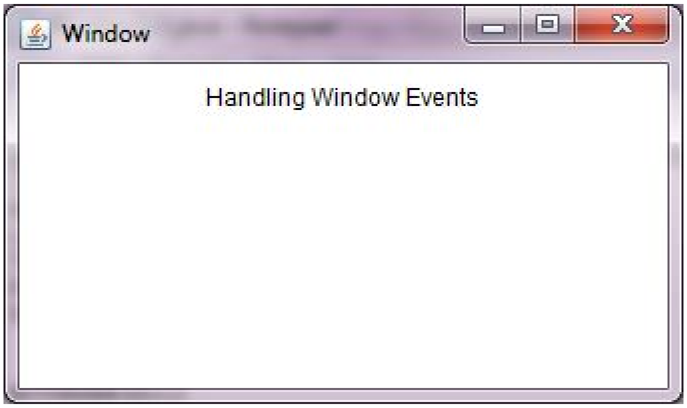


Figure 1

At the command prompt you are displayed two messages, due to the execution of methods in sequence:

- windowActivated()
- windowOpened()

```
Window Activated
Window Opened for the first time
```

When you click on the minimize button of this window to minimize it, the window is deactivated and deiconified.

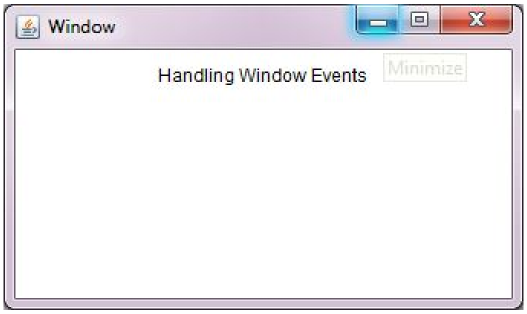


Figure 2

Hence, two new messages are displayed on the command prompt, due to the execution of methods in sequence:

- windowDeiconified()
- windowDeactivated()

```
Window Iconified/minimized
```

```
Window Deactivated
```

When you bring up the minimized window, the window is reactivated.

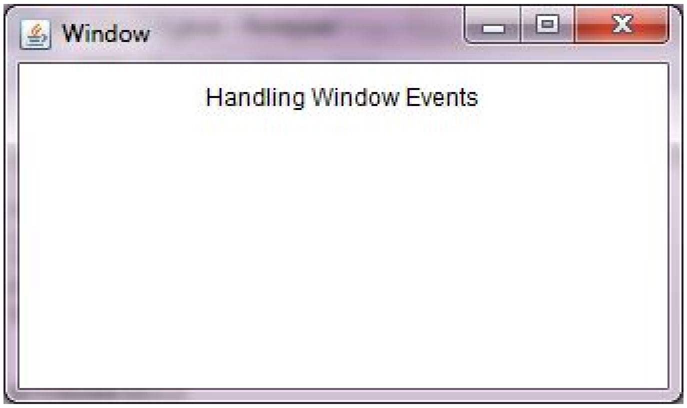


Figure 3

Hence, a new message is displayed on the command prompt, due to the execution of the method:

- windowActivated()

```
Window Activated
```

When you click on the (x) button of this window to close it, the window is deactivated and closed.

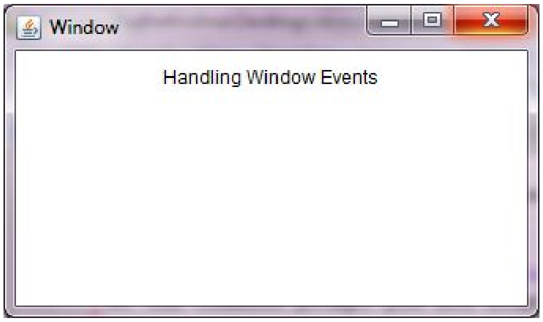


Figure 4

Hence, three new messages are displayed on the command prompt, due to the execution of methods in sequence:

- windowClosing()
- windowDeactivated()
- windowClosed()

```
Window Closing

Window Deactivated

Window Closed
```

Java WindowListener Interface

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.

Methods of WindowListener interface

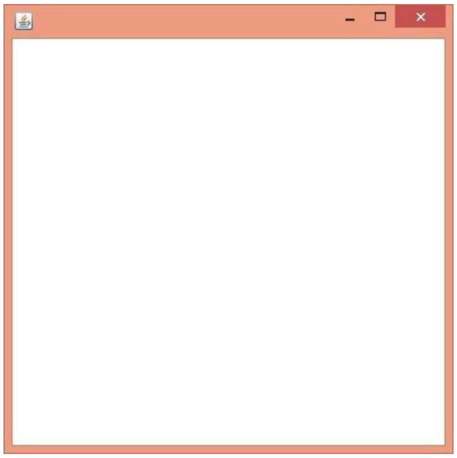
The signature of 7 methods found in WindowListener interface are given below:

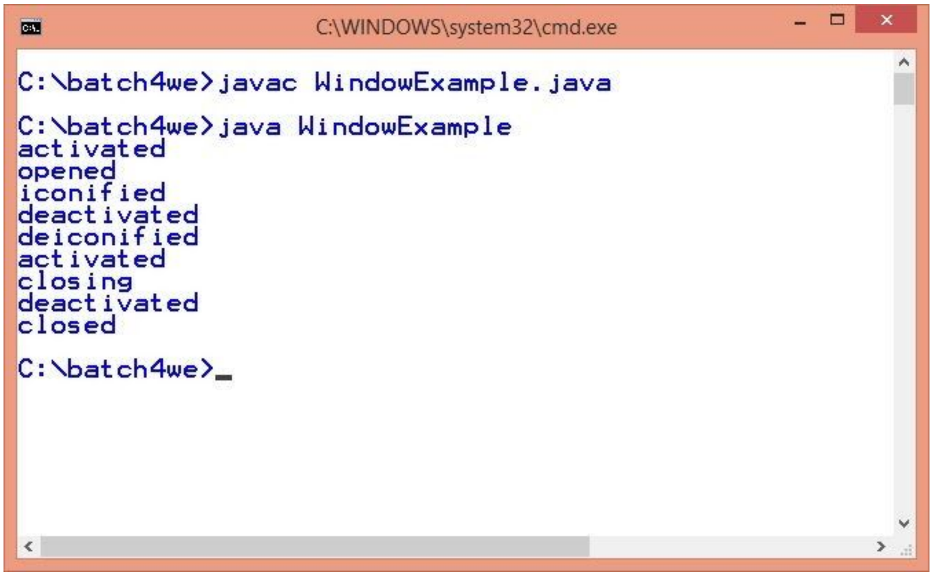
1. **public abstract void** windowActivated(WindowEvent e);
2. **public abstract void** windowClosed(WindowEvent e);
3. **public abstract void** windowClosing(WindowEvent e);
4. **public abstract void** windowDeactivated(WindowEvent e);
5. **public abstract void** windowDeiconified(WindowEvent e);
6. **public abstract void** windowIconified(WindowEvent e);
7. **public abstract void** windowOpened(WindowEvent e);

Java WindowListener Example

```java
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowExample extends Frame implements WindowListener{
    WindowExample(){
        addWindowListener(this);

        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }

public static void main(String[] args) {
    new WindowExample();
}
public void windowActivated(WindowEvent arg0) {
    System.out.println("activated");
}
public void windowClosed(WindowEvent arg0) {
    System.out.println("closed");
}
public void windowClosing(WindowEvent arg0) {
    System.out.println("closing");
    dispose();
}
public void windowDeactivated(WindowEvent arg0) {
    System.out.println("deactivated");
}
public void windowDeiconified(WindowEvent arg0) {
    System.out.println("deiconified");
}
public void windowIconified(WindowEvent arg0) {
    System.out.println("iconified");
}
public void windowOpened(WindowEvent arg0) {
    System.out.println("opened");
}
}
```

Output:





```
C:\batch4we>javac WindowExample.java

C:\batch4we>java WindowExample
activated
opened
iconified
deactivated
deiconified
activated
closing
deactivated
closed

C:\batch4we>_
```

AWT FocusListener Interface

Introduction

The interfaceFocusListener is used for receiving keyboard focus events. The class that process focus events needs to implements this interface.

Class declaration

Following is the declaration for java.awt.event.FocusListener interface:

```
public interface FocusListener
extends EventListener
```

Interface methods

S.N.	Method & Description
1	void focusGained(FocusEvent e) Invoked when a component gains the keyboard focus.
2	void focusLost(FocusEvent e) Invoked when a component loses the keyboard focus.

Methods inherited

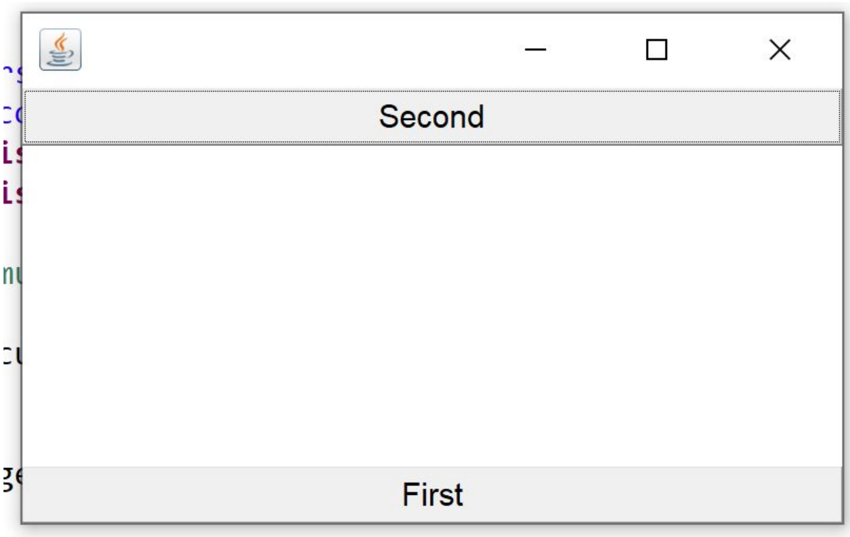
This class inherits methods from the following interfaces:

- java.awt.event.EventListener

```
package AWTDemo;
import java.awt.*;
import java.awt.event.*;

public class FocusListenertest extends Frame implements
FocusListener
    {
     Button b1,b2;

     public FocusListenertest()
         {
         add(b1=new Button ("First"),"South");
         add(b2=new Button ("Second"),"North");
         b1.addFocusListener(this);// registering
         b2.addFocusListener(this);
         setSize(200,200);
              }
     public void focusGained(FocusEvent fe) //method of focuslistener
         {
         if(fe.getSource()==b1)
         System.out.println(b1.getLabel()+"gained");
         if(fe.getSource()==b2)
         System.out.println(b2.getLabel()+"gained");
         if(fe.isTemporary())
         System.out.println("Temporary Focus");
     }
    public void focusLost(FocusEvent fe) //in focusevent "getID()"is a method
       {
       if(fe.getSource()==b1)
       System.out.println(b1.getLabel()+"lost");
       if(fe.getSource()==b2)
       System.out.println(b2.getLabel()+"lost");
    }
    public static void main(String a[])
       {
       new FocusListenertest().setVisible(true);
    }
}
```

Output:

```
Firstgained
Firstlost
Secondgained
Secondlost
```



Method	Description
object getSource()	Returns the object on which the event occurred

Java Adapter Classes

Java adapter classes *provide the default implementation of **listener interfaces.*** If you inherit the **adapter class**, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	**Listener interface**
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

AWT Event Adapters

Adapters are abstract classes for receiving various events. The methods in these classes are empty. These classes exists as convenience for creating listener objects.

AWT Adapters:

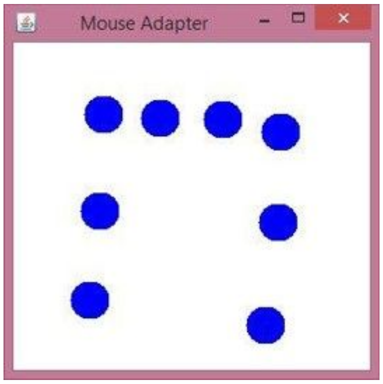
Following is the list of commonly used adapters while listening GUI events in AWT.

Sr. No.	Adapter & Description

1	**FocusAdapter**: An abstract adapter class for receiving focus events.
2	**KeyAdapter**: An abstract adapter class for receiving key events.
3	**MouseAdapter**: An abstract adapter class for receiving mouse events.
4	**MouseMotionAdapter:** An abstract adapter class for receiving mouse motion events.
5	**WindowAdapter:** An abstract adapter class for receiving window events.

Ex-1 MouseAdapter

```
package AWTDemo;
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
    Frame f;
    MouseAdapterExample(){
        f=new Frame("Mouse Adapter");
        f.addMouseListener(this);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        Graphics g=f.getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(e.getX(),e.getY(),30,30);
    }
public static void main(String[] args) {
    new MouseAdapterExample();
}
}
```


Mouse Adapter