Java Thread Method.

	Java type	method	Description
1	void	start()	It is used to start the execution of thread.
2	void	run()	It is used to do an action for a thread.
3	static void	sleep()	It sleeps a thread for the specified amount of time.
4	static thread	currentThread()	It returns a reference to the currently executing thread object
5	void	join()	It waits for a thread to die.
6	int	getPriority()	It returns the property of the thread
7	void	setPriority()	It is changes the property of the thread
8	string	getName()	It return the name of the thread
9	void	setName()	It change the name of the thread.
10	long	getId()	It return the id of the thread.
11	boolean	isAlive()	It tests if the thread is alive.
12	static void	yield()	It cause the currently executing thread object to pause and allow other threads to execute temporarily

→ Java defines two ways in which this can bee accomplished

① you can implement the Runnable interface

② you can extend the Thread class, itself

① Implementing Runable :-

→ The easiest way to create a thread is to create a class that implements the Runnable Interface.

→ Runnable abstracts a unit of executable code.

→ you can construct a thread on any object that implements Runable.

→ To implement Runnable, a class need only implement a single method called run( ) which declared with public void run())

Ex: program to create thread with runnable interface.

```
public class mythread implements Runnable{
    private int i;
    public static void main (String [] args){
        MyThread t1 = new MyThread ();
        My thread t2 = new MyThread ();
        Thread thread 1 = new Thread (t1);
        Thread thread 2 = new Thread (t2);
        thread 1.start ();
        thread 2.start ();
    }
    @ Override
    public void run ( ){
    for (i=0; i<5; i++) {
    try {
    Thread.sleep (500);
    S.O.P (Thread.currentThread ().getName()
                + ": " + i);
    }
    catch (InterruptedException ex) {
        S.O.P (ex);
    }}
    }}
```

thread priority in multithreading

Whenever we create a thread in java, it always has some priority assigned to it, priority can either be given by JVM while creating the thread or it can be given by the programmer explicitly.

priorities in threads is a concept where each thread is having a priority which in layman's language one can say very object is having priority here which represented by numbers ranging from 1 to 10

- The default priority is set to 5 as excepted
- Minimum priority is set to 1
- Maximum priority is set to 10.

(1) public final int getPriority(): java.lang.Thread.getPriority() method return priority of given thread.

(2) public final void setPriority(int newPriority): java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority.

```
public class MyThread extends Thread{
  public static void main(String[]args){
    MyThread t1=new MyThread();
    MyThread t2=new Thread();
    t1.start();
    t2.start(); }
  @Override
    public void run(){
      for(int i=0;i<5;i++){
        try{
        Thread.sleep(500);
        System.out.println(Thread.currentThread().
                    getName()+":"+i);
        }
        catch(InterruptedException ex){
        s.o.p(ex);
        }}}
```

traffic signal

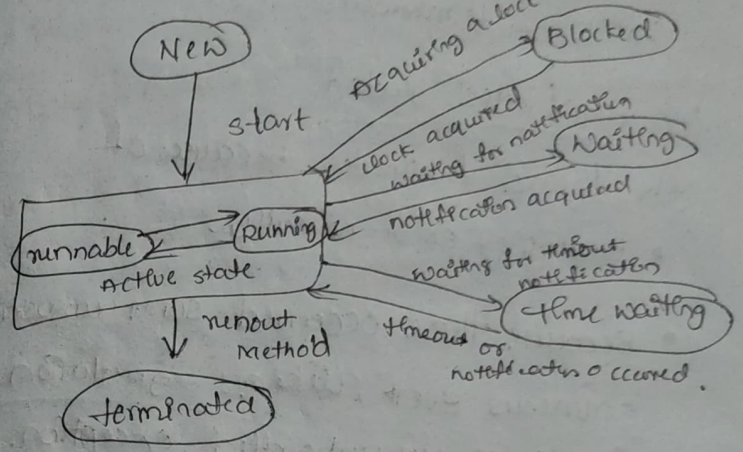
```
import java.util.*;
import java.io.*;
class traffic extends thread
{
public void run()
 {
for(int i=0;i<3;i++)
 {
try{
system.o.p("red light is glowing");
Thread.sleep(1000);
}
catch(exception e)
{
System.out.println("turned off");
 } try{
```

```
S.o.p ("yellow light is glowing");
Thread.sleep(1000);
}
catch (Exception e)
{
S.o.p ("turned off");
}
try {
S.o.p ("green ");
Thread.sleep(1000);
}
catch (Exception e)
{
System.out.println ("turned off");
}}}
class traffic signal
{
public static void main (String[] arg)
{
~~case 1: new~~
traffic t = new traffic();
t.start();
}
}
```

Life cycle of a Thread

A thread in java at any point of time exists in any one of the following states. A thread lies only in one of the shown state at any instant.

multiple states



(1) New thread:
When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state.

(2) Runnable state:
A thread that is ready to run is moved to a runnable state.

(3) Blocked: the thread will be in blocked state when it is trying to acquire a lock but currently the lock is acquired by the other thread. the thread will move from the blocked state to runnable state when it acquire the lock

(4) waiting state:-
the thread will be waiting state when it calls wait() method (or) join() method. It will move to the runnable state when other thread will notify or that thread will be terminated.

5) terminated state:-
the thread terminates because of
- Because of exist normally.
- Because there occurred some unusual erroneous event, like a segmentation fault or an unhandled exception.

---

abstract class:-
A class which is declared with a abstract key word is known as an abstract class in java
→ In abstract class must be declared with an abstract key word
→ It can be have abstract and non-abstract methods.
→ It can have constructors and static methods also.

Abstract classes & Interfaces:-
Abstract is a keyword it is used for classes & methods { abstract is a process of hiding the implementation details and showing only functionality to the user.
→ Abstractmethod:- does not have any implementation body only have declaration.
→ abstract class:- atleast one abstract method should be here.

```
Abstract class shape
{
Abstract public double Area();
Abstract public double perimeter();
}
class circle extents shape
{ double radius;
  circle (double radius)
  {
  this.radius = radius;
  }
  public double Area() {
return 3.14 * Radius * radius; // π r²;
}
public double perimeter() {
return 2 × π * radius; } }
class rectangle extends shape
{ int l, b;
  rectangle (int l, int b)
  { this.l = l;
    this.b = b;
```

```
}
public double area()
{
return l*b;
}
public double perimeter()
{return 2(l+b);
}}
public class abstract class{
  p.s.v.m (s.args[])
{rectangle r=new rectangle (10,20);
  s.o.p (r.area()); //200
  s.o.p (r.perimetr()); //2(10+20)
                        //=60
circle c = new circle (7.2);
s.o.p (c.area);
s.o.p (c.perimeter)}}
```

Interface in java

the interface in java is a mechenism to achieve abstraction. there can be only abstract methods in the java interface, not method body.

```
Interface
interface shape
{
double area();
double perimeter();
}
class circle implements shape
{ double radius;
  circle (double radius)
  { this.radius = radius;
  }
public double area()
{return 3.14 * radius * radius;
}
public double perimeter()
{return 2*3.14*radius;}
}
class rectangle implement shape
{
int l, b;
rectang (int l, int b)
{this.l=l;
 this.b=b;}
public double area()
 {return l*b;
}
public double perimeter()
{return 2(l+b);}}
public class interface shape
Abstract class
{ p.s.v.m (string args[])
{
rectangle r=new rectangle (10,20);
  s.o.p ("Area of rectangle "+r.area());
s.o.p ("perimeter of rectangle "+r.perimeter());
}}
```

Inter class :- nested class, It contains has a relationship A class defines with an another class

Syntax:

```
class outerclass
{
class innerclass
{
}
}
```

Ex:-

```
class car
{
class wheels
{
int wheels;
public void run()
{
s.o.p ("car has 4 wheels");
}}
public class innerclass
{p.s.v.m(s arg[]){
for creating object class
Parent class.chid classname
car.wheels cw=new car.wheel();
cw.run();
}}
```

Inner classes are two types

(1) static inner class

(2) non-static inner class

(1) S.I.C → class within class followed by static key word

ex:-

```
class outerclass
{
int p=30;
static g=40;
static class innerclass{
int k=50;
void add(){
s.o.p ("sum"+(j+k));
}}
s.o.p (innerclass.k);
innerclass.add();
public class static
inner class
{ psv.m(s.args[])
{
outr class o=new outrclass();
s.o.p(o.i);
s.o.p(outrclass.j);
}
```

② local inner class:-

local :- It belongs to particular block. {local inner class}

class :- class defined within method

Ex:-

```
class outerclass
 {
 int i =30;
 int j = 90;
void add()
 {
 S.o.p ("sum is "+(i+j);
  class innerclass {
    int k=50;
    int l=60;
    void sum()
   {
S.o.p ("sum is"+(k+l));
}}}
public class local inner class {
 p.s.v. m (s.arg[]){
 Outerclass o=new outerclass();
      o.add();
   }}
```

AWT components and Event Handlers

AWT components :-

AWT (abstract window toolkit) is java's original platform-dependent windowing, graphics and user-interface widget toolkit. Below are some key AWT components.

(1) frame :- A top-level window with a title and a border

(2) panel :- A generic container for holding AWT components

(3) Button :- A labeled button that can trigger an action when clicked.

(4) Label :- A display area for a short text string.

5) Text field :- A single-line text input field

(6) Text Area :- A multi-line text input area.

7) Check box :- A box that can be checked or unchecked

(8) choice :- A drop-down list of items

(9) List :- A list of items that allow multiple selections.

Event Handlers

In Java, event handling is the mechanism that controls the events, such as actions, mouse movements, and key presses. Key event handler interface and classes include

1. **ActionListener**: used for handling action events like button clicks.
2. **MouseListener**: used for handling mouse events
3. **KeyListener**: used for handling keyboard events.

Simple calculator program using AWT

```
import java.awt.*;                    import javax.swing.*;
import java.awt.event.*;              import swing.*;
public class SimpleCalculator extends JFrame
                                      swing
        implements.
ActionListener {
    // Declare components
JTextField input1, input2, output;
JButton addButton, subButton, mulButton, divButton;
    // constructor to set up GUI
    public SimpleCalculator() {
JLabel label1 = new JLabel ("number1:");
JLabel label2 = new JLabel ("number2:");
JLabel label3 = new JLabel ("Result");
 input1 = new JTextField (10);
 input2 = new JTextField (10);
 output = new JTextField (10);
 output.setEditable (false); // Result field should be non-editable
 addButton = new JButton ("+");
 subButton = new JButton ("-");
 mulButton = new JButton ("*");
 divButton = new JButton ("/");
 addButton.addActionListener (this);
 subButton.addActionListener (this);
 mulButton.addActionListener (this);
 divButton.addActionListener (this);
        //set layout manag. and add components
 setLayout (new FlowLayout());
 add (label1);
 add (input1);
 add (label2);
 add (input2);
 add (label3);
 add (output);
 add (addButton);
 add (subButton);
 add (mulButton);
 add (divButton);
 setTitle ("Simple Calculator");
 setSize (250, 200);          setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
 setVisible (true);
 addWindowListener (new WindowAdapter() {
 public void windowClosing (WindowEvent we) {
 System.exit(0); }});
 }
 public void actionPerformed (ActionEvent ae) {
 try {
 double num1 = Double.parseDouble (input1.getText());
 double num2 = Double.parseDouble (input2.getText());
 double result = 0;
 if (ae.getSource() == addButton) {
 result = num1 + num2; }
 else if (ae.getSource() == subButton) {
 result = num1 - num2; }
 else if (ae.getSource() == mulButton) {
 result = num1 * num2; }
 else if (ae.getSource() == divButton) {
 result = num1 / num2; }
 output.setText (String.valueOf (result)); }
 catch (NumberFormatException e) {
 output.setText ("Invalid input"); }
 catch (ArithmeticException e) {
 output.setText ("Error"); }
    public static void main (String[] args) {
 New SimpleCalculator(); }}
                                      swing
```

Unit 3

Abstract class

① Abstract class can have abstract and non-abstract methods

→ Abstract class doesn't soppart multiple inheritance

→ abstract class can have final, non-final static, and non-static variables.

→ Abstract class can provide the implementation of interface

→ The abstract keyword is used to declare abstract class

→ An abstract class can be extended using keyword "extends"

→ A java abstract class have class member like private, protected, etc

Ext

public abstract class shape {
Public abstract void draw();
}

Interface

→ Interface can have abstract methods.

→ Interface soppats multiple inheritance

Interface has only static and final variables

→ Interface can't provide the implementation of abstract class.

→ The Interface keyword is used to declare interface

→ the Interface can be implemented using keyword "implements".

→ members of a java interface are public by default.

Ext

Public interface Drawable {
void draw();
}

create a student rollno, name, course, gender, class, address with textboxes for taking input from the user (without any functionality) and checkboxes for selecting the course, radio buttons for selecting gender with appropriate background colour program

```
import java.applet.Applet;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class students extends Applet {
    public static Label L1,L2,L3,L4,L5,L6;
    public static TextField t1,t2,t3,t4;

public static JComboBox cb;
    public static void main (String[] args){
    Frame f=new Frame ("student");
        L1=new Label ("RollNo");
    L1.setBounds (50,80,50,20);
    L2=new Label ("Name");
    L2.setBounds (50,110,50,20);
    L3 = New Label ("class");
    L3.setBounds(50,140,50,20);
    L4 = new Label ("Gender");
    L4.setBounds(50,170,50,20);
    L5 = new Label ("course");
    L5.setBounds(50,200,50,20);
    L6 = new label ("Address");
    L6.setBounds(50,230,50,20);
    String course[] = {"DS","DAA","OOPS",
cb=new JComboBox (course);
cb.setBounds (140,200,80,20);

    t1=new TextField();
    t1.setBounds(140,80,100,20);
    t2 = new TextField();
    t2.setBounds(140,110,100,20);
    t3 = new TextField();
    t3.setBounds(140,230,100,20);
CheckboxGroup(); checkbox box1=new
checkbox ("012",false,cbg); checkbox box2=new
checkbox ("310",false,cbg); checkbox box3=new
checkbox ("811",false,cbg); box1.setBounds
                                (140,140,20,20);
box2.setBounds(200,140,40,20);
box3.setBounds(240,140,40,20);
CheckboxGroup cbg L=new
CheckboxGroup(); Checkbox box4=new
Checkbox ("male",false,cbg1);
 Checkbox box5=new
 Checkbox ("female",false,cbg2);
 box4.setBounds(140,170,60,20);
 box5.setBounds(200,170,60,20);
  Button b=new Button(140,280,70,30);
  L=new Label ("submitted");
  L.setBounds(140,320,60,30);
  L.setVisible(false);
```

```
b.addActionListener(new ActionListener(){
@java.lang.Override
public void actionPerformed(ActionEvent e)
{ L.setVisible(true);
}});
f.add(L1);
f.add(L2);
f.add(L3);
f.add(L4);
f.add(L5);
f.add(t1);
f.add(t2);
f.add(box1);
f.add(box2);
f.add(box3);
f.add(box4);
f.add(box5);
f.add(L5);
f.add(b);
f.add(t3);
f.add(b);
f.add(L);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
f.addWindowListener(new WindowAdapter()
{ public void windowClosing(WindowEvent e){ System.exit(0);
});}
```

output