**Threads**

1. A thread is a sub program in a program which is used to perform a task.
2. A thread a small unit of a program which is used to perform a task.
3. A thread is also called as a light weight process.

How to create a thread?

|  |
| --- |
| We can create threads in 2 ways   1. Extend the Thread class and override the run method 2. Implement Runnable interface and override the run method |

Creating a thread by extending a Thread class

class One extends Thread

{

    @Override

    public void run()

    {

        for(int i=1;i<=10;i++)

        {

            System.out.println("i="+i);

        }

    }

}

class Two extends Thread

{

    @Override

    public void run()

    {

        for(int i=100;i<=110;i++)

        {

            System.out.println("i="+i);

        }

    }

}

public class ThreadDemo1

{

    public static void main(String[] args)

    {

        One t1=new One(); //we have created a thread task: 1 to 10 print

        Two t2=new Two(); //we have created another thread task: 100 to 110 print

        //Note: don't call the run() method directly here, if you call them execution will be done sequentiallly

        //threads won't get started independently.

        //if you want to  start thread independently call the start method

        t1.start();

        t2.start();

    }

}

Output:

i=100

i=101

i=1

i=102

i=2

i=3

i=4

i=5

i=103

i=104

i=6

i=105

i=106

i=7

i=107

i=8

i=108

i=9

i=109

i=10

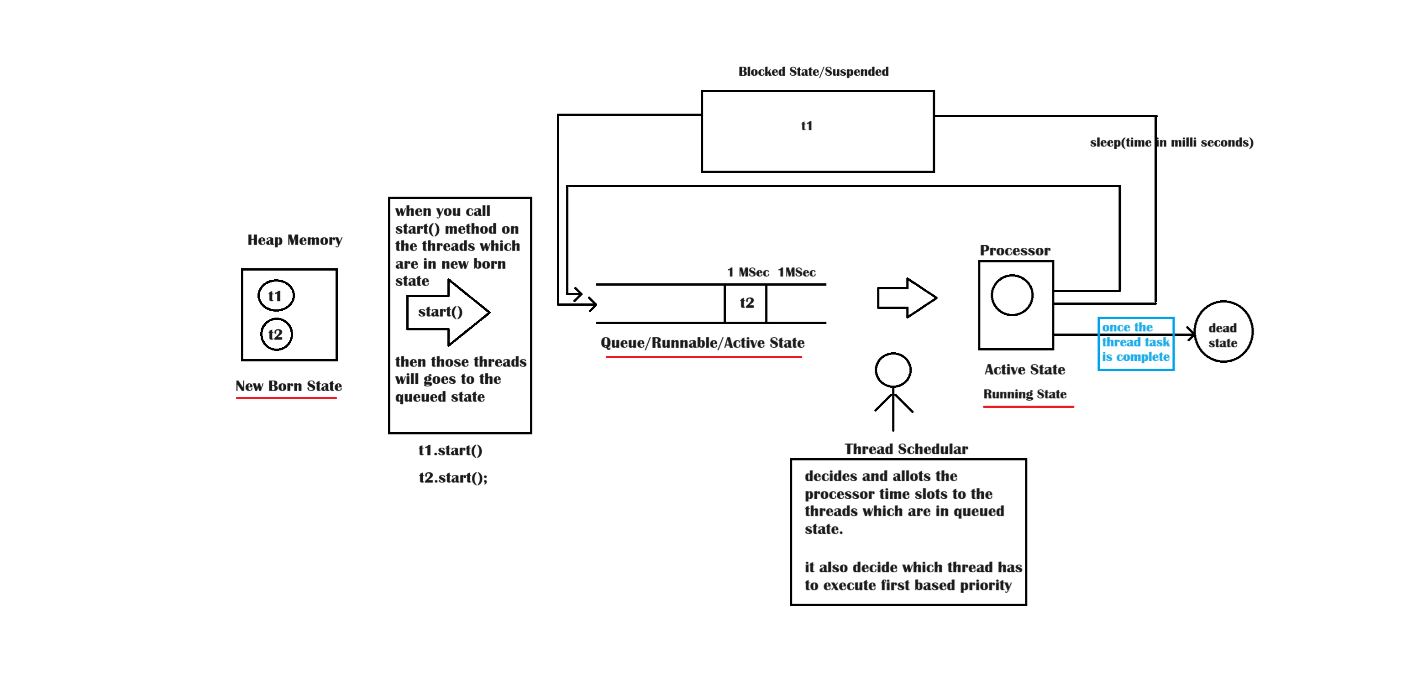
i=110

What is the default priority of a thread?

|  |
| --- |
| The default priority of a thread is 5(normal priority) |

What are the different states of a thread?

|  |
| --- |
| 1. New born state 2. Queued or Runnable state or Active 3. Running or Active 4. Blocked state 5. Dead State |



Example on suspending a thread?

|  |
| --- |
| We can suspend a thread by using the following methods   1. Sleep 2. Suspend and resume () 3. wait and notify |

Can a run() method throw InterruptedException

|  |
| --- |
| No, we can’t throw Interrupted Exception at the time of overriding run () method. Because overridden method of Runnable interface is not throwing any exception. |

class One extends Thread

{

    @Override

    public void run()

    {

        for(int i=1;i<=10;i++)

        {

            if(i==3)

            {

                try{

                    System.out.println("Nenu nidra pothunna......");

                    sleep(2000);

                }catch(InterruptedException ex)

                {

                 System.out.println("Yevudra nannu interrupt chesindeeeeeee");

                }

            }

            System.out.println(getName()+":\t"+i);

        }

    }

}

class Two extends Thread

{

    @Override

    public void run()

    {

        for(int i=100;i<=110;i++)

        {

            System.out.println(getName()+":\t"+i);

        }

    }

}

public class ThreadDemo1

{

    public static void main(String[] args)

    {

        One t1=new One(); //we have created a thread task: 1 to 10 print

        t1.setName("first");

        Two t2=new Two(); //we have created another thread task: 100 to 110 print

        t2.setName("second");

        //Note: don't call the run() method directly here, if you call them execution will be done sequentiallly

        //threads won't get started independently.

        //if you want to  start thread independently call the start method

        t1.start();

        t2.start();

        System.out.println("t1 priority:\t"+t1.getPriority());

        System.out.println("t2 priority:\t"+t2.getPriority());

    }

}

Output:

t1 priority: 5

t2 priority: 5

first: 1

first: 2

Nenu nidra pothunna......

second: 100

second: 101

second: 102

second: 103

second: 104

second: 105

second: 106

second: 107

second: 108

second: 109

second: 110

first: 3

first: 4

first: 5

first: 6

first: 7

first: 8

first: 9

first: 10

Example on setting priority

class One extends Thread

{

    @Override

    public void run()

    {

        for(int i=1;i<=10;i++)

        {

            if(i==3)

            {

                try{

                    System.out.println("Nenu nidra pothunna......");

                    sleep(2000);

                }catch(InterruptedException ex)

                {

                 System.out.println("Yevudra nannu interrupt chesindeeeeeee");

                }

            }

            System.out.println(getName()+":\t"+i);

        }

    }

}

class Two extends Thread

{

    @Override

    public void run()

    {

        for(int i=100;i<=110;i++)

        {

            System.out.println(getName()+":\t"+i);

        }

    }

}

public class ThreadDemo1

{

    public static void main(String[] args)

    {

        One t1=new One(); //we have created a thread task: 1 to 10 print

        t1.setName("first");

        Two t2=new Two(); //we have created another thread task: 100 to 110 print

        t2.setName("second");

        //Note: don't call the run() method directly here, if you call them execution will be done sequentiallly

        //threads won't get started independently.

        //if you want to  start thread independently call the start method

        t1.setPriority(Thread.MAX\_PRIORITY); //10

        t2.setPriority(Thread.MIN\_PRIORITY); //1

        t1.start();

        t2.start();

        System.out.println("t1 priority:\t"+t1.getPriority());

        System.out.println("t2 priority:\t"+t2.getPriority());

    }

}

Join() method of Thread class

|  |
| --- |
| Join() method makes the parent thread to wait until the joined thread completes it’s task. |

class One extends Thread

{

    @Override

    public void run()

    {   for(int i=1;i<=10;i++)

        {   System.out.println(getName()+":\t"+i);

        }

    }

}

class Two extends Thread

{

    @Override

    public void run()

    {

        for(int i=100;i<=110;i++)

        {

            System.out.println(getName()+":\t"+i);

        }

    }

}

public class ThreadDemo3

{

    public static void main(String[] args) throws InterruptedException

    {

        Thread main=Thread.currentThread();

        System.out.println(main.getName()+" thread started...");

        //main.sleep(1000);

        One t1=new One();

        Two t2=new Two();

        t1.start();t2.start();

        t1.join();

        //t2.join();

        System.out.println(main.getName()+" thread task ends here...");

    }

}

Output:

main thread started...

Thread-1: 100

Thread-1: 101

Thread-0: 1

Thread-1: 102

Thread-0: 2

Thread-1: 103

Thread-0: 3

Thread-0: 4

Thread-1: 104

Thread-0: 5

Thread-0: 6

Thread-0: 7

Thread-0: 8

Thread-0: 9

Thread-0: 10

Thread-1: 105

Thread-1: 106

main thread task ends here...

Thread-1: 107

Thread-1: 108

Thread-1: 109

Thread-1: 110

Example on creation of thread in second way

|  |
| --- |
| import javax.swing.JFrame;  class One extends JFrame implements Runnable  {      public void run()      {          for(int i=1;i<=10;i++)          {   System.out.println(i);          }      }  }  public class SecondWay  {      public static void main(String[] args)      {          One o=new One();          Thread t=new Thread(o);//thread is created          t.start();      }  }  Output:  1  2  3  4  5  6  7  8  9  10 |

**Synchronization**

|  |
| --- |
| If an object is synchronized that object can’t be accessed by more-than one thread at a time. If one thread accessing the remaining threads has to wait.  Synchronized objects maintains a key that can be obtained by a thread which enters first. |

We can implement synchronization in 2 ways

1. by declaring methods as synchronized
2. by using synchronized block

class Monitor

{

    synchronized void display()

    {

        Thread t=Thread.currentThread();

        System.out.print(t.getName()+" Thread is entered into the monitor....[");

        for(int i=1;i<=10;i++)

        {

            System.out.print(i+",");

        }

        System.out.print("]\n");

    }

}

class One extends Thread

{

    Monitor monitor;

    One(Monitor monitor)

    {this.monitor=monitor;

    }

    public void run()

    {

        monitor.display();

    }

}

class Two extends Thread

{

    Monitor monitor;

    Two(Monitor monitor)

    {   this.monitor=monitor;

    }

    public void run()

    {

        monitor.display();

    }

}

public class Synchronization

{

    public static void main(String[] args) {

        Monitor monitor=new Monitor();

        One t1=new One(monitor);

        Two t2=new Two(monitor);

        t1.start();

        t2.start();

    }

}

**Implementing synchronization using synchronized block**class Monitor

{

   void display()

    {

        Thread t=Thread.currentThread();

        System.out.print(t.getName()+" Thread is entered into the monitor....[");

        for(int i=1;i<=10;i++)

        {

            System.out.print(i+",");

        }

        System.out.print("]\n");

    }

}

class One extends Thread

{

    Monitor monitor;

    One(Monitor monitor)

    {this.monitor=monitor;

    }

    public void run()

    {

        synchronized(monitor){

            monitor.display();

        }

    }

}

class Two extends Thread

{

    Monitor monitor;

    Two(Monitor monitor)

    {   this.monitor=monitor;

    }

    public void run()

    {

        synchronized(monitor){

            monitor.display();

        }

    }

}

public class Synchronization

{

    public static void main(String[] args) {

        Monitor monitor=new Monitor();

        One t1=new One(monitor);

        Two t2=new Two(monitor);

        t1.start();

        t2.start();

    }

}

**Another Example on synchronized block (where t1 and t2 threads can’t access monitor object at a time but t3 can access simultaneously)**

class Monitor

{

   void display()

    {

        Thread t=Thread.currentThread();

        System.out.print(t.getName()+" Thread is entered into the monitor....[");

        for(int i=1;i<=10;i++)

        {

            System.out.print(i+",");

        }

        System.out.print("]\n");

    }

}

class One extends Thread

{

    Monitor monitor;

    One(Monitor monitor)

    {this.monitor=monitor;

    }

    public void run()

    {

        synchronized(monitor){

            monitor.display();

        }

    }

}

class Two extends Thread

{

    Monitor monitor;

    Two(Monitor monitor)

    {   this.monitor=monitor;

    }

    public void run()

    {

        synchronized(monitor){

            monitor.display();

        }

    }

}

class Three extends Thread

{

    Monitor monitor;

    Three(Monitor monitor)

    {   this.monitor=monitor;

    }

    public void run()

    {   monitor.display();

    }

}

public class Synchronization

{

    public static void main(String[] args) {

        Monitor monitor=new Monitor();

        One t1=new One(monitor);

        Two t2=new Two(monitor);

        Thread t3=new Three(monitor);

        t1.start();

        t2.start();

        t3.start();

    }

}

Output:

hread-0 Thread is entered into the monitor....[Thread-2 Thread is entered into the monitor....[1,2,3,4,5,6,7,8,9,10,]

1,2,3,4,5,6,7,8,9,10,]

Thread-1 Thread is entered into the monitor....[1,2,3,4,5,6,7,8,9,10,]

**Wait() method : it can suspend a thread and also make the thread to release the key**

**Example on usage of wait() method**

class Monitor

{

    synchronized void display()

    {

        Thread t=Thread.currentThread();

        System.out.print(t.getName()+" Thread is entered into the monitor....[");

        for(int i=1;i<=10;i++)

        {

            if(i==5)

            {

                System.out.println(t.getName()+" Thread is released the key ");

                try{

                    wait(1000);

                }catch(InterruptedException ex)

                {

                    ex.printStackTrace();

                }

            }

            System.out.print(i+",");

        }

        System.out.print("]\n");

    }

}

class One extends Thread

{

    Monitor monitor;

    One(Monitor monitor)

    {this.monitor=monitor;

    }

    public void run()

    {

        monitor.display();

    }

}

class Two extends Thread

{

    Monitor monitor;

    Two(Monitor monitor)

    {   this.monitor=monitor;

    }

    public void run()

    {

        monitor.display();

    }

}

public class Synchronization

{

    public static void main(String[] args) {

        Monitor monitor=new Monitor();

        One t1=new One(monitor);

        Two t2=new Two(monitor);

        t1.start();

        t2.start();

    }

}

Output:

Thread-0 Thread is entered into the monitor....[1,2,3,4,Thread-0 Thread is released the key

Thread-1 Thread is entered into the monitor....[1,2,3,4,Thread-1 Thread is released the key

5,6,7,8,9,10,]

5,6,7,8,9,10,]

**Concepts we have covered till now**

* + 1. **Introduction**
    2. **Features**
    3. **Jdk installations**
    4. **VSCode installation**
    5. **JCL**
    6. **Java API**
    7. **Escape sequences**
    8. **Naming conventions**
    9. **Compilation**
    10. **Execution**
    11. **Data types**
    12. **If**
    13. **If..else**
    14. **If..else..if**
    15. **Nested if**
    16. **Switch**
    17. **Array**
    18. **1D arrays**
    19. **2d arrays**
    20. **3d arrays**
    21. **Jagged arrays**
    22. **For loop**
    23. **For each**
    24. **While**
    25. **Do..while..**
    26. **Functions(methods)**
    27. **Class**
    28. **Object**
    29. **Instance variables**
    30. **Static variables**
    31. **Instance methods**
    32. **Static methods**
    33. **Instance block**
    34. **Static block**
    35. **Constructors**
    36. **This keyword**
    37. **String constant pool**
    38. **Accessor and mutator methods**
    39. **Inheritance**
    40. **Super()**
    41. **Method overloading**
    42. **Method overriding**
    43. **Upcasting & Down casting**
    44. **Static and dynamic binding**
    45. **Compile time and runtime Polymorphism**
    46. **Abstract classes**
    47. **Interfaces**
    48. **Inner classes**
    49. **Anonymous inner classes**
    50. **Functional interfaces**
    51. **Lambda expressions**
    52. **Singleton**
    53. **Immutable object**
    54. **Creating record using contextual keyword called ‘record’**
    55. **Exception Handling**
    56. **Try**
    57. **Catch**
    58. **Finally**
    59. **Throw**
    60. **Throws**
    61. **Nested try**
    62. **Types of exceptions**
    63. **User defined exceptions**
    64. **Mult catch exceptions**
    65. **IOStreams**
    66. **Types of streams**
    67. **Serialization**
    68. **Deserialization**
    69. **Threads introduction**