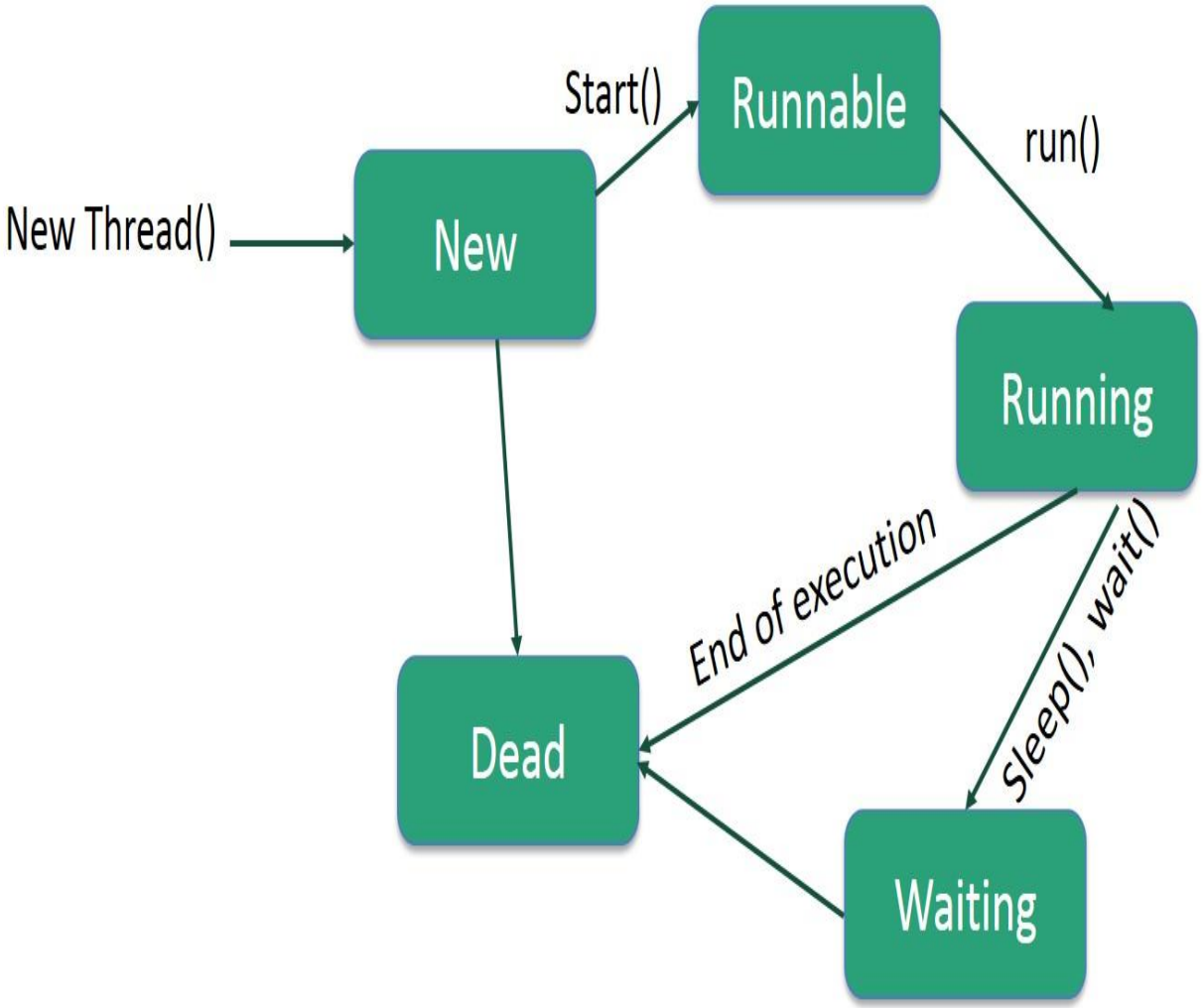
MULTITHREADING

Life cycle of a Thread



1) New

The thread is in new state if an instance of Thread class is created but before the invocation of start() method.

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated

A thread is in terminated or dead state when its run() method exits.

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.

Commonly used methods of Thread class:

- public void run(): is used to perform action for a thread.
- public void start(): starts the execution of the thread. JVM calls the run() method on the thread.
- public void sleep(long miliseconds): Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
- public void join(): waits for a thread to die.
- public void join(long miliseconds): waits for a thread to die for the specified miliseconds.

Thread creation in Java

There are two ways to create a thread:

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

- **Extending** the java.lang.Thread class
- **Implementing** the java.lang.Runnable Interface

The Thread and Runnable are available in the java.lang.* package

1) By extending thread class

- The class should extend Java Thread class.
- The class should override the run() method.
- The functionality that is expected by the Thread to be executed is written in the run() method.
- void start():

  Creates a new thread and makes it runnable.

void run():

  The new thread begins its life inside this method.

METHOD 1

```
class MyThread extends Thread {
    public void run() {
    System.out.println(" this thread is running ... ");
    }
    }
    class ThreadEx1 {
    public static void main(String [] args ) {
    MyThread t = new MyThread();
    t.start();
    }
    }
```

2) By Implementing Runnable interface

- The class should implement the 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().

public void run()

- The class should implement the run() method in the Runnable interface
- The functionality that is expected by the Thread to be executed is put in the run() method

METHOD 2

```
class MyRunnable implements Runnable {
    public void run() {
            System.out.println(Thread.currentThread().getName() );
            }
}
public class Main {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread myThread = new Thread(myRunnable);
        myThread.start();

        }
}
```

OUTPUT

```
C:\personal>javac Main.java

C:\personal>java Main
Thread-0
```

EXAMPLE 1 (Sleep method)

```
class MyRunnable implements Runnable {

    public void run() {
        try {
            for(int i = 0; i < 5; i++) {
            System.out.println("Child Thread: " + i);
            Thread.sleep(500);
            }
            } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
            }
         }
    }
```

```
public class Main {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread myThread = new Thread(myRunnable);
        myThread.start();
       try {
            for(int i = 0; i < 5; i++) {
            System.out.println("Main Thread: " + i);
            Thread.sleep(1000);
            }
       } catch (InterruptedException e) {
        System.out.println("Main thread interrupted.");
       }
    }
}
```

OUTPUT

```
C:\personal>javac Thread2.java

C:\personal>java Thread2
Child Thread: 0
Main Thread: 0
Child Thread: 1
Main Thread: 1
Child Thread: 2
Child Thread: 3
Main Thread: 2
Child Thread: 4
Main Thread: 3
Main Thread: 4
```

THREAD PRIORITY

Thread priorities are the integers which decide how one thread should be treated with respect to the others.

• Thread priority decides when to switch from one running thread to another, process is called **context switching**

• A thread can voluntarily release control and the highest priority thread that is ready to run is given the CPU.

In place of defining the priority in integers, we can use

- MIN_PRIORITY
- NORM_PRIORITY
- MAX_PRIORITY

EXAMPLE (Priority)

```
class EvenThread extends Thread {
public void run()
{
for(int i=0;i<10;i++)
{
    if(i%2==0)
    {
    System.out.println(i);
    }
   }
}
}
class OddThread extends Thread {
public void run()
{
for(int i=0;i<10;i++)
{
    if((i%2)!=0)
    {
    System.out.println(i);
    }
   }
}
}
```

```
public class Multithread4 {
public static void main(String args[])
{
 // create a new thread
 Thread t1=new EvenThread();
 t1.start();
 t1.setPriority(2);
 Thread t2=new OddThread();
 t2.start();
 t2.setPriority(9);
 System.out.println("Priority of thread1=" +
t1.getPriority());
 System.out.println("Priority of thread2=" +
t2.getPriority());
 }
}
```

Output

```
[Running] cd "d:\subashini new\kongu\OOPS\pgms\java pgms\" && javac
Multithread4.java && java Multithread4
0
2
4
6
8
1
3
5
7
9
Priority of thread1=2
Priority of thread2=9
```

Synchronization in Java

- Concurrently running threads share data and two threads try to do operations on the **same variables at the same time.**
- This often results in corrupt data as two threads try to operate on the same data.
- By using the synchronize only one thread can access the method at a time and
- A second call will be blocked until the first call returns or wait() is called inside the synchronized method.

Why use Synchronization

- To prevent thread interference.
- To prevent consistency problem.

EXAMPLE 1

```
class Updateaccount  {
   int a=100;
   synchronized void update(int b)
   {
      for(int i=0;i<3;i++)
      {
      a=a+b;
      System.out.println(a);
      try{
      Thread.sleep(5000);
      }
      catch(InterruptedException e)
      {

      }

      }
   }
}
```

```
class SThread {
public static void main(String args[])
{
 // create a new thread
 Updateaccount obj= new Updateaccount();
 Thread t1=new Thread()
 {
   public void run()
   {
      obj.update(10);
   }
 };
 Thread t2=new Thread()
 {
   public void run()
   {
      obj.update(100);
   }
 };
 t1.start();
 t2.start();
 }
}
```

OUTPUT

```
D:\subashini new\kongu\JP\pgms>javac SThread.java

D:\subashini new\kongu\JP\pgms>java SThread
210
210
310
310
410
410

D:\subashini new\kongu\JP\pgms>javac SThread.java

D:\subashini new\kongu\JP\pgms>java SThread
110
120
130
230
330
430

D:\subashini new\kongu\JP\pgms>javac SThread.java
```

Without synchronisation

With synchronisation

EXAMPLE 2

```
class Table{
synchronized void printTable(int n){ //method 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{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

```
class Thread6_ntsync{
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: synchronized keyword USED

5
10
15
20
25
100
200
300
400
500

Output: (synchronized NOT Used)

5
100
10
200
15
300
20
400
25
500

MULTITHREADING

```
public class ThreadExample {
  public static void main(String[] args) {
       HelloTask thread1 = new HelloTask();
     WearMaskTask thread2 = new WearMaskTask();
     UseSanitizerTask thread3 = new UseSanitizerTask();
      thread1.start();
     thread2.start();
     thread3.start();
  } }
class HelloTask extends Thread {
  public void run() {
     while (true) {
        System.out.println("Hello!");
        try {
           Thread.sleep(1000); // Sleep for 1 second
        } catch (InterruptedException e) {
           System.out.println("Thread Interrupted");
        }      }   }  }
```

```
class UseSanitizerTask extends Thread {
  public void run() {
     while (true) {
        System.out.println("Use Sanitizer!");
        try {
           Thread.sleep(5000); // Sleep for 5 seconds
        }
         catch (InterruptedException e) {
            System.out.println("Thread Interrupted");
        }      }
  }  }
```