

MultiThreading

Note: Before starting Multithreading we should have to know

What is thread?

Thread is a sub part of the process or it is a light weight process.

Q. What is the process?

Process is a running program in ram called as process or program in execution called as process.

Q. What is multi threading?

Multi threading means executing more than one thread simultaneously or waiting for each other called multithreading or to utilize an idle period of time called multi threading.

Note: Threading is a concept of an operating system.

Java is a language which provides implementation to thread concepts.

How to implement thread using JAVA

If we want to implement thread using java we have two ways

1. Using Thread class
2. Using Runnable interface

How to implement thread using Thread class

Steps

-
1. **Add java.lang package in application**
Note: java.lang is a default package java means we do not need to import it.
 2. **Create a user defined class and inherit the Thread class in it.**
Thread class is a member of java.lang package which contain methods or provide Thread implementation feature to us

```
class ABC extends Thread
{
}
```

Note: we inherit all Thread properties in class ABC means we can say ABC is also thread.

3. **Override run() method for writing thread logics**

run() method is used for writing the thread logics means when developer writing logic which work as thread or execute when thread start as well as JVM execute it simultaneously with other threads then we required logic in run() method

```

class ABC extends Thread
{
    public void run()
    {
        write here your thread logics
    }
}

```

Note: run() is not a method of Thread class internally it is method of Runnable interface and Thread is a implementer class of Runnable interface
Runnable is functional interface as well as it is marker interface in JAVA

```

interface Runnable
{
    public void run();
}
class Thread implements Runnable
{
    public void run(){
    }
}
class ABC extends Thread
{
    public void run()
    {
    }
}

```

Q. What is the functional interface in JAVA?

Functional interface means interface containing only one abstract method called Functional interface.

Q. What is the marker interface in JAVA?

Marker interfaces means interface may not contain any method or may contain method but java provides a special runtime environment to them for execution called as marker interface.

4. Create object of Thread child class and use Thread class method for manage the thread

void start(): this method is used for start the thread means when we call start() method then internally run() method get executed

Note: in threading we required to call run() method manually it is internally executed when we call start() method

public static void sleep(int milliseconds) throws InterruptedException : this method is used to hold the thread execution for a specified time period and again re-execute after some specified time period.

public void stop(): this method is used for terminate the thread or destroy the thread

public void join() throws InterruptedException : this method is used to hold the thread execution whenever the running thread is not completed.

public boolean isAlive(): this method is used to check if the thread is running or not if running return true otherwise return false.

public void setPriority(int priority): this method help us to set the priority of thread

public int getPriority(): this method returns the current priority of thread and every thread has default normal priority.

void setDaemon(boolean): this method can set thread as daemon thread when we pass true value in it

boolean isDaemon(): this method checks the working thread as daemon or not if working is daemon thread then returns true otherwise returns false.

Object class method for manage the thread

void wait(int milliseconds): this method can set conditional wait with thread

void wait(): this method can set unconditional wait with thread.

void notify(): this method call waits thread one by one single at time in Queue format means first in first out format.

void notifyAll(): this method calls all waited threads at time in the first out format.

Example with two threads and execute in waiting of each other

class F extends Thread

```
{
    public void run()
    {
        try{
            for(int i=1; i<=5; i++)
            { System.out.println("First Thread is "+i);
              Thread.sleep(10000);
            }
        }
        catch(InterruptedException ex)
        { System.out.println("Error is "+ex);
        }
    }
}
```

class S extends Thread

```
{ public void run()
    { try{
        for(int i=1; i<=50; i++)
        { System.out.println("Second Thread is "+i);
          Thread.sleep(1000);
        }
    }
    catch(InterruptedException ex)
    { System.out.println("Error is "+ex);
    }
  }
}
```

public class ThreadProApp

```
{
```

```

public static void main(String x[])
{
    F f1 = new F();
    f1.start();
    S s1 = new S();
    s1.start();
}
}

```

Example: Suppose consider we have Two threads and first thread has 10 second waiting period and second thread has 1 second waiting period so by default JVM execute first thread and second in waiting of first and first in waiting of second but we want to execute second thread when first thread is finish even first thread waiting period.

Note: for implement above approach we can use join() method with first thread means call join() method with first thread after start() method of First thread

Example with source code

```

class F extends Thread
{
    public void run()
    {
        try{
            for(int i=1; i<=5; i++)
            {
                System.out.println("First Thread is "+i);
                Thread.sleep(10000);
            }
        }
        catch(InterruptedException ex)
        {
            System.out.println("Error is "+ex);
        }
    }
}

class S extends Thread
{
    public void run()
    {
        try{
            for(int i=1; i<=50; i++)
            {
                System.out.println("Second Thread is "+i);
                Thread.sleep(1000);
            }
        }
        catch(InterruptedException ex)
        {
            System.out.println("Error is "+ex);
        }
    }
}

public class ThreadProApp
{
    public static void main(String x[])throws InterruptedException
    {
        F f1 = new F();
        f1.start();
        f1.join();
        S s1 = new S();
    }
}

```

```

        s1.start();
    }
}

```

Example: we want to execute the first thread 3 times and after terminate it. Note we can use the stop() method to terminate or destroy the thread.

class F extends Thread

```

{
    public void run()
    {
        try{
            for(int i=1; i<=5; i++)
            { System.out.println("First Thread is " +i);
              if(i==3)
              {stop();
              }
              Thread.sleep(10000);
            }
        }
        catch(InterruptedException ex)
        { System.out.println("Error is " +ex);
        }
    }
}

class S extends Thread
{ public void run()
  { try{
      for(int i=1; i<=50; i++)
      { System.out.println("Second Thread is " +i);
        Thread.sleep(1000);
      }
    }
    catch(InterruptedException ex)
    { System.out.println("Error is " +ex);
    }
  }
}

```

```

public class ThreadProApp
{
    public static void main(String x[])throws InterruptedException
    { F f1 = new F();
      f1.start();
      f1.join();
      S s1 = new S();
      s1.start();
    }
}

```

Synchronization and Asynchronization in Threading

```

class Table
{
    public void showTable(int x)
    {
        try{
            for(int i=1; i<=10; i++)
            { System.out.printf("%d X %d = %d\n",i,x,i*x);
              Thread.sleep(1000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
}

class Two extends Thread
{
    Table table;
    void setTable(Table table)
    { this.table=table;
    }
    public void run()
    { table.showTable(2);
    }
}

class Three extends Thread
{
    Table table;

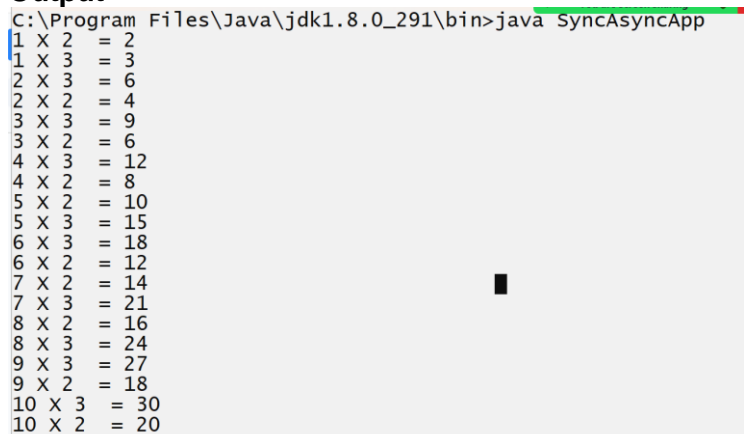
```

```

void setTable(Table table)
{ this.table=table;
}
public void run()
{ table.showTable(3);
}
}
public class SyncAsyncApp
{
    public static void main(String x[])
    {
        Table t1 = new Table();
        Two tw =new Two();
        tw.setTable(t1);
        tw.start();
        Three th = new Three();
        th.setTable(t1);
        th.start();
    }
}

```

Output



```

C:\Program Files\Java\jdk1.8.0_291\bin>java SyncAsyncApp
1 X 2 = 2
1 X 3 = 3
2 X 3 = 6
2 X 2 = 4
3 X 3 = 9
3 X 2 = 6
4 X 3 = 12
4 X 2 = 8
5 X 2 = 10
5 X 3 = 15
6 X 3 = 18
6 X 2 = 12
7 X 2 = 14
7 X 3 = 21
8 X 2 = 16
8 X 3 = 24
9 X 3 = 27
9 X 2 = 18
10 X 3 = 30
10 X 2 = 20

```

If we want to mark your object or resource synchronized we have synchronized keyword

Example of synchronization

```

class Table
{
    public synchronized void showTable(int x)
    {
        try{
            for(int i=1; i<=10; i++)
            { System.out.printf("%d X %d = %d\n",i,x,i*x);
              Thread.sleep(1000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
}
class Two extends Thread
{

```

```

    Table table;
    void setTable(Table table)
    { this.table=table;
    }
    public void run()
    { table.showTable(2);
    }
}
class Three extends Thread
{
    Table table;
    void setTable(Table table)
    { this.table=table;
    }
    public void run()
    { table.showTable(3);
    }
}
public class SyncAsyncApp
{
    public static void main(String x[])
    {
        Table t1 = new Table();
        Two tw =new Two();
        tw.setTable(t1);
        tw.start();
        Three th = new Three();
        th.setTable(t1);
        th.start();
    }
}

```

wait(),notify() and notifyAll()

wait(): wait() method is used for hold thread execution for specified time period and it is method of object class internally

There are two types of wait method

Conditional wait : conditional wait means if a thread holds its execution for a specified time period and re-execute itself after some time period called conditional wait.

Syntax: void wait(int milliseconds): this is your conditional wait

Unconditional wait: unconditional wait means if thread hold its execution and not re-execute self after specified time period called as unconditional wait and if thread is in unconditional wait we required to provide notification to thread for re-execution purpose and for that we have two methods

Syntax: void wait(): unconditional wait means not specify time period for re-execution purpose.

Note: if we want to recall l unconditional waiting threads for re-execution we have notify() and notifyAll() methods.

void notify() : notify() method help us to call only one thread at time for re-execution purpose and call waited thread in FIFO order means using first in first out format

void notifyAll(): notifyAll() method helps to call all waiting threads at time.

Example with source code

```
import java.util.*;
class Table
{
    public synchronized void showTable(int x)
    {
        try{
            for(int i=1; i<=10; i++)
            { System.out.printf("%d X %d = %d\n",i,x,i*x);
              if(i==5)
                  { wait(); //waiting for 1 minute
                  }
              Thread.sleep(1000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
    public synchronized void recallThread(){
        try
        {
            notifyAll();
        }
        catch(Exception ex)
        {
            System.out.println("Error is "+ex);
        }
    }
}
class Two extends Thread
{
    Table table;
    void setTable(Table table)
    { this.table=table;
    }
    public void run()
    { table.showTable(2);
    }
}
class Three extends Thread
{
    Table table;
    void setTable(Table table)
    { this.table=table;
    }
    public void run()
    { table.showTable(3);
    }
}
```

```

}
public class SyncAsyncApp
{
    public static void main(String x[])
    {
        Table t1 = new Table();
        Two tw =new Two();
        tw.setTable(t1);
        tw.start();
        Three th = new Three();
        th.setTable(t1);
        th.start();

        do{
            Scanner xyz = new Scanner(System.in);
            String msg=xyz.nextLine();
            if(msg.equals("restart"))
            {
                t1.recallThread();
            }
            if(msg.equals("stop"))
            { System.exit(0);
            }
        }while(true); //infinite loop
    }
}

```

```

C:\Program Files\Java\jdk1.8.0_291\bin>java SyncAsyncApp
1 X 2 = 2
2 X 2 = 4
3 X 2 = 6
4 X 2 = 8
5 X 2 = 10
1 X 3 = 3
2 X 3 = 6
3 X 3 = 9
4 X 3 = 12
5 X 3 = 15
restart
6 X 3 = 18
7 X 3 = 21
8 X 3 = 24
9 X 3 = 27
10 X 3 = 30
6 X 2 = 12
7 X 2 = 14
8 X 2 = 16
9 X 2 = 18
10 X 2 = 20

```

Q. What is the difference between wait() and sleep() method?

1. The wait() method can work in synchronized block only and sleep() can work in synchronized as well as asynchronous block.
2. wait() is method of Object class and sleep() is static method of Thread class
3. wait() may be conditional or unconditional but sleep always works with conditional wait

Q. What is the difference between notify() and notifyAll()?

notify() method can call waited thread but call only one thread at time in FIFO order and notifyAll() can call waited thread but call all threads at time in FIFO order.

Q. Can we use notify() and notifyAll() in an asynchronous block?

No if we want to call notify() and notifyAll() must be call in synchronised block otherwise we get exception IllegalMonitorException

Q. What are the similarities between notify() and notifyAll() methods?

1. Both are member of Object class
2. Both can work with only synchronised block
3. Both methods are used to call unconditional waiting thread.

How to create Thread using a Runnable interface

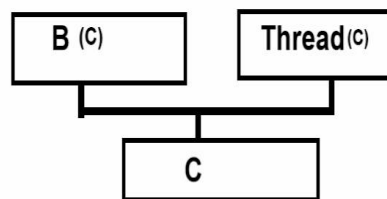
Q. Why do we need to create a thread using Runnable interface?

Some times we have situations of Multiple inheritance in thread implementation then we cannot use Thread class for thread implementation because multiple inheritance situations cannot be handled by more than one classes just it is handled by interface.

Example: Suppose we have class C and C has parent class name as B and we want to create C class as thread class then we cannot inherit Thread class in C class.

If we want to create a Thread using a Runnable interface we have the following steps.

```
class B
{
}
class C extends B,Thread
{
}
```



Note: if we think about code implementation of thread we have two parent classes name as B and Thread and we try to inherit these parent classes in child class C But according to rule of JAVA we cannot inherit multiple parent classes in single child class so this situation say implement thread implementation by using multiple inheritance technique so we have option name as interface for that purpose java provide one interface to us known as Runnable interface for achieve multiple inheritance.

How to implement thread using Runnable interface

Steps to implement thread using Runnable interface

- a. **Add java.lang package in application**

```
import java.lang.*;
```

- b. **Create class and implements Runnable interface in it**
c. **Override its run() method and write its logic**

```
class ABC implements Runnable
{
    public void run()
    {
        try{
            for(int i=1; i<=5; i++)
            {
                System.out.printf("I = %d\n",i);
                Thread.sleep(10000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
}
```

- d. **Create object of Runnable implement class**

```
ABC a1 = new ABC();
```

- e. **Create object of Thread class and pass Runnable implementer reference in it**

If we want to use thread using Runnable interface then we have to use Thread using the following constructor.

Thread(Runnable): if we think about this constructor it contain Runnable interface as parameter means here we can pass reference of Runnable using upcasting as well as we can pass reference any child of Runnable

Example with source code

```
class ABC implements Runnable
{
    public void run()
    {
        try{
            for(int i=1; i<=5; i++)
            {
                System.out.printf("I = %d\n",i);
                Thread.sleep(10000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
}
```

```

    }
}
}
public class TestRunApp
{
    public static void main(String x[])
    {
        ABC a = new ABC();
        Thread t = new Thread(a);
    }
}

```

- f. **Use thread method for work the thread**
Using thread class reference we can use Thread class method for manage thread like as start() etc

Thread Priority

```

class ABC implements Runnable
{
    public void run()
    {
        try{
            for(int i=1; i<=5; i++)
            {
                System.out.printf("I = %d\n",i);
                Thread.sleep(10000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
}
public class TestRunApp
{
    public static void main(String x[])
    {
        ABC a = new ABC();
        Thread t = new Thread(a);
        t.start();
    }
}

```

Example: Consider we have Two robots and we share some task them and we want to robots should perform task simultaneously in waiting of each other

```

class Box
{ private int boxId;

```

```

private String boxName;
public Box(){
    boxId=1;
}
public void setBoxId(int id)
{ this.boxId=id;
}
public int getBoxId(){
    return boxId;
}
public void setBoxName(String boxName)
{ this.boxName=boxName;
}
public String getBoxName()
{ return boxName;
}
public synchronized void moveBox(String roboName)
{try{
    int firstRoboBox=1,secondRoboBox=2;
    for(int i=1; i<=5; i++)
        {   if(roboName.equals("First"))
            {
                System.out.println(boxName+" "+(firstRoboBox)+" --->" +roboName);
                firstRoboBox=firstRoboBox+2;
            }
            else{
                System.out.println(boxName+" "+(secondRoboBox)+" --->" +roboName);
                secondRoboBox=secondRoboBox+2;
            }
            Thread.sleep(1000);
        }
    }
catch(Exception ex)
{ System.out.println("Error is "+ex);
}
}
}
class RoboFirst implements Runnable
{   Box box;
    private String roboName;
    int boxId=1;
    void setBox(Box box,String roboName)
    { this.box=box;
      this.roboName=roboName;
    }
    public void run()
    {
        box.moveBox(roboName );
        ++boxId;
    }
}
class RoboSecond implements Runnable
{ Box box;
    private String roboName;

```

```

void setBox(Box box,String roboName)
{ this.box=box;
  this.roboName=roboName;
}
public void run()
{
    box.moveBox(roboName);
}
}
public class RoboApp
{
    public static void main(String x[])
    {
        Box b = new Box();
        b.setBoxName("Box");
        RoboFirst rf=new RoboFirst();
        rf.setBox(b,"First");

        RoboSecond rs = new RoboSecond();
        rs.setBox(b,"Second");

        Thread first=new Thread(rf);
        first.start();

        Thread second = new Thread(rs);
        second.start();

    }
}

```

Thread Priority and daemon threads

Q. What is thread priority?

Thread priority is a concept to decide the execution or thread starting priority means when we have multiple threads and if we want to know when and which thread should be started then thread priority comes into picture.

There are three types of thread priority?

1. Max Priority
2. Min Priority
3. Norm Priority

If we want to work with thread priority we have some integer constant provided by Thread class and these constant values are public static in thread class

```

class Thread
{
    public static final int MAX_PRIORITY=10;
    public static final int MIN_PRIORITY=1;
    public static final int NORM_PRIORITY=5;
}

```

So you can access thread priority by using class name because all priority constant variables are static

```
public class PriorityApp
{
    public static void main(String x[])
    {
        System.out.println("Max priority  "+Thread.MAX_PRIORITY);
        System.out.println("MIN priority  "+Thread.MIN_PRIORITY);
        System.out.println("NORMAL priority  "+Thread.NORM_PRIORITY);
    }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac PriorityApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java PriorityApp
Max priority  10
MIN priority  1
NORMAL priority  5

C:\Program Files\Java\jdk1.8.0_291\bin>
```

If we want to set priority on thread we have following method

void setPriority(int priority): this function is used for set the thread priority

int getPriority(): this function is used for return priority of thread or current priority of thread.

Note: windows operating system is not support to thread priority of java properly

Example with source code

class A extends Thread

```
{
    public void run()
    {
        try
        {
            for(int i=1; i<=5; i++)
            { System.out.printf("Thread First = %d\n",i);
              Thread.sleep(1000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
}
```

class B extends Thread

```
{
    public void run()
    {
        try
        {
            for(int i=1; i<=5; i++)
            { System.out.printf("Thread Second = %d\n",i);
              Thread.sleep(1000);
            }
        }
        catch(Exception ex)
        { System.out.println("Error is "+ex);
        }
    }
}
```



```

    }
}
public class PriorityApp
{
    public static void main(String x[])
    {
        A a1 = new A();
        B b1 = new B();
        b1.setPriority(Thread.MAX_PRIORITY);
        a1.setPriority(Thread.MIN_PRIORITY);
        a1.start();
        b1.start();
    }
}

```

Note: every thread has a default priority known as normal priority

Q. Can we create an application or java program without thread?

No, we cannot create any java program without thread because every java program contains the default thread name as main thread and it is started by JVM and the default priority of main thread is normal priority.

```

public class TestMainThreadApp
{
    public static void main(String x[])
    {
        Thread t = Thread.currentThread();
        String name = t.getName();
        System.out.println("Current thread name is " + name);
        int priority = t.getPriority();
        System.out.println("Thread priority is " + priority);
    }
}

```

C:\Program Files\Java\jdk1.8.0_291\bin>javac TestMainThreadApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java TestMainThreadApp

Current thread name is main

Thread priority is 5

C:\Program Files\Java\jdk1.8.0_291\bin>

Q. What is daemon thread in JAVA?

Daemon thread means thread which executes in the background to provide service to another thread called a daemon thread.

Means in short we can say background working thread known as daemon thread in JAVA.

Example: Garbage collector is daemon thread in JAVA

Q. Can a user own thread as daemon thread?

Yes user can create own thread as daemon thread and for that user has following methods

void setDaemon(boolean): this method helps us to set user thread as daemon thread when we pass true value in it

boolean isDaemon(): this method checks if the working thread is a daemon thread or not if the working thread is a daemon thread return true otherwise return false.

Example of daemon thread

```

class M extends Thread
{
    public void run()

```

```

{
    try{
        for(int i=1; i<=5; i++)
        { System.out.println("I is "+i);
          Thread.sleep(1000);
        }
    }
    catch(Exception ex)
    { System.out.println("Error is "+ex);
    }
}
}

public class TestMainThreadApp
{ public static void main(String x[])
  {
    M m1 = new M();
    m1.setDaemon(true);
    m1.start();
  }
}

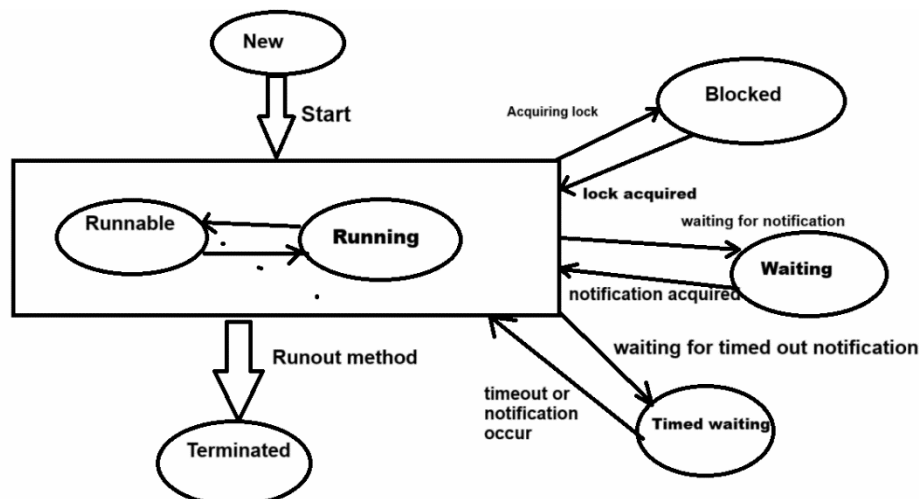
```

Thread LifeCycle

Thread life cycle is the process of tracking thread creation to thread termination as well as intermediate activity or work done by thread called thread life cycle.

If we want to work with thread life cycle we have some important phases of thread life cycle

1. New State
2. RunnableState
3. Blocked State
4. Waiting State
5. Timed waiting state
6. Terminate state



Above diagram shows the thread life cycle

New State: when we create an object of thread class but thread not yet started called as new state of thread.

```

class A extends Thread
{
    public void run()
    {
        //write here your logics
    }
}

```

A a1 = new A(); // new state

Runnable state: if thread is ready to run is moved to runnable state or may running thread status is also known as runnable state and thread can move for run any instant of time is responsibility of thread scheduler means thread scheduler is responsible for allocate cpu to thread for execution means in short we can say thread is in runnable state when we call a start() method or when run() is in execution

```

class A extends Thread
{
    public void run()
    {
        //write here your logics
    }
}

A a1 = new A(); // new state
a1.start(); //runnable state

```

blocked : thread will be block state when it is trying to acquired lock but the current lock is acquired by the other thread. So thread will be move from block to runnable state.

Waiting state: thread will be in waiting state when its call wait() method or join method it will move from runnable state when other thread will notify or thread will be terminated.

Timed waiting: thread is time waiting when we call thread with timeout parameter and when specified time out then thread automatically move from waiting to runnable state e.g sleep() method of Thread class

Terminated state: terminate means when we destroy thread or stop the execution of thread called as terminated state

So termination happen in three condition

1. User can terminate thread manually by calling stop method
2. When runtime exception occur in thread execution then thread may be terminate
3. When thread execute its all logic then thread scheduler can terminate thread properly

//Remaining point we want to cover in JDK 1.8

- a. Executor services
- b. Thread pooling
- c. Concurrency API
- d.

Assignment

Question 1: perform task using asynchronization technique

```
class Product
{
    private int id;
    private String name;

    //setter and getter
}

class MachineOne extends Thread
{
    public void setProduct(Product ...p)
    {
    }
    public void run()
    {
    }
}

class MachineTwo extends Thread
{
    public void setProduct(Product ...p)
    {
    }
    public void run()
    {
    }
}
```

Task1:
you have to provide two products count in MachineOne and MachineTwo
and both machine should develop the products simultaneously and every product
development has 5 second time and product name should form
MachineOneProd1
MachineTwoProd1
MachineOneProd2
MachineTwoProd2

Question2: Perform task using synchronization

```
class Product
{
    private int id;
    private String name;

    //setter and getter
}

class MachineOne extends Thread
{
    public void setProduct(Product ...p)
    {
    }
    public void run()
    {
    }
}

class MachineTwo extends Thread
{
    public void setProduct(Product ...p)
    {
    }
    public void run()
    {
    }
}
```

Task1:
you have to provide two products count in MachineOne and MachineTwo
and both machine should develop the products simultaneously and every product
development has 5 second time and product name should form
MachineOneProd1
MachineOneProd2
MachineOneProd3
.....
MachineOneProd10
MachineTwoProd1
..
..
MachineTwoProd10

Example3:

Scenario Overview: We have a bank account class where multiple threads are trying to access and modify the balance by performing deposit and withdrawal operations. We need to ensure thread safety when these operations are happening concurrently.

Program Steps:

1. **Create a BankAccount Class:** This class will have methods for deposit and withdrawal. We'll make these methods synchronized to avoid race conditions.
2. **Create a Task Class:** This class will represent the deposit/withdrawal operation. It will implement the `Runnable` interface.
3. **Run the Application:** We'll create multiple threads to simulate concurrent deposits and withdrawals on the same bank account.

Example4:

Scenario: Race Game Simulation

Scenario Overview: In this simulation, we have multiple players, each running in their own thread. The goal is for each player to race to the finish line. The position of each player will be updated in each thread, and we'll simulate the race by allowing each player to randomly move a small distance forward on each update.

Program Steps:

1. **Create a `Player` class:** Represents each player in the game.
2. **Implement the `Runnable` interface:** Each player's movement will be handled by a separate thread.
3. **Run the race:** We'll create a number of players, start their threads, and simulate the race.
4. **Race Outcome:** Once a player reaches the finish line, the race will end, and we will announce the winner.