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.

The differences between Multithreading and Multiprocessing are:

Multithreading	Multiprocessing
In multithreading, multiple work or task are done in one process by the use of threads.	In multiprocessing, multiple processes are done together by processing.
For eg: A video being played. The audio, visuals and volume uo and down and all the other tasks that are being done together to play the video, etc.	For eg: VLC is playing a video and the files are being downloaded, all the process is being done together, etc.

We can create n number of threads under one process.

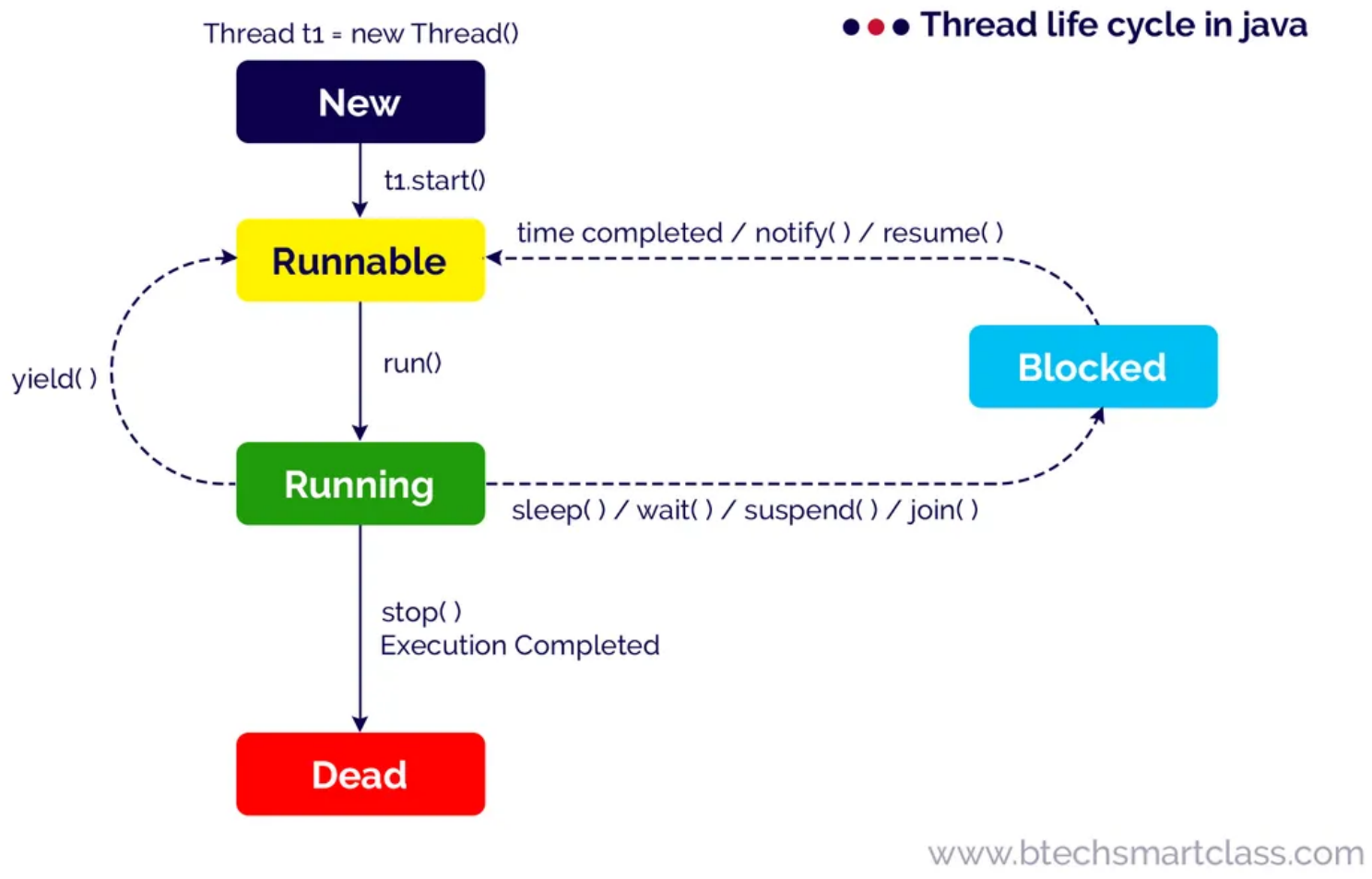
Creating Thread in Java:

- Using Runnable
- Using Thread

Using of Runnable interface is the best approach because the thread class might cause problems for multiple inheritance. When trying to extend another thread class and trying to perform multiple inheritance. Whereas Runnable does not cause any problems like that.

```
public class MainThread1 implements Runnable{
    /**
     * To create a thread in java, there are two ways:
     * -> Using Runnable Interface
     * -> Using Thread Class
     */
    // run() method is from the Runnable interface
    @Override
    public void run(){
        // Performing a task
        for(int i = 0; i < 10; i++){
            System.out.print(i + " ");

            // Using a sleep method to give a break for 1 second for every iteration
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }

    public static void main(String[] args) {

        MainThread1 mt1 = new MainThread1();

        Thread t1 = new Thread(mt1);

        // Object of another thread
        MainThread2 mt2 = new MainThread2();

        // When multiple threads are executing together. It is known as concurrent execution

        t1.start();
        mt2.start();

    }
}

public class MainThread2 extends Thread{

    @Override
    public void run(){
        //Performing a task for thread
        for(int i=10; i>1; i--){
            System.out.print(i+" ");
            try{
                Thread.sleep(1500);
```

```
            }
            catch(Exception e){
                throw new RuntimeException(e);
            }
        }
    }

}
```



Thread Operations

- Thread class provides methods to perform operations with threads.
- This thread class is present in java.lang package. So, we don/’t need to import this package.
- Some important thread methods:
    1. **public String getName()** - returns the name of the thread.
    2. **public void setName(String name)** - set the name of thread.
    3. **public void run()** - contain the task of thread.
    4. **public void start()** - start thread by allocating resources.
    5. **public long getId()** - returns the id of thread.
    6. **setPriority(p), getPriority()** - set and get the priority.
    7. **sleep(), join(), interrupt(), stop(),** etc.

```
public class ThreadOperations {

    public static void main(String[] args) throws InterruptedException {
        System.out.println("Program started...");
        int a = 12 + 34;
        System.out.println("The value of a: " + a);

        Thread t = Thread.currentThread();
        String threadName = t.getName();
        System.out.println("The thread running currently is: " + threadName);

        // Setting a new thread name
        t.setName("MyNewMain");
        System.out.println("The new thread name is " + t.getName());

        // sleep() method needs to be kept in try/catch block because it throws exception
        try {
            Thread.sleep(3500);
```

```
        }
        catch(Exception e){
            throw new RuntimeException(e);
        }

        // To get the ID of the main thread
        System.out.println(t.getId());

        // Starting the user defined thread.
        UserThread tu = new UserThread();
        tu.start();

        System.out.println("Program ended....");
    }

    public static class UserThread extends Thread{

        @Override
        public void run(){
            // task for thread
            System.out.println("This is user defined thread.");
        }
    }
}
```

Daemon Thread

- Daemon Thread in java is a service provider thread that provides services to the user thread.
- **setDaemon(Boolean)**
- **public Boolean isDaemon()**
- **Garbage Collector is best Example of Daemon Thread**

Producer Consumer Problem

Multi-Threading in Java: In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

- The producer’s job is to generate data, put it into the buffer, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

In this problem, we need two threads, Thread t1 (produces the data) and Thread t2 (consumes the data). However, both the threads shouldn’t run simultaneously.

```
public class Company {

    int n;
    boolean f = false;
    // f=false: chance: producer
    // f=true chance: consumer

    synchronized public void produceItem(int n) throws InterruptedException {
        // producer will wait for it's chance due to wait()
        if(f){
            wait();
        }
        this.n = n;
        System.out.println("Produced: " + this.n);

        // consumer's chance
        f = true;

        // consumer gets notified about it's chance
        // This is how inter thread communication is achieved.
        notify();
    }
    synchronized public int consumeItem() throws InterruptedException {

        // consumer will wait for it's chance with wait()
        if(!f){
            wait();
        }
```

```
        System.out.println("Consumed: " + this.n);

        // After consuming, the producer chance will come
        f = false;

        // the producer gets notified because of notify() about it's chance
        notify();
        return this.n;
    }

}

public class Producer extends Thread{

    Company comp;
    public Producer(Company comp){
        this.comp = comp;
    }

    @Override
    public void run(){
        int i = 1;
        while(true){
            try {
                this.comp.produceItem(i);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            try{
                Thread.sleep(1500);
            }
            catch(Exception e){
                throw new RuntimeException(e);
            }
            i++;
        }
    }

}

public class Consumer extends Thread{
    Company comp1;
    public Consumer(Company comp1){
        this.comp1 = comp1;
    }
    @Override
    public void run(){
        while(true){
            try {
                this.comp1.consumeItem();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            try{
                Thread.sleep(2000);
            }
            catch(Exception e){
                throw new RuntimeException(e);
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {

        Company company = new Company();
        Producer pro = new Producer(company);
        Consumer con = new Consumer(company);

        //Calling the Threads
        pro.start();
        con.start();

    }
}
```