**Multithreading**

Executing multiple threads (sub process, small task) at single time

Music playing.

Updating time.

Animation is running

It is implemented in games

Updating score

Directions

Element movement

**Difference between process and thread**

1. Process is a program which is in executing state where thread is a subpart of the process
2. Process is a heavy weight where thread is of light weight.

**2 ways to create a thread**

* **By extending Thread class**
* **Implementing Runnable interface**

**Note:** thread class present in java.lang package

Main thread is the default thread created by jvm

1. **By extending Thread class**

**package thread;**

**public class Test extends Thread{**

**@Override**

**public void run() {**

**}**

**public static void main(String[] args) {**

**Test t =new Test();**

**t.start();**

**}**

**}**

**Thread life cycle**

**1. new state:**

**we are creating the object of test class**

**2. runnable state:**

**new thread reaches runnable state as we call start method**

**3. Running state:**

**when we say t.start(); it will start executing the run method**

**the implementation of start method is done in such a way that it will**

**start executing the thread**

**after runnable stage Jvm will allocate the processor for thread and**

**thread scheduler will decide which thread to be executed based on**

**the algorithm**

**now thread is start executed**

**4. dead state:**

**thread is complete its task**

**2.5 Non runnable state**

**between runnable and running state thread can be in non-runnable state.**

**waiting or sleeping state.**

**after sometime it may come to runnable state and then it move forward to**

**running state.**

**package java.lang;**

**class Thread implementing Runnable{**

**// constructors**

**// methods**

**start()**

**run()**

**sleep()**

**}**

**public interface Runnable{**

**public void run();**

**}**

**We cant invoke same thread multiple times**

**package thread;**

**public class Test extends Thread{**

**@Override**

**public void run() {**

**System.*out*.println("Thread is running");**

**}**

**public static void main(String[] args) {**

**Test t =new Test();**

**t.start();**

**// t.start();// we cant invoke thread multiple times**

**// we are getting java.lang.IllegalThreadStateException**

**}**

**}**

**Implementing runnable interface**

**package thread;**

**public class Test implements Runnable{**

**@Override**

**public void run() {**

**System.*out*.println("Thread is running");**

**}**

**public static void main(String[] args) {**

**Test t =new Test();**

**// start method is present in thread class so we need to**

**// create an object of thread class and pass test class reference into**

**// the thread class object**

**Thread th = new Thread(t);**

**th.start();**

**}**

**}**

Which is the efficient way of creating the thread class

By implementing the runnable interface

**Reason**:

In programming we are using inheritance concept. If in case if a class is already inheriting from a class when it tried to inherit thread class, there we come across the concept of **multiple inheritance**.

We know that one class can inherit a class and implement n number of interfaces. So, implementing interface is the best way of creating the thread.

**Performing single task from single thread**

**package thread;**

**public class Test extends Thread{**

**@Override**

**public void run() {**

**System.out.println(“thread is running”);**

**}**

**public static void main(String[] args) {**

**Test t =new Test();**

**t.start();**

**}**

**}**

**Performing single task from Multiple thread**

**package thread;**

**public class Test extends Thread{**

**@Override**

**public void run() {**

**System.*out*.println("thread is running");**

**}**

**public static void main(String[] args) {**

**Test t1 =new Test();**

**t1.start();**

**Test t2 = new Test();**

**t2.start();**

**}**

**}**

**Performing Multiple task from single thread**

**This is not possible**

**Performing Multiple task from multiple thread**

**package thread;**

**public class Test extends Thread{**

**@Override**

**public void run() {**

**System.*out*.println("Test thread is running");**

**}**

**public static void main(String[] args) {**

**Test t1 =new Test();**

**t1.start();**

**thread1 th1=new thread1();**

**th1.start();**

**thread2 th2= new thread2();**

**th2.start();**

**}**

**}**

**class thread1 extends Thread{**

**@Override**

**public void run() {**

**System.*out*.println("thread1 is running");**

**}**

**}**

**class thread2 extends Thread{**

**@Override**

**public void run() {**

**System.*out*.println("thread2 is running");**

**}**

**}**

**// execution depends on algorithm and decided by jvm ie thread scheduler**

**//every thread executes simultaneously**

**Constructors and methods of thread**

1. **Thread() 🡪 default constructor**
2. **Thread(Runnable target) 🡪 thread th=new thread(t);**
3. **When we extending thread**

**Thread(String name) 🡪 when we create a thread by default provide name. if we want we can do it explicitly.**

1. **When we implementing runnable**

**Thread(Runnable target, String name);**

1. **Thread(Threadgroup tg , Runnable target)**
2. **Thread(Threadgroup tg , String name)**
3. **Thread(Threadgroup tg , Runnable target,String name)**
4. **Thread(Threadgroup tg , Runnable target,String name,long stacksize)**

**Methods**

1. **run()**
2. **start()**
3. **currentThread 🡪 static**
4. **isAlive() 🡪 thread is in execution state or not**
5. **getName(), setName(String name)**
6. **demon thread related**

**isDaemon() setDaemon(boolean b)**

**true – daemon thread created**

**false- not created**

1. **getPriority() setPriority(int priority)**
2. **preventing thread execution method**

**sleep() yield() join()**

**Thread interruption method**

**interrupt();**

**isInturrupted();**

**Inturrupted();**

**Inter thread communication**

**Wait() notify() notifyAll()**

**getName() and setName() method**

* **for main method**

**package thread;**

**public class Test{**

**public static void main(String[] args) {**

**System.*out*.println(Thread.*currentThread*().getName());**

**Thread.*currentThread*().setName("tom");**

**System.*out*.println(Thread.*currentThread*().getName());**

**// System.out.println(1/0);**

**//Exception in thread "tom"**

**}**

**}**

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**// how to change threadname here**

**Thread.*currentThread*().setName("hello");**

**System.*out*.println("Thread Name : "+Thread.*currentThread*().getName());**

**//Thread-0 default name assigned by jvm**

**System.*out*.println("running");**

**}**

**public static void main(String[] args) {**

**System.*out*.println("Thread Name : "+Thread.*currentThread*().getName());**

**Test t= new Test();**

**t.start();**

**Test t2=new Test();**

**t2.start();**

**Test t3=new Test();**

**t3.setName("tom");**

**t3.start();**

**}**

**}**

**isAlive()package thread;**

**public class Test extends Thread{**

**public void run() {**

**System.*out*.println(Thread.*currentThread*().isAlive());**

**}**

**/\***

**true**

**true**

**true**

**\*/**

**public static void main(String[] args) {**

**Test t= new Test();**

**t.start();**

**System.*out*.println(Thread.*currentThread*().isAlive());**

**System.*out*.println(t.isAlive());**

**}**

**}**

**Daemon thread**

* it runs in the background of another thread
* it provide service to the thread

eg : garbage collector

it works in the background of the main thread

it resolve problems of main method like memory full it delete waste objects

ex: spelling checker in ms word

**Methods on demon thread**

**Public final void setDaemon(Boolean b){}**

**Public final Boolean isDaemon(){}**

**Note1: before starting the thread we need to create the daemon thread. Else exception illegal-thread-state**

**Note: we cant create main method as daemon thread because it is already started by JVM.**

**Daemon thread inherit properties from its parent**

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**if(Thread.*currentThread*().isDaemon())**

**System.*out*.println("daemon thread");**

**else**

**System.*out*.println("childthread");**

**}**

**public static void main(String[] args) {**

**System.*out*.println("main thread");**

**Test t= new Test();**

**t.setDaemon(true);**

**t.start();**

**}**

**}**

**When main method is not doing any task it wont execute daemon thread. Because its purpose only to provide service to main thread**

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**if(Thread.currentThread().isDaemon())**

**System.out.println("daemon thread");**

**else**

**System.out.println("childthread");**

**}**

**public static void main(String[] args) {**

**// System.out.println("main thread");**

**Test t= new Test();**

**t.setDaemon(true);**

**t.start();**

**// System.out.println();**

**}**

**}**

**We cant set main thread as daemon thread and we cant set thread as a daemon thread after its running**

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**if(Thread.*currentThread*().isDaemon())**

**System.*out*.println("daemon thread");**

**else**

**System.*out*.println("childthread");**

**}**

**public static void main(String[] args) {**

**// Thread.currentThread().setDaemon(true);**

**Test t= new Test();**

**t.setDaemon(true);**

**t.start();**

**t.setDaemon(true);**

**}**

**}**

**Thread priorities**

* JVM provides priority to each thread
* For these priorities Jvm allocates the processor
* Priorities are represented in the form of integer

And it ranges from 1 – 10

* **1🡪MIN\_PRIORITY**
* **5🡪NORM\_PRIORITY**
* **10🡪MAX\_PRIORITY**
* **We cant use 0 or >10 or negative values as priority**

**Methods used in priority**

**Public final void setPriority(){}**

**Public final int getPriority(){**

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**System.*out*.println(Thread.*currentThread*().getPriority());**

**}**

**public static void main(String[] args) {**

**System.*out*.println(Thread.*currentThread*().getPriority());**

**// by defaultly priority of main thread is 5**

**// and it is inherited by the sub thread also so we get the output as 5 5**

**Test t= new Test();**

**t.start();**

**// now setting the priority**

**t.setPriority(10);**

**// t.setPriority(11);// IllegalArgumentException**

**thread1 t1=new thread1();**

**t1.setPriority(1);**

**t1.start();**

**}**

**}**

**// priority depends on platform windows does not support priorities.**

**class thread1 extends Thread{**

**@Override**

**public void run() {**

**System.*out*.println(Thread.*currentThread*().getPriority());**

**// System.out.println("thread1 is running");**

**}**

**}**

**Sleep() method**

**Public static native void sleep(long milli) throws interrupted exception**

**{**

**}**

**package thread;**

**public class Test extends Thread{**

**public void run(){**

**for(int i=0;i<5;i++) {**

**System.*out*.println(i);**

**try {**

**Thread.*sleep*(500);**

**} catch (InterruptedException e) {**

**}**

**}**

**}**

**public static void main(String[] args) {**

**for(int i=6;i<10;i++) {**

**System.*out*.println(i);**

**try {**

**Thread.*sleep*(100);**

**} catch (InterruptedException e) {**

**}**

**}**

**Test t= new Test();**

**t.start();**

**}**

**}**

**Yield ()**

* Internally it uses sleep method
* It stops the current executing thread and give a chance to other threads for execution
* In this method thread provides the hint to the thread scheduler. Then it depends on thread scheduler to accept or ignore the hint.
* Public static native void yield(){}

**package thread;**

**public class Test extends Thread{**

**public void run(){**

**Thread.*yield*(); // stop thread execution and let main thread to execute**

**for(int i=0;i<5;i++) {**

**System.*out*.println(i+" "+Thread.*currentThread*().getName());**

**}**

**}**

**public static void main(String[] args) {**

**// output depend upon the thread scheduler**

**// it may accept or reject the request**

**// Thread.yield(); // stop main thread execution and let thread to execute**

**Test t= new Test();**

**t.start();**

**for(int i=10;i<15;i++) {**

**System.*out*.println(i+" "+Thread.*currentThread*().getName());**

**}**

**}**

**}**

**Join() method**

if a thread wants to wait for another thread to completes its task then we should use join() method

**physical exam 🡪 written 🡪 medical exam**

these 3 cant done simultaneously written exam to be wait until physical exam is over and mE to wait until written exam

**Methods**

**Public final void join() throws Interrupted Exception**

**Public final synchronized void join(long ms){}**

**Public final synchronized void join(long ms,int ns){}**

**physical exam 🡪 written 🡪 medical exam**

**package thread;**

**public class Test{**

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

**Physical ph=new Physical();**

**ph.start();**

**ph.join();**

**Written w=new Written();**

**w.start();**

**w.join();**

**Medical m=new Medical();**

**m.start();**

**m.join();**

**}**

**}**

**class Physical extends Thread{**

**public void run() {**

**System.*out*.println("physical starts");**

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**}**

**System.*out*.println("physical ends");**

**}**

**}**

**class Written extends Thread{**

**public void run() {**

**System.*out*.println("written starts");**

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**}**

**System.*out*.println("written ends");**

**}**

**}**

**class Medical extends Thread{**

**public void run() {**

**System.*out*.println("medical starts");**

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**}**

**System.*out*.println("medical ends");**

**}**

**}**

**Interrupt methods**

* **It is used to interrupt an executing thred**
* **This method will only work when the thread is in sleeping state or waiting state**
* **If a thread is not in sleeping state or waiting state then calling an interrupt method will perform normal behavior // unused command**
* **It throw interrupted exception**

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**try {**

**for(int i=0;i<5;i++) {**

**System.*out*.println(i);**

**//Thread.*sleep*(1000);**

**}**

**} catch (InterruptedException e) {**

**System.*out*.println("thread interrupted"+e );**

**}**

**}**

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

**Test t= new Test();**

**t.start();**

**t.interrupt();**

**}**

**}// try without sleep method**

**// interrupt() is unused command if we are not in sleeping state**

**Interrupted () and isInterrupted ()**

* These two methods used to check whether a thread is interrupted or not
* Interrupted method clears the interrupted status from true to false if thread is interrupted
* isInterrupted is only to check interrupted status.
* If we call interrupted method more than once it will change the result but isinterrupted wont.

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**System.*out*.println(Thread.*interrupted*());**

**// it change the state of interruption so program executes**

**try {**

**for(int i=0;i<5;i++) {**

**System.*out*.println(i);**

**Thread.*sleep*(1000);**

**}**

**} catch (Exception e) {**

**System.*out*.println("thread inturupted"+e);**

**}**

**}**

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

**Test t= new Test();**

**t.start();**

**t.interrupt();**

**}**

**}**

**isInterrupted ()**

**package thread;**

**public class Test extends Thread{**

**public void run() {**

**System.*out*.println(Thread.*currentThread*().isInterrupted());**

**// it will not change the state of interruption so program terminates**

**// after one iteration**

**try {**

**for(int i=0;i<5;i++) {**

**System.*out*.println(i);**

**Thread.*sleep*(1000);**

**}**

**} catch (Exception e) {**

**System.*out*.println("thread inturupted"+e);**

**}**

**}**

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

**Test t= new Test();**

**t.start();**

**t.interrupt();**

**}**

**}**

**Synchronization**

**package thread;**

**class App extends Thread{**

**static BookSeats *b*;**

**int seats;**

**public void run() {**

***b*.bookSeats(seats);**

**}**

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

***b*=new BookSeats();**

**// person 1 booking 37 seats**

**App jerry= new App();**

**jerry.seats=37;**

**jerry.start();**

**// person 2 booking 33 seats**

**App tom=new App();**

**tom.seats=33;**

**tom.start();**

**}**

**}**

**class BookSeats {**

**int totalSeats=50;**

**void bookSeats(int seats) {**

**if(seats<=totalSeats) {**

**System.*out*.println(seats+" seats booked successfully");**

**totalSeats-=seats;**

**System.*out*.println("seats left : "+totalSeats);**

**}**

**else {**

**System.*out*.println("booking closed");**

**}**

**}**

**}**

**What is Synchronization**

**It is the process by which we control the accessibility of multiple threads to a particular shared resource**

**Problems which can occur without Synchronization**

* Data inconsistency
* Thread interference

**Advantages of Synchronization**

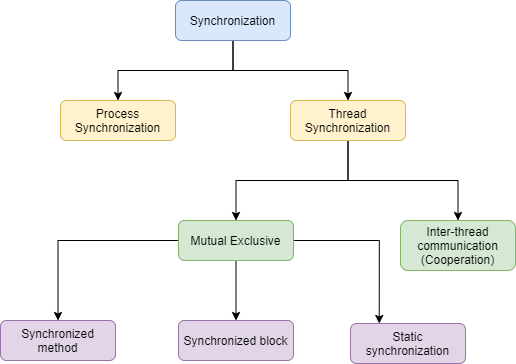
* No data inconsistency
* No thread interference

**Disadvantages**

1. Increase the waiting period of threads
2. Create performance problem

**To overcome the disadvantages of synchronization**

**Java provides java.util.concurrent**



**Wait() notify() notifyAll()**

**Synchronized Keyword**

**package thread;**

**class App extends Thread{**

**static BookSeats *b*;**

**int seats;**

**public void run() {**

***b*.bookSeats(seats);**

**}**

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

***b*=new BookSeats();**

**// person 1 booking 37 seats**

**App jerry= new App();**

**jerry.seats=37;**

**jerry.start();**

**// person 2 booking 33 seats**

**App tom=new App();**

**tom.seats=33;**

**tom.start();**

**}**

**}**

**class BookSeats {**

**int totalSeats=50;**

**synchronized void bookSeats(int seats) {**

**if(seats<=totalSeats) {**

**System.*out*.println(seats+" seats booked successfully");**

**totalSeats-=seats;**

**System.*out*.println("seats left : "+totalSeats);**

**}**

**else {**

**System.*out*.println("only "+totalSeats+" left .enter in this range");**

**}**

**}**

**}**

**package thread;**

**class App extends Thread{**

**static BookSeats *b*;**

**int seats;**

**public void run() {**

***b*.bookSeats(seats);**

**}**

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

***b*=new BookSeats();**

**App jerry= new App();**

**jerry.seats=37;**

**jerry.start();**

**App tom=new App();**

**tom.seats=14;**

**tom.start();**

**}**

**}**

**class BookSeats {**

**int totalSeats=50;**

**void bookSeats(int seats) {**

**/\***

**here thread is waiting until other thread executes**

**it consumes more time so we need to use synchronized**

**block**

**System.out.println(Thread.currentThread().getName());**

**System.out.println(Thread.currentThread().getName());**

**\*/**

**// two threads are executed in parallel except code**

**// inside synchronized block**

**System.*out*.println(Thread.*currentThread*().getName());**

**System.*out*.println(Thread.*currentThread*().getName());**

**synchronized(this) {**

**if(seats<=totalSeats) {**

**System.*out*.println(seats+" seats booked successfully");**

**totalSeats-=seats;**

**System.*out*.println("seats left : "+totalSeats);**

**}**

**else {**

**System.*out*.println("only "+totalSeats+" left .enter in this range");**

**}**

**}**

**// two threads executed in parallel**

**System.*out*.println(Thread.*currentThread*().getName());**

**System.*out*.println(Thread.*currentThread*().getName());**

**}**

**}**

**Static Synchronization**

**package thread;**

**class App extends Thread{**

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

**BookSeats b1=new BookSeats();**

**Mythread1 t1=new Mythread1(b1,7);**

**t1.start();**

**Mythread2 t2= new Mythread2(b1, 6);**

**t2.start();**

**BookSeats b2=new BookSeats();**

**Mythread1 t3=new Mythread1(b2,7);**

**t3.start();**

**Mythread2 t4= new Mythread2(b2, 6);**

**t4.start();**

**}**

**}**

**class BookSeats {**

**int totalSeats=20;**

**synchronized void bookSeats(int seats) {**

**if(seats<=totalSeats) {**

**System.*out*.println(seats+" seats booked successfully");**

**totalSeats-=seats;**

**System.*out*.println("seats left : "+totalSeats);**

**}**

**else {**

**System.*out*.println("only "+totalSeats+" left .enter in this range");**

**}**

**}**

**}**

**class Mythread1 extends Thread{**

**BookSeats b;**

**int seats;**

**Mythread1(BookSeats b,int seats){**

**this.b=b;**

**this.seats=seats;**

**}**

**public void run() {**

**b.bookSeats(seats);**

**}**

**}**

**class Mythread2 extends Thread{**

**BookSeats b;**

**int seats;**

**Mythread2(BookSeats b,int seats){**

**this.b=b;**

**this.seats=seats;**

**}**

**public void run() {**

**b.bookSeats(seats);**

**}**

**}**

**Static Synchronization**

**package thread;**

**class App extends Thread{**

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

**BookSeats b1=new BookSeats();**

**Mythread1 t1=new Mythread1(b1,7);**

**t1.start();**

**Mythread2 t2= new Mythread2(b1, 6);**

**t2.start();**

**BookSeats b2=new BookSeats();**

**Mythread1 t3=new Mythread1(b2,7);**

**t3.start();**

**Mythread2 t4= new Mythread2(b2, 6);**

**t4.start();**

**}**

**}**

**class BookSeats {**

**int totalSeats=20;**

**synchronized void bookSeats(int seats) {**

**if(seats<=totalSeats) {**

**System.*out*.println(seats+" seats booked successfully");**

**totalSeats-=seats;**

**System.*out*.println("seats left : "+totalSeats);**

**}**

**else {**

**System.*out*.println("only "+totalSeats+" left .enter in this range");**

**}**

**}**

**}**

**class Mythread1 extends Thread{**

**BookSeats b;**

**int seats;**

**Mythread1(BookSeats b,int seats){**

**this.b=b;**

**this.seats=seats;**

**}**

**public void run() {**

**b.bookSeats(seats);**

**}**

**}**

**class Mythread2 extends Thread{**

**BookSeats b;**

**int seats;**

**Mythread2(BookSeats b,int seats){**

**this.b=b;**

**this.seats=seats;**

**}**

**public void run() {**

**b.bookSeats(seats);**

**}**

**}**

**How to overcome this**

Using static synchronized method.

**package thread;**

**class App extends Thread{**

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

**BookSeats b1=new BookSeats();**

**Mythread1 t1=new Mythread1(b1,7);**

**t1.start();**

**Mythread2 t2= new Mythread2(b1, 6);**

**t2.start();**

**BookSeats b2=new BookSeats();**

**Mythread1 t3=new Mythread1(b2,8);**

**t3.start();**

**Mythread2 t4= new Mythread2(b2, 6);**

**t4.start();**

**}**

**}**

**class BookSeats {**

**static int *totalSeats*=20;**

**static synchronized void bookSeats(int seats) {**

**if(seats<=*totalSeats*) {**

**System.*out*.println(seats+" seats booked successfully");**

***totalSeats*-=seats;**

**System.*out*.println("seats left : "+*totalSeats*);**

**}**

**else {**

**System.*out*.println("only "+*totalSeats*+" left .enter in this range");**

**}**

**}**

**}**

**class Mythread1 extends Thread{**

**BookSeats b;**

**int seats;**

**Mythread1(BookSeats b,int seats){**

**this.b=b;**

**this.seats=seats;**

**}**

**public void run() {**

**b.*bookSeats*(seats);**

**}**

**}**

**class Mythread2 extends Thread{**

**BookSeats b;**

**int seats;**

**Mythread2(BookSeats b,int seats){**

**this.b=b;**

**this.seats=seats;**

**}**

**public void run() {**

**b.*bookSeats*(seats);**

**}**

**}**