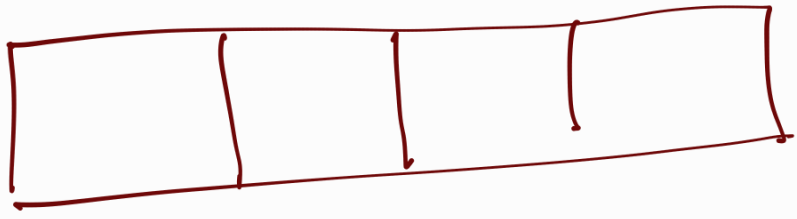
Thread → OS

Process



Fork

Pen

i/p-int → P1 → T1

2 → T2

E-commerce → Choose Item

→ Choose Recipe

Read card

5 Item

Bill order

$ Receipt Bill

Java Thread

Thread → a process in execution.

↓

Allows program to operate more effectively by performing multiple things at a time.

↳ Complicated tasks can be performed in background without interrupting the main program.

↳ Each process can be scheduled and prioritized. – Job scheduling

Ex: Task 1 | P1 | P2 | P3 | P4 |

Scheduling w.r.t time.

P1 – schedule (100s)
P2 – Start ( )
P3 – Schedule (200s)
P4 – Schedule (300s).

Task 2 | P1 | P2 | P3 | P4 | →

"more on these in OS"

P4 P1 P3 P2

prioritizing.

Thread Life Cycle

Class : Thread

Thread t = new Thread();

when a thread is created

NEW

t.start()

Ready to run.

Runnable

t.run()

t.wait()
t.sleep()

Running

in exen | process executed

waiting/ Blocked

after some interval.

t.stop()

terminate

Stop

* → run() is overridden in each program.

Creating Thread

Inheritance

extends Thread class

Interface.

implements Runnable interface

Extends Thread

```
public class ST extends Thread {
    public void run() {
        S.o.pln("This is a simple Thread");
    }
    p.s.v. main(String[] args) {
        ST myThread = new ST();
        myThread.start();
        S.o.pln("Thread executed");
    }
}
```

Implementing Runnable interface.

```
public class ST implements Runnable {
    public void run() {
        S.o.pln("runnable thread");
    }
    p.s.v. main(String args[]) {
```

```
ST obj = new ST();
Thread myThread = new Thread(obj);
myThread.start();
S.o.println("Thread executed");
    }
}
```

Note:

(a) If a class extends Thread class, the thread can be run by creating an instance of the class and call its run() method.

(b) If a class implements the Runnable interface, the thread can be run by passing an instance of the class to a Thread object constructor and then calling the threads start() method.

Methods of Thread class

→ Start()

→ run() ⇒ implements by calling Start().

→ Sleep (miliseconds) ⇒ Suspending thread

↓ Static method

throws an exception, so should be used inside "try-catch" block

→ join() ⇒ waits for thread to comple -te its procesp.

Static method

multithreading + try-catch ⇒ is used for first thread, then the second thread has to wait, till first is executed.

Note: Sleep() & join() are static methods. — No object is need to call. Instead

inside try-catch

```
Thread.Sleep (1000);
Thread.join ( ) ;
```

→ getID() ⇒ gives the ID of a thread

→ getName() ⇒ name of thread
Start from 0.
Ex: Thread=0
Thread=1
Thread=2

→ setName(String) ⇒ default thread Name will be replace with the String passed.

→ getPriority() ⇒ Priority values ranges from 1 to 10
MIN_PRIORITY - 1
default ← NORM_PRIORITY - 5
MAX_PRIORITY - 10

→ setPriority(int) - use to change the priority b/w 1-10
Ex: setPriority(10) → highest priority.

Drawback as we cannot know which thread will be executed first. Due the slm constraint.

→ isAlive( ) → to know the status of thread.

- True ⇓ Running
- False. ⇓ completed execution.

} Boolean.

```
class ThreadClass extends Thread
{
    public void run()
    {
        for (int i = 0; i<=5; i++)
        {
            try{ Thread.sleep(500);
                S.O.println(i); }
            catch(Exception e)
            { }
        }
    }

class tmethods{

    P.S.V.M(S[]a)
    {
        ThreadClass t1 = new
                ThreadClass();

        S.O.println("ID="+t1.getID());
```

```
s.o.pln("Name=" + t1.getName());

t1.setName("NewName");

S.o.pln("new name=" + t1.getName());

s.o.pln("Priority=" + t1.getPriority());

t1.setPriority(1);

s.o.pln("Priority=" + t1.getPriority
        after);

// we can't predict the thread
   PC executes 1000 of thread.

t1.start();

}

}

ThreadDemo t2 = new ThreadDemo();

t2.start

t1.start()

try {

t1.join();

} catch(Exception e)
```

Before join 2 threads executed simultaneously

t2.start():

after joint. t2 wait for t1 to execute.

---

at class

(RH)

① sleep() → MultiThreadDemo3.java
after every child thread is executed main thread start.

(RH)

② isAlive() → MultiThreadDemo4.java

(VR) JoinAliveThreads.java.

(RH) MultiThread_Demo5.java.

③ setPriority()
setPriority()
getName()
setName()
} MultiThread_Demo6.java

join() → Synch.java

④ Thread_OddEven.java.

Thread Synchronization

```
t1.start()
t2.start()
t3.start()
```

} order of execution is not predictable

```
t1      t2      t3
t3      t1      t2
t2      t3      t1 ....
```

When Synchronization is applied.

Synchro

t1 locked → after completion → unlock.

waiting: t2, t3

t2 → locked

t3

All achieved using join()

By default 100 - 1000 lines to be executed in Synch then 100-1000 joins need to be used.

to avoid this we used synchronization.

Synchronization achieved in 3 ways.

(1) Synchronized Keyword.
↓
prefixed to method (on which Statement written that has to be executed as thread)

(2) using Synchronized block.

Synchronized (this) Applied on which the method
{
≡ Add lines to be Synchronized
}

(3) Synchronized Static block.

→ Applied for method

Synchronized Static returntype method name ()
{
}

Note: we are not applying Synchronized thread in run() - we are creating different method and call of inside run().

Pgm - VR

Synch.java => each thread object has a String.
join()
They will be executed one after the other.
Not like before where all the thread exec simultaneously.

Synch1.java => replace join with Synchronize method