DEMONSTRATION - 10
#LECTURE - 26

e. getmessage()
e. toString()

→ Compile time error.
→ Run-time error.
→ Simple try catch block
→ Try with multiple catch.
→ Multiple errors with simple catch
→ Finally in try catch block
→ Exception handling using throw Statment.
→ Nested try catch block.

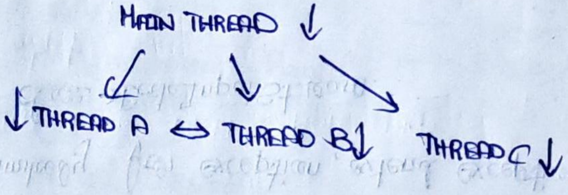
MULTITHREADED PROGRAMMING IN JAVA
#LECTURE - 27

**A SINGLE THREADED PROGRAM**

```
begin              class ABC {
 ↓ body                 p.s.v.m() {
end                             }
                   }
```

**A MULTITHREADED PROGRAM**

Threads may switch or exchange data/result among them.



```
Public class x{
    main(){
        read(){}
        sort(){}
        calc(){}
        Search(){}
        Print(){}
    }
}
```

## Single threaded execution

$$\begin{tikzcd}
\text{read()} \arrow[d] \\
\text{sort()} \arrow[d] \\
\text{calc()} \arrow[d] \\
\text{search()} \arrow[d] \\
\text{Print()}
\end{tikzcd}$$

## Multithreaded execution

$$\begin{tikzcd}
 & \text{read()} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Sort} \arrow[dr] & \text{calc()} \arrow[d] & \text{search()} \arrow[dl] \\
 & \text{Print()} &
\end{tikzcd}$$

## MULTITASKING

Multi-tasking ≡ time-sharing.

↳ CPU is kept active
CPU is fully utilized when CPU is idle.

```
P1:    Busy        Busy        Busy      Busy
P2:          Busy        Busy       Busy       Busy.
```

## MULTIPROCESSING

Computer works several tasks in ||el.
multi processing units.

## PROCESS

DEF: Executable program loaded into mem.
Has it's own address space: Variables & OS (in mem).
Each process may execute diff program.
Communicate via OS, files, networks.
May contain multiple threads.

## THREAD

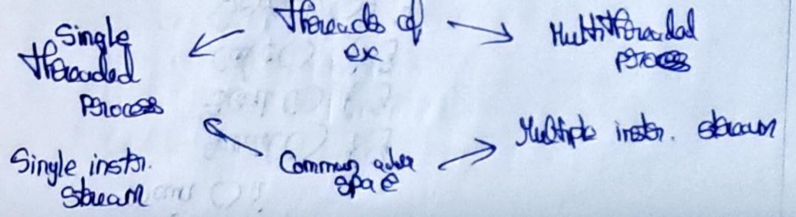
DEF: Sequential executed stream of instruction.
Also known as lightweight process
Has own execution Content: PC, call stack.
Communicate Via shared acess data.
Multiple thread in process execute same program.

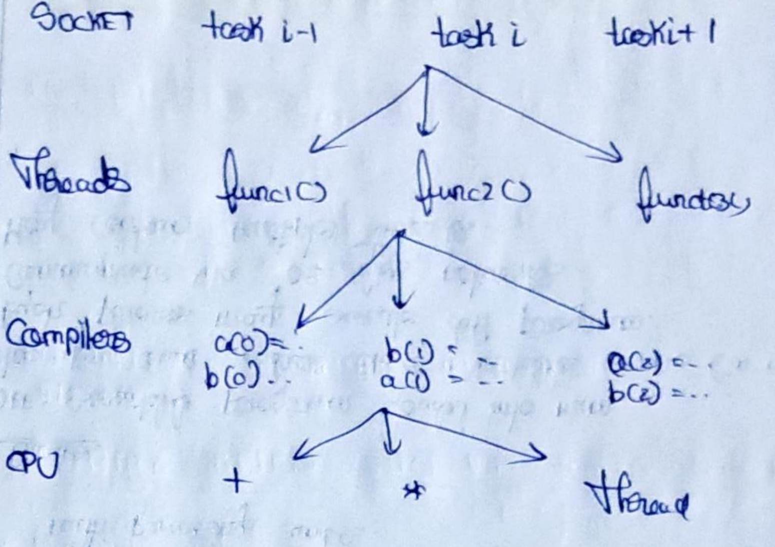
Eg: Multithreaded server.
Multithreaded/Paralel file copy.

## How MULTITHREADING ?

Single + multi-threaded process

* Threads are light weight process within a process.





## CODE GRANULARITY

Code item
Large grain (task level)
Program.

Medium grain (control level)
Function (thread)

Fine grain (data level)
Loop (compiler).

Very fine grain (multiple issue) with hardware.

## A THREAD IS....

* It is a piece of code that runs concurrently with other threads

* Each thread is a statically ordered seq. of instr.

* Thread are being extensibly used to express "concurrency" on both single & multiprocessor machines

* Programming a task having multiple thread of control.
  ↳ multithreaded / multithread programming.

## JAVA THREADS

→ synchronization
→ thread scheduling
→ inter-thread comm.

Java garbage collector is a low prio. thread.

Methods ⇒ current Thread
yield
sleep
resume
start
run
stop
setPriority
getPriority
suspend.

(✻) Everything about thread is in $\underset{\text{PACKAGE}}{\underline{\text{java. lang}}}$. $\underset{\text{CLASS.}}{\underline{\text{thread}}}$ + interface runnable

$$\begin{tikzcd}
\text{Inactive} \arrow[r, "\text{start()}"'] & \text{Alive} \arrow[r, "\text{run()}"'] & \text{Dead.}
\end{tikzcd}$$

Runnable is interface.
So it can be multiply inherited.

$$\begin{tikzcd}
& \text{Thread} \arrow[dl, bend right, "\text{extends}"'] \arrow[dr, "\text{Implements}"] & \\
\text{Thread (class)} \arrow[dr] & & \text{Runnable (Interface)} \arrow[dl] \\
& \text{run() method (override)} &
\end{tikzcd}$$

## CREATING THREADS

* Thread class

    Public class Thread extends obj {}

* Runnable interface

    Public interface runnable {
        Public Void run();
    }

## MORE THREAD CLASS METHODS

| | |
|---|---|
| Thread | currentThread() |
| String | getName() |
| Void | interrupt() |
| boolean | isAlive() |
| Void | join() |
| Void | SetDaemon() |
| Void | SetName() |
| Void | SetPriority() |
| Static Void | sleep() |
| Static Void | yield() |

## CREATING THREAD USING Thread class

```
class our_thread extends Thread {
    run(){
    }
}
under main. [ our_thread a = new our_thread();
             a. start()
```

## MULTITHREADED PROGRAMMING IN JAVA - II

#LECTURE ~~28~~

## CREATING THREAD OBJECT VIA THREAD CONSTRUCTOR.

* Runnable interface

    → Create obj implementing runnable interface
    → Pass it to thread obj via constructor.

```
[ Thread T = new Thread(new MYT);
     ↑ { Thread x   x = new Thread x().
       { Thread t1 = new Thread(x);
          t1.start();
using runnable.
```

**STATES OF A THREAD:**

Java thread can be in anyone of the state,

- NEW - allocated & waiting for start()
- RUNNABLE - begin exec.
- RUNNING - currently exec.
- BLOCKED - waiting for I/O.
- DEAD - finished.

transitions ⇒

Invoking methods in class thread,
new(), start(), yield(), sleep(),
wait(), notify().

other,
sheduler
I/O
returning from run.

$$\begin{tikzcd}
{} \arrow[r, "\text{new}"'] & \text{NEW} \arrow[r, "\text{start}"'] & \text{runnable} \arrow[dd, shift right, "\text{shedule}"'] & \\
& & & \text{blocked} \arrow[ul, "\text{notify, notifyAll, IO complete, sleep expired, join complete execute}"'] \\
& & \text{running} \arrow[uu, shift right] \arrow[ur, "\text{IO, sleep, wait, join, suspend}"'] \arrow[d, "\text{terminate}"'] & \\
& & \text{dead} &
\end{tikzcd}$$

start()
suspend() ≠ stop(), resume()
sleep(int n)
yield()

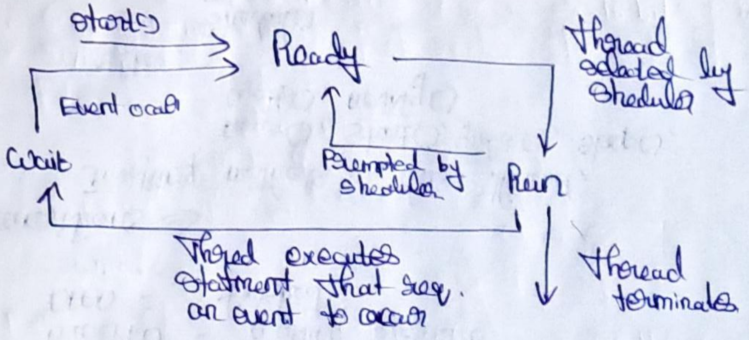
**JAVA THREAD TYPES**

→ user
→ Daemon (provides general service, never terminate, calles -setDaemon() before start())

Program termination...
* all user thread finished
* Daemon thread finished by JVM.
* Main program finishes

SHEDULING POLICY

→ non- preemptive
→ preemptive



start()
Ready
Thread selected by Sheduler
Event occur
Wait
Preempted by sheduler
Run
Thread executes statment that req. an event to occur
thread terminates

THREAD SHEDULING OBS

* order in which threads are selected for execution is indeterminate
  ↳ Depends on Sheduler

* Threads can ~~[illegible]~~ block indef. (STARVATION).
  ↳ if other thread always exe. first.

* thread sheduling can cause data races
  ↳ modify Same data from multiple threads
  ↳ result depends on thread exec. order.

* Synchronization
  ↳ Control thread exec order
  ↳ eliminates data race

PRIORITY OF THREADS

Each thread has priority which affects order in which it shedules for running

thrnameSet Priority (int Number)

MIN_PRIORITY=1
NORM_PRIORITY=5
MAX_PRIORITY=10.

## THREAD SYNCHRONISATION

when 2/more process attempts to acess a shared resource, it should be sync. to avoid conflict.

Synchronized (obj) { block of statments(s) }

↑

This prevents from sim acess.

STACK ⇒ one is pushing, other popping.

## DEMONSTRATION - XI
#LECTURE - 29

getID() helps to get ID of the thread.

suspend() + resume()

↓

wait() + notify()

## IO - STREAM - I
#LECTURE - 30

### IO STREAM

* Java treats flow of data as stream.
* They are classified as → I/O

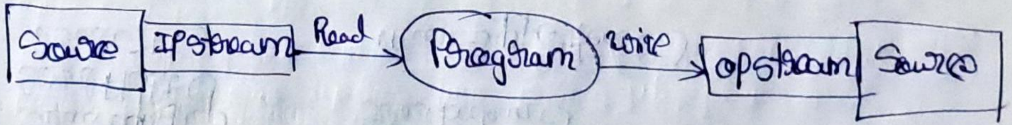
$$\begin{tikzcd} & \text{I/O} \arrow[dl] \arrow[dr] & \\ \text{Input stream} & & \text{output stream} \end{tikzcd}$$

PACKAGE: java.io

→ Input stream: Keyboard, mouse, mem, disk, network.
→ op stream: Screen, printer, mem, disk, network.

Reading data into a prog

write data to dest. from PROGRAM..

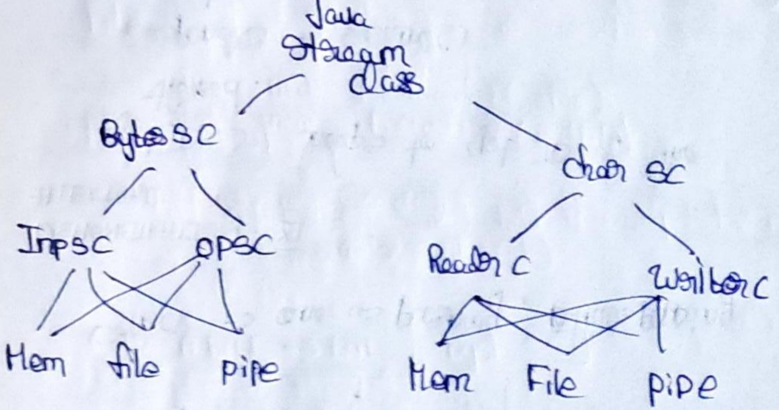


## JAVA CLASSES FOR I/O

java.io ⇒ Packages

Byte stream classes ⇒ 8bits (1byte)
Char stream class (sc) ⇒ 16 bits (2byte)

IPSC → I/P stream classes
OPSC → o/p stream clas



BYTE SC

JAVA IP SC

* used to read 8-bit bytes & support a no. of input related methods.

Inputstreamclass

read() read (byte b[])
read (byte B[], n, m)

available()
skip (n).

reset()
close().

DataInputStream

readShort()
read Int()
readLong()
readFloat()
read(UTF()

readLine()
readchar()
readBoolean().

mark supported ()

## JAVA OP STREAM CLASS

used to write 8-bit bytes & suppor no of methods

outputstream

write()
write (byte b[])
write(byte b[], n,m)
close()
flush()

DataoutputStream

writeshort()
Int()
Long ()
Float()
UTF()
Double()
Line()
char()
Boolean()