Module III

→ Exception Handling =

handling errors:

Exception types:

① checked (E):
compile time → prgm errors.
(convert source code to executable code)

② unchecked (E):
runtime → prgm errors.
(user's errors)
↓
* Arithmetic errors
↳ 10/0;
* numbers format
↳ String str="hello";
int num = —
* nullpointer (E).
↳ String str=null;
S.o.p(str.length());

Among these errors various handling operations we use → TRY, catch, throw, Throws, finally.

* Exception handling is a mechanism that allows java prgms to handle various exceptional condtn.
* An Exceptional condtn is a prbm (Ex) → Exceptional like symantic violation of the lang & prgm defined errors that prevents the continuation of the execution of a method.
* whn an (Ex) condtn occurs, an (Ex) is thrown.
* If the JVM / runtime envnmnt detects the symantic violation, the virtual machine implicitly throws an (Ex).
* 5 keywords are :-
(1) TRY (2) CATCH (3) THROW (4) THROWS (5) FINALLY.
* 2 types of (Ex) —

(1) Checked (Ex):	(2) Unchecked (Ex):
• The clses that are directly inherit the throwable cls except runtime (Ex) & error → C.(Ex).	The clses that inherit the run time (Ex) → U.(Ex) eg → Arithmetic (Ex) no-format (Ex) Nullpointer (Ex)

I TRY & CATCH :

eg. ①

```
public class javaExeg {
    public static void main (Str [] args){
        try {
            int data = 100/0;
        }
        catch (Arithmetic exception e)
        {
            S.o.println (e);
        }
        S.o.println ("rest of code")
    }
}
```

possible Errors → try -ൽ എഴുതുന്നു
A particular error വരാവുന്ന സാധ്യത → catch
(ആ error ഉണ്ടാകുന്നതിനു മുമ്പിലായി അതിനു
ഉണ്ടാകുന്നതിനു execute ചെയ്യും)

o/p

Exception in main java lang
Arithmetic exception / by zero.
rest of code.

eg — ②

```
class test {
    p.v.s.m (String args[]) {
        int x, y
        int a=10, b=5, c=5
        x = a/(b-a)
        S.o.println (x);
        y = a/(b+c)
        S.o.println (y);
    }
}
```

prgm without trycatch.

with

```
class test {
    p.s.v.m (string args[]) {
        int x,y;
        int a=10, b=5, c=5;
        try {
            x = a/(b-a)
            S.o.println (x);
        }
        catch (Exception e) {
            S.o.println ("Exception handled");
        }
        y = a/(b+c);
        S.o.println (y);
        S.o.println ("end of block");
    }
}
```

o/p
→ Exception handled
1
end of block.

~~MC questions~~

II Multiple catch block:

```
Public class multiplecatchblock {
    p. s. vim (string args[]) {
        try {
            int a[] = new int[5];
            a[5] = 30/0;
        }
        catch (Arithmetic exception e){
            s.o.p ln ("Arithmetic
                exception occurs");
        }
        catch (array index out of bound
                exception e){
            s.o.p ln ("array indx out of
                bound exception occurs");
        }
        catch (exception e){
            s.o.p ln ("exception occurs");
        }
        s.o.p ln("rest of the code");
    }
}
```

III finally: [finally { } ... point ...]

```
package javaExceptionExample,
    public class javaExceptionExample {
        public static void main (str args[])
        {
            try {
                int a=10;
                int b =10;
            }
            catch (Exception e) {
                s.o.p ln ("Exception handled")
            }
            finally {
                s.o.p ln ("finally block
                    executed");
            }
```

→ Exception handled
finally block executed.

finally block used to execute the necessary code of the prgm.
It is executed whether an exception is handled / not.

IV Throw =

```
package javaExExample,{
    void voter(int age){
        if(age<18){
            throw new ArithmeticException
                ("you are not eligible for
                    voting");
        }
        else{
            s.o.pln("you are eligible");
        }
    }
p.s voidmain (str args[]){
    JavaException Example obj =
            new Exception Ex();
        obj.voter(17);
    }
}
```

→ You are not eligible for voting.

Throw keyword is used to create a custom errors.
Throw statement is used together with an exception type

Exception handle krne ke liye throws keyword use kiya jata hai. ①

V Throws =

```
p. cls Java Ex Example {
    void numcheck (int num) throws IO
① ——  Exception, class not found exception
    {
        if (num ==1) {
            Throw new Io Exception("Io
                    Exception");
        }else{
            Throw new class Not
                Exception ("class not found");
        }
    }
p.s.v.m (str args[]) {
    Java Exception Example obj= new
                    JavaexceptionEx();
    try{
        obj.num check (1);
    }
    catch{
        s.o.pln(e);
    }
}
```

→ I/o Exception.

(Ex) → Exception.
(S) → Stream

* Java throws keyword used to declare an Exception, it gives an info to the prgmr, there may occur an Exception

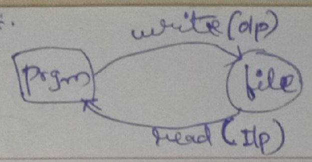
⇒ ## I/O Streams =

* Java prgms performs I/P Stream.
* A stream is a logical entity that can either ~~connect~~ collect frm a I/P device/ handover to an O/P device any data/ info as a flow of a bits / chars.
* A prgm uses an I/P Stream to read data frm a source & it uses O/P Stream to write data to the destination.
* In Java, 3 Streams are created automatically → System.out
  → System.in
  → System.err

* Connection b/w Java prgm & a file
  → I/O Stream → I/P Stream
  → O/P Stream.

* logical connection
* If we are passing binary data then it's binary Stream
* " " char " , then a char Stream.

file → hard disk.



I/O Streams for permanent Storage (hard disk)

Java Streams
- Binary (S)
  - I/P (S) (read data from file)
  - O/P (S) (file → write)
- Char (S)
  - Reader (S)
  - Writer

Process { I/P (S), O/P (S), Reader, Writer }
opr { read(), write() }

* ## 2 types of Stream clses:

* Java Stream based I/O is ~~fitted~~ build upon 4 abstract clses:— I/P (S), O/P (S), Reader & writer
* I/P & O/P (S) are designed for ~~byte~~ binary streams & reader & writer are designed for char (S).

I ## I/P (S) & ~~O/P (S)~~ Reader:

The I/P (S) cls is an abstract cls that defines method to read sequencially form a Stream of bytes.

Java provides subclses for the I/P (S) cls for reading from

file, String buffer objects & byte arrays.

II O/p (S) & writer :

O/p (S) cls is an abstract cls that defines methods to write a (S) of bytes sequencially.

Java provides subclses of the O/p (S) cls for writing to files & byte arrays.

→ **Buffered Stream =**

* It is a storage or to convrt datatype.
  (ഒരു ഫോർമാറ്റിൽ നിന്നും മറ്റൊരു ഫോർമാറ്റിലേക്ക് move ചെയ്യുന്ന Storage)
* working/use → to read I/p. < from keyboard. / from files.

* ~~After reading~~, read char from char I/p stream

(keyword) — InputStreamReader (Convrt bytecode to char)
① ↓
System.in (Buffer ൽ I/p read ചെയ്യുന്നത്)

Str ആയി char-നെ read ചെയ്യാൻ —
* read ()
* readLine () } Buffer reader methods.

---

① ഇവിടെ Char ആയി buffer ആയി store ചെയ്തിരിക്കുന്നു.
അതിനെ access ചെയ്യാൻ use ചെയ്യുന്ന "BufferReader" Keyword.

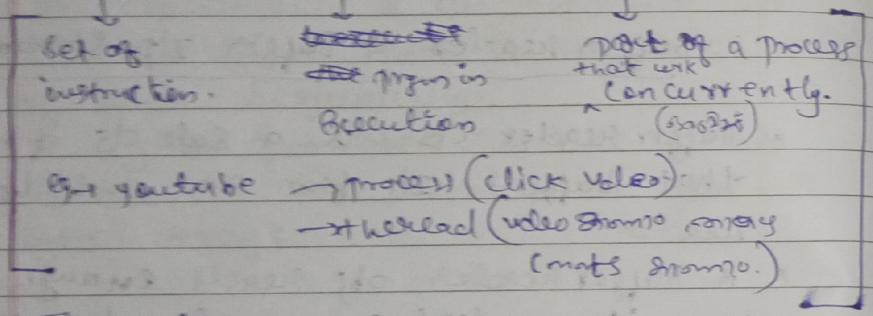
Buffer reader & Buffer I/p Stream :

* Buffer reader cls is a sub cls of reader that buffers I/p from an underlying reader.
* A Buffer reader obj reads enough chars from its underlying reader to fill a relatevly large buffer & then its satisfies read opr by supplying chars that are already in the buffer
* Buffer reader → reads char / str.
  ↓
  2 methods ① read () → read single char
  ② readLine () → read multiple char str.

① InputStreamReader ir = new InputStreamReader(System.in)
BufferedReader br = new BufferedReader(ir);
(access ചെയ്യാൻ)

int a = Integer.parseInt(br.readLine());
float a = float.parseInt(" ");
String str = br.readLine();

→ prgm vs process vs threads:



* prgm:
It is an executable file containing the set of instruction to perform a specific job on yr comp.

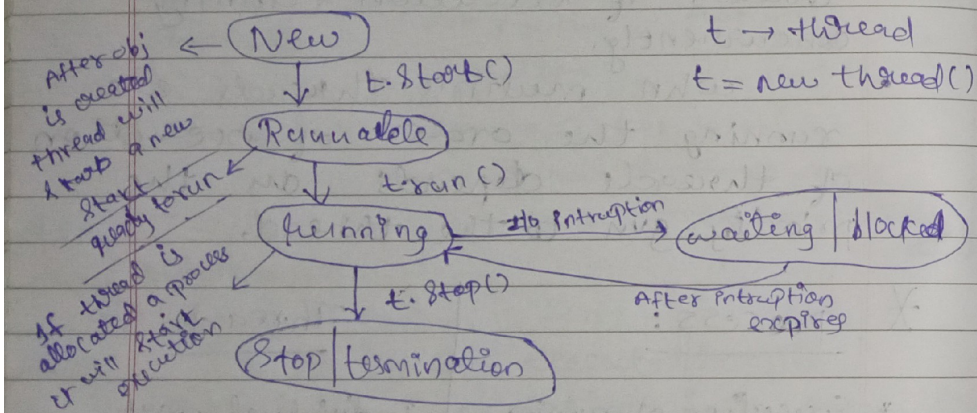
* process:
It is an executing instance of a prgm.
It is an abstract cls defined in the 'java.lang' package, that encapsulates the runtime info of a prgm in execution.

* Threads:
- It is a part of a process
- It is also called as light weight process.
- The Java virtual machine allows an appln to have multiple threads of execution running concurrently.
- When multiple threads are running the order of execution of threads depends on the priority given to the thread.

* process:	Thread:
* Execution of running prgm	Individual part of a prgm
* Each prgm has its own address space	It may share common address
* Heavy weight process	light weight process
* Context switching is high cost	context switching is low.
* Not under control of Java	Under control of Java (bz Thread is a cls in Java)

* CPU is provided inorder to execute many tasks at a time.	Inorder to execute many threads from a process at a time

Thread lifecycle =



1) New =

whenever a new thread is created it is always in the new state. For a thread in the state the code has not been run yet & thus has not began its execution.

2) Active =

when a thread invokes the start method it moves from new state to active state. Active state contains 2 states.

a) Runnable —

A thread that is ready to run is then moved to the runnable state.

Here, the thread may be running/ may be ready to run at any ~~given~~ evnt instnce of time.

b) Running —

when a thread gets the CPU it moves from ~~running~~ runnable to running state.

3) Blocked / waiting =

whnevr a thread is inactive for a span of time then either the thread is in ~~the~~ blocked state or in a waiting state.

4) Stop —

A terminated (t) means the (t) is no more in the system.

The (t) is dead & there is no way i can respan the (t).

→ Creating a thread =

2 ways → 1) inheritnce (2) Interface.

* Acquiring (prop) from prnt cls to ~~[illegible]~~ child cls. (already 1 cls is there)

* class Thread

Single thread ↓ child cls. Thread ↳ prnt cls.

{ class Single Thread, extends Thread.

↓ After obj creation (start() initial [illegible])

↓ then implicitly run. [illegible] (run ())

↓ run-[illegible] [illegible] give the prgm.

1) eg

```
class S.T e.thread {
    public void run () {
        S.o.p ("Hello");
}}
class Single Thread {
    p. s. v. main (...) {
        Single S = new Single();
        S. Start ();
}.
```

Creating a thread ← (S → thread name → a variable)

(start goes to run ().) run () → "Hello"

↓

Creating a thread using inheritnce

2) Interface =

* used to achieve multiple inheritnce & abstraction.
* Only used abstract method — writing only method declaration & not definition.
* 2 keywords → ~~Int~~ interface ↳ implemntation.
* eg →

```
class Single implements Runnable {
    public void run () {
        for (int i=0; i<5; i++)
            S.o.pln (i);
    }
}
class Single Thread {
    p. s. v. m (....) {
        Single S = new Single();
```

Thread t = new Thread (5);
t.start ();
}
}

* Creating thread def:

- The thread ~~class~~ cls in the Java.lang package creates & controls threads in Java prgms.
- The execution of Java code is always under the control of thread obj.
- This can be done in 2 ways—
1) by implementing the runnable interface
2) by extending the thread cls.

I Implementing the runnable interface:-

* The easiest way to create a thread is to create a cls that implemnts the runnable interface.
* Runnable apstracts a unit of executable code.
* The runnable interface defines a a single method 'run' meant to contain the code executed in the thread.

II By Extending the thread cls :-

* The 2nd way to create a thread is to create a new cls that extends & then to create an instnce of that cls.
* The extending cls must overwrite the run method which is the entry point for the new thread.
* It must also → start method to begin execution of the new thread

→ Synchronization = (Syn)

eg→ In a cls, only 1 tr can take the cls at a time

usage→ to avoid inconsistency (...)

working → by using a locking mechanism.

(Syn) (...) ... monitor process ... Activ.

* The process of allowing only 1 thread at a time to complete the task entirely
* Syn in Java is the capability to control the access of multiple threads to any shared resource.

* It is mainly used to prvnt thread intrfnce & prevnt consistncy prbm

* Adv:
- we can execute multiple task of an appln at a time
- reduces complexity of a big appln
- Hlps to improve performnce of an appln drastically.
- utilise the max resources of multi processer system
- Reduces development time of an appln
- All the threads are independnt, any unaccepted exception hpns in any of thread will not lead to any appln excist.

Disadv:
- Thread sync- cally extra overhead to developrs
- shares the common data accross the thread might cause the data inconsistncy.
- Difficult in managing the code in terms of debugging or writing the code

→ Daemon threads: (De)

2 types → non-daemon → to execute logic part of thread
→ Daemon → To provide service

2-to handle JVM
Service provide
Daemon JVM stop otherwise continue its working

* In Java, Daemon threads are long priority threads that runs in the background to perform task such as provide services to the user.
* The life of the (De) depends on the mersy of the user thread, meaning that when all user threads finish their execution the JVM terminates automatically.

(De) thread.

```
* public class ThreadDemo {
      p.s.v.m (...) {
        RunnableDemo r = new RunnableDemo();
        Thread t = new Thread(r);
        t.setDaemon(true);
```

```
        t.start();
        s.o.p/n(t.isDaemon());
    }
}
pulelic cls RunnableDemo implements
                                Runnable{
    p.v.run(){
        s.o.p/n("child thread")
    }
}
```