JAVA

15/2/19.

Electronic gadgets ⇒ connected through network ⇒ to control such scenario

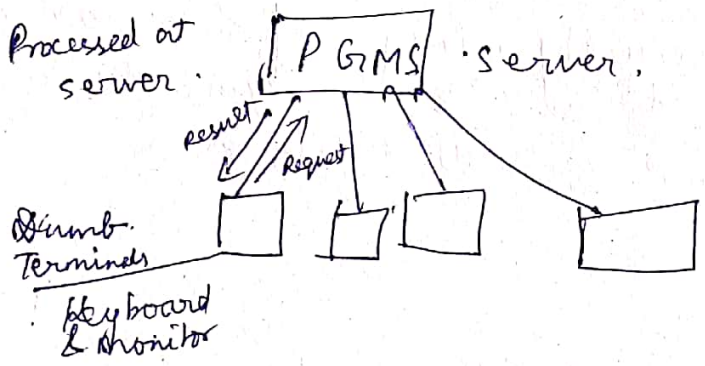
↓

systems with small mem.
Robust against the failures

- JAVA BYTE CODES ARE SMALL
- SOURCES OF ERROR : GOTO, POINTERS ARE AVOIDED.
- STRONG EXCEPTION HANDLING CAPABILITY.

Scientists at switzerland (defence) → communicating among the labs

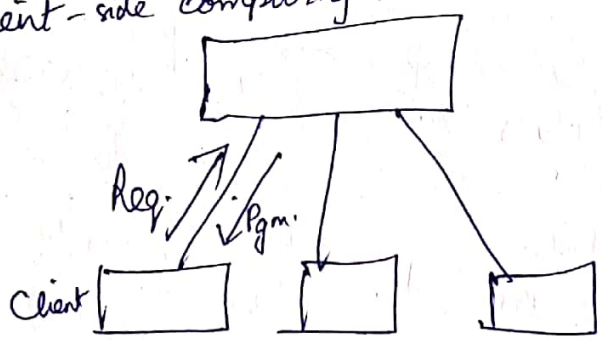
CLIENT – SERVER computing.



server side computing.

Server is providing services to no. of clients.

Client-side computing.



↓ gets the prgm. and computes the result for itself.

Bytecode:- Java source code translated into bytecode which is an intermediate code & does not correspond to any physical machine

→ Bytecode corresponds an imaginary m/c called JVM

Java source code → Bytecode is transferred to client

↓ Compiled → Java run time environment understands it.

Bytecode

↓

Client side

↓

Interpreter → Translates bytecode to object code specific to the environment.

Security ⇒ Security manager of client environment dictates what the Apn can do

It cannot work with local file system.

Specialised Interfaces.

Interfaces → LOTS OF GUI elements

Handling / appearance of such elements may not be same for diff. environment.

JAVA provides generic classes for such components. It will interact with corr. class in the client's peer

JAVA IS
- SIMPLE
- Object oriented
- Network Savvy
- ROBUST
- interpreted
- SECURE
- Architecture neutral
- Portable.
- High Perf.
- Multithreaded
- Dynamic

Simple:
1) No preprocessor.
2) Rich Library support.
3) No. operator overloading
4) No global var/ff.
5) No structure.
6) No pointer
7) No, go to

Object oriented.
1) No thing can be done without having a class.
2) ~~Everything~~

Network Savvy
1) Provides lot of support.
2) Related to network prog.

Interpreted.
1) Bytecodes are interpreted.

Robust
1) It has strong exception handling features.
2) avoids the feature that are common sources of erro

Secure
1) Governed by security manager.

Architecture neutral.
1) Bytecode is not specific to any h/w.

High Performance
1) Interpreter ⇒ slow
So Just in time compiler.

Multithreaded.
1) Supports multithreaded programming.

Dynamic
1) Works across internet

Java source file.

—— .java

A source file can have multiple class definitions

- But atmost one public class.

filename : <public class name>.java

- ~~If there is a class~~ In a file, only one class can have main(). filename: <class with main()>.java

---

Java code.

- application → there is a main function → Standalone
- applet → included as a part of html file. → It has to be one. through a browser. → mention name of class in the <applet tag>
  an applet does not have any main()

$javac Firstprog.java. corr. to each class a <> class file is invalid.

$java Firstprog

↓

classname containing main().

public static void main (String args [])

- public → accessing outside the class.
- static → Can be called without any object.
- void → return type
- args [] → Or Bit this, first element will be the value provided on execution

main (char *argv [], int argc)

- argv → first element will show the command/filename.

→ All the member fns in a class are to be defined inside the class.

By default in C++ is private.

JAVA ; by default its like public

```
Public static void main (.....) {
  Student S;  → S is a ref. to Student.
```

System.gc(); → for garbage collection
→ removal of unreferenced objects.

Datatype

Numeric data type

- byte ⇒ 1 byte
- short ⇒ 2 bytes.
- int ⇒ 4 bytes
- long ⇒ 8 bytes.
- float ⇒ 4.
- double ⇒ 8

char → 2bytes

boolean → 1 bit.

boolean x = (true / false)

Numeric data type.

```
int x;
float f;
x = f;
```

Java

↓

type-safe.

If there is loss in precision ⇒ error

widening conversion is allowed (implicit)

implicit narrowing conversion

↓

error.

```
x = (int) f.
```

```
Sample x, y;
```

Reference assignment → x = y.

x & y referring to same object.

```
if (x == y)
```

⟹ not content.

two objects referring to same.

```
Sample {
  int a;
  public boolean equals (Sample t)
  {
    if (a == t.a && ---)
      return true;
    else
      return false
  }
```

System.out.println(String)
concatenated using +.

System.out.println("Object is:" + obj);

⇓

Object ref.
↳ toString()
in the
Object
class.

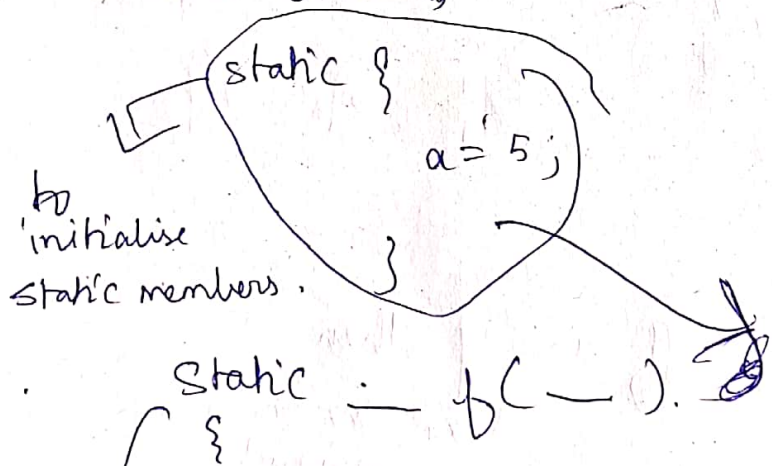
Arguments.

⇓

An object is always passed by reference.
object reference.

A primitive data type ⇒ by value.

Static block



To call static mem.
classname.<static members>

Static — f(—).
{
}

there will be no invoking instance.

Constructor

→ default constructor

↓

constructor with no argument non static

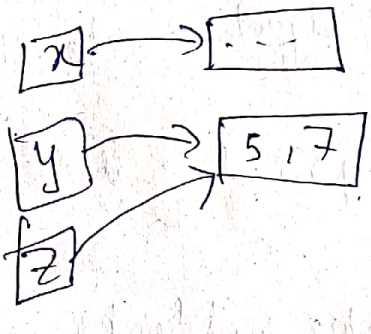
Sample x = new Sample();

↓

object creation.

In sample class:

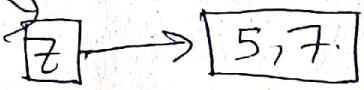
```
Sample (int i, int j).
{
    Sample y = new Sample(5,7);
}
Sample z = y;
```



If not written

Copy constructor (if written)

```
Sample (Sample t)
{ a=t.a;
  b=t.b;
}
```



If x = y then then the memory previously pointed to by x then →

Giaribage collector

Jcwa run time system periodically runs garbage collector. If an object is there which is not being referred will be removed by the garbage collector

No destructors ~~apparently~~.

Explicit removal of orphaned memory.

→ System.gc();

---

finalize → similar to destructor in C++

protected void finalize (){ }

before the collection of an object by garbage collector, this is automatically called.

default

public → public in subclass.

private → inherited but not accessible in subclass.

protected → protected

A super

↑ → A(A t){ }

B sub.

B(B t)

{ super(t);

}

Super class reference can be assigned with sub class reference.

Super class ref. can be assigned with sub class reference.

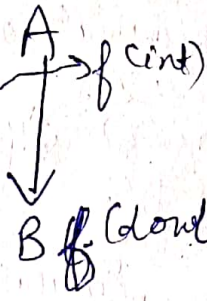
Super class reference can refer to both super class object & sub class.

super ⇒ constructor

super. super class members

In sub class trying to access members of super class with same name.

int i;
double d;
B b;
↳ subclass ref.
b = new B ( );
b.f(i);



across the class (betⁿ super & sub) fⁿ. overloading occurs.

Expⁿ involving short/int/byte.
↓↓
all will be int operⁿ

```
short a,b,c;
int x
c = a+b;   } → error
   int
x = a+b;
```

var = (rem. exprⁿ.)
mixed type.
↓↓
will be converted to highest precision involved.

final class sample{
} ↳ cannot be inherited further.

final method f(int){ }
↳ cannot be overridden

Function overloading in case of inheritance.

```
class A {
  f(int, int)
  {
  }
  g(int, int).
}
```

```
class B extends A {
  f(int)
  {
  }
  g(int)
}
```

function overriding in C++ same func name

```
int x, y;
B b = new B();
b.f(x, y); → call to f(int, int) of class A.
```

C++
fn overloading across the class is not allowed.

Java
fn overloading across the class is allowed.

Java
fn overriding means → define a superclass fn in subclass with same name & sign.

C++
Runtime polymorphism override base class fn. is derived class with same signature.

Runtime polymorphism: Dynamic method dispatch (super class ref.)

Super class ref. can refer to superclass or subclass object. Eg:
```
A a;
a = new A();
a = new B();
```
Using this access is limited to super class portion (runtime poly.)

From ① a.g(x) X
a.f(x) → call to f() in B (if overridden) else goes to A always.

Final Keyword

final int x = ___; constant.
final method ⇒ cannot be overridden.
final class ⇒ cannot be extended further.

Abstract class

A class will have atleast one abstract method
Only prototype declaration and preceded by the keyword abstract.
Such class can have non-abstract / concrete methods.

```
abstract class Sample
{
    abstract void show(void); //at least
                              // one needed.
    ...f()
    {
    }
};
```

Static methods & constructors cannot be made abstract.
Any class that inherits an abstract class must implement all the abstract methods.
Otherwise it is also to be declared as abstract class.

abstract_A → abstract f1()
             abstract f2().

B.

(If f2 not)
```
_f2(__)
{
}
```

abstract_A a; // can be written (since it is a reference)
(same as abstract_A *p; in C++)

a = new abstract_A();
a = new B();

Nested Class

A class can be defined inside another class.

```
class A{
    class B{
    }
}
```

Nested class → static
Nested class → Non-static (inner class)

- Inner class is defined within the scope of enclosing /outer class.
- Inner class is not visible outside the scope.
- Inner class can access all the members of outer class
  Outer class can access public members of inner class.

```
class Item
{
    int icode;
    string iname;
    CostDetail cost;
    Item (int ic, string in,
          float bp, float tr)
    { icode = ic;
      iname = in;
      cost = new CostDetail(bp,tr);
    }
    void show(void);
    { cost.show();
    }
```

```
private class costdetail
{   (private only for def. of inner class).
    public float baseprice;
    public float tanrate;
    costdetail(-,-)
    {
    }
    costDetail()
    {
    }
    public void show();}
```

A class whose object is used only as a component of another class, make the class a private inner class.
(used in GUI application)

A → outer class
B → inner class
On compilation
A.class :
B$A.class

```
import java.util.Scanner;
Scanner s = new Scanner (System.in);
String st = s.nextLine(); to take string input
            nextByte();
            nextInt();
            nextDouble();
```

Package

Two classes cannot have same name. If same name is used collision occurs.
Difficulty in providing meaningful names to the classes avoiding the clash.

package ⇒ * partitions the namespace.
In diff. package same name can appear

- A package is a collection of classes, sub-packages and interfaces. → Related classes that work for common task. (package deals with I/O, network prog.)
- Package provides further visibility control.

file1.java

```
package LibraryMember;
class Person
{
}
class Member extends person
{
}
```

- Package may span over multiple files.
- Packages are wrapped to the file system directories.
- Must have a directory with the package name.

file2.java

```
package LibraryMember;
class ---
{

}
```

```
$mkdir LibraryMember
$cd LibraryMember.
$vi file1.java
$javac file1.java.
```

```
$mkdir lib
$cd lib
$mkdir mem.
$cd mem.
$vi (——)
$javac (——).
```

→ Package of type Library member.

→ package LibraryMember.

```
$ java Runpkg ↵ (error)
$ cd ..
$ java LibraryMember.Runpkg ↵
```

OS has CLASSPATH = —;—;—;—

To work with elements of some packages:

import x.y.z;

pkg.subpkg.z.class

x.y.* (compiler will search what all classes we have referred to and include only those but all classes not used).

Logindirectory under which directory for my package are created.

CLASSPATH = $CLASSPATH;—

(current directory has been always included).

A file can have multiple import statement.

If two packages with same class then we need to refer to that class with a qualifier

Class

public

Caccessible everywhere

visible ")

(not more than one public class in a prog.) ??

default / (none)

visible only within the package

Members

Members.

Data members or methods (accessibility / visibility)

For public class (visible)

Members Category	Within the class	Within subclass of same pkg.	within subclass of other pkg	with subclass of other pkg.	within non-subclass of other pkg
private	✓	✗	✗	✗	✗
default	✓	✓	✓	✗	✗
protected	✓	✓	✓	✓	✗
public	✓	✓	✓	✓	✓

Interface — (similar to abstract class).

- To provide the design guidelines
- Consists of no. of method prototype declaration without any implementation.
- All data members are static & final.

```
(access specifier) interface my_interface
{
    declare the data & members.
}
```

(public / default / not specified)

```
class A implements my_interface.
{
    → must implement all the methods of the interface.
}
class A Incomplete implements ____
{       ↳ abstract class.
}
```

one can have reference of interface

my_interface m; ⇒ can refer to object of the classes
m = new A(); that implement the interface.

m. ⇑
members of the interface only.

- A class can implement multiple interfaces.

class A implements I1, I2 extends B
{
:
}

(In both only one ref of a function should be present

② I_1 interface I_2 extends I_1
⇕
I_n

Array

An array object holds elements of same type.
public data member ⇒ length

```
int[] x = new int[5];
int x[] = new int[5];
int x[] = {1,2,3,4};
int x[] = new int[_];
int y[]; y = x;
```
reference assignment

Reassignment

```
int x[] = new int[0];
x = new int[5];
for(i=0; i<x.length; i++) { }
```
public member

```
int[][] x = new int[5][4];
x[][] = {{-,-,-},{-,-,-}}
```

x is a reference to array → array of int.

To find sum:

```
for(i=0; i<x.length; i++) s = s + x[i];
```

OR

```
s=0;
for(int v: x) // in each iter takes one element, puts into v
{ s = s+v;
}
```

Contd.

→ working with one array
→ accessed sequentially
→ subscript is not known.

Variable Arguments :

```
int f (int ... x)
{
   s=0;
   for (int v: x)
   {
      s=s+v;
   }
}
```

```
int a, b, c;
f(a); f(a,b); f(a,b,c)
```

Ragged tray All rows may not be of same size.

```
int [][] x = new int [5][];
for (i=0; i<x.length; i++)
{
    x[i] = new int [k];
}
```

String

```
string st1 = "____";  ____ string object is created.
String st2;
   st2 = st1;
string SH = "  ";
   or
   = new string ("  ");
```

st1.length() => length of the string.

```
String x[] = new String [5];
   x[i] = "____";
```

array reference → x.length

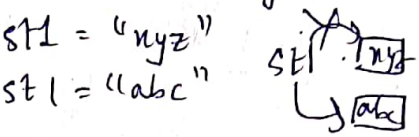
x[i].length()

string.

char charAt(index);
String toLowerCase();
String toUpperCase();
String trim();
String replace(char1, char2)

* once created a string object cannot be changed (immutable)

st1 = "xyz"
st1 = "abc"



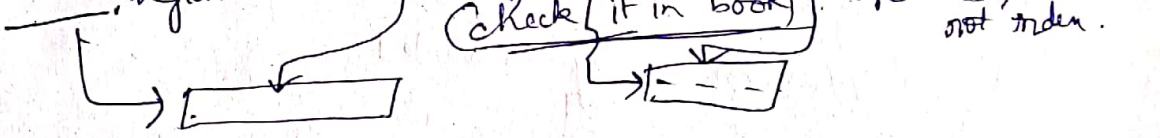
trim() removes all leading & trailing white spaces.

st.replace('a', 'x') // all occurance of a will be replaced by x & new string is returned.

Substring Matching

boolean startsWith(string) | s1.startsWith("abc");
boolean endsWith(string) | s1.endsWith

regionMatches(start1, string to search, start2, n) — How many characters



returns boolean not index.

substring:
substring(start)
substring(start, end)
[start, end).

toCharArray()
getCharArray(start, end, char array ref, start2)
indexOf(char)
↓
1st occurance of character if not found returns -1.

indexOf(char, start)
lastIndexOf(char)
(char, pos n)
indexOf(string) // (string, start)
lastindexOf(String)
(string, pos)

to compare string objects based on content.
boolean | st1.equals(st2);
st1.compareTo(st2)

Static

valueOf( ); // Returns a string obj. with basic/pri. type, as its content
↓
basicType

Wrapper Class

Corresponding to each primitive type → wrapper class

char → Character
byte → Byte
float → Float
int → Integer.

Wrapper Classes, holds the value of primitive type & also provide number of methods to work with the objects

Convert primitive type to object

int i)

Integer I = new Integer(1); // primitive class object to pass as ref.

I = 5; // automatically converted.

⇓

boxing.

i = I; // unboxing.

Object to primitive type

Integer I = new Integer(_);

int i)

i = I.intValue()

type → float, Byte, ...

→ primitive object to String object

Static

String Integer.toString(int)
or
String String.valueOf(int)

Float.toString(float);

String to primitive type.

Integer.parseInt(String)
Float.parseFloat(String)

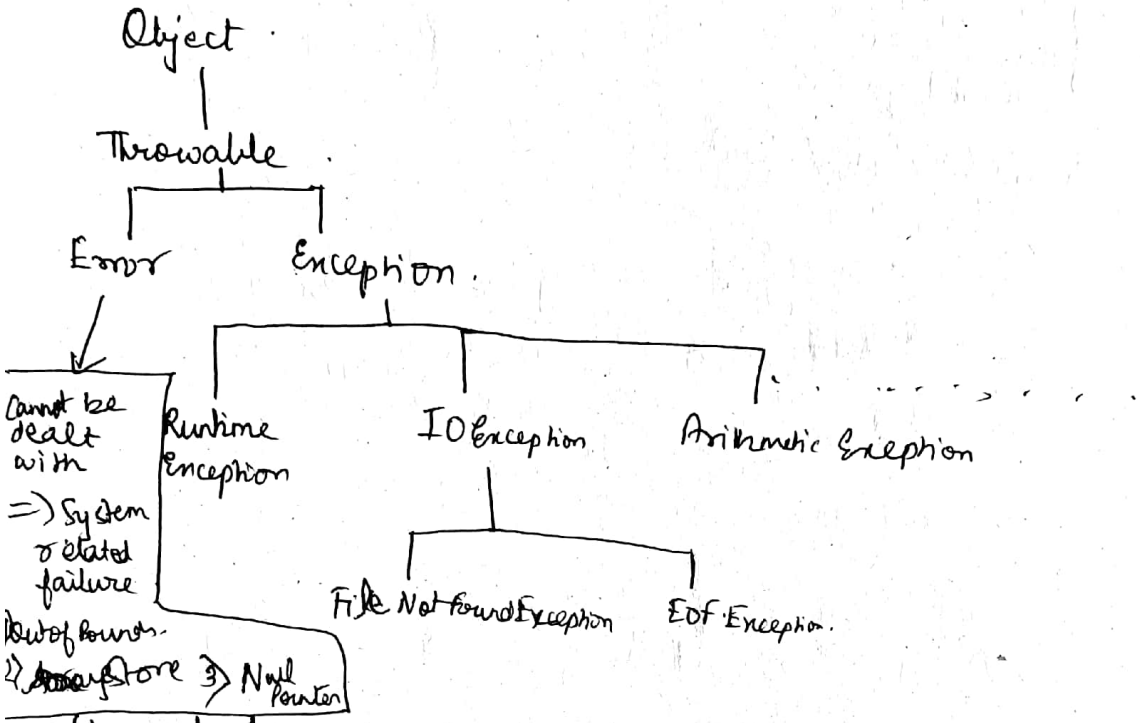
Integer.MIN_VALUE
, MAX_VALUE

Character

boolean { isLower(ch), isUpper(ch), isDigit(ch), isWhiteSpace(ch) } — Char

char toLowerCase(char)

Exception Handling



Unwanted situations may be caused by programmer OR due to system failure

End user

⇓

Improper Input.

JVM running out of memory

File does not exist, measure not taken by programmer.

Exception

Arithmetic Exception

checked

If not handled then detected at compile time

Bound to address checked exception

unchecked

Occurs at the time of execution

If not handled, compiler will not report but abrupt termination occurs at the time of execution

To handle exception,

```
try {
   =
   =
   =
}
catch
{
  }
catch{

  }
finally{
  }.
```

followed by try block atleast one catch block OR finally block must be then to handle the exceptions occurring in try block.
If not handled

In try block of a method
Exception occurs.
↓ yes
does it handle?
Yes → any suitable try block?
Yes → Execute → any finally block.
Yes → Execute
No → unhandled
No → any finally block.
Yes → Execute
No → unhandled
No. → any finally block.
Yes → Execute
No → unhandled

catch blocks
⇓
Checked sequentially during argument matching

Catch
⇓
argument of 'Exception' type.

```
catch (Exception e)
{
}
catch (NullPointerException)
{
}
```

control will never be checked

Handler with more general exception object as argument must be placed later, specific ones are to be placed at the beginning.

```
catch (Exception e)
{
    e.getMessage();
}
```

finally block : helps to trap the other exception not caught in the previous catch blocks.

↳ Why finally {} over simple sequential statements

```
void f() throws ArithmeticException
{
    try{
        ....
    }
    // (If no catch) then compiler identifies it as 'Checked Exception' and gives error
}
```

class myexception extends Exception

File Handling.

① TextFile.

raises IOException

```
PrintWriter outf = new PrintWriter (filename);
    outf.println ( );
    outf.close();
```

ASCII file

if file is existing then old data is lost

② to append data in existing textfile.

may IO Exception

```
FileWriter f = new FileWriter (filename, true);
PrintWriter pw = new PrintWriter (f);
    outf.println (_____);
```

1) to append

eg. false → ~~overwrite~~ erase the existing

③ Read data from text file.

IO Exception

```
File f = new File (filename);
Scanner s = new Scanner (f);

To check the end of file.
while(s.hasNext ( ))
{

}
```

File class

- ↳ .canRead ( )
- ↳ .canWrite()
- ↳ .isDirectory ().
- ↳ .exists()
- ↳ .list() → gives String[] list of all sub-dir.
- ↳ .listFiles() → gives File[]
- ↳ .isHidden()

Binary files. (IO Exception).

→ To store data.

```
FileOutputStream f = new FileOutputStream(filename);   // path
DataOutputStream d = new DataOutputStream(f);
    d.writeByte(_);
        // Int(_)
        // float(_)
    d.writeUTF(_); // For strings.
d.close();
```

Old data is lost.

To retain old data (append)

```
FileOutputStream f = new FileOutputStream(filename, true);
```

→ To read data.

```
FileInputStream f = new FileInputStream(filename);
DataInputStream d = new DataInputStream(f);
    boolean eof = false;
    while(!eof)
    {
        try{
            d.read();
        }
        catch(EOFException e)
        {
            eof = true;
        }
    }
```

readByte
// Int

Writing user defined classes object into file.

Serializable

```
class Student implements Serializable.
{
    int roll;
    String name;
    Address ad;
```

→ this class must also implement Serializable

Any user defined class object must implement serializable to write/read to/from a file.

↓

no data members & no method.

↓

It gives an intimation to compiler to serialize the object data.

To write whole object

```
FileOutputStream f = new ____(filename, true)
ObjectOutputStream o = new ____(f);

o.writeObject(myObj);

FileInputStream f = new ____(filename, true);
ObjectInputStream o = new ____(f);

o.readObject(myObj).
```

Thread.

multitasking

- Process based
  - (Parent & child) ⇒ diff. Context
  - Context switching time consuming
- Thread based.
  - (parent & child) ⇒ same content.
  - Content switching is not an issue.

Inter process communication is difficult than Inter thread communication.

Thread is a light weight process.

In C# — Posix Thread PThread.

In Java — Thread

① What it will do? → code } Specify.
② On what it will work? → data }

To create threads.

- Extending the Thread class
- Implementing the Runnable Interface (it implements Runnable interface)
  - preferable. because we shall be having the freedom to extend another class

Thread — Implemented → Runnable.

Thread — Extend.

mythread

```
class MyRunnable implements Runnable.
{
    String x;
    MyRunnable()
    {
    }
    public void run()
    {
    }
}
```

code for the thread.

In main()

```
MyRunnable ro = new MyRunnable(----);
Thread t1 = new Thread(ro);
t1.start();
if(t1.isAlive())
    t1.join();
```

Runnable object.

code and data is made available to the thread

In the corresponding class there is a run() which will be the code for the thread.

data.

local variable of run() and the Runnable instance data.

A new thread is created alongside main() Thread

But we have to make sure that the main() thread does not finishes before the thread finishes

class MyThread extends Thread
{
{data}
public void run(){ }
MyThread() { super(this); start(); }
MyThread(——) { super(); }
}
MyThread mt = new ——();
Thread t1 = new Thread(mt);
t1.start();

To create 2 threads with same code & same data.

MyRunnable r1 = new MyRunnable();
—— r2 = new ——;
Thread t1 = new Thread(r1);
Thread t2 = new Thread(r1);
Thread t3 = new Thread(r2);

Two different threads
↓
different code

class MyRunnable1 implements Runnable
{ Mydata d;
constructor
public void run()
}

class MyRunnable2 implements Runnable
{
Mydata d;
constructor
public void run(){ }
}

class Mydata {
constructors
}

In main

```
MyData md1 = new MyData();
MyData md2 = new MyData();
MyRunnable1 r1 = new MyRunnable1(md1);
MyRunnable2 r2 = new ____2(md2);
Thread t1 = new Thread(r1);
Thread t2 = new Thread(r2);
t1.start();
t2.start();
```

same data diff. code.

→ t1.stop().
after this we cannot make a call t1.start().

→ t1.getName().

→ currentThread() // Returns the ref. to the thread running.
(static method).

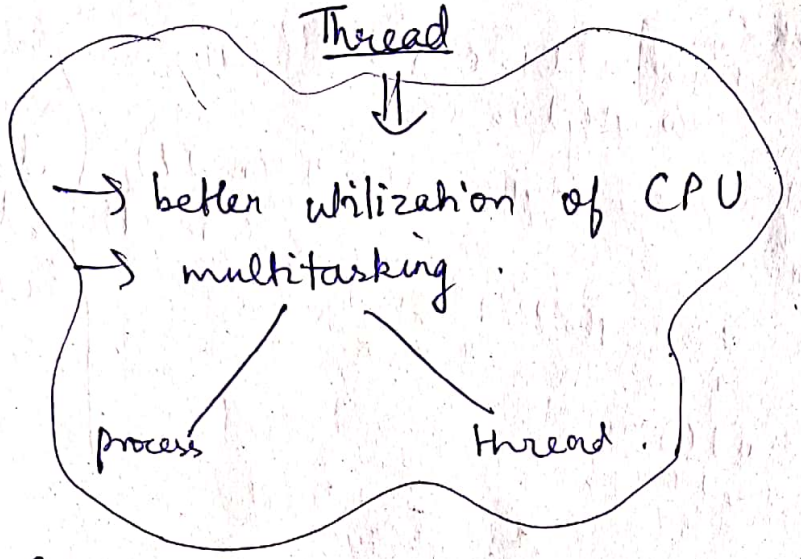
→ t1.setName()

```
main()
MyRunnable r1 = new MyRunnable(____);
____ r2 = ____
Thread t1 = new Thread(r1);
Thread t2 = new Thread(r2);
t1.start();
t2.start();
```

Sleep() is a static function of thread class.
throws. InterruptedException.

```
try {
    Thread.sleep(1000);
}
catch (InterruptedException e) {
}
```

not a proper way to ensure that t1 & t2 finishes before main.




```
if (t1.isAlive()) {
        t1.join(); // main thread waits till t1 ends.
}
if (t2.isAlive())
{
    t2.join();
}
```

How different threads are scheduled is OS dependent, hence to some extent JAVA is platform dependent

Mutual Exclusion

Shared data accessed (may be modified) by no. of threads simultaneously

Improper updation
Race condition.

to avoid → access has to be serialized.

```
class Mydata
{
    int a;
    public void modify (int k)
    {
        a=a+k;
    }
    Mydata (int i)
    {
        a=i;
    }
    int ret_value()
    {
        return a;
    }
}
```

this is indivisible

Execution is to serialized one at a time
mutual exclusion

```
class MyThread implement Runnable
{
    Mydata d;
    Mythread (Mydata d1)
    {
        d = d1;
    }
    public void run ()
    {
        for (i=10; i<=100; i+=3)
        {
            d.modify (i);
            S.O.Pln (d, retvalue);
        }
    }
}
```

```
Mydata d = new Mydata (5);
Thread t1 = new MyThread (Int);
Thread t2 = ———— (Int);
    t1.start();
    t2.start();
```

MyThread Int = new MyThread (d);

Declared the method which should work acc. to mutual exclusion principle, declare it as

```
synchronized public void modify (int k)
{
}
```

Only if both the threads are using the same object

If we cannot make modification to the mydata class,

then

in

```
public void run()
{
    for (i=10; i<=100; i=i+3)
    {
        synchronized (d) {
            d.modify(i);
            S.O.Pln(d.ret_value());
        }
    }
```

→ object on which lock of mutual exclusion to be placed

Deadlock.

Communicate

```
class Data
{
    int x;
synchronized void set(int a){
        x=a;
    }
synchronized int get(){
        return x;
    }
}
```

```
class SetThread.
{
    int x;
    boolean consumable = false;
synchronized void set(int a){
        if (consumable)
            wait(); // only inside synchronized
        x=a; consumable = true;
        notifyAll(); //  //
    }
synchronized int get(){
        if (!consumable) wait();
        consumable = false;
        int v = x;
        notifyAll();
        return(x);
    }
```

main

Data

SetThread

GetThread

Thread

Thread

A Thread

functio

```
class SetThread implements Runnable
{
    Data d;
    SetThread (Data x){
        d = x;
    }
    public void run()
    {
        int i=1;
        while(true) {
            d.set(i); i++;
        }
    }
}
class GetThread implements Runnable
{
    Data d;
    GetThread (Data x){ d=x;}
    public void run()
    {
        while (1) {
            S.O.Pln(d.get());
        }
    }
}
```

Producer.

producer (set) produces value and contd. further produced till consumed.

consumer - consumes if anything is there for consumption.

Consumer.

main

```
Data d = new Data();
SetThread sr = new SetThread(d);
GetThread gr = new GetThread(d);
Thread st = new Thread(sr);
Thread gt = new Thread(gr);
```

till here

2nd class Test.

A thread can make progress within a synchronized function only when it has a lock.

.wait()

Interthread Communication.

wait()
notify()
notifyAll()
] Synchronized block.

Reader Writer Problem.

on same resource (data)

Reader can access => no writer is working on data & no pending write request.

Writer can access => if no read/write is currently working on data

```
class RdWtLockPolicy
{
    private int rcnt = 0;
    private int wcnt = 0;
    private int wrq = 0;
    synchronized public void readLock(){
        while ( wcnt > 0 || wrq > 0)
        {
            wait();
        }
        rcnt++;
    }
    synchronized public void readUnlock() {
        rcnt--;
        notifyAll();
    }
    synchronized public void writeLock()
    {
        wrq++;
        while (rcnt > 0 || wcnt > 0) {
            wait();
        }
        wcnt++;
        wrq--;
    }
```

```
synchronized public void writeUnlock()
    {
        wait--;
        notifyAll();
    }
}
```

```
class Data {
    int x;
    RdWtLockPolicy l;
    public Data() {
        x = 0;
        l = new RdWtLockPolicy();
    }
    public int read()
    {
        int k;
        l.readLock();
        k = x;
        l.readUnlock();
        return(k);
    }
    public void write(int k)
    {
        l.writeLock();
        x = k;
        l.writeUnlock();
    }
}
```

To provide priority to writer and to avoid starvation of writer

```
class Reader implements Runnable {
    Data d;
    // constructor()
    // run().
    k = d.read();
}
```

Documentation of previous 2 pages.

Level Top Level Top-1:
list of objects

Read/Write if locking the list. then only one object can be operated on.

If object level is locked, then other object can be operated on also.

(*) Level at which mutual exclusion is applied is a important issue.

Built-in Classes

StringBuilder, StringBuffer } like String but mutable

Content of such objects can be modified.

similar But, methods of StringBuffer are synchronized & those of StringBuilder are not so.

StringBuilder

StringBuilder sb = new StringBuilder(); ①
allocated space of 16 character

new StringBuilder(Size); ②
Size → space

sb.capacity() of
↳ size. (size' no. of char.

sb.length() ⇒ 0
no. of actual char. stored.

new StringBuilder(String). ③
↳ allocation String length + 16

Sb = "ab cd" X

majority of string functions are available in StringBuilder.

sb. append (string)
⇓ appended.

sb. replace (start, end, string)
[start, end) ↳ replacing string.

sb. delete (start, end). → [start, end)

sb. deleteCharAt (position)
sb. setCharAt (posn, char).

String Tokenizer.

breaking a string into no. of components (tokens)
⇓
tokenizing.

delimiters.
⇓
separating the components.

import java.io.util.StringTokenizer;

String s;
s = ——;

StringTokenizer st = new StringTokenizer (s);
= new StringTokenizer (string1, string2);
to be tokenized.

delimiter characters.

"@."

zyxwyz.me@gmail.com
Tokens :—

boolean hasMoreTokens().

countTokens()
⇓
no. of tokens.

nextToken()
⇓
Returns the next token.

while (st.hasMoreTokens()) {
    S.O.Pln (st.nextToken());
}.

String[] split() in String class.
split(delimiter String).

((and))
↑↑↑↑↑
all together constitute the delimiter.

" [@./-] "
either one of them
2) Regex.

ArrayList (methods are not synchronized)

S.O.Pln (arrayList objects)

java.util.ArrayList

Like array, is a collection of objects
- unless specified, can hold objects of different types
- size can grow & shrink.

ArrayList list = new ArrayList();
⇓
can hold 10 objects.
without further growing, can store 10 elements.
11th element ⇒ expansion (automatic)

ArrayList(n)
↓
capacity.
list.capacity() ⇒ no. of elements it can store without growing.

size() ⇒ no. of elements present.

add (element) → to append element
add (inden, element) => to insert at inden.
remove (inden) => to remove element at inden.
get-(inden) => returns the element at inden

(Object class 'Object')

→ x = ()list - get (inden)

↓ casting 'Object' type to 'desired' type.

ArrayList <String> list = new ArrayList <String>();

ArrayList <int>

X

|| <Integer>

for (String s : list)

{

}.

Vector (methods are synchronized)

addElement (element)

addElementAt (inden, element)

removeElement (element)

removeElementAt (inden)

copyInto (Array)

size(), capacity();

toString

equals.

Date

epoch

Hash

HashMa

( ) h

casting

boolean

boolean

Date — java.util.Date

Date() creates date object initialized with current date & time.

String toString()

Date d = new Date()

epoch ⇒ 1st Jan 1970 00:00:00 millisecond.

d.getDate() ⇒ day of the mth

d.getDay() ⇒ 0-6 (0 - may be Sunday)

d.getMonth() ⇒ month 0-11

getHour()
getMinute()
getSeconds()

d.equals(__)
__.before(__)
__.after(__)

~~Date~~ ~~d1 = d.getTime();~~

int k1 = d.getTime();

Date d2 = new Date();

k2 = d2.getTime();

(k2 - k1) time for this task

HashMap → not synchronized

key Value Pair
Object Object

Synchronized HashTable

HashMap hm = new HashMap();

hm.put("abc", new Integer(5));
String

( ) hm.get(key)

casting

boolean ~~key~~ containsKey(key)

boolean containsValue(value)

entrySet( )

↓

a set of all entries in the HashMap.

→ Set set = hm.entrySet();

→ Iterator i = set.Iterator() ↳ one element is of type Map.entry

Map.entry m;

while (i.hasNext())

{

m = (Map.entry) i.Next()

m.getKey()

m.getValue()

}

HashMap<T1, T2> hm = new HashMap<T1, T2>( );

↓

key

In class T1

{

≡ attr.

int hashCode() {

{

≡ returns bucket number

}

boolean equals (key){ } .

}

When type of key & value is specified;

for (Map.entry m : hm)

{

}

Stack → push (element)

pop() , peek() isEmpty(), size()