JAVA

→ Small and simple.
→ Object Oriented
→ Network Savy
→ Interpreted
→ Architecture Neutral
→ Multithreaded
→ Portable
→ High Performing
→ Dynamic
→ secured
→ Robust

→ Instructions are such that can be translated into one or two byte codes.
→ (friend func, operator overloading,) confusing features are dropped.
→ Class is the basic unit of programming.
→ No variable or function can be outside the ~~Co~~ class.
→ Rich support for ~~net~~ programming in network enviroment.

JAVA Source Code $\xrightarrow{\text{Compile}}$ BYTE code (Not specific to any physical machine)
Translation is w.r.t a virtual machine called JVM.

→ At the time of execution, (at client side), interpreter takes byte code and translates it specific to the client machine.

→ Language provides the 1st layer of security.

→ No Goto

→ supports multi-threaded programming that increases system throughput.

→ Same byte code can be ported to other machines

→ JIT (Just in time compiler).
converts the whole byte code into object code specific to the machine.

JAVA

Application
- Stand alone program
- directly executed
- write the application
- Compile it
- Run it using JAVA interpreter / JIT compiler

Applet
It is placed in HTML file using Applet tag.

<APPLET code = JAVAFILE.class>
</APPLET>

- Can be tested with Applet viewer too.
- Java code can be executed on applet viewer too.

BASIC DATA TYPES:

- Integers
  - Byte (1 byte)
  - short (2 byte)
  - int (4 byte)
  - long (8 byte)
- Real
  - float (4 byte)
  - Double (8 byte)
- Char (2 byte)
- Boolean (1 bit)

→ short and byte, both are converted to int during arithmetic operations.

short/byte x, y, z;

y+2 (Implicitly changed to int)

x = y+2; error
short int

→ Narrowing conversion is not automatic.

int x;
float y;

x = y; error // Narrowing conversion
→ we need explicit casting.

int x;
short y;

y = (int) x; ✓

Display :

System.out.println(string); → Breaks the line after printing.
System.out.print(string); → Does not break the line.

```
System.out.println("abcd");
System.out.print("efgh");
Syst.out.print("ijkl");
```

Output:

abcd
efghijkl

Take Input:

```
import java.util.Scanner;
```
=> Scanner is an utility.

```
Scanner S = new Scanner(System.in)
```
↳ Standar I/P (keyboard)

```
S.nextLine()
```
↳ to take string input.

```
int n;
```
n = S.nextInt() → to take integer input.
nextFloat().

CLASS:

```
class Sample
{
```
=> All methods are to be defined inside the class

```
}
```
No semicolon.

```
Class Sample
{
    public static void
```

public static void main (String args[])

void → Return type

public → Accessible everywhere
As called from outside

static → member fn / class method
to be called without any object.

Inheritance

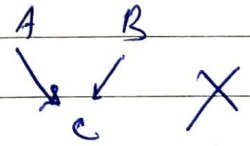
	C++	JAVA
A	Parent	Super class
↓		
B	Child	Sub class

private - not accessible in sub class
protected - accessible in sub class
default - accessible (within package)
public - accessible everywhere.

```
class B extends A
{
    // statements
}
```

→ Multilevel inheritance is allowed

→ Multiple inheritance not allowed



→ If a super class method is redefined in a sub ~~Child~~ class

- Signature is same → overriding
- Signature is not same → Overloading.

C++

Across the class func. overloading is not possible.

JAVA

Across the super and sub class func. overloading is possible.

A f(int)
↓
f(void)
B.

B b = new B();
B object.

b.f() — [f() of B]

int i;

b.f(i) [f(int) of A].

A g(void)
↓
B g(void)

→ A super class reference can refer to both, super class & instance or sub class instance.

A a → super class ref. a.g() → g() of A
a = new B → sub class ref a.g() → g() of B

* Definition of overridden fn. is chosen based on the instance type it refers.

A → f(int, float)

↓

B →

```
class B extends A
{
    void f(int x, float y)   ~ override
    void f(int x, int y)     ~ overload
}
```

Good practice

```
@override
void f(int x, int y)   X //should be overridded
```

→ Tells the compiles that this fn is overriding a superclass fn.

→ Sub class constructor can initiate only the additional attributes declared in sub class.

```
B()                          B(int a, int b)
{ super();                   { super(a);
}                            }
```

→ Call to super() must be the first statement of sub class constructor.

→ If no call is made, default super class const. is called.

A int n; → not private
   int y; → ''

B int n.

f(...)
{
  n = ... ; refers to n of B.
  super.n = ... ; refers to n of A
  y = ... ; refers to y of A.

Similarly, super.method() to refer to method of super class.

works like const

→ final float pi = 3.14;
  cannot be changed

→ A method also can be declared as final
  cannot be overridden.

→ A class can also be delared final
  → no further extension of the class is allowed.
  → no more inheritance.

final class A
{
}

class B extends A
{
}

✗ Not possible.

Abstract Class

```
abstract class A
{
  .
  .
  .
}
```

```
A a = new A()
```

✗ Direct instance of abstract class is not allowed.

→ An abstract class can have one or more abstract methods. (may not have)
Abstract method → method with no implementation.

```
abstract void f(void);
```

→ If a class is having one or more abstract method, the class must be delared abstract.

→ Static method and constructor cannot be abstract.

→ A class that extends an abstract class must implement all the abstract methods of super class or else the sub class should also be declared abstract.

→ An abstract class refrence can be delared.

A → abstract
↓
B

```
A aref;
aref = new B();
```

Suppose System.out.println(String) ← Basic data type auto converted to string

Now, A a = new A();
we want System.out.println(a);
↓
It will look for toString() in that class.

```
class A
{
public string toString(void)
    {
        return (roll + name);
    }.
```

A → A(int)

```
A a1 = new A(5);
A a2 = new A(5);
if (a1 == a2)
{ sys--- ("yes")
```

Reference checking / Not content checking.

Nested Class

→ Class defined inside another class is called nested class.

Nested Class

Static Nested Class | Non-static Nested Class or inner class

```
Class A { Enclosing Class (Cannot be static)
{
    :
    class B
    { :
      :
    }
  static class C
    { :
      :
    }
}
```

class B → nested class / inner class

static class C → static nested class

→ can be declared public/private etc.

Non-static Nested Class – can access all members including private members.

static - nested class – can access only static members.
→ only default access.

→ If private not visible outside enclosing class
If public then visible.

Why Nested class ?

→ for logical grouping of the classes.

→ To implement aggregation.
special kind of association between classes.

→ Better encapsulation.
→ Code becomes more readable and maintainable.

* Inner class cannot have any static member.
* Outer class can create the instance of inner class. Can work with the public members of inner class.

```
class Item
{
    String icode;
    String iname;

private class CostInfo {
        float basePrice;
        float excise;
        float vat;

    CostInfo(float b, float..)..
    {
        .
        .
    }

Public float getPrice(..).
    {
{ .. }
```

```
class OuterClass
{
    class StaticNestedClass
    {
    }
}
```

```
OuterClass.StaticNestedClass staticClassInstance
        = new OuterClass.StaticNestedClass(...);
```

not related with any instance of outer class.

But the inner class needs instance of outer class.

Array

→ In JAVA array is actually an object.

```
int n[] = new int[5];
```

n ~~refers~~ refers to an array of 5 integers.

```
int []n = new int[5];

int []n = {5,4,7,8};

int n[][] = new int[5][2];

int n[5][2] = {1,2,3,5,9,4,6,8,7,0};
                )

int n[][] = new int[5][];
```

Different row can have different no. of columns.

```
n[0] = new int[7];
n[1] = new int[5];
```

→ Ragged array: array with variable column size.

n[i].length to get length.

~~boolean~~ ~~string1.equals(string2);~~

~~true if content same.~~

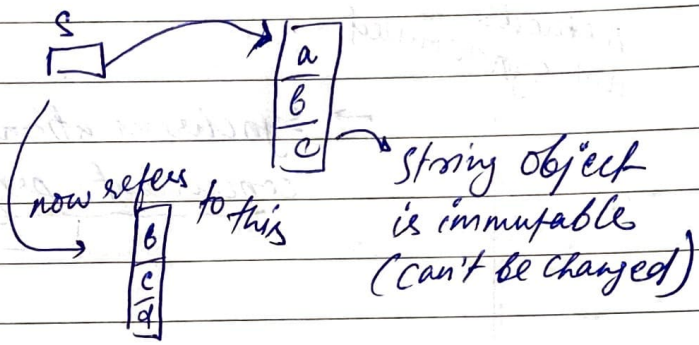
Strings

→ handled as object and not null terminated.

String s = "abc";

:

s = "bcd";



length() is a method here.

String s1 = "abc";

s1.charAt(0); returns a

String s1.trim(); removes leading and trailing spaces.

boolean s1.startsWith("sa"); ← substring.

boolean strin1.equals(string2);
true if same content.

s1.split(delimiter);

Mutable String :

- StringBuilder
- StringBuffer → methods ~~not~~ synchronized.

methods not synchronized

→ synchronization is important for concurrent programing.
↳ multiple process/threads try to work with same data simul-

StringBuilder

StringBuilder s = new StringBuilder();
↳ empty string of length 16.

new StringBuilder(string)
↳ deep copy
↳ string length + 16

StringBuilder s = "abc"; ✗

new StringBuilder(length)
↳ empty string of specified length.

→ conventional string methods are available.
trim(), indexOf() .. etc.

StringBuilder s_1, s_2 ;

s_1.append(s_2) ;
At the end of object referenced by s_1 content of s_2 will be added.

s_1.insert(posn, s_2) ;
In s_1, at posn, content of s_2 is inserted.

s_1.replace(start, end, String);

s_1.delete(start, end);
s_1.deleteCharAt(posn);
s_1.setCharAt(posn, char);
s_1.reverse();

Tokenizing a String
→ breaking the string into components which are called tokens.
→ delimiter to identify the tokens.

import java.util.StringTokenizer;

StringTokenizer st = new StringTokenizer(String) ;

String split() "[.|]" either . or | as delimiter

→ Cannot use multichar as a whole delimitter

→ boolean

```
while (st.hasMoreTokens())
{
    System.out.println(st.nextToken());
}.
```

st.countTokens()
no. of tokens left in the tokenizer.

PACKAGE

→ A package, consists of/container of classes and ~~str~~ sub packages.

→ logical grouping of related classes.

→ to partition the namespace
→ to avoid namespace collision.
→ better access control mechanism.

Source file:

→ In a source file of a package, ~~first~~ first statement will be package statement.
→ package mypack;
→ can have only one such statement.
→ If required can take help of other packages using 'import'. can have multiple import statements.

→ A package may be defined across multiple source files.

ACCESS CONTROL

	Private	(Package Public) Default	Protected	Public
Same Class	✓	✓	✓	✓
Same Package Sub Class	×	✓	✓	✓
Same Package Non-sub class	×	✓	✓	✓
Diff. package sub class	×	×	✓	✓
Diff. Package Non sub-class	×	×	×	✓

Outer Class

- **Public** — Accessible in all packages.
- **Default** — Accessible only in same package

→ Automatically java.lang package is imported.

INTERFACE:

→ Purely abstract class.
→ All the methods are abstract, no implementation is there.
→ Can have attributes
final and static.
→ Access specifies to an interface ⇒ public/default.
→ Interface can be extended.

```
interface I2 extends I1
{
    .
    .
}
```

→ A class can implement one or more interfaces.

```
Class A implements I1
{
    .
    .
}
```

Must implement all the methods of I_1.

Otherwise

```
abstract class A Incomplete implements I1
{
    .
    .
}
```

→ A class can implement multiple interfaces.

Class A implements I_1, I_2, I_3
{
:
}

→ No multiple inheritance but interface allows that.

→ We cannot have an instance of interface.

I iref;
↳ reference to an interface.

I { f1(); f2(); }
Class A implements I.
{
: f1(); f3();
} f2(); f4();

iref = new I(); ✗
iref = new A(); ✓

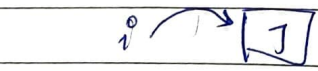
iref.f1(); ✓ iref.f3(); ✗
iref.f2(); ✓ iref.f4(); ✗

WRAPPER CLASS:

→ corresponding to each basic type, there is wrapper class.

int → Integer
char → Character
float → Float
:

Boxing { Integer i; → Reference to integer object.
i = new Integer(7);



Unboxing { int i;
i = i; automatic conversion.

Character

Boolean isLetter(char);
isDigit(char);
isLetterOrDigit(char);
isLowercase(char);
isUppercase(char);

Object to Basic type

int i = I.intValue();
typValue();

from string to Basic type

int i = Integer.parseInt(string);
Float.passeFloat(string);

Basic type to string

Integer.toString(basic data typ); (Int)

Float.toString(float);

from string to wrapper Object

Integer.valueOf(String);

EXCEPTION HANDLING

Object
|
Throwable
→ Error
  ↓ Problem related to runtime environment
→ Exception

Exceptions maybe checked exceptions or unchecked exceptions.

- checked exceptions ↓ Checked at the time of compilation. Unless measures are taken compilation will not take place
- unchecked exceptions ↓ occurs during runtime. compiler cannot check.

```
try {

}
catch (...)
{

}
finally {   //optional.

}
```

Statements

Exception in try block → No → catch blocks are skipped, if finally block exists, it is executed then statements execute.

↓ Yes

is there catch/finally block?
- No ↓ abrupt termination
- yes → suitable catch?
  - No → If finally present, execute. Termination.
  - yes → Execute catch → Execute finally → Execute statements

→ Catch block with specific type of exception object are to be placed before one with general type

Custom Exception :

```
class ~~CustomException~~ extends Exception
{
    class Student
    {
        int score;

        getScore(int k) {
        try{
            if(k<0 || k>100)
            { throw new InvalidScoreException("score
              range");
            }
        }
```

```
Class InvalidScoreException extends Exception
{ String msg;
  int value;

InvalidScoreException(String m, int v){
        msg = m;
        value = v;
}
```

```
public String toString(){
    return msg + "Score:" + value;
}

catch(InvalidScoreException e){
    System.out.println(e);
}
```

Collection Framework

→ Collection is a unit for a group of elements.

→ Collection framework provides number of interfaces and classes implementing the interfaces for storing and manipulating a of collection of objects

(Interface) collection

(Interface) List

(classes) implementing the Interfaces: ArrayList, vector, stack, linked list

Collection Interface

clear() ⇒ clears all elements
capacity() ⇒ ~~How~~ no. of element that can be stored without new memory allocation
size() ⇒ no. of elements.
add(element)
compare(element)

→ The collection framework also provides algorithms.

collections.binarySearch(list, value, comparator);
used for ordering

binarySearch(list, value)
already sorted.

ARRAYLIST :

→ Dynamic array.
→ depending on requirement size can grow and shrink.
→ Elements can be added or removed.

ArrayList a = new ArrayList();
default - allocation 10 elements.

→ Elements maybe of different types.

a.add("abc");
a.add(10);

get(index)
to get value

String s;
s = a.get(0); x
s = (string) a.get(0)
implicitly typecasting.

→ If arraylist type is not specified, retrieving a value and putting it in a variable requires casting.

class Student {

}

ArrayList <Student> slist = new ArrayList();

ArrayList <int> ✗
ArrayList <Integer> ✓

Iterable ⇒ Iterable is an object over which we can iterate.

Iterator <T> iterator();
↓ returns an iterator for a collection with element type T.

List ⟶ Iterable Object, not an iterator.

⇒ Access all the elements in an Arraylist.

```
Arraylist <student> slist = new - - - ;

for(int i=0 ; i < s.list.size; i++)
{
    sysout (slist.get(i));
}
```

```
Iterator < student > it = new Iterator(); slist.iterator();
    student s;
while ( it.hasNext())
{   s = it.next();

}
```

Access all elements in forward direction.

```
ListIterator<Student> lit = new slist.listIterator();
                          = new slist.listIterator(slist.size());
```

position default=0

```
boolean equals(Object o)
{
    if(o == null) return false;

    if(o instanceof Student)
    {  if(o.roll ==
       // other comparisons
    }
}
```

```
ArrayList<Integer> ilist;
```

→ Collections.sort(ilist); Ascending order

→ collections.reverse(ilist); After above method for descending order.

```
class Student implements Comparable<Student>
{  int roll;
   other attributes.
int compareTo(Student s)
{  return (roll - s.roll);
}
```

Now, we can call

```
Collections.sort(slist); // Ascending Order
```

```
class StudentScoreCompare(student s1, student s2)
{
    public int compare(Student s1, Student s2)
    {
        return(s1.score - s2.score);
    }
}

Collections.sort(slist, new StudenScoreCompare());
        // Ascending Order

@override
int hashCode()
{
    return(0);
}
```

LinkedList

```
LinkedList();
LinkedList(int capacity);
LinkedList(collection);

add(element)
```

element() => returns the first element.

- get(index);
  getFirst();
  getLast();

peek(); → Return 1st Element.
peekFirst();
peekLast();

poll(); → returns and removes 1st element.
pollFirst();
pollLast();

Stack

empty();
push(element);
pop() → removes top element
peek() → gives top element.

PriorityQueue<String> nameList = new PriorityQueue<String>();
Element type

add(__) ; to add elements.

→ Elements stored according to priority. (Ascending Order)

PriorityQueue<Student>

class Student implements Comparable<Student>
{

```
:
:
int compareTo(Student s)
{
    return (score - s.score);
}
}
```

peek() ⇒ gives top priority element.

ArrayList ⟷ Vector

Methods are not synchronized — Methods are synchronized.

Map Interface

↓ collection of key (unique) value pairs.

HashMap Class / HashTable (synchronized)

↓

cannot be iterated directly.

Hash Map

HashMap();

HashMap(capacity);

HashMap<key type, Value Type> hm = new hashMap<—,—>();

hm.put(key, value);

isEmpty();
clear();

containsKey(key);
containsValue(value);

value ← hm.get(key)

System.out.println(HashMap)

[key1 = value1, key2 = value2, ...]

Set<Entry<keytype, valuetype>> s

s = hm.entrySet();
Returns set of entries

for(Map.Entry<key type, value type> m : s)
{
m.getKey();
m.getValue();

FILE HANDLING :

Text file
Binary file } store data permanently.

→ Read
→ write.

import java.io.*;

file handling
↳ throws IOException.

Text file

```
PrintWriter pw = new PrintWriter(filename);
pw.println(data);
pw.close();
```

old data will be lost

Appending

FileWriter fw = new FileWriter(filename, Boolean)
true old data remains
false cleared.

PrintWriter pw = new PrintWriter(fw);

Reading from text file :-

```
File f = new File(filename);
Scanner s = new Scanner(f);
while(s.hasNext())
{
    a = s.nextInt()
    :
}
s.close();
```

f.exists(); → checks if file exist or not.

f.isDirectory(); → if it is a folder.
f.isFile(); → if a file.

Binary File

```
FileOutputStream f = new FileOutputStream(filename);
DataOutputStream d = new DataOutputStream(f);
d.writeInt();
d.writeFloat();
:
f.close();
```

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

Random Access file

```
RandomAccessFile ra = new RandomAccessFile (filename, "r");
```

file opening mode string.

"r" → reading
"w" → writing
"rw" → file will be created if not exists, or appended.

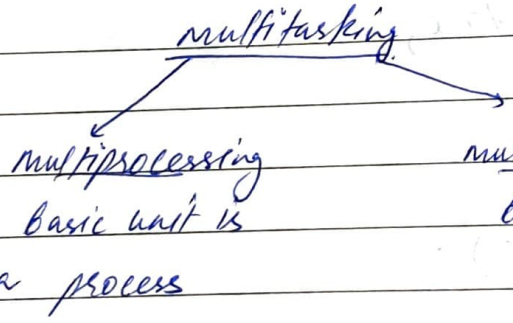
ra.seek(byte posn);

Storing Objects :

class Student implements Serializable
{
.
.
}
.

Serializable has no methods inside
→ to inform the compiler in order to store instances in file,
→ object has to be converted into string of bytes.

THREAD :



→ process has own address space
→ Thread in a process share the same address space.
→ Interprocess communication is difficult w.r.t inter thread communication.
→ context switching for process is costlier.

Thread

needs a code and data on which the code will run.

class Thread.

Thread (runnable instance)

1) By implementing runnable interface

2) By extending Thread class

```
MyRunnable
class MyThread implements Runnable
{
    : attributes.
}
```

```
MyRunnable r = new MyRunnable ( );
                ↳ runnable instance.

Thread t = new Thread (r);

t.start();
```

```
class MyThread extends Thread
{
    : attributes

  public void run ( )
      {
        :
      }
}
```

MyThread t = new MyThread();

```
Class Account
{
  float balance;

  public void update(float n)
  { ~~balance~~
      balance += n;
  }
  Account(float a){ balance = a; }
}

AcctRunnable implements Runnable.
{
  ~~float n;~~ Account a;

  AcctRunnable(Account n){ a = n; }

  public void run()
  { float ~~acct~~ amount;

      a.update(amount);
  }
}.
```

```
public void run()
{ float amount;
  synchronize(a)
  { a.update(amount)
  }
}.
```

Synchronized statement

```
Account ac = new Account(1000);
AcctRunnable r1 = new AcctRunnable(ac);
   --  --  r2 =  --  --  --  (ac)
Thread t1 = new Thread(r1);
 --  --  t2 =  --  --  (r2);

t1.start();
t2.start();
```

Concurrent Access

→ Multiple threads try to access the same data simultaneously. If they try to modify the data.

→ Concurrent access must be serialized (one after another) to ensure data consistency.

→ Use the key word 'synchronized'.

```
synchronized public void update(float n)
{
    balance += n;
}
```

→ Every Object instance in Java has a monitor/lock associated.
→ For the class also there is a lock.
→ Lock does not play any role till the instance calls a synchronized method.

An instance calls a synchronized method –
→ first of all the lock on the object has to be gained

gets it → proceeds with the method
→ when execution is over of synchronized method, gives up the lock.

does not get it → waits till lock is given up by others
→ it gets the lock then

→ Synchronized statement → in previous page

Interthread Communication:

wait()
notify()
notifyAll()
} inherited from the Object class and cannot be overriden.

```
class Data {
    int n;                          boolean chk=false;   → true when something is written
synchronized void set(int i)
    { n=i;                          if(chk){ try{ wait(); }
                                    catch(..){ }
    }                               chk=true;
                                    notifyAll();
synchronized int get()
    { return n;                     if(!chk){ try{ wait(); }
    }                               catch( ){ }
                                    chk=false;
                                    notifyAll();
}
```

```
Class Reader implements Runnable.
{ Data d;
Reader (Data a) { d = a;}.

public void run ()

while(true)
{
    System.out.println(d.get)
  }
}

Class Writer implements Runnable
{.
    Data d;
Writer (Data a) { d=a; }
public void run()
{     int i=1
    while (true)
  {  a.set(i)
      i++;
   }.
  }

Class ---
{
    p s v main()
  {    Data d;
  Reader rd = new Reader(d);
  Writer wt = new Writer(d);
Thread rthread = new Thread (rd);
  --- wthread = ---  (wt);
  wthread.start();
  rthread.start();
```

```
try {
        wthread.join();
        rthread.join();
    }
    catch(- - -)
    {

    } - - -
```

→ If we call wait() and notify() and notifyAll() outside of any synchronized method or synchronized block, it will give rise to IllegalMonitorException

* Reading is allowed if no other thread is writing or has made a request to write.
* Write is allowed if no read is there.

```
class ReadWriteLockPolicy
{
    int rcount = 0;
    int wcount = 0;
    int wrqcount = 0;

synchronized void readlock() throws InterruptedException
{   if (rcount > 0 || wcount > 0)
        wait();
```

GUI

JFC (Java Foundation Classes)
- Provides the framework for creating GUI applications.

java.awt
(abstract windowing toolkit)
javax.swing.

awt classes - takes the help of another layer of peer classes.
↓
corresponds to specific OS.

→ Appearance of GUI component may vary from OS to OS.
→ Difficult to extend awt classes.

swing classes
→ Lightweight
→ No peer class.

GUI Components/Elements :

Label → to display a fixed string.
TextBox → To take input from user.
Enter your Name [ ]

RadioButton → Normally, appear in group and only one can be chosen in a group.
Student level
○ UG
○ PG
◉ PhD

CheckBox → Multiple Options (No Grouping)

Skill Set

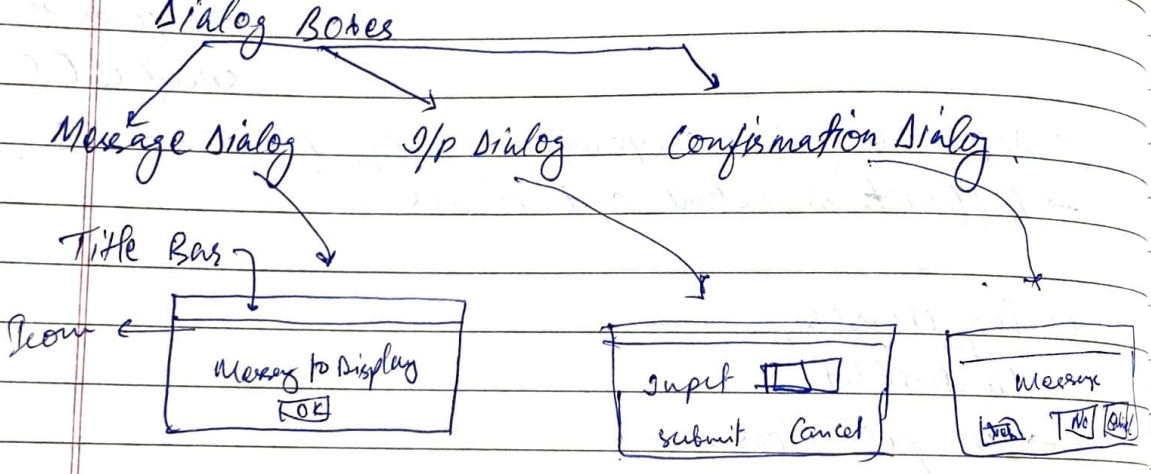
- [x] JAVA
- [ ] PYTHON
- [x] C
- [x] C++
- [ ] ML

ComboBox → Combination of List and Box.

List → A drop down list from where selection can be made

Button - An interface element, clicking on it gives rise to an event

Dialog Boxes



import javax.swing.*;
or JOptionPane

JOptionpane.showMessageDialog(component parent, Object message
null (center of screen)

JOptionPane.showInputDialog(. . .)

```
Int i;
String s;
s = JOptionpane.showInputDialog(null, "Enter score
if(s != NULL)
    i = Integer.parseInt(s);
```

for GUI

→ The last statment must be System.exit(0);

int v;

v = JOptionPane.showConfirmDialog(. . .)

JOptionPane.YES_NO_OPTION.
JOptionPane.YES_NO_CANCEL_OPTION.

YES → 0, NO → 1, CANCEL - 2.

CREATING WINDOW

↓ is a component that can hold other components.
So, it is a container.

→ Simple containes that can be displayed as window is a frame.

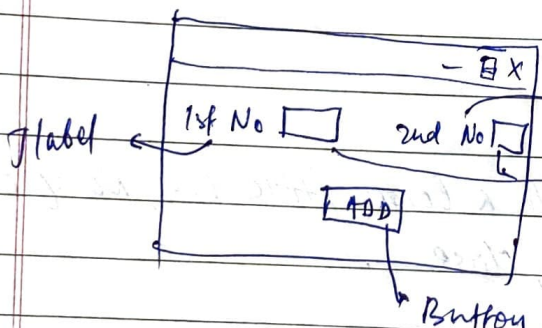
Swing

JFrame.

Displayable window with a border, title bar, No. of buttons for minimize, maximize, close.

```
impor javax.swing.*;

Class - - - .
{ public static void Main ( - - )
  {
     JFrame f = new JFrame ("Sample Frame");
     f.setSize(200,100);   // width, height.

     f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        f.setVisible(True);
    }
}

class MyWindow extends JFrame
{
    :
    :

    MyWindow(..)
    {
        MyWind("---");
        setSize(__)
        setDefault--();
        setVisible();
    }
}
```



Sum of -- and -- is ---

Event Handling:

→ An event is an action in the program, say clicking on the button.

→ Event has a source. The component that has given rise to the event.

↳ In this case the button.

→ Source generates an event object event.
↳ holds the description of the

→ Event source may be associated with one or more Listeners.
→ The event object is passed to the particular method of particular listener and gets executed.

Button → Source:
ActionEvent Object.
ActionListener interface.
↓
actionPerformed(ActionEvent).

JPanel
Not displayable.

```
class Addition extends JFrame.
{
    JLabel l1, l2;
    JTextBox t1, t2;
    JButton b1;
    JPanel p1;

    Addition()
{   super("Addition");
    makePanel(--);
    setSize();
    SetDefaultC--();
    add(p1);
    setVisible(true);
}
```

```
void makePanel()
{
    l1 = new JLabel("1st No");
    l2 = new JLabel("2nd No");
    t1 = new JTextField();
    t2 = new JTextField();
    p1 = new JPanel();
    b1 = new JButton("ADD")
    p1.add(l1);
    p1.add(t1);
    p1.add(l2);
    p1.add(t2);
    p1.add(b1);
    p1.addActionListener(new MyListeners());  → Listener Registration
}
```

→ we have to import java.awt.event;

```
private class MyListeners implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {   int v1, v2;
        String s1, s2;
        s1 = t1.getText(); s2 = t2.getText()
        v1 = Integer.parseInt(s1);
        v2 = ---(s2);
        JOptionPane.showMessageDialog(...)
    }
}
```

→ getSource() a reference of source command.

If there's a b_2 for subtraction, Checking and performing.

(e.getSource == b_1)

getActionCommand() // If b_1 is not accessible.

↓

String → text on the source button.

JTextField

setEditable(false); // user cannot enter data.

setText(--);

pack(); can be used ~~instead~~ instead of setSize();

Layout.

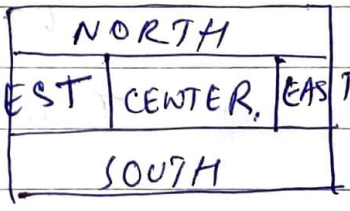
→ Layout Manager object governs the layout of the object.

→ Positioning and sizing of the components.

- BorderLayout → default for Frame
- FlowLayout → default for Panel.
- GridLayout — no of rows and columns.

→ components are added row wise, once it does not fit goes to next row.

→ Screen is divided into 5 parts.



BorderLayout.

TFrame f, TPanel p.

p.setLayout(new BorderLayout());

Instance of Layout Manager.

p.add(c1, BorderLayout.EAST/WEST/...);

f.setLayout(new FlowLayout());

JRadioButton

JRadioButton rb1, rb2, rb3;

rb1 = new JRadioButton("sub1");
rb2 = ---------;
rb3 = ---------;

ButtonGroup g = new ButtonGroup();

g.add(rb1);
g.add(rb2);
g.add(rb3);
→ logical group, mutually exclusive

rb1.isSelected();
rb2.isSelected();

JCheckBox

→ If already selected, again clicked, it will be deselected

JCheckBox cb1 = new JCheckBox("...");
cb2 = ---

add(cb1);

ItemEvent
ItemListener.

public void ItemStateChanged (ItemEvent) → e.

reference of checkbox source = e.getItemSelectable().

Borders:

→ Put a border around a component.
→ Border object specifies the style.

→ BorderFactory class can be used to create the borders objects.
  ↳ diff. methods for diff. style.

component.setBorder(BorderFactory.createTitledBorder(String));

JList

→ from a list users can select one or more.

JList list = new JList();
  or = new JList(array of Objects)
JPanel p = new JPanel();
p.add(list)

list.setSelectionMode. { SINGLE_SELECTION
  . SINGLE_INTERVAL_SELECTION
  . MULTIPE_INTERVAL_SELECTION.

getSelectedValue()
↳ if value selected, returned as object
we have to typecast.

getSelectedValues():
↳ array of objects.

getSelectedIndex()
→ -1 if none selected. or index of 1st item

Event Handling
ListSelectionListener
ListSelectionEvent.

public void valueChanged(ListSelectionEvent e)
{
:
}

list.setVisibleRowCount(n)

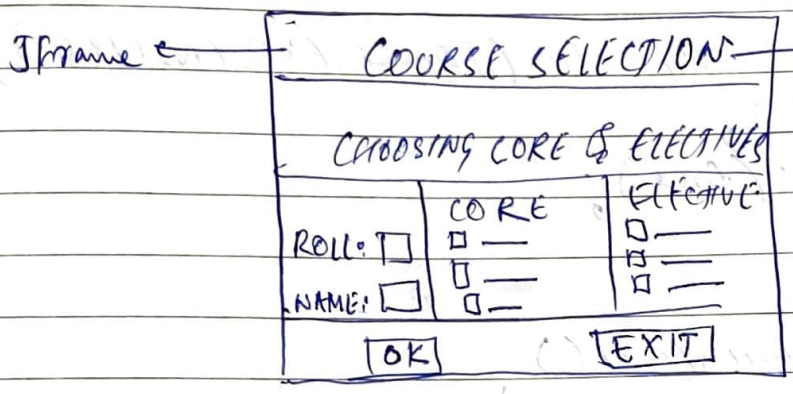
JScrollPane sp = new JScrollPane(list)
p.add(sp)

adding the scroll pane will put the list also

JComboBox
List + TextBox.

JComboBox cbox new JComboBox()
. new JComboBox(Array of Object
cbx.setEditable(true/false);

cbm.setItem();
cbm.setIndex();



class CourseSelection extends JFrame
{
    HeadingPanel hp;
    CandidatePanel cp;
    CoreCoursePanel cop;
    ElectivePanel ep;
    JPanel bp;

    CourseSelection()
    { super("CourseSelection");
      hp = new HeadingPanel();
      cp = new CandidatePanel();
      cop = new CoreCoursePanel();
      ep = new ElectivePanel();

      PrepareButtonPanel();

```
add(BorderLayout.NORTH, hp);
add(_____.WEST, cp);
add(_____.CENTER, cop);
add(_____.EAST, ep);
add(_____.South, bp);

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
pack();
setVisible(true);
}

Void PrepareButtonPanel()
{ b1 = new JButton("OK")
  b2 = new JButton("EXIT")

private class MyListener extends ActionListener
  {
    public void actionPerformed(ActionEvent e)
    { if(e.getSource() == b2)
        System.exit(0);
      else {
            .
            .
            }
    }
b1.addActionListener(new MyListener());
b2.  -  "  -  "

bp = new new JPanel();
bp.add(b1);
bp.add(b2);
```

APPLET

→ It is a program associated with web page and executed by the web browser as a part of web page.
  ↳ special version of JVM that executes the applet.

→ User makes a request to access a web page, web server transfers the page alongwith the associated applet code to the user's system. Browser at the user's system executes the code.

Application Vs Applet

→ Application is a standalone program, it has main() that gets executed.
→ Applet does not have main(). Once it is loaded certain other methods are there to initiate execution
→ Applet cannot be executed independently. It needs a browser

Restrictions of Applet

→ Cannot read/write/delete with local file system.
→ Cannot run any O.S. procedure on local system.
→ Cannot access user information or local system information.
→ Can communicate only with the intended web server.

→ To protect user's system against malicious code causing damage/spying.

Why Applet?

→ To create the programs which may be used by other users across the network.
→ To make a web page dynamic.

How :

→ Create the applet source file (.java)
→ Compile it to obtain (.class) file.
→ Include this ~~file~~ (.class) file in HTML file.
→ Open the .HTML file using browser.

```
import javax.swing.*;
import java.awt.event;

public class Sample extends JApplet
 {
    JLabel l1;
```

GUI app → extends JFrame → constructor
Applet → extends JApplet → won't write constructor

→ calls constructor to create instance.
→ instance is created by the browser when code is loaded

```
public void init()
 {
    l1 = new JLabel("My Applet");
    add(l1);
 }
}
```

XY2.HTML

```
<APPLET CODE = "Sample.class" width = — Height = —>
</APPLET>
```

→ GUI Applet is an application controlled by the Browser. It does not have its own window. uses the window provided by the Browser.

setTitle ()
setSize ()
setDefaultCloseOperation ().
pack ()
setVisible ()

} NOT ~~allowed~~ allowed on Applet.

Life Cycle