HERO NOTES
MONAWAR SONS LAHORE

~~Name & Address~~	~~Office~~	~~Fax~~	~~Residence~~	~~Mobile~~

"Course Outline"

NON-OOP

↳ (remaining indexing at last)

HERO NOTES
MONAWAR SONS LAHORE

Name & Address	Office	Fax	Residence	Mobile

Bye No.

"Course Outline"

This Diary belongs to

"Jamshaid Naseem"

Contact me at:

0309-0821168

"رَبِّ يَسِّرْ وَلَا تُعَسِّرْ وَتَمِّمْ بِالْخَيْرِ"

Computer:

Computer is an integrated set of algorithm and structure.

Algorithm:

step-by-step solution of problem known as Algorithm.

Algorithm Type:

(i) Flow Chart (visual representation)

(ii) Psudo Code (simple human language)

(iii) Source Code (high-level language)

Software:

The integible set of Instructions.

Syntax :

The grammetical agguement of words known as syntax.

Grammes :

Set of pre-defined rules for a languge known as grammes.

Bit : (Binary Integes)

Bit is a digital signal which can be "Off or On".

Programming Paradigms

Sequential Programming:

The programming in which code is written in a sequence is called sequential Programming.

Functional Programming:

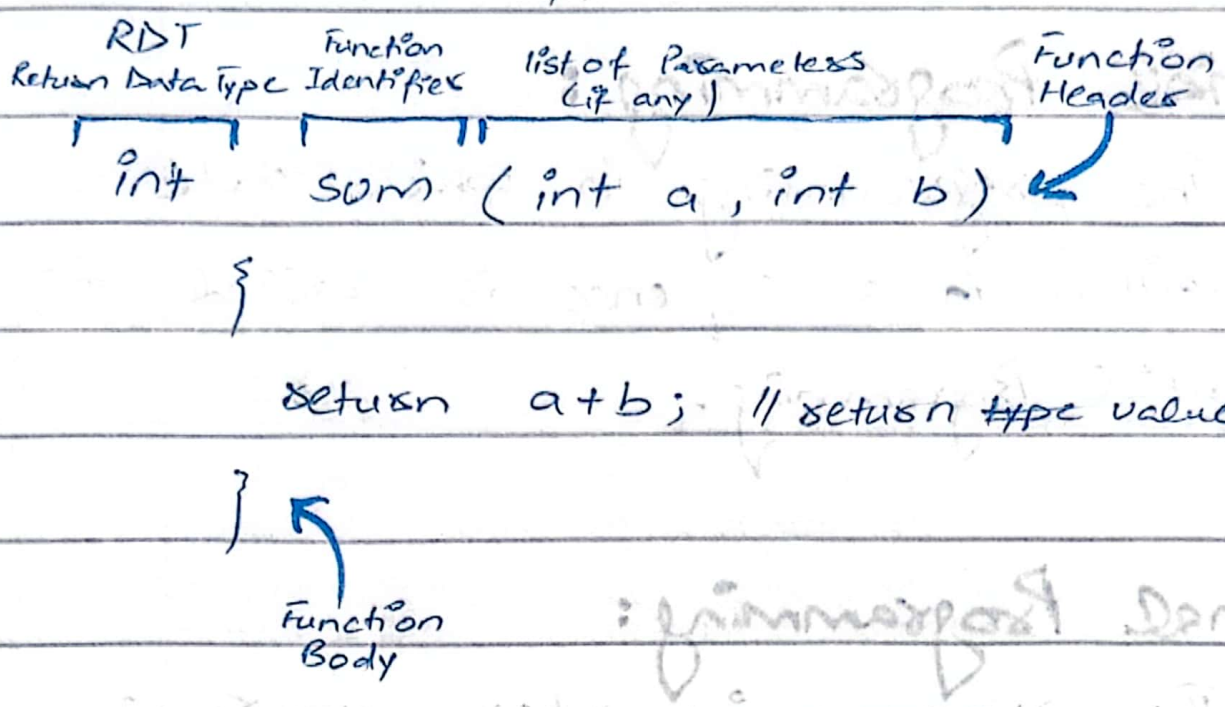
The programming where programs are constructed by applying and composing.

(i) Function Defination

(ii) Function Call

(iii) Function Prototyping

(i) "Function Defination":



Parameters are the input of any Function. (if any)

Return Value are the output of any Function. (if any)

(ii) "Functional Call":

To call function.

```
sum ( 5, 10);
```

Function Identifier — list of Parameters (if any)

Note : :

Parameters in defination of function is called **formal parameters.**

Parameters in Function calling is called **Actual Parameters.**

(iii) **"Function Prototyping" :**

In some programming languages, we cannot call function above the function defination. To resolve that mess, we use the concept of prototyping in stast of programming.

```
int sum (int a, int b);
```

"Function Defination's Function Header"

Object-Oriented Programming:

Small talk was a language which introduced the concept of classes and objects.

The language failed as whole but this concept became famous with time.

In OOP real world objects are converted (mimiced) into digital objects.

Class:

Class has three school of thoughts.

(i) First School of thought

Classifying Programming Information into organized and systematic Manner.

(ii) Second School of thought

Class is the written description of an Object.

OR

Class is template of Object.

(Object: Physical Occurance of a Class)

(iii) Third School of thought

A Class is an Abstract Data Type. (Abstract : Imagination)

Being an Abstract Data Type, three things are important.

(i) Declaration of Data.

(ii) Declaration of Operation.

(iii) Encapsulation of data and operation.

Entity is a person, place or object or any concept for which an organization wants to record or process data.

Attributes are the characteristics of entity for which an organization wants to record.

- Class Represent "Entity Set".
- Entity are Data Members of class.
- Attributes are Operations (Member Function) of class.

```
Class Student {
    int roll no;
    String name;
    void input() { ≡ }
    void output() { ≡ }
}
```

Data Members

Member Function

Encapsulation

(i) Data Hiding

(ii) Data Abstraction

Premitive Data Type

The data type which is recognize by compiler is called premitive data type.

i.e int, float, long, double etc.

Non-premitive data type:

User-defined data type is called non-premitive data type.

Structure is not a abstract data type.

class or methods types: "MAM Types"

Members Access Modifiers

Private:

In private, data members are restricted to same class only.

Default:

Available in same package and accessible to all classes in that pkg.

Protected :

Available within the Package and accessible outside the package too if we make a child class of same class.

Public :

Accessible everywhere.

Note :

→ String is a premitive data type for java, As String is recognized by java.

→ MAM "Member Access Modifier" &
MAS "Member Access Specifier"
specifies the access of class's member.

Developers

There are two types of developers.

(1) Application Developer

(2) Library Developer

Encapsulation

Encapsulation is about two things.

(1) Data Hiding

(2) Data Abstraction.

If there is any Abstraction on data then we will hide it and provide a limited access to the users or another developers.

Package

→ Collection of classes.

→ package name should be unique.

→ package name cannot be repeated.

Comments

→ single line comment //

→ multiple lines comment

/* - - - - - - - - - - - -

- - - - - - - */

Token

Every characters use in coding is a token.

```
int sum() {  → 'token'
'token'
      return 5 + 6;   → 'token'
}
      'token'   'token'
```

"Data Types in Java"

int integeral data types
use to display whole numbers
5, -32, -9 etc. It is 4 byte. contains

bool boolean data type
use to display true or false.
takes only one bit.

char characters data type
use to display a single
characters either it can be
alphabetic, numeric or special characters.

string combination of characters, depends
on the characters it contained;
"PAKISTAN", "OFFICE2" etc.

void void is also a data type.
null or undefined data type.
it returns nothing.

float float data type contains
decimal point data.
float contains 4 bytes.

double double data type contains
decimal point data but it is
a bigger container than float.
It contains 8 bytes.

long long data type contains integral
type of data but it is a
bigger container than int.
It contains 8 bytes.

Type Casting

One data type converts to another data type is known as type casting. There are two types of casting use in java.

(1) Implicit Casting

(2) Explicit Casting

Implicit Casting:

The type of casting done by compiler itself is known as implicit casting.

→ It can convert small values to larger values.

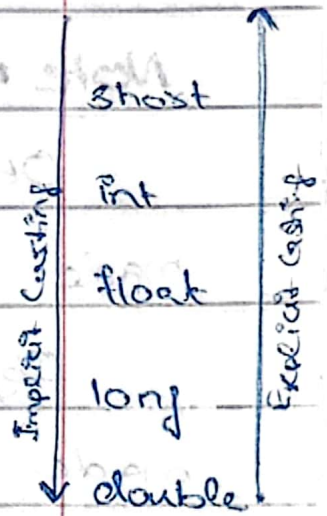
→ Not convert larger values to smaller values.

Explicit Casting:

Type of casting done by user forcefully called explicit casting.

→ Explicit casting can ~~(lose)~~ cause lost of data.

→ Data is not lost in implicit casting because it is done by compiler.



Compiler always operates same data types.

Integral sum:

```
int sum(int a, int b)
```

" data types must be compatible. to type cast."

Objects :

Objects are the physical occurance of classes.

Note :

Objects of smaller data types made in the stack memory.

Objects of larger data types made in heap memory and their reference variable is present in stack memory to access the object.

Objects and Reference Variable

In java all the primative data types other than **string** create their objects in stack memory. This declaration is called "static declaration."

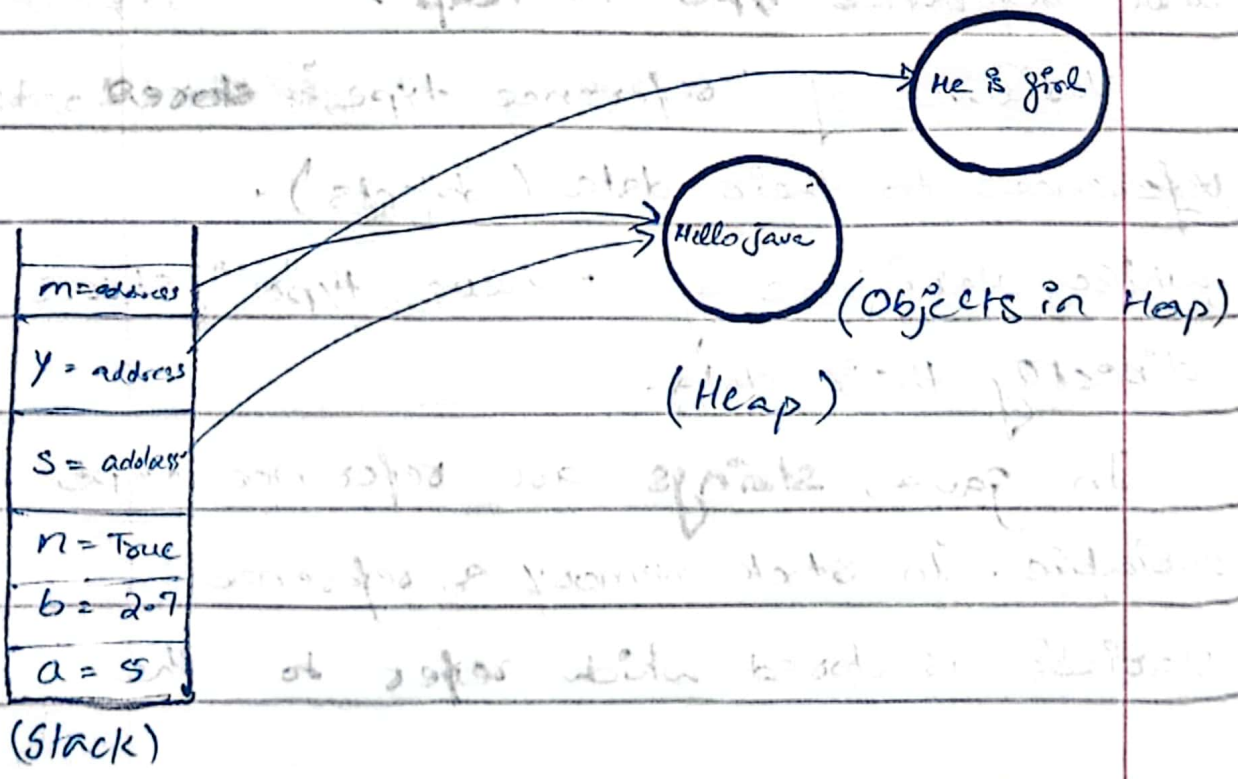
Java doesnot provide you choice to save value type ~~or~~ and reference type value to stack or heap. It restricts you to save **value type** to **stack** and **reference type** to **heap**.

Variables of "reference type" stores references to their data (objects). while variables of "value type" stores directly their data.

In java, **strings** are reference type variable. In stack memory a reference variable is stored which refer to the

data (object) present in heap.

```
int a = 5;
float b = 2.7;
bool n = True;
String s;
s = new String("Hello Java");
String y = new String("He is girl");
String m = new String("Hello Java");
```



Summary :

while working with classes, we declare a "reference variable" of that "class_type" and then, we use the operator **new** to initiate an object of that "class_type" and store the address of the object into the "reference variable".

```
String   s  =  new  String("Hello Java");
```

String → Class
s → reference variable
= → "new" operator
new String("Hello Java") → objects

```
Scanner  sc  =  new  Scanner(System.in);
```

```
sc.nextInt();
```

→ the reference variable to object created in heap can access all the methods of Scanner class.

Garbage Collector :

When we overwrite the code and refer the variable to another object and previous object is in heap without being refferred to any variable.

The "Garbage Collector" deletes the object and reclaims the storage for later use.

You can also run garbage collector on your own. by using the Syntax :

```
System.gc();
```

General Knowledge to Use Pre-defined Classes and Methods

Java comes with a wealth of classes which are called **predefined classes**. In addition, every predefined "class" contains many predefined methods.

Classes are organized as a collection of packages called **class libraries**.

A particular package can contain several classes and each class can contain several methods.

There are two types of methods

(1) Static Method

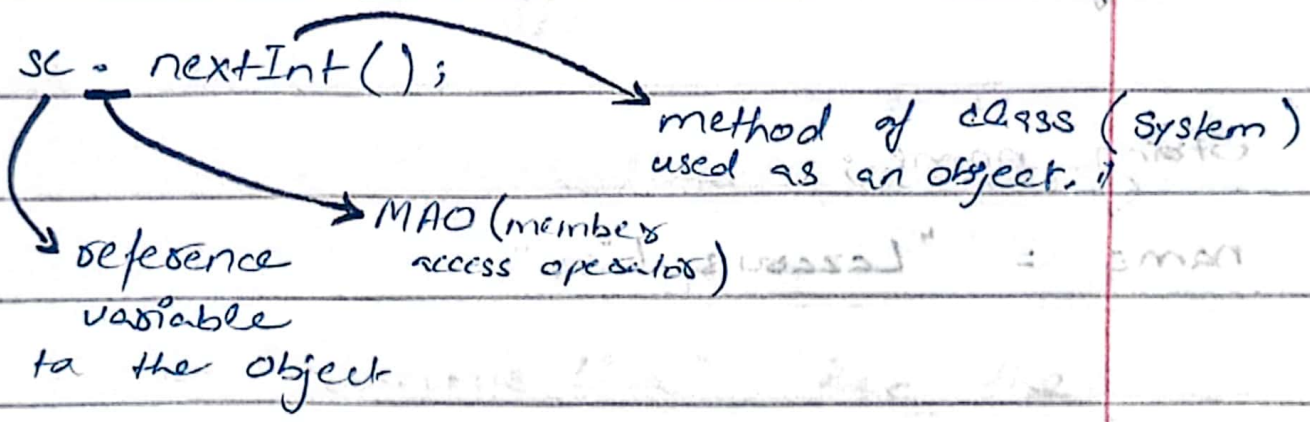
(2) Non-Static Method

To use a predefined method in a program:

- you need to know the name of the class containing method.
- you need to know the name of the package containing that class.
- Import this class from the package in the program.
- You also need to know about the parameters the method takes, the type of each parameter. and the order of the parameters.
- You must also know about the return type of the method or loosely speaking, what the method produces.

In Java, the dot (.) is an operator called the **member access operator.**

```
Scanner sc = new Scanner(System.in);
sc.nextInt();
```



Class String

This section will explain how to use **String** methods to manipulate strings.

```
String name;
name = "Lazarus. Lisa"

// String Object with the value
"Lazarus Lisa" is initiated and address
of this object is stored in "name".
```

The Class **String** provide various methods that allow us to process strings in various ways.

The **Index** (position) of the first characters is 0, the second is 1 and so on.

The general expression to use a **String** method on a **String** variable is :

StringVariable . StringMethodName (parameters) ;
(if any)

As name = "Lazarus Lisa", then the value of expression will be:

name . length()	Returns length of the string "name"
12	
name . charAt(6)	Returns the character at position 6
s	

name.indexOf('z') 2	Returns the index value of char 'z' in name.
name.substring(8, 12) Lisa	Returns the substring starts from 8 and ends at (12-1).
name.replace('L', 'F') Fazarus Fisa	Returns the string by replacing 'L' with 'F'.
name.toUpperCase() LAZARUS LISA	Returns the string after changing all the lower characters to upper characters.

Input / Output

A program perform three basic operations:

- it gets data into a program
- it manupulates the data
- it outputs the result.

Output:

To print anything on the output screen. we use member of class "System".

```
System.out.println("Hello World");
System.out.print("Hello world");
System.out.printf("Hello %s", "world");
```

Input

```
Scanner sc = new Scanner(System.in);
sc.nextLine();
```

Passing Numeric Strings

A string consisting of only an integer or a floating-point number, optionally preceded by a minus sign, is called a **numeric string**.

"6723"

"-782.59"

To convert a string consisting of an integer or floating-point to value of a type int or float respectively,

- Integer.parseInt("623") = 623
- Float.parseFloat("6.2") = 6.2
- Double.parseDouble("345.78") = 345.78

To perform any operation;

Integer.parseInt("34") + Integer.parseInt("45")

= 34 + 45 => 79

Using Dialog Box for Input / Output

Str = JOptionPane.showInputDialog("Enter your name and press OK")

→ To open a panel with written prompt to enter a name.

let's turn to "showMessageDialog" output.

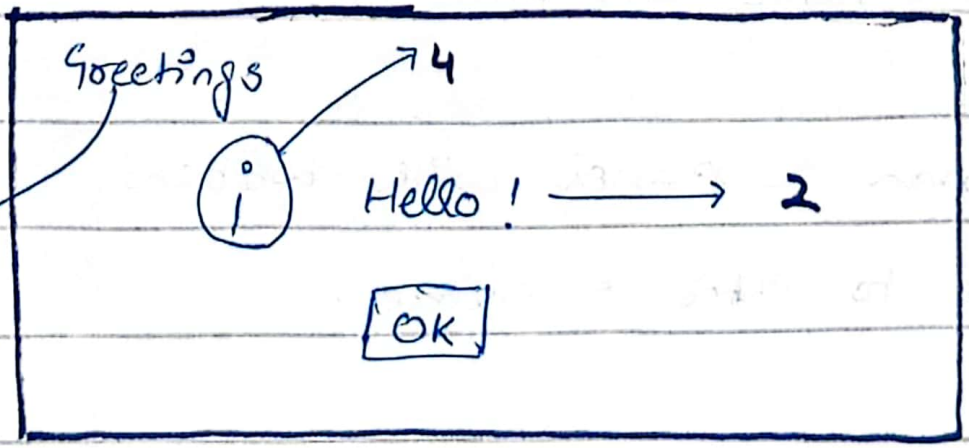
JOptionPane.showMessageDialog(parentComponent, messageStringExpression, boxTitleString, messageType);

Example

~~JOptionPane.showMessageDialog(parentComponent,~~

JOptionPane.showMessageDialog(null, "Hello!", "Greetings", JOptionPane.INFORMATION_MESSAGE);

(1) (2) (3) (4)



Constructors:

A constructor is a special member function with the identified same as class name.

- constructor have no return type.
- constructor doesnot return anything explicitly.
- A constructor automatically get created in class if not get created manually.

```
Class myCar {

public myCar() {          <= "constructor"

    }

}
```

Constructor Overloading

Constructor Overloading in Java refers to the use of more than one constructor in an instance class.

However, each "overloaded constructor" must have different **Signatures**.

Signature are different if

- number of parameter changed.
- position of parameter changed.
- Type of parameter changed.

```
public class ImySpace{
    int a;
    int b;
*   public ImySpace(int num1, int num2){
        a = num1;
        b = num2;
    }
```

* public Myspace(int num1){
    a = num1;
    b = a;
}

In this example, two constructors are created in class both have different signatures by having changed number of parameters. ~~and changed position~~

It is also an example of ad hoc polymorphism. (page # 111)

Types of Constructors on Page 38

"Static & Non-Static"

Static:

- A "static method" is a class method and belongs to the class itself.
- If a constant is static, there will be only one copy of that class. (i.e. one copy per class).
- Static variables are created (~~with the use of~~) when the program starts and destroyed when the program stops.
- Static is called by class name within the same class or in another class.
- Static methods can be called without creating an instance of the class.
- Static method can be called directly, from within the class they are declared, whereas non-static method can be called from other classes.

⇒ Class.staticmembers;

Non-Static :

- A "non-static method" is an instance method and belongs to each object that is generated from the class.
- If a constant is not-static, Java will allocate a memory for that constant in every object of the class. (i.e. one copy of the constant per object)
- Instance (non-static) variables are created when an object is created with the use of the keyword "new" and destroyed when the object is destroyed.
- non-static member are called with the object name in other classes.
- non-static method cannot be called without creating an instance of that class.

=> Class refvar = new constructor();

refvar.nonstaticmember();

Types of Constructors

There are basically two types of constructors.

- Implicit Constructor
- Explicit Constructor

Implicit Constructor

Implicit Constructor are created be compiler in case ~~user~~ developers didn't created it Explicitly.

Explicit Constructor

Explicit Constructor are created by developers explicitly and data members can be initialized in an Explicit Constructor.

There are ~~two~~ three types of Constructors.

- Default Constructor

- Parameterize Constructor
- Copy Constructor

(i) Default Constructor

Explicit Constructor with no parameter and no initialization. i.e.

```
public ClassName () {
}
```

(ii) Parameterize Constructor

Explicit Constructor with parameterss and initialization i.e.

```
public ClassName (int Variable) {
Variable = constant;
}
```

(iii) Copy Constructor

Copy Constructor are such constructor which another object of the same class as a parameter to initialize.

```
public ClassName(ClassName Object){
    .
    .
    .
}
```

```
ClassName ref = new ClassName();
ClassName ref2 = new ClassName(ref);
```

ANONYMOUS APPROACH

Anonymous Approach is use to create disposable objects (An Object without any reference variable).

```
Scanner sc = new Scanner(System.in);
String section = sc.next();
String sec = section.charAT(0);
```
} Regular Approach.

```
String section = sc.next().charAT(0);
```
} Anonymous Approach

```
String s1 = new String("HELLO");
String s2 = s1.toLowerCase();
```
} Regular Approach

```
String s1 = new String("HELLO").toLowerCase();
```

"Constant and its Types"

Constant is a fixed, well-defined and non-changable value. There are two types of Constants.

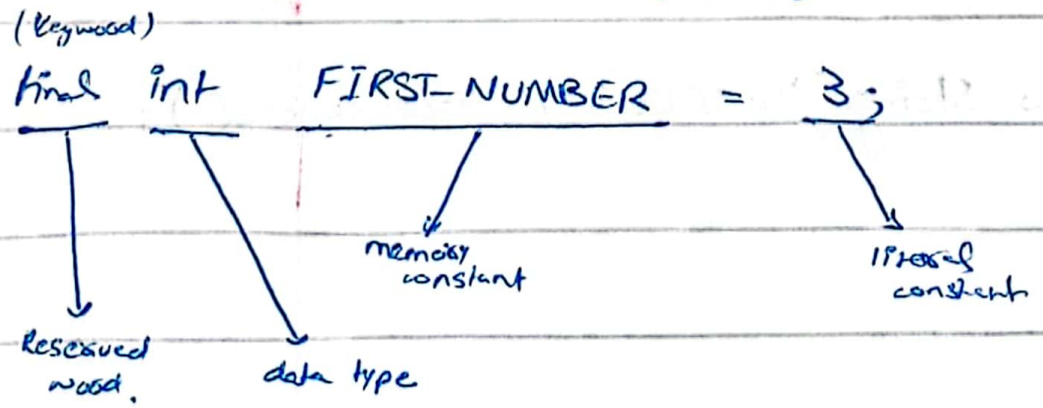
(i) "literal constants"

The values which are constant by self. i.e. 5, 3, 7, -3, etc.

- constants themselves (actual constants)

(ii) "memory constants"

The values which have an identifier and are constants. i.e.



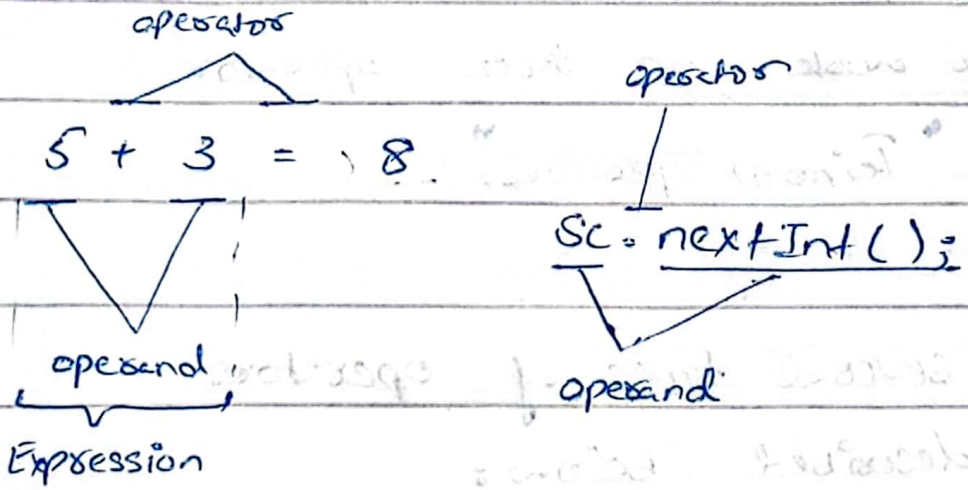
"Expression & Condition"

EXPRESSION

- Combination of Operator and Operand is known as "Expression". which evaluates to a single value.

Operators : Arithmetic, Logical, Assignment etc.

Operands : On which operation is performed.



CONDITION :

- The expression which returns boolean value is called Condition. i.e.

  $a > b$, $a \leq c$ are conditions.

- All conditions are expressions but all expressions are not condition.

Operators and Types

Operators are used to perform different operations between Operands.

- If operator work on single operand known as "Unary Operator".
- If operator work on two operator known as "Binary Operator.
- If operator work on three operator known as "Ternary Operators."

These are several types of operators which are described below:

ARITHMATIC OPERATORS:

These are different Operators.

+	binary	to add
−	binary	to subtract
*	binary	to multiply
/	binary	to divide

%	binary	to reminder
++	uniary	to add 1 to (higher presidence)
++	uniary	to add 1 (lower presidece)
--	uniary	to subtract 1 (higher presidence)
--	uniary	to subtract 1 (lower presidence)
? :	ternary	to check condition.

RELATIONAL OPERATORS:

There are different relational operators as follow:

("Expression using relational operators are mostly conditions")

>	binary	greater than
<	binary	lower than
>=	binary	greater than equal to
<=	binary	less than equal to
==	binary	equal to
!=	binary	not equal to

LOGICAL OPERATOR:

There are three logical operators.

"AND operator" && binary

C1	C2	R	
T	T	T	"Both Statements Must be True"
T	F	F	
F	T	F	
F	F	F	

"OR Operator" || binary

C1	C2	R	
T	T	T	"Any of Statement Should be True"
T	F	T	
F	T	T	
F	F	F	

"NOT operator" ! Unary

c1	R
T	F
F	T

"To change the result"

ASSIGNMENT OPERATOR :

= binary to assign a value to a variable.

COMPOUND ASSIGNMENT OPERATOR :

a+=2	Unary	a = a+2
a-=2	Unary	a = a-2
a*=2	Unary	a = a*2
a/=2	Unary	a = a/2

a %= 2	Uniary	a = a %. 2
a &= 2	Uniary	a = a & 2
a |= 2	Uniary	a = a | 2

Presidence & Assosiativity

Presidence:

The operator which have higher presuidence resolved first every operator have its own presuidence.

Assosiativity:

The operator having same presidence get resolved from L to R or R to L which is known as Assosiativity of Operator.

- Assignment Operator Assosiativity is Right to Left.
- Arithmetic Operator Assosiativity is Left to Right.

Side Effect of Operators :

Any operators by which the value of operand gets changed called as side effect of operators.

+, -, *, / have no side effect.

but, =, ++, --, += ____ have side effect.

a = a % 2

a = a * 2

a = a | 2

iativity

or which have resolved first its own presuidence.

Ass

having same presuidence

ge. resolved from L to R or R to L

which is known as Assosiativity of Operator.

- Assignment Operator Assosiativity is Right to Left.
- Arithmetic Operator Assosiativity is Left to Right.

Number System

Binary	0-1	2 digit decimal system
Octal	0-7	8 digit decimal system
Decimal	0-9	10 digit decimal system
Hexadecimal	0-9 a-f	16 digit decimal sys

Decimal to Binary :

Binary to Decimal :

- $(125)_{10} = ?$

256	128	64	32	16	8	4	2	1
0	0	1	1	1	1	1	0	1

$= (001111101)_2$

- $(001111101)_2 = ?$

$(0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 1)_2 = (125)_{10}$

(256 128 64 32 16 8 4 2 1)

$= (125)_{10}$

Octal to Binary :

Binary to Octal :

- $(573)_8 = (?)_2$

5	7	3
4 2 1	4 2 1	4 2 1
1 0 1	1 1 1	0 1 1

$= (101111011)_2$

- $(\overline{101100101111})_2 = (?)_8$

010	110	010	111
4 2 1	4 2 1	4 2 1	4 2 1
2	6	2	7

$= (2627)_8$

Binary to Hexadecimal:

Hexadecimal to Binary:

- $(1101100101010011101011)_{2} = (?)_{16}$

2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1

$(1101\ 1001\ 0101\ 0011\ 1010\ 1011)_{2}$

3 6 5 4 14 11

$= (3654eb)_{16}$

10 = a

11 = b

12 = c

13 = d

14 = e

15 = f

- $(2af3)_{16} = (?)$

2 a = 10 f = 15 3

8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1

0010 1010 1111 0011

$= (0010101011110011)_{2}$

Octal to Hexadecimal:

decimal to Octal:

Hexadecimal to decimal:

These conversions or vice versa can occur using binary number system.

Octal → binary → decimal

Hexadeci → binary → octal.

etc.

BITWISE OPERATOR:

Bitwise Operators are used to resolve the operation not bitwise.

Bitwise AND: &

	8	4	2	1
11	1	0	1	1
& 6	0	1	1	0
2	0	0	1	0

11 & 6 => 2

Bitwise OR: |

	8	4	2	1
11	1	0	1	1
| 5	0	1	0	1
16	1	1	1	1

11 | 5 => 16

Bitwise XOR: ^

	8	4	2	1
7	0	1	1	1
^ 9	1	0	0	1
14	1	1	1	0

7 ^ 9 => 14

Left-Shift Operator: <<

a = 5;

a << 1

8	4	2	1
0	1	0	1
1	0	1	0

=> 10

Right-Shift Operator : >>

a = 3

a>>1

=> 1

8	4	2	1
0	0	1	1
0	0	0	1

unSigned Right-Shift Operator : >>>

In unsigned Right-Shift Operator most significant bit use to express sign does moves.

a >>> -10

=> 2147483643

sign bit

1 0 ... 0 1 0 1 0

0 1 0 ... 0 1 0 1 ..

"Control Structure"

In control structure, we control the flow of execution of statement(s) by implementing any condition or Iterations based on condition etc. Some control structures are described bellow:

If - statement

```
If (condition)
    Statement (s);
```

i.e.

```
If (a > 5)
    sout ("a is greater than 5");
```

If-Else Statement

```
If (condition)
    statement(s);
else
    statement(s);
```

i.e.

```
if (age > 18)
    sout ("Eligible");
else
    sout ("Not Eligible");
```

Nested-If statement

```
If (condition)
    If (condition)
        statement(s);
```

i.e.

```
If (marks >=60)
    If (attendence >= 80)
        sout ("Pass");
```

Nested Else-If :

```
If (marks >= 90)
    sout ("A Grade");
else if (marks >= 80)
    sout ("B Grade");
else if (marks >= 70)
    sout ("C Grade");
else if (marks >= 60)
    Sout ("D Grade");
else
    Sout ("Fail");
```

```
If (condition)
    Statement(s);
else if (condition)
    Statement(s);
else
    Statement(s);
```

Compound Statement Operator

If in general if and else control single statement under some condition. but using compound statement operator you can ensure that more than one statement depends upon if-else.

i.e.

```
if(marks > 60)
    sout("pass");
```

Ex:-

```
if (mark > 60) {
    sout("pass");
    sout("congratulations !!"); }
```

compound statement operator

Switch - Statement:

```
switch (expression) {
 case constant1:
        statement(s);
        break;
 case contant2:
        statement(s);
        break;
 default:
        statement(s);
        break;
}
```

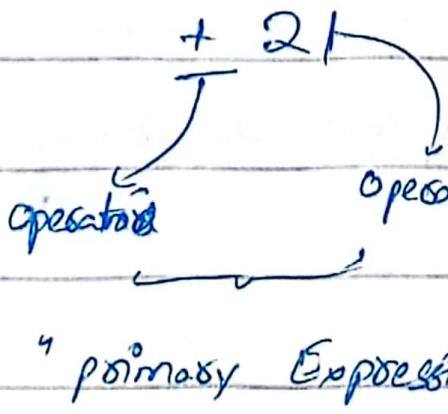
```
switch (op) {
    case '+':
        sout ("Addition");
        break;
    case '-':
        sout ("Subtraction");
        break;
    case '*':
        sout ("Multiplication");
        break;
    default:
        sout ("You entered wrong operator");
        break;
    }
```

" Switch is much faster than nested else-if in runtime thats why where we can

use switch passing, we should use it "

Primary Expression

As we know, expression is combination of operator and operand. Any expression with only one operand have "+" operator by default that is why it is called as "Primary Expression"



Dangling else-if and switch

Any else-if which have no last "else" and switch which have no "default" statement known as "Dangling else-if" and "Dangling switch Respectively".

Using Switch as Expression

Modern Switch / Switch Expression

```
int numOfLetters = switch (day)
{
case MONDAY, FRIDAY, SUNDAY → 6;
case TUESDAY → 7;
case THURSDAY, SATURDAY → 8;
case WEDNESDAY → 9;
};
```

- In modern switch we can return the value of switch.

```
return switch (op) {
    case '+' → a+b;
    case '-' → a-b;
    case '*' → a*b;
    case '/' → a/b;
    case '%' → a%b;
    default → 0;
};
```

- A switch expression should cover all possible values.
- we use lamda or arrow "→" in switch expression and break statement is implied in it.

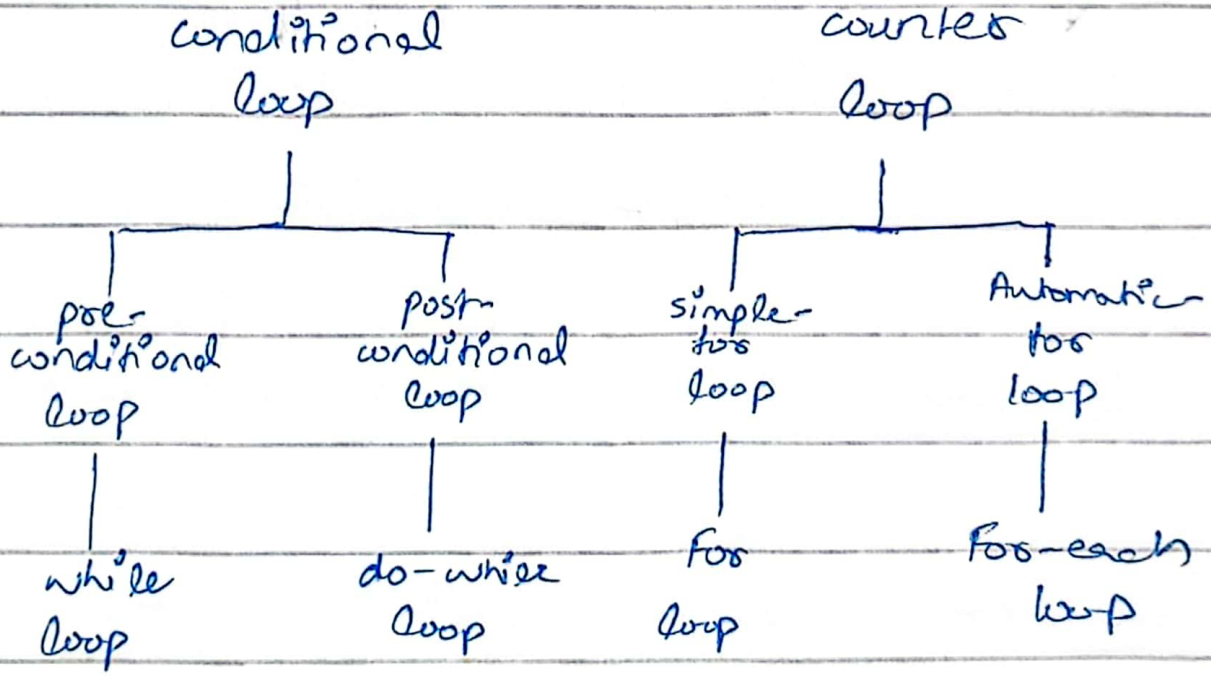
- we can directly assign switch expression to some variable.

```
String s = switch(expression)
        { case label -> return ;
          case label -> return ;
        };
```

- can't left switch expressions dangling

"Loops / Iterations"

A loop is defined as a block of code that executes multiple times depending on some condition. There are two types of loops

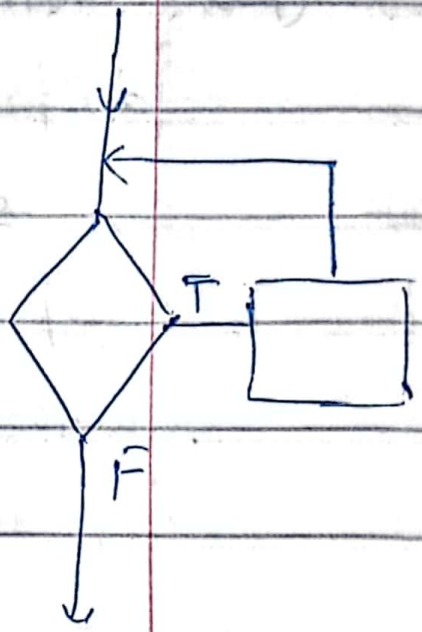


** method name starting with "has" must have boolean return type. (convention) **

i) while - loop

(pre-conditional loop)

```
while (condition){
    statement(s)
}
```

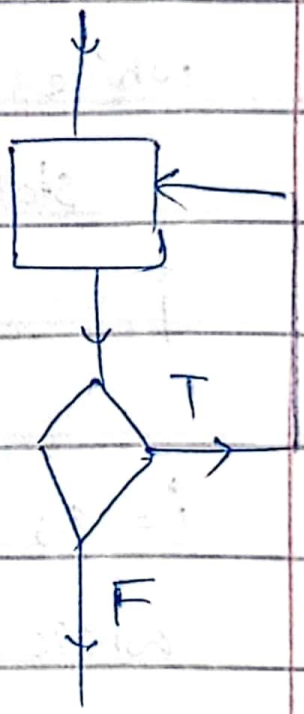



```
i = 1;
while( i <= 5){
    sout (i);
    i++;
}
```

(ii) do-while

(post-conditional loop)

```
do {
    statement(s);
}
while (condition);
```




```
do {
// calculations
}
while (any condition);
```

- It will minimum execute the block one time.

(iii) For- loop

(Simple for loop)

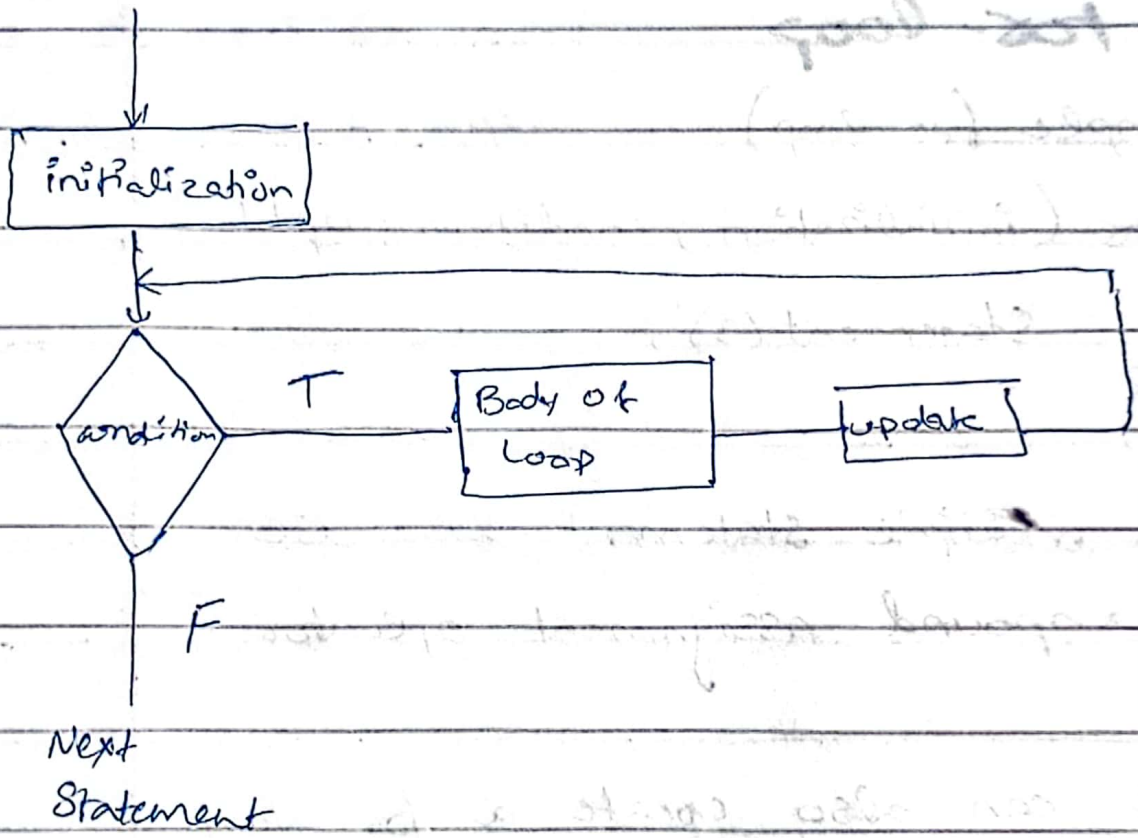
- for (initialization ; condition ; update) {
  Statement (s) ;
  }
- for multiple statement we use compound assignment operator { }.
- we can also create a for loop with null body.

for (initialization; condition; update) ;

- we can initailize and update multiple variables in same loop.

Variables which are involve in condition are ~~counter~~ counter variable.

Syntax:


initialization
condition
T
Body of Loop
update
F
Next Statement

(iv) For-Each loop

For-Each loop is a read-only loop will be use to traiverses any collection.

↓ iterate.

data type ↑ ref variable

```
for (type var : collection) {
        Statement (s);
}
```

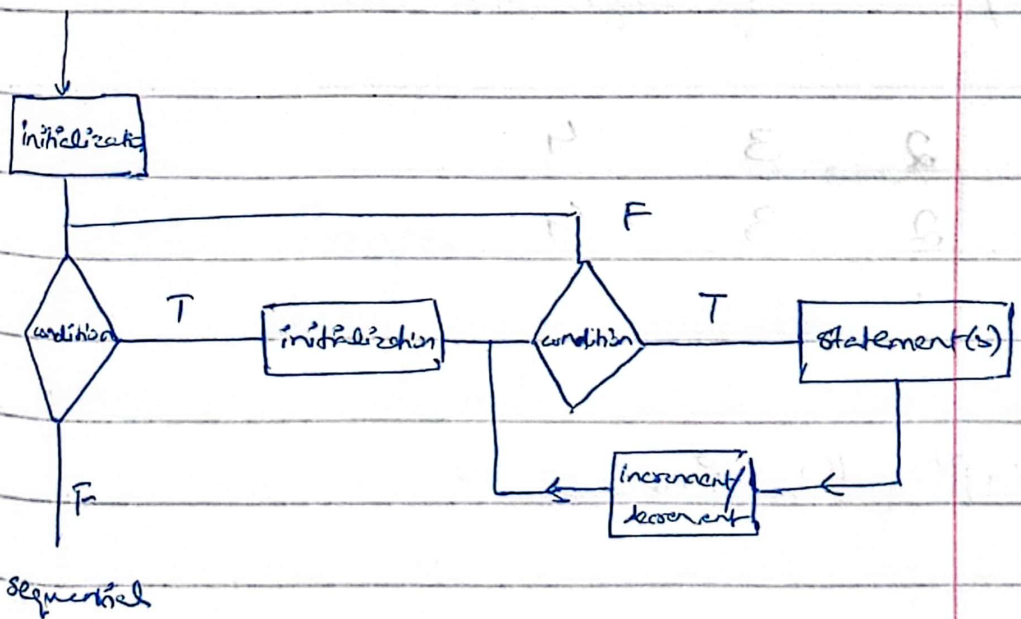
collection to iterate.

statements to execute.

Nested- loop

A nested loop is basically a loop within another loop's body.

For every iteration of outer loop inner loop will execute maximum times.



lets take an example of for nested loop

```
for(int i=1; i<=10; i++){
    for(int j=1; j<5; j++){
        sout(j)
    } sout(" ")
}
```

1 2 3 4
1 2 3 4
:
:

upto 10 times.

Images Processing

Image

Image is a two Dimensional data (width, Height). made up of mathematical shapes, visuals or Pixels.

There are two type of Images.

(i) Vector Image

Vector Image contains geometrical and mathematical information that is why it can easily be magnified. and zoomed in and zoomed out with clear scope.

(ii) BitMap / Roasted Image

Screen is made up of tiny level bulbs which are known as pixel.

The Image contains limited information about pixels. That's why it cannot be too much maximized. It is also called raster Image.

Mapping bits (pixels) on a 2 dimensional grid can known as a bitmap image.

Important Points

- Images used today commonly are bitmap images that is why they start getting blur as we zoom in.

- .jpg, .png, .apng. .bmp are basically different compression algorithms which compress the information of pixels.
- .bmpg has highest quality as it contains maximum information with less memory space.

Exception Handling

Handling Runtime Errors or Exceptions is Exception Handling. Exceptions can cause crashing your software.

Using Exception Handling technique will first try that code and if it does not ~~Also~~ throw any exception it will execute smoothly.

Otherwise we will catch that exceptions and show the error or any thing we want to proceed in catch block.

After it a finally block present which will execute eventually either run-time error occurs or not.

Syntax

```
try {
        statement(s);
    }
catch ( Exception ex){
    sout ( ex.getmessage());
    }
finally {
        statement(s);
    }
```

If run-time error occurs

will execute eventually

if you want to catch any specific exception then write specific catch block first and general catch block later.

File Class

Java File class is basically used to represent files and folders pathnames in an abstract manner.

The File object represents the actual file/directory on the disk.

```
File file = new File( URI );
                      "pathname"
                        or
                    "constructor overloaded."
```

Syntactical String (String with proper Syntax) is called URI.

ImageIO Class

Java ImageIO class is used to read and write image on disk using static functions ImageIO.

```
ImageIO.read(file);
ImageIO.write(file);
```

BufferedImage Class

Image loaded from Hard Disk into RAM is called a Buffered Image.

This class relies on data fetching and setting methods of Rasters.Image.

```
BufferedImage sourceImage = null;
```

BufferedImage → class
sourceImage → reference variable.

For having an image file in IDE to do any process on it;

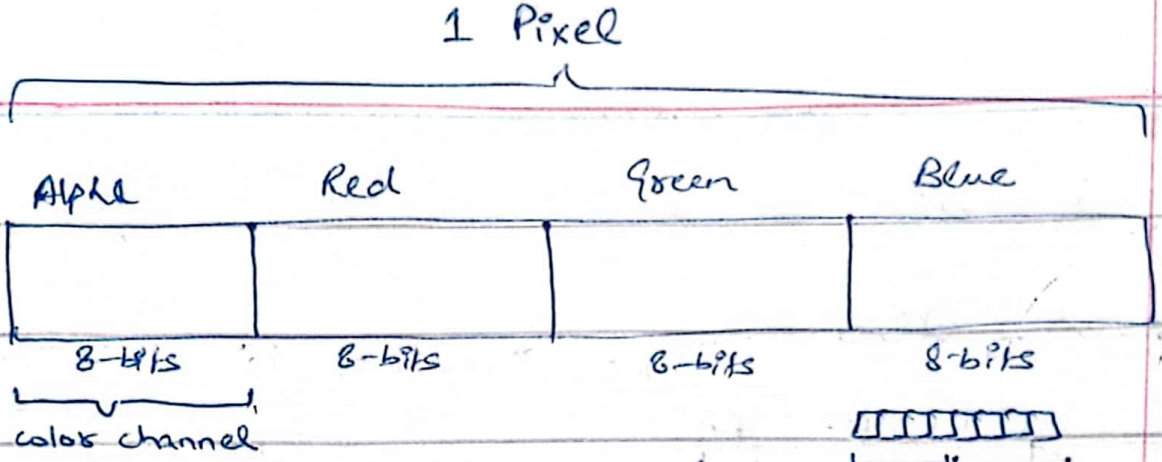
```
BufferedImage referenceVariable = ImageIO.read(
                                    file);
```

"Filtess in Image Processing"

For applying or creating different filtess using Image Processing, you change the values of RGB system present in the image.

For that first you need to seperate RGB value from a pixel of the Image.

For ARGB



0xFF

to display hexadecimal

maximum value = 15

In, hexadecimal ; 15 = F;

So, for seperating "Alpha"

alpha = pixel >> 24;

Alpha color channel will be shifted to very right

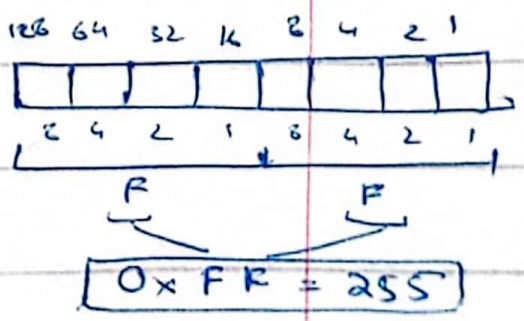
Similarly for seperating Red

Red = pixel >> 16;

Red color channel will be shifted to very right but Alpha channel still exist in pixel. for that do (bit-wise and) with 8-bits value so all other channels will be nullify

Red = Red & (0xFF);

as 0xFF = 255;

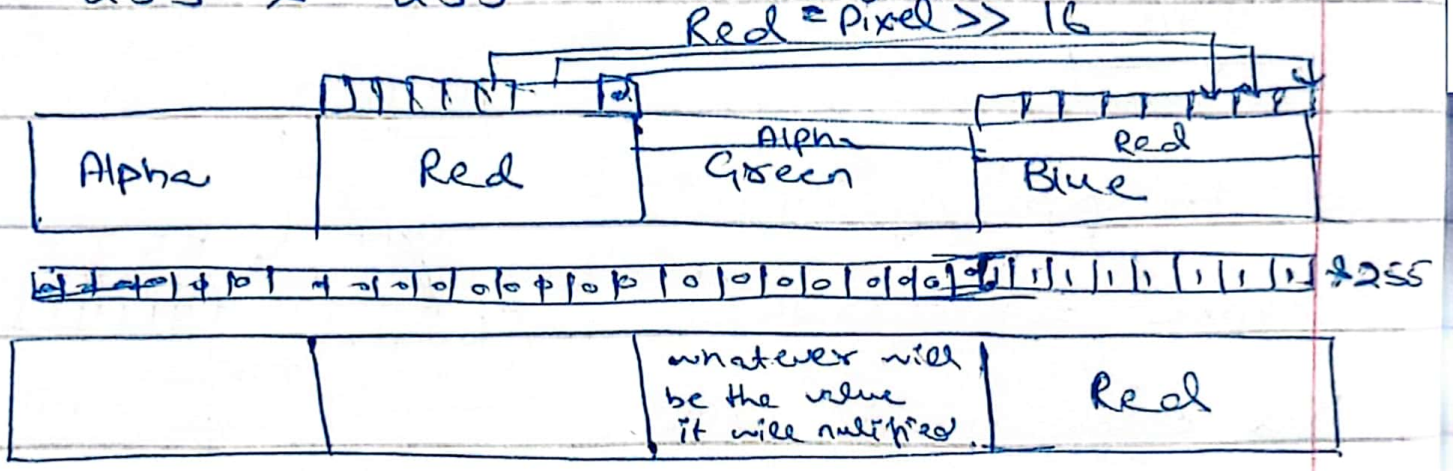


For example if red is;

253 so after doing bit-wise AND & it will be

Red & 255

253 & 255



So when we do the similar logic to seprate All four color channels.

alpha = pixel >> 24; (right - shift)

red = pixel >> 16 & 0xFF; (right - shift)

green = pixel >> 8 & 0xFF;

blue = pixel & 0xFF;

As blue is already at very right positon.

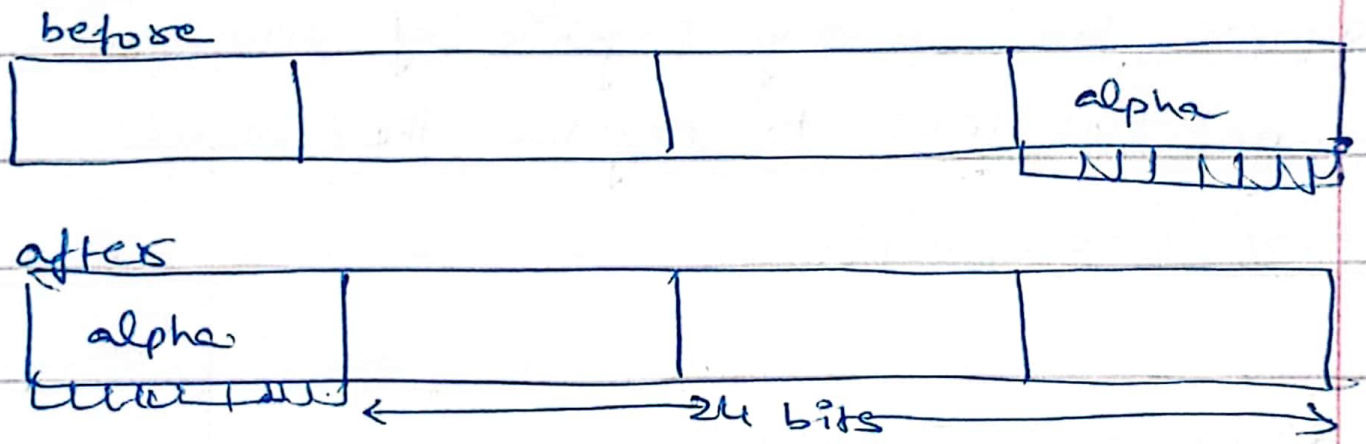
now you have the value of every color channel in independent vasicbles. so you create any filter by using any logic or any PCS (panga in computer science) for example for creating negative of any image, you have to negative the value of every color channel.

```
alpha = 255 - alpha;
red = 255 - red;
green = 255 - green;
blue = 255 - blue;
```

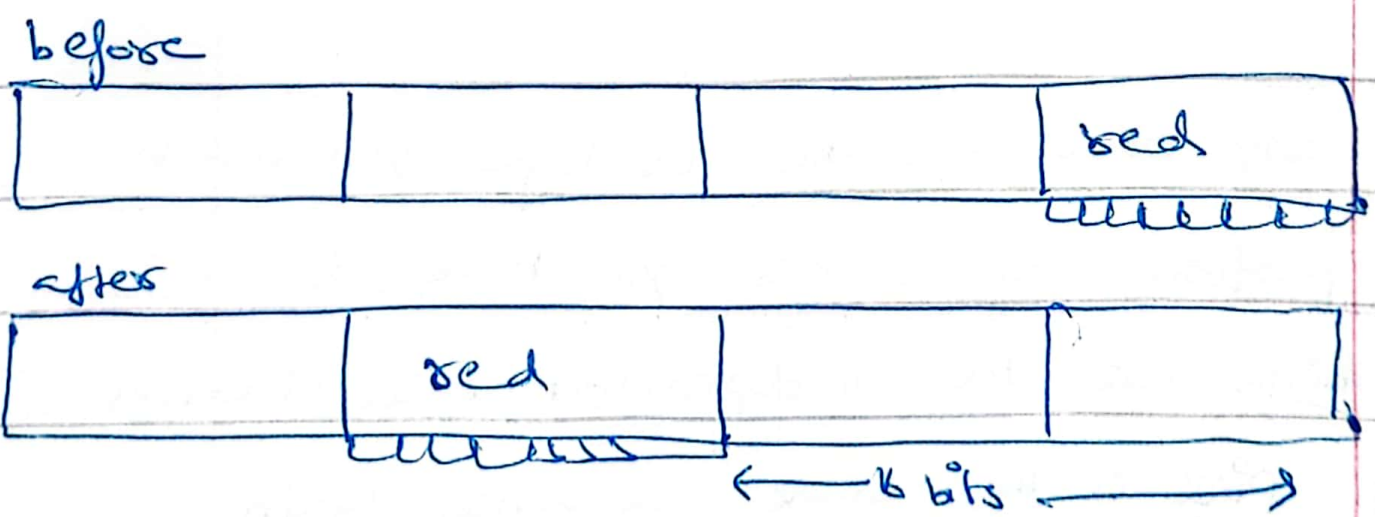
as any other operation or logic you want to perform after this you have to combine all the independant color channel of pixel in the same sequence again.

For that first we have to send every color channel to its original position.

alpha = ~~alpha~~ pixel << 24; (left-shift)



red = ~~red~~ pixel << 16; (left-shift)

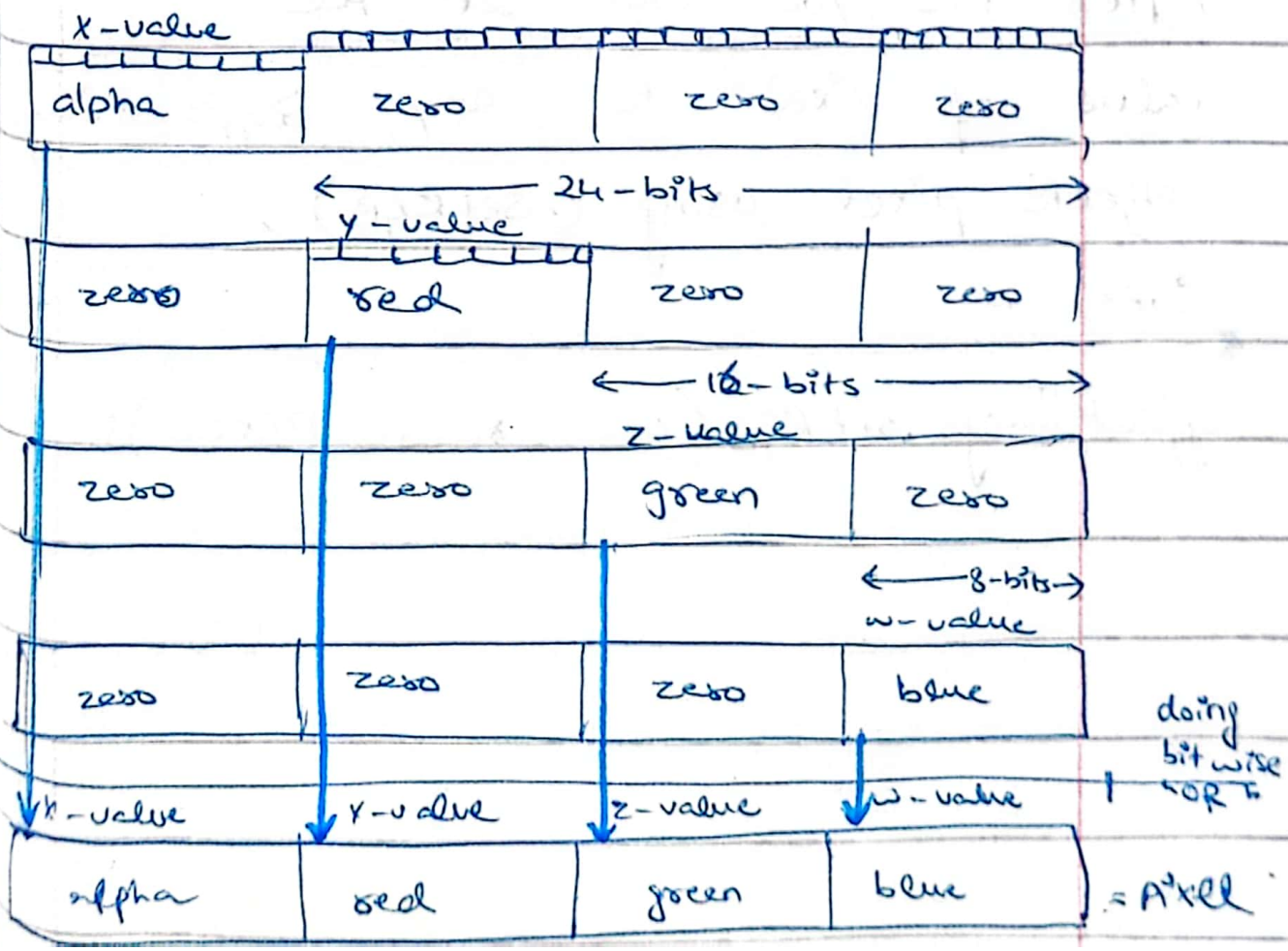


same for,

pixel
green = ~~green~~ << 8

and blue is already on its original position.

now by doing (bit-wise OR) we will have the pixel value again without losing information and in the appropriate sequence.



it will be like

```
pixel = alpha<<24 | red<<16 | green<<8 | blue;
```

this is how you can manipulate the value of color channels in every pixel of the image using any logic and creates filtes.

After this you can set the value of pixel to any BufferedImage object's pixel using (.setRGB),

i.e.

```
effectImage.setRGB(col, row, pixel);
```

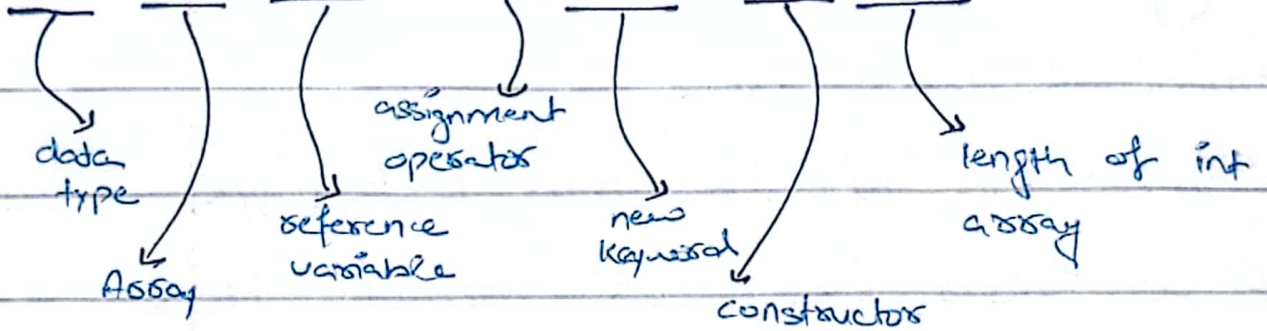
"Arrays"

- Primitive Data Type.
- Combination of Homogenous Data Type.
- A contigious (continues) memory block of same data type.
- Arrays are referenced type.
- Strings and Arrays are only referenced primitive data type.
- Arrays are declared dynamically. means object created in heap and reference variable stored in stack.
- Once Array declared its length can never be changed.
- {} is literal constant of arrays.

```
int [] list1 = {1, 2, 3, 4, 5}
int [][] list2 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
```

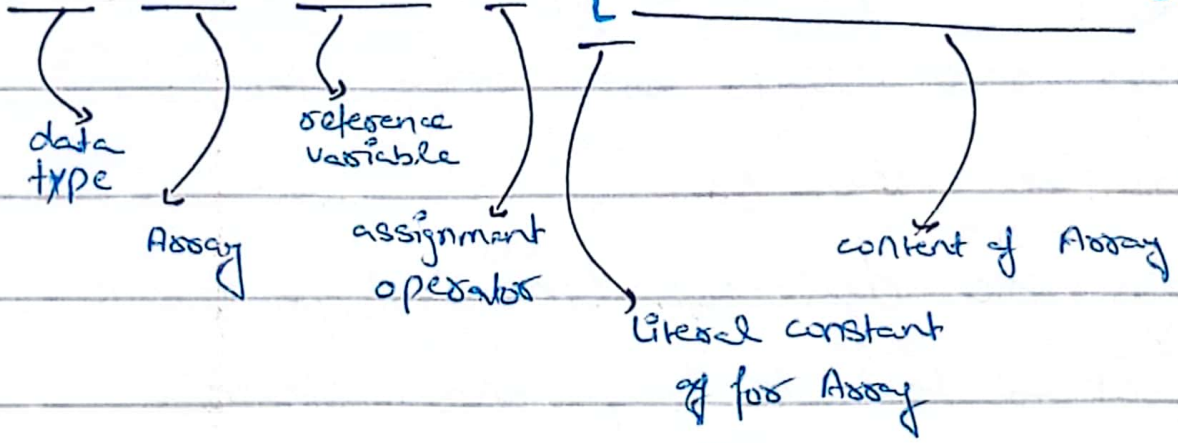
SINGLE DIMENSION ARRAY

```
int [] list1 = new int[5];
```



index 0 1 2 3 4

```
int [] list2 = {1, 2, 3, 4, 5}
```



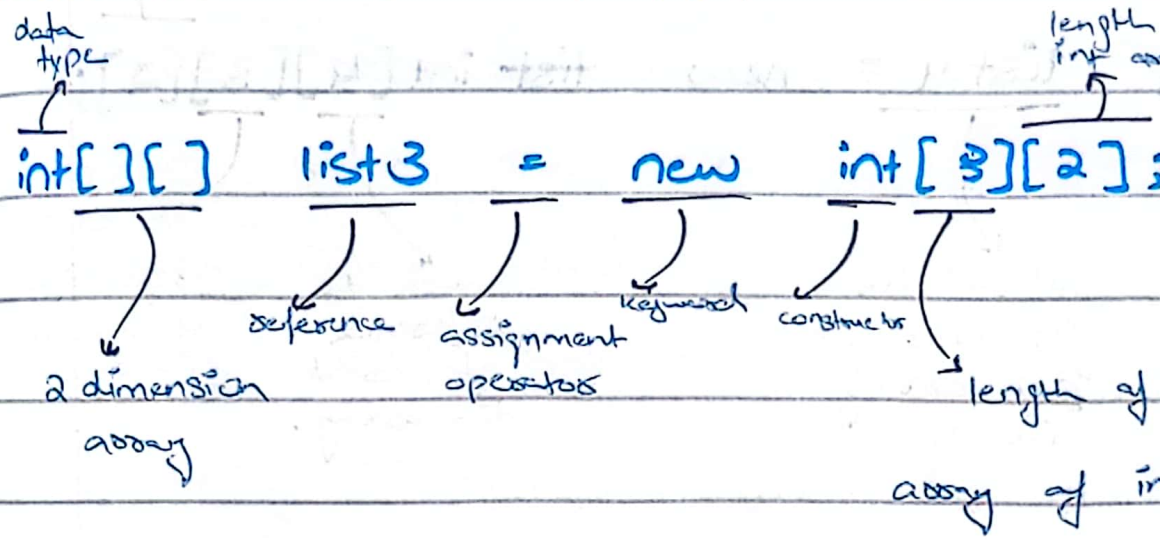

```
int[] list1;  → new way
int list1[];  → old way
```

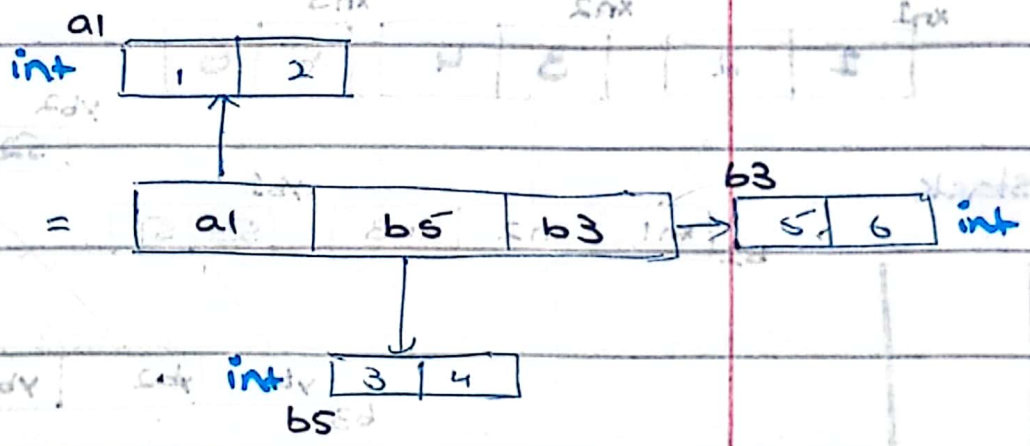
To access

```
→list1[3];
Array→list2[2];
```

index

MULTI DIMENSION ARRAY





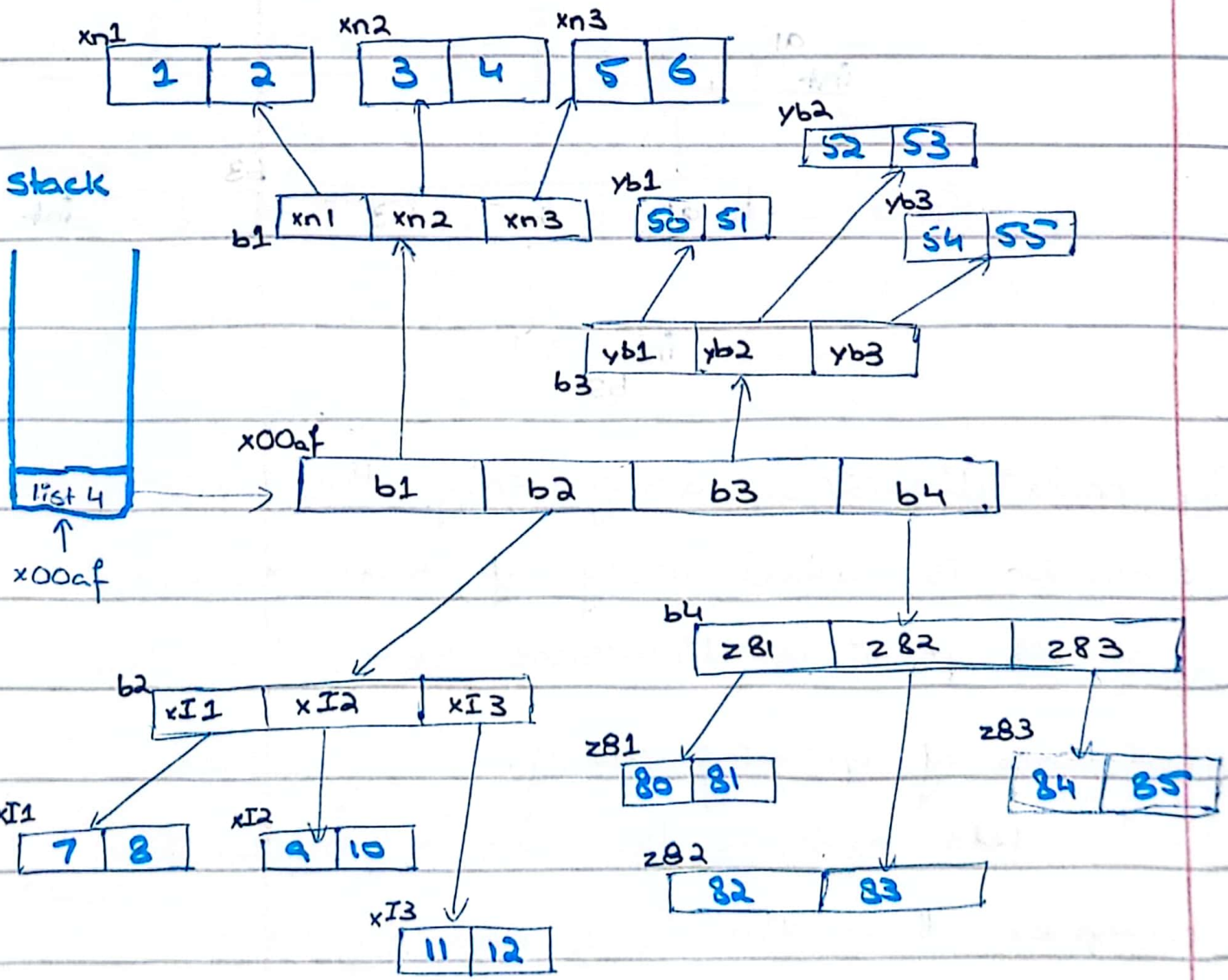
In multi dimensional array only the last dimension is actual array of that data type all previous dimensions store the reference of upcoming array.

lets have a look on a more complex three dimensional Array.

actual int array

```
int [][][] list4 = new int[4][3][2];
```

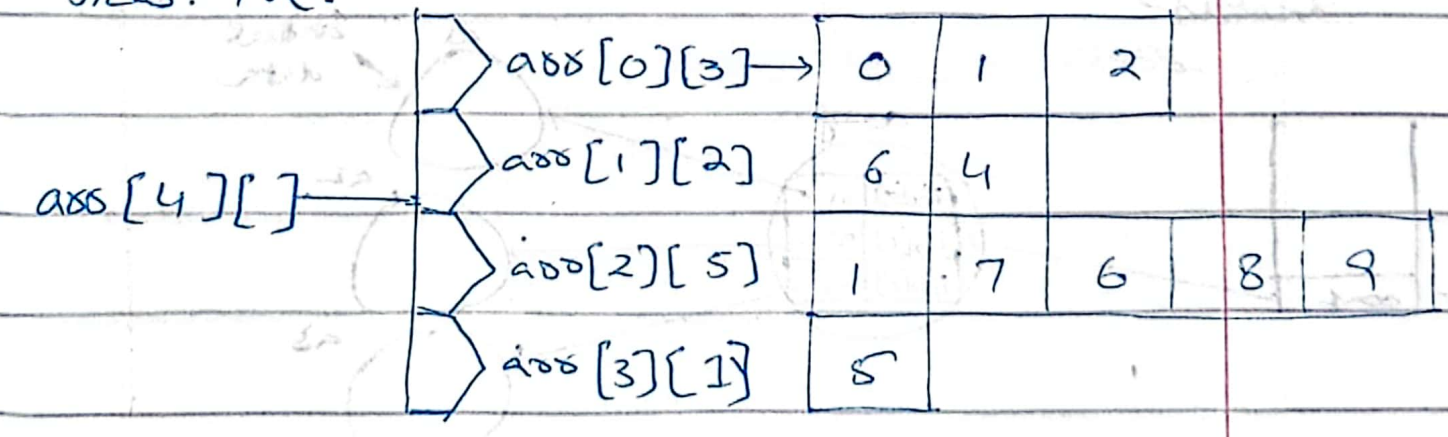
- `list4` → Stores the reference of first array i.e. 4
- `[4]` → Stores the reference of second dim array.
- `[3]` → Stores the references of int array



In this previous example list 4 reference variable stores the reference of arrays at address "x00af" which contains addresses of arrays which further contains the addresses of array which actually contains the Integer data.

GG
JAGGED ARRAYS

A Jagged Array is an array whose elements are arrays, possibly of different sizes. i.e.



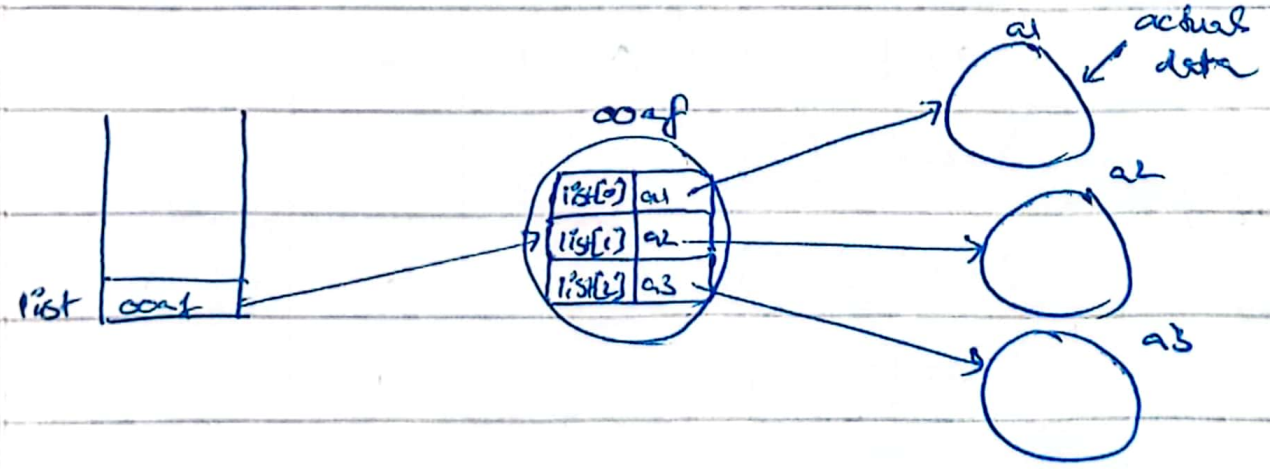

```
int[][] arr = new int[4][];
```

REFERENCE TYPE ARRAY

Array itself is always a reference type of data as object of Array always created in heap. regardless that which type of Array it is. i.e. int[] or double[].

but further if we create Array of a reference data type. Array will further contain the references of objects. i.e.

String [] list = new list[3];

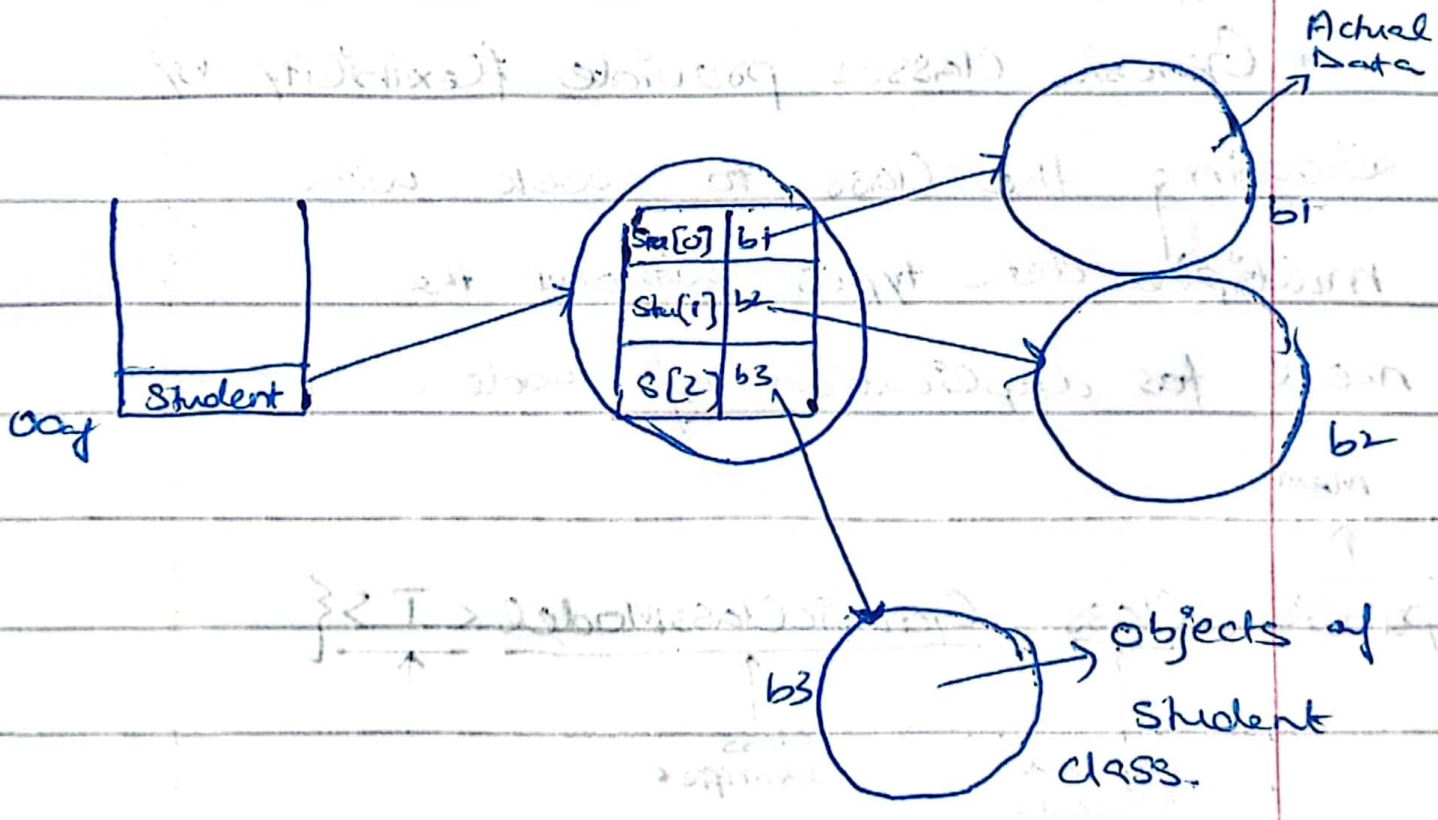


In such cases, Array will contain the

address of data stored in another object.

lets look at another example of a custom class. Student

```
Student[] s = new Student[3];
```



Generic Class:

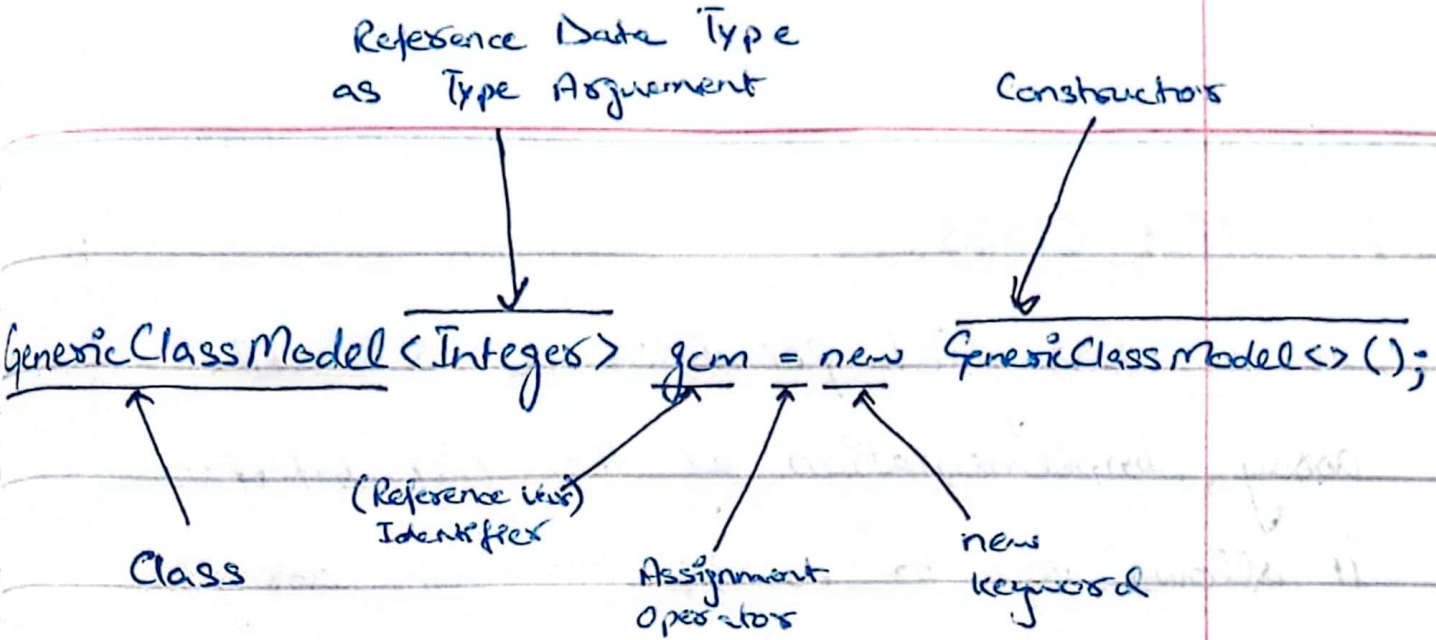
- Generic classes encapsulate operations that are not specific to a particular data type.
- Generic classes provide flexibility by allowing the class to work with multiple data types without the need for duplication of code.

MAM

```
public class GenericClassModel<T>{
    .
    .
    .
    .
    .
    .
    .
    .
    .
}
```

- `class` — Keyword to create class
- `GenericClassModel` — Class Identifier
- `<T>` — Type Argument which will specifies the data Type at the time of creating an instance (Object) of class.

Reference Data Type should be passed as Type Argument to ensure type safety and enable the class to operate correctly.



- Generic classes also known as "Template Classes".
- A few generic classes are as follow:

(i) ArrayList <E> (ii) LinkedList <E>

(iii) Optional <T> (iv) HashSet <E>

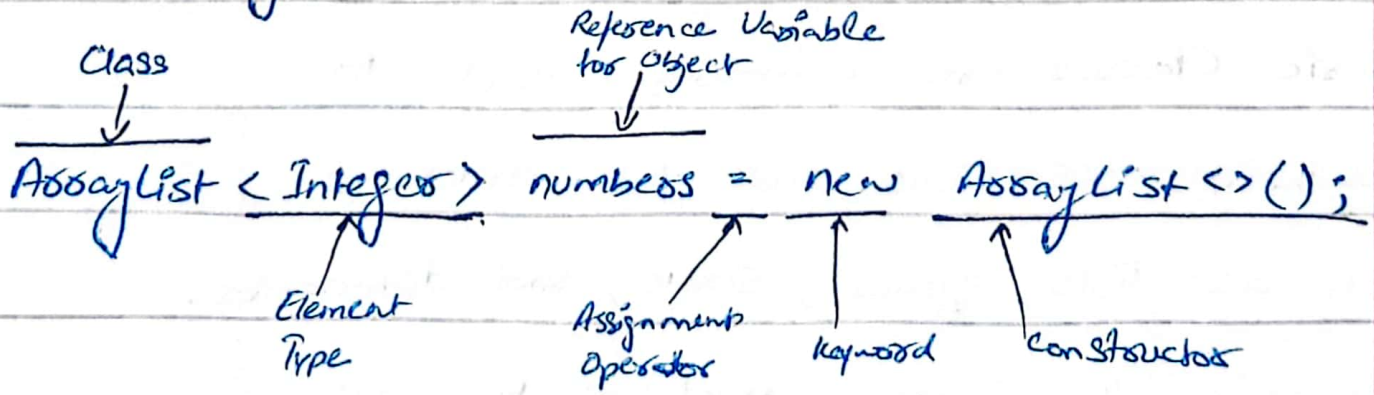
(v) PriorityQueue <E> etc.

- Generic Classes are commonly used to implement collections and data structures such as lists, queues, stack, and dictonaries. They allow theses data structures to work with various element types.
- It can also be called polymorphism of objects. (Pg # 112)

ArrayList Class

ArrayList provides a resizable array implementation of the List Interface. It allows you to create dynamic arrays that can grow and shrink in size as needed.

You can create the instance of an ArrayList (object of the class ArrayList) by specifying the type of elements (which will make it a specified class) within the angle brackets.



Unlike regular array, ArrayList automatically resize themselves as elements are added or removed. The initial size of an ArrayList is zero or specified during initialization.

i.e.

= new ArrayList<>(500);

initial capacity.

When elements are added beyound current capacity, the ArrayList automatically increases its capacity by allocating a larger internal array and copying the existing element.

To add any elements,

numbers.add(1);

numbers.add(8);

To access any element,

numbers.get(1);

↳ Index

To set any element,

```
numbers.set(0, 20);
```

0 → Index, 20 → value

To remove any element,

```
numbers.remove(2);
```

2 → Index

To check if it is empty,

```
numbers.isEmpty();
```

To get the size of list,

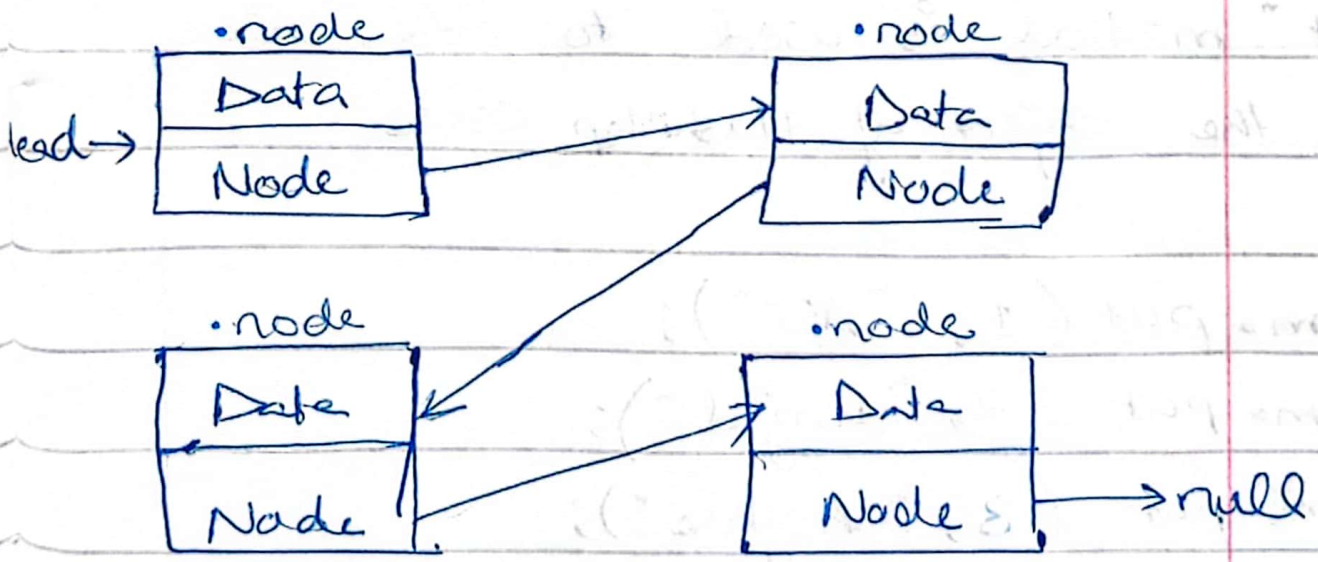
```
numbers.size();
```

and much more.

LinkedList Class

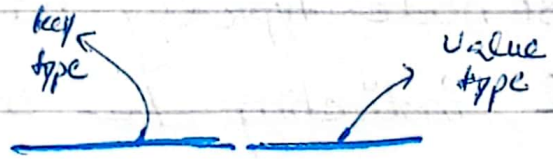
A Linked List is a data structure that consists of a sequence of nodes, where each node contains data and a reference (or link) to the next node in the sequence.

The last node typically has a reference to null, indicating the end of the list. LinkedList provides dynamic memory allocation and efficient insertion and deletion operation.



HashMap

HashMap represents a key/value pair whose every unique key refers to its value. Once the value is stored you can retrieve it using its key.



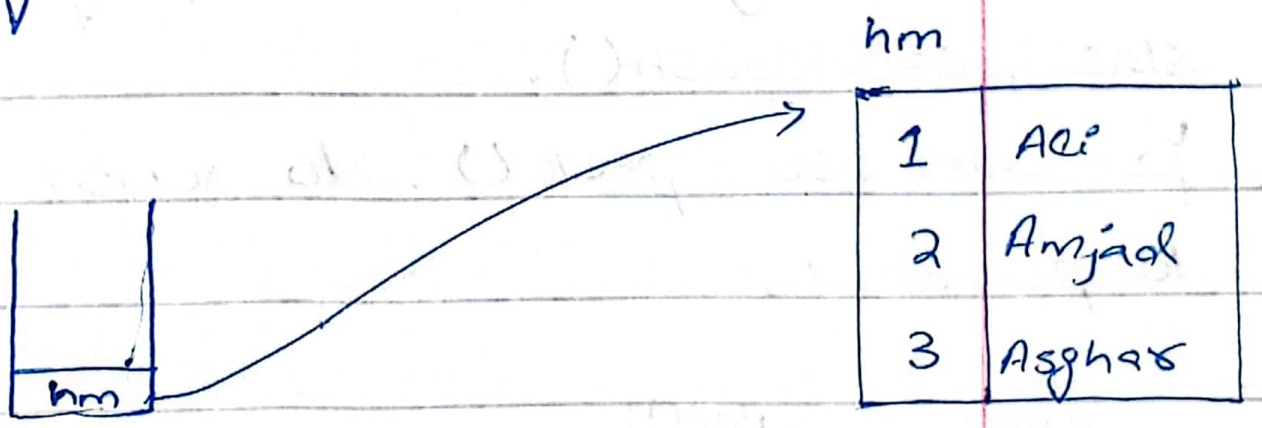

```
HashMap<Integer, String> hm = new
        HashMap<>();
```

"put" method is used to add value to the object of HashMap class.

```
hm.put(1, "Ali");
hm.put(2, "Amjad");
hm.put(3, "Asghar");
```

"get" method is used to get the value by passing that unique key as parameter.

```
hm.get(2);
```



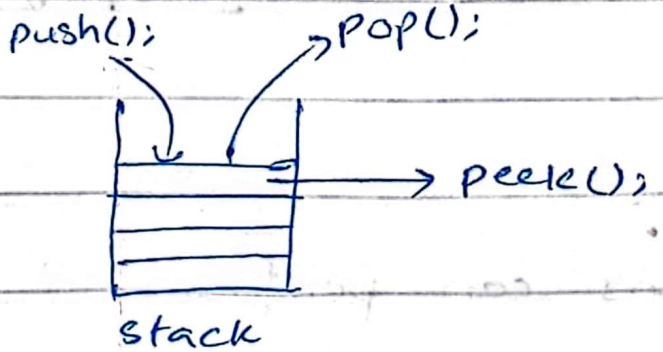
Stack

Stack Collections can perform limited operations as it considered as a filtered stack with the principle of LIFO (Last In First Out).

so, you can ~~put~~ pop, ~~pull~~ push

or peek in stack.

To remove and return the top element, call pop().
To put an object on the top of the stack, call push().
You can use peek(). to return the to pelement.



Inheritance:

Inheritance is another basic pillar of OOP. because it allows the creation of hierarchical classifications.

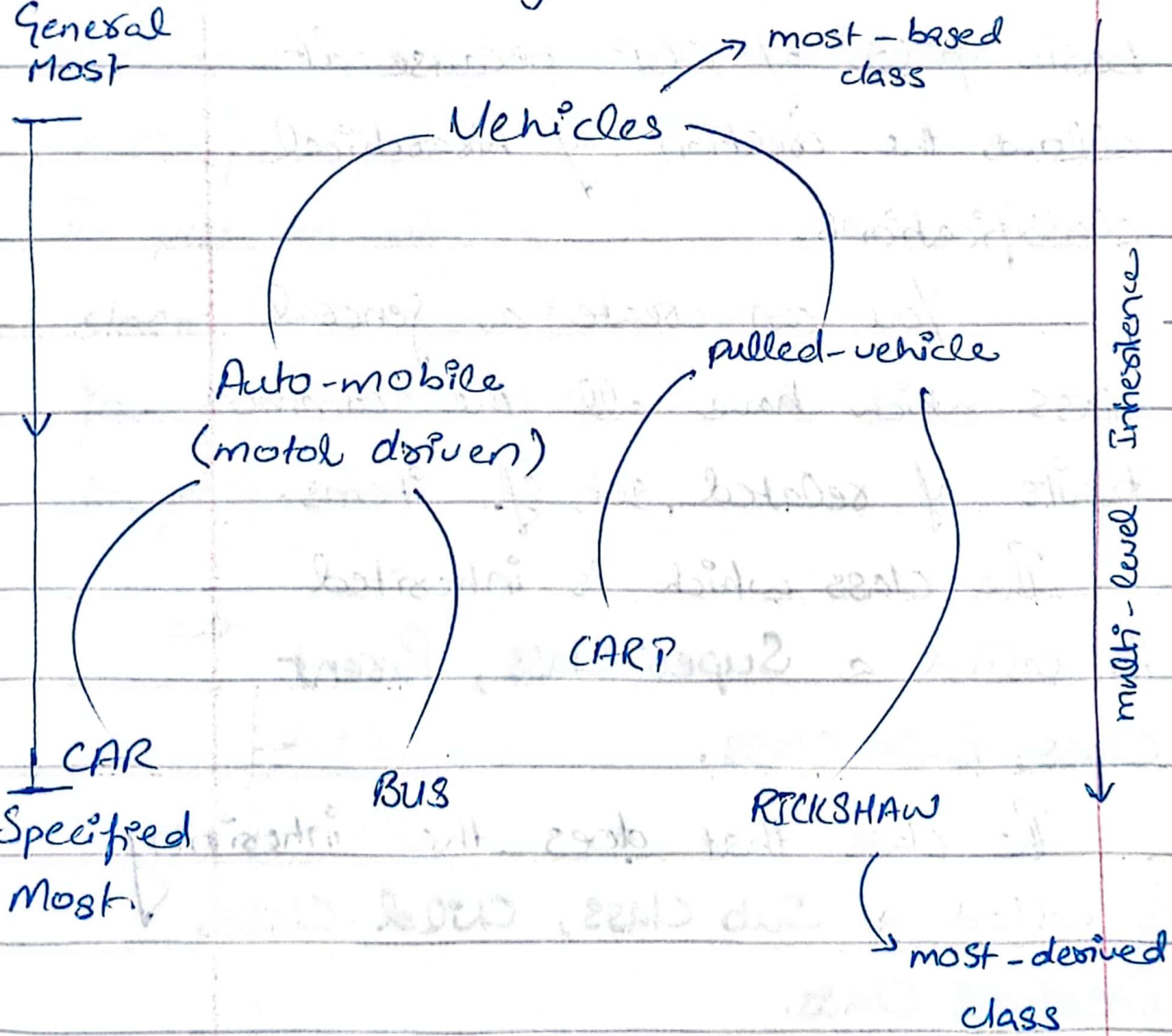
You can create a general class which have all the common traits of related set of items.

The class which is inherited is called a **Super Class, Parent Class, Base Class.**

The class that does the inheriting is called a **Sub Class, Child Class, Derived Class.**

A sub-class is a specialized version of a super class. sub-class inherits all the members of super class.

we can also perform multi-level inheritance in Java.



RIKSHAW will have all the members of (pulled-vehicle) and (pulled-vehicle)

already inherited all the members of Vehicles class. so RIKSHAW have indirectly inherited vehicles class too.

Reusability :

Inheritence provides reusability of code. The code written in vehicles class will be reusable till the most derived class otherwise we have to write it again and again.

We can inherit a class which fulfilled "is a" rule.

derived class is a base class.

RIKSHAW is a Vehicle.

Generalization

Generalization can be defined as a process where the grouping is created from multiple entity sets and ~~(and)~~ the Specialization takes a sub-set of the higher level entity and it formulates a lower-level entity set.

Simply, when a class represents the shared characteristics of more than one class, is called generalization. for example, Vehicle is a generalization of Bike, Car and Truck.

we inherit the general class towards specialize classes until we have concrete class.

because of generalization, A parent class reference variable can store an object of child class as child have the meta data of parent class but an child object reference variable cannot store the object of Parent reference variable.
Animal a = new Animal();

Generalization

Generalization co
defined as a process wh
grouping is created from
entity sets and ~~(and)~~ the Specialization takes a sub-set of the higher level entity and it formulates a lower-level entity set.

Simply, when a class represents the shared characteristics of more than one class, is called generalization. for example, Vehicle is a generalization of Bike, Car and Truck.

we inherit the general class towards specialize classes until we have concrete class.

Polymorphism

The word "poly" means "many" and "morphism" means "states". Different states of a same functions is called polymorphism.

There are two types of polymorphism.

(i) ad hoc polymorphism

(ii) parametric (or real) polymorphism.

Ad hoc polymorphism

Ad-hoc polymorphism is also known as "overloading". It allows multiple methods with the same name but different signature within single or multiple classes. For example constructor overloading (pg no. 34)

Ad hoc polymorphism means overloading of "function" or "method" can happen is same class of parent-child class while real polymorphism can only happen is parent-child class. (method overriding)

Parametric (Real) Polymorphism:

Parametric Polymorphism allows methods ~~(based on)~~ or classes to be written generically and operate on multiple types using type parameters.

For example generic classes (pg # 99)

Method Overriding

Method Overriding also referred as real polymorphism. Method overriding occurs when a subclass declares a method that is already defined in its parent class with the same name, same return type, and same signature).

When a method in a sub-class overrides a method in its parent class, the sub-class provides its own implementation of the method, replacing the implementation inherited from the parent class.

Here's a detailed example:

```
class Animal {
    public void makeSound() {
        Sout ("Animal makes sound");
    }
}
```

(P.T.O)

```
class Cat extends Animal {
    ~~public~~
  @ override
      public void makeSound(){
      sout(" meow");
      }
  }

class Dog extends Animal {
    @ override
      public void makeSound(){
      sout (" woof");
      }
```

(0.T.9)

```
public class Main {
    public static void main(String[] args) {
    Animal a1 = new Cat();
    Animal a2 = new Dog();
        a1.makeSound(); // output : "meow";
        a2.makeSound(); // output : "woof";
    }
}
```

This is an example that how a child class over-rides the method present in a parent class. Over-riding can only happens between parent and child class. This is the real polymorphism.

In C# language, there are two types of overriding:

- Non-Virtual Overriding
- Virtual Overriding

"Non-Virtual Overriding" also known as

→ Early Binding
→ Compile-time Binding
→ Static Binding

~~In non-virtual overriding, the address~~

Non-Virtual Over-riding:

In non-virtual over-riding, when you created a reference variable of any type, it automatically binds it to the object of the same class, either you assigned it an object of same class or any other child class.

It means that if a function foo(); created in parent class, over-ride into the child class with different functionality. then if you assigned the object of child to reference of parent. (page # 113)

```
Parent P = new Child();
```

still P gets bind with the functionality of function defined in Parent Class.
That's why it is called
Static Binding As binding will be static.

Early Binding As the binding will be
Compile-time early on compile time.

"Virtual-Overriding" also known as

→ Late-Binding
→ Run-Time Binding
→ Dynamic Binding

Virtual ~~Binding~~ Overriding :

In- Virtual ~~Setting~~ Overriding, When you created a reference variable of any type. it does not get bind its function with any functionality including the functions which have throughout the same functionality.

In such overriding, RTTI (Run-Time Type Identifications) ~~ass~~ occuss which decides that which object this reference variable is pointing to. If its the same object of that Type then that initial functionality get binds.

but if its is pointing towards any child objecti. The functions which

get inherited into child class and does not over-ride the functionality binds through that parent functionality

On the other hand if the function get over-ride into the child class than that objects functionality binds withe function.

That's why it is called.

Dynamic Binding As it depends on the object which is assigned.

Late Binding / **Run-Time Binding** As it ouccur on the run-time after considering the object.

Abstraction

As we discussed Earlier that we mimic the real world objects into the digital objects. but we created a few classes in programming which are not actual concrete real world objects but we use these classes to create concrete objects. and also get benefit of generalization using that class. That class would be known as a "Abstract Class" Abstract class couldn't have an object itself as it is not a concrete class.

Simply, Abstract class is created to gain Inheritance advantages.

Abstract function:

A function present in parent class (only if the parent class is abstract) but all of its child classes over-ride the functionality of that function means that function is never going to execute, In that case, we made that function abstract so that abstraction of not getting executed once could be terminated.

We made a function abstract by adding the keywood "abstract" removing its body and putting a terminator on the end.

Question:

If abstract classes cannot create objects than why ~~their~~ they have constructors ?

~~can not create~~

Abstract ~~Class~~ Class does not create objects but its child classes (concrete classes) calls its constructor implicitly in their default constructor's first as the concrete class inherits all its methods that's why its constructor exists.

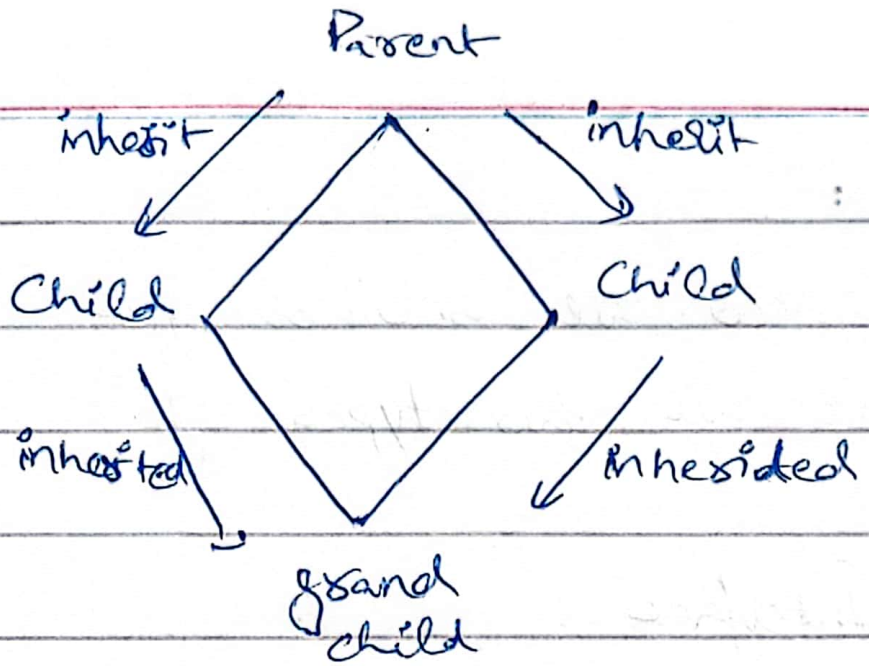
Virtual Inheritance:

Java does not provide multiple inheritance (single class expanding more than one classes).

~~Most people~~ It is wrong in general that most people say that Java does not provide multiple inheritance because of diamond problem.

But C language had already addressed Diamond Problem using virtual Inheritance.

Using Virtual word that ~~attri~~ & data member come to the concrete class once.



i.e. diamond problem

So, its not the reason for not giving multiple inheritance but it was design plan.

Interfaces:

Interfaces are another type in java. There are two types of Interfaces.

- Primary Interface
- Secondary Interface

Primary Interface

Data Members and member functions of a class are referred as its first or primary interface. As "Human" is a primary Interface of a class Human.

Secondary Interface

The word Interface is commonly refered for its secondary Interface.

As primary Interface is known as class. Secondary Interface in Java Programming language is an abstract type that is used to declare a secondary behaviour that every class must implement which implements the interface.

Interfaces are declared using the interface keyword. and may only contain method signature and constant declarations.

```
public interface ICookingExpert {

}
```

Interface must fulfill the "as-a" relation.

For Example Interface of a "Cricket Player" so Student as a Cricket Player.

Interface of a "Cooking Expert" Reckers as a Cooking Expert

So such interfaces can implement in classes respectively.

"Interfaces can Generalize different type of classes on specific seconds Interface. (abstract class)" while inheritance can only Generalize same child classes on parent's primary Interface."

Also Interface have no constructs, no data member. while abstract class have.

[redacted]

Two classes can only be related to each other in only two ways (any two classes can have only two relations.)

- Inheritance (page # 107)
- Association

Association:

Association in Java is a connection or relation between two separate classes that are set up through their object. i.e. Object of one class present in another class.

There are two types of Association.

(i) Composition

(ii) Aggregation

There are two school thoughts on Composition and Aggregation.

Composition	Aggregation
1st:-	
• Mandatory Class	• Optional Class
2nd :-	
• Only one class as a data member.	• Multiple classes as data members.

2nd school of thought is much valid.

Inner Classes:

When a class is created in another class it is called Inner Class. ~~or member class~~.

These are four types of Inner Classes.

(i) Inner Member Class

(ii) Inner Local Class

(iii) Inner Anonymous Class

(iv) Inner Static Class

Inner Member Class:

Any class created in another class as a data member then it will be called "Inner Member Class" and it can access all others members too.

Inner Local Class :

Any class created inside the member function of another class is called "Inner Local Class" and it can only access the local members of that function.

Inner Anonymous Class :

It is an inner class without a name and for which only a single object is created. An "Inner Anonymous Class" is useful when making an instance of an objects with certain "extras" such as overriding methods of a class or interface without actually creating a child class or a class which

implements the ~~method~~ interface respectively.

Inner Static Class:

A static class is a class that is created inside a class is called "Inner Static Class". It cannot access non-static data members and methods. It can be accessed by outer class name.

HERO NOTES
MONAWAR SONS LAHORE

~~Name & Address~~ ~~Office~~ ~~Fax~~ ~~Residence~~ ~~Mobile~~

"COURSE OUTLINE"

collections

Advance OOP

These are many more generic classes as collection but these are important ones.