Practice Test Questions:

(1) Rohit wants to add the last digits of two given numbers.
For Example,
If the given numbers are 267 and 154, the output should be 11.
Below is the Explanation:
Last digit of the 267 is 7
Last digit of the 154 is 4
Sum of 7 and 4 = 11
Write a program to help Rohit achieve this for any given two numbers.
Note: The sign of the input numbers should be ignored.
i.e;
If the input numbers are 267 and 154, the sum of last two digits should be 11.
If the input numbers are 267 and -154, the sum of the last two digits should be 11
If the input numbers are -267 and 154, the sum of the last two digits should be 11
If the input numbers are -267 and -154, the sum of the last two digits should be 4

For Example:

Input	Result
267 154	11
267 -154	11

(2) Tak
Patt
If y
follo

Inpu
Tak

Outp
If th
samp

Exam
aba

Outp
a
ad
ad
ad
ad

Exa
ab

Outp
No

(3) Writ
the

Com

(2) Take a string, your task is to print the Pattern as shown in sample output. It prints it from the middle of the word such that it follows the following pattern as in sample output.

Input format:
Take a string from stdin.

Output format:
If the string length is odd, print the pattern as in sample output. Else print No Pattern.

Example Input:
abada

Output:
a
ad
ada
adaa
adaab

Example Input:
ab

Output:
NO PATTERN

(3) Given two strings S1 and S2, remove all the characters from S1 which is present in S2.

Constraints
1 <= String length <= 200

Sample Input 1
experience
enc

Sample output 1
xperi

④ Find if a String s is substring of string t if it is, returns the index of the first occurrence else returns -1

Sample Input 1:
thistest 123 string
123

Sample output 1
8

* Sample answer to add last Digits:

```
Public class AddlastDigits
{
    Public static int addlastDigits (int input1,
                                     int input2)
    {
        return Math.abs (input1 %.10) +
               Math.abs (input2 %.10);
    }
    public static void main (String[] args)
    {
        System.out.println (addlastDigits (123,23));
    }
}
```

Answer: Java

1)
```
import java.util.*;
Public class AddlastDigits
{
    Public static int addlastDigits (int input1, int
                                     input2)
    {
        return Math.abs (input1%10) +
               Math.abs (input2%10);
    }
    Public static void main (String[] args)
    {
        Scanner sc = new Scanner (System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();
        System.out.println(addlastDigits (a,b));
    }
}
```

4)
```
import java.util.*;
Public class CFG
{
    static int isSubstring (String s1, String s2)
    {
        int a = s1.length();
        int b = s2.length();
        for (int i=0; i<=a-b; i++)
        {
            int j;
            for (j=0; j<b; j++)
            {
                if (s1.charAt(i+j) != s2.charAt(j))
                    break;
            }
```

```
            if (j==b)
                return i;
        }
        return -1;
    }
    Public static void main (String args[])
    {
        Scanner sc = new Scanner (System.in);
        String s1 = sc.next();
        String s2 = sc.next();
        int res = isSubString (s1, s2);
        System.out.println(res);
    }
}
```

(3)
```
import java.util.*;
Public class INFO
{
    static final int NO OF CHARS = 256;
    static int [] getCharCountArray (String str)
    {
        int count [] = new int [NO OF CHARS];
        for (int i=0; i< str.length(); i++)
            count [str.charAt(i)]++;
        return count;
    }
    static String removeDirtyChars (String s
                                    String mask
    {
        int count [] = getCharCountArray (mask
        int ip_ind = 0, res_ind = 0;
        char arr [] = str.toCharArray();
        while (ip_ind != arr.length)
        {
```

```
        char temp = arr[ip_ind];
        if (count[temp] != 0)
        {
            arr[res_ind] = arr[ip_ind];
            res_ind++;
        }
        ip_ind++;
    }
    str = new String(arr);
    return str.substring(0, res_ind);
}
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    String s1 = sc.next();
    String s2 = sc.next();
    System.out.println(removeDirtyChars(s1, s2));
}
}
```

Author Van Hoff — strictly comply with specifications (Target to Java)

06/23

Theory: 1

* Java support compiler and interpreter.
* Purely object oriented.

(*) In C depends on processor size of data values.
~~not change in Java~~

① Architectural neutral
② Platform Independant

These two reasons why we use Java.

→ Java was developed by 5 members team
James Gosling

→ Named Oak (1995)

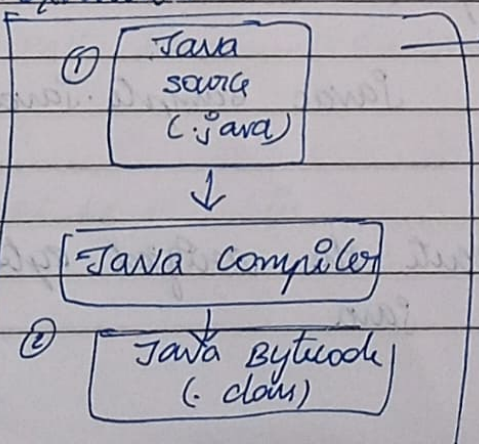
Green Team initiated in 1991
Java 1.0 in 1996 (first public implementation)

Design Goal:
→ Architectural neutral
→ Platform Independant

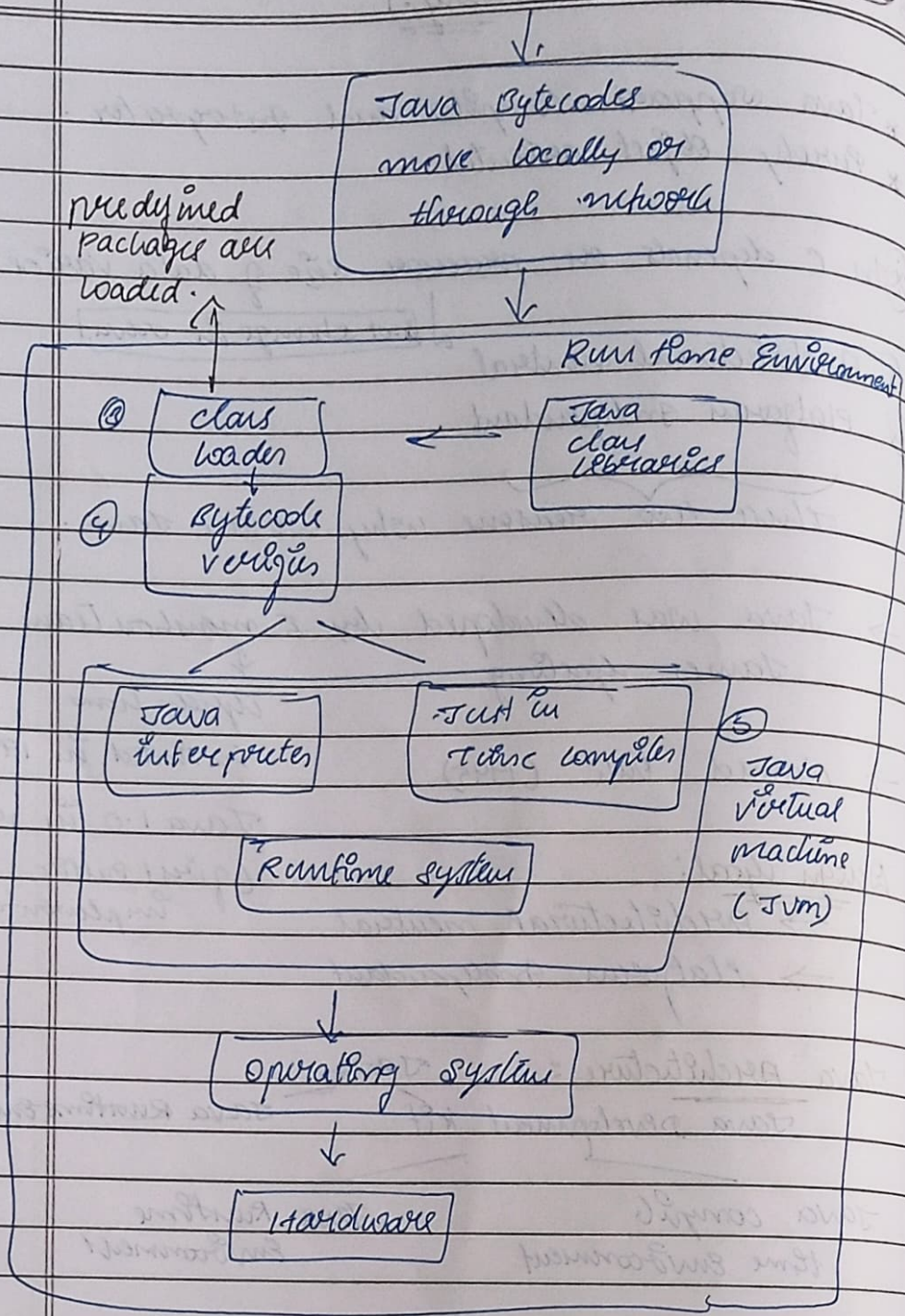
Java Architecture:
Java Development Kit
Java Runtime Env.

Java compile time Environment
Java Runtime Environment

① Type the code in any editor and save the file as .java extension



import java.lang.*; → default package



ex 3

To compile the source code:

javac sample.java

To Execute the verified bytecode:

java

[Always use CamelCase type for class name]

Simple Java code:

variable name (can be anything)

```
class Student {
    public static void main(String args[]) {
        System.out.println("Welcome");
    }
}
```

Java

- Static method → called by a class itself
- Non-static method → used an object / instance

`System.out.println` — `System`: class, `out`: implicit object, `println`: non-static method; `ln` — goes new line

→ Constant string written with double quotes.
→ Statement end with semicolon.
↓
compile time error if it is not followed.

(x) class name and filename can / or not be same
because during compilation filename is used
— execution classname is used.

If more than one class present, that no. of bytecode will be created.

(x) Copy the Java Path to connect the source code and Java compiler.

The path should be till bin.

(X) Always main class is executed

Drive — properties — Advanced system settings — Path — Edit the Path

To set the Path temporarily:

Root user > cd Desktop
" > cd Desktop > set Path = "Path"
" " " > javac Sample.java
" " " > java Student

O/P

Security

Public Private default

JVM: — It is platform specific used to make java to be platform independent

— O/P is bytecode

* Language Basics:

Data

fixed changing (variable)

↓ ↓

constants variables

Command line arguments — During execution if input is given.

(X) .length — property [counts the input]
↓ count the ip
binded with args

(X) Path, classpath to set the path permanently

ParseInt — To convert string to integer

classname.ParseInt(args);
⇓
Integer.ParseInt(args[0]);

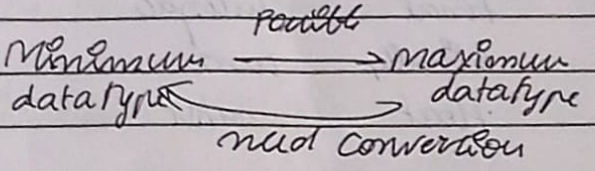
(X) If command line input is not given its length value is 0.

↳ int

0.0f — float

Fractional value is always a double, therefore to consider it a float use 'f' with

0.0 — double
0.0f — float



Good Programming Practice:

int n1, n2; X

int n1; ✓ good for readability
int n2; no error will be produced at line 1.

* Pascal case:
StudentInt
* Camel case:
studentInt

// - Single command line
/* */ - multiple command line

* Java API: frameworks

Multithreading — multiple activities at a time. [multitasking is achieved]

* Keywords:

abstract	continue	for
assert	default	goto
boolean	do	if
break	double	implements
byte	else	import
case	enum	instanceof
catch	extends	int
char	final	interface
class	finally	long
const	float	native

* Primitive data types:

	default value	size (in bits)
byte	0	8
short	0	16
int	0	32
long	0L	64
float	0.0f	32
double	0.0d / 0.0	64
char	'\u0000'	16
boolean	false	1

(X) To print more than one argument, the separator is '+' System.out.println("Hello"+"Name");

Example:

```
class Test{
    public static void main(String[] args){
    int x;
    System.out.println(x);
    }
}
```

→ Compile time error because it violates the rule declared & initialize

* Types of Variable

* Local variable — Tied to a method
* Instance variable — Tied to an object (Non-static)
* Global variable — Tied to class (static variable)

Shared by all instances of a class

* Operators: ① Arithmetic operators — modulo (depends on first operand)
② Unary operators (+A, -A, ++A or A++, --A or A--)
③ Relational operators (comparison)

Ex: 35%.6 = 5
-35%.6 = -5
35%.-6 = 5

Comparing Strings:

If we use '==' in strings it will check only the memory is same

(X) to check the value .equals method is used.

Precedence: Arithmetic > Relational > logical
(=, !=, >=, <=) > (>, <)

(4) logical operators — && , ||

↓

short circuit operator — does not evaluate second operand if first operand is sufficient to determine the value of the expression

(5) Simple Assignment operator
(6) Bitwise operator
- Bitwise AND, OR, XOR
shift operators << and >>

Bitwise AND - &
" OR - |
XOR - ^

negative value — sign bit - 1
value - 2's complement

>> or << - sign bits are maintained
>>> or <<< - sign bits are not maintained

(1) If i is an int and s is a short, how do you assign i to s?

b.s = (short) i;

(2) Why is main method special in Java program?

Notes: [Java for Java]

* Not a pure object oriented as it provides support for primitive data types (like int, char)
* Java does not provide low-level programming functionalities like pointers / operator overloading
* Always contains class and objects.
* used in all kinds of applications as
  - mobile Application
  - web Application
  - client - server Application
  - Enterprise Application

* Compared to C++, Java is more maintainable because
  - non-primitives are always references. we cannot pass large objects to functions as in C++, we always pass as reference.
  - no pointers, therefore bad memory access is not possible.

* Faster than Python code execution, Slower than C++ code Execution.

* Java does static type checking where Python does not.

↓

process of verifying the type safety of a program based on analysis of a program's text.

— type checking at compile time.

Java Terminology

① Garbage collector: In Java, programmer can't delete the objects.
To delete or recollect that memory JVM has a program called Garbage Collector — collect objects that are not referenced.

② ClassPath: file path where java runtime and java compiler look for .class file to load.

* Four main concepts of OOPs are:
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism.

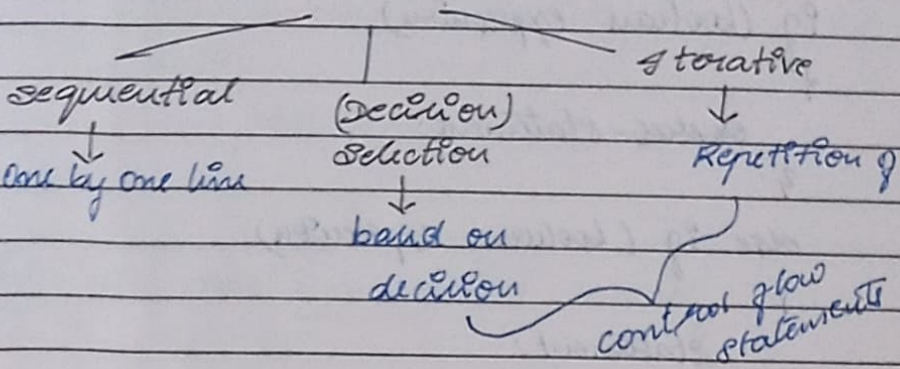
* The main features of java that make it robust are garbage collection, Exception Handling and memory allocation.

* Array out of bound, buffer overflow, etc corruption are not possible.

10/6/23

Theory - 2

* Flow control statements:



There varieties:

1. Selection — if, if-else, switch
2. Iteration — while, for, do-while
3. Jumping — break, continue, goto return

↓

moving from one line to another line without any constraint.

Selection:

1. Simple if —

Syntax:
```
if (boolean expression)
{
    statement;
}
Next statement
```

2. if-else —

Syntax
```
if (boolean expression)
{
    statement;
}
else
{
    statement;
}
next statement;
```

③ for else-if ladder:

```
if (boolean expression)
{
    statement;
}
else if (boolean expression)
{
    statement;
}
else
{
    statement;
}
next statement;
```

④ Nested-if :

```
if (boolean expression)
{
    if (boolean expression)
    {
        statement;
    }
}
else
{
    statement;
}
next statement;
```

⑤ Switch : — multipath selector

```
switch (expression)
{
```

nextInt → Integer next → String nextLine → Paragraph nextDouble → double nextFloat → float

case 1:
    block statement;
    break;
default:
    statement;
}

(X) To create an object name

classname objectname = new classname();

To use scanner to read —

import java.util.scanner;

or

import java.util.*;

Input Mismatch Exception:
↳ if the datatype of the value changes this would be the default error.

* Iteration:

(1) while — Entry control loop

while (condition)
{
    Body of the loop
}

(2) do while —

do
{
    Body of the loop
} while (boolean expression);

③ for loop:

```
for(initialization; condition; increment/decrement)
{
    Body of the loop;
}
```

enhanced for loop: — to pick one by one element from an array

```
for(declaration : expression)
{
    Body of the loop
}
```

Example:

```
Public class sample{
    Public static void main(String [] args)
    {
        int [] numbers = {10, 20, 30, 40, 50};
        for(int i : numbers)
        {
            System.out.Println("i: "+i);
        }
    }
}
```

⊗ # Second Part should be containerized data [group of values] should not be a single value if single value is given, it will throw an error.

Break : — Terminate the iteration or switch case block.

If break statement is used in normal selection statement it will throw an error.

Continue : — used only in looping statement

If used in ~~normal~~ selection statement throws an error.

Return : used in function

Good Programming Practice:

1. Always use {} for if statement.
2. One declaration per line
3. Avoid array and variable declaration in same line.

Situation:

1. Read a value from user and check odd or even.
   if ... else
2. ~~Grade~~ Value calculation
   else-if ladder / switch / Nested if

Theory - 3

Array

↳ class Demo

{

public static void main (String args[])

{

int num[] = new int[6];

```
int x[];
x = new int[10];

for (int i : num)
{
    System.out.println(i);
}
```

Single statement initialization:

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

Array

(1.) Resizing is not possible in Java produce compile time error if related.

Both are same

```
int[] x = {10,20,30};
int[] x = new int[] {10,20,30};
```

(2.) length is a property to count the number of array.

To copy this array:

```
Public static void arraycopy(Object s, int sIndex, Object d, int dIndex, int length)
```

Example:

```
System.arraycopy(source, 0, dest, 0, source.length);
```

In C, in 2-D array row size is optional and column size is mandatory.

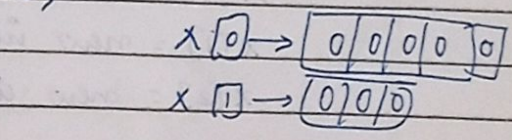
Two Dimensional Arrays:

In Java row size is mandatory and column is optional.

```
int [][] y = new int [3][3];
int [][] y = { {1,2,5}, {4,5,6}, {7,1,9} };
int [][] y = new int [][] {1,2,3,4,5,6,7,8,9};
```

The length of column can vary for each row and initialize number of columns in each row.

```
int [][] x = new int [2][];
x[0] = new int [5];
x[1] = new int [3];
      (or)
int [][] x = new int [2][];
x[0] = {1,2,3};
x[1] = {1,2};
```



Array out of Bound — when the row is not created

null — when row is created but it's columns is not created and when the user try to print the row.

null exception — when null condition and user try to get it's column value.

Example:

```
class Demo {
    public static void main(String args
    {
        int x[][] = new int[5][];
        System.out.println(x[0]);  // null
        System.out.println(x[0][0]); // Null
    }
}
```

Example:

```
int [][] x = new int[3][];
x[0] = new int[] {0,1,2,3};
x[1] = new int[] {0,1,2};
x[2] = new int[] {0,1,2,3,4};
```

Example:

```
import java.util.*;
public class SumOfArray {

    public static void main(String [] args)
    {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int array[][] = new int[n][];
        int sum = 0;
        for(int i=0; i<n; i++)
        {
            int col = sc.nextInt();
            for(int j=0; j<col; j++)
            {
                array[i][j] = sc.nextInt();
                sum += array[i][j];
            }
        }
        System.out.println(sum);
```

// 1D

```
import java.util.Scanner;
class Demo
{
    public static void main (String args[])
    {
        Scanner sc = new Scanner (System.in);
        int n= sc.nextInt();
        int sum = 0;
        int x[] = new int [n];

        for( int i=0 ; i<n; i++)
        {       x[i] = sc.nextInt();
                sum+ = x[i];
        }
        System.out.println(sum);
    }
}
```

Matrix Addition:

```
import java.util.Scanner;
class Demo
{
    public static void main (String args[])
    {
        Scanner sc = new Scanner (System.in);
        int n = sc.nextInt(); int n = sc.nextInt();
        int array[][] = new int [n][];
        int b[][] = new int [n][];
        int sum[] = new int [n];
        for (int i=0; i<n; i++) {
            int col = sc.nextInt();
            for (int j=0; j<col; j++)
```

```
{

Scanner sc = new Scanner(System.in);
int n = sc.nextInt(); int m = sc.nextInt();
int array[][] = new int[n][m];
int array2[][] = new int[n][m];
int result[][] = new int[n][m];

for(int i=0; i<n; i++)
    for(int j=0; j<m; j++)
        array[i][j] = sc.nextInt();

for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        array2[i][j] = sc.nextInt();

for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        result[i][j] = array[i][j] +
                       array2[i][j];

for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        System.out.print(result[i][j]+" ");
    System.out.println();

}
}
```

Theory - 04

Classes and Objects

↳ group of objects

related activity = behaviours
variables - class variable / class instance

4 varieties of Access specifier:

- public → anywhere
- protected → Restricted (same class and the derived class)
- private → within the class
- default → within the folder

→ method declaration syntax:

```
return type method name (arguments)
{

}
```

Example:

To collect student information

```
class Student {
    int roll_no;
    String name;

    void getDetails (int r, String n)
    {
        rollno = r;
        name = n;
    }
    void display()
    {
        System.out.println(rollno + " " + name);
    }
}
```

```
class Demo {
    Public static void main (String[]
    {
        Student s = new student ();
                    ↓
                //to specify
                null val
        s.getDetails (101, "AAA");
        s.Display ();
    }
}
```

To declare static method:

```
static void m1 ()
{
    System.out.println ("Hello");
}
```

To call m1() within the class where it is created to call

— m1(); // Just call the name.

But to call the class outside the class

Student.m1();

Classname.methodname

To Pass object as argument:

```
void getDetails (int n, String n, Student
{
    rollno = n
    name = n
    s.showmethod ();
}
```

(X) Static — memory is same maintained

Variable is static:

```
Static int rollno;  // declaration

Student.rollno ;    // call variable
```

Ex:

```
class Student
{
    static int x;
    int y;
    void getDetails (int a, int b)
    {
        x=a;
        y=b;
    }
    void display ()
    {
        System.out.println(x+" "+y);
    }
}

class Demo{
    Public static void main (String args[])
    {
        Student s = new Student ();
        s.getDetails (10,20);
        s.display ();
        Student s1 = new Student ();
        s1.getDetails (100, 200);
        s1.display ();
        s.display ();
    }
}
```

O/P: 10 20 — last updated values is maintained bq it is static.
100 200
100 200

// To differentiate class variable and argument.

```
class Student
{
    int x;
    int y;
    void getDetails(int x, int y)
    {
        this.x = x;
        y = b;
    }
```

(X)

this — cannot be used for static variable.
this cannot be used in static method
↓
represent class variable.

constructor: — Special method
- doesn't contain return type
- used to initialize object
- if name is same name as classname.

— creation of object automatically invoke the constructor

Example:

Default constructor:

```
class Student {
    int x;
    int y;

    Student() {          // default constructor
        x = 20;
        y = 10;
    }
```

```
Student (int a, int b)
{                                   // Parameterized
    x = a;                          constructor
    y = b;
}

Student (int a)
{
    x = a;
    y = 0;
}
}

Public class Demo2
    Public static void main (String ar[])
    {
        Student s = new Student ();
        s.display();
        Student s1 = new Student (100, 200);
        s1. display();
        Student s2 = new Student (1);
        s2. display ();
    }
}

// this keyword for method

Student ()
{
    this ("REC");   ---- calls the constructor with single String parameter.
    x = 0;
    y = 1;
}
student (string s
{
    x = a;
    y = b; system.out.println(x);
```

(X) Static block – Executed first before the main function for pre-processing.

```
static {
    System.out.println("Static block");
}
```

05/06/23

Theory – 06

String and String Buffer / Builder

Immutable — mutable

```
String s = new String(); // Empty String
String s = "";
```

String and String Buffer are declared as final means constant.

(X) → Convert non String to String.

```
int a = 10
String s = a + "" // 10
String s = (String) a;
```

To create String:

```
String str = "abc";
(OR)
char data[] = {'a', 'b', 'c'};
String str = new String(data);

String str2 = new String(str);
```

↳ Construct a String object by passing another object.

String class methods:

(1) The length() method returns the length of the string.

Ex: System.out.println("Varun".length());

(2) Concat(s)

String myName = "Varun";
String s = "my name is " + myName + ".";

(3) charAt

Public char charAt(int index)

— method returns the character at specified index. Index ranges from 0 to length() - 1.

char c;
c = "abc".charAt(1); //c = "b"

(4) equals method — compare two strings

public boolean equals(object anobject)

(5) equalsIgnoreCase() —
considers cased case

Public boolean equalsIgnoreCase(String anString)

(6) startsWith() — Test if this string starts with specified prefix.

Public boolean startsWith(String ...

"January".startsWith("Jan")

(7) endsWith() — used to check if the given string ends with

(8) compareTo() — which is bigger or smaller

0 — equal
1 — String1 > String2
-1 — String1 < String2

Public int compareTo(String anotherString)
Public int compareToIgnoreCase(String anotherString)

Ex: s1.compareTo(s2);

(9) IndexOf — search for first occurrence of a character or substring

Returns -1 if the character does not occur

(10) Public int indexOf(char)
↳ search for a character

Public int indexOf(String)
↳ search for a String

String str = "How are you doing today";
str.indexOf('t')
str("are")
str.indexOf("are");

(8) To getch last occurrence

str.lastIndexOf('a');

Example:

stava.lastIndexOf("a");

(9) To getch middle occurrence

public int IndexOf (char, int fromIndex);

public int IndexOf (String, int fromIndex);

Example:

String str = "How was your day today?";
str.IndexOf('a', 6);
str.IndexOf("was", 2);

↓

returns 'w' character address

(10) ~~Substring()~~ subString() subString();

public String subString (int beginIndex)

Eg: "unhappy".subString(2) returns "happy"

public String subString (int beginIndex,
int endIndex)

(11) subStr();

str.subStr(int beginIndex, no.of characters);

Str.contains(substr) — To check whether the substring is present or not

(12) concat() —

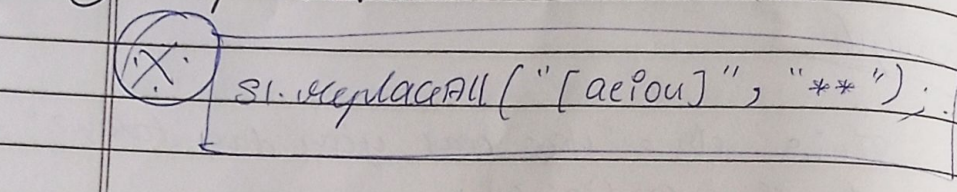
str.concat(anotherstring)

(13) Replace() —

str.replace(char, char)

Public String replace(char oldchar, char newchar)

(14) replaceAll() —



(15) trim() — Returns a copy of the string with leading and trailing whitespace omitted.

Public String trim()

String s = " Hi Mom! ".trim();

s: "Hi Mom!"?

(16) valueOf() — convert a character array into to string.

Public static String valueOf(char array)

(17) toLowerCase() — String to Lowercase

toUpperCase() — " " Uppercase

str.toLowerCase();

str.toUpperCase();

String Buffer:

S = "CSE ECE ECE BNG"

↳ when used with split method it returns String array with default delimiter is the whitespace

```
String s[] = s.split();
(or)
String s[] = s.split(" ");  [ignoring the delimiter].
```

String Buffer:

```
StringBuffer()
StringBuffer(int)
```

main
```
StringBuffer insert(int index, String str)
StringBuffer append(int index, String ch)
```

```
StringBuffer append(String str);
StringBuffer append(char ch);
```

(2) delete a specific substring

```
Public StringBuffer delete(int start, int end)
```

(3) replace() - replace(start, end, String 2)

(4) substring()

(5) reverse() - StringBuffer reverse()

(6) length()

Problems:

(1) Count the number of lowercase and Uppercase alphabets in a given input.

```
import java.util.*;
class Demo
{
    public static void main(String arg[])
    {
        Scanner s = new Scanner(System.in);
        String str = s.next();
        int lc=0; int uc = 0;
        int n = str.length();
        for(int i=0; i<n; i++)
        {
            char ch = str.charAt(i);
            if(ch>='a' && ch<='z')
                lc++;
            else if(ch>='A' && ch<='Z')
                uc++;
        }
```

for weighted case

```
if(ch>='a' && ch<='z')
    wt = ch - 96;
```

Theory - 07

Inheritance

Existed class — Super class / Parent class
Newly created class — Sub class / child class

* Association, Aggregation and Composition:

```
class A {
    int x, y;
    A() {
        x = 0;
        y = 0;
    }
    A(int p, int q) {
        x = p;
        y = q;
    }
    void display() {
        System.out.println(x + " " + y);
    }
}

class B extends A {
    int z;          // To inherit
    B() {
        super();    // default
        z = 100;
    }               // super (x);
    B(int a, int b, int
        super(a, b);
        z = c;
    }
    void display() {
        System.out.println(z);
    }
    void sum() {
        System.out.println("sum = " + (x+y+z));
    }
}

class Demo {
    public static void main(String args[])
    {
        B b = new B();
```

System.out.println("main");
b.display();
b.display1();
b.sum();

— First line of the constructor be this() or super()

← default super()

Association

↓ ← relationship between objects.

- one - one
- one - many
- many - one
- many - many

Aggregation

↓

is a part of

(has a)

Composition

↓

is a part both should be present.

[Restriction among a Aggregation

(contains)

Eg (obj instanceof Doctor)

// Valid Statement

else if (obj instanceof Manager)

{ // Statement; }

else

// Statement;

Theory - 08

Wrapper class — Associate class of primitive datatypes.

↳ All collection classes in Java is object

Object representing in Primitive datatype

(X) Converting primitive data types into objects is called wrapping.

```
int i = 10;
Integer ive = new Integer i;
```

(X) Performance in terms of memory and speed

Representing in 12-16 bytes, compared to 4 in an actual integer.

Retrieving a value of an integer use the method Integer.intValue().

— enable manipulate primitive data type

String str = "100";
int i = Integer.parseInt(str);

Constants:

1. MAX_VALUE — $2^{31}-1$
2. MIN_VALUE — -2^{31}
3. NaN
4. POSITIVE_INFINITY
5. NEGATIVE_INFINITY

* Integer class:

int - intValue()

constructor for Integer:

Integer(anum);
Integer(String); // Error.

Public class Sample {

Public static void main(String[] a

{

Integer i = new Integer(100);
System.out.println(i);
System.out.println(i.intValue

}

}

byteValue(), doubleValue(), floatValue(),
ShortValue() longValue().

Character class:

Character (char c) — Constructor

char charValue() — getting character value.

① MAX_VALUE — largest character value
② MIN_VALUE — smallest value
③ type —

character.toUppercase

Boolean class:

— Boolean (boolean bValue)
(or)
Boolean (String str)

if str = "true" or str = "false" it can be read either in upper or lowercase.

Float class Long class

Double class byte.valueOf() ByteValue()

— byte b = 12;
Integer i1 = (int) b;

* I/O streams: byte stream — Binary files
character stream — Text files

— an abstraction that either produces or read.

multimedia content
— Binary file.

— use Unicode and internationalized in character streams

Predefined Streams

— System.out — console
— System.in — terminal
— System.err — console
↳ To display error messages.

System Properties

— getProperties()
— getProperties(String key)
— setProperties(Properties prop)

java.version
java.home

```
Properties x:
    System.getProperties()
    x.list(System.out);
```

File, Buffer, object } Binary data

File, Buffer, Input/ } character stream

Serialization — [object → file]

InputStreamReader & OutputStreamReader — Bridge between Byte stream and character stream

// Either input stream or output stream not as both.

BufferedReader(Reader inputReader)

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

① ~~X~~ br.read() — to read one character [1 byte of data]

return -1 [no input]

② br.readLine() — Read a line of string

Theory - 09

Method overriding / Polymorphism

* Compile time
  - operator overloading / overloading
  - Function overloading
  - Ex: (+)

Run-time
- Overriding
- Override the same signature method in subclass.

Example:

```
class Sample {
    int sum(int a, int b)
    {
        return a+b;
    }
    int sum(int a, int b, int c)
    {
        return a+b+c;
    }
    float sum(float a, float b)
    {
        return a+b;
    }
}
```

→ Overriding happens only when inheritance occurs.

```
class A1 {
    A1()
    {  System.out.println("A1 is no arg constructor");
    }
    A1(int a)
    {
        System.out.println("A1 is constructor");
    }
}
```

Final is related to constants:

Extends class — Overall Super class

Variable as a final — works like a constant

method as a final — cannot be overridden

class as a final — cannot be inherited.

→ Variable when declared as a final either initialized while declaration or using constructor.

Abstract class

↳ Security can be provided (0-50%) in Java for methods alone.

— It will not contain any implementation.
↓
occurs in subclass.

← A class which has abstract method is considered to be abstract class.

Theory - 10

Introduction to Interfaces

Inheritance:
1. Single Inheritance
2. multilevel Inheritance
3. Hierarchical Inheritance
4. multiple Inheritance (A) (B)
5. Hybrid Inheritance

(X) Multiple Inheritance is not supported in Java.

To achieve multiple Inheritance → Interface is required.

We can create Interface from an Interface and a ~~class~~
~~but couldn't create class from Interface~~
but not an Interface from a class.

Keyword — implements (to create class from an Interface)

* Interface members:

Interface InterfaceName {

Body }

class ClassName implements InterfaceName {

Body
}

(1) — under Interface — all methods are always by default public and abstract.

(2) — a variable under Interface is always by default, public static and final.

(3) Cannot create instance for Interface.

Example:

```
public Interface FirstInterface {
    int addition1(int x, int y);
    float division1(int m, int n);
    void display();
    int VAR1 = 10;
    float VAR2 = 20.65;
}
```

Example:

```
class myClass implements FirstInterface {
    public int addition1(int a, int b)
    {
        return (a+b);
    }
    public float division1(int i, int j)
    {
        return (i/j);
    }
    public void display()
    {
        System.out.println("Variable 1 : "+VAR1);
        System.out.println("Variable 2 : "+VAR2);
    }
}
```

MyClass m = new MyClass();
m.add Method();

Marker Interface

An Interface without no method

(x) — Interface will not have Constructor method

* Package — folders

Theory - 11

import static java.lang.math.*;
↳ for math Package.

— In c if there is an error in line 5 it will come to end of file.

— In Java that line is handled separately

Predefined — Exceptions are present

Ex: Always ArrayIndexOutOfBoundException.

User defined —

1. Try
2. catch
3. throw

(X) Exception is present in language

(Q) What is an Exception?

— Ex: Arrayout of Bound
— divide by 0.

Exception Handling techniques

Even error does not occur finally block executes.

Types of Exception

Throwable

Error

Exception

Checked Exception

↓

compile Java the programmer to handle Exception

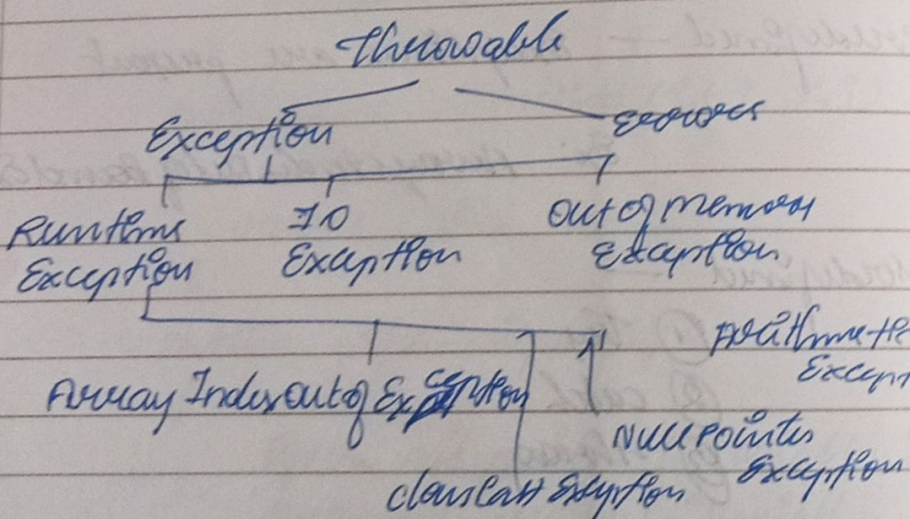
Ex: SQL Exception

Unchecked Exception

↓

Run time Exception

Ex: NULL Pointer Exception



(X) cannot float floor Division cannot be applied for float division.

INull Pointer
Array Index out of Bounds Exception
Number format Exception
class cast Exception
negative array size Exception
} Runtime error

class Not found Exception
IO Exception
Interrupted Exception
} compile time error.

(X) once we complete try block start with catch block.

——— can have n number of catch block.

/child class — specific Exceptions

throws clause

e.getmessage() — Instead of System.out. println

Theory - 12

Marked interface — Interface with no members and methods

↓

Special permissions during Runtime.

(X) Object class — Parent class of all the classes in Java

↓

- toString()
- hashCode()
- getClass()
- clone() — copy the value into different object

↓

Example of interface.

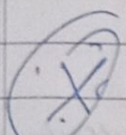
clone()
↳ is a protected method of Object class
↳ can be only implemented by cloneable interface.

CloneNotSupported

Example on cloning:

```
class xyz implements cloneable {
    int a;
    int b;
    double c;

xyz cloneTest() {
    try {
        return (xyz) super.clone();
    }
    catch (cloneNotSupportedException e) {
        System.out.println("Cloning not allowed");
        return this; }
```



```
x2 = x1.cloneTest()   // cloning x1 [Diff copies]
x2 = x1               // copy x1 [same copy]
```

Example:

```
class cloneDemo1 {
    Public static void main (String [] args)
```

Serialization: Process of saving an objects state to a sequence of bytes as well as the process of rebuilding those bytes into a live object.

Serialize the objects of a class that implements Serializable interface

(7) Serialization and Deserialization
& Java Independent.

Object Input Stream Object Output Stream

Serializable objects:

How to write to an object output stream

FileOutputStream out = new FileOutputStream("the file");

ObjectOutputStream os = new ObjectOutputStream(out);

os.writeObject("today");
os.writeObject(new Date());
os.flush(); // just in case.

How to write/Read from an object output stream

FileInputStream in = new FileInputStream("the file");

ObjectInputStream s = new ObjectInputStream(in);

String today = (String) s.readObject();
Date date = (Date) s.readObject();

transient — is used in object serialization

transient int a;
static int b;
↳ the variable static will not be stored in serialization.

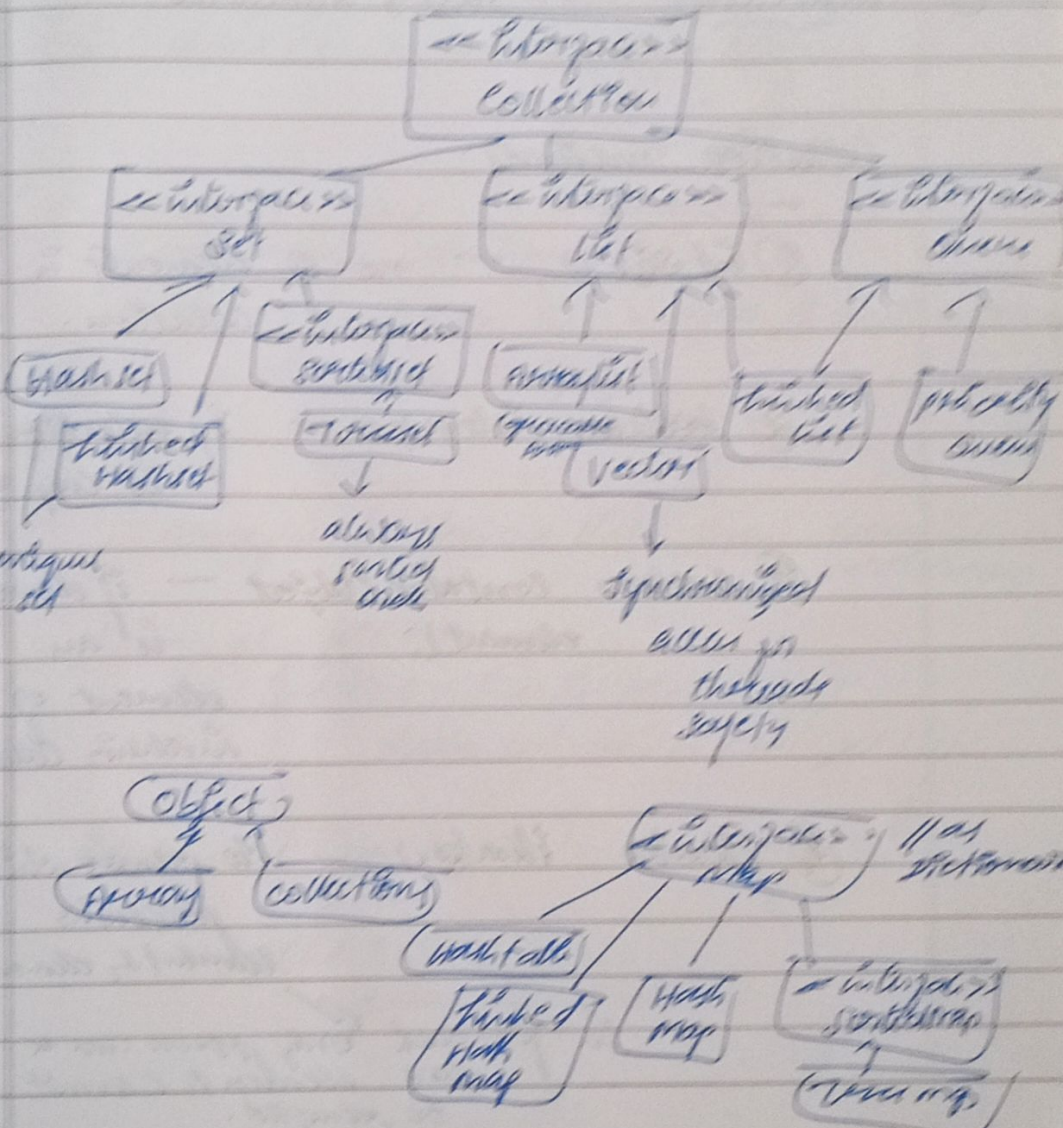
Collections

↳ group of objects.

↳ collection framework has three parts

- core interfaces
- general implementation
- algorithms such as searching, sorting

Advantages
* Reduces programming effort
* Increases performance
* set of API's



① In Hashset —
Value can be duplicated but set can should be unique.

② Linked Hashset — ordered with unique set.

③ ArrayList — growable array
↓ implements
✗ Random Access interface — marked interface.

④ LinkedList — Doubly linked list.

Collection methods:

① int size() — no. of elements in collection.

② boolean isEmpty() — if collection is empty

③ boolean contains(Object element); — if element is an element of invoking collection

④ Iterator iterator() — to remove while processing elements by element

in for-each only process can be done meaning it cannot be removed.

⑤ boolean containsAll(Collection c);
Returns true if invoking collection contains all elements of c

⑥ boolean addAll(Collection c);
Add all elements to c.

⑦ boolean removeAll(Collection c);

⑧ boolean retainAll(...);
↳ object only retained common everything else.

⑨ void clear();

⑩ object[] toArray();

⑪ Object[] toArray(Object a[]);

* Arraylist class
import java.util.Arraylist;
Arraylist<Integer> list = new Arraylist<Integer>();
list.add(0, new Integer)
store integer

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

// normal for loop
for(int i=0; i<list.size(); i++)
System.out.println(list.get(i));

// Enhanced for loop
for(int i : list)
System.out.println(i);

// using Iterator

Iterator list = list.iterator();
while(list.hasNext())
System.out.println(list.next());

Iterator() — boolean hasNext();
↓
void remove();
↓
e.next();
↓
returns next element in the data type.

List Iterator — previous() and next() method to present to access.

Linked List:

void addFirst(object x)
void addLast(object x)
object getFirst()
void getLast()
object removeFirst()
Object removeLast().

Vector: Synchronized for new thread safety

Vector v1 = new Vector();

To sort ArrayList :

```
Collections.sort(list);
```

Generics

Before Generics — type is not specified

```
List myIntegerList = new LinkedList();
myIntegerList.add = (new Integer(0));
Integer x = (Integer) myIntegerList.iterator().next
```

Ex:

```
List<Integer> myIntegerList = new LinkedList
myIntegerList.add = new Integer(0).<Integer>()
Integer x = myIntegerList.iterator().next();
```

Theory - 13

* Set
* Has map

* Set: — unordered collection.

TreeSet | HashSet — Linked HashSet

Use HashTable for storage

TreeSet — string - alphabetical order.

CompareTo()
Comparator — to specify the sorting order of Integer.

Comparable — java.lang()
Comparator — java.utils

```
class ByName implements Comparator
                        <Student>
{
    Public int compare(Student 01,
                       Student 02)
    {
        return 02.name.compareTo(01.name);
    }
}
```

Map

Key - value pairs (key type : value type)

put() — for adding elements
get() — for accessing an element
remove() —

- TreeMap —