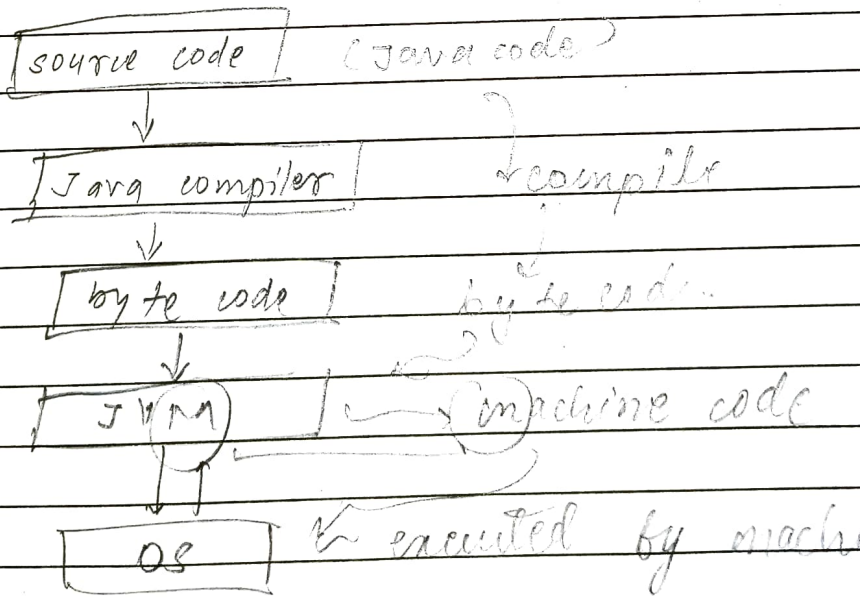
Introduction to java

Java architecture and feature → JVM, JRE, JDK

- Java architecture is a collection of components i.e. JVM (Java virtual machine), JRE (Java runtime environment), JDK (Java development kit).
- It integrates the process of interpretation and compilation. It defines all the process involved in creating java program → interpretation + compilation. JDK, JRE, JVM
- Java architecture explains each and every ~~thepter~~ step of how a program is compiled and executed.

This can be explained by following :-

- There is a process of compilation and interpretation of program in java
- Java compiler convert java code → byte code
- After this JVM convert byte code → machine code.
- Then machine code is executed by machine



Prabodh
PAGE No.:
DATE:

components of java architecture

→ JVM Byte to machine
→ JRE
→ JDK

JVM (Java Virtual Machine)

- main feature of Java is WORA. (Write Once Run Any where).
- The feature states that we can write our code once and use it anywhere or in any OS.
- This can only be done by JVM.
- Main task of JVM is to convert byte code → machine code.

package name; → capital.

```
package com.company;
default public class Main {
function → public static void main(String[] args) {
            // write your code here
            System.out.println("hello World");
        }
}
```

a package in Java is used to group related classes. Think of it as a folder in a file directory.

class name

return type

access modifier

help to run without help of object

Basic Structure Of Java Program.

1. Functions (group of code) — user defined data type

→ group of code to use for problem solving. like adding two numbers

→ like a plumber for plubing

2. Classes — desired data type (group of function)

→ collection of function contains many method.

→ group of worker.

naming convection

- for classes we use ~~Pa~~ PascalConvention → uppercase
  - MyNameIsAman
- for function we use camelCase convention.
  - ↳ myNameIsAman
  - lower case

// - double forword slash for comment

function →

Class →

variable and data type

variable → is like a container where we can store data
→ data storage (store same type)

datatype → int, byte, bool, long, float, double, char, short

anatomy of java program

Documentation Section → suggested set of comme t
Package statement → optional.
import statement → optional for input
interface statement → optional s'
class defination → optional

Main Method Class → essential
{
Main Method Defination;
}

Data types → two (primitive + non-primitive).

Primitive
(no.) byte → int
dec → float → long (int)
char → double (float)
→ bool → short (float)

true false

non-primitive
derived.
=

variables

a variable is given a name, such as area, age height. The name uniquely identify each variable assigning a value to the varible and retrieving the value stored.

some example —
- int for integers (whole numbers)
- double - floating, decimals.
- string : for text such as "hello" or "good morning"

example.

```
String name = "Aman";
```

declaring type of variable is important and it will only store the same type data.

Variable rule ↴
we cant start with digit int 1arr; ✗
→ case sensitive int arr;
→ shouldn't be a keyword
→ whitespace not allowed.
→ can contain, characters, - character, digit if upper are satisfied.

Data types.
primitive data
1) byte → stores integer, -128 to 127
- 1 byte (8 bit)
- Default 0. $-\frac{2^8}{2}$ to $\frac{2^8}{2}-1$

2) short
- value range from $\frac{-2^{16}}{2}$ to $\frac{2^{16}}{2} - 1$
→ 2 byte (16 bit)
→ default 0

3) int → value range from $\frac{-2^{32}}{2}$ to $\frac{2^{32}}{2} - 1$
→ 4 byte (32 bit)
→ default value 0

4. float →
→ take 4 byte (32 bit)
• default is 0.0f

5). long → $2^{64}/2$ to $2^{64} - 1$
→ take 8 byte (64 bit)

6) double → 8 byte

7 char → 0 to 65535 ($2^{16}-1$)
→ 2 byte, (because support unicode)
→ default 0.

8) boolean. → value can be true or false
→ size depends on JVM
→ default false.

Java program to add 3 numbers.

```
public class Main {
public static void main (String[] args) {

    System.out.print(" The sum of number : ");
    int num1 = 5 ;
    int num2 = 5 ;
    int num3 = 5 ;
int sum = num1 + num2 + num3;
    System.out.println(sum);
  }
}
```

~~Java basic syntax~~

~~object~~ :- operators

1. arithmetic operators (+, -, ÷, *)
2. relational operator
3. Bitwise operator
4. logical operator
5. assignment operator
6. Misc.
7. ternary

arithmetic operator :

(+) addition → adds value on either side of operands
(-) substraction → substract right hand operand from left hand operand.
(*) multiplication → multiplies values on either sides of operands.
(/) Division → divides left hand operand by right hand operand.

Relational Operator.

(==) equal to → checks if the value of two operands are equal or not.

(!=) not equal to →

(>) greater than

(<) less than

(>=) greater than or equal to

(<=) less than or equal to.

Bitwise

(&) and (|) or (^) xor

(~) compliment

(<<) left shift

(>>) right shift

(>>>) zero fill

logical operator

(&&) logical and (||) logical or

(!) logical not

Assignment Operator

(=) this assign values from right side operands to left side operand.

(+=) it adds right operand to left operand and assign value to left operand.

(-=) it substracts right operand from left operand and assing the result to left operand.

all these following are having same

(/=) (%=) (*=)

(&=) (|=)

ternary operator

this operator consist three operands.

variable x=(expression)? if true : if false;

example.

```
public class Test{

    public static void main (string args[]){
    int a, b;
    a=10;
    b = (a==1)? 20 : 30;
    System.out.println("value of b is" + b);

    b = (a==10)? 20 : 30;
    System.out.println("value of b is" + b);
    }
}
```

output: value of b is 30
value of b is 20.

Getting user input:

There are so many methods to get input in java but easier is to get (import the Scanner class to use scanner object:

```
import java.util.Scanner;
```

to use this class we create an instance of class by using following system:

```
Scanner myVar = new Scanner(System.in)
```

you can now read in different kind of input data that user enter.

some methods:

```
read a byte      - nextByte();
     short       - nextShort();
     int         - nextInt();
     long        - nextLong();
     float       - nextFloat();
     double      - nextDouble();
     boolean     - nextBoolean();
complete line    - nextLine();
     word        - next();
```

string → (sequence of characters → 'a' + 'b' + 'c' = "abc"

a string is a object that represents the sequence of characters

'characters' "string"

we use + to concatenate strings

```
string firstName, lastName;
firstName = "Aman";
lastName = "Yadav";

System.out.println("My name is" +firstName+" "+
lastName);
```

Time converter problem ↓

it takes input as day and convert to second.

1 day = 24 hour
1 hour = 60 min
1 min = 60 sec

→ 24 hour = 1440 min

1440 × 60 = 86400 second.

code ↓

```
import java.util.Scanner;
public class Program{
    public static void main(String args[]){
        Scanner scanner = new Scanner (System
.in);
        int days = scanner.nextInt();
        int seconds = (days) * (24) * (60) * (60);
        System.out.println(seconds);
    }
```

Prabodh
PAGE No.:
DATE:

Conditionals & loop.

conditional statement are used to perform different actions. based on different condition.

The if statement is one of the most frequently used conditional statement.

if the if statements condition expression evaluates to true, the block of code inside the if statement is executed. if it found to be false then set of code after closing braces executed.

Syntax

```
if (condition) {
        // executes when the condition is true
}
```

following comparison operator can be used
<, >, !=, ==, <=, >=

Example:

```
int x = 7;
if( x<42) {
        System.out.println( "Hi");
}
```

Output: Hi

if .. else statement

if statement can be followed by else statement, which executes when the if condition is false.

Example:

```
int age = 30;
if(age<16) {
        System.out.println("Too young");
}
else {
    System.out.println("welcome!");
}
```

output: welcome!

nested if statement

you can use another if else statement inside if else statement.

Example:

```
int age = 25;
if (age >0){
        System.out.println("welcome!");
} else {
    System.out.println("Too young");
    }
} else {
    system.out.println("Error");
    }
}
```

Else if statements :-

instead of using if else statements, you can use the else if statement to check multiple condition.

example :-

```
int age = 25;
if (age <= 0){
        System.out.println("Error");
}else if (age <= 16){
        System.out.println("Too young");
}else if (age <= 100){
        System.out.println("Welcome!");
}else{
        System.out.println("Really?");
}
```

logical operator

logical operator are used to combine multiple condition.

lets say you wanted your program to output "welcome!" only when the variable age is greater than 18 and the variable money is greater than 500.

There is two methods → first is nested loops and also we can use AND logical operators.

nested :-

```
if (age > 18){
        if (money > 500) {
                System.out.println("Welcome!");
    }
}
```

logical operator.

```
if (age > 18 && money > 500) {
        System.out.println("Welcome!");
}
```

OR operator (||)

The or operator (||) checks if any one of the condition is true.
The condition become true, if any one of the operand evaluates to true.

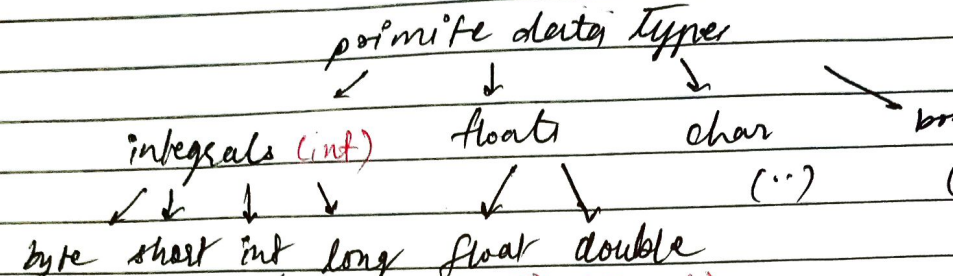
```
int age = 25;
int money = 100;
if (age > 18 || money > 500) {
    System.out.println("Welcome!");
}
```

the not operator will used to reverse above ↑

Prabodh
PAGE No. :
DATE :

CCOH

Java (literals)



To select → which data type we should ~~not~~ be aware about more and less.

Literals →

a constant value which can be assigned to the variable is called as a literal.

example : byte age = 39; → literal

Char ch = 'A'; → char

float f1 = 5.6; → by default double
to get float
= 5.6f;

double d1 = 4.860;

long aged = 5666666666L;

String S = "Aman";
↳ literal.

Keyword →
, which are reserved by java compiler.

user input

to get input java has a scanner class.

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

↳ read from the keyboard.

```
int a = s.nextInt();
```

↳ This is the method to read from keyboard (integer in this case).

```
import java.util.scanner;

public class TakingInput {
    System.out.println
    public static void main (String[] args){
        System.out.println("taking input from us
        Scanner sc = new Scanner(System.in);");
        System.out.println(" Enter no. 1 ");
        int a = sc.nextInt();
        System.out.println("enter no. 2");
        int b = sc.nextInt();
        int sum = a+b;
        System.out.println(" The sum of these
numbers is ");
        System.out.println(sum);
    }
}
```

a quick quest

write a program to calculate percentage of a given studen in exam for five sub. (out of 100)

total → for (variable).

```
import java.util.scanner;
public class Main {
    public static void main (String [] args) {
        System.out.println (" Enter total marks: ");
        Scanner s= new Scanner (System.in);
        int gt = s.nextInt();
        float marks;
        System.out.println (" enter the mark of 1st sub:");
            float a=s.nextFloat();
        System.out.println (" enter the mark of 2nd sub:");
            float b = s.nextFloat();
        System.out.println (" enter the mark of 3rd sub:");
            float c = s.nextFloat();
        System.out.println (" enter the mark of 4th sub:");
            float d = s.nextFloat();
        System.out.println (" enter the mark of 5th sub:");
            float e = s.nextFloat();
        System.out.println (" enter the mark of 6th sub:");
            float f = s.nextFloat();

                marks = a+b+c+d+e+f;
        float percentage = (marks/gt)*100;
        System.out.println (" percentage = " + percentage + "%");
    }
}
```

Prabodh
PAGE No.:
DATE:

Practice - sets :-

① write a program to sum three numbers in java.

```
import java.util.Scanner;

public class Main {
    public static void main (String [] args) {
        Scanner S = new Scanner (System.in);
        int a = s.nextInt();
        int b = s.nextInt();
        int c = s.nextInt();
        int sum = a+b+c;
        System.out.println("Sum of numbers" + sum);
    }
}
```

write a program to calculate CGPA using mark of ~~three number~~ (out of 100).
sub.

```
import java.util.Scanner;
public class Main {
    public static void main (String [] args) {
        Scanner s = new Scanner (System.in);
        System.out.println("Enter the mark of 1st sub:");
        float a = s.nextFloat();
        System.out.println("Enter the mark of 2nd sub:");
        float b = s.nextFloat();
        System.out.println("Enter the mark of 3rd sub:");
        float c = s.nextFloat();
        float d = (a+b+c)/30;
        System.out.println("cgpa" + d);
    }
}
```

Pradeep
PAGE No. :
DATE :

hello, <input name> have a good day.

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        String name = s.nextLine();
        System.out.println("Hello, " + name + " have a nice day!");
    }
}
```

operator	Description	associativity	Rank.
.	member selection	L → R	1
()	function call	L → R	1
[]	array element refrence	L → R	1
−	unary minus	R → L	2
++	increment	R → L	2
−−	decrement	R → L	2
!	logical negation	R → L	2
~	one's complement	R → L	2
(type)	casting	R → L	2
*	multiplication	L → R	3
/	division	L → R	3
%	modulus	L → R	3
+	add	L → R	4
−	subtract	L → R	4
<<	left shift	L → R	5
>>	right shift	L → R	5
>>>	—u— with filled 0	L → R	5

Pradeep
PAGE No.:
DATE:

operator, types and expression

operators are used to perform operation on variable and value

operator
↑
7 + 5 = 12
↓ ↓ ↓ result
operand operand

Types of operator
discussed earlier.

Java (associativity) ~ (precedence).

(+, -) is having less precedence then (*, /)

precedence ~ int a = 6 * 5 - 34 / 2 ; (due to L to R)
the bigger precendence, the operator will be executed first. * = / in precendence -
= 30 - 17 = 13

associativity ~
associativity of * and / is left to right.
int b = 60 / 5 - 34 * 2 ;
= 30 - 68 (due to L to R)
= 12 - 68
= - 56

see chart ↓
page no - 85

as well as
2

Quick Quiz = how will you write following expression in java

$\frac{x*y}{2}$, $\frac{b^2-4ac}{2a}$, v^2-u^2 , $a*b-d$

$v*v-u*u$

x*y/2 ; (b*b-4*a*c)/(2*a) ;

Java Data types

Resulting data type after arithmetic expression

Result = b+s → int
s+i → int
l+f → float
i+f → float
c+i → int
c+s → int
l+d → double
f+d → double

b - byte
s - short
i - int
l - long
f - float
d - double
c - char

Quick Quiz what will be the value of following expression (x

int y = 7
int x = ++y + 8;
value of x = ? (16);

char a = 'B';
a++; (a = 'C')

Prabodh
PAGE No.:
DATE:

Practice - Set

1. What will be the result of following expression.

float a = 7/4f * 9/2f;

precedence is equal.

so therefore according to associativity will go left to right

= 1.75 * 4.5

= 7.875

2. write a java program to encrypt grade by adding 8. Decrypt it to show correct grade.

```
public class Main {
    public static void main (String[] args){
        char grade = 'B';
            (char) type casting
        grade = (grade + 8);
                //encrypted

        grad = (char) (grade - 8);
                // decrypted;

        System.out.println (grade);
    }
}
```

Padoda
PAGE No. :
DATE :

3. use the comparison operator to find out whether a given number is greater than the user or no

4. write following expression in java program

$$\frac{v^2 - u^2}{2as}$$

5. find the value of following expression.

int x = 7

int a = 7 * 49/7 + 35/7

value of a?

3

```
import util.java.Scanner;
public class Main {
    public static void main (String[] args){
        Scanner s = new scanner (System.in);
//let given number be 40.
    int a = 40;

    int b = s.nextInt();
    if (a > b) {
            System.out.println("user entered
no. is smaller then given number.");
        }
else {
        System.out.println("user entered no. is
greater then given number.");
    }
}
```

4. public class Main {
    public static void main(String[] args) {
        int v, u, a, s;

        int x = (v*v - u*u) / (2*a*s)

        System.out.println(x);
    }
}

5. int x = 7
   int a = 7*49/7 + 35/7;

left to right →

= 343/7 + 5;

= 49 + 5;

int a = 54;

Java Strings

String → this is a sequence of characters.
this is a class (not a primitive data type) but we can use that as primitive data type.

strings are immutable and can't be changed

initiated :

```
String name;
name = new String ("Aman");
```

```
String name = "Aman";
```
reference — object

types of print in Java:

```
System.out.print();
System.out.println();
System.out.printf();
System.out.format();
```
both are same

to input string :-

- ```
  Scanner sc = new Scanner (System.in)
  String st = sc.next();
  ```
  (for one word)
  ```
  nextLine();
  ```
  (for a whole line)

Pradeep
PAGE No.:
DATE:
  ```

String Methods →

we can change cases, find length of string

commonly used string method:

```
                    0 1 2 3
String name = "Aman";
```

1. name.length → (here it is 4) (length)
→ String name = "Aman"; int value = name.length();
2. System.out.println(value);

output: 4

2. name.toLowerCase() → (it will return the string lowercases)

```
String name = "Aman";
String lstring = name.toLowerCase();
System.out.println(lstring);
```

output: aman

3. name.toUpperCase() → same as ↑

```
String name = "Aman";
String ustring = name.toUpperCase();
System.out.println(ustring);
```

output: AMAN

4. name.trim → returns a new string after removing all leading and trailing spaces from original strings.

```
String s = "  Aman  ";
String t = s.trim();
System.out.println(t)
```

output: Aman

Substring → (start)

System.out.println(name.substring(2));

output :- an (index start from 0)

System.out.println(name.substring(2,3));

output : a

name.replace('r','p') → takes char.

String name = "Harry";

System.out.println(name.replace('r','p'));

output :- Happy

System.out.println(name.replace("rry","ier"));

→ Haier

System.out.println(name.replace("r","ier"));

→ Haierierry

→ System.out.println(name.startsWith("Ha"));

→ true

else false

same as it work with ends with

System.out.println(name.endsWith("n"));

→ true

i. name.charAt(2) → it will give ('a');

System.Out.println(name.charAt(2));

out → a

ii. → name.indexOf("ry"));

returns first index

out → 3

String modifiedname = "harryyry";
System.out.println(modified.name.indexOf("ry", 4));

if → -1 then there is no ends

indexOf → search from front

last index of

modifyname.lastIndexOf("ry"); lastIndexOf → search from end

→ name.equal("Harry");
it will check value of the string.
System.out.println(name.equals("Harry"));

→ true. (remember Java is a case sensitive language).

Prabodh
PAGE No.:
DATE:

name.equalsIgnoreCase( );

it will ignore the case of string it will n
check values.

Escape sequence Charach

Sequence after '\'

doublequotes (\")

for one \ → \\

\n → new line \t → tab

\' → singlequote

etc.

Practice-set-3

1. write java program to convert string to lower case.

```
import java.util.scanner;
public Class Main{
    public static void main(String[] s){

    Scanner s= new Scanner(System.in);
    String str = s.next line();

    String lowerStr = str.lower toLower(case));
    System.out.println(lowerStr);

    }
}
```

→ AmeN

⇒ amen

2. write a java program to replace spaces with underscore

```
public class Main{
    public static void main (String[] s)
    {
    String str = " Hello this is space with
underscore";
    String lower = str.replace(' ', '_');

    System.out.println (lower);
}
}
```

_Hello_this_is_space_with_underscore

③ write a Java program to fill in a letter template which looks like below.

letter = "Dear <|name|>, Thanks a lot";

to replace ↑

replace <|name|> with a string (some name)

```
public class Main{
    public static void main(String[] s)
{

    String letter = "Dear <|name|>, Thanks
a lot!";

String repl = letter.replace("<|name|>", "harry");

    System.out.println(repl);

}
}
```

"" for string '' for char

→ Dear harry, Thanks a lot!

④ write a java program to detect double and triple spaces in a string.

```
public class Main {
    public static void main(String[] s){

    String str = "this string contains
double and thriple space";

System.out.println(str.indexOf("  "));

System.out.println(str.indexOf("   "));
```

(5) write a program in format the following letter using escape sequence characters.

letter = "Dear Aman, This java course is nice. Thanks!"

letter should be like :

Dear Aman,
    This java course is nice.
Thanks.

```
public class Main {
    public static void main(String[] s) {
        string str = ("Dear Aman, \n\t
This Java course is nice. \n thanks. ");

        System.out.println(str);
    }
}
```

Conditional Statements

→ decision - making -

→ if - else statement
→ switch statement.

④ if - Else Statement

this's syntax look like c and C++.

(condition) of

```
if (expression) {
        statements ;
    }
else {
    statements ;
    }
```

true condition

false condition

optional in if else

first example :—

```
public class Main {
    public static void main (String [] s){
        int age = 18 ;
        if (age > 19){
            System.out.println ("you can drive.");
        }
        else {
            System.out.println ("you can't drive.");
        }
    }
}
```

→ you can't drive

Relational operators → ==, >=, <, >, <=, !=

== → comparism

!= → not equal to

= for assignment

Relational and logical operators for Statements

&& , || , ! } used to p

&& → and

|| → or

! → not

logical

And operator

T && T = T		T || T = T
T && F = F		T || F = T
F && F = F		F || T = T
F && T = F		F || F = F

```
public class Main{
    public static void main (String [ ] s){
        boolean a = true;
        boolean b= false;
        if(a && b){
            System.out.println("Y");
        }else{
            System.out.println("N");
        }
    }
}
```

true && false = false

Switch case → Statement

first will discuss

else - if clause. → ladder

```
if (condition) {
}
else if (condition) {
}
else if (condition) {
}
else {
}
```

false → false → true → execute

not checked

Switch case →

it is used when we have to select from choices

```
switch (input) {
    case :
        statement;
    break;
    case :
        statement;
    break;
    case :
        statement;
    break;
    default :
        statement;
}
```

if we dont use break it will print further cases after true. including defaults.

enhanced switch case.

```
switch (var {
    case --> { // code } ;
    case --> { } ;
    case --> { } ;
    default -->  ;
```

Practice set

1. what will be the output ?

```
int a = 10;
if (a == 11) {
        System.out.println("I am 11");
    }
else {
        System.out.println("I am not 11");
    }
```

2. write a program to find out whether a student is pass or fail if require total 40% and atleast 33% in each subject to pass.
assume 3 sub. as input from user.

input of 3 sub → 1 sub / 100 >= 33% ✓
each ✓
total → total / 3 × 100 >= 40% ✓

```
import java.util.scanner;
public class main {
    public static void main(String[] ){
        Scanner s = new Scanner(System.in);
        Byte a=s.nextByte();
        system.out.println("Enter the number of
your first subject:");
        byte a = s.nextByte();

        System.out.println("Enter the number of
your second subject:");
        byte b = s.nextByte();

        system.out.println("Enter the number of
your third subject:");
        byte c = s.nextByte();

        float avg = (a+b+c)/3.0f;

if(avg >=40 && a >=33 && b>=33 && c>=33){
            System.out.println("you cleared
every subject !!!");
        }
        else {
            System.out.println("you failed!!!");
        }
    }
}
```

3. calculate income tax paid by an employee to govt

income slab	tax
2.5l - 5.0l	5%
5.0l - 10.0l	20%
above 10.0l	30%

0 - 2.5l no tax

30
100

use if-else ladder

4. write a program to find out the day of week
for 1 → monday
2 → tuesday
.
.
.
so on

Java Exercise - 2

1. Rock Paper scisor Game

```
import java.util.scanner;
import java.util.Random;

public class Main {
public static void main (String[] args ){
    Scanner sc = new Scanner(System.in);

while (true){
    String[] rps = {"r", "p", "s"};
```

```
String computerMove = ops[new Random().nextInt(ops.length)];

String playerMove;

while (true){
    System.out.println("please enter your move(r,p,or s);
        playerMove = scanner.nextLine();
if(playerMove.equals("r")||playerMove.equals("p")||
playerMove.equals("s")){
            break;
    }
System.out.println(playerMove + "is not a valid move");
}

System.out.println("Computer played :" + computerMove);

if(playerMove.equals(computerMove){
    System.out.println("The game was a tie");
}
else if(playerMove.equals("r")){
    if(computerMove.equals("p")){
        System.out.println("you lose!");
}
else if(computerMove.equals("s")){
    if(computerMove.e
        System.out.println("you win!");
    }
}
else if(playerMove.equals("p")){
    if(computerMove.equals("r")){
        System.out.println("you win");
    }
else if(computerMove.equals("s")){
        System.out.println("you lose");
    }
}
```

Prabodh
PAGE NO.:
DATE:

```
        else if (playerMove.equals("s")){
            if (computersMove.equals("p")){
                System.out.println("You win!");
            }
        else if (computersMove.equals("r")){
                System.out.println("You lose!");
            }
        }

        System.out.println("Play again? (y/n)");
        String playAgain = scanner.nextLine();

        if (!playAgain.equals("y")){
            break;
        }

    }
}
```

class and object

Defining a class

syntax

```
class classname [extends superclassname]
{
    [field declaration ;]
    [method declaration ;]
}
```

example:

```
class Rectangle
{
    int length;
    int breadth;
}
```

method declaration

```
type methodname (parameter list)
{
    method body;
}
```

it has four basic part:

1. the name of method (method name)
2. type of value the method returns (type)
3. a list of parameters (parameter list)
4. body of the method (method body).

example

```
class Rectangle
{
    int length, breadth;
    void getdata(int x, int y){
        length = x;
        width = y;
    }
    int rectArea(){
        int area = length * breadth;
        return(area);
    }
}
```

creating objects.

In Java objects are created using new operator

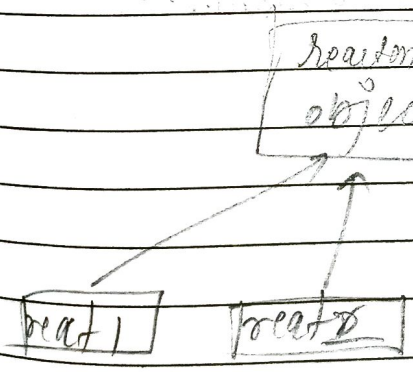
Action	Statement	Result
Declare	Rectangle rect1	
Instantiate	rect1 = new Rectangle()	

as for upper example:

```
{ Rectangle rect1;
  rect1 = new Rectangle();
```

→ Rectangle rect1 = new Rectangle();

the method Rectangle() is a default constructor



accessing class members

objectname.variablename = value ;
objectname.methodname(parameter list);

Method overloading.
The methods who have same name but different parameter list and different defination. Then this method is called method overloading.

This is also known as polymorphism

```
class Room
{
    float length;
    float breadth;
    Room (float x, float y)          // constructor 1
    {
        length = x;
        breadth = y;
    }
    Room (float x)                   // constructor 2
    {
        length = breadth = x;
    }
void area()
    {
float length * breadth);
    area }   Sout (area);

public static void main (String[] s){
        Room r = new Room (10f, 20f);
        r.area();
        Room r2 = new Room (20f);
```

```
        r2.area();
    }
}
```

to get largest from 2 numbers.

```
class Large
{
    byte m, n;
    Large (byte x, byte y){
        m=x;
        n=y;
    }
    Byte Largest(){
        if (m≥n);
            return m;
        else
            return n;
    }
void display ()
    {
        int big = largest();
        System.out.println("Largest value = "+big);
    }
public static void main(String [] a){
        Large l = new Large(50, 40);
        l.display();
    }
```

Constructors

constructors are used to enable an object to initialize itself when it is created.
constructor have the same name.

They do not specify any return type.

```
class Rectangle {
    int length;
    int width
    Rectangle ( int x, int y)        // also known as parameterized constructor / constructor method
    {
        length = x;
        breadth = y;
    }
    int rectArea ( ) {
        return ( length * breadth );
    }
}
```

Static members —

8.16.

Design a class to represent a bank account

Data members: Name of depositor
type of account
account number
Balance amount in account

Methods: to assign initial values.
to deposit an amount
to withdraw an amount after
to display name & balance.

Bank.java

Java method.

some time size of java program increase by a lot.

method : if we use same specific code multiple times or logic

DRY → do not repeat yourself

syntax → method is a function written inside a

```
returnType name() {
    // method body
}
```

→ it will just copy from the args.

```
static int sum(int a, int b) {
    int c = a + b;
    return c;
}
```

calling a method.

called by creating an object

example

```
call obj = new call()   - creating
obj.sum(a, b);          - calling
```

if we use public then

OOPS

method overloading → later

method can't be overload by changing return type

void return type → is used when we don't want return anything

static → it associate with class rather than object and can be shared by all object.

→ two or more method having same name but have different purpose. ∴ so also have different parameters.

static fun () { }
Static fun (int a) { }
↳ parameter

Argument are actual

Java variable arguments. (varargs)

Recursion in java

a function in java which call it self to solve problems or apply logic is called recursion.

factorial
fibonacci series.

Inheritance
→ single, multi
→ method overidding
→ dynamic model dispatch,
→ abstract class

Interface & Package
↓ visibility

wrapper class.
autoboxing and unboxing
enumeration & metadata

exception handling

Interface

- Interface is a mechanism used in java to define multiple roles of an object, so that same object can play different roles.
- role is defined in java application by set of abstract method which are called role methods.
- all methods of an interface are by default, abstract method and public method.
- we can't create object of an interface but we can create refrence variable
- a class can implement multiple interface so that it can play multiple roles.

Features of interface

- interface is a refrence type in java. Similar to class.
- Collection of abstract methods
- along with abstract methods, an interface can also contain constants, default methods, static methods nested types.
- Method body only exist for default method and static method.

Similarities and Dissimilarities between Interface and Class.

Similarity

1. Interface can contain any number of method.
2. An interface is written in java file with .java extension, with the name of interface matching file.
3. The byte code of an interface appears in a .class file.
4. Interface appears in package, and their corresponding byte code file must be in a directory structure that match package name.

Dissimilarity

1. You can't instantiate an interface.
2. Interface doesn't contain any constructor
3. All methods are abstract
4. Can't contain instance field only field that can appear in an interface is declared as both static and final.
5. An interface is not ~~declared~~ extended by a class; it is implemented by a class
6. Interface can extend multiple interfaces.

Inheritance in Java.

Inheritance is used to borrow properties & method from existence existing class

[Phone] ⟶ [Smart phone]

Super class ⟶ Sub class

Subclass extends superclass.

① Declaring inheritance

inheritance in java is declared using "extends" keyword

superclass
↓
subclass

subclass extends superclass.

Example

vehicle → car animal → dog animal → cat vehicle → truck

when class inherits from a superclass, it inherits part of superclass method and field.

Java doesn't support multiple inheritance i.e. two class can't be super class for a subclass.

[super] [super] → [sub]

Quick Quiz
create a class Animal and desire dog.

```
class Animal {
    public void legs (int x) {
        int y = x;
        System.out.println("no. of legs" + y);
    }
    public void colour (String s) {
        String l = s;
        System.out.println ("colour of animal
+ l);
    }
}
class Dog extends Animal {
    public void sound () {
        System.out.println ("dog is barking!");
    }
}
public class Main {
    public static void main (String args []) {
        Dog d1 = new Dog ();
        d1.legs (4);
        d1.colour ("black");
        d1.sound ();
    }
}
```

output: no. of legs 4
colour of animal black
dog is barking!

Constructor in Java

getter and setter and access modifier

access modifiers
specify method / property accessible or not.

private (keyword) not directly accessible. → can be accessed with method and it'll support data hiding.
Default
protected (keyword)
public (keyword)

implement

getter doesn't return
getter does return.

quiz create getter and setters, area, circumference.

constructor in java.

public Classname( ) { }

with parameter, or without (default)

and also it can be overload.

note: constructor can take parameters without being overload.
→ constructor can be more than 2.

Quick Quiz
overload employee construct and initialize the salary to 10000.

```
class Employee {
    public String name;
    public int id;
    public float salary;

    public Employee(){                    //default constructor
        name = "enter your name";
        id = 0;
        salary = 10000;
    }
    public employee(String s, int i, float sal) {
            name = s;
            id = i;
            salary = sal;
    }
}
class Main {
    public static void main(String s[]){
        Employee e1 = new Employee("Aman", 1, 12
    00);
        e1.display();
    }
}
```

Constructors in Inheritance

if we make constructor in base it can called by derived also.

when we have constructor in both derived and base then both will be executed but base constructor will be executed first. when make derived object.

when base class is overloading with an parameterized constructor then to execute that we have to use super() keyword.

super(a, b); if (a and b are integers) or else

This & Super keywords

This keyword:

this is a way for us to refrence an object of the class without it's being created.

this.area = 2;

just like we create object for class that just refer to class.

Super keyword.

is a reference variable used to refer immediate parent class.

- can be used to refer the immediate parent class instance variable
- → can be used to invoke parent class meth
- → can be used to invoke parent class constr

Abstract ... nent.

Earlier inheritance practice question

2. create a class rectangle and use inheritance to create another class cuboid' try to keep it close at real world scenario as possible.

```
class Rectangle{
    public int l;
    public int b;
    public int area(int a, int b){
        a=l;
        b=b;
        return a*b;
    }
}

class cuboid extends Rectangle {
    public int B;
    super(x,y);
    public int volume(int a, int b, in()
    {   c=B
        super(a,b);
        return area*B;
    }
class Main {
    public static void main(String arg[])
        cuboid c1 = new cuboid(1,2,3);
        c1.volume();
    }
}
```

Prabodh
PAGE No.:
DATE:

abstract class and method
↳ not having physical existance
only existing in thought.

abstract method :
a method that is declared without implement ation.

abstract void moveTo(double x, double y)

abstract class.
↳ which includes abstract methods.

one abstract method makes whole class abstract class. In abstract class we can use other methods also.

→ can create reference.
we can't create object of abstract class.

shape (abstract)
↳ circle
↳ rectangle
↳ rhombus

method overridding

for(polymorphism).

overloading.
→ same class two method with different parameters but with same method name.

V/S

override
→ same name of method but in two different class.

→ when an object of subclass is created and overrided method is called, then the method which is implemented in subclass is called an executed.

→

Dynamic Method Dispatch

superclass reference = object of subclass

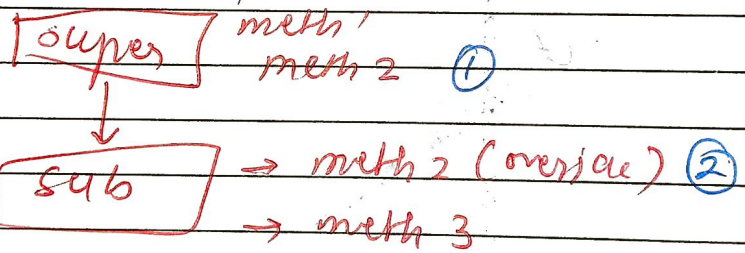
⟷

but not vice-versa

a obj = new b();

⟹ method of this is executable.

to get runtime polymorphism



scenario: super obj = new sub() → allow

obj.meth2() → ② is called (method of object)

obj.meth3() →

obj.meth3 (not allow,

only superclass method allow)

```
class Phone {
    public void showTime(){
        System.out.println("time is 23:25 ");
    }
    public void on(){
        System.out.println("turning on
phone...");
    }
}
class SmartPhone extends Phone {
    public void music(){
        System.out.println("playing music");
    }
    public void on(){
        System.out.println("turning on Smart
Phone...");
    }
}
public class Main{
    public static void main(String[]as){
        Phone obj = new SmartPhone();
```

reference of phone but object of smartphone.

```
        obj.showTime();
        obj.on(); // smartphone on();
    }
}
```

Object Oriented programming in java.

Solving a problem by creating object is one of the most popular approaches in programming.

in python we use procedural programming

DRY → do not repeat yourself.

class → [Jee application] ⇒ filled by student ⇒ [application for that student]

(name, %, father & mother etc already)

[class] → object instantiation ⇒ [object]

object → is an instantiation of class. when a class defined template is defined, memory is allocated after instantiation.

How to model a problem in OOPs.

we identify,

Noun : class :→ employee

adjective : attributes → name; age; salarie.

verb : methods :→ getSalary(); increment();

Terminology

(i) Abstraction :- Hiding internal details.
[showing only essential info]

 ⇒ use this phone without bothering about how its done.

(ii) Encapsulation =>
the act of putting various component together (in a capsule).

in java. encapsulation simply means that the sensitive data can be hidden from the user.

3. inheritance :- act of deriving new things from existing things.

rickshaw → e-rickshaw
phone → smartPhone
inheritance implements DRY.

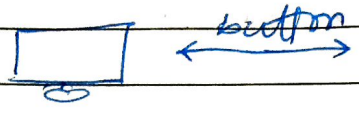
4. Polymorphism → one entity many forms.

smart phone → phone
smat phone → calculator
smartphone → camera
smartphone → music system.

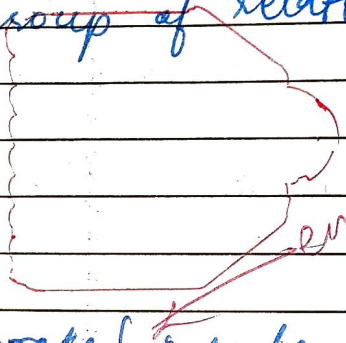
~~Custom Class~~ (it is a group of method which has no object to be implemented)

Interface → yes reference ✓ extended

in eng → interface is point where two system meet and interact.



in java interface is a group of related methods with empty bodies.



example:-

```
interface Bicycle {
    void applyBrake(int decrement);
    void speedup(int increment);
}
class AvonCycle implements Bicycle {
    int speed = 7;
    void applyBrake(int decrement) {
        speed = speed - decrement;
    }
    void speedup(int increment) {
        speed = speed + increment;
    }
}
```

we use interface because we can implement multiple interface and can also be extended one.

abstract class can only be extends ones.

interface implements multiple time / extends one abstract extends ones -

→ we can't modify properties in interface as they are final (always).

→ we can create properties.

Why Multiple inhesitance in java. is not allowed.

multi inhesitance face problem when there exist method with same signature in both super class.

Due to such problems, java doesn't support multi inheritance directly but similar concept can be achieved using interface.

A class can implement multiple interface and extends a class at the same time.

Note:- (i) interface in java is bit like class. but with

(ii) fields will be constant. and default method. they can be override, but methods can be implemented (necessary).

(iii) you can create reference of interface but not object.
create.

(iv) methods are public by default.

DATE :

Binary converter.

```
import java.util.Scanner;
class Converter {
    public static String toBinary(int num){
        String binary = "";
        while(num>0){
            binary = (num%2) + binary;
            num /= 2;
        }
        return binary;
    }
}
public class program{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        System.out.println(Converter.toBinary(x));
    }
}
```

four concept of OOP :- ENCAPSULATION
INHERITANCE
POLYMORPHISM
ABSTRACTION

Encapsulation

The idea behind encapsulation is to ensure that implementation details are not visible. The variable will be hidden from other class. accessible only through methods of current class. This is called data hiding.

To achieve encapsulation, declare class variable as private and provide public getters and setters.

for example :-

```
class BankAccount {
    private double balance = 0;
    public void deposit(double x) {
        if (x > 0) {
            balance += x;
        }
    }
}
```

This implementation hides balance variable. only accessible through deposit method.

Benifits :

→ control of the way data is accessed or modified.
→ more flexible and easily changed code.
→ ability to change one part of code without effecting other part.

DATE :
PAGE No. :

```
import java.util.Scanner;
public class Main
{
    public static void main (String [] args){
        Scanner read = new Scanner (Sy
        int a = read.nextInt();
        Pupil pupil = new Pupil();
            pupil.setAge(a);

    }
}
class Pupil {
    private int age;
    public void setAge(int a) {
            this.age = age;
            if(a>6){
                System.out.println("Welcom
            }
            else
            {
                System.out.println("Sorry
            }
    }
    public int getAge (){
        return age;
    }
}
```

inheritance:

inheritance is the process that enables one class to acquire the property (method and variable) of another, the information is placed in a more manageable and hierarchical order.

The class inheriting the property of another is subclass (also known as child or derived).
the class whose properties are inherited is superclass (base class or parent).

To inherit we have to use keyword extends.

it inherits all non-private variables and methods.
(∵protected methods/variable are accessible to subclass.).

example

```
class Animal {
    protected int legs;
    public void eat(){
        System.out.println("Animals eats");
    }
}
class Dog extends Animal{
        Dog(){
            legs = 4;
        }
}
public class Main{
    public static void main(String[] args){
```

```
        Dog d = new Dog();
        d.eat();
    }
}
}
```

constructor are not member method. therefore they cant be inheritated by subclass.
However the constructor of superclass is called when the subclass is instantiated.

```
class A {
    public A(){
        System.out.println("new A");
    }
}
class B extends A{
    public B(){
        System.out.println("new B");
    }
}
class Program {
    public static void main(String[] args){
        B obj = new B();
    }
}
```

we can access super class from the subclass using the super keyword.

Polymorphism

polymorphism means having many forms.
A call to a member method will cause different implementation to be executed depending on the type of object invoking method.

eg →

```
class Animal {
    public void makeSound() {
        System.out.println("Grr...");
    }
}
class Cat extends Animal {
    public void makeSound() {
        System.out.println("Meow");
    }
}
class Dog extends Animals {
    public void makeSound() {
        System.out.println("Woof");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal a = new Dog();
        Animal b = new cat();
        a.makeSound();
        b.makeSound();
    }
}
```

method overriding.

rules → should have same return type and args.
→ access level can't be restricted.
→ final and static method can't be overridden
→ if method can't be inherited, it can't be overridden.
→ constructors can't be overridden.

method overriding is also known as runtime polymorphism.

Method overloading.
when methods have same name but different parameters in a class is known as method overloading.

method overloading is known as compile time polymorphism

ABSTRACTION

Data abstraction. provides the outside world with only essential information, in a process of representing essential feature without including implementation details.

abstraction is achieved using abstract classes and interface.
→ abstract class is defined using abstract keyword.
→ if a class is declared abstract it can not be instentiated (you can't create object of that type)

Pabodi
PAGE NO.:
DATE:

→ to use abstract class we have to inherit it from another class.
→ any class that contain abstract methods must declared as abstract.

abstract method is a method which is declared without implementation (without braces & & followed by ;)

```
abstract void walk();
```

Interface

interface is completely abstract class that only contain abstract methods.
→ Defined only using interface keyword.
→ may contain only static final keyword.
→ can't contain constructor because interface can't be instantiated.
→ interface can extends other interface
→ A class can implement any number of interface.

eg→

```
interface Animal {
    public void eat();
    public void makesound();
}
```

properties —

- an interface is implicitly abstract. we don't need to use abstract keyword while declaring interface.

- each method in interface is also implicitly abstract

- method in an interface are implicitly public.

A class can inherit from just one superclass, but can implement multiple interface!

use the keyword implements to use an interface with your class.
when you implement an interface, you need to override all of its method.

```
interface Animal {
    public void eat();
    public void makeSound();
}
class Cat implements Animal {
    public void makeSound() {
        System.out.println("meow");
    }
    public void eat() {
        System.out.println("omnomnomnom");
    }
}
```

Type Casting

assigning a value of one type of variable into another type is known as type casting.

to cast a value to a specific type. place the type in parantheses and position it in front of value.

```
int a = (int) 3.14;
```

this will cast the value 3.14 to integer with resulting value 3.

```
double a = 42.571;
int b = (int) a;
```

java supports automatic type casting of integers to floating points. since there is no loss of precision.
on other hand, type casting is mandatory when assigning floating points.

Down casting

casting an object of superclass to its subclass is called down casting.

```
Animal a = new Cat();
((Cat) a).makeSound();
```

this will try to cast variable a to the Cat type and call its makeSound() method.

Up casting

you can cast an instance of subclass to its super class.

```
Animal A = new Cat();
```

java automatically upcast Cat type variable to animal type.

Anonymous Class.

→ way to extend existing class on fly.

```
class Machine{
    public void start(){
        System.out.println("...Starting...");
    }
}
```

when creating machine object, we can change start method.

```
public static void main(String[] args){
    Machine m = new Machine(){
        @override public void start(){
            System.out.println("woooo"); }
        };
    m.start();
}
```

output: woooo

after the constructor call, we have opened curly braces and have overridden method.

The @override annotation is used to make your code easier to understand

The modification is only applicable to current obj

```
class Machine {
    public void start(){
        System.out.println("starting");
    }
}
```

Pabodh
PAGE No.:
DATE:

anonymous class are way to extend.

```
public static void main (String[] args){
    Machine m1 = new Machine (){
    @Override public void start(){
        System.out.println ("Wooo");
    }
    };
    Machine m2 = new Machine();
    m2.start();
}
```

output: Starting.

inner class

java support nesting classes; a class can be a member of another class.

creating another class is simple. just write a class within a class. unlike a class an inner class can be private, it can't be accessed from an object outside of class.

```
class Robot {
    int id;
    Robot (int i){
        id = i;
        Brain b = new Brain();
        b.think();
    }
    private class Brain {
        public void think(){
            System.out.println (id + " is thinking");
        }
    }
```

Enums. special type used to define collection of co

```
enum Rank {
    SOLDIER,
    SERGEANT,
    CAPTAIN}
```

```
Rank q = Rank.SOLDIER;
```

Java API

Java API is a collection of classes and interfa that have been written for you to use

packages can be imported using import key. -word.

```
import java.awt.*;
```

awt → contains painting and interface graphics and image.

base → shape class.
abstract → area. width → attribute.

square circle
subclass.

```
import java.util.Scanner;
abstract class shape {
    int width;
    abstract void area();
}
class Circle {
    int x;
    public Circle(int x){
        this.x = x;
    }
    System.out.println(Math.PI * x * x);
}
}
class Square {
    int x;
    public Square(int x) {
        this.x = x;
    }
    System.out.println(x*x);
}
}
```

Exceptions

an exception is a problem that occurs during program execution, exception causes abnormal termination of programming.

exception handling is a powerfull mechanism that handles runtime error to maintain normal flow

exception can be handled with a try catch block between the code.

Syntax:

```
try {
        //some code
} catch (Exception e) {
        //some code to hande error
}
```

a catch statement involves declaring the type of exception you are trying to catch. if an exception occur in the try block, the catch block that follows the try is checked. if the type of exception that occurs is listened in a catch block much as an argument is passed into another method parameter. The exception type can be used to catch all possible exception.

```
public class MyClass {
    public static void main (String[] args) {
        try {
            int a[] = new int[2];
            System.out.println(a[5]);
        } catch (Exception e) {
            System.out.println("An error occured");
        }
    }
}
```

without try/catch block this code should catch the code. as a[5] doesn't exist.

Throw

The throw keyword allow you to manually generate exception from your method. Some of the numero as available exception types includes. IndexOutOfBound exception, ArithmeticException and so on.

eg:-

```
int div(int a, int b) throws ArithmeticException {
    if (b==0) {
        throw new ArithmeticException ("Division by zero");
    }
    else {
        return a/b;
    }
}
```

the throw statement in the method defines type of exception the method can throw.

multiple exception can be defined in throw statement using comma-seperated list.

Exception Handling

a single try block can contain multiple catch block that can handle different exception seperately.

```
try {
    // some code
} catch (ExceptionType1 e1){
    // catch block
} catch (ExceptionType2 e2){
    // catch block
} catch (ExceptionType3 e3){
    // catch block
}
```

all catch block should be ordered from most specific to general.

00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34
40	41	42	43	44

```
for (i=0 ; i<5 ; i++) {
    for(j = 0; j
```

Array list stores object thus type specified must be class type such as Integer for int, Double for double

Array list : java API provides special classes to store and manipulate group of object one such class is arraylist (ArrayList)

Standard java array are of fixed length. which means after they created they can't be expanded or shrink

ArrayList are created with initial size but when this size is exceeded the collection is automatic enlarged.
When object are removed the array list may shrink in size

it is important to import java.util.ArrayList;

```
import java.util.Arraylist;
ArrayList colors = new ArrayList();
ArrayList<String> colors = new ArrayList<String>(10);
```

Pradeep
PAGE NO. :
DATE :

Arraylist class provides a useful number of method for manipulating object.

the add() method add new object to array
conversely remove() method remove —11—

example:

```
import java.util.ArrayList;
public class Main{
    public static void main(String[] args){
        ArrayList<String> colours = new
            ArrayList<String>(10);
        colours.add("red");
        colours.add("blue");
        colours.add("green");
        colours.add("pink");
        colours.remove("green");

        System.out.println(colours);
    }
}
```

output → [red, blue, pink]

other use ful method.

- contains() Return true if the list contain the specified element.
- get(int index) Returns the element of at the specified position in list
- size() — returns the number of element in list
- clear() — removes all element from list.

(∵ indexing start with zero 0)

Linked List:

linked list is very similar syntax with Array List.

```
import java.util.LinkedList;
public class Main {
    public static void main(String[] arg) {
        LinkedList<String> c = new LinkedList
        <String>();

        c.add("red");
        c.add("blue");
        c.add("green");
        c.add("orange");
        c.remove("green");

        System.out.println(c);
    }
}
```

output: [red, blue, orange]

④ Cant specify initial capacity in linked list

ArrayList	Linked List
→ when rapid access to data.	→ large number of insert/delete

HASH MAP

Hash map is used to store data collection as key and value pair. One object is used as key (index) to another object (the value).

put(), remove() and get()
add delete access.

key index	value

```
import java.util.HashMap;
public class Main{
  public static void main(String[] args){
      HashMap<String, Integer> points = new
          HashMap<String, Integer>();
    point.put("Aman", 19);
    point.put("Nilima", 20);
    point.put("Garima"; 22);

    System.Out.println(points.get("Aman"));
  }
}
```

out : 19

HashMap can't contain duplicate key
adding a new item with a existing key will ~~override~~ overwrite.

containsKey(), containsValue();

SETS

A set is collection that can not contain duplicate elements. It models the mathematical set abstraction

one of the implementation of set is HashSet

```
import java.util.HashSet;
public class Myclass {
    public static void main (String[] args){
        HashSet<String> set = new HashSet<String>();
        set.add("A");
        set.add("B");
        set.add("C");
        System.out.println(set);
    }
}
```

Out: [A, B, C]

size() to get number of elements

Sorting List

for the manipulation of data in different collection java . API provides Collections Classes. which is included in java.util.package.

one of the most popular Collection class method sort(). The methods are static (no need of object)

```
public
import java.util.ArrayList;
import java.util.Collections;
public class Main {
  public static void main (String[] args) {
    ArrayList<String> animal = new ArrayList<
        String>();

        animal.add ("tiger");
        animal.add ("dog");
        animal.add ("lizard");
      Collection.sort(animal);
      System.out.println(animal);
    }
}
```

out [dog, lizard, tiger]

Collections.sort)

also do same for <Integer>

Other useful method.

max (Collection c) – return the maximum element in c as determined by natural ordering.

min(Collection c) – return minimum –"–

reverse (List list) – reverse sequence in list

shuffle (List list) – shuffles (i.e. randomize) the element in list.

Iterators

an Iterator is an object that enables cycle through a collection, obtain or remove objects

iterator() — that returns iterator to the start of collection.

hasNext() — return true if there is atleast one more element; else return false

next() — return next object and advance the iterator.

remove() — removes last object that was returned by next() from collection

iterator Class must be imported

```
import java.util.Iterator;
import java.util.LinkedList;

public class Main{
    public static void main(String[] args){
        LinkedList<String> animal = new LinkedList
        <String>();
        animal.add("fox");
        animal.add("cow");
        animal.add("dog");

        Iterator<String> it = animal.iterator();
        String value = it.next();
        System.out.println(value);
    }
}
```

Ouput: fox