CONTENTS PAGE

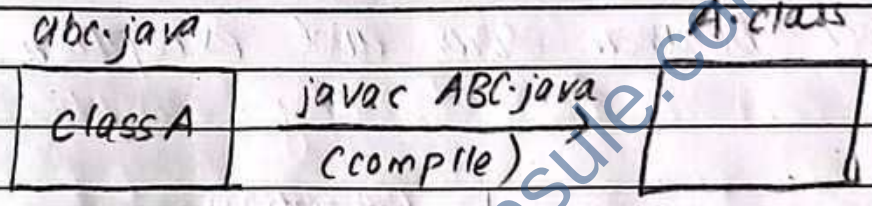
OOP with Java

November 2024

Group "A"

(1) What'll convert Java code into byte code?

⇒ The Java Compiler converts Java source code into bytecode.



(2) How does for each differ from for loop?

⇒ For each loop is specially designed for iterating over elements in a collection, such as array, lists or sets, that doesn't require explicit indexing as contrast to for loop.

eg: int a[] = {4, 5, 6};

```
for (int v: a) {
        sout(v);
}
```

Define package.

Package is a namespace that organizes classes, interfaces and subpackages into a structured hierarchy.

eg. import package.name.Class;

Difference between error and exception.

Error	Exception
Recovering from error is not possible	Exception can be handled
Errors are unchecked type	Exceptions include both checked as well as unchecked.
eg: StackOverflowError	eg: ArithmeticException

(5) What is generic programming?

=> The programming that allows us to write general algorithm, classes or methodes that can operate on any data type using a type parameters (<T>) is called generic programming.

eg: Test<String> t = new Test<>();

(6) Define typecasting.

=> It is the method of converting one data type into another.

- Explict type casting
- Implicit typecasting

eg: int a = (int) 3.4 // explicit

(7) What is the name of the class that is base class for all classes.

=> Object class is the superclass of every classes. ie we can write,

Object ob = new Anyclass();

(8) When do you use inner class?

=> Inner class can be used when we need access to all variables and methods of its outer class and may refer to it directly.

It helps to perform encapsulation, and in enhancing functionality of the code.

(9) Can you convert float to integer implicitly?

=> No, it requires explicit typecasting, because its narrowing conversion.

```
int x = (int) 5.6;
```

(10) What is the role of static field?

=> A static field is shared by all instances of a class. It belongs to the class, not to any object.

```
Static int count; // same value among all.
```

Group - B

(11) How do you handle big numbers? Illustrate with an example.

⇒ BigInteger and BigDecimal classes are used to handle very large numbers that can't fit in primitive types like int or long.

eg:

```
import java.math.BigInteger;

class BigNumberDemo {

    public static void main (String[] args) {

        BigInteger x = new BigInteger("12345691258492
        BigInteger y = new BigInteger("98765432104052

        BigInteger sum = x.add(y);

        System.out.println("Sum = " + sum);

    }

}
```

2) WAP to find the value of a^b for given a and b.

```
import java.util.Scanner;

Class Value {
    public static void main (String[] args){

    Scanner sc= new Scanner (System.in);
    sout ( "Enter a and b:" );
    int a = sc.nextInt ();
    int b = sc.nextInt ();
    double power = Math.pow (a, b);
    sout( a + "^" + b + "=" + result );
    }
}
```

(14) WAP that declares and initializes an integer array of size 15 and display only the perfect square integers.

```
class Array {
    public static void main (String[] args)
    int [] arr = {4, 5, 9, 10, 16, 49, 64, 98};
    for (int n : arr) {
        int root = (int) Math.sqrt(n);
        if (root * root == n)
            Sout(n + " ");
    }
}
```

5) Do we need to define default constructor all the time? Justify.

=) No, we don't need to define it every time

This is because if we don't define any constructor Java automatically provides a default constructor that initializes objects with default values (0, null, false)

However, on defining constructor (parameterized constructor), then we must explicitly define the default one if we need it.

```
class Rect {
    int l, b;
    Rect (int le, int br)
    {
        l = le;
        b = br;
    }
}
Rat () {
```

```
class Demo {
    public static void main (String[] args)
    {
        Rect r1 = new Rect (2, 5);

        Rect r2 = new Rect (); // Error
    }
}
```

However,

```
class Rect {
    int l, b;
}

class Demo {
    psvm (String[] args) {
        Rect 1 = new Rect (); // correct
    }
}
```

16) WAP to display the sum of 5 numbers using generic method.

```
class Demo {

public static <T extends Number> double
sum(T[] num)
 {
     for(T n : num) {
         total += n.doubleValue();
   }
   return total;
}

public static void main (String[] args)
{
   Integer[] nums = {2, 3, 5};

   Double[] dnums = {2.0, 3.8, 9.9};

   Sout("Integer sum = " + sum(nums));
   Sout(" Double sum = " + sum(dnums));
}
}
```

Group C

(17) Why do we need wrapper class? Illustrate with an example.

⇒ Need of Wrapper class:

i) To convert primitive data types into objects.
ii) Java collections (like ArrayList) do not support primitives, only objects.
iii) Access utility methods for converting to string, parsing values, getting MIN/MAX constants.
iv) Autoboxing and Unboxing

eg:
```
class WrapperDemo
    public static void main(String[] args){
    Integer a = Integer.valueOf(100);
    String st = a.toString();
    Sout("Equivalent string is" + st);
    Sout("Upper limit:" + Integer.MAX_VALUE);
    Sout("Lower limit:" + Integer.MIN_VALUE);
    }
}
```

(18) WAP to create your own exception as invalid string such that when the string given by user is of odd length.

```
import java.util.Scanner;

class ExceptionDemo {
    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    try {
        Sout("Enter string");
        String str = sc.nextLine();
        if(str.length() % 2 != 0) {
            throw new InvalidStringException();
        }else{
            sout("valid string");
        }
    } catch( InvalidStringException e){
        sout(e);
    }
}
}
```

```
class InvalidStringException extends Exception{
public String toString(){
        return("Invalid String, must be of even length
}
}
```

(19) WAP to read an integer and if the integer is positive write into the file "pos.dat" otherwise "neg.dat".

```
import java.io.*;
class FileWrite{
    public static void main(String[] args)
    Scanner sc= new Scanner(System.in);
    try{
        out("Enter an integer:");
        int num= sc.nextInt();
        if(num>=0)
            FileWriter fw= new FileWriter("pos.dat");
        else
            FileWriter fw= new FileWriter("neg.dat"
        fw.write(num+" ");
        fw.close();
}
```

```
        catch (Exception e){
            Sout(e);
        }
    }
}
```

20) Create a class named College with instance variables ename and crank. Add a method named setCollege with parameters name and rank. Add another method named showRecord to display values of these variables. Now derive a class named Student with member variables name and roll. Also add two methods setStudent and displayRecord to set Parameters and to display records in Student. Now in main(), declare 4 objects of Student and display the records of Student whose name ends with "na".

```
import java.util.Scanner;

class College {
    String ename;
    int crank;

    void setCollege(String name, int rank) {
        ename = name;
        crank = rank;
    }

    void showRecord() {
        sout("College name:" + ename);
        sout("College rank:" + crank);
    }
}

class Student extends College {
    String name;
    int roll;

    void setStudent(String n, int r, String cname, int cr
    {
        name = n;
        roll = r;
        setCollege(cname, crank);
    }
```

```
void displayRecord(){
    sout("Name:" + name);
    sout("Roll no: " + roll);
    showRecord();
}
}

class mainDemo{
    p s v m (s[] args){
    Scanner sc= new Scanner(System.in);
    Student[] s = new Student[4];
    for(int i=0; i<4; i++){
    s[i] = new Student();
    sout("Enter name, roll no., college name and
        college rank");
    String name = sc.nextLine();
    int roll = Integer.valueOf(sc.nextLine());
    String cname = sc.nextLine();
    int crank = Integer.ValueOf(sc.nextLine());
    s[i].setStudent(name, roll, cname, crank);
}
```

```
sout ("Students whose name ~~starts~~ ends with 'ra' :")
for (int i=0; i<4; i++){
        if (s[i].name.endsWith ("ra")){
                s[i].displayRecord();
        }
}
sc.close();
}
}
```

Group D

(21) Can you imagine Java Program without this pointer? What is the role of this super keyword?

WAP to create following types of array and print as follows.

1
2 3 4
5 6

Yes, a Java Program can exist without using this pointer.

This keyword is used to refer the current object on behalf of which member method or constructor is called. It helps in differentiating between instance variables and parameters, when they both have same name.

```
Class Example {
    int value
    Example (int value) {
    this.value = value;
                 ↳ parameter
    }  instance variable
}
```

This is not needed when there is no ambiguity between variable and parameter.

The super keyword is used to refer to the parent class (superclass) from a subclass.
The roles of super keyword are:
i) To call the parent class constructor
ii) To call parent class method
iii) To access parent class variables.

i) To call the parent class constructor

```
class Parent {
    Parent() {
        sout("Parent constructor");
    }
}

Class Child extends Parent {
    Child() {
        super(); // calles Parent();
        sout("Child constructor");
    }
}
```

ii) To call parent class methods:

```
class Parent {
        void show() {
                sout ("Parent method");
        }
}

class Child extends Parent {
        void show() {
                super.show();
                sout ("Child method");
        }
}
```

iii) To access parent class fields

```
class Parent {
        int n = 20;
}
class Child extends Parent {
        int n = 40;
        void display() {
                sout (super.num + num); // 20+40=6
        }
}
```

```
class JaggedArray {
    public static void main(String[] args) {
        int[][] arr = {
                        {1},
                        {2,3,4};
                        {5,6};
                      };

        for (int i=0; i<arr.length; i++)
        {
            for (int j=0; j<arr[i].length; j++) {
                System.out(arr[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

(22) Differentiate between concrete class and abstract class.
Why the process of implementing multiple inheritance using concrete class is prohibited in Java, and how can it be solved?

Abstract class	Concrete class
A class declared using the abstract keyword that cannot be directly instantiated.	A class that can be fully instantiated.
Can have or cannot have abstract methods	Cannot contain abstract method.
An abstract class cannot be declared as final.	A concrete class can be declared as final.

In Java, Multiple inheritance means that one class tries to inherit features from more than one parent class.

It is not supported by Java using concrete classes because it leads to ambiguity and confusion (The Diamond Problem).

```
eg: class A {
        void show() {
            sout ("From A");
        }
    }
    class B {
        void show() {
            sout ("From B");
        }
    }
    class C extends A, B {  // not allowed
        // creates ambiguity
    }
```

Java introduced interfaces to achieve multiple inheritance without ambiguity.

An ~~inherited~~ interface only contains method declarations (no body), so there is no conflict in inherited code.

```
interface A {
    void show();
}

interface B {
    void show();
}

class C implements A, B {
    public void show() {
        Sout ("Defined in class C");
    }
}
```

Group A

(1) The primary reason for java being architechural neutral is due to bytecode. Bytecode is a highly optimized set of instructions that is executed on Java runtime environment known as Java Virtual Machine (JVM). Bytecode can run at any application having JVM. It is based on 'Write Once; Run Anywhere' mechanism. making it architechural neutral.

(2) Wrapper classes are the classes in Java that is used to objectify primitive data type integers. Every primitive type has its respective wrapper class.

int → primitive

Integer → wrapper.

(3)

Automatic type conversion. Implicit type casting can be done when data type having small size are to be converted into ~~diff~~ large sized data type.
It is done automatically in program.
There is no loss of value.

```
int a = 4
float n = a
```

(4)

Different types of comments:

i) Single line comment

```
// Enter single line comment
```

ii) Multi line comment

```
/* This is
   multiline
   comment */
```

iii) Documentation comment

```
/** @link
   @... */
```

this keyword in java is used to refer to the current object that is being referred by member method or constructor.

this keyword is used to remove ambiguity when the name of referiance variable and parameter passed is same

eg:
```
ABC {
    int a = 2;
    ABC (int a) {
        this.a = a;
          ↳ reference
```

Declaring any variable as final will restrice modification / changing of the variable. It is used to prevent data hiding.

eg:
```
final int a = 5; // cant be altered.
a = 3; // CTE
(wrong)
```

(7)

Method overloading is the process of declaring multiple method with same method name but different method signature (i.e different parameter).

It gives compile time Polymorphism.

(8)

Unchecked exception are those exceptions that inherits Runtime exception.
They are not checked at compile time.
eg: Arithmetic Exception
ArrayIndexOutOfBound Exception

(9)

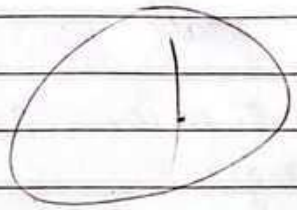
Switch case is multi branch statement

Syntax:

```
switch (expression) {
case 1:
        // statement if true
        break;
case n:
        // statement if matches
        break;
default:
        // 
}
```

10 Methods of Math class are:

Math.pow();
Math.sqrt();
Math.add();



Group B

(11)

```
Class TernaryDemo {
    public static void main(String[] args){
    int a=5, b=10, c=8;
    int large;
    large = (a>b)&&(a>c)?a:(b>c)?b:c;
                // ternary condition
    System.out.println("Largest number:"+large);
    }
}
```

Above code is the demo of Ternary conditional operator. It can be used as an alternative of if-else statement that helps to check condn at one line only.

Syntax:

(expression) ? value if true : value if false.

The above program checks and prints largest integer among a, b & c, ~~etc~~ using Ternary conditional operator.

Output:

Largest number: 10

2)

```
Class PalindromeArray {

    public static void main (String [] args)
    {
        int i;
        int a[] = {21, 22, 102, 202};
        System.out.println ("Palindrome numbers :");
        for (i=0; i<a.length; i++)
        {
            int n, rev;
            int n= a[i];
            while (n>0) {
                n= n%10;
                rev = rev*10 + n;
                n= n/10;
            }
            if (rev == n) {
                System.out.print (n + " ");
            }
        }
```

I/O:

```
Palindrome numbers :
22  202
```

(19) Encapsulation is the process of hiding / binding the data into single unit. It uses private and public access specifiers to achieve encapsulation.

```
class Encapsulation{
    public static void main(String[] args)
    {
        private int a; b // Here reference variables
        private int b;          are declared Private

        public void setA(int a){
            this.a = a;
        }
        public void setB(int b){
            this.b=b;
        }
        public void getA(){
            System.out.print(a);
        }
        public void getB(){
            System.out.print(b);
        }
        // Here, public member methods are used
        to access and set private variable
    }
}
```

Autoboxing and Unboxing are terms used in wrapper class.
Autoboxing is the automatic conversion of primitive data type into wrapper class.

eg:

```
Class Autoboxing {
    public static void main (String[] args)
    {
        int a = 5;
        Integer I1 = new Integer(a); //autoboxing
        Integer I2 = a;
        System.out.print("After boxing: "I2);
    }
}
```

Unboxing is the automatic conversion of wrapper class into primitive data type.

```
class Unboxing {
    public static void main (String[] args)
    {
        Integer a = 25;
        int i = a;      //unboxing
        System.out.print("After unboxing :" i);
    }
}
```

15

```
Class String {

    public static void main (String[] args)
    {

        String str[]= "ABA"

        String str[] = {"ABA", "ABC", "BDB"};

        for (String s : str ) {

        if (s.compareTo (startsWith(), endsWith()) == 0
        {

            Scout(s);
            System.out.println(s + " ");

        }
    }

    System.out.print ("These are string that
                       starts and ends with
                       same character");

}

}
```

Group C

```
Class Student {
    String name;
    int age;
    Student (String name, int age) {
        this.name = name;
        this.age = age;
    }
    void display() {
        system.out.println("Student with highest age");
        System.out.print(age);
    }
}

Class Demo {
    public static void main(String[] args)
    {
        Student s[] = new Student [10]

        System.out.print
        for (int i=0; i<10; i++)
        {
            system.out.print("Details of student"+(i+1));
            System.out.print("Enter name:");
            String name = sc.nextLine();
```

```
System.out.print("Enter age:");
int age = Integer.valueOf(sc.nextLine());

s[i] = new Student(name, age);
}

Student highest = s[0];

for(int i=0; i<10; i++)
{
    if(s[i].age > highest.age){
        highest = a[i];
    }
}

highest.display();
}
}
```

18

Multiple Inheritance can be achieved in java using interface. This is because in interface abstract method is declared (i.e method doesn't have body) due to which ambigiousity is prevented.

```
interface A {
        void display(); // abstract method
    }

interface B {
        void display(); // abstract method
    }

class C implements A, B {  // multiple inheritance implemented
        void display() {
            System.out.print("Multiple Interface Inheritance");
        }
}

class Demo {
    public static void main(String[] args) {
        C ob = new C(); // object created.
        ob.display();
    }
}
```

In above example, class C implements interface A and B. Hence multiple inheritance can be achieved using interface.

But, multiple inheritance cannot be achieved using class in java. This is because class consists of non abstract method, and multiple inheritance in this case, can result in ambiguousity.

```
class A {
        display A() {
                System.out.print("hi");
                int a=2;
        }
}
class B {
        display B() {
                System.out.print("bye");
                int b=4;
        }
}

Class C extends A, B {    // error
        // not possible, ambigious methods
}
```

	String Buffer	String Builder.
i)	Available since the earliest java version	Available since JAVA-5
ii)	Synchronization is done ie multiple thread cant be accessed at a time.	Synchronization isnt done, multiple thread can be accessed at a time
iii)	Due to synchronization, it is slower	Since synchronization isnt done, String builder is fastest.
iv)	Multi Thread safe buffer. environment is provided	~~Thread~~ Not multi thread safe.
v)	Multiple thread programs cannot be implemented at a time	Multiple thread programs can be implemented at a time.
(vi)	StringBuffer sb = new StringBuffer(5); sb.append=10;	StringBuilder sb = new StringBuilder(10); sb.append=20;

Group D

(21)

Exception is the event that disrupts the flow of the program. Exception is the object thrown in the program.

The process of handling the error or exception at runtime to minimize runtime error is known as exception handling.

Keywords used in exception handling are:

- throw exception
- try
- catch
- finally
- throws

① try and catch

=> try and catch is used at a same time simultaneously. try keyword in java throws exception, or event that can be wrong.
Catch keyword in java catches the exception thrown by try keyword.
There can be multiple catch for try keyword.
Nested try-catch is also possible.
Catch comes after try; it is mandatory.

② Finally

=> Finally keyword is used to execute necessary program, code or test. Finally can come after try-catch keyword.
The program written under finally keyword is executed no matter what.
Even after try-catch exception is handled, finally exception executes.

Try-catch and Finally Demo

```
class Demo{
    public static void main (String[] args)
    {
        try {
            int b=5/0; //exception thrown
            System.out.print("Exception is thrown");
        } catch (Exception e) {
            sout ("Arithmetic exception catched");
        }
        finally {
            System.out.print("finally is executed
                no matter what");
        }
    }
}
```

(3) Throw and Throws

Throw keyword is also known as userdefined/ customized exception. It is explicitly thrown by user when the probability of exception occuring in program is high. From throw keyword userdefined exception related to even can be made and required message can be printed after catching the exception.

Throws keyword in java is used to declare exception in member method. It is also explicitly thrown by user when the probability of exception occuring in program is high.

```
import java.util.Scanner;
class Demo {
    public static void main (String[] args){
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter age:");
            int a= sc.nextInt();
            if (a<0){
                throw new AgeException()
            } else {
                system.out.print("Valid age");
            } catch (Exception e) {
                system.out.print(e);
            }
        }
    }
```

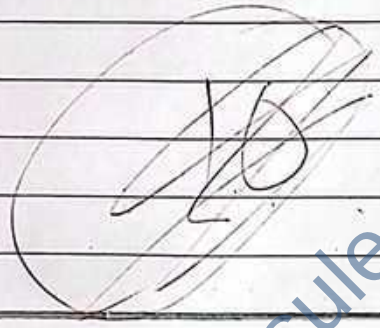
```
import java.util.Scanner;
class ExceptionMobile {

    public static void main(String[] args)
    {   long num;
        Scanner sc= new Scanner(System.in) {
        int n, c=0;
        System.out.print("Enter mobile number:");
        num= sc.nextLong();

        while (num>0) {
                num= num/10;
                c++;
        }
        if (c!=10) {
                throw new NotInRangeException() {
                    System.out.print("Thrown")
            } else {
                    System.out.print("You'll be called later
                                      in this number");

        catch (NotInRangeException e)
                System.out.print(e);
        }
}
```

```
Class NotInRangeException extends Exception {
    public String toString() {
        return ("Invalid mobile number");
    }
}
```



Inheritance is

Demo of Throw

```
import java.util.Scanner;
Class Demo5
        public static void main (String[] args)5
        Sout ("Enter age:");
        Scanner sc= new Scanner(System.in);
        int a= sc.nextInt();
        If (a<0) 5
          throw new AgeException()
        } else 5
          System.out.print ("Valid");
        }
        catch(AgeException e) 5
                System.out.print (e);
        }
}
```

10

22 Inheritance is the OOP principle that states that sub class / child class inheritances features, variables and members as from their respective super class / parent class.

Inheritance gives practical application in Java. Benefits of using inheritance are:

i) Reusability → Features, member methods and variables used / defined in super class can be reused in sub class easily. This helps in optimization of resources.

ii) Extendability → Child class not only inherits the features of parent class, but it can also extend the methods used in superclass. Multiple sub class ~~can~~ can extend a superclass.

Uses of super keyword in java are

i) It can be used to directly access parent variable used in parent method. This reduces ambiguity when ~~overriding~~ of variable is found in both super and sub class.

ii) Super keyword can also be used to access method from parent class.

```
class Employee {
        int sal = 5000;
    }
class Programmer extends Employee {
        int sal = 7000;
        void Display() {
            int Total = sal + super.sal;
            System ("Total salary" + Total);
        }
    }
class Demo {
        public static void main (String[] args)
        {
            Programmer P = new Programmer();
            P.display();
        }
    }
```

Output

Total Salary = 12000.

```
import java.util.Scanner;
abshact class Shape {
        int
    abstract getArea(); // abshact method
}

Rectangle extends Shape {
        int get_Area() {  //overriden
        Scanner sc = new Scanner (System.in);
        Sout ("Enter length and breadth:");
        int l = sc.nextInt();
        int b = sc.nextInt();

        int a = l*b;
        System.out.print ("Area of Rectangle:" a );
    }

}

Square extends Shape {
        int get_Area() {
        Scanner sc= new Scanner (System.in);
        Sout ("Enter length of square:");
        int l= sc.nextInt();
        System.out.print ("Area square:" (l*l));
    }
}
```

```
class Demo {
    public static void main(String[] args)
    {
        Square s = new Square();
        s.get_Area();
        Rectangle r = new Rectangle();
        r.get_Area();
    }
}
```

Pre Board. 2023
SDC

Group 'A'

(1) Write a command to compile the java code.

⇒ To compile a Java program, we use javac command followed by file name.

eg: javac Myprogram.java

(2) Differentiate between while and for loop.

	For loop	While loop
(i)	used when the no. of iterations is known	used when no. of iterations is unknown and loop continuous until specific condn is met.
(ii)	Initialization done often in the loop header.	Initialization before the loop starts

(iii)	Syntax:
	for(initialization, condn, inc/dec)
	{
	statement;
	}
		Syntax:
		while (condn) {
		Statements;
		inc/dec;
		}

(3) What are the rules for automatic type conversion?

=> When a value of a smaller or less precise type is assigned to a variable of a larger or more precise type, it is converted automatically.

eg: int a=5;
float b=5

(4) How can you declare array using java?

datatype[] arrname;
arrname = new datatype[size];

OR.

dataType[] arrname = new dataType[size];

(5) What is the use of super keyword?

=> Super keyword is used to:

(i) Access parent class fields.
(ii) Calling parent class methods
(iii) Calling parent class constructor

(6) What is bytecode?

=> Bytecode is a highly optimized set of instructions designed to be executed by Java runtime system which is called JVM which is platform independent that makes running a program in wide variety of environments i.e Windows, Linux, etc.

(7) What is method overriding?

=> Defining same method in parent as well as a child class is known as method overriding. It is done to provide own implementation in the child class for the method that have already been specified in the parent class.

(8) Differentiate & and | bitwise operators?

Bitwise AND (&)	Bitwise OR (|)
Sets a result bit to 1 only if both corresponding bits of the operands are 1.	Sets a result bit to 1 if at least one of the corresponding bits of operands is 1.

(9) What is the use of generic in java?

=> Generics allow us to write code that works with different data types using a single class, interface or methods. Instead of creating separate version of each type, we use type parameters like <T> to make code reusable and type safe.

(10) How can you take the string input in java using Scanner class.

=> To take a string input using the Scanner class, we use nextLine() method.

eg: String str = sc.nextLine(); // Reads entire line

Group B

(11) WAP to print the palindrome numbers from 10 to 100.

```
class Palindrome {
    public static void main(String[] args) {
    System.out.print("Palindrome numbers are:");
    for(int i=10; i<=100; i++){
        int num = i;
        int rev = 0;
        int temp = num;
        while(temp>0) {
            int digit = temp%10;
            rev = rev*10 + digit;
            temp /= 10;
        }
        if(i==rev) {
            System.out(i);
        }
    }
}
```

(12) WAP to print the sum of numbers of an array

```
Class ArraySum
{
    public static void main (String[] args)
    {
        Scanner sc = new Scanner;
        sout ("How many numbers ?");
        int n = sc.nextInt();
        int [] arr = new int [n];
        sout (" Enter numbers: ");
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
            sum += arr[i];
        }
        sout ("Sum is:" + sum);
    }
}
```

(13) WAP to create a generic constructor in java.

```
class GenConstructor {
    public <T> GenConstructor(T a, T b) {
        sout(a);
        sout(b);
    }
}
class Demo {
    public static void main(String[] args)
    {
        GenConstructor g = new GenConstructor(1,2);
    }
}
```

(4) Difference between & and && operator.

	&	&& (short circuit operator)
Operator	It is a "Bitwise Operator"	Logical AND operator
	It evaluates both left and right side of the expression.	It only evaluates the left side of the expression.
	Operates on "Boolean datatype" as well as operates on "bits."	It operates only on "Boolean datatype"
	Always checks both sides, even if the first one is false.	Stops early if the first condn is false

	int a=5, b=10;
	if (a>10 & b++>5) {
	sout ("True");
	}
	sout (b); // 11
		int a=5, b=10;
		if (a>10 && b++>5)
		{
		sout ("True");
		}
		sout (b); // 10

(15) WAP to print the fibonacci numbers using recursion.

```
public class Fiboman {
    public static int fibo(int n) {
        if (n==0) {
            return 0;
        } else if (n==1) {
            return 1;
        } else {
            return fibo(n-1) + fibo(n-2);
        }
    }
    public static void main (String[] args) {
        Scanner sc = new Scanner (System.in);
        int a = sc.nextInt();
        for (int i=0; i<a; i++)
        {   int result = fibo (a);
        }
    }
}
```

16) WAP to print the name of the animal whose name starts with character 'a' from the string array.

```
class AnimalName {
    public static void main(String[] args)
    {
        String[] animal = {"Ant", "dog", "ape", "Asp"};
        for (String str : animal) {
            if (str.toLowerCase().startsWith("a")) {
                System.out.println(str);
            }
        }
    }
}
```

Group C

(17) Create a class Currency with member variables Rs and paisa. Create an array to hold 10 currency objects and print Currency amt with highest value.

```
Class Currency {
        int rs;
        int paisa;

        void setData(int rs, int paisa){
         this.rs = rs;
         this.paisa = paisa;
        }

        int getTotal(){
          return rs*100+paisa;
        }

        void printInfo(){
          sout("Rs" + rs + "and" + paisa + "paisa");
        }
}
```

```
public class Main
{
    public static void main (String[] args)
    {
        Scanner sc = new Scanner (System.in);
        Currency [] c = new Currency [10];

        for (int i=0; i<10; i++) {
        c[i] = new Currency ();
        sout ("Enter Rs for currency + (i+1)+":");
        int rs = sc.nextInt ();
        sout ("Enter Paisa for currency " + (i+1));
        int paisa = sc.nextInt ();
        c[i].setData (rs, paisa);
    }

Currency highest = c[0];
for (int i=1; i<10; i++)
  if (c[i].getTotal () > highest.getTotal ()) {
        highest = c[i];
    }
    highest.printInfo ();
}
}
```

~~Sout ("Curr~~

(18) WAP to demonstrate the use of nested try-catch.

```
class NestedTryCatch{
    public static void main (String[] args){
    int a=7;
    int[] arr={2,7,8};
try{
    try{
        sout(a/0);
    }
    catch (ArithmeticException e){
        Sout ("Inner exception caught");
    }
    try{
        Sout (arr[4]);
    }catch (ArrayIndexOutOfBoundsException e){
        Sout ("Inner exception caught");
    }
}
catch (Exception e){
        Sout ("General outer exception caught")
}
}}
```

19) WAP to demonstrate the dynamic method dispatch.

```
class Vehicle {
    public void run() {
        sout ("Running");
    }
}
class Splender extends Vehicle {
    public void run() {
        sout ("Running @ 40 kmph");
    }
}
class Avenger extends Vehicle {
    public void run() {
        sout ("Running @ 60 kmph");
    }
}
class DMA {
    public static void main (String[] args) {
        Vehicle v;
        v = new Splender(); // obj of a child class
        v.run();
        v = new Avenger();
        v.run();
    }
}
```

(20) WAP to write some text into file. Then read the content of the file and print number of characters it contain.

```
import java.io.*;
import java.util.Scanner;

class FileDemo {
    p s v m (s[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            sout ("Enter some text:");
            String text = sc.nextLine();

            FileWriter fw = new FileWriter("file1.txt");
            fw.write(text);
            fw.close();

            FileReader fr = new FileReader("file2.txt");
            int ch, c = 0;
            sout ("File content:");
            while ((ch = fr.read()) != -1) {
                sout ((char)ch);
                c++;
            }
            fr.close();
```

```
sout("no of characters:" + c);

} catch (Exception e) {
        sout(e);
}
sc.close();
}
}
```

Group D

(21) Explain OOP with suitable example? How can you achieve multiple inheritance in java?

=> OOP in java is based on four main principles:

1. Polymorphism
2. Encapsulation
3. Inheritance
4. Abstraction

(1) Encapsulation

The process of binding variables/data and methods into a single unit and controlling access with access modifiers.

The data is made private and public methods are created to work on two data, and getters and setters are used.

eg:

```
public class Encapsulation{
    private int a,b;
    public void setA(int a){ // setter methods.
        this.a = a;
    }
    public void setB(int b){
        this.b = b;
    }
    public int getA(){ // getter methods
        return a;
    }
    public int getB(){
        return b;
    }
}
public class Test{
    psvm (String[] args){
        Encapsulation ob = new Encapsulation();
        ob.getA(10);
        ob.setB(20);
        sout(ob.getA());
        sout(ob.getB());
    }
}
```

(2) Inheritance

=> The process that allows a child class to acquire and inherit the properties and methods of another parent class

eg:
```
class Vechile {
        void start () {
                sout ("Started");
        }
}
class Car extends Vechile {
        void honk () {
                sout ("honks");
        }
}
class Test {
        psvm (S[] args) {
        Car c = new Car ();
        c.start (); // inherited
        c.honk ();
        }
}
```

(8) Polymorphism

⇒ It allows object to take many forms.
They are of two types:
- Compile-time (method overloading)
- Run time (method overriding)

eg: Runtime polymorphism.

```
class Animal {
        void sound() {
                sout ("Animal makes sound");
        }
}

class Dog extends Animal {
        void sound() {
                sout ("Dog barks");
        }
}

public class Test {
        p s v m (String[] args) {
        Animal a = new Dog(); //runtime polymorphism
        a.sound();
}}
```

4) Abstraction

Hiding implementation details and showing only the functionality to the users using abstract class ; or interface.

eg:

```
abstract class Shape {
        abstract void draw();
}

class Circle extends Shape {
        public void draw() {
                System.out.println("circle drawing.");
        }
}

public class Test {
        public static void main (String[] args) {
                Shape s = new Circle();
                s.draw();
        }
}
```

The user doesnt need to know how Circle is drawn internally ; just calls draw().

Part 2

Java doesnot support multiple inheritance using classes to avoid ambiguity (Diamond Problem) which arises when a class inherits from two ~~To solve it~~ parent classes.

Java provides interfaces as alternative. Interfaces allow a class to implement multiple behaviours without inheriting from multiple classes. This solves ambiguity since interfaces only define method signatures.

eg:

```
interface A {
        void methodA();
}
interface B {
        void methodB();
}
class C implements A, B {
        public void methodA() {
                sout("method A");
        }
        public void methodB() {
                sout("method B");
        }
}
```

```
public class main {
    public static void main(String[] args)
    {
        Class ob = new C();
        ob.methodA();
        ob.methodB();
    }
}
```

Output : Method A
Method B

(22) What is package? List some built in packages. Create a package calculator having a class Calculator in it. This class has a method that accepts two integer values and return their sum. Now use this package from another class Myclass in package 'Mypackage' to create obj of calculator, perform addition by calling method and display sum.

=> Package is a name space in which classes, interfaces, and sub-packages are stored, making it easier to manage and organize code.
→ They help avoiding name conflicts
→ control access
→ improve code maintainability.

Built in packages are the predefined libraries provided by the Java Development Kit that contain large number of classes and interfaces to perform common tasks such as i/o operations, data structures, networking, and more.

These packages offers ready-to-use functionalities.

- java.sql
⇒ Provides classes for accessing and processing database.
connection.class, DriverManager, etc.

- java.lang
⇒ contains classes and interfaces that are fundamental to the design of Java programming language.
String, StringBuffer, System, Math, Integer

- java.util
⇒ contains utility classes such as collections, date/time, array manipulation tools.
Calender, ArrayList, TimeZone.

- java.io
⇒ for handling system operations and file handling classes

- java.net
⇒ for implementing network applications eg URL class

- java.awt
⇒ for painting graphics and images.
Color, Font, Graphics.

Part 2

File : Calculate.java

```
package Calculate;

public class Calculator {
        public int add(int x, int y) {
                return x+y;
        }
}
```

File : MyClass.java

```
package myPackage;
import Calculate.Calculator;

public class MyClass {
        public static void main(String[] args) {
                Calculator c = new Calculator();
                int sum = c.add(5,25);
                System.out.println("Sum = " + sum);
        }
}
```