Section A Java

⇒ Decision making :-

1. If Statement :

- It is used When we want to test a Condition.

Syntax : If (condition)
{ }

Program :

```
import java.util.Scanner;
Class If {
{
public static Void main (String [] args) {
int age; System.out.println ("Enter Your Age!");
Scanner x = new Scanner (System.in);
   age = x.nextInt ();
   if (age >= 18)
{ System.out.println ("Eligible for Vote..!");
   }
}
}
```

Output

```
Enter Your Age!
19
Eligible for
Vote.
```

2. If else :

- It is used when we want to test a two conditions either if condition or else statement for a single condition.

Syntax :
```
If (condition)
{ }
else
{ }
```

Program :

```
import java.util.Scanner;
Class IfElse {
Public static void main (String [] args){
int age;
System.out.println ("Enter Your Age :- ");
  Scanner s = new Scanner (System.in);
    age = s.nextInt();
if (age >= 18){
System.out.println (" Eligible for Vote -- "); }
else {
  System.out.println (" Not eligible for Vote -- "); }
} }
```

Output

```
Enter Your Age :- 19
Eligible for Vote
```

3. Else if ladder :

- It is used when we have only one if block and multiple if else if blocks and at last one else block.

Syntax :
```
If (condition)
{}
else if (condition)
{}
else if (condition)
{}
else
{}
```

Program :

```
import java.util.Scanner;
class ElseIf {
Public Static void main (String [] args) {
  int age;
  Scanner s = new Scanner (System.in);
     age = s.nextInt();
  System.out.println ("Enter Your Age");
     if (age <0) {
      System.out.println ("Invalid age"); }
```

```
else if (age >= 18){
 System.out.println("Eligible for vote"); }
else {
 System.out.println("Not eligible for vote"); }
 }
}
```

Output

Enter Your Age : -2
Invalid age

4. Nested if else statement:

- Whenever we define if else block, inside this if else block we define another if else block is called Nested if else statement

Syntax:-

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

Program

```
Class Nested
{
Public Static void main (String [] args){
   int a=10, b=20, C=30;
 if (a>b)
 {
  if (a>c)
 { System.out.println (a); }
 else
 { System.out.println (c);
       }
 }
 else {
  if (b>c)
  { System.out.println (b);
        }
  else{
   System.out.println (c);
        }
    }
  }
}
```

Output

30

5. Switch Statement :-

- Switch Statement is a multiple choice decision making selection statement, it is used when we want to select one case out of multiple cases known as Switch Statement.

Syntax :

```
Switch(Exp)
{
Case 1: Statement 1;
        break;
Case 2: Statement 2;
        break;
default : St;
}
```

Program :-

```
import java.util.Scanner;
class Switch {
public static void main(String[] args) {
int a=10, b=20, ch;
System.out.println("Enter user choice");
Scanner s = new Scanner(System.in);
ch = s.nextInt();
```

```
Switch (ch) {
Case1: System.out.print("Sum" + (a+b));
       break;
Case2: System.out.print("Subtract" + (a-b));
       break;
Case3: System.out.print("Multi" + (a*b));
       break;
Case4: System.out.print("Division" + (a/b));
       break;
default: System.out.print("Invalid Choice");
}
}
```

Output

Enter user choice - 1
Sum - 30

⇒ Looping Statement :-

- Whenever we have to repeat certain statement several times is called loop.

1. While loop :- While loop is a pre test loop. It is used when we don't know about the no. of iterations in advance.

- It is also known as entry control loop.

Syntax : while (condition)
{}

Program :-

```
class While {
public static void main (string [] args) {
  int n = 1;
   While (n <= 10)
   { System.out.println("Learn Coding \n");
       ++n;
     }
  }
}
```

Output
Learn Coding
Learn Coding
Learn Coding
Learn Coding
Learn Coding
Learn Coding
Learn Coding
Learn Coding
Learn Coding
Learn Coding
Learn Coding

2. Do While loop :-

- Do While loop is a pre test loop. It is used when we want to execute a loop body at once even condition is false.
- It is also known as exit control loop.

Syntax : do
{ }
While (condition);

Program :-

```
class DoWhile {
public static void main (String []args){
  int n=1;
   Do
   { System.out.print(n);
     ++n;
    }
  While (n<0);
  }
}
```

3. For loop : For loop is the most commonly used loop, it is

used when we want to perform intilization, condition, incr/decr operation in single line.

- It is also known as entry control loop.

Syntax :-

```
for (intialization ; condition; incr /decr)
  { }
```

Program

```
class Forloop {
Public static void main( String [] args) {
 for (int i = 1 ; i<=10 ; i++) {
  System.out.print (i + " "); }
 }
}
```

Output

1 2 3 4 5 6 7 8 9 10

4. For each loop :-

For each loop is mainly used to fetch the values from a collection like array.

Syntax :- for (datatype var1 : var2)

{ }

Program:-

```
Class ForEach{
Public static void main(String []args){
int a[] = {10, 20, 30, 40, 50};
for(int b: a){
    System.out.println(b + " ");
        }
    }
}
```

Output

10
20
30
40
50

5. Nested for loop:-

- A for loop which contains inside for loop is called nested for loop.

Syntax:
```
for(intialization; condition; incr/decr){
    for(intialization; condition; incr/decr){}
        }
```

Program :

```
class Nested{
public static void main (String[] args){
int i, j;
for (int i=1; i<=5; i++)
{
for (int j=1; j<=5; j++)
 {
  System.out.print ("*");
    }
  System.out.println();
 }
}
```

Output

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

⇒ Constructor :

- Constructor is a special type of method whose name is same as class name.
- The main purpose of constructor is to initialize the object.
- Every java class has a constructor.
- A constructor is automatically called at the time of object creation.
- A constructor never contain any return-type Including void.

Syntax :
```
class class-name
{
    class-name()
    { }
}
```

Types
1. Default Constructor
2. Copy Constructor
3. Parameterized Constructor
4. Private Constructor

1. Default Constructor : A constructor which donot have any parameter is known as default constructor.

Syntax :-

```
class A {
  A()
  { }
}
```

Program :

```
class A {
int a, String name;
 A(){
  a = 0; name = null; }
void show (){
 System.out.print (a + " " + name);
    }
}
class B {
Public static void main (String [] args) {
   A ref = new A();
     ref.show();
   }
}
```

Output

0 null

2. Parameterized Constructor :-

- A Constructor through which we can pass one or more parameter is called parameterized constructor.

Syntax :
```
class A{
A(int x ; string y)
{ }
}
```

Program :-

```
class A {
int x , y;
A() {
 x=a ; y=b ;
 }
void show() {
System.out.print (x + " " + y); }
}
class B {
public static void main (String [] args) {
   A q = new A (100, 200);
   q.show (); }
}
```

Output

100 200

3. **Copy Constructor :-** Whenever we pass object reference to the constructor then it is known as copy constructor.

Syntax :

```
class class-name
{
 Class-name (obj ref)
  { }
}
```

Program :

```
class A{
int a; string b;
A(){
a=10, b="learn coding";
System.out.println(a+" "+b);
  }
A(A ref){
  a=ref.a
  b=ref.b;
System.out.println(a+" "+b);
  }
}
class B{
```

```
public static void main (String [] args) {
    A a1 = new A();
    A a2 = new A(a1);
    }
}
```

Output

```
Io learn Coding
Io learn Coding
```

4. Private Constructor :

In Java it is possible to write a constructor as a private but according to the rule we can't access private member outside of class.

Syntax :

```
class class_name
{
    private class_name()
    {}
}
```

Program :-

```
class A {
    Int a; double b; String c;
    Private A() {
```

```
a=10; b=3.58 ; c = "Ishan" ;
System.out.println (a+""+b+""+c);
 }
Public static void main (String [] args) {
      A a = new A();
   }
}
```

Output

10 3.58 Ishan

⇒ Constructor Overloading :

Writing more than one constructor with different parameter is known as the constructor overloading.

Program :

```
class A {
int a ; double b; String c;
A() {
a=100, b=3.56 ; c = "AmKush" ; }
A(int x){
  a=x; }
A(double y, String z){
  b=y; c=z; }
}
class B {
```

```
Public static void main (String [] args){
    A r = new A();
    A r2 = new A(10);
    A r3 = new A(23.89, "Ankush");
System.out.println(r.a + " " + r.b + " " + r.c);
System.out.println(r3.b + " " + r3.c);
 }
}
```

Output

```
100 3.56 Ankush
10
23.89 Ankush
```

⇒ Method Overloading :-

- Whenever a class contain more than one method with same name but different types of parameters is known as method overloading.

Syntax: return_type method_name (para 1);
return_type method_name (para 1, para 2);

Program :

```
class A {
```

```
Void add(){
int a=10, b=20, c;
 c = a+b;
System.out.println(c);
}

Void add(int x, int y){
int c;
c=x+y;
System.out.println(c);
}

void add(int x, double y){
  double c;
  c=x+y;
  System.out.println(c);
  }
public static void main(String[]args){
    A a = new A();
     a.add();
     a.add(100,200);
    a.add(50, 4.5);
     }
}
```

Output

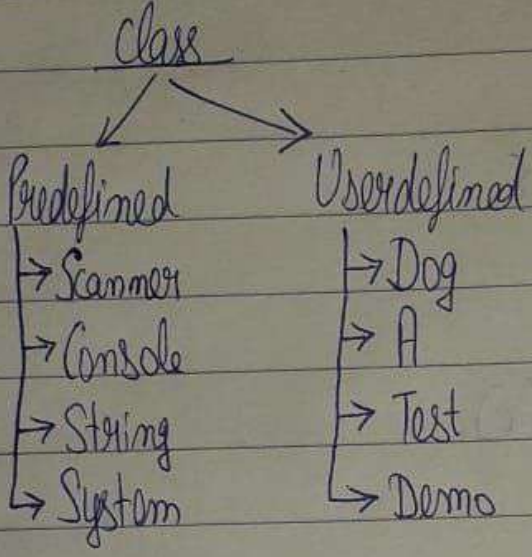
30

300

54.5

→ Class :

- Class is a collection of objects and it does not take any space on memory.
- Class is also called as blueprint.



Userdefined class - A class which is created by java programmer is called user defined class.

Object :-

- Object is an instance of class that executes the class.
- Once the object is created it takes up space like other variables in memory.

Program :

```
class Demo {
int a = 10 ; String b = ankush ;
```

```
void show (){
System.out.print(a + " " +b); }
}
class Test{
public static void main( String [] args){
    Demo x = new Demo();
    x.show();
  }
}
```

Output

lo ankush

⇒ Introduction to Java:

- Java is a class based object oriented programming language which is developed by James Gosling and his friends in the year 1991.

- The purpose of java is to develop is write ones run any where and it is a platform independent language.

- The first version of Java (JDK 1.0) was released on the year 23rd Jan, 1996 by Sun microsystem.

Syntax :-
```
class class-name {
    Public static void main (string [] args) {
        System.out.print (" ");
    }
}
```

Rule –
1. Save → class-name.java
2. Compilation → Javac class-name.java
3. Execution → Java class-name

Program :
```
class A {
    Public static void main (String [] args) {
    System.out.print ("Welcome to Coding World "); }
}
```

Name of the Chapter _____ Date _____

History of Java :

- Java is a totally computer based programming language developed by Sun microSystem (James Grosling, Mike Sheridan & Patrick naughton).

- In the year 1991 James Grosling & his friend start a project
  Team → Green Team

- James Grosling → greentalk
  Extension → .gt

- Greentalk → oak
- The project was originally called oak designed for embedded systems and consumer electronic devices.
- Later on it was renamed Java (after Java coffe) and officially released in 1995.

- Oak
  ↙ ↘
  National tree (USA, France etc) — Oak technology

- The main goal of Java was Write once Run Anywhere meaning a program written in Java can run on any Syste

that has the Java Virtual Machine (JVM).

- Java → Setbox, television, Remote etc
- Java → Internet programming

→ Features of Java :-

① Object oriented programming :

- Java follows the principles of object oriented programming language which means that everything in java has an object that has data and behaviour.

- The four main pillars of oops are encapsulation, Inheritance Polymorphism and Abstraction.

Example :
Imagine you're building a ride booking app like Ola and uber. You can create car class ~~properties~~ with properties like color, speed and methods like start() and stop.
From this class you can create multiple objects like
Car c1 = new Car();

2. Platform Independent :- Java code write once, run anywhere.

Java source code compiled into byte code using the Java Compiler.

- This bytecode is not specific to any machine and can run anywhere on any system using JVM machine (Java Virtual Machine).

- Example :- If you develop a billing system in Java on a Windows Computer. You can run the same program on linux or MAC machine without rewritting the code. Just ensure that JVM is installed.

3. Simple language :-
- Java is designed to be easy to use and understand.
- It removes the complex features from C++ like pointers, operators overloading and multiple inheritance through classes, making it easier for beginners to learn.

- Example : Creating a program to calculate the area of a rectangle is straight forward in Java. You don't need to worry about managing memory or writing complex Syntax.

4. Secure :
- Java provides strong security features like bytecode verification, exception handling and no explicit memory access.

- It runs program in a controlled environment, preventing harmful code from affecting your System.

- Example :- Online banking applications and android apps use Java because it is difficult for hackers to misuse memory.

5. Portable :-

- Java programs are portable, meaning they can run any hardware or software environment without modification.

- Since java compiles to byte code and JVM available on many platforms, the same code can move easily across devices.

- Example : If a company creates a Java based employee management System, the same System can be copied to multiple branch offices with different computer and still work perfectly.

6. Compiled and Interpreted :-

- Java combines the benefits of both compiled and interpreted languages.

- First Java Codes is compiled into byte code and then the JVM interprets the byte code into machine language at runtime.

Example :

When you write a Hello World program in Java, the compiler turns it into bytecode. When you run it the JVM reads and executes it line by line.

⇒ Comparison b/w C, C++ and Java :-

	C	C++	Java
1.	It is a platform dependent language.	1. It is also a platform dependent language	1. It is platform independent language.
2.	It is a procedural oriented language.	2. It is a object oriented language.	2. It is a pure object oriented language.
3.	It supports pointer.	3. It supports pointer.	3. It does not support pointers.
4.	It does not support database connectivity.	4. It does not support database connectivity.	4. It supports database Connectivity.
5.	It does not support multithreading & Interface.	5. It also does not support multithreading & Interface.	5. It supports multi threading & Interface.
6.	It does not support Inheritance	6. It supports single, multilevel and multiple Inheritance support	6. It supports single, multi level & heirarichal Inheritance support

7.	It is compiler based language	7. It is also compiler based language	7. It is compiler based and Interpreter based language.
8.	In C file is saved with extension. C	8. In C++ file is saved with extension .C++	8. In Java file is saved with extension .Java
9.	In C their are three types of loops.	9. In C++ their are also three types of loops.	9. In Java their are four types of loops.

⇒ Datatypes :-

- In java, a datatype defines the kind of data a variable can hold.

- It tells the compiler :-

(1) What type of data is being stored like a number, character or text.

(2) How much memory to allocate.

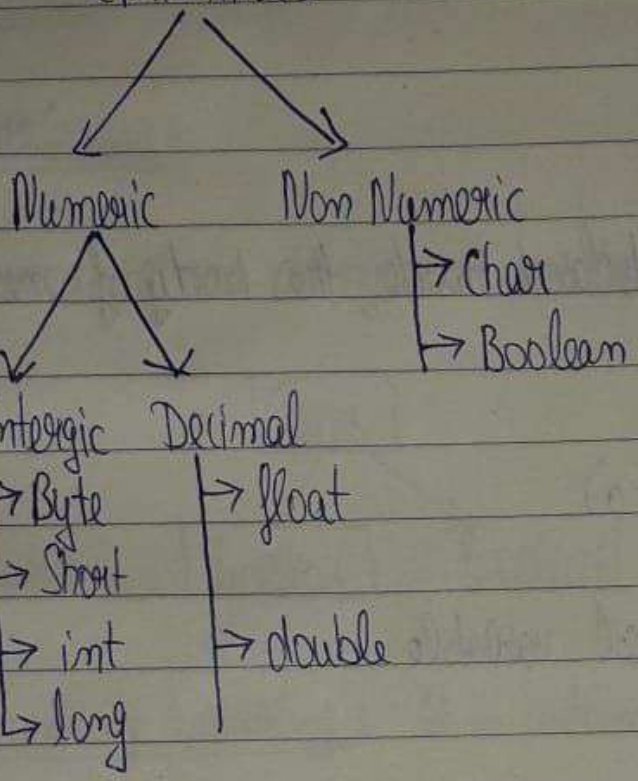
(3) What operations are allowed on that data.

- Datatype specifies the different sizes and values that can be

Stored in the variable.

- There are two types of datatype are :

① Primary (Primitive)



② User defined (Non-Primitive)

- → Class
- → Interface
- → Arrays
- → String

⇒ <u>Variables in Java</u> :-

- Variable is the name of memory location.
- In other words we can say it is a user defined name which is given by user.

- Each variable has -
  ① A type → What kind of data it can hold.

2. A name → a label you use to access that data.
3. A value → the actual data stored in it.

- Variable can store any type of values.

Types of variable :

1. Local variable :

- A variable which is declared inside the body of method is called local variable.

Syntax :-
```
void fun(int a)
{
  int x; // Local variable
}
```

2. Instance Variable :

- A variable which is declared inside the class but outside of all the methods called instance variable.

Syntax :-
```
class A{
  int a; // instance variable
  public(){}
}
```

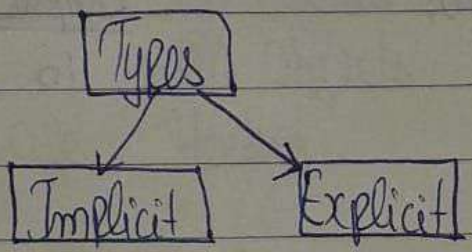
3. Static Variable :-

- A variable which is declared with the help of static keyword called static variable.

Syntax :- static int x;

⇒ Type Casting :-

- Converting one datatype to another datatype is known as type casting



1. **Implicit typecasting** : It is automatically performed by the compiler.

Program

```
class A{
Public static void main (String []args){
   int a=10;
   double b = a;
   System.out.print(b);
   }
}
```

Output

10.0

2. Explicite typecasting :-

- By default the compiler, does not allow the explicit typecasting.

Program

```
Class A {
public static void main (String [] args) {
  double a = 10.5;
  int b = (int) a;
  System.out.print(b);
  }
}
```

Output

10

⇒ Tokens :-

- Tokens is the smallest elements of a program that is identified by a compiler.
- The compiler reads your program token by token to understand what you wrote.
- Every java statements and expressions are created using tokens.

Types

① **Keywords** : These are reserved words that have special meaning.

example - int, class, void, public

② **Identifiers** : Names given to -

① variables
② methods
③ classes
④ Objects

Eg - age, Student

③ **Literals** : Fixed values written directly in code.

Eg: 10
"Ayan"
true

④ **Operators** : Symbols that perform operations.

Eg:- + - * / % = == >

⑤ **Seperators** : characters that separate code elements.

Eg:- (), {}, [], ;, .

⇒ Keywords:

- Keywords in java are special reserved words that have predefined meanings in the language.
- You cannot use them as a variable names, class names or methods names.

Keywords

int
if
else
switch
for
while
do

⇒ Identifiers:

- In Java, an identifier is the name of the variable, method, class, package or interface that is used for the purpose of identification.
- Is known as identifiers.

Note : (1) Keyword can't be used as a identifier.
(2) Identifier are case sensitive
(3) We can't use whitespace in b/w identifier.
(4) Identifier always starts with letters $ or _.

Section-B

→ Interface :

- Interface is just like a class, which contains only abstract method.
- To achieve interface java provides a keyword called implements.
- Interface methods are by default public & abstract.
- Interface variables are by default public + static & final.
- Interface method must be overriden inside the implementing class.
- Interface is nothing but deals between client and developer.

Program

```
import java.util.Scanner;
interface Client {
  Void input();
  Void output();
  }
class Raju implements Client {
  String name, double Salary;
public void input(){
 Scanner s = new Scanner(System.in);
    name = s.nextline();
System.out.print("Enter Username:");
    Salary = s.nextDouble();
```

```
System.out.println("Enter Salary:");
}
Public void output() {
System.out.println(name + " " + Salary);
}
public static void main(String[]args) {
    client r = new Raju();
      r.input();
      r.output();
    }
}
```

Output

Enter Username: Ankit
Enter Salary: 50.35
Ankit 50.35

⇒ Interface Variable :

```
interface CustomerRaj {
  int amount = 5;
  Void purchase();
}
Class Seller Ram implements Raj {
@Override
```

```
public void purchase(){
System.out.println("Raj needs " + amount + " Kg rice");
 }
}
class Check{
public static void main (String [] args){
  Customer Raj r = new Seller Ram();
   r.purchase();
   System.out.println(customer Raj.amount);
  }
}
```

Output

Raj needs 5 Kg rice
5

⇒ Interface method :

```
interface client{
void webdesign();
void webdeveloper();
 }
abstract class RajTech implements client{
@Override
 public void webdesign(){
```

```
System.out.println("Green, navbar");
 }
}

class RahulTech extends RajTech {
@Override
 public void webdeveloper() {
System.out.println("Html, CSS, Javascript");
 }
}
class check{
public static void main (String [] args){
     RahulTech r = new RahulTech();
     r.Webdesign();
     r.Webdeveloper();
 }
}
```

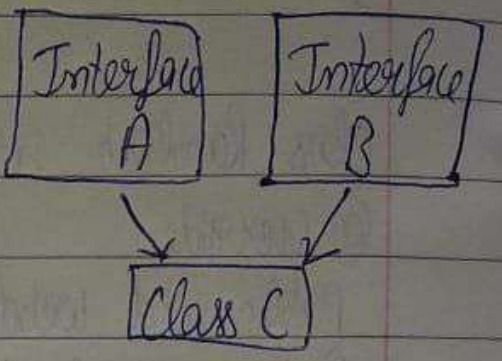
Output

Green, navbar
Html, CSS, Javascript

⇒ Multiple Inheritance through interface : We can achieve multiple inheritance through interface because interface contains only

Only abstract method, which implementation is provided by the sub classes.

Note: Class C extends A, B ✗



Class C implements A, B ✓

Program :-

```
Interface A{
 Void show(); }
interface B{
Void show();
 }
class C implements A, B{
Public void show(){
System.out.println("Interface A & B"); }
Public Static void main(String[] args){
     C r = new C();
      r.show();
  }
}
```

Output

Interface A & B

→ Interface JDK 1.8 (Default method):-

- Before JDK 1.8 interface can only have abstract methods and all the abstract methods of interfaces must be overriden in implementing class as well as methods are public & abstract by default.

Program:

```
interface A{
void a1();
void a2();
default void a3(){
 System.out.println("may or may not override in implementing
                                    classes");
  }
}
class B implements A{
  Public void a1(){
  System.out.println("Class B a1()");
        }
  Public void a2(){
    System.out.println("Class B a2()");
  }
}
```

```
Class C implements A {
Public void a1(){
 System.out.println("class C a1()");
   }
Public void a2(){
  System.out.println("class C a2()");
   }
}

Class D implements A{
public void a1(){
  System.out.println("class D a1()");
   }
Public void a2(){
  System.out.println("Class D a2()");
   }
}

Class Main{
Public static void main (String [] args){
     B b = new B();
     b.a1();
     b.a2();
     b.a3();
    C c = new C();
```

```
c.a1();
c.a2();
c.a3();
D d = new D();
 d.a1();
 d.a2();
 d.a3();
 }
}
```

Output

Class B a1()
Class B a2()
may or may not override in implementing classes
Class C a1()
Class C a2()
may or may not override in implementing classes
Class D a1()
Class D a2()
may or may not override in implementing classes

→ Interface JDK 1.8 onwards (Static method) :-

- From JDK 1.8 onwards interface can have default & static method.

Program:

```
interface A {
public static void show()
{
 System.out.println("Can't override interface static methods");
 }
}
class Test{
public static void main(String[] args){
    A.show();
   }
}
```

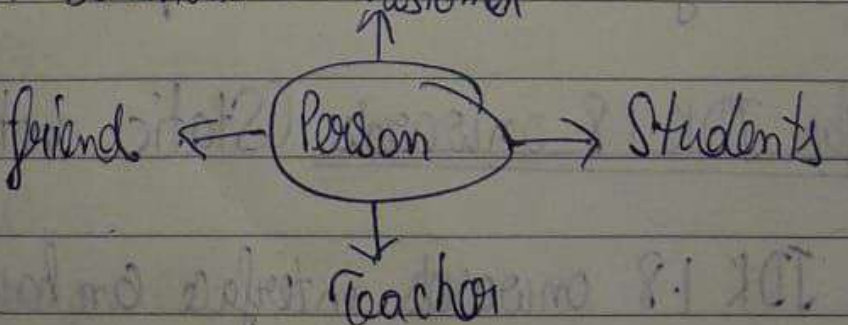
Output:

Can't override interface static methods

⇒ Polymorphism:

- Polymorphism is the greek word whose meaning is same object having different behaviour.

Example:



Types

→ Compile Time Polymorphism → A polymorphism which is exists at the time of compilation is called compile time or early binding or static polymorphism.

→ Runtime Polymorphism → A polymorphism which exists at the time of execution of a program is called runtime polymorphism.
Eg :- method overriding

→ Encapsulation:

- Encapsulation is a mechanism through which we can wrapping the data members and member methods of class in a single unit called encapsulation.

- It declare the class variable as a private.
- It declare the class methods as a public

Eg: class is the best example of encapsulation.

⇒ Difference between interface & abstract class:

Interface	Abstract class
1. Interface contains only abstract method.	1. Abstract class contains both abstract & non abstract method.
2. It supports multiple inheritance	2. It does not support multiple Inheritance.
3. By default interface methods are Public abstract.	3. There is no restriction on abstract class method modifier.
4. By default interface variables are public + static + final.	4. No need to declare variable as a public + static + final.
5. Interface Keyword is used to declare interface.	5. Abstract Keyword is used to abstract class.
6. Interface does not allow to declare a constructor.	6. Abstract class cannot declare constructor.

⇒ Array :

- Array is an object in java, which contains similar types of data in a contiguous memory location is known as

array.

Syntax : datatype var-name [];

1. One dimensional array :-

- An Array having one index is called one dimensional array.
- All the elements of one dimensional array are grouped into single array and referenced with single index.

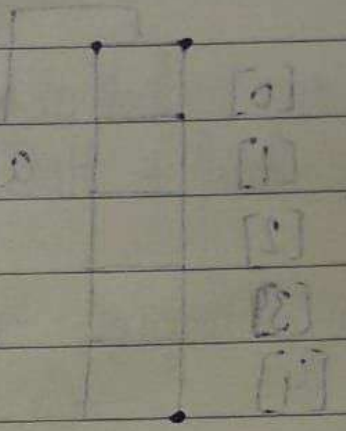
Program :-

```
Class Demo {
Public static void main (String [] args) {
int a [] = new int [5];
    a[0] = 10;
    a[1] = 20;
    a [2] = 30;
```

```
a[3] = 40;
a[4] = 50;
for (int i = 0; i<5; i++){
   System.out.println(a[i]);
  }
}
}
```

Output

```
10
20
30
40
50
```



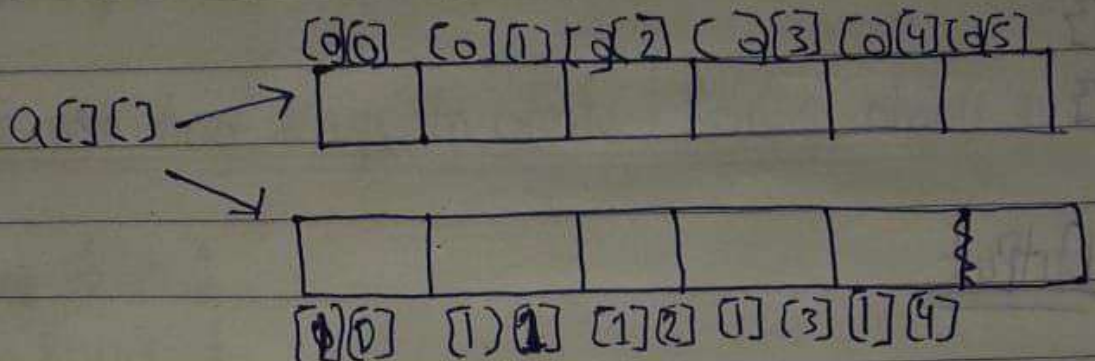
2. Two Dimensional array :-

- Two dimensional array is a type of multidimensional array in which there are two index values
  (1) One indicates to rows
  (2) Another indicates to columns

- Array having 2 index values is called two dimensional array.

- Arrays are stored in rows and columns i.e. matrix format.

```
a[] = new int [2][3];
```



Program:

```
import java.util.Scanner;
class Demo{
Public static void main(String []args){
  int a[][] = new int [2][2];
  Scanner sc = new Scanner(System.in);
   for(int i=0; i<2; i++){
   for(int j=0; j<2; j++){
    System.out.print("Enter array elements");
  a[i][j]= sc.nextInt();
    }
  }
System.out.print("Matrix:\n");
for(int i=0; i<2; i++){
 for(int j=0; j<2; j++){
```

```
System.out.print(a[i][j]+" ");
    }
  }
}
```

Output

Enter array elements

10
20
30
40

Matrix

10 20
30 40

⇒ String

- In Java Strings are objects that allows us to store sequence of character which contain alpha numeric values enclosed in double (may) quotes.

- Strings are immutable in java.
- It contains the methods that can perform certain operations on strings (concat(), equals(), length() etc.)

⇒ There are two ways to create String object :-

① String literal
② New Keyword

① String literal :

- String objects are stored in a special memory area known as string constant pool.
- Each time a string literal is created, the JVM checks the string constant pool.
- If the string is already exists in the pool, a reference to the pooled instance is returned.
- If string does not exists in the pool, a new string instance is created and placed in the pool.

Program :

```
class A{
```

```
Public static void main(String[] args) {
  String a = "ankit";      // literal
  System.out.print(a);
  String b = "ankit";      // literal
  System.out.println(b);
  a.concat("Kumar");
  System.out.println(a);
  }
}
```

Output

ankit
ankit
ankit

2. New Keyword :-

```
String a = new String("ankit");
  a.concat("Kumar");
```

Program

```
class A {
Public static void main (String [] args){
    String a = new String("ankit");
```

```
System.out.println(a);
String b = new String ("ankit");
System.out.println(b);
a = a.concat ("Kumar");
System.out.println(a);
}
}
```

Output

ankit
ankit
ankit Kumar

→ Vector :

- A vector in java is a legacy class introduced in jdk 1.0 before the collections framework.
- It is found in the package:
  import java.util.vector;

Properties :-

① Dynamic Array : It grows automatically when elements are added / removed.

Duplicates allowed → like array list, it can store duplicate elements.

Synchronized: All methods are Synchronized, which makes it thread safe, but slower compared to array list.

Syntax:-

```
Vector < Type > Vector = new Vector <> ();
```

Program:-

```
import java.util.Vector;
Public class Vector {
public static void main (String [] args){
  Vector < String > fruits = new Vector <> ();
    fruits.add("Apple");
    fruits.add ("Mango");
    fruits.add ("Banana");
    fruits.add ("Apple");
  System.out.println ("First fruit: " + fruits.get(0));
    for (String fruit: fruits)
{   System.out.println (fruit);
    }
    fruits.remove ("Banana");
System.out.println ("After removing Banana:" + fruits); }
}
```

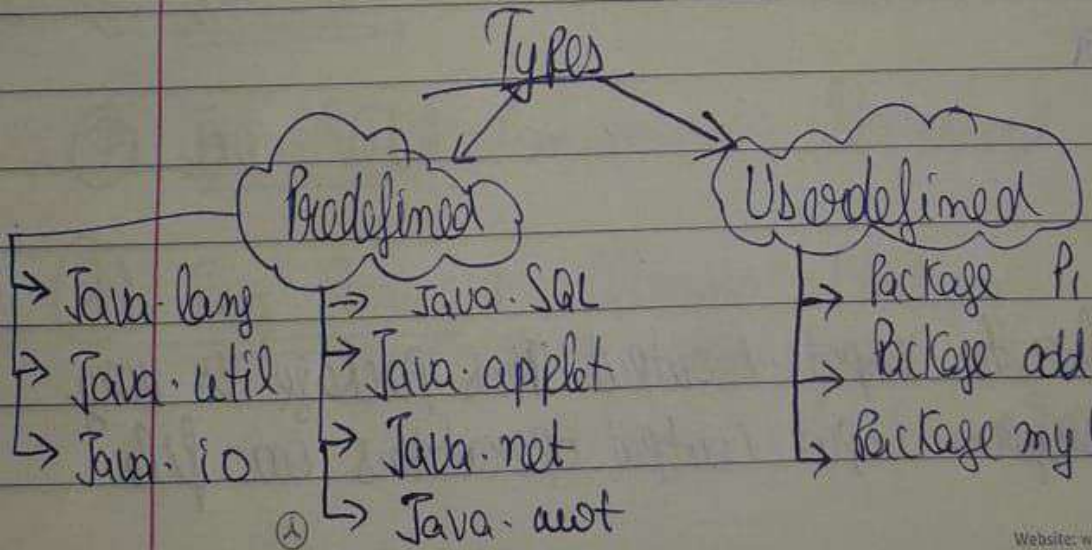
Output

First fruit: Apple
Apple
Mango
Banana
Apple

After removing Banana: Apple
Mango
Apple

⇒ Package :-

- A Package arrange number of classes, interfaces and sub package of same type into a particular group.
- Package is nothing but folders in Windows.



① Predefined Package:

- The package which are already created by java developer are called predefined package.

① Java.lang:

- It is the default package which also known as heart of java.
- Because without using this package we can't write even a single program.
- We need not to import this package.

Ex: Integer

② Java.util :-

- This package is used to implement data structure of java.
- It contains the utility classes also known as collection framework.
- Ex: Vector

③ Java.io :

- IO stands for the input / output. This package is very useful to perform input / output operations on file.

Ex: File, FileReader etc...

④ Java.applet:

- This package is mainly used to develop GUI related applications.
- Applet programs are web related program created at server but executed at client machine.

⑤ Java.awt:

- AWT stands for abstract window tool Kit.
- It is also used to developed GUI applications.
- The only difference between applet and awt is, awt programs are stand alone programs and it contains main().

Ex: Button

⑥ Java.net: URL, URL Connection

⑦ Java.SQL: Connection, Statement

⑧ Java.Swing: JFrame, JButton
JTextfield etc..

2. Userdefined Package

- The package which are created by user for their own purpose called user defined Package.

Syntax : Package Package_name;

Rules : ① Package statement must be first line of the program.

② The way of compilation of these classes would be different.

Program

```
Package Ram;
  Class A{
   Private void show ()
   { System.out.println ("Learn Coding");
     }
Public static void main(String [] args){
     A a = new A();
     a.show();
   }
}
```

Output

Learn Coding

⇒ **Inheritance:**

- Whenever we construct a new class from existing class in such a way that new class can access all the features & properties of the existing class is known as Inheritance.

- In Java extends keyword is used to perform Inheritance.
- It provides code reusability.
- We can't access private members of class through inheritance.
- Method overriding only possible through Inheritance.
- A sub class contains all the features of super class, so we should create the object of sub class.

Syntax:

```
class A
{ }
class B extends A
{ }
```

Types

1. Single Inheritance:-

- Single / Simple Inheritance is nothing but which contains only one super class and only one sub class is called Simple / Single Inheritance.

Program:

```
Class Student{
Int roll, marks;
String name;
void input(){
System.out.println("Enter Roll, marks & Name:");
  }
}

Class Ram extends Student{
void display(){
 roll = 1; marks = 89; name = "Ishan";
System.out.println(roll+""+marks+""+name);
 }
public static void main (String[]args){
    Ram r = new Ram();
      r.input();
      r.display();
  }
}
```

Output

Enter Roll, marks & Name:
1 89 Ishan

2. Multilevel Inheritance :-

- In Multilevel inheritance we only have one super class and multiple sub classes called multilevel inheritance.

Syntax :

```
Class A
{}
Class B extends A
{}
Class C extends B
{}
```

Program :-

```
class A{ int a, b, c;
void add(){
  a = 10; b = 20;
  c = a+b;
 System.out.println("Addition of two numbers: " + c);
 }
void sub(){
   a = 20 ; b = 10;
   c = a-b;
 System.out.println("Subtraction of two numbers: " +c);
 }
}
```

```
Class B extends A {
Void multi() {
  a = 10; b = 20;
  c = a * b;
System.out.println("Multiplication of two Numbers: " + c);
 }
Void div() {
   a = 100; b = 20;
   c = a / b;
System.out.println("division of two Numbers!" + c);
   }
}

Class C extends B {
  Void remainder() {
  a = 10;  b = 3;
   c = a % b;
System.out.println("Remainder of two Numbers: " + c);
  }
}

Class Test {
 Public Static void main (String [] args) {
   C r = new C();
r.add();      r.multi();      r.remainder();
r.sub();      r.div();          }
                             }
```

Output

30 -: Adelition of two numbers
1 -: Subtraction of two numbers
200 -: Multiplication of two numbers
5 -: Division of two numbers
1 -: Remainder of two numbers

3. Multiple Inheritance:

- Whenever a sub class wants to inherit the property of two or more super classes that have same method, java compiler can't decide which class method it should inherit.

- Then their might be a chance of memory duplication i.e. a reason java does not support multiple inheritance through classes.

4. Hierarchical Inheritance:

- A inheritance which contains only one super class and multiple sub class and all sub class directly extends super class called Hierarchical inheritance

Syntax: Class A
{ }

```
Class B extends A
 { }
Class C extends A
 { }
```

Program

```
Class A {
 void input() {
  System.out.println("Enter your Name: ");
   }
}

Class B extends A {
 void show() {
  System.out.println("My name is Ankush");
   }
 }

Class C extends A {
 void display() {
  System.out.println("My name is Ankita");
   }
 }
```

```
class Demo{
Public static void main(String[] args){
 B q = new B();
 C q2 = new C();
  q.input();
  q.show();
 q.inPut();
 q.display();
  }
}
```

Output

Enter your Name:
Ankush
Enter your Name:
Ankita

→ Method overriding:-

- Whenever we are writing method in super and sub class in such a way that method name and parameter name must be same is called method overriding.

Syntax:
```
class A{
    void show()
    {}
}
```

```
Class B extends A {
 void show()
  {}
}
```

Program

```
Class Shape{
 void draw(){
  System.out.println("Can't say shape Type");
    }
 }
Class Square extends Shape {
 @override
   void draw(){
  System.out.println("Square Shape");
    }
 }

Class Demo{
public static void main(String[] args){
   Shape s = new Square();
     s.draw();
    }
}
```

Output

Square Shape

Super Keyword:

Super Keyword refers to the objects of super class, it is used when we want to call the super class variable, method & constructor through sub class object.

- Whenever the super class and sub class variables and method name both are same than it can be used only.
- To avoid the confusion between super class and sub class variables and methods that have same name we should use super keyword.

Syntax:

```
Class A  //super
{
 A()
{}
}
Class B extends A{
 B()
{}
}
```

Program :-

```
Class A {
 Int a = 10;
  }
Class B extends A {
 int a = 20;
 Void show() {
 System.out.println(a);
 System.out.println(super.a);
   }
}
```

```
Class Test {
 Public static void main (String[] args) {
        B q = new B();
       q.show();
       }
  }
```

Output

20

10

→ Final Keyword :-

- Final is a modifier which provides restriction in Java we can use final in three ways :-

① Variable ② method ③ Class

① Final Variable :
- Once we declare a variable as a final we can't perform assignment.

Syntax : Final int A = 10;

Program :-

```
class final {
Public static void main (string [] args) {
    final int A = 10;
     System.out.println (A);
  }
}
```

Output
10

② Final method :
- whenever we declare a method as a final it can't be overriden to our extended class.

Syntax : final void m1 ()
{ }

Program :-

```
Class final{
void mNumber()
{
System.out.println("9988140799");
  }
final void atm PIN()
{
  System.out.println("894");
    }
}

Class Thief extends final{
@ Override
  void mNumber(){
  System.out.println("9988140799");}

@ Override
  void atm PIN(){
    System.Out.println("894");
      }
}
Class Test{
  Public static void main(String[]args){
  Thief t = new Thief();
  t.mNumber(); t.atm PIN();}
      }
```

Output

Overriden error method is final

3. Final class :

Whenever we declare a class as a final it can't be extended to sub class.

Syntax :- Final class A
{ }

Program

```
final class Test{
 void mNumber(){
  System.out.println("628472848");
    }
void atmPIN(){
   System.out.println ("695"); }
}
Class Thief extends Test{
 @ Override
   void mNumber(){
    System.out.println ("628472848"); }
  @ Override
    void atmPIN(){
```

```
System.out.println("695");
}
}
```

```
class final{
Public Static void main(String[] args){
    Thief t = new Thief();
    t.mNumber();
    t.atmPIN();
  }
}
```

Output

Can not inherit from final Test class Thief extends Test

→ Abstract class :-

- A class which contains the abstract Keyword in its declaration is called abstract class.

- We can't create object for abstract class.
- It may or may not contain abstract methods.
- It can have abstract & non abstract methods.
- To use an abstract class you have to inherit it from sub classes.

Syntax : abstract class A
{
}

Program

```
abstract class animal
{
}
class Demo
{
public static void main ( String [] args){
    animal a = new animal();
  }
}
```

Output
animal is abstract
We cannot initiated

→ Exception Handling

- An exception is unexpected / unwanted sitation that occured at runtime is called exception.

Ex - Powercut exception

- In exception handling, we should have an alternate source through which we handle the exception.

- The object oriented mechanism has provided the following techniques to work with exception :-

① try ② Catch ③ throw ④ throws ⑤ finally.

Program

```
Class Test
{
Public static void main(String [] args) {
System.out.println(" main method started ");
 int a = 10, b = 0, c;
   try {
    c = a/b;
    System.out.println(c);
    }
Catch (Exception e) {
System.out.println(" can't divide by zero");
  }
System.out.println(" main method ended ");
  }
}
```

1. try : A try block is used to wrap code that might cause an error.

Example : try{
int a = 10/0;
}

- If an exception happens here, the program does not stop.
- Java looks for a matching catch block (immediately)

2. Catch - handles the exception

- A catch block catches and handles the error throws in try.

Eg:
```
try {
    int a = 10/0;
    }
catch (Arithmetic exception e) {
  System.out.println("You cannot divide by zero");
  }
```

- Prevents program crash.
- Gives a meaningful message.
- Can have multiple catch blocks for different exception types.

3. finaly — Code that always executes
- The finally block runs.

① If exception occurs
② If exception does not occur.
③ Even if you use return inside try or catch.

- It is mainly used for clean up:
① Closing files.
② Closing database connections
③ Releasing resources

Eg:-
```
try {
    int a = 10/2;
    }
catch(Exception e){
  System.out.println("Error");
   }
finally {
 System.out.println("This will always run");
  }
```

4. Throw → Manually Throw an exception

- You use throw when you want to create and throw your own exception.
- Here the programmer decides when an error occur.

Eg:- throw new ArithmeticException("Manual Error").

5. Throws :- (Declares possible exceptions in a method)

- Used in method headers.
- It tells the calling code: "This method may throw an exception. Handle it!"

Eg: Void readData() throws IOException {
}

Caller must handle it:

```
try {
  readData();
  }
catch (IOException e) {
  System.out.println("File error");
    }
```