Unit-1

⇒ Java was initially developed in 1991 named as "oak" but was renamed "java" in 1995.

⇒ Java programming language was originally developed by Sun microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystem's java Platform.

⇒ It provides write once, Run anywhere (WORA), providing no-cost run-times on popular platforms.

* JDK:- (Java Development Kit).

→ JDK contains tools needed to develop the java programs, and JRE to run the programs.

→ The tools include compiler (javac.exe), java application launcher (java.exe), Applet viewer, etc...

⇒ Compiler converts java code into byte code.

⇒ java application launcher opens JRE, loads the class and invoke its main method.

⇒

JRE = JVM + Java package classes (like util, math, lang, awt, swing etc) + runtime libraries.

* JRE (Java Runtime Environment).

→ JRE contains JVM, class libraries and others supporting files.

→ ~~It does~~ It does not contain any development tools such as compiler, debugger, etc.

* JVM :- (Java Virtual Machine).

→ JVM called virtual because it provides a machine interface that doesn't depend on the operating system and machine hardware architecture.
→ When we compile a java file, output is not an .exe but it is a .class file.
→ .class file consists of java byte codes which are understanduble of JVM.
→ JVM interprets byte code into the machine code.
→ JVM is platform dependent.

* OOPs Concepts :-

1) class :-

→ "A class is a blueprint or prototype from which objects are created."
→ Simply class is collection of objects.
→ It defines attributes and methods.

2) Object :-

→ "An object is an instance of a class."
→ An object means anything from real life
→ Every object has at least one unique identity.
→ An object is a component of a program that knowns how to interact with other pieces of the program.
→ An object is the variable of the type class.

3) Abstraction:-

→ "Abstraction represent essential features without including the background details."
→ Implemented in class to provide data security.
→ We use abstract class and interface to achieve abstraction.
→ It provides only needed details, others details hide.

4) Encapsulation:-

→ "Wrapping up (Binding) of a data and functions into single unit is known as encapsulation."
→ The data is not accessible to the outside world, only those functions can access.
→ It is which is wrapped together within single unit.

5) Inheritance:-

→ "Inheritance is the process, by which class can acquire the properties and methods of another class.
→ The mechanism of deriving a new class from an old class is called inheritance."
→ The New class is called derived class and old class is called base class.
→ There are 5 types of inheritance.
1) Single Inheritence
2) multiple "
3) multilevel "
4) Hybaid "
5) Hierchical "

6) Polymorphism :-

→ "Polymorphism means the ability to take more than one form."
→ It allow a single name to be used for more than one related purpose.
→ We use method overloading and method overriding to achieve polymorphism.

7) Message passing :-

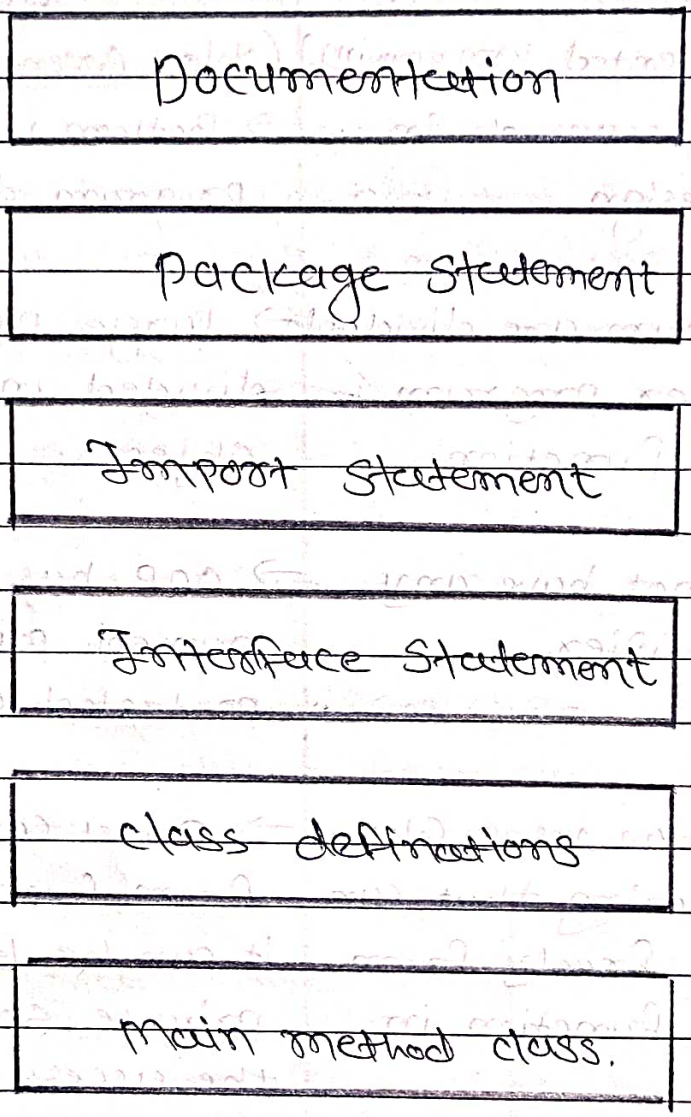
→ A program contain set of object that communicate with each other.
1) Creating classes that define objects and their behavi
2) Creating objects from class definition.
3) Establishing communication among object.

Static Binding	Dynamic Binding
→ Type of object is determined at compiled time known as static binding.	→ Type of object is determined at run time is known as dynamic binding.
→ Static binding uses Type information for binding	→ Dynamic binding uses Object to resolve binding
→ Static, private, final methods & variables are resolved using static binding.	→ Virtual methods are resolves during runtime based upon runtime object
→ Overloaded methods are resolve using static binding.	→ overridden methods are resolve using dynamic binding at runtime.

* Difference between OOP and POP.

POP (Procedure Oriented Programming)	OOP. (Object Oriented programming)
→ Top Down approach in program design	→ Bottom up approach in program design
→ Large program are divided into smaller program known as function.	→ large program are divided into classes and objects.
→ POP does not have any access specifiers.	→ OOP has access specifiers named public, private, protected etc.
→ Most function uses Global data of sharing that can be accessed freely from function to function in system.	→ Data can't move easily from function to function it can be kept public, or private, so we can control the access of data.
→ Adding of data & function is difficult.	→ Adding of data & function is easy.
→ OOPs concepts are missing	→ OOPs concepts are available and can be used easily.
→ We can't declare Namespace directly.	→ We can use name space directly. Ex- using namespace std;
→ Ex- C, Fortran, pascal etc...	Ex- C++, Java, C# etc...

* Structure of Java :-



1) Documentation Section :-

→ Documentation Section contain comments for the programmer. Ex- programmer Name, Date of program, ~~etc~~ author details etc.

- There are two comments.

1) Single comments : (//)

2) multiline comments :- (/* ---- */)

2) Package Statement:-

→ The 1st statement allowed in java file is a package statement.
→ Package is nothing but a group of classes & other functionality.
→ Ex- Package Student;

3) Import Statement:-

→ The next thing after a package statement may be a number of import statements. This is similar to the #include statement in C. There are two types of package users defined or inbuilt.
→ Ex- import java.io.*;

4) Interface Statement:-

→ An interface is like a class but includes a group of method declarations.

5) class definition:-

→ Java is object oriented so we need to declare class in each and every program.
→ classes are used to map the objects of real-world programs.

6) Main method:-

→ Every java program requires main method same as cond c++.

→ main method creates objects of various classes and establishes communication between them.

* Creating, Compiling and Executing a java program.

→ /* write a program to print "My First Program in java" */

⇒ import java.util.*; ← Import Package
class MyFirstProgram ← class name
{
main method → public static void main (String [] args)
{
System.out.println ("My First Program");
}
}
↑ To print

⇒ Save this program with class name. Class name and file name should be same., So save as MyFirstProgram.java.

* To run java program written in Notepad or Notepad++, open command prompt and compile program.

1) Javac Filename.java
→ 'javac' compiles source code to byte code and generate class file.

2) Java classname

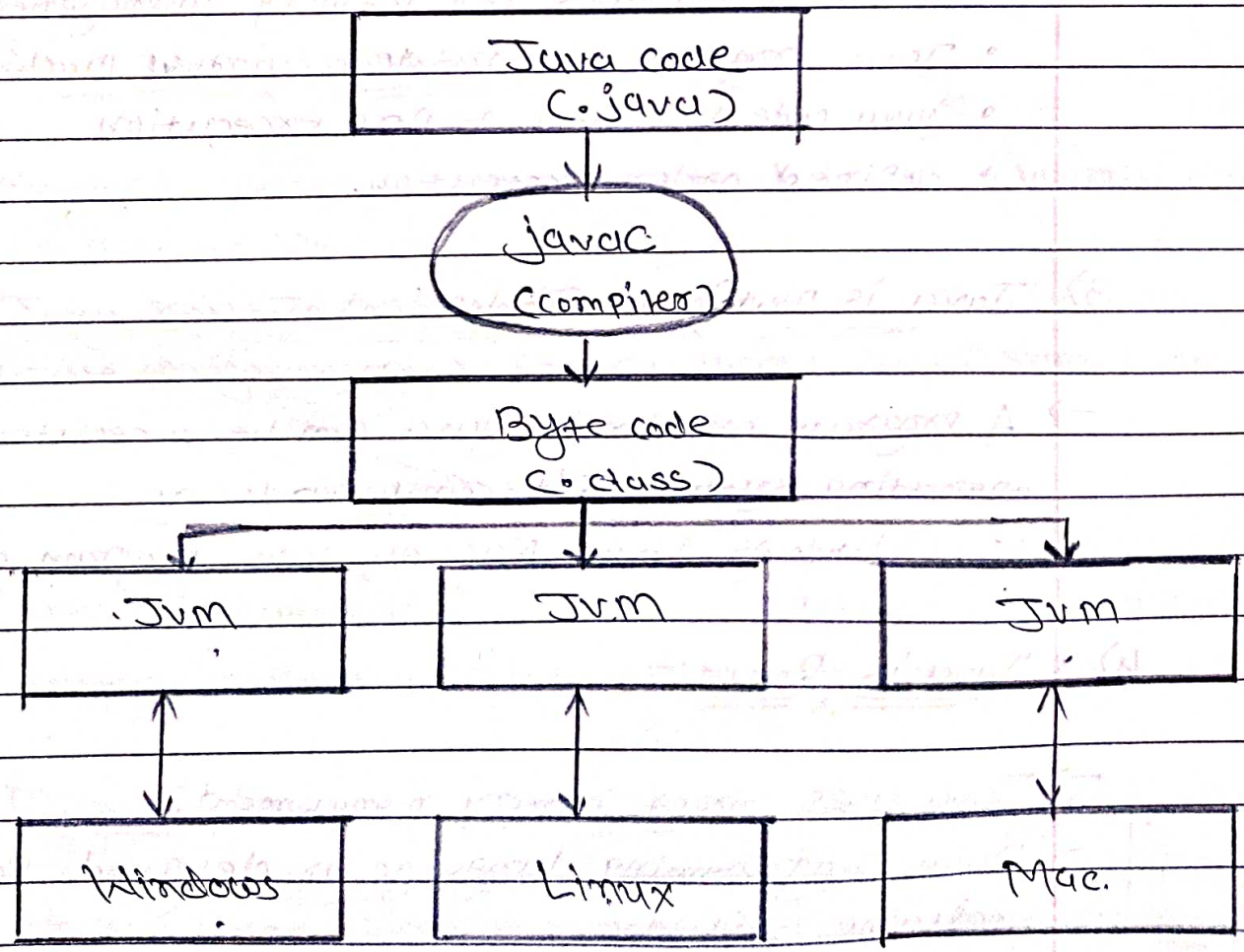
'java' Interprets the bytecode stored in class file and executs the program.

⇒ E:\DemoJava> javac MyFirstProgram.java
drive, Folder, compile, filename

E:\DemoJava> java MyFirstProgram
Interprete, class name

* Byte code :-

→ Byte code is nothing but intermediate representation of java source code produce by java compiler.



* Features of java :-

1) Simple :-

- It's simple because it contains many features of C and C++.
- Java removes complexities because it doesn't use pointers, storage classes and go to statement.
- Java is easy to learn and it's syntax is quite simple.

2) Java is secure :-

→ Java is secure because :
- No use of pointers for memory management.
- Java program run inside a virtual machine (JVM)
- java code verified before execution
- defined order execution.

3) Java is platform Independent :-

→ A program written in java can be executed on any operating system with any hardware.
Ex- Windows, Linux, Mac. etc java program execute.

4) Java is Robust :-

→ Java uses Strong memory management.
- Java programming language is designed for highly reliable software.

5) Java is OOP lang:-

→ Java purely follows the concepts of OOPs.
Ex- if we want to create Simple Hello world program in java, we need to create class.
- OOPs concepts supports code reusability and maintainability.

6) Java can be compiled and Interpreted.

→ Java uses both compile and Interpreter, firstly Compiler translates the java source program into byte code then java interpreter interprets this byte code to machine code.

7) Java support multi threading:-

→ ~~Modern~~ modern programs need to do several things at a same time.
- Java multithreading features make it possible to write program that can do many tasks simultaneously

8) High performance:-

→ Java program are faster than program or scripts written in purely interpreted languages.

9) Dynamic:-

→ At run time, java environment can extend itself by linking classes that may be located on remote server on a network.

* Variable:-

→ "A Variable is something that is used in a program to store data in memory." OR

"A Variable is a name used to refer to some location in memory – a location that hold a value."

→ A Variable may take different values at different time during execution of the program.

→ Data type gives meaning and capacity to variable

Ex- `Int a;`
`float b,c,d;`

Data Type ↑ ↑ Variable

* Rules for Variable Name:-

→ Every Variable name should start with alphabets (a to z & A to Z), digit (0 to 9), underscore (_) symbol.
→ The Variable name can't have space.
→ Variable names are case-sensitive
→ No keywords should access variable name
→ First character must be an alphabet or an underscore.

* Keywords:-

→ keywords are those words who reserved, already has been thier meaning has been fixed!!

- This keywords can't be used as variable name.

Category	keywords.
Access Modifiers	Private, protected, Public.
class, method, Variable modifiers	abstract, class, extends, final, new, Static, implements, ~~final~~, strictfp, Synchronized, Volatile, transient, native, interface, ~~instanceof~~
Flow control	break, case, continue, default, do, else, for, if, instanceof, return, Switch, while
Package control	import, package
Primitive types	boolean, byte, char, double, float, int, long, short.
Error handling	assert, catch, finally, throw, try, throws.
Enumeration	~~enum~~ enum
Others	Super, this, void
Unused	const, goto.

* Identifiers:-

→ They refer to the name of variables, functions, arrays, classes etc.
→ Created by the programmers.
→ Identifiers are user-defined names combination of letter and digits.

* Rules for identifiers :-

→ 1st character in an identifier must be an alphabet or underscore (_)
→ Must contain only letters, digit or underscore.
→ Uppercase & lowercase letters are distinct, identifiers are case sensitive.
→ Special character, blank space or keywords are not allowed within an identifiers.

* Constant :-

→ Variables, constants are data storage, but variables can vary, constants do not change.
→ You must initilize a constant when you create it, you can't assign new value later, after constant is initilized.

Ex- #define PI 3.14
    const int a=2;

(Figure Next Page.)

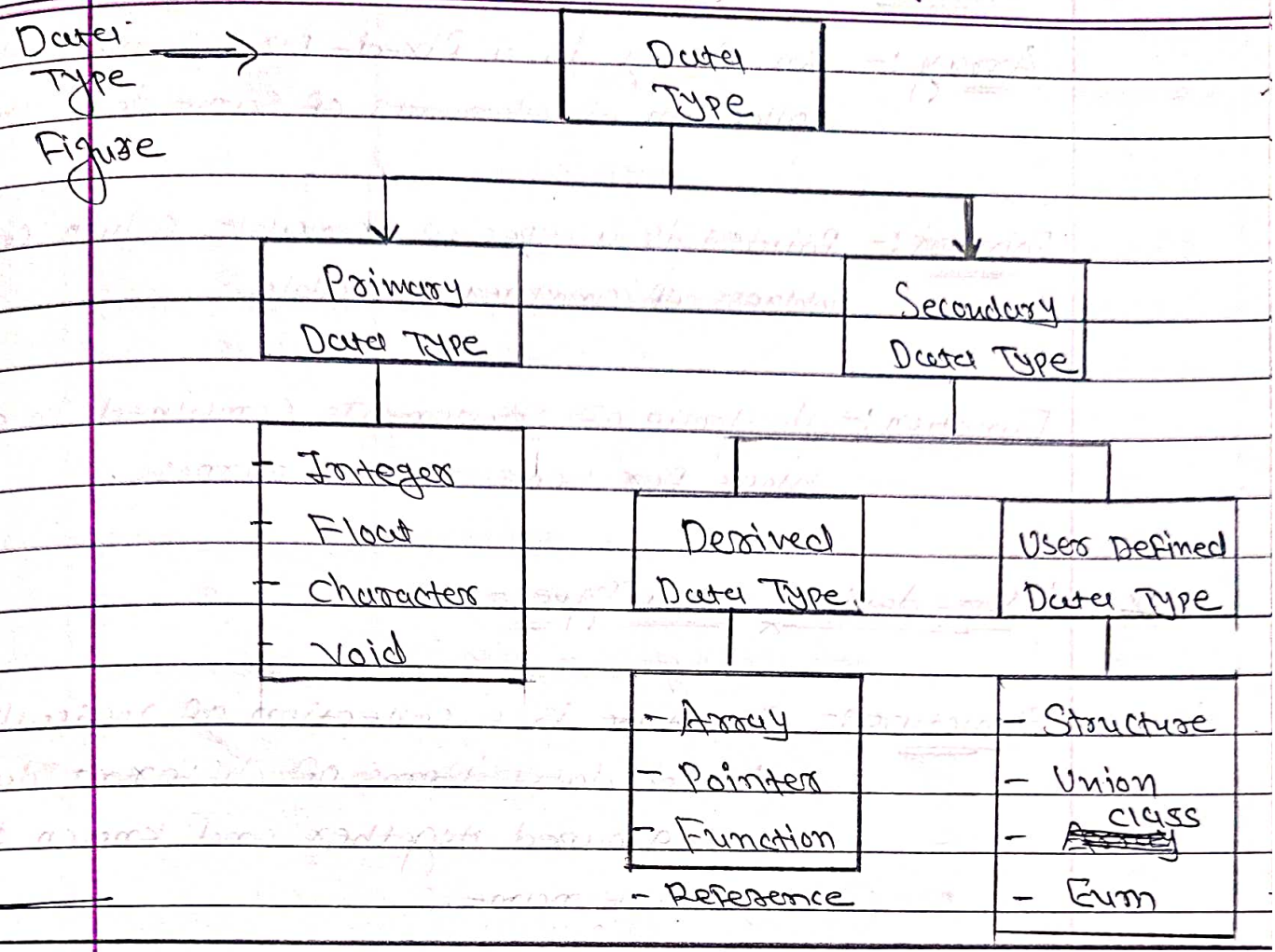
* Data type :-

→ "The data type of a programming element refers to what kind of data it can hold and how that data is stored."

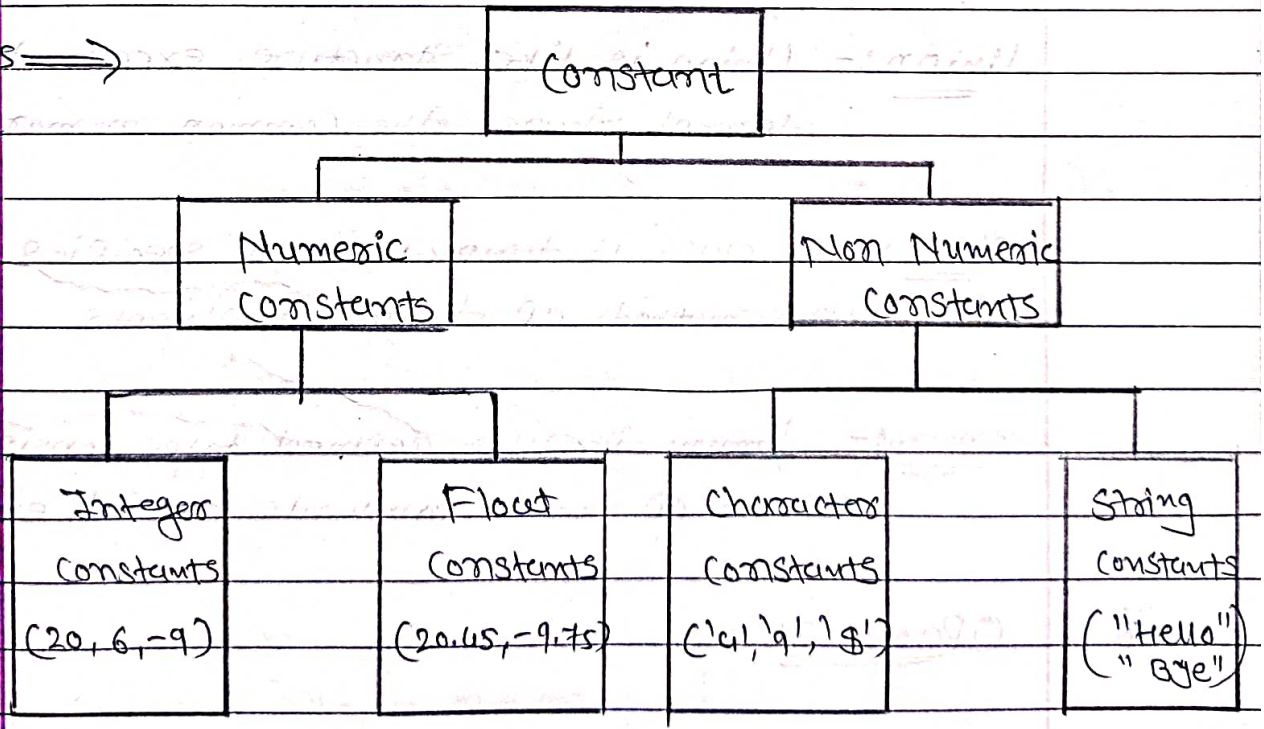
→ Data types are used to store various types of data that is processed by program.

→ We select appropriate data type based on our data and its operations.

Data Type Figure ⟶



Constants ⟶ Figure



* Derived Data type:-

Array:- An Array is a fixed-size sequenced collection of elements of same data type.

Pointer:- Pointer is a special variable which contains address of another variable.

Function:- A group of statements combined in one block for some special purpose.

* User-defined Data Type:-

Structure:- Structure is a collection of logically related data items of different data types grouped together and known by single name.

Union:- Union is like structure, except that each element shares the common memory.

Class:- A class is template that specifies the fields and methods of things or objects.

enum:- Enum is user-defined type consisting of a set of named constants called enumerator.

* Operators:-

1) Airthmetic operators:-

- Airthmetic operators are used in mathematical expression in the same way that are used in algebra.

Operator	Meaning	Example
+	Addition	5+2=7
−	Substraction	5-2=3
X	Multiplication	5*3=15
/	Division	20/2=10
%	Modulus (Reminder)	20%2=0
++	Increment operator increases integer value by one	a=5 a++
−−	Decrement operator decreases integer value by one	a=5 a−−

Ex-

```
class operator
{
  public static void main (String args[])
  {
    int a=22;
    int b=2;
    System.out.println("A="+a+"B="+b);
    System.out.println("sum="+(a+b));
    System.out.println("sub="+(a-b));
    System.out.println("mul="+(a*b));
    System.out.println("Div="+(a/b));
    a++;
    System.out.println("A="+a);
    b--;
    System.out.println("B="+b);
  }
}
```

O/p-

A=22 B=2

sum=24

Sub = 20
mul = 44
Div = 11
A = 23
B = 1

2) Assignment operater :-

→ Assignment operators are used to assign some value to the operand.

Operater	Meaning	Example
=	Assign from one value to other one	a=5
+=		a+=5
-=		a-=5
*=		a*=5
/=		a/=5
%=		a%=5

Ex- class operator2
{
public static void main (String args[])
{
int a=20;
System.out.println ("A=" +a);
a+=10;
System.out.println ("a=" +a);
a-=10;
System.out.println ("a=" +a);
a*=10;
System.out.println ("a=" +a);

```
a/=2;
System.out.println("a="+a);
a%=2;
System.out.println("a="+a);
}
}
```

O/P:-

```
A=20
a=30
a=20
a=200
a=100
a=0
```

3) Relational operators:-

→ All of the relational operators result in a boolean value.

Operator	Meaning	Example
==	Equality	if(a==b)
!=	Not equal	if(a!=b)
<	Less than	if(a<b)
>	Greater than	if(a>b)
<=	Less than equal to	if(a<=b)
>=	Greater than equal to	if(a>=b)

Ex- Class operators3

```
{
Public Static void main (String args[])
{
```

```
int a=20, b=5;
System.out.println("a==b" + (a==b));
if(a>b && a<100)
{
  System.out.println("A<100 and a>b");
}
  if(a!=10)
  {
    System.out.println("A is not equal to 10");
  }
}
```

O/P - a==b false
A<100 and a>b
A is not equal to 10.

4) Logical operators -

→ Logical operators are used to combine more than one condition.

Operator	Meaning	Example
&&	Logical and	(a>b && a>c)
||	Logical OR	(a>b || a>c)
!	Logical NOT	!(a>b)

Ex -
```
class operators4
{
  Public Static void main (String args[])
  {
```

```
int a=20, b=5, c=12;
if (a>b && a>c)
{
  System.out.println ("A is max");
}
  if(!(b>a))
  {
    System.out.println ("a is >b");
  }
    if (a>b || a>c)
    {
      System.out.println ("a>b || a>c");
    }
  }
}
```

O/P-

```
A is max
a is >b
a>b || a>c.
```

5) Bitwise operators:-

→ Bitwise operators can be applied to the integers type long, int, short, char and byte.

Operator	Meaning	Example
&	Bitwise AND	a&b
|	Bitwise OR	a|b
^	Bitwise exclusive OR	a^b
<<	left shift	a<<b
>>	Right shift	a>>b

Truth table for bitwise &, | and ^.

A	B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Ex-
```
class operators
{
    public static void main (String args[])
    {
        int a=2 // 0010 (byte code of 2)
        int b=4 // 0100 (byte code of 4)
        int c= a&b;
        System.out.println("a&b="+c);
                    // and condition = 0000
        c=a||b;
        System.out.println("a||b="+c);
                    // or condition = 0110
        c=a^b;
        System.out.println("a^b="+c);
                    // ^ (a^b)
        int no=8; // 0000 1000
        System.out.println("no="+no);
    // left shifting bytes with 1 position
        no=no<<1; // should be 16. i.e 0001 0000
    // equivalent of multiplication 2
        System.out.println("value of no after left
                    Shift:"+no);
        no=-8;
```

// right shifting bytes with sign | position
no = no >> 1; // should be 16 i.e 0001 0000

// equivalent of division 2
System.out.println("value of no after right
shift with sign:" + no);
}
}

o/p - a & b = 0
a | b = 6
a ^ b = 6
no = 8
value of no after left shift = 16
value of no after right shift with signed
sign: -4

6) Ternary | Conditional Operaters :-

→ The ternary operaters "? :" is operaters to take 3 operands.

Syntax :- Value = (expression ? value1 : value2);

Ex - class operaters 6
{
public static void main (String args[])
{
int a = 12;
int b = 3;
int ans = a > b ? a : b;

System.out.println (ans + "is max");
}
}

O/P- 12 is max.

* Types of Variable:-

1) Local Variable:-

→ Local Variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructors or block.

Ex- void printAll()
{
int x=5;
for (int i=1; i<=x; i++)
{
// Here i, x are local variable
}
}

2) Instance Variable:-

→ Instance Variable are created when an object is created with the new keyword and destroyed when the object is destroyed. Instance Variable access outside a method, constructor or blocks.

Ex- class a1
{
int rno;
String name; // here rno and name both are instance variable

```
A1 S1;
S1.rno = 5;
```

3) Class / Static Variable :-

→ A Variable that is declared as static is called static variable.
- Static variables are created when program starts and destroyed when program stops. It can be accessed by calling with class name.

Ex-
```
class A1
{
    static int cnt; // here cnt is static variable
}
A1.cnt = 5;
```

* Advantages of OOP :-

→ Troubleshooting is easier with OOP.
→ Code Reusability
→ productivity
→ Data Redundancy
→ Code Flexibility
→ Solving problems
→ Security.

Unit-2

* Control Statement:-

- We can control the order of execution of the program based on logic and values.
- Control Statement can be classified into selection statements, Iteration statements and branching statements.

1) Selection Statements:-

- Selection statement is also known as conditional or decision making statement.

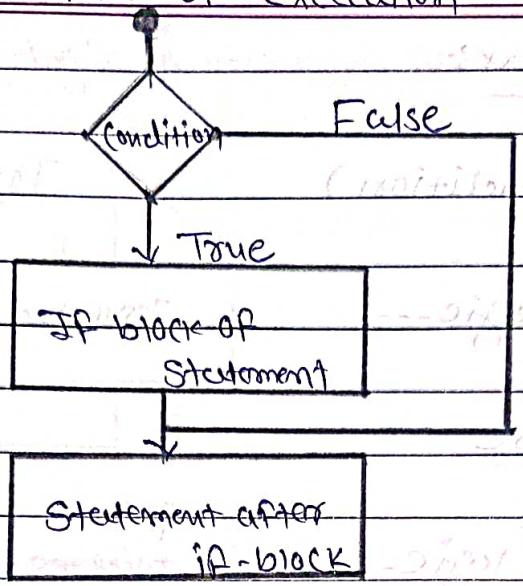
* IF Statement:-

→ There are cases when we would like to execute some logic when a condition is TRUE.
- condition usually results from a comparison of two values. If condition is TRUE then control goes to between IF.

Syntax:-

```
if(condition)
{
  Statement 1; // Executes when condition is true.
}
Statement after if-block
```

Flow of Execution



Ex-
```
class if1
{
    public static void main (String args[])
    {
        int a;
        int a=5;
        if (a==5)
        {
            System.out.println ("yes the value of A=" + a);
        }
    }
}
```

O/P -

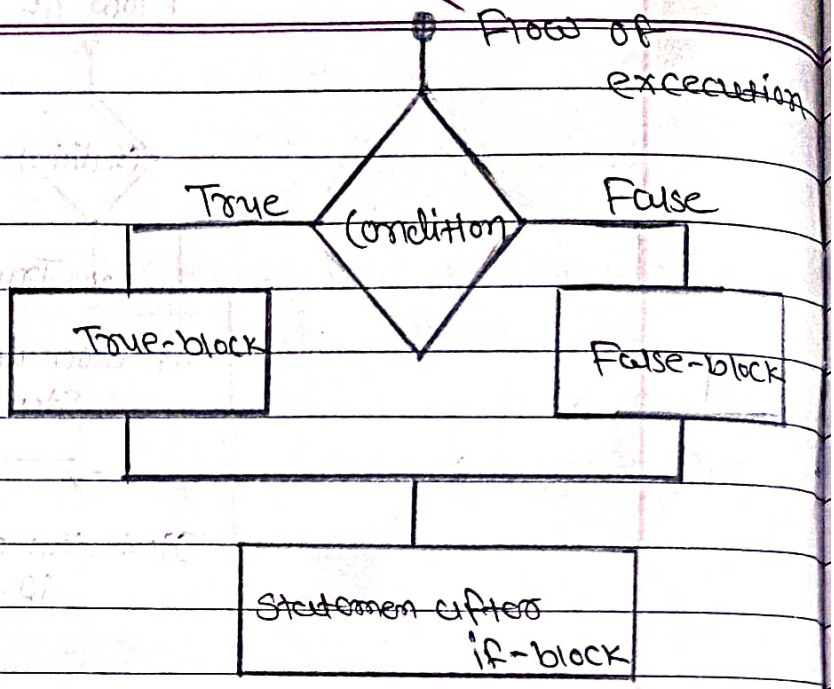
yes the value of A=5

* **If-else :-**

→ There are cases when we would like to execute some logic when condition is TRUE and some other logic when condition is False.

Syntax:-

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



Ex- class if2
{
public static void main (String args[])
{
int a=20;
if (a%2==0)
{
System.out.println ("No is even");
}
else
{
System.out.println ("No is odd");
}
}
}

O/p- No is even

* IF-- else if:-

→ There are cases when we would like to execute some more condition when a condition is TRUE, and some

Others condition logic when the condition is false.

Syntax:-

```
if (test expression 1)
  {
     Statement 1;
  }
else if (test expression 2)
   {
     Statement 2;
   }
else
    { Statement 3;
    }
```

Ex- class if3

```
{
   public static void main (String args[])
   {
     int time=22;
     if (time<10)
     {
      System.out.println("good morning");
     }
     else if (time<18)
      {
        System.out.println ("good day");
      }
     else
      {
        System.out.println ("good evening");
      }
   }
```

o/p- Good evening.

* Nested ... If :-

→ These are we writing if statement inside another if statement is called nested if statement.

Syntax:-

```
if (condition-1)
  {
    if(condition-2)
      {
        inner-if block of statements
      }
  }
```

Ex-

```
class if4
{
  public static void main(String args[])
  {
    int a=11,b=33, c=22;
    a=5;
    if(a>b)
    {
      if(a>c)
        System.out.println("A is max");
        else
        System.out.println("c is max");
    }
    else
    {
      if(c>a)
      System.out.println("B is max");
```

```
else
    System.out.println("c is max");
  }
 }
}
```

O/P- B is max.

* Switch-case:-

→ When you are comparing different values with some expression, we can use switch case statement.
- It executes one of several group of statements depending on the value of an expression.
- If any case statement is satisfied logic will run that only and then control will come out of switch block.

Ex-

```
class Switch1
{
  public static void main(String args[])
  {
    String color = "red";
    switch(color)
    {
      case "orange":
        System.out.println("wait");
      case "green":
        System.out.println("Drive the car");
      case "red":
        System.out.println("Stop the car");
```

Ex- class switch1
{
public static void main (String args[])
{
String color = "red";
switch (color)
{
case "orange":
System.out.println ("wait");
break;
case "green":
System.out.println ("Drive the car");
break;
case "red":
System.out.println ("Stop the car");
break;
default:
System.out.println ("wrong output");
}
}
}

o/p-
Stop the car.

2) Iteration / loop Statement:-

→ Iteration also known as looping Statements.
- looping structures are used when a group of Statements is to be executed repeatedly, until a condition is TRUE or until a condition is False.

* For loop:-

- For loop excecutes a block of statements a specified number of times.
- It's most common and popular loop used in programming language.

Syntax:- for (initilization; condition; step)
{
logic....
}

Ex-

```
Class loop1
{
Public static void main (String args[])
{
int i;
for (i=1; i<=10; i++)
{
System.out.println(i);
}
}
}
```

O/p- 1
2
3
4
5
6
7
8
9
10

* While loop :-

Syntax :-

```
while (up to condition);
{
  logic...;
}
```

→ While loop keeps executing until the condition against which it tests remain true.

- while statement always checks the condition before it begins the loop.

Ex- Class loop2

```
{
  public static void main (String args[])
  {
    int no=1;
    while (no<=5)
    {
      System.out.println(no+"=="+(no*no));
      no++;
    }
  }
}
```

O/p-

1==1
2==4
3==9
4==16
5==25

* Do while loop :-

→ Do while loop makes sure that the code block is always executed at least once.

- usually we used ~~switch~~ Switch case inside do while loop.

- Repeats a block of statements while a boolean condition is FALSE or until condition become TRUE.

Syntax:-
```
do
{
    logic...;
}
while (up to condition);
```

Ex:-
```
class loop3
{
  public static void main (String args[])
  {
    int no=1;
    do
    {
      System.out.println(no+"=="+(no*no));
      no++;
    }
      while (no<=5);
  }
}
```

O/P-
```
1==1
2==4
3==9
4==16
5==25
```

Array:-

→ Arrays in java are non-Primitive data types that store elements sequence of a similar data type.

→ There are two types of arrays ① single-dimensional arrays have only one dimensional, while multi-dimensional have 2D, 3D and nD dimensions.

- **Declare Array:-**

— here we can declare an array

```
dataType arrayName[];
```

- dataType - it can be primitive data type like int, char, double, byte etc.
- arrayName - it is an identifiers.

Ex- `double data[];`
dataType ↑ ArrayName

- **Allocate memory to array:-**

```
int Array[];          // declaring array
Array = new int[20];  // allocating memory to array
```

OR

```
int Array[] = new int[20];
                      // combing both statement in one
```

Initilize Array in java:-

⇒ In java array, each memory location is associated with a number. The number is known as an array index. We can also initilize array in java using index numbers.

Ex- int age = new int[5];
age[0] = 12;
age[1] = 4;
age[2] = 6;

age[0]	age[1]	age[2]	age[3]	age[4]
12	4	6	2	5

* One-dimensional array with example:-

→
```
class a1
{
  public static void main (String args[])
  {
    int a[] = {33,3,4,5};
    for (int i=0; i<a.length; i++)
      System.out.println(a[i]);
  }
}
```

O/P- 33
3
4
5

⇒ one-dimensional array example using Scanner

Ex-2

```
import java.util.Scanner;
Public class Array1
{
  Public static void main (String args[])
  {
    int n;
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter no of elements
                        you want Store:");
    n=sc.nextInt();
    int array[]=new int[10];
    System.out.println("Enter elements
                        of array:");
    for(int i=0; i<n; i++)
    {
      array[i]=sc.nextInt();
    }
    System.out.println("Array elements are:");
    for(int i=0; i<n; i++)
    {
      System.out.println(array[i]);
    }
  }
}
```

O/P- Enter no of elements you want store: 4
Enter elements of array:
12
14
18
9

Array elements are:
12
14
18
9

* Two-dimensional array with example:-

=> class a1
{
public static void main (String args[])
{
int array[] = {{1,2,3}, {2,4,5}, {4,7,8}};
for (int i=0; i<3; i++)
{
for (int j=0; j<3; j++)
{
System.out.println (array[i][j]+" ");
}
System.out.println ();
}
}
}

O/p- 1 2 3
2 4 5
4 7 8

=> Two-dimensional array with Scanner:-

```
import java.util.Scanner;
Public class c11
{
  Public Static void main (String args[])
{
   int m,n,i,j;
   Scanner sc= new Scanner (System.in);
   System.out.println("Enter no of rows:");
    m= sc.nextInt();
   System.out.println("Enter no of columns:");
    n= sc.nextInt();
    int array[][] =new int [m][n];
   System.out.println("Enter elements of arrays:");
    for (i=0; i<m; i++)
    for (j=0; j<n; j++)
     array[i][j] = sc.nextInt();
    System.out.println("Elements of array are:");
    for (i=0; i<m; i++)
    {
      for (j=0; j<n; j++)
      System.out.println (array[i][j]+" ");
      System.out.println();
    }
  }
}
```

O/P- Enter no of rows: 3
Enter no of columns: 3
Enter elements of arrays:
1
2
3
4
5

6
7
8
9

Elements of array are:

1 2 3
4 5 6
7 8 9

Fill() method:

→ This method assigns the specified data type value to each element of the specified range of array.

Ex-
```
import java.util.Arrays;
public class a1
{
  public static void main (String args[])
  {
    int array[] = {2,2,1,8,3,2,9};
    Arrays.fill (array, 10);
    System.out.println ("Array completely filled" +
        " with 10\n" + Arrays.toString (array));
  }
}
```

O/p- Array completely filled with 10

[10, 10, 10, 10, 10, 10, 10]

→ If we want, assign value in some part then we can also assign with using Index.

```
import java.util.Arrays;
Public class q2
{
  Public static void main (String args[])
  {
    int array[] = {2,2,1,8,3,2,9};
    Arrays.fill (array,1,5,10);
    System.out.println (Arrays.toString(array));
  }
}
```

O/P- [2,10,10,10,10,2,9]

⇒ 2D Array:-

```
import java.util.Arrays;
Public class q3
{
  Public static void main (String args[])
  {
    int array[][] = new int[3][4];
    for (int row[]: array)
      Arrays.fill (row,10);
      System.out.println (Arrays.deepToString(array));
  }
}
```

O/P- [[10,10,10,10],[10,10,10,10],[10,10,10,10]]

Arrays.sort() Method :-

⇒ This method are used with arrays in order to search, sort, compare, insert elements or return a string representation of an array.

Ex- import java.util.Arrays; (Ascending order)

```
public class a4
{
  public static void main(String args[])
  {
    int array[] = {13,7,6,45,21,9,101,102};
    Arrays.sort(array);
    System.out.println("Modified array[]:"
                       + Arrays.toString(array));
  }
}
```

O/P- Modified array[]: [6, 7, 9, 13, 21, 45, 101, 102]

⇒ If you want to change in some part you also using index number.

Ex- import java.util.Arrays;

```
public class a5
{
  public static void main(String args[])
  {
    int array[] = {13,7,6,45,21,9,2,100};
    Arrays.sort(array,1,5);
    System.out.println("Modified array[]:"
                       + Arrays.toString(array));
  }
}
```

O/p- Modified array[]: [13,6,7,21,45,9,2,100]

(*) (Descending order):-

```
import java.util.Arrays;
import java.util.Collections;
public class q6
{
  Public static void main (String args[])
  {
    Integer array[] = {13,7,6,45,21,9,2,100};
    Arrays.sort (array, Collections.reverseOrder());
    System.out.println ("Modified array[]:"
        + Arrays.toString (array));
  }
}
```

O/p- Modified array[]: [100,45,21,13,9,7,6,2]

Arrays.equals ():-

If you want to check whether two arrays are equal or not. Two arrays are considered equal if both arrays contain same no of elements with index.

```
import java.util.Arrays;
Public class q7
{
  Public static void main (String args[])
  {
```

```
int arr1[] = new int [] {1,2,3,4};
int arr2[] = new int [] {1,2,3,4};
int arr3[] = new int[] {1,2,4,3};

System.out.println("is arr1 equals to arr2:"+
                    Arrays.equals(arr1, arr2));
System.out.println("is arr1 equals to arr3:"+
                    Arrays.equals(arr1, arr2));
}
}
```

O/P- is arr1 equals to arr2: true
is arr1 equals to arr3: False.

Arrays.binarySearch():-

⇒ Arrays.binarySearch() method searches the specified array of the given data type for the specified value using binary search algorithm.

⇒
```
import java.util.Arrays;
public class as
{
public static void main (String args[])
{
int intArray[] = {10,20,15,22,35};
char charArray[] = {'g','p','q','c','i'};
Arrays.sort(intArray);
Arrays.sort(charArray);
```

```
int intKey = 22;
char charKey = 'g';

System.out.println(intKey + " found at index= "
    + Arrays.binarySearch(intArray, intKey));
System.out.println(charKey + " found at index = "
    + Arrays.binarySearch(charArray, charKey));
```

O/P-

22 found at index = 3
g found at index = 1.

String:-

⇒ String is immutable type. "String is basically an object that represent sequence of char values"

→ Java String class provides a lot of methods to perform operations on Strings such as compare(), concat(), equals(), copy(), length(), Split(), Substring(), replace(), toUpperCase(), toLowerCase(), trim() etc.

→ Java String is immutable which means it can't be changed. Whenever we change any String, a new instance is created. You also used StringBuffer and StringBuilder for char sequence implement.

⇒ There are two ways to create String object

1) By String literal.
2) By new keyword.

Ex- Public class q9
=
{
Public static void main (String args[])

String s1 = "java"; // Creating by Java String literal

Char ch[] = {'s','t','r','i','n','g','s'}; // Converting char array to String

String s2 = new String(ch);

String s3 = new String("example"); // Creating java String by new keyword.

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}
}

O/p- java
Strings
example.

1) Get length of String:-

class q10
{
Public Static void main (String args[])
{
String s1 = "Hello World!";
System.out.println("String:" + s1);
int length = s1.length();
System.out.println("Length:" + length);

```
    }
}
```

o/p - String : Hello World!
Length : 12

2) Join two Java Strings :-

→
```
class a1
{
  Public static void main (String args[])
  {
    String s1 = " Java ";
    System.out.println (" First String :" + s1);

    String s2 = " programming ";
    System.out.println (" Second String :" + s2);

    String s3 = s1.concat (s2);
    System.out.println (" Joined String :" + s3);
  }
}
```

o/p - First String : Java
Second String : Programming
Joined String : Java Programming.

3) Compare two String :-

→
```
class a2
{
  Public static void main (String args[])
```

```
String s1 = "java programming";
String s2 = "java programming";
String s3 = "python programming";

boolean result1 = s1.equals(s2);
System.out.println("String first and second are
                    equal:" + result1);
boolean result2 = s1.equals(s3);
System.out.println("Strings first and third are
                    equal:" + result2);
```

O/p- Strings first and second are equal: true
Strings first and third are equal: false.

4) Reverse String:-

→
```
class c1
{
  public static void main(String args[])
  {
    String s1: "Hello";
    String s2: " ";
    System.out.println("original String:" + s1);
    for(int i=0; i<s1.length(); i++)
    {
      s2 = s1.charAt(i) + s1;
    }
    System.out.println("Reversed
                        String:" + s2);
  }
}
```

O/P- original String : Hello
Reversed String : olleH

5) Uppercase, lowercase:-

```
→ Class a2
{
  Public static void main (String args[])
  {
    String txt = "Hello World";
    System.out.println (txt.toUpperCase());
    System.out.println (txt.toLowerCase());
  }
}
```

O/P- HELLO WORLD
hello world.

6) indexOf() :-

```
→ class a3
{
  Public static void main (String args[])
  {
    String S1 = "Please locate where locate occurs.";
    System.out.println (S1.indexOf ("locate"));
  }
}
```

O/P- 7

7) Copy String:-

⇒
```
import java.util.Scanner;
public class a1
{
  Public static void main (String args[])
  {
    String s1, s2;
    Scanner scan = new Scanner (System.in);

    System.out.println ("Enter the string:");
    s1 = scan.nextLine();
    s2 = s1;

    System.out.println ("\n s1 = " + s1);
    System.out.println ("\n s2 = " + s2);
  }
}
```

O/P- Enter the string: Ankit
s1: Ankit
s2: Ankit.

8) Split String:-

→
```
class a1
{
  public static void main (String args[])
  {
    String s1 = "Java is a good programming language";

    String result[] = s1.split (" ");
    System.out.println ("result = ");
    for (String str : result)
```

```
{
    System.out.println (str + ", " );
  }
 }
}
```

O/P- result = java, is, a, good, programming, language,

9) Replace String:-

```
→ class a1
{
  public static void main (String args[])
  {
    String s1 = "bat ball";

    System.out.println (s1.replace ('b', 'c'));
  }
}
```

O/P- cat call.

10) String trim():-

```
→ class a1
  {
    public static void main (String args[])
    {
      String s1 = "Learn java programming";
      System.out.println (s1.trim());
    }
  }
```

O/p- Learn java programming.

=> (Remove whitespace in start or end of string).

Unit-4

(Exception Handling)

* Exception Handling:-

→ "An exception is a problem that arises during execution of a program."

→ When exception occurs program is disrupted and program terminates abnormally.

→ "So when runtime unexpected / unwanted / abnormal situation that occur at runtime called exception and the process of handling exception is called Exception handling."

⇒ Java there are 3 categories of Exceptions:

1) Checked Exceptions:- A checked exception is an exception that occurs at the compile time. Ex- IOException, SQLException etc.
- Also called compile time Exception.

2) Unchecked Exception:- An unchecked exception is an exception that occur during the execution. Ex- logic errors or programming bugs
- Also called Runtime Exception.

3) Errors:- These are not exceptions at all, but problems that arise beyond the control of user or programmer.
Ex- Out of Memory Error,
Virtual Machine Error Exception.
JVM not installed.

* Exception Handling Mechanism :-

1) try
2) catch
3) finally
4) throw
5) throws.

* Using try and catch :-

→ 1) try :-

→ "Java try block is used to enclose code that might throw an exception or risky code."
→ It must be used within a method.
→ Java try block must be followed by either catch or finally block.

2) catch :-

→ "Java catch block is used to handle the Exception."
→ It must be used after try block only.
→ You can use multiple catch block with single try.

Ex:-
```
Class A
{
    Public Static void main (String args[])
    {
        System.out.println ("main method started");
        int a=10, b=0, c;
        c=a/b;
        System.out.println ();
```

```
        System.out.println("main method ended");
    }
}
}
```

O/p:- main method Started
Exception in thread "A" java.lang
Airthmetic Exception / can't divide by zero.

⇒ In this example, first statement execute, but in Second statement 10 can't divide by zero so display Airthmetic Exception., Exception occur and that why program disrupted and statement 3 not display. We use try and catch method here.

Using try catch method:-

```
class A
{
  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(e);
    }
```

```
        System.out.println ("main method ended");
    }
}
```

O/P- main method started
Aiathmetic Exception.
main method ended.

→ In this case exception occur but statement 3 also execute.

3) Finally:-

→ A Finally keyword is used to create a block of code that follows a try or catch block.
→ A Finally block of code always executes whether or not exception has occurred
→ A finally block appear at end of catch block.

Ex:- (No error in try block) catch block skip

```
class A1
{
    public static void main (String args[])
    {
        try
        {
            System.out.println ("method started");
            int a=20, b=2, c;
            c=a/b;
            System.out.println (c);
            System.out.println ("java");
```

```
}
catch (Airthmetic Exception e)
{
  System.out.println ("can't divide by zero");
}
finally
  {
    System.out.println ("finally block execute");
  }
    System.out.println ("program ended");
}
```

O/P- method started
10
java
finally block executed
Program ended.

Ex:- (Error in try block) - Program stop when exception and redirect to catch block.

```
class a1
{
  Public static void main (String args[])
{
  try
  {
    System.out.println ("method started");
    int a=20, b=0, c;
    c= a/b;
    System.out.println (c);
    System.out.println ("java");
  }
```

```
catch (ArithmeticException e)
{
    System.out.println("can't divide by zero");
}
finally
{
    System.out.println("finally block execute");
}
System.out.println("program ended");
}
```

O/P- method started
ArithmeticException.
Can't divide by zero
finally block execute
program ended

→ (In this example, Exception occur and redirect to catch block and catch block execute. both condition Exception occur or not finally block always execute.)

4) Throw :-

→ Throw keyword is used to explicitly throw an exception.
→ We can throw either checked or unchecked exception using throw keyword.
→ Throw keyword we can write Exception manually by user.

Ex:-
```
class a1
{
  public static void main (String args[])
  {
    int a=10, b=0; c;
    c= a/b;
    System.out.println(c);
  }
}
```

O/p- Airthmetic Exception.

(In this case Jvm decide and show error of Exception but if user use throw keyword and display exception manually)

Ex:-
```
class a1
{
  public static void main (String args[])
  {
    throw new
    ArithmeticException();
  }
}
```

O/p- Airthmetic Exception.

5) Throws :-

→ Throws keyword is used to declare an exception.
→ If method doesn't handle exception, method must declare it using throws keyword.

Ex- class a1
{
~~void div(int~~
int a, b, c;
void div(int a, int b) throws ArithmeticException
{
if (b==0)
{
throw new ArithmeticException();
}
else
{
c=a/b;
System.out.println(c);
}
}
public static void main(String args[])
{
a1 a=new a1();
a.div(20,0);
}
}

O/p- ArithmeticException

Users-Defined Exception:-

→ We can create own exception that is known as custom exception or users-defined exception.
- we can have our own exception and message.

Ex:- class MyException extends Exception

```
{
    MyException (String s)
    {
        super(s);
    }
}

class a1
{
    public static void main (String args[])
    {
        try
        {
            int i=3;
            int j=5;
            if (i<j)
            {
                throw new MyException ("i less than j");
            }
            else
            {
                System.out.println (i-j);
            }
        }
        catch(MyException e)
        {
            System.out.println (e);
        }
    }
}
```

o/p - i less than j