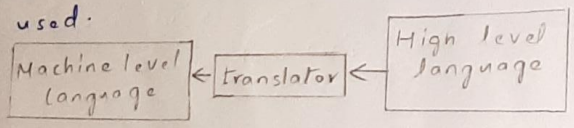
1 Introduction to Programming In JAVA

What is programming

The process of giving set of instructions for a computer to perform any specific task known as programming.

* A data structure is a named location that can be used to store and organize data. And, an algorithm is a collection of steps to solve a particular problem.
* Data structures & Algorithms are the Foundational blocks of Programming.
* Computer understands only Binary language (0 or 1) or Machine level language or low level language.
* (5+2) → to convert this to binary is very difficult so higher level languages are made. like java, c++ etc.
* We code in java, c++ etc but computer understands only Machine level language so to solve this problem a translator is used.



* There are 2 types of translator -

compiler -
A compiler scans the program all at once and compile

Interpreter -
A Interpreter scans the program line by line

* Syntax check is also done with translation if we deny the syntax the compiler give us error.

Programming Paradigms

- Procedural - code is return in set of procedures. We also call it as functions / Subroutine eg- c, basic, pascal. (Input → calculation → print) procedure
- Functional
- object oriented

→ Functional programming paradigm - code is return in pure functions
eg- Javascript & python.

function - A block of code that performs a specific task and when we call that block of code it runs

pure functions - they follows the above 2 properties
(i) They follow strict control flow (for same i/p there will be same o/p)
(ii) They do not change any external factor if there is a variable outside the function that changes the value inside the function doesn't call as pure function

→ object oriented programming paradigm -
In oop data is organised in the form of classes and objects rather than function & procedure eg- Java, c++

class - user defined blueprint (template)

object - real world entity (Instance of a class)

eg- class → person — Methods - A object can do (talk, walk)
object → Me — Attributes - features an object has (2 eyes, 2 legs)

class - car
object - BMW, Audi
Attributes - Yr of Manufacture, color, Model No.
Method - Accelerate, brake

* Any language can follow any of the paradigm. eg- java can follow procedural, functional or object oriented.

* JAVA is a class based object oriented programming language. which makes it very easy to write, compile & debug code.

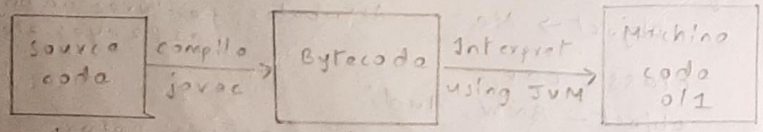
* Java was developed by James Gosling in the year 1995 and it was later acquired by oracle corporations

* Java is used in Multiple fields
  → Developing Mobile Applications, Web Applications
  → also used in Big data processing & Embedded systems.

Important Features of Java language

* Java is an object oriented programming language (application revolve around the concept of classes & objects)
* Java is platform Independent
* Java is fairly simple & secure
  (Java has automatic garbage collection so we dont require to deallocate memory explicitely. and Java doesnt have pointers & Multiple Inheritance)
* Java has a large standard library

Java Architecture

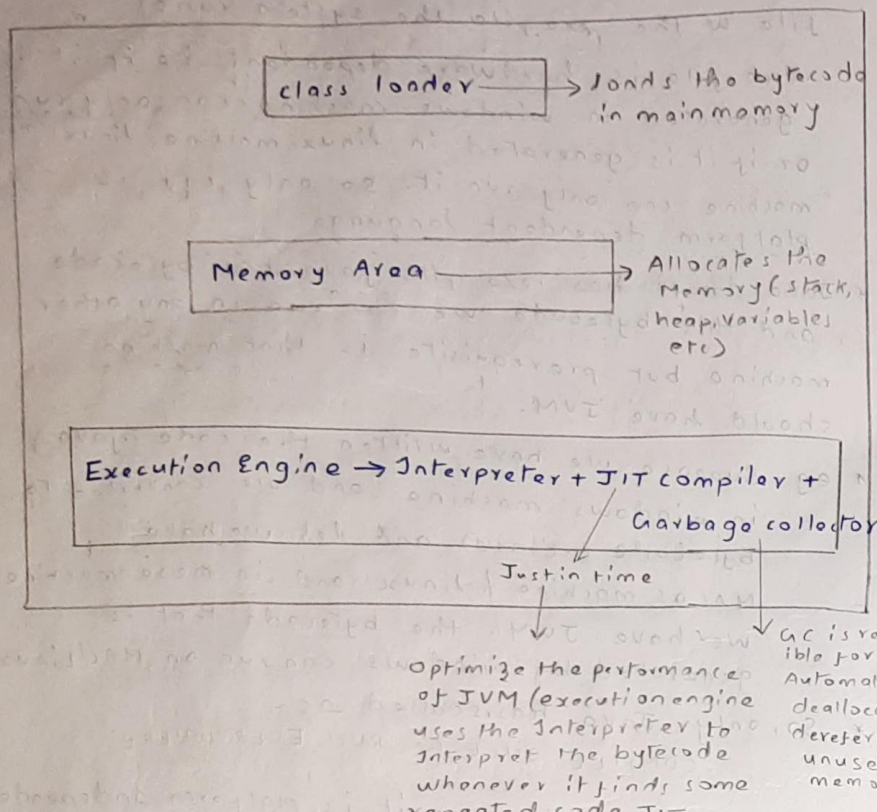


* Java is platform Independent (we can compile the code anywhere and the code can run in other systems eg: windows, linux, mac etc)
* unlike in c++ (.cpp file is converted to .exe file & the .exe file the system runs)
  .exe file is hardware dependent i.e if it is generated in windows machine it can only run in windows machine or if it is generated in linux machine, linux machine can only run it. so only c++ is platform dependent language
* Java doesnt has .exe file java has bytecode and this bytecode we can run in any other machine but prerequisite is that machine should have JVM.
* eg - Suppose we have written the code (.java) in windows machine and we converted to bytecode (.class) and let we have MAC os machine / linux and in these machines we have JVM. the bytecode that is generated by windows can run on MAC/Linux.
* So, only Java has is called as -
  COMPILE ONCE & RUN EVERYWHERE
* It is portable because it is platform Independent
* JVM is Machine dependent.

Java architecture components
* JVM - Java Virtual Machine
* JRE - Java Runtime environment
* JDK - Java Development kit

JVM (Java Virtual Machine)

* JVM provides an runtime environment to execute the bytecode in to Machine code



JRE (Java Runtime Environment)

JRE = JVM + Runtime libraries

JDK (Java Development Kit)

JDK = JRE + Development tools

↓

Javac, debugging tools, monitoring tools, Javadoc

• learn pdf Notes

Number System

Decimal	Binary Integer/2 ↑ fraction*2 ↓	octal Integer/8 ↑ fraction*8 ↓	Hexadecimal Integer/16 ↑ fraction*16 ↓
Binary	Decimal $\ldots 2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}$ Multiply	octal 3 bits	Hexadecimal 4 bits
octal	Decimal $\ldots 8^3\ 8^2\ 8^1\ 8^0\ 8^{-1}$ Multiply	Binary 3 bits	Hexadecimal step1 - 0 → B step2 - B → H
Hexadecimal	Decimal $\ldots 16^3\ 16^2\ 16^1\ 16^0\ldots$ Multiply	Binary 4 bits	octal step1 - H → B step2 - B → O

* Decimal Number System
  - → Numbers from 0 to 9 are used eg - 1923
  - → Base 10
  - → eg - $(3451)_{10}$

$$3\times10^3 + 4\times10^2 + 5\times10^1 + 1\times10^0$$
$$= 3451$$

* Binary Number System
  - → Numbers (digits) are 0 and 1 eg - 1001
  - → Base 2
  - → eg - $(1001)_2$

$$1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$$
$$8 + 0 + 0 + 1 = 9 \quad (1001)$$
$$8\ 4\ 2\ 1$$

* Conversion of Binary to Decimal

1. $(1\ 0\ 0\ 1)_2$
   3 2 1 0

$(1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= 9$

2 $(1\ 1\ 0\ 1)_2$
3 2 1 0

$(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

unit digit of = 13 | unit digit of 11 | unit digit of 110 | unit digit of 1101

Algorithm -

1. Find the unit digit of the given number (using modulo % operator we get the last digit of a Number)
2. Multiply that unit digit Number by power of 0 to Number 2 (for binary) 8 (octal) 16 (Hexadecimal) i.e power = $2^0 = 1$ & add and store in variable sum = 0.
3. Increment power by power * 2 i.e (1 2 4 8 . .)
4. Divide the Number by 10 to remove the last digit of the Number.

* Conversion of Decimal to Binary

1 $(9)_{10}$

2 | 9		
2 | 4	1	
2 | 2	0	
1	0	

1 0 0 1

2 $(13)_{10}$

2 | 13		
2 | 6	1	×1
2 | 3	0	×10
1	1	×100

×1000

⇒ 1 1 0 1

Algorithm -

1. Find the parity of the given number (using modulo % operator we get the remainder) i.e num % 2 = ~ (parity)
   2 because we have to convert Decimal to binary or 8 for octal and so on.
2. Multiply the parity by power=1 and Increment power by power * 10 and add & store in sum = 0.
3. Divide the Number by 2 until the number is greater than 0.

- 2 codes (VS code)

4

Variables & Datatypes

* Java output

→ System.out.println(); //prints on next line
→ System.out.print(); // prints on same line

* Hello world program

```
class HelloWorldJava {
    public static void main(String[]args) {
        System.out.println("Hello world program
                                    In Java");
    }
}
```

* How to run the code in IntelliJ

1. cd /src/
2. javac Main.java // compile (className)
3. It generates an Bytecode : Main.class
4. java Main → class Name
5. output Appears

- The above 1,2,4 points mean that the source code is compiled to Bytecode and it is interpreted by JVM to convert in to Machinecode.

* everything in Java is written in classes

* class - It is a user defined blueprint consisting of Attributes & methods

eg- class - car

Attributes - Company Name, color, price

Methods - run(), openTrunk(), openwindow()

objects - BMW, Audi etc.

↓ Instance of a class

→ So we have to find the color of BMW the syntax is BMW.color / Audi.color / BMW.openwindow().

Objectname.Attribute/Method.

* public static void main (String [] args)

- Access Modifier
- using static keyword
- return type
- function name
- command line Arguments

public - everyone can Access

we can call Method/Attribute without creating an object of the class

main is the entry point of the Java programme

Command line Arguments eg -

* In IntelliJ eg we type java Hellowworld one two

from uth line of how to run code

className → args[0], args[1]

Sout[args[0]]; o/p - one

* System.out.println();

- System - class by default. It provides standard i/p, o/p & error o/p streams
- out - It is an object of printStream class. It is standard output stream
- println() - function/Method in printstream class

* Java Variables - titles of reserved Memory location (or) Name of a Memory location.

⇒ Syntax of Declaring a Java Variable

```
DataType Variable_name = value ;
```

eg - int money = 2000000;

String name = "Suraj";

code -

```
class Main {
    psvm {
        int money = 1000;
        Sout(money);
        money = 5000;
        Sout(Money);
    }
}
```

o/p - 1000
5000

* Java Naming Convention

* lowercase - eg - suraj, vaishnavi (packages are written in lowercase)
* uppercase - eg - SURAJ, VAISHNAVI (constants are written)
* camelcase - functions / variables are written in camelcase eg - suraj Vaishnavi, physics Wallah
  first letter small & Immediate next word first letter capital
* Pascal case - class is written in pascal case
  eg - Both starting & Immediate words are capital
  Suraj Vaishnavi

* snake case/- words are separated by underscore.
eg- college_wallah

→ constants are written in upper snakecase
eg- MIN_CAPACITY

→ upper snake case are also called as screaming snakecase

* Rules for Naming Variables in Java

rule 1 - Variable name should not begin with a number
eg - 1stnum X

rule 2 - whitespace are not permitted
eg - money borrowed

rule 3 - A java keyword (reserve word) cannot be used as a variable name
eg - int float ;

rule 4 - It is prefered to use camelcase while naming variables.
(variable names with more than one word with all lowercase letters for the first word w capitalization of the first letter of each subsequent word)

rule 5 - when creating variables, it is prefered to give them meaningful names
eg- age ✓ a X
money ✓ m X

rule 6 - all lowercase letters should be used when creating one-word variable names.

* Java Identifiers -
Name of any component in Java called as Identifier eg- name given to any class, method, package, function, variable.

* Rules -

1. All Identifiers should begin with a letter (A to Z | a to z), currency character ($) or an underscore(_).
2. After the first character, Identifiers can have any combination of characters
3. A keyword cannot be used an an Identifier
4. Identifiers are case sensitive
5. whitespace is not permitted.

legal	Illegal
openTrunk(); ✓	open_Trunk(); X
money@ ✓	@money X

* Java Datatypes

ASCII values
48-57 - 0-9
65-90 - A-Z
97-122 - a-z

primitive
* built in
* hold single values
* cannot be divided into simpler DT
  - boolean
  - char
  - byte
  - short
  - int
  - long
  - float
  - double

Non-primitive
* user defined
* Memory address of a variable - it holds
* Complex - they can be divided. they can have multiple values
  - string
  - Arrays
  - classes
  - Interfaces

* Primitive Data Types

1 boolean -
boolean flag = true/false ;

2 Byte -
byte b = 60;
size: 1 byte / 8 bits
Range: -128 to 127

3 short -
short s = 1000;
size: 2 bytes / 16 bits
Range: -32,768 to 32,767

4 int - int a = 10000;
size = 4 bytes / 32 bits
Range: -2^{31} to $2^{31}-1$

5 long -
long l = 12345678;
size - 8 bytes / 64 bits
Range - -2^{63} to $2^{63}-1$

6 Double -
double d = 12.4567893 ;
size - 8 bytes / 64 bits

7 Float -
Float f = 123.456f;
size - 4 bytes / 32 bits

8 char -
char ch = 's';
size - 2 bytes / 16 bits

- pdf Notes
- Assignment Questions

5
Taking Input

* Java Scanner class

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

- import - it is used to import package
- java.util - library package → collection of classes, interfaces, subpackages
- Scanner - class

At the begining of our program we have to write this for taking user input

→ Scanner class is used for taking user input
In the package name util we have Scanner class.

* object of Scanner class - syntax

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

- Scanner - class Name
- sc - object Name
- new - Keyword
- Scanner - class Name
- System.in - standard i/p we have to take input from keyboard

* codes in vs code

* program to read characters using Scanner
class - In java we do not have nextChar() method so we have to take next() method (string method) to read characters.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter name: ")
        char ch = sc.next().charAt(2);
        System.out.println(ch);
    }
}
```

i/p - Suraj
o/p - r

* Java Scanner Methods to Take Input

1. nextInt()
2. nextFloat()
3. nextBoolean()
4. nextLine() // prints after space.
5. next() // Doesn't print after space
6. nextByte()
7. nextDouble()
8. nextShort()
9. nextLong()

- pdf Notes
- Assignment Questions

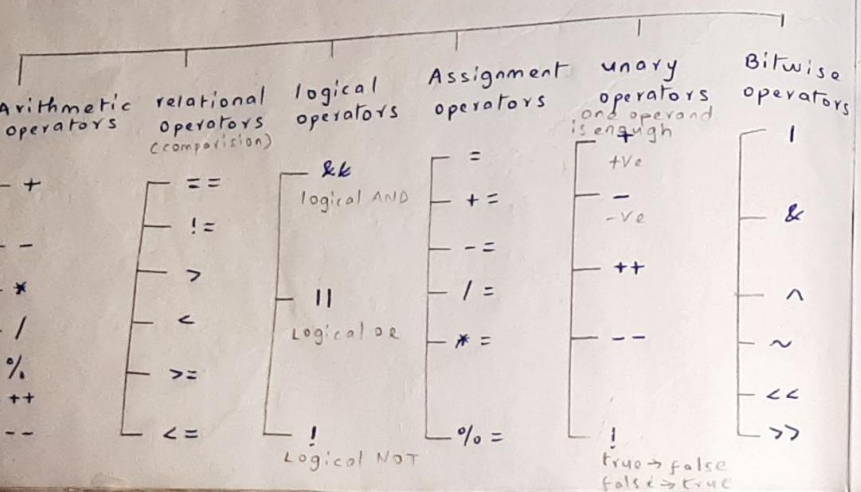
6

Operators

* Java Operators

eg - 3 + 7 = 10

3 → operand1, + → operator, 7 → operand2, 10 → result



* Post Increment operator

1. Assign/compute
2. Increment

eg - x = 5
z = x++

o/p → z = 5
x = 6

* Pre Increment operator

1. Increment
2. Assign

eg - x = 5
y = ++x

o/p → x = 6
y = 6

* precedence - power of operators

(which is more powerful that we will use first)

* Associavity - If there are operators of same precedence we use Associavity i.e Left to Right or right to left

category	operators	Associavity
Postfix	++ --	L → R
unary	++ -- + - ! ~	Right to left
Multiplicative	* / %	L → R
Additive	+ -	"
shift	<< >> >>>	"
Relational	< <= > >=	"
Equality	== !=	"
Bitwise AND	&	"
Bitwise XOR	^	"
Bitwise OR	|	"
logical AND	&&	"
conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= &= |=	Right to left

PRECEDENCE ↓

- pdf Notes
- Assignment Questions

12

Bitwise Operators

* Bitwise OR → if any 1 bit is also 1 it gives 1

  64 32 16 8 4 2 1

  eg- 9 → 1 0 0 1
  10 → 1 0 1 0
  ~~9 10~~ 9 | 10
  1 0 1 1 ⇒ 11

* Bitwise AND → both bits should be 1 then it gives 1

  9 & 10
  9 → 1 0 0 1
  10 → 1 0 1 0
  1 0 0 0 ⇒ 8

* Bitwise XOR → if both bits are same then o/p is 0
  if both bits are different then o/p is 1

  9 ^ 10
  9 → 1 0 0 1
  10 → 1 0 1 0
  0 0 1 1 ⇒ 3

* Bitwise complement → inverts all the bits (NOT)

  eg- 9 → 1 0 0 1
  ~9 → 0 1 1 0 ⇒ 6

* left shift (2 operands are required)

  eg- 9 << 1 ⇒ 18

  The No we want to operate | How many times we want to operate

  9 << 1
  _ 1 0 0 1
  1 0 0 1 0 ⇒ 18

* 9 << 2

  _ _ 1 0 0 1
  1 0 0 1 0 0 ⇒ 36

* Right shift

  eg → 9 >> 1

  1 0 0 1 _
  1 0 0 ⇒ 4

* 9 >> 2

  1 0 0 1 _ _
  1 0 ⇒ 2

Shortcut

left shift -
$a << b = a * 2^b$

Right shift -
$a >> b = \frac{a}{2^b}$

7

Conditionals

* If statement

  Syntax -

```
if (condition) {
  //code
}
```

→ If we not keep curly braces after the if statement only 1st line will execute.

* If-else statement

  Syntax -

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

* If-else-if statement - used for multiple conditions

Syntax:

```
if(condition-1){
    //code
}else if(condition-2){
    //code
}else{
    //code
}
```

* Nested if-else statement

```
if(condition-1){
    if(condition-2){
        // code
    }else{
        //code
    }
}else{
    //code
}
```

* conditional operators

→ logical-AND operator &&

→ logical-OR operator ||

→ Ternary operator

Syntax: `String result = (condition-1) ? ans1 : ans2;`

if condition-1 is true answer is ans1 else ans 2

- Ternary operator must return some result. so only the variable result is used in syntax.

* switch case

Syntax:

```
Switch (expression){
    case x:
        //code
        break;
    case y:
        //code
        break;
    default:
        //code
}
```

- pdf Notes
- Assignment Questions

13

Introduction to OOPS in Java

* Attributes are also called as Member Variables

* characteristics of object -

			eg -
1	Identity	unique name or object	BMW, Toyata
2	State	attribute	color, Model No ↓ ↓ red 750i
3	Behaviour	Methods	Accelerate, break

* create a class

- Access Modifier
- class keyword
- class name
- Body

```
public class Car{
    //class Body
}
```

* Access Modifiers

- public → Accessible by everyone
  eg- roads, streetlights
- private → eg- Mobile phone, whatsapp text (only you can Access)
- Protected → eg- assets like property (we can Inherit our parents property)
- default → If we dont keep any access modifier it comes under default case
  eg- TVs, fridge
  ↓
  Accessible within home but not outside

* create a class Name person with a Variable age.

```
public class person1
    int age = 19;
    psvm()
    }
}
```

* create an object

Syntax-

```
ClassName objectName = new className();
```

eg- Car BMW = new Car(); → constructor

Person Rohan = new Person();

* create an object of person class

```
public class Person1
    int age = 19;
    public static void main(String[]args) {
        Person Suraj = new Person();
        Sout(Suraj.age);   o/p = 19
}
```

* How to Access the class Attributes / object Attributes or methods.

using dot operator we can Access

Syntax:
objectName . Attribute/Method.

eg- refer previous page last line

* If we not assign any values for Attributes in class the o/p will be null for string & 0 for Integer. (Because it is member Variable)

* In one file there will be only one public class & In one java file we can have multiple classes

* The class for which we will keep public should be same as filename.

* In real time development it is better practice to write main method in different class & the class for which we want to perform in different class.

* We can create multiple objects in one single class

* new keyword is used to Allocate memory in Java

* creating an object from a class is known as Instantiating

* oop is a programming paradigm based on the concept of classes & objects.

- pdf Notes
- Assignment Questions

14

Methods in Java

* Java Methods

It is a block of code performing some action which runs only when it is called.

* Suppose we have a program to calculate a factorial of a number multiple times. Instead of writing factorial logic multiple times call that method wherever necessary

big code => { factorial();
factorial(); }

↳ Instead write this ↑

* functions & methods are the 2 words which are used interchangably but there is a certain difference it is functions are called just by their name eg- bark() etc but methods is called using an object any function inside a class which has to be called using an object called as method.

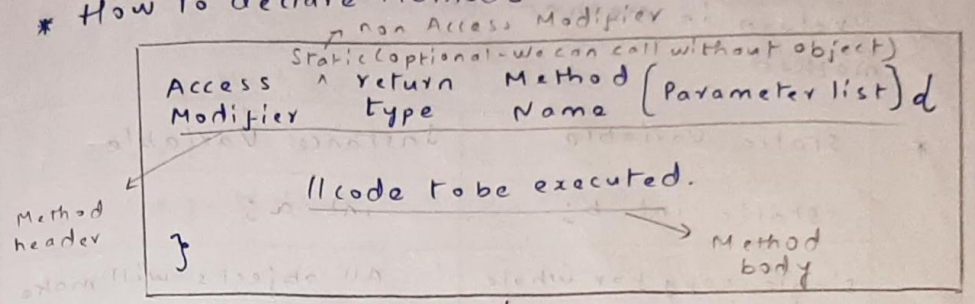
eg- Suppose we have bark() function inside an class and we have an object Fluffy using fluffy if we call bark() i.e fluffy.bark() called as an method.

* Method is the correct word in java because everything in java is written in classes only.

* Why are methods important in Java
  - → write once, reuse many times
  - → Time save
  - → Duplicate code reduces
  - → readable and Maintainable

* Types of Methods
  1. User defined Methods
  2. standard library Methods (built-in)

* How to declare Methods



eg- public class Main {

```
public int sum(int a, int b){
    //code to be executed.
}
```

}

* Method signature => Method Name + Parameter list
  Sum(int a, int b)

* Access specifie => public, protected, private, Default eg- public

* Returntype => the o/p type we want to return eg- int

* Method Name => eg- Sum

* parameters => eg- int a, int b

* Method body => eg- int sum = a+b; return sum;

* Calling a Method

To call a method in java, you have to write the Method's Name followed by two parenthesis () a semicolon.

eg. Sum();

* If we are using void as return type in a method use sout in method only or else if we are using some return type in method like int, string use return in method and sout in Main method.

*

static Variable	Instance (normal) Variable
static int b;	int a;
single copy for whole class (object independent)	All objects will make their own copy.

* Methods & Variables can be static.

* Java program to add 2 Numbers using Methods.

```
import java.util.Scanner;
class AddTwoNumbers{
    public int Add(int a, int b){
        int sum = a+b;
        return sum;
    }
}
public class Main{
    public static void main(String[] args){
        AddTwoNumbers obj = new AddTwoNumbers();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter First Number");
        int a = sc.nextInt();
        System.out.println("Enter Second Number");
        int b = sc.nextInt();
        int Sum = obj.Add(a, b);
        System.out.println(Sum);
    }
}
```

* Standard library Methods

1. sqrt

   eg. Math.sqrt(24); // 4.89

2. floor

   eg. Math.floor(5.3); // 5.0

3. ceil

   eg. Math.ceil(5.3); // 6.0

4. pow (requires 2 values) pow(x, y) => x^y

   eg. Math.pow(5, 2); (5^2) (a^b) ($a \wedge b$) // 25.0

- sqrt, floor, ceil, pow return double value

* return; doesn't change anything if we have void return type

* Packages (refer IntelliJ code for creating & Importing a package)

  It is a collection of similar classes, subpackages & interfaces

* How to create an package

  In IntelliJ Idea click on

  src → New → package

  again click on Name of the package and create one more package called as subpackage

* we can write Multiple classes inside the package and we can import them outside the package. by creating an object.

In IntelliJ after creating an package and a class inside the package. come out of the package and create a class. after that create an object of the class that we have created in the package. It will come like (import pw.skills.App) ← eg

* If we write (import pw.skills.*;) it will import all the classes)

* Constructors (refer VS code for better understanding)

It is a special type of method which is used to initialize objects

→ every class has an default constructor. If we not make a constructor it will call the default constructor.

eg- Alegebra obj=new Alegebra();
↓
default constructor

→ Constructor name & class Name are equal & constructor doesnt have any return type

* refer code in VS code for how to create our own constructor

→ If we pass the parameters by creating our own constructor then it is called as parameterized constructor

* Access Modifiers
-refer lec-13 Notes

Access Modifier	within class	outside class Same package	outside package non-child class	outside package child class
public	✓	✓	✓	✓
Protected	✓	✓	X	✓
Private	✓	X	X	X
Default	✓	✓	X	X

- pdf Notes
- Assignment Questions

15

Scope of Variables

* region from where a variable can be Accessed

* In Same scope any 2 variables names cant be same

* In different scope Variable names can be Same

* 3 levels of scope
1 class level scope
2 method level scope
3 Block level scope

* class level scope

eg- class Algebra {
    int a = 10; // Member Variables
    int b = 5;
    int add() {
        return a+b;
    }
    int sub() {
        return a-b;
    }
}

→ The variables a, b can be Accessed anywhere in whole class ~~anywhere~~ (in method, constructor anywhere it can be accessed).

* Method level scope

These are the variables that are declared inside the method and cannot be Accessed outside the method.

eg- class Algebra {
    int add() {
        int p = 10;
        int q = 5;
        return p+q;
    }
    int sub() {
        return p-q; ✗ // from above we cannot access p & q outside the method
        int p = 20
        int q = 10
        return p-q; ✓ // we have to declare again
    }
}

* Block level scope

These are the variables that are declared inside curly braces '{' & '}'

→ A block of code may exist on its own

eg {
    //code
}

→ or it can belong to an if/for or any statement

eg-
class Algebra {
    void demo() {
        {
            int b = 10;
            Sout(b); // b can be Accessed
        }
        ✗ Sout(b); // b cannot be Accessed because it is out of scope
        {
            int b = 5;
            Sout(b);
        }
    }
}
public class Main {
    psvm(String[] args) {
        Algebra obj = new Algebra();
        obj.demo();
    }
}

o/p- 10
5

* Formal Parameters / Just parameters
these are defined during function definition

eg- public static int sum(int a, int b){ ← Formal parameters

return a+b;

}

* Actual Parameters
The parameters when that we are passing when we are calling a function.

eg- Sum(5, 2);
Actual parameters / Arguments

olp - 7

* Pass by value and Pass by reference are 2 different ways to pass Actual Parameters

* Pass by value copies the value of Actual parameters & the called function creates its own copyies.

* In pass by reference they passes the parameters as reference / address & the called function does not creates its own copy. (It operates on the original values that are passed).

* Pass by value example

```
public class Main {
    static void changevalue(int a) {
        a=a*100;
        Sout("Inside value:" + a)
    }
    public static void main(String[] args) {
        int a=10;
        Sout("Before changing value:" +a)
        change value(a);
        Sout(" After changing value:" +a)
    }
}
```

olp - Before changing value: 10
Inside value: 1000
After changing value: 10 // doesn't change because java uses pass by value

* The value a of method and the value a of main method are different because both Addresses are different because pass by value copies the value of Actual parameters & the called function creates its own copy

* Java always has pass by value java doesn't have pass by reference

* Memory Address is represented in Hexa decimal

• Pdf Notes

8.

Loops

* Iterative statements / Loops
  They are helpful if we want to do things repeatedly
  eg- we can print "HelloWorld" 1000 times using loops in few seconds.

* Types of Loops (refer VS-code codes for example)
  → while Loop
  → For Loop
  → Do-while Loop

* while Loop
  syntax:

```
while (condition) {
    // code
}
```

* For Loop
  syntax:

```
for(init-statement; condition; final-expression){
    //code
}
```

* Sum of first n natural numbers Formula

$$\frac{n(n+1)}{2}$$

* Sum of square of n natural numbers using Formula

$$\frac{n(n+1)(2n+1)}{6}$$

* In for loop : 1 init statement
  2 condition
  3 final expression
  } are optional but Semicolon is must.
  i.e for(; ;){

}

but we have to initialize somewhere to run the for loop Acc to question.

eg-

```
int n = 10
int i = 1;
for( ; i<=n; ){
    cout(i);
    i++
}
```

* Do-while Loop

syntax:

```
do {
    //code
} while(condition)
```

* first Iteration is done before checking the condition from second Iteration the condition is checked.

Q print the sum of stream of integers
user gives the numbers sum all of them
until the given condition is true
eg:- if num is -1 stop & print sum.

* what to choose between for & while Loop
→ If you know that how many times you have to do the task then use for loop else use while loop.

* break keyword
It stops the itteration Acc. to the condition.

```
eg- for(i=1; i<10; i++)
    {
      if(i==5){
        break;
      }
      Sout(i)
    }
```

o/p - 1, 2, 3, 4 (it doesn't print up to 5) because of break keyword

* continue keyword
(current itteration stops & next itteration begins)
eg- In ticket counter 1st person, 2nd begins 3rd... so on are in a queue for taking tickets 1st person, 2nd person will take tickets but if 3rd person got some work and he dont want to take tickets he will come out of the queue and from 4th person the tickets are immersed (meaning that 3rd person will say continue & the queue will begin from 4th person)

// print all the values between 1 to 50 except multiples of 3.

For Loop

```
for(int i=1;i<=50;i++)
{
  if(i%3==0){
    continue;
  }
  Sout(i);
}
```

while Loop

```
int i=1;
while(i<=50){
  if(i%3==0){
    i++; //difference but
    continue; for & while
  }          loop.
  Sout(i);
  i++;
}
```

(In for loop it increments the value after continue keyword but in while it doesn't so we have to use another increment in while loop)

* using labels with continue & break keyword

eg -

```
myloop: for(int i=1; i<=50; i++){
          if(i%3==0){
            continue myloop;
          }
          Sout(i);
        }
```

→ there is no difference but if we have nested loops labels are important

eg -

```
for(int i=1;i<=10;i++){
  for(int j=1;j<=10;j++){
    break;
  }
}
```

→ break keyword will be for 2nd for loop but if we want 1st for loop to use break use labels i.e.

```
outer: for(int i=1;i<10;i++){
  inner: for(int j=1;j<10;j++){
    break outer;
  }
}
```

9

Problems on single Loop

* Count the number of digits for a given number n

eg-
num = 9104
count = 4

→ 9104/10 = 910
→ 910/10 = 91
→ 91/10 = 9
→ 9/10 = 0.9

hint:
→ num/10 = removes the last digit

Algorithm -

* Take i/p from the user
* Iterate an while loop until the number is not equal to 0
* divide the number by 10 to remove the last digit i.e
  12345/10 = 1234
* Parallely increment count.
* repeat step 3.

* Sum of digits for a given number n

eg-
num = 9104
Ans = 14.

hint:
→ num%10 (gives last digit)
→ num/10 (removes last digit)

→ num = 9104
Sum = 0
while (num != 0)
Ans = 9104 %10 = 4
Sum = Sum + Ans;
num = num/10;
print sum.

* Reverse the digits of a number

eg-
num = 9104
Ans = 4019

→ 9104 %10 = 4
→ Sum = (0*10)+4
→ 9104/10 = 910

hint:
multiply the Ans by 10 and add the last digit

Algorithm -

* Take Input from the user
* Iterate an while loop until until the num not equal to 0.
* num%10 = lastDigit
* Sum = (sum*10) + lastdigit (Sum = 0)
* num/10 = removes last digit

* print the first n factorial numbers

eg-
1. num = 2
   Ans = 1×2 = 2
2. num = 4
   Ans = 1×2×3×4 = 24

Logic:
hint:
→ multiply all the previous numbers till the given number

fact = 1
fact = fact * i

* find a raised to the power b

eg- a^b
2^5 => 2×2×2×2×2

hint:
Iterate till b

* Sum of following series S = (1-2+3-4..n)

1-2+3-4 = -2

hint:
even = subtract
odd = add

• pdf Notes

10 problems on Nested Loops

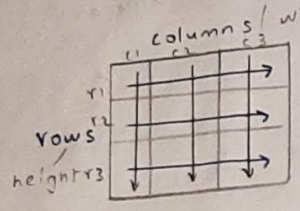
* Nested Loops

```
for(int i=1;i<=3;i++){
    for(int j=1;j<=3;j++){
        System.out.print(j+" ");
    }
    System.out.println();
}
```

(" " → space; print → prints on same line; println → after one iteration it takes on next line.)

o/p -
```
1 2 3
1 2 3
1 2 3
```

*



* Iterative variables naming convention is i & j (we can use other names also)
* Pattern printing

1. Rectangular Pattern

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

n = 3
w = 7

Inner for loop:
n1 → 7 stars
n2 → 7 stars
n3 → 7 stars

↓ outer for loop

Algorithm:

* Input the height & width i.e n = 3, w = 7
* For the height use the outer loop
* For the width use the inner loop
* after printing all columns of a row, print a new line after the inner loop.

2. Hollow Rectangular Pattern

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

h = 4 w = 6

print the star for 1st & last row & column, other wise print blank space. (h) (w)

3 Triangular Pattern

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

Iterate inner loop until the value of outer loop

4. Inverted triangular Pattern

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

Iterate inner loop until the value of rows needed add +1 & subtract with outer loop value i.e i (r+1-i)

method 2 - Iterate the outer loop in reverse order & inner loop iterate it to the value of outer loop.

1 Analyze rows & columns
2 Derive relation between rows & columns
3 what to print & where to print

5. Pyramid pattern

h = 4

	spaces	stars
h1 →	3 (4-1)	1 (2*1-1)
h2 →	2 (4-2)	3 (2*2-1)
h3 →	1 (4-3)	5 (2*3-1)
h4 →	0 (4-4)	7 (2*4-1)
	(h-i)	(2*i-1)

5 Numerical Rectangular Pattern - Type 1 h=7

```
1 2 3 4 5 6 7
2 3 4 5 6 7 1
3 4 5 6 7 1 2
4 5 6 7 1 2 3
5 6 7 1 2 3 4
6 7 1 2 3 4 5
7 1 2 3 4 5 6
```

h1 → (1 → 7)
h2 → (2 → 7) (1)
h3 → (3 → 7) (1-2)
h4 → (4 → 7) (1-3)
h5 → (5 → 7) (1-4)
h6 → (6 → 7) (1-5)
h7 → (7) (1-6)

(i → h) (1 → i-1)

6 Numerical Rectangular Pattern type-2

```
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
```

(1 - r)
(1 - c)

7 Numerical Rectangular Pattern type 3

```
1 2 3 4 5
        5
1       5
1       5
1       5
1 2 3 4 5
```

8 Numerical Rectangular Pattern type 4

```
1 2 1 2 1 2
2 1 2 1 2 1
1 2 1 2 1 2
2 1 2 1 2 1
```

r = i, c = j

11 1	12 2	13 1	14 2
21 2	22 1	23 2	24 1
31 1	32 2	33 1	34 2

(i+j) % 2 == 0 => 1 else 2

9 Numerical Triangular Pattern

```
1
1 2
1 2 3
1 2 3 4
```

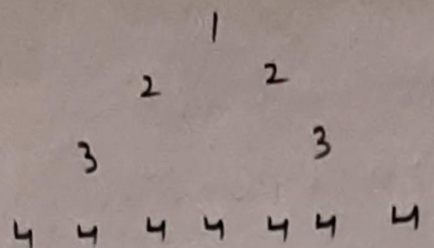
10 Numerical Pyramid pattern r=4

```
      1
    1 2 1
  1 2 3 2 1
1 2 3 4 3 2 1
```

	spaces	Numbers
R1 →	3 (4-1)	1
R2 →	2 (4-2)	1 to 2 & 1
R3 →	1 (4-3)	1 to 3 & 2 to 1
R4 →	0 (4-4)	1 to 4 & 3 to 1
	1 to (r-i)	(1 to i) & (i-1) to 1

(HW)

11. Hollow Triangular Pyramid



- pdf Notes
- Assignment questions.

2

Installation of IntelliJ (Watch video)

(Download from Document given)

step1 - download JDK (x64 Installer)

step2 - download IntelliJ (community version)

3

Installation of VS code (Watch video)

(For c++)

1 Dowload MinGW (sourceforge.net)

2 Download vscode

3 Dowload JDK