JAVA PROGRAMMING

**MEMORY:**-Sequence of bytes.

## Basic Rule or Camel Casing:

Class Name – First letter is capital

Function name or method name –first Word Letter Small and another Word letters are Capital.

Object – Use new key word to create object by using classes.

Constraints – Capital all letters e.g.: PIE,BRAND.

## GOOD PROGRAMING PRACTICE:

##### Curly Braces:

##### Tabs:

##### Exception Handling:

Try{

All statements;

}

catch(Exception ex){

System.out.print(“Error occurred:”+ex.getMessage());

}

catch(Arithmetic Exception ex){

}

finaly{  
 }

##### Commenting:

// ->single line comment

/\*

\*/ -> multiline comment.

##### Debugging: Breakpoint, Watching values.

## SPECIAL PROPERTIES:

Variable\_name.length -> To get Actual length of an variable. Eg : array.

## Access Modifiers:

Private: Specific Class.

Default: Specific Package.

Public: Any class and package.

Protected: Subsiding Class - Same class and same package, Different Package subclass.

## RESERVE AND INITIALIZATION:

### Primitive data types:

* byte
* boolean
* char
* short
* int
* long
* float
* double

### Wrapper Classes: (objects) for some process.

* Byte
* Boolean
* Character
* Short
* Integer
* Long
* Float
* Double
* String

Auto Boxing: When you store primitive type in wrapper type.

Auto Un Boxing: When you store wrapper type into primitive.

Eg:

  int n1 = 10;

        Integer n2 = n1;

        System.out.println(n2);

        Integer n3 = 100;

        int n4 = n3;

        System.out.println(n4);

## String:

String constant pool-in heap memory for reduce the memory.

#### Mutable String: change

##### String Buffer:

To change Strings

Syntax:

StringBuffer Alays\_Name=new StringBuffer(Str:”navin”);

String Builder: Same as StringBuffer.

#### Immutable String: Unchanged

Default String is immutable.

### Constants or literals:

Decimal - 45,200,…

Octal –07,066,0……

Integer Hexadecimal –0xA,0xbb,0x…

Binary –0b….

Using underscore – 9000\_20000- for readability

Float Normel-o.4,23.00,….

3\*10^5 – 3e5,3E+5,300000.

Character – single quotes (‘ ’)

String – double quotes (“ “)

Backspace - \b

Special character New line - \n

Horizontal tab - \t

Form feed –\f

Carriage Return - \r

Boolean True

False

Null – null.

# OUTPUT:

## System. Out(PrintStream)

* **System. out. print – unformatted output**
* **System.out.println - unformatted output**

Eg: System.out.print(a + ”\n”);

* **System.out.printf– formatted output**

Eg: System.out.printf(“\n%d”,a);

## Aliasing System. out:

Step 1: Import java.io.PrintStream;

Step2: PrintStream cout=System.out;

Step3: use cout.printf(“”);

#### FORMAT SPECIFIERS:

%b – Boolean

%d – Integers

%f – Float

%s – String

%s – Date

%tT -24 hour format

%tr – 12 hour format

##### Format specifiers for date time component:

$tH – Hour part

$tM – Minute part

$ts – Second part

$tA – Weak days

$TA – Weak days (caps)

$ta – Weak days (simple formate)

$tB – Month name

$TB – Month name (caps)

$tb – Month name (simple format)

$td – Date part (2 digits)

$te – Date part (1 or 2 digits)

$tY – Year part (4 Digits)

$ty – year part (2 digits)

Eg : System.out.printf(“%1$tH:%1$tM:%1$tS”,Calendar.getInstance());

## INPUT:

#### Scanner:

* Inbuilt object
* Use to get console input
* Import java.util.Scanner;

Syntax:

Scanner alays\_name = new Scanner (system.in);

New -> to create Physical objects.

Eg: Scanner in = new Scanner(system.in);

nex<Type>(); -> to store input into the variable .

Types:

* nextLine() -> get next string
* next() -> get single word
* nextInt -> get integer
* nextDouble ->get floation point numbers

Syntax:

Variable\_name = alays\_name.nex<type>();

Eg: NoOfYears = in.nextInt();

## BufferedReader:

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        int N = Integer.parseInt(bufferedReader.readLine().trim());

## ALU OPERATIONS:

### BINARY OPERATORS:

* SIMPLE ARITHMETIC OPERATORS
* BITWISE SHIFT OPERATORS
* RELATION OPERATORS

### UNARY OPERATORS:

* INCREMENT OPERATORS

## Math. Lib Methods:

1. Math.random() ->Give double value as Result.
2. Math.pow(Base value,Squer value)
3. Math.sqrt(value)

### SIMPLE ARITHMETIC OPERATORS:

1. +

2. –

3. \*

4./ -> Type casting by using (float). Eg: a=(float)b/(float)c;

5. %

##### INCREMENTER AND DECREMENTER:

1. ++

2.—

##### OPERATOR PRECEDENCE:

1. ()

2. ++,--

3. \*,/,%

4. +,-

##### STANDARD LIBRARY FUNCTION FOR CALCULATIONS (MATH LIBRARY)

1. Math.pow(a,5);

### RELATIONAL OPERATORS (FOR DECISION MAKING)

1. ==

2. !=

3. <

4. >

5. <=

6. >=

### LOGICAL OPERATORS:

1. &&

2. ||

3.!

### BITWISE SHIFT OPERATORS:

1. <<
2. >>
3. >>>

### BITWISE OPERATORS:

1. & -> Bitwise AND
2. | ->Bitwise Inclusive OR
3. ^ ->Bitwise Exclusive OR

### TERNARY OPERATORS:

1. : ? – short cut for if else statements for some use case.

### ASSIGNMENT OPERATORS:

1. +=
2. -=
3. \*=
4. /=
5. =
6. %=
7. ^=
8. &=
9. |=
10. <<=
11. >>=
12. >>>=

## DATA CONVERSION:

##### Data types:

1. Primitive data types
2. Wrapper class data types

##### Conversion Methods:

1. Primitive to wrapper
2. Wrapper to primitive.

##### Primitive to wrapper: valueOf()

Eg:

int a = 10;

Integer b;

b = Integer.valueOf(a);

##### Wrapper to primitive: typeValue()

Eg:

a =b.intValue();

##### Parse function: parse<Type>(variable Name);

Eg:

String s= “124.88”;

double a; fuction Name

a=Double.parseDouble(s);

Wrapper Class Name

Eg2:

S= Double.valueof(a+5).toString(); -> Merge to functions.

## Type casting: one data type to another data type conversion.

double d = 1.4;

        System.out.println(d);

        int e = (int) d;

        System.out.println(e);

#### Up casting: Assign sub class to super class

 a in = new b();

#### Down casting: Assign super class to sub class

b in1 = (b) in;

## CONTROL STATEMENTS:

### IF ELSE IF ELSE: (Multiple conditions one value)

1. If(conditions)

Statements;

1. If(condition)

Statements;

Else

Statements;

1. if (condition)

statement;

else if(condition)

statement;

else if(condition)

statement;

else

statement;

### SWITCH ..CASE : (One condition multiple values)



Switch(condition Name) Eg:

{

case condition value:

Statements;

break;

case condition value:

statements;

break;

default:

Statements;

}

### UPDATED SWITCH CASE STATEMENT:(-> operator or :yield key word instead of break;)

Result=Switch(condition Name)

{

case Multiple condition values(by separate by , operator) ->Statement;

case condition value ->statement;

default ->Statements;

}

## LOOPING STATEMENTS:



1. WHILE(CONDITION)

{

STATEMENTS;

}

1. do{

Statements;

} while(condition)

3. For loop

For( initialization; condition; increment)

{

Statement;

}

3. for(initialization; condition; increment)

{

Statements;

}

4. Enhanced for loop:

For(data\_type allays: Name of variable)

Eg:

public static void main(String[] args) {

    int n[] = new int[4];

    Scanner in = new Scanner(System.in);

    for (int i = 0; i < n.length; i++) {

      n[i] = in.nextInt();

    }

    for (int nw : n) {

      System.out.println(nw);

    }

  }

## Methods:

Normal Method: Normal format.

Method Overloading:

Same method name but multiple parameters and variable types.

Eg:

* Public void add(int a, int b)
* Public void add (float a, int b)
* Public void add(int a, int b, int c)

## Method overriding:

* Use inheritance area.
* It has same method name and parameters but deferent class.
* It will avoid super class method.
* It will accept sub class method.

Eg:

    public static void main(String[] args) {

        b in = new b();

        in.show();

    }

}

class a {

    public void show() {

        System.out.println("in super class");

    }

}

class b extends a {

    public void show() {

        System.out.println("in sub class");

    }

}

## Stack and Heap Memory in JVM:

### Inside tools in JVM:

Stack

Heap

### Stack:

* Every methods has own stack.
* Store date in Sequence.
* It store only local variables.

### Heap:

* Common for a class.
* It store Instance variable.
* String constant pool in heap.

## Array:

* Store Sequence of data as same data type.

Types:

1. Single Array
2. Multi dimension Array
3. Jagged Array

### Declaration and Initialization:

Single dimension Array:

Int nums[]={,,,,};

Int nums[]=new int[4];

Multi-dimension Array:

Int nums[][]=new int[2][2];

Jagged Array:

Int num[][]=new int[3][];

* + - Num[0]=new int[3];
    - Num[1]=new int[4];
    - Num[2]=new int[2];
  + To store and Retrieve memory from jagged array using variable\_name.length key word.
  + Eg: for(j=0;i<num.length;j++)

for(i=0; i<num[i].length; i++)

#### Drawbacks of Array:

* Size is fixed
* Can store only one data types values

### Array of Object:

Can create class as a array type

Eg:

Class\_name Allays\_Name[]=new Class\_Name[5];

## Important key Words:

Static (static) - Common for all, directly use with the class name.

New (new) – Class to object.

This (this) – It’s Represent current object.

Extends (extends) – join more classes together. Eg : class B extends A{}

Pakege (pakege) – to create new folder.

Import – to import the pakeges.

Final(final) :

* Variable – to create constant variable.
* Method – to strop overriding.
* Class – to stop inheritance.

Abstract:

* Cannot create direct object.
* Can create reference abstract class.
* Abstract methods only within the Abstract class.

Interface:

Type coupling.

Instead of Abstract key word.

Use implements instead of extends.

(Switch case concept)

## Important Method:

Supper() (supper())– Execute the constructor of supper class.

This() (this()) – Execute the constructor of same class.

## CONSTRUCTORS:

* To set default values.
* Constructor names same as class name.
* Constructor newer contain return data type.
* Every time if you create new object it will call the constructor.

Eg:

Class name=Dinesh{};

Constructor name=public Dinesh(){}

### CONSTRUCTOR OVERLOADING:

To give optional parameters.

Eg:

public class newl {

    public static void main(String[] args) {

        System.out.println("Dinesh");

        human a = new human("raja", 22);

        System.out.println(a.name + a.age);

        human b = new human();

        System.out.println(b.name + b.age);

    }

}

class human {

    String name;

    int age;

    public human() {

        name = "Dinesh";

        age = 22;

    }

    public human(String a, int b) {

        name = a;

        age = b;

    }

}

# OBJECT ORIENTED PROGRAMING:

Object – Properties and Behavior.

Use – new key word to create object.

## CLASS:

It Is a template or Blueprint.

### Encapsulation:

* To protect the members.
* Private key word use this method.

Eg:

#### Variable:

Private datatype variable\_Name;

#### Method or Function:

Private returndatatype method\_name (parameters){}

### Inheritance:

* Code re-usability.
* Extends key word use this method.
* In this method data inherit from Super class to sub class. (super – sub or parent – child or base – derived)
* Add additional feature without change super class.
* Java don’t support multiple inheritance -> because Ambiguity problem.

Eg:

#### Single level inheritance:

Class sub\_class\_name extends super\_class\_name{}

#### Multi level inheritance:

Class new\_sub\_class \_name extends super\_sub-class\_name{}

Super class <- super sub class <- sub class,,,,,,,,,,,,,,.

## POLYMORPHISM:

* Means – Many behavior.
* Execution will change based on situation.
* Can assign derived class to base class type.
* Overall control of multiple deferent types of classes.

TYPE:

Compile time Polymorphism: -> Overloading.

Run time Polymorphism: ->Overriding.

Eg:

public class newl {

    public static void main(String[] args) {

        a in = new b();

        in.show();

        b cg = new b();

        in.show();

    }

}

class a {

    public void show() {

        System.out.println("in super class");

    }

}

class b extends a {

    public void show() {

        System.out.println("in sub class");

    }

}

### INNER CLASS: Create a Class within the class.

#### Creation of inner class:

class A {

    void show() {

        System.out.println("In Show ");

    }

    class B {

        void show1() {

            System.out.println("IN Show1");

        }

    }

}

#### Declare the Object:

A obj=new A();

        obj.show();

        A.B obj1=obj.new B();

        obj1.show1();

## If you create inner class as static:

##### Class creation:

class A {

    void show() {

        System.out.println("In Show ");

    }

    static class B {

        void show1() {

            System.out.println("IN Show1");

        }

    }

}

##### Declaration:

 A obj = new A();

        obj.show();

        A.B obj1 = new A.B();

        obj1.show1();

### Anonymous inner class:

Use only Mani function.

Syntax: A obj = new A() {methods or variables};

public static void main(String[] args) {

        A obj = new A() {

            void show() {

                System.out.println("New Show ");

            }

        };

# INTERFACE:

Interface is not a class

It is a design format

All the variable by default final and static in the interface

All the methods by default public abstract in the interface

interface A {

int a=10; ->Final and Static variable so we want to initialize the variable.

string b;

void show(); ->if a method closed by (;) this method called public abstract method.

void config(); (Short cut of public abstract method)

}

Implements key word uses to create a implement class of interface class.

interface A {

    void show();

    void config();

}

class B implements A{

}

class C implements A{

}

The type B must implement the inherited abstract method A.config()

The type C must implement the inherited abstract method A.config()

 public static void main(String[] args) {

               Scanner in = new Scanner(System.in);

        System.out.println("Enter the input");

        int a = in.nextInt();

A obj;

        if (a == 1) {

            obj = new B(); ->Declaration

       obj.config();

       obj.show();

        } else {

            obj = new C();

            obj.config();

            obj.show();

        }

    }

}

interface A {

    void show();

    void config();

}

class B implements A {

    public void show() {

        System.out.println("in B show");

    }

    public void config() {

        System.out.println("in B config");

    }

}

class C implements A {

    public void show() {

        System.out.println("in C show");

    }

    public void config() {

        System.out.println("in C config");

    }

}

#### MULTIPLE INTERFACES:

interface A {

    void show();

    void config();

}

interface x{

    void run();

}

**class B implements A,x { ->Defining the multiple interfaces.**

    public void show() {

        System.out.println("in B show");

    }

    public void config() {

        System.out.println("in B config");

    }

    public void run(){

        System.out.println("in running"); ->We want to implement new interface abstract method also.

    }

}

#### INHERITENCE OF INTERFACE:

interface C extends A {

       void show();

    void config();

    }

#### REMEMBER:

Class inherit Class ->Extends

Interface inherits Interface ->Extends

Class inherits Interface ->Implements