I 10/9/19

# 1. Fundamentals Of OOP

Unit 1

* Programming Paradigms

- Programming
  Collection of instructions that perform a specific task

- Paradigm
  It is also to be termed as method (or) processor to solve can some problem.

- Programming Paradigm
  It is an approach to solve a problem using programming language.

* Types of Programming Paradigms

(1) Procedure oriented programming (POP) (or) System oriented programming

(2) Object oriented programming (OOP)

(3) Moduler programming

(4) Generic programming

(1) Procedure Oriented Programming (POP)

A program in a procedure language is a list of instructions where each statement tells the computer to do something.

→ It focussing on procedures (functions) and algorithms

→ Where the program become larger it is divided into functions and each function has clearly defined

Ex:
```
main()
{
  int a,b,c,d,e,f;
  c=a+b;
  d=a-b;
  e=a*b;
  f=a/b;
}
```

```
sum();
{
c=a+b;
pf(c);
}
```

→ Diiuding the programming into functions / modules.

Ex: C, PASCAL, FORTRAN, BASIC
of POP

* Characteristics of POP

1) It is focussing on process / procedure rather than data.
2) Functions are written to solve a problem.
3) Most of the functions share the global data.

Ex:
```
int top=-1;
main()
{ push()
  { top++;
  }
  pop()
  { top--;
  }
}
```

4) It is a top-down approach and highest priority is given to functions.

* Disadvantages of POP

(i) Every function has complete access to the global variables.

(ii) The new programmer can correct the data accidentally by creating new function. (iii) Similarly if new data is to be added, all the function needed to be modified to access the data.

(2) Object Oriented Programming (OOP)

It is a programming style which is associated with concept of classes & objects. And various other concepts. like

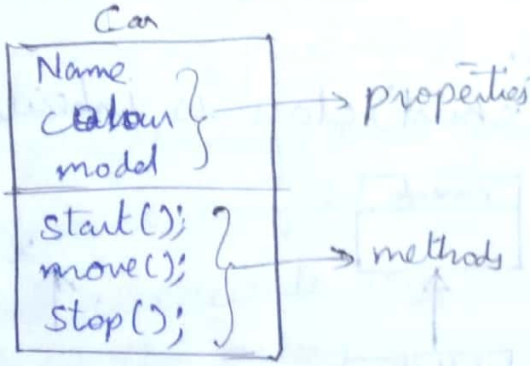
(i) Data Abstraction (ii) Encapsulation

(iii) Inheritance (iv) Polymorphism.

- Class:-

It is a blueprint of an object which defines properties (variables) and member functions (methods).

Ex:-




- Variables

These are member locations reserved for storing values, size of the memory reserve depending on datatypes.

- Method

It is a block of code which only runs when it is called, methods are used to perform specific tasks. * Method should be inside of the class.

- **Object**

Instance of a class.

Ex: (1) Orange, banana apples objects of fruits class.

(2) Chairs, tables, sofas objects of furniture class.

(i) **Data Abstraction**

Representing the essential features (or) highlighting the essential information & hide the internal implementation (details).

Ex: Car Stirring, ATM System

(ii) **Encapsulation**

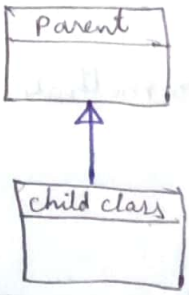
Binding the data (or) the process of binding the data members and member functions.

Ex: Every JAVA class is an example encapsulation.

(iii) **Inheritance**

One class is allowed to inherited the features of another class.

Ex: derived (child) class is inherited by parent class.




(iv) **Polymorphism**

poly = many, morphism = forms

→ Single action in different ways.

Ex: Sum(), Sum(int a, int b), Sum(float a), Sum(double a, double b), Overloading & overriding

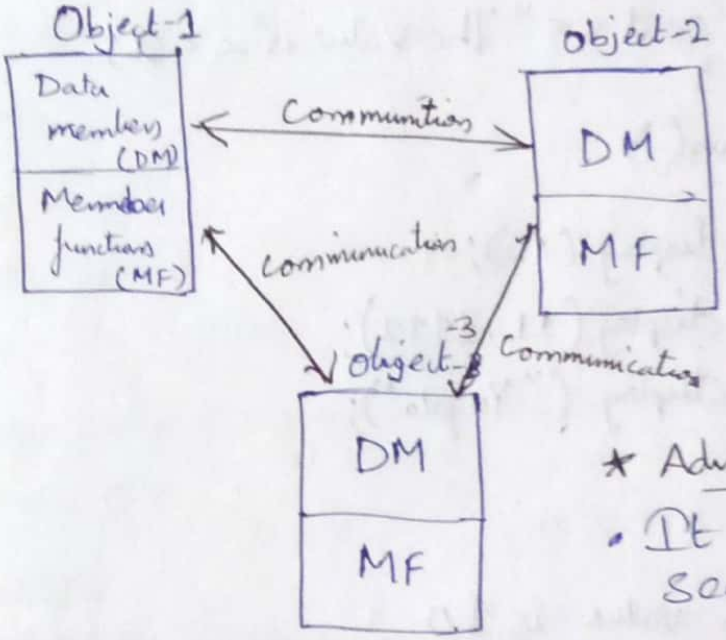
* Characteristics of OOP

- Improvement over the POP.
* Ex: data abstraction is introduced in addition.

* Languages of OOP

C++, JAVA, C#, Smalltalk.

* Structure of OOP




* Advatage

- It is more secure.

(3) Generic Programming

In this programming, that you r not writing the source code ie compile as it is.

→ But, that we write templates of source codes that compiler in the process of compilation, transforms into source code.

Ex: templates of C++.

* Structure of template

template < typename myt > return_type function name (arguments)

→ PTO

(myt - my template)

- template < typename myt >. returntype function name (arguments)
- Write a C++ program to display some number, decimal values & some string information using templates

```
template < typename T >
void display (T x)
{
  cout << " The value is " << x;
}
main()
{
  display (10);
  display (11.1111);
  display ("Yugala");
}
```

- Output

  The value is 10
  The value is 11.1111
  The value is Yugala.

---

## (4) Modular Programming

It is a programming paradigm, in which we divide the whole program into some sub programs (or) modules.

→ It is a main program broken into the sub parts.

Ex.

```
main()
{ module 1()          module 3()
  {                   {
  }                   }
  module 2()          module 4()
  {                   {
  }                   }
}
```

→ It leads to reusable the code, debugging and finding errors easily.

→ Take the POP and add the modularity.

→ Take the OOP & add the modularity.

- Advantages

– This approach allows simplicity and avoiding the complexity of large program.

– The 1000's of lines of code are divided into modules.

main adv

- Reusability

→ Allows user to reuse the functionalities without typing the whole program again.

→ The modular programming is closely related to POP & OOP

- Languages

ADA, COBOL, C#, FORTRAN, PYTHON, PERL.

13/6/17

Q. Write a java program to perform

(1) addition of two numbers.

(2) Arithematical operations

(3) To calculate area of circle.

(4) To find whether given number is +ve/-ve.

(5) to display 1-10 using while loop, do-while loop & for loop.

(6) To print stars using for loop.

(a)

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

(b)

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

7) To find the largest of two numbers & three numbers.
8) To calculate sum of five numbers.
9) To display factorial of a number ($n^n$, $n=5$)
10) To display correct grade of a student

1)
```
class add
{
  public static void main(String args[])
  {
    int a,b,c;
    a=100;
    b=200;
    c=a+b;
    System.out.println("The value c="+c);
  }
}
```
ln = new line
class, Keyword(output)

- Save: add.java
  ↳ ALL FILES
- '+' concatination of 2 statements
- compile: javac add.java
- argcos args (anything)
- Run: java add
- Output: The value c=300

3)
```
class areac
{
  public static void main(String args[])
  {
    double r=5, a;
    a=3.14*r*r;
    System.out.println("area of circle="+a);
  }
}
```

- Save: areac.java
- compile: javac areac.java
- run: java areac
- Output: area of circle=78.50

4)
```
class posneg
{
  public static void main(String args[])
  {
    int num=-5;
    if(num==0)
    {
      System.out.println("The number is zero");
    }
    else if(num<0)
    {
      System.out.println("negative number");
    }
    else
    {
      System.out.println("positive number");
    }
  }
}
```
Keywords, method

- Save: posneg.java
- Compile: javac posneg.java
- run: java posneg
- Output: negative number

5) Class Numwhile

(a)
```
{ public static void main (String args[])
  { int i=1;
    while(i<=10)
    { System.out.println("i");   // not necessary since it is not text msg.
      i=i+1;  // i++;
    }
  }
}
```

- Save: Numwhile.java
- Compile: javac numwhile.java
- Run: java numwhile
- Output:
1
2
3
4
5
6
7
8
9
10

numdowhile

(b)
```
int i=1;
do
{
System.out.println(i);
i++;
} while (i<=10);
```

int i; numfor

(c)
```
for(i=1; i<=10; i++)
{
  S.o.p(i);
}
```

6) (b)
```
int i,j;
for(i=1; i<=5; i++)   → Outer
{
  for(j=1; j<=i; j++)   → Inner
  {
    S.o.print("*");
  }
  S.o.println();
}
```

(a)
```
int i,j;
for(i=1; i<=5; i++)
{
  for(j=5; j>=i; j--)
  {
    S.o.print("*");
  }
  S.o.println();
}
```

19/7/19 (ii) BASICS OF JAVA | Unit-1

* Java Basics

(1) History of Java

In the year of 1990's sun microsystems (US company) has a Green Project to develop software for consumer electronics devices.

→ Java was developed by BILL JOY, JAMES GOSLING, PATRIK NAUGHTON at Sun microsystems in 1991.

→ James Gosling is father of Java. The Green project introduced in 1990's, this language initially called "Oak" Language, but was renamed as "JAVA" in 1995.

→ The primary motivation of this project is platform independent. Java is platform independent.

* Platform Independent: Write once, run anywhere

Ex: C, C++ are languages are platform dependent. C & C++ can written in windows and do not execute in Linux. But Java can written in windows & can execute in any operating system. (JVM is required).

→ On Jan 23rd, 1996, ~~JDK~~ jdk1.0 ~~version~~ was released without classes.

→ On In the year 1997, jdk1.1 was released with classes with API's (Application Programming interface).

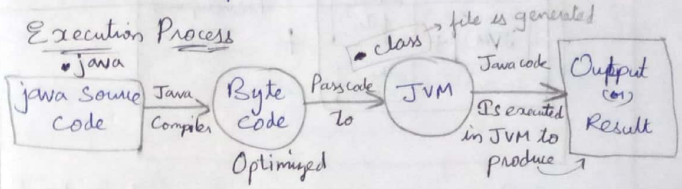
→ In 1998, J2SE versions was released. The main aim of J2SE version is swings concept, graphical API's, collection frame works (Light weighted components) (Applet package) (Util package, IO package, etc)

→ Advanced JVM is introduced in j2SE1.3 versions.

* j2SE stands for Java 2 Standard edition

→ Java has compiler & Interpreter.

* Execution Process




→ Computer can only execute the code in binary form.

→ Computer doesnot understand high level language

* Compiler

It translates the entire source code into machine code (or) any intermediate code.

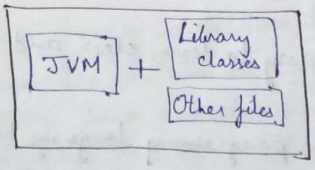
* Interpreter

It reads one statement from the source code & ~~transmitted to translate~~ it into machine code using JVM.

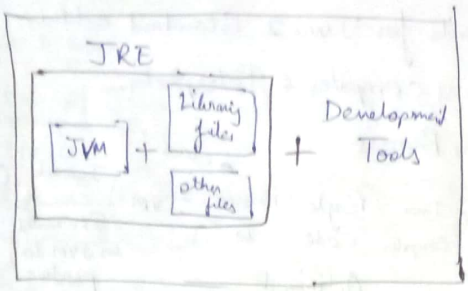
* JDK, JVM, JRE

• JVM: Java Virtual Machine.

• JRE: Java Runtime Environment



JRDK - Java Development Kit




→ JDK provides the runtime environment & development tools for checking the source code & developing the source code.

→ JRE provides the only runtime environment.

20-7-19

(2) Features of JAVA (or) JAVA BUZZ Words

• Java Buzz words (Java def)

(i) Java is Simple
(ii) Java is object oriented
(iii) Platform independent
(iv) Secured
(v) Robust (Strong) ← cannot destroy
(vi) Architectual Neutral
(vii) Portable
(viii) High Performance
(ix) Distributed
(x) Multitalented
(xi) Compiled & Interpreted

(i) Simple

Java is very easy to learn, clear and easy to understand.

→ Java is simple programming language because, JAVA syntax is based on C++ programming.

→ Java has removed many features (or) complexed features.
Ex: External operator concept & Operator overloading concept.

→ Java has a garbage collector.

→ There is no need to remove or free the data because java has automatic garbage collector.

(ii) Object Oriented

Everything in Java is an object. Object oriented programming methodology simplifies the software development and maintainance.

→ OOP concepts are
Ex Class, Object, Data Abstraction, Data encapsulation
Ex Inheritance, Polymorphism.

(iii) Platform Independent

• Platforms

Platform = Operating System + microprocessor.

→ Java is ~~Architecture~~ write once, run anywhere language.

→ Ex JAVA code can run on multiple programs platforms
Ex Windows, Linux, Mac/OS etc.

→ Java has 2 components
(a) Runtime environment (b) API (Application Programming Interface)
Ex JRE

(iv) Secured

→ Java is best known for its security

→ Java achieves the protection by confining a JAVA program to the Java execution environment & not allowing to other parts of the computer.
Ex:- If java program is executed in JVM, it does not (Run time environment) allow other alien components / other programs.

(V) Robust

It means strong.

→ Java is robust because it uses strong memory management (or) it has exception handling & type checking mechanisms.

• Exception

It is runtime error.

Ex: a=10, b=10, c=a/b=2

→ if a=1, b=0, c=1/0=∞ (program will collapse).

(VI) Architectual Neutral

→ In java, they are no implementation dependent.

Ex- in C programming, int datatype occupies 2 bytes for 32 bits system & 4 bytes of memory for 64 bits system.

But, in Java 4 bytes of memory for both (32 bit system & 64 bit systems.

(VII) Portable

→ Java is portable because it facilitates (make easy) to carry the java byte code to any platform.

→ Java program will run virtually & any computer connected to internet.

(VIII) High Performance

→ Java is high performance.

→ Java is faster than other traditional languages

Ex- Java byte code is close to the native code. (platform understandable code which is provided by JVM)

• Native code

It is only understand by platform.

(IX) Distributed

Java is distributed because it facilitates the users to create distributed applications in java.

→ The distributed application is possible in RMI (Remote Method Invocation).

(X) Multi threaded

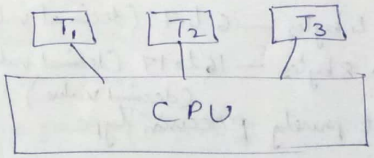
• Thread

It is a tiny/small program.

→ More than one threads executed in single CPU parallely is called Multi Threaded.

→ The main advantage of multi threaded is it does not occupy seperate memory in each thread, it shares the common memory.

Ex- Thread $T_1$, $T_2$, $T_3$ are executed in CPU




(XI) Compile & Interpreter

→ Java combined both compiler & interpreter

→ Java source code into byte code through java compiler. The byte code is interpreted which generates machine code that can be executed by JVM. Therefore java is compiled & interpreter

2/7/19

## Datatypes in Java

- Datatype
  It is specific about size & value of a variable.
→ Datatype is declaring the variables.
→ Java is strictly type checking language.
  Ex: int a = 4.5 ✗ not allowed

- Primitive Datatypes
  These datatypes consist of simple values. Types predefined
  Types of value predefined by java.

(i) Integer category
  (i) Byte → 1 byte = 8 bits
  (ii) Short → 2 bytes = 16 bits
  (iii) int → 4 bytes = 32 bits
  (iv) Long → 8 bytes = 64 bits

(ii) Floating category
  (i) float → 4 bytes — 6 to 7 (decimal values)
  (ii) double → 8 bytes — 15 to 17 (decimal values)
  (decimal value)

- Float is not purely precision type.
  eg float a = 4.5; ✗
  float a = 4.5f; or 4.5F; ✓

- double a = 4.55; ✓
  double a = 4.55D; ✓
  double a = 4.55d; ✓

(iii) Char type
  char → 2 bytes = 16 bits
  char a = 'A'.

(iv) Boolean type
  It can check true / false.
→ By default, boo value of boolean is "false"

- Example program

```
boolean flag;
boolean a = true;
if (a == flag)
  print TRUE;
else
  print FALSE;
```

→ Program

```
class Boolean
{ p.s.v.m(String args[])
  {boolean flag;
   boolean a=true;
   if (a==flag)
    S.o.println("TRUE");
   else
    S.o.println("FALSE");
  }
}
```

- Implicit conversion type casting

To convert one datatype to another datatype without any force internally.

Eg:- short a;
byte b = 10;
a = b;

- Explicit typecasting

To convert one type to another type externally (or) explicitly (or) forcefully.

Ex:- when you are assigning a large value to a smaller variable value type, then you need to perform explicit typecasting.

Ex (i) ~~int~~ int a
long b = 10;
a = (~~long~~ int) b;

- Syntax: variable 1 = (type) variable 2

24/7/19

## Variables in java:

Imp

In java, there is no global variable concept.

### Variable

These are nothing but reserved memory locations to store values.

→ Java has 3 variables

(i) Local variable
(ii) Instance variable
(iii) Static variable.

(i) LOCAL VARIABLE

Declared inside the class and inside the main method is called Local Variable

→ It can be accessed without an object.

(ii) INSTANCE VARIABLE

Declare the variable inside the class & outside of the main method is called Instance variable.

→ It can be accessed through object.

(iii) STATIC VARIABLE

Static variable pre-declaration with static keyword.
It can be accessed without object (or) with class.

→ The static block will be executed first.

```
class classname
{
    int b=20; // instance variable
    static int c=30; // Static variable

    public static void main(String args[])
    {
        int a=10; // local variable

    }
}
```

Instance area | static { } executed first.

Static area

+ WAJP to demonstrate JAVA variables.

```
class varjava
{ int b=20;   // Instance variable
  static int c=30; // Static variable
  public static void main(String args[])
  {
    int a=10 // local variable
    System.out.println("local variable a="+a);
    varjava obj=new varjava();
    System.out.println("Instance var b="+obj.b);
    System.out.println("Static variable c="+c);
    System.out.println("Static variable with class c="
                       +varjava.c);
  }
}
```

without obj — with obj

## Arrays in java

- **Arrays**

It is a collection of homogeneous data elements.

→ Arrays is a collection of a fixed number of data elements.

- **Advantages**

→ These are fixed in sized

→ It is not allowed heterogeneous data elements.

→ Large number of values ~~string~~ can be stored in a single variable

→ In advance, we know the size of elements.

- **Syntax of array**

datatype variablename[] = {values};

datatype variablename[] = new datatype[size];

Eg:- int val[] = new int[3]; → allocate memory at runtime (dynamic memory allocation)

int[] val = new int[3];

- WAJP to demonstrate 3 integer values using arrays

int a[] = {10,20,30};

Eg:- class Arrayd

```
{p.s.v.m(String args[])
    {int a[]={10,20,30};
     S.o.println(a[0]);
     S.o.println(a[1]);
     S.o.println(a[2]);
    }
}
```

26/7/19

→ By default package is "lang" package, it is available in JAVA (or) JVM.

→ Arrays memory allocation is dynamically (or) allocated at runtime.

eg: int a[] = ~~int~~ new int[5].

1) WAJP to demonstrate arrays.

2) WAJP to read 'n' numbers & display 'n' numbers using arrays.

3) WAJP to find sum of 'n' numbers using arrays.

1)
```
import java.util.*;
class Arr1
{ public static void main (String args[])
  { int a[] = new int[4];
    int i;
    a[0] = 10;
    a[1] = 20;
    a[2] = 30;
    a[3] = 40;
    System.out.println("Array elements are");
    for(i=0; i<4; i++)
    { System.out.println(a[i]);
    }
    for(i=0; i<a.length; i++)   // Keyword
    { System.out.println(a[i]);
    }
  }
}
```

2)
```
import java.util.*;
class Arr_Read
{ public static void main (String args[])
  { int a[] = new int[10];
    int i,n;
    Scanner obj=new Scanner(System.in);
    System.out.println("Enter n");
      n = obj.nextInt(); // n = Integer.parseInt(obj.nextLine);
    for(i=0;i<n;i++)
    { a[i]=obj.nextInt();
    }
    System.out.println("The Array element are");
    for(i=0;i<n;i++)
    { System.out.println(a[i]);
    }
  }
}
```

3)
```
import java.util.*;
```

3)
```
import java.util.*;
class Arr_Sum
{ public static void main (String args[])
  { int i,n,sum=0;
    Scanner obj = new Scanner(System.in);
    System.out.println("Read n");
    n = obj.nextInt();  //reading
    int a[] = new int[n]; //creating memory
    System.out.println("Enter array elements");
    for(i=0;i<n;i++)
    { a[i] = obj.nextInt();
    }
    for(i=0;i<n;i++)
    { sum = sum + a[i];
    }
    System.out.println("Sum of n=" +sum);
  } //main
} //class
```

int a[10]

## * 2D - Array

- Syntax-1

  datatype variablename[][] = {values};

  datatype [][] variablename = {-----};

- Syntax-2

  datatype[][] var1 = new datatype [rows size][col size];

Ex: int[][] a = new int[2][2];

| a[0][0] | a[0][1] |
|---|---|
| 1 | 0 |
| a[1][0] | a[1][1] |
| 0 | 1 |

⟺

| 1001 | 1005 |
|---|---|
| 1 | 0 |
| 1009 | 1013 |
| 0 | 1 |

27/7/19

Q: WAJP to read and display a matrix (m x n) (m = rows, n = columns)

```
import java.util.*;
class Matrix
{ public static void main(String args[])
  { int m,n,i,j;
    Scanner S = new Scanner(System.in);
    System.out.println("Enter m,n");
    m = S.nextInt();
    n = S.nextInt();
    int a[][] = new int[m][n];
    System.out.println("Read m*n matrix");
    for(i=0;i<m;i++)
    { for(j=0;j<n;j++)
      { a[i][j] = S.nextInt();
      }
    }
```

Reading

```
System.out.println("Display matrix");
for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  {
    System.out.println(a[i][j]+"\t");
  }
  System.out.println();
}
}
```

| 1001 | 1005 |
|---|---|
| a[0][0] | a[0][1] |
| 2 | 3 |
| 1009 | 1013 |
| a[1][0] | a[1][1] |
| 4 | 5 |

→ Command line Argument : pass the argument at run time.
Ex: args[0]; args[1] ... args[n-1]

→ ParseInt method converts string value into integer. This ParseInt method is available in Integer class.

- Method

Method initial letter starting with lower case, after completion of one word, second word starting letter must be in upper case (capital). Ex: parseInt.

The class initial letter should be capital, after completion of 1st word, second word initial letter is also capital.
Ex: DataInputStream, Integer.

- Scanner [import java.util.*;]

It is a class provided by "util" package

→ 'nextLine' is a method to read a line. It is available in Scanner class.

→ throws IOException → It is present in util package but not in io package
[import java.io.*;]

* For each → present in util & lang package.

. Syntax | for (initialisation; condition; inc/dec) → For loop syntax

for(datatype variablename : Arrayname)

→ For each faster than for. It is new version of arrays. Each element of array will be passed to for each (or) For each variable retrive array cell one after the another.

Ex:- for(int b:a)

String names[] = {"Lilly","Laila"
"Riya","Maya"};

for(String name:names)

10: WAJP to print 4 values using For-each.

2) WAJP to print names of a string using For-each

1)
```
import java.io.*;
class foreach1
{public static void main(String args[]) throws IOException
 { int a[]={5,10,15,20};
   for(int b:a)
    {System.out.println(b);
    }
 }
}
```

2)
```
import java.io.*;
class Foreachnames
{ public static void main (String args[]) throws IOException
  {
    String names[]= {"Lalla","Lassi","Lilly"};
      for(String name: names)
        {
          System.out.println(name);
        }
  }
}
```

6) WAJP to demonstrate switch case using go-to, continue and break keywords.

```
class for1
{ p.s.v m(String SI])
  {int a[]={10,20,30,40,50,60,70,80,90,100};
    for(int i=0; i<10; i++)
      {if (i==5)
        {
          break; [continue]
        }
        S.o.println(i);
      }
  }
}
```

break

```
0
1
2
3
4
```

continue

```
0
1
2
3
4
6
7
8
9
```

```
class for2
{ p s v m (String S[])
  {int a[]={10,20,30,40,50,60,70,80,90,100};
    for (int i=0; i<10; i++)
      {if (i%2==0)
        {
          break; (continue);
        }
        System.out.println(i);
      }
  }
}
```

break

continue

```
1
3
5
7
9
```

- Static — can execute programs with objects & without object methods.
  - Information stored in heap formate (memory)
  - It allows static, local & instance variables (for this obj is needed)
- IO → throws IOException
- Scanner → util package
- main - It is given by James Gousling

31/7/19

# OPERATORS

Operator is a symbol that perform an operation.

Ex: c = a + b

operator, operand

a, b, c = operands

=, + = operator.

→ We have 3 different types of operators ie

(i) Unary operator

(ii) Binary operator

(iii) Ternary operator

(i) Unary operator

If an operator acts on single variable. it is called Unary operator.

Ex: -x, x++, --x

(ii) Binary operator

If it acts on 2 variables, then it called Binary operator

Ex: a+b

(iii) Ternary operator

If it acts on 3 variables, then it is called Ternary operator.

Ex: Conditional operator: ? :

## I Arithematic operator

This operators are used to perform fundamentals of arithematic operations like addition (+), subtraction (-), multiplication (*), and division (/) and modulus (%).

Ex: c = a+b, c = a/b.

→ Addition operator (+) is also used to join two strings.

Ex① Str1 = "Yugala";
Str2 = "Sree";
Str3 = Str1 + Str2
o/p - Str3 = Yugala Sree

② a = 10, b = 20
S.O.println("a="+a+"b="+b);
o/p: a=10 b=20

## II Unary Operator

As the name indicates unary operators, they act only one operand.

→ (a) Unary minus (-)
(b) Unary Increment (++)
(c) Unary Decrement (--)

(a) Unary minus (-) operator

This operator is used to negate a given value.

Negate (or) negation means converting -ve to value to +ve vice versa.

Ex: int x = 5
System.out.println(-x); // output : -5
System.out.println(-(-x)) // output = 5

(b) Unary Increment (++) operator

This operator increase the value of a variable by 1

Ex: x variable value increamented by 1, when we use ++ operator.

(i) Pre increment operator

Writing ++ before a variable is called Pre-increment operator.

Ex: ++x

(ii) Post increment operator

Writing ++ after a variable

Ex: x++

- Example for pre & post increment operator

```
x=1
System.out.println(x)   // o/p → x=1
System.out.println(++x) // o/p → x=2
System.out.println(x)   // o/p → x=2

x=1
S.o.println(x);   // o/p → x=1
S.o.println(x++); // o/p → x=1
S.o.println(x);   // o/p → x=2
```

- ++a * a++ a=7

  8 * 8 = 64

Precedence: ++ > *

(c) Unary decrement (--) operator

(i) Pre decrement operator

→ writing -- before a variable.

Ex: --x

(ii) Post decrement operator

Writing -- after a variable

Ex: x++

- Example for pre & post decrement operator

```
x=1
S.o.println(x)   // x=1
S.o.println(--x) // x=0
S.o.println(x)   // x=0
```

```
x=1
S.o.println(x)   // x=1
S.o.println(x--) // x=1
S.o.println(x)   // x=0
```

- --a * a-- a=7

  6 * 6 = 36

* NOTE:

→ Pre-decrementation means first decrement the value then perform the operation.

→ Post-decrementation means first perform the operation then decrement the value.

→ Pre-increment means first increment the value then perform the operation.

→ Post-increment means first perform the operation then increment the value.

III Arithmetic Assignment operators

+=, -=, *=, /=

Ex: a = a+b; this can be written as

→ a+=b

Ex: c = c*b can be written as c*=b

IV Relational Operators

These operators are used to compare 2 variables

Eg: >=, <=, >, <, !=, ==

V Assignment operators

Right side variable is assigning to left side variable

Ex: a=b , a+b=c X not accepted.

1) WAJP to demonstrate Arithematic operators.

```
import java.util.*;
class Arth
{ public static void main(String[] args)
  { int a,b,c;
    Scanner s=new Scanner(System.in);
    a=s.nextInt();
    b=s.nextInt();
    c=a+b;
    System.out.println("c="+c);
  }
}
```

O/P
```
7
3
c=10
```

2) WAJP to join 2 Strings

```
class String1
{ public static void main(String[] args)
  { String s1="Yugala";
    String s2="Sree";
    String s3=s1+s2;
    System.out.println(s3);
  }
}
```

O/P YugalaSree

3) WAJP to demonstrate unary minus, increment & decrement operators.

```
class unary1
{ public static void main(String[] args)
  { int x=5;
    System.out.println("Unary minus");
    S.O.println(-x);
    S.O.println(-(-x));
    S.O.println("Post Increment Operator");
    S.O.println(x);
    S.O.println(x++);
    S.O.println(x);
    S.O.println("Pre Increment Operator");
    S.O.println(-x);
    S.O.println(++x);
    S.O.println(x);
    int y=++x*x++;
    S.O.println(y);
    S.O.println("Post Decrement Operator");
    S.O.println(x);
    S.O.println(x--);
    S.O.println(x);
    S.O.println("Pre Decrement Operator");
    S.O.println(x);
    S.O.println(--x);
    S.O.println(x);
    int z=--x*x--;
    S.O.println(z);
  }
}
```

o/p
Unary minus
-5
5
Post Increment Operator
5
5
6
Pre Increment Operator
6
7
7
y= 64
Post Decrement Operator
9
9
8
Pre Decrement Operator
8
7
7
z=36

4) WAJP to demonstrate boolean operators.

```
Class Bool1
{ public static void main (String[] args)
  { System.out.println ("Boolean Operators");
    boolean a=true;
    boolean b=false;
    System.out.println (a&b);
    S.O.println(a|b);
    S.O.println(a&a);
    S.O.println(b|b);
    S.O.println(!a);
    S.O.println(!b);
  }
}
```

o/p
Boolean Operators
false
true
true
false
false
true

5) WAJP to demonstrate bitwise operators.

```
Class Bit1
{ public static void main (String[] args)
  { S.O.println ("Bitwise operators");
    S.O.println("Boolean values");
    S.O.println(true & false);
    S.O.println(true | false);
    S.O.println(true ^ false);
    S.O.println("Integral values");
    S.O.println(4 & 5);
    S.O.println(4|5);
    S.O.println(4^5);
    S.O.println("Bitwise Compliment operators");
    S.O.println(~4);
  }
}
```

o/p
Bitwise Operators
Boolean Values
false
true
true
Integral Values
4
5
1
Bitwise Compliment Operators
-5

6) WAJP to demonstrate condition operator (or) ternary operator.

```
Class cond1
{ public static void main (String[] args)
  { System.out.println ("Conditional Operators");
    int x = (10<20)? 30:40;
    S.O.println (x);
  }
}
```

o/p
Conditional Operators
30

NOTE

Operators Precedence order (Priority order)

1) ( ), [ ] — declaration of arrays (method / function)
2) ++, -- (increment & decrement operators)
3) *, /, % (multiplication, division, modulus)
4) +, - (addition & subtraction)
5) >=, <=, ==, >, <, !=, == (relational operators)
6) &, |, ! (Boolean operators) — and, or, not
7) &, |, ^ (EXOR), (Bitwise operators), ~, <<, >>
8) Logical and (&&), Logical or (||), Logical not (!) (Logical operators)
9) Conditional operator (? :) [Ternary operator]
10) Assign = (Assignment operator)

* Capacities

byte + byte = int (integer)
byte + short = int
short + short = int
byte + long = long
int + int = int
long + double = double
float + long = float
char + char = int
char + double = double

21/8/19

## Logical Operators

These operators are used to construct compound conditions. A compound condition is a combination of several simple conditions. (&&, ||)

Ex: if ((a>b) && (a>c))
simple condn, simple condn

if ((a<b) || (b<c))

## Bitwise Operator

It works on boolean values & Integer values.

Ex: &, ^, ~, |

Ex: ~~System.out~~ These operators acts on individual bits ('0' and '1') of the operands.

(i) And operator (&)

It returns true, iff both conditions are true

```
System.out.println(4&5);
```

```
        2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
4  ←     0   0   0   0   0   1   0   0
5  ←     0   0   0   0   0   1   0   1
-------------------------------------
         0   0   0   0   0   1   0   0  → 4
```

(ii) OR operator (|)

System.out.println(4|5) It returns true, iff at least one condition is true

```
  0 0 0 0 0 1 0 0
  0 0 0 0 0 1 0 1
-----------------
  0 0 0 0 0 1 0 1
```

(iii) Ex-OR (^)

System.out.println(4^5)

```
  0 0 0 0 0 1 0 0
  0 0 0 0 0 1 0 1
-----------------
  0 0 0 0 0 0 0 1  → 1
```

T ^ T = F
F ^ F = F
T ^ F = T
T ^ F = T

It returns true iff both arguments arguments are different.

(negation)

(IV) Bitwise Complement operator (~)

System.out.println(~4)

4 = 0|0000100 → [OR] = ~(4+1)

~4 = 1|1111011 = -5

negative

00000100 → 2's complement

1

00000101

→ Negation operation (-5) is applicable for only for integer not boolean

(V) Bitwise Left shift operator (<<)

Ex: int x=4;

System.out.println(x<<2);

$2^7$ $2^6$ $2^5$ $2^4$ $2^3$ $2^2$ $2^1$ $2^0$

0 0 0 0 0 1 0 0 = 4

→ 0 0 0 1 0 0 0 0 (add 0s) ⇒ $2^4$ = 16

This operator shifts the bits of the number towards left, a specified number of positions.

→ If we write x<<n, the meaning is to shift the bits of 'x' towards left by 'n' positions.

(VI) Bitwise Right shift operator (>>)

System.out.println(4>>2)

x = 4

0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 1 = 1

(add 0s)

3/8/19

## Boolean Operators

This operator acts on boolean variables & produce the boolean results.

→ & (and), ! (not), | (or)

boolean a = true; b.

boolean b = false;

System.out.println(a&b); false

S.O.println(a&a); true

" " " (a|b); true

" " " (!a); false

" " " (!b); true

" " " (b|b); false

## Conditional Operators

The only possible ternary operator in java is conditional operator.

Ex: var = (condition)? value (or) variable : value (or) variable

Ex: int x = (10>20)? 30 : ((40<50)? 60 : 70)

S.O.println(x)

o/p: 60.

int x = (10<20)? 30 : 40

S.O.println(x)

o/p = 30.

## New Operator

Create a new object. Using new operator, we can allocate memory to the class and create the object.

Syntax

```
classname obj = new classname();
                    ↑ constructor
        (or)
classname obj = new constructor;
```

## Deallocation

Garbage collector is responsible for to free the data.

→ Destroying the useless objects responsibility is for garbage collector.

## Square Brakets Operator ([])

We used it to declare & create arrays.

Ex: `int a[] = new int[10];`

`int [] a = new int[10];`

## * Control Statements

1) If
2) if else / Nested if
3) while
4) do-while
5) for
6) foreach
7) Switch
8) Transfer Statements (break, continue)

## * CONTROL STATEMENTS

- Switch Case

When there are several statements / operations / options and we have to choose only one option from the available source.

Ex: WAJP to print colours.

```
import static java.util.*;
class Color1
{ public static void main(String a[])
  { int c;
    Scanner obj = new Scanner (System.in);
      c = obj.nextInt();
      Switch (c)
      { case 1 : S.o.p ("Red"); break;
        case 2 : S.o.p ("Green"); break;
        case 3 : S.o.p ("Yellow"); break;
        default : S.o.p ("No colour");
      }
  }
}
```

- Break Statement

• Break Statement

We can use break statement in the following places.

(i) Inside loop
(ii) Inside condition
(iii) Inside switch
(iv) Inside labelled block

→ Break can terminate the block, not program.

Ex1: if (a==b)
    break;

Ex2: for (i=0; i<5; i++)
    { if (i==4)
        { break;
        }
      S.o.p (i);
    }

o/p: 0
1
2
3

• Label:

```
class Demo1
{ p.s v m (String s[])
  { boolean x=true;
    b1:{
    b2:{
    b3:{
        S.o.println ("Block 3");
        if(x) .break b2;
       }//b3
       S.o.println ("Block 2");
      }//b2
     S.o.println ("Block 1");
    }//b1
  }//main
}//c
```

Output
Block 3
Block 1

• Continue statement

```
class Continue1
{ p.s v main (String args[])
  { int a[] = {10,20,30,40,50,60,78,80,90,100};
    int i;
    for(i=0; i<10; i++)
    { if (i<5)
        continue;
      S.o.println(a[i]);
    }
  }
}
```

output
60
70
80
90
100

→ Java Continue is used inside a loop to repeat the next iteration of the loop.

→ Inside switch, the continue statement not allowed.

• Java continue = It continues, the same loop.

• C continue = It continues next statement

5/8/19 (1) INTRODUCTION OF CLASSES | Unit 2

(Classes, objects, methods, object reference variables)

* Classes

It is a model for creating objects.

→ This means, the properties and actions of the objects are written within the class.

* Properties

These are variables (or) properties are represented by variables.

+ Method

Actions of the objects are represented by methods. (or)

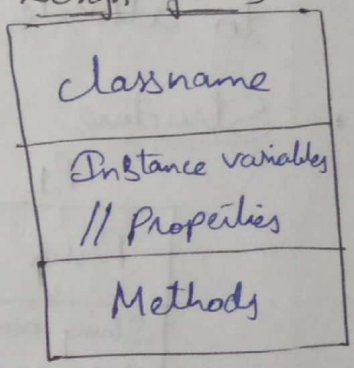
→ Methods performs specific tasks & it is called by object.

• Structure of Class

(Syntax)

```
class classname
{ properties // Instance variables
        &
  Methods // Actions of objects
}
```

• Design of class




Ex Example of class

```
class Person
{
  String name;
  int age;        } Properties
  void Showperson() // Methods
  {
    S.o.println("Name = " + name + "Age = " + age);
  }
}
```

## Objects

Objects are creation of object.

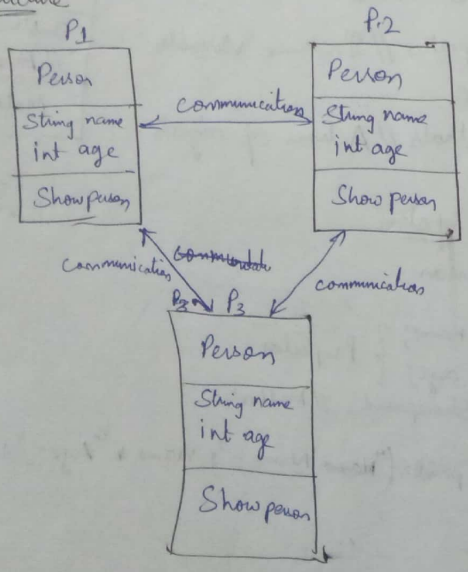
→ Object is a instantiation of class.

→ To create an object, following syntax is used.

Ex:
```
class Mclass
{
  public static void main(String args[])
  {
    //classname objname = new classname();
    eg: Person P1 = new person();
        Person P2 = new person();
  }
}
```

→ The "new" keyword operator is used to create an object ie to allocate memory to object class.

• Structure




Q. WAJP to create a person class and an object P1 to person class.

```
class Person                          public
{                                     access from anywhere
  String name;
  int age;
  public void showPerson()  → 1 method
  {
    System.out.println("name = " + name + "\n age = " + age);
  }
}
```

• By using main

```
class Mclass1                  mclass1
{ public static void main(String s[])
  { Person P1 = new Person();
    P1.name = "Prasen";
    P1.age = 15;
    P1.showPerson();   ← object   → method
  }
}
```

→ Save: Mclass1.java

→ Compile: javac Mclass1.java

→ Run: java Mclass1

→ Output

name = Prasen
age = 15

NOTE
* We cannot write java program without main class.
For writing java program atleast 1 class.
** → Without class, we cannot access java program.
But we can access in c++.

7/8/18

Q- Write a java program to demonstrate person class with parameterised method

```
Class Person
{ String name;
  int age;
  void getperson(String n1, int a1)
   { name = n1;
     age = a1;
   }
   void showperson()
   { S.o.println("Name = "+name+" Age = "+age);
   }
}
class Mclass3
{ public static void main(String args[])
  { Person P1 = new Person();
    P1.getperson("Pooja", 18);
    P1.showperson();
  }
}
```

- Save Mclass3.java
- Compile: javac Mclass3.java
- Run: java Mclass3
- Output

  Name = Pooja
  Age = 18

1) WAJP to perform multiple objects.
2) WAJP to perform area of a rectangle.

2) II

```
Class Rect
{ int l;
  int b;
  Void getRect(int x, int y)
   { l = x;
     b = y;
   }
  void showRect()
   { int ar = l*b;
     S.o.println("Area of Rectangle = "+ar);
   }
}
class mclass4
{ p.s.v.m(String args[])
  { Rect r1 = new Rect();
    r1.getRect(4,5);
    r1.showRect();
  }
}
```

- Save: mclass4.java
- Compile: javac mclass4.java
- Run: java mclass4
- Output - Area of Rectangle = 20

1) class Person

```
{ String name;
  int age;
  void showPerson();
  {
    S.O.println("Name="+name);
    S.O.println("Age="+age);
  }
}
class mainperson
{ P S V main(String S[])
  {
    Person P1=new Person();
      P1.name="Naveen"
      P1.age="25"
      P1 showPerson();
    Person P2=new Person();
      P2.name="Karthik"
      P2.age="18"
      P2 showPerson();
    Person P3=new Person();
      P3.name="Rithwik"
      P3.age="20"
      P3 showPerson();
  }
}
```

Save Mainpersonjava

Output

Name = Naveen
Age = 25
Name = Karthik
Age = 18
Name = Rithwik
Age = 20

2) Object Reference WAJP to perform +, -, *, / using parametrised methods

```
class Arith
{
  void sum(int a, int b)
  {
    int s;
    s=a+b;
    S.O.println("Sum="+s);
  }
  void sub(int a, int b)
  {
    int s;
    s=a-b;
    System.out.println("Sub="+s);
  }
  void mul(int a, int b)
  {
    int s;
    s=a*b;
    S.O.println("Multiplication="+s);
  }
  void div(int a, int b)
  {
    int s;
    s=a/b;
    S.O.println("Division="+s);
  }
```

* Object Reference Variable

```
class Operations
{  p.s.v.m (String[] args)
   {  Arth x = new Arth();
      x.sum(1,2);
      x.sub(5,3);
      x.multi(10,5);
      x.div(12,4);
   }
}
```

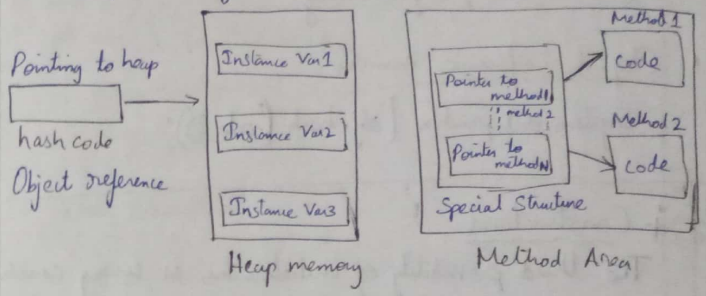
Save: Operations.java

Output

Sum = 3
Subtraction = 2
Multiplication = 50
Division = 3

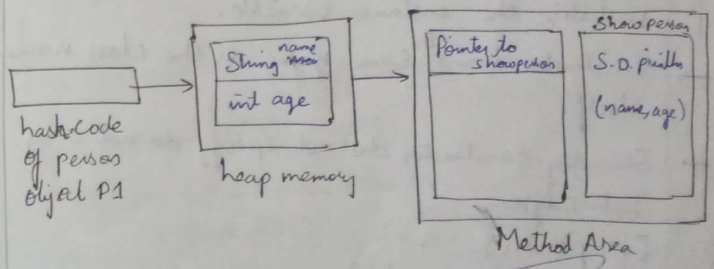
* Object Reference Variable

It contains address of the object which is declared in heap memory.




Ex:

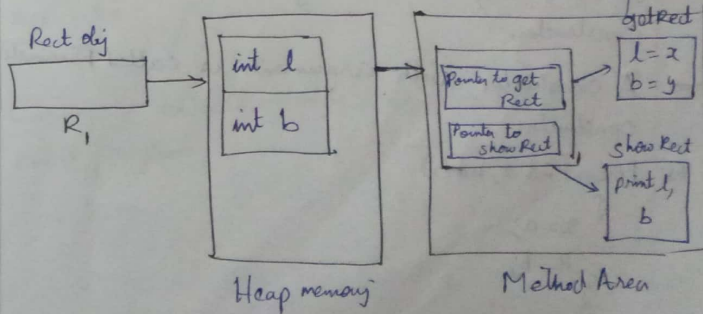
I




* System.out.println (obj.hashCode());
  P1.hashcode();
  S1.hash Code();
  R1.hash Code();

8/8/19

II



* NOTE

Heap memory is portion of memory, it is allocated at run time / dynamically.

- Object Reference number

System.out.println (obj.hashCode());

29/8/19 Rakesh Bday

* Constructors

The third possibility of initialisation is using constructor.

→ A constructor is similar to a method that is used to initialise the instance variable.

→ Constructor have the same name as the class name itself.

→ Secondly constructor do not specify ~~do not~~ any return type.

Eg: person() ✓
person(int a, int b) ✓
void person() ×

- Default Constructor

A constructor without arguments is called Default constructor.

→ A constructor with arguments is called Parameterised constructor.

Ex: Person (int a, int b)
{
x = a;
z = b;
}

Q: WAJP to demonstrate class Person constructor.

Method III

```
class Person
{ String name;
  int age;
  Person() // class name
  { name = "John";
    age = 30;
  }
  void ShowP1()
  {
    S.o.println("Name = " + name);
    S.o.println("In Age = " + age);
  }
}
class Conmain1
{ p s v main (String args[])
  { Person P1 = new Person();
    P1.ShowP1();
  }
}
```

Same:- Conmain.java

Output

Name = John
Age = 30

→ Constructor is called only once, per object for the entire program.

→ A constructor is automatically called & executed at the time of creating an object

→ A constructor is called & executed only once per object. This means, when we create an object, the constructor is called.

→ If you want to call second constructor (or) second time constructor we should create same class name with different objects

Q: WAJP to perform addition of two numbers using Constructors

```
class Addition
{ int a,b;
  Addition(int x, int y)
    { a=x;
      b=y;
    }
  Void ShowAdd()
    { int c;
      c=a+b;
      S.o.println("Add="+c);
    }
}
class Addcon
{ p s v main(String args[])
  { Addition a1=new Addition(10,20);
    a1.ShowAdd();
  }
}
```

Output
Add = 30

Q: WAJP to perform addition of 2 numbers using Scanner class & return value

```
import java.util.*;
class Addition
{ int a,b;
  Addition(int x, int y)
    { a=x;
      b=y;
    }
  int Showadd()
    { int c;
      c=a+b;
      return(c);
    }
}
class Addcon
{ p s v main(String args[])
  { Scanner s1=new Scanner(System.in);
    int x=s1.nextInt(); //10
    int y=s1.nextInt(); //20
Addition a1=new Addition(x,y);
    int z=a1.Showadd();
    System.out.println("Z="+z);
  }
}
```

Output
4
6
Z=10

10. WAJP to perform area of circle & rectangle using constructors & methods.

```
import java.util.*;
class Rec_area
{
  int len, bre;
  Rec_area(int a, int b)
    { len = a;
      bre = b;
    }
  void area_Rec()
    {
      int area;
      area = len * bre;
      S.o.println("Area of rectangle = "+area);
    }
}
class Circle_area
{ double rad;
  Circle_area (int r)
    { rad = r;
    }
  void area_Circle()
    { double area;
      area = Math.PI * rad * rad;
      S.o.println("Area of circle = "+area);
    }
}
```

```
class Area_Con1
{ p s v main(String args[])
  {
    Scanner obj = new Scanner(System.in);
    S.o.println("Enter length breadth of rectangle=");
    int a = obj.nextInt();
    int b = obj.nextInt();
    Rec_area r1 = new Rect_area(a,b);
    S.o.println("Enter radius of circle");
    int r = obj.nextInt();
    Circle_area c1 = new Circle_area(r);
    r1.area_Rec();
    c1.area_Circle();
  }
}
```

Difference b/w constructors & methods

| Constructor | Method |
|---|---|
| 1) It is used to initialise the variables | 1) It is used to initialise the variables & perform the operations & calculations |
| 2) Constructor name & class name both are same | 2) Method name & class name both are different |
| 3) Constructor is called when memory is allocated to the object | 3) Method is called after creating the object. |
| 4) Only once per object | 4) Multiple methods for single object. |
| 5) Constructor called automatically | 5) Method is called (executed) with object. |

10/8/19

## Method Overloading

Writing two or more methods in the same class, such a way that each method has same name, but method signatures are different is called Method Overloading.

## Method Signature

It represents method name with parameters.

Ex:- One/more methods have same name but different signatures.

```
void add();
void add(int a, int b);
void add(double a, double b);
```

- Syntax

```
class classname()
{
  // Data members
  Same Methodname();
  Same Method name(int);
  Same Method name(int a, int b, int c);
  Same Method name(double a, double b);
}
```

- NOTE

→ JVM should decide which method is actually called by the user at run time.

Q: WAJP to demonstrate method Overloading.

```
class Add1
{ void add(int a, int b)
  { int c=a+b;
    System.out.println("Add="+c);
  }
  void add(double x, double y)
  { double z=x+y;
    System.out.println("Add of double="+z);
  }
}
class Addmain
{ public static void main(String s[])
  { Add1 a1 = new Add1(); // constructor
    a1.add(5,7);
    a1.add(5.77, 6.77);
  }
}
```

Output

Add = 15

Add of double = 12.1

* Constructor Overloading

More than 1 constructors are repeated within a class is called Constructor Overloading.

- Syntax

```
class Classname
 { classname()
   {
   }
   classname (datatype parameterised);
    {
    }
 }
```

Ex:

```
class Bank
 { String Cname;
   int accno;
     Bank()
     {
     }
     Bank(name1, acc1)
     {
     }
 }
```

Q: WAJP to demonstrate Constructor overloading.

```
class Bank
{ String name;
  int acc_no;
  Bank()
  { name = "John";
    acc_no = 5662;
  }
  Bank (String n1, int a1)
  { name = n1;
    acc_no = a1;
  }
  void ShowBank ()
  { System.out.println ("Name = "+name);
    S.o.println ("\nAcc no="+ acc_no);
  }
}
class Bvscon
 { public static void main (String args[])
   { Bank b1 = new Bank();
     b1.ShowBank();
     Bank b2 = new Bank ("RAM", 4555);
     b2.ShowBank();
   }
}
```

- Output

```
Name = john
Acc no = 5662
Name = RAM
Acc no = 4555.
```

## Static Members

In some situations, we want to have common instance variables & methods for all objects. This can be achieved by declaring the instance variables, instance methods of a class as static.

→ Instance variable & instance methods accessed with objects (. operator). Let us assume that, we want to define a member that is common to all objects & accessed without particular object. The static members are

(i) Static variable (ii) Static methods.

```
class classname
{
  static int a;          // Static variable
  static void push();    // Static method.
}
```

→ Static variables & Static methods accessed through class name.

1) WAJP to demonstrate Static variables.

2) WAJP to perform multiplication & division using Static methods.

2)
```
class Staticmethod
{
  static float mul(float a, float b)
  {
    return a*b;
    // float c = a*b;
    // return (c);
  }
  static float div(float a, float b)
  {
    float c = a/b;
    return (c);
  }
}
```

```
class MainStatic
{
  public static void main(String args[])
  {
    float x = Staticmethod.mul(5.0f, 4.0f);
    S.O.println("Mul = "+x);
    float y = Staticmethod.div(x, 2.0f);
    S.O.println("Div = "+y);
  }
}
```

Output

Mul = 20.0
Div = 10.0

1)
```
class Static
{
  static int a,b;
  public static void main(String args[])
  {
    a=10;
    b=20;
    S.O.println("Static variable a = "+a);
    S.O.println("Static variable using class b = "+Static.b);
  }
}
```

O/P

Static variable a=10
Static variable using class b=20

Q. WAJP to demonstrate area of circle using static method.

```
class Circle_area
{ Static double rad=3;
  p s v main (String args[])
  { Circle_area c1=new Circle_area();
    c1. area_Circle(rad);
  }
  Static void area_Circle(double r)
  {
    S.o.println("Area of circle = "+(Math.PI *r*r));
  }
}
```

Output

* this [Keyword]

"this" refers to the object of the parent class. Generally we write for instance variables, constructors & methods in a class.

→ When an object is created to a class, a default reference is created internally to the object. The internal / default reference is "this".

Q. WAJP to demonstrate "this" keyword.

```
class A
{ int x; // Instance variable
  void var1(int x)
  {
    this.x = x;   -> Local variable
  }
  void print()
  { S.o.println(x);
  }
}
class Amain
{ public static void main(String args[])
  { A obj = new A();
    obj.var1(5);
    obj.print();
  }
}
```

Output

5

Q: WAJP to demonstrate constructors & methods using 'this' keyword. (without using objects)

not much imp

```
class Sample
{
  int x;
  Sample()
  {
    this(55);
    this.access();
  }
  Sample(int x)
  {
    this.x=x;
    S.o.println(x);
  }
  Void Access access()
  {
    S.o.println(x);
  }
}
class Thismain
{
  public static void main(String args[])
  {
    Sample obj=new Sample();
  }
}
```

Output
55
55

16/8/19

# Inner Class:

A class which is declared inside another class, such type of classes are called Inner classes.

→ Inner classes is introduced in java 1.1 version to fix GUI bugs. Now, slowly programers started using (graphic User Interface) regular coding also.

Ex: University consist of several departments, without existing university, there is no class of existing department. Hence, we have to declare, department class inside University class.

- Syntax of Inner Class

```
class Outerclass
{
  // Data members & methods of outer class (DM)
  class Innerclass
  {
    //Data members & methods
  }
}
class Mainclass
{
  p s v main(String args[])
  {
    Outerclass obj=new Outerclass(); // memory allocation for outer class
    obj.Method of Outerclass();
    Outerclass.Innerclass i=obj.new Innerclass();
  } // memory allocation for inner class
}
```

(outer class object; inner class object; Outer class object; constructor)

Save: Mainclass.java

Q) WAJP to demonstrate Inner class.

```
class University
{
  void ShowUniversity()
  { System.out.println("KITS Warangal");
  }
  class Department
  {
    void ShowDepartment()
    { System.out.println("ECE");
    }
  }
}
class Innermain
{
  public static void main(String args[])
  {
    University u= new University();
    u.ShowUniversity(); //Method
  University.Department d=u.new Department();
      d.ShowDepartment();
  }
}
```

Output

```
KITS Warangal
ECE
```

* Wrapper classes:

Convert It converts primitative datatypes to Objects.

→ Wrapper class is a class whose objects contains primitive datatypes.

| Datatype | Wrapper class | Constructor |
|---|---|---|
| byte | Byte | byte, String |
| short | Short | Short, String |
| int | Integer | int, String |
| long | Long | long, String |
| float | Float | float, double, String |
| double | Double | double, String. |
| char | Character | char |

Eg

```
class Wrapper
{
  p s v main(String args[])
  {
    Integer i1 = new Integer(100);
    Integer i2 = new Integer("100");
    System.out.println(i1+i2); //200
    Float f1 = new Float(5.5);
    S.o.println(f1);
    Character c= new Character('a');
    S.o.println(c);
  }
}
```

Output

```
200
5.5
a
```

1) WAJP to perform +, -, *, / using double Wrapper class.

2) WAJP to perform area of circle using Wrapper class.

2)
```
class Wrapper3
{
  public static void main(String[] args)
  {
    Double r = new Double(3.6);
    System.out.println("Area of circle = "+(3.14*r*r));
  }
}
```

Output

Area of circle = 40.6944

---

1)
```
class Wrapper2
{
  public static void main(String[] args)
  {
    Double d1 = new Double(3.2);
    Double d2 = new Double("4.3");
    S.O.println("Sum = "+(d1+d2));
    S.O.println("Subtraction = "+(d1-d2));
    S.O.println("Multiplication = "+(d1*d2));
    S.O.println("Division = "+(d1/d2));
  }
}
```

Output

Sum = 7.5

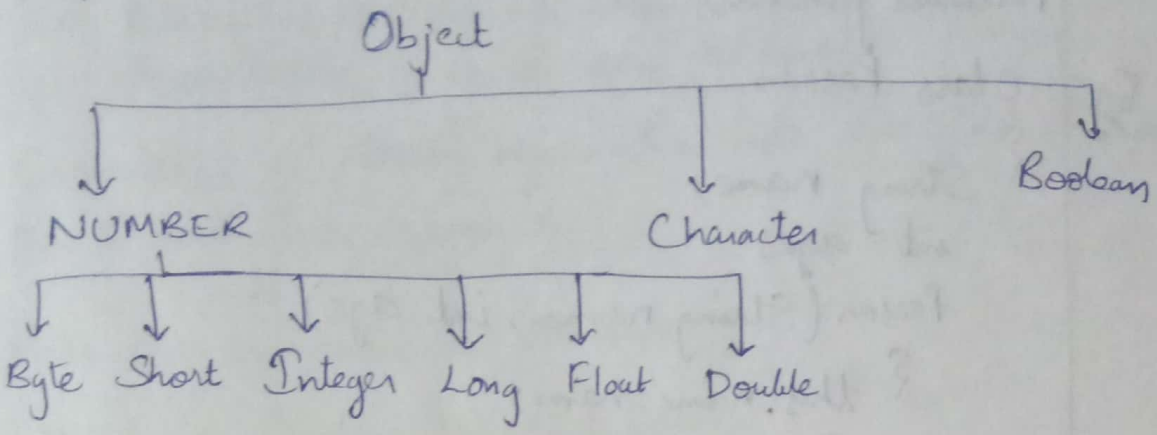
Subtraction = -1.09

Multiplication = 13.76

Division = 0.7441860465116278

11/8/19

* Wrapper class




* Methods of Wrapper classes

(i) toString()
(ii) valueOf()
(iii) xxxxValue()
(iv) parseYYYY()

(i) toString()

This method is belongs to the object, overriding to object class.

• Advantages

(a) The toString() method can display the object information
Ex: classname@hashcode.
[Test@17321]

(b) The toString() method display the wrapper class content.

* Example of toString()

```
class Test
{ p s v m(String args[])
  { Test obj = new Test();
    System.out.println(obj.toString()); // Test@17321
    Integer i = new Integer(100);
    System.out.println(i.toString()); //100
  }
}
```

Output: Test@1a79071
100

c) It can returns the class properties or class member functions.

Eg:
```
class Person
{ String name;
  int age;
  Person (String name, int age)
   { this.name=name;
     this.age = age;
   }
  Public String toString()
   {
    return name + "\n" + age + "\n";
   }
}
class toStringmain
{ public static void main (String args[])
   {
    Person obj = new Person("Prasen", 15);
    System.out.println (obj.toString());
   }
}
```

- Save: toStringmain.java
- Output
  Prasen
  15

(ii) Parse xxx ()

Ex: ParseInt() - converts String to int
ParseDouble() - converts string to double

→ Conversion of string information into specified datatype.

Eg: Command line arguments.

⇓

Pass the argument at runtime.

Q: WAJP to display Argument value.

```
class args1
{ p s v m (String args[])
   { int i = Integer.parseInt (args[0])
     System.out.println (i);
   }
}
```

Save: args1.java

Output compile: javac args1.java

run: java args1 10 ← Note: If we use Command argument line, then we should give value at run time.

Output
10

- If we use Data input Stream method, rather we should give value after runtime.

(iii) valueOf()

→ valueOf() method is inbuilt method of Wrapper class. It prints the value of string argument (or) integer argument.

Q: WAJP to perform addition two numbers using valueof method.

```
class Value1
{
  p s v main(String args[])
  {
    Integer x = Integer.valueOf(100);
    Integer y = Integer.valueOf(200);
    System.out.println(x+y);
  }
}
```

Output
300

(iv) xxxValue()

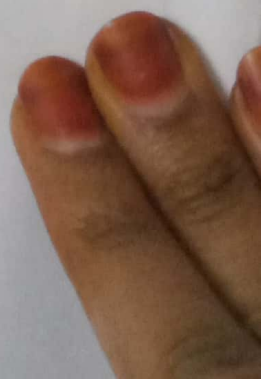
Int Value() – retrieves the integer wrapper class value

doubleValue() – retrieves the double wrapper class value

→ This also inbuilt method of wrapper class for retrieving (or) getting the object value.

Eg: (i)
```
Integer i = new Integer(100);
int x = i.intValue();
int y = i.intValue();
S.o.println(x+y);
```

(ii)
```
Float f = new Float(5.5);
float x = f.floatValue();
float y = f.floatValue();
S.o.println(x+y);
```



19/8/19
Exp 4 day

# (ii) INHERITANCE

Unit-2

* **Inheritance**

It is the process of creation of new things from already existing ~~one~~ one's.

→ Inheritance is basically done by creating new classes, re-~~using~~ using the properties of existing one's.

→ The mechanism of ~~new~~ deriving a new class from an old class. The old class is known as base class, Super class (or) parent class. and new one is known as derived class, sub class (or) child class.

• **Defination of Inheritance**

Inheritance of java is a mechanism in which one object acquire all the properties & behaviours of a parent object.

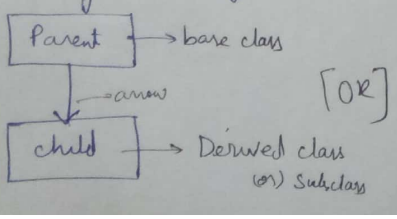
→ We have 4 types of Inheritance, they are

(i) Single Inheritance
(ii) Hierarchical Inheritance
(iii) Multiple Inheritance
(iv) Multilevel Inheritance

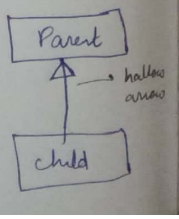
(i) **Single Inheritance**

It enables a derived class to inherit ance properties & behaviours from a single parent class.

Ex:-



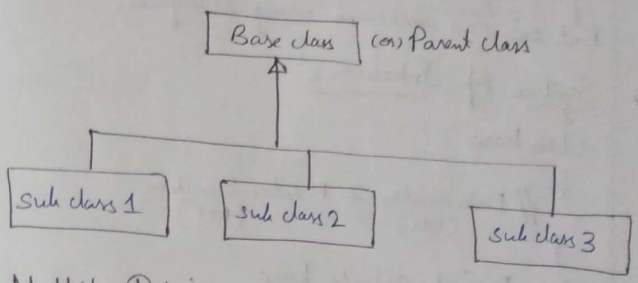

[OR]




(ii) **Hierarchical Inheritance**

A class has more than one child classes (or) sub classes.

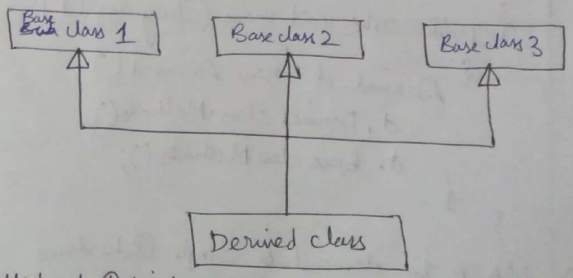
Eg




(iii) **Multiple Inheritance**

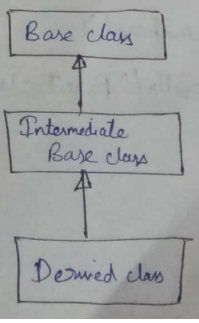
A class can inherit the properties of more than one parent class.

Eg




(iv) **Multilevel Inheritance**

When a class extends a class, which extends another class.



* NOTE

Java doesnot support multiple inheritance because of ambiguous error.
But we can overcome using interfaces.

• Syntax of Inheritance

```
class base
{
  // Data members & Member Functions
       (DM)              (MF)
}
class Derived extends base
{
  // DM & MF
}
class Inheritance
{
  public static void main (String args[])
  {
    Derived d = new Derived();
      d.Derived class Methods();
      d.base class Methods();
  }
}
```

1 Q WAJP to demonstrate Single Inheritance.

2 WAJP to perform +, -, *, / using Single inheritance

1)

```
class base
{
  public void showbase()
  {
    System.out.println("Parent class method \n");
  }
}
class derived extends base
{
  public void showderived()
  {
    System.out.println("Child class method \n");
  }
}
class Inher1
{
  p s v main(String args[])
  {
    derived d = new derived();
      d.showderived();
      d.showbase();
  } //main
} //Inheritance
```

+ NOTE

→ "Extends" is a keycoard to connects parent class & derived class.

21/8/19

2)

```
class A ---> parent
{
  void add(int x, int y)
  {
    S.o.println("Add = "+(x+y));
  }
  Void sub(int x, int y)
  {
    S.o.println("Subtraction = "+(x-y));
  }
}
class B extends A ---> child
{
  void mul(int x, int y)
  {
    S.o.println("Multiplication = "+(x*y));
  }
  Void div(int x, int y)
  {
    S.o.println("Division = "+(x/y));
  }
}
```

```
class Inher2
{
  p s v main (String args[])
  {
    B obj = new B();
    obj.add (10,20);
    obj.sub (20,10);
    obj.mul (20,5);
    obj.div (15,3);
  }
}
```

* Subclass Constructor (or) Super() Constructor

→ A subclass constructor is used to construct the instance variables of both the subclass & super class.

→ The subclass constructor uses the keyword "super" to invoke the constructor of parent method (or) super class.

• Properties of Super method / Subclass constructor

1) Super may only used within the subclass constructor.
2) To call the super class constructor, it must appear as the first statement within the subclass.
3) The parameters in super method must match the order & type of instance variable declared in the super class.

Q: WAJP to demonstrate Subclass constructor.

```
class A
{
  A()
  { S.O.println ("PARENT CLASS CONSTRUCTOR");
  }
}
class B extends A
{
  B()
  { super();
    S.O.println ("Subclass constructor");
  }
}
class Inher3
{ p s v main (String args[])
  {
    B obj = new B();
  }
}
```

Output
PARENT CLASS CONSTRUCTOR
Subclass Constructor.

Q: WAJP to find area of room & volume of bedroom using subclass constructor.

```
class Room
{ int len;
  int bre;
  Room(int len, int bre)
  { this.len = len;
    this.bre = bre;
  }
  void area()
  { S.o.println ("Area of room=" + (len * bre));
```

```
class BedRoom extends Room
{ int height;
  BedRoom( int len, int bre, int height)
  {
    super ( len, bre);
     this.height = height;
  }
  void volume ( )
  {
    S.O.println("Volume = "+ (len * bre * height));
  }
}

class Inher4
{
  p s v main (String args[])
  {
    BedRoom Obj = new BedRoom(5,7,8)
        obj.area();
        obj.volume();
  }
}
```

Output

area of room = 35

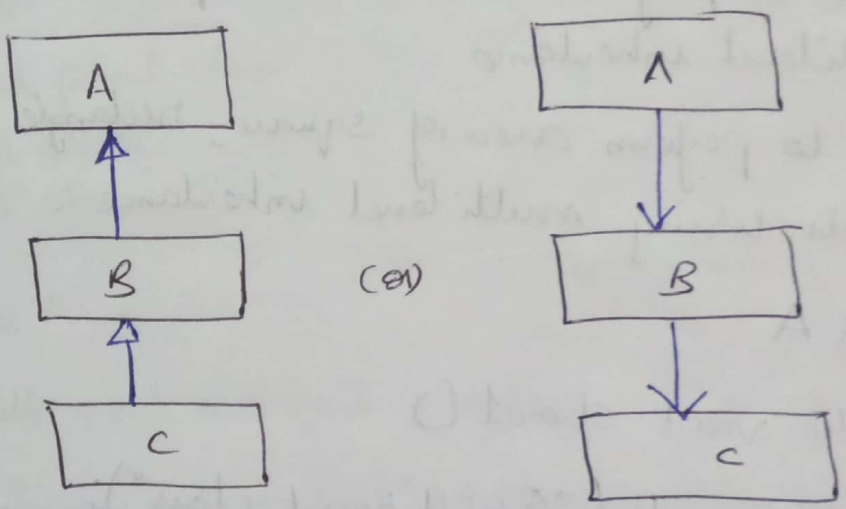
Volume = 280

4/9/19

# Multilevel Inheritance

When a class, extends to a class, which extends to another class then this is called Multilevel Inheritance.

Ex: class A serves as base class for the derived class B, Class B serves as base class for the derived class C. Therefore, indirectly class B is the intermediate base class of A, C.




- Syntax

```
class A
{
 // Data Members & Member functions
}

class B extends A
{
  // DM & MF
}

class C extends B
{
  // DM & MF
}
```

```
class mainc
{
    C obj = new C();
    obj.A class DM & MF
    obj.B class DM & MF
    obj.C class DM & MF
}
```

10. WAJP to demonstrate multilevel inheritance.

2) WAJP to perform +,- arithematic operations using multilevel inheritance

3) WAJP to perform area of square, rectangle & circle methods using multilevel inheritance.

1)
```
class A
{ public void show1()
  { S.o.println("Grand parent class");
  }
}
class B extends A
{ public void show2()
  { S.o.println("Parent class");
  }
}
class C extends B
{ public void show3()
  { System.out.println("Child class");
  }
}
```

```
class Multilevel1
{ p s v main(String args[])
  { C obj = new C();
    obj.show1();
    obj.show2();
    obj.show3();
  }
}
```

Output

Grand parent class
Parent class
Child class

2)
```
class A
{ public void add(int x, int y)
  { System.out.println("Addition = "+(x+y));
  }
}
class B extends A
{ public void sub(int x, int y)
  { System.out.println("Subtraction = "+(x-y));
  }
}
class C extends B
{ public void mul(int x, int y)
  { S.o.println("Multiplication = "+(x*y));
  }
}
```

```
class D extends C
{ public void div (int x, int y)
  { S.o.println ("Division = "+(x/y));
  }
}
class Multiher2
{ public static void main(String[] args)
  { D obj = new D();
    obj.add(2,3); → S.o.p("Arithematic Operations");
    obj.Sub(12,6);
    obj.mul(7,8);
    obj.div(20,4);
  }
}
```

Output

Arithematic Operations

Addition = 5

Subtraction = 6

Multiplication = 56

Division = 5

3)

```
class A
{ public void Squ(int x)
  { S.o.p("Area of Square = "+(x*x));
  }
}
class B extends A
{ public void rec(int x, int y)
  { S.o.p("Area of Rectangle = "+(x*y));
  }
}
class C extends B
{ public void cir(double x)
  { S.o.p("Area of Circle = "+(Math.PI*x*x));
  }
}
class Multiher3
{ public static void main(String[] args)
  { C obj = new C();
    obj.Squ(5);
    obj.rec(6,8);
    obj.cir(7);
  }
}
```

Output

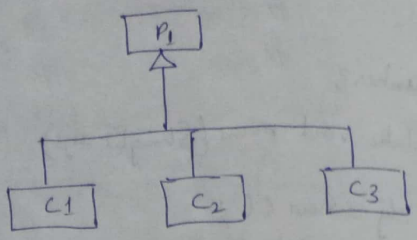
Area of Square = 25

Area of Rectangle = 48

Area of Circle = 153.93804002589985

## Hierarchical Inheritance

A parent class having more than one child class




- Syntax

```
class base
{
  // DM & MF
}
class child1 extends base
{
  //DM & MF
}
class child2 extends base
{
  //DM & MF
}
class Mainclass
{
  p s v m(String args[])
  {
    child1 c1 = new child1();
      c1.child1 DM & MF
      c1.parent DM & MF
    child2 c2 = new child2();
      c2.child2 DM & MF
      c2.parent DM & MF
  }
}
```

1) WAJP to demonstrate Hierarchical Inheritance

2) WAJP to display Students details of Mid 1 & 2 using Hierarchical Inheritance

2)

```
class Student
{ String name;
  int roll_no; // Instance variables
  void SetStudent(String n, int r)
                          local variables
  { name = n;
    roll_no = r;
  }
  void ShowStudent()
  {
    S.o.println("Name :" + name + "\n Roll no =" + roll_no);
  }
}
class Mid1 extends Student
{
  int m1, m2;
  void mid1Marks(int s1, int s2)
  { m1 = s1;
    m2 = s2;
  }
  void showMarks1() -> S.o.p("Mid 1 Marks");
  { S.o.println("Sub1 =" + m1 + "Sub2 =" + m2);
  }
}
class Mid2 extends Student
{
  int m1, m2;
  void mid2Marks(int s1, int s2)
  { m1 = s1;
    m2 = s2;
  }
  void showMarks2() -> S.o.p("Mid 2 Marks");
  { S.o.p("Sub1 =" + m1 + "Sub2 =" + m2);
  }
}
```

```
class Hie2
{ p s v main(String args[])
  {
    Mid1 obj1 = new Mid1();
    Mid2 obj2 = new Mid2();
     obj1.setStudent ("Vijay", 72);
     obj1.showStudent ();
     obj1.mid1Marks(20,22);
     obj1.showMarks1();
     obj2.mid2Marks(22,23);
     obj2.showMarks2();
  }
}
```

Output

```
Name = Vijay
Roll no = 72
Mid 1 Marks
 Sub1 = 20
 Sub2 = 22
Mid 2 Marks
 Sub1 = 22
 Sub2 = 23
```

6/9/19

# (i) POLYMORPHISM

Unit-3

(i) Overloading
(a) Method Overloading
(b) Constructor Overloading

(ii) Overridding

* **Polymorphism**

Ability to exist in different forms is called Polymorphism [OR] a state having many structures (or) shapes) is called Polymorphism.

* **Method Overriding**

The methods of the super class & Subclass have same name along with their signatures, then it is called Method Overridding.

[OR]

Overriding means having two or more methods with the same method parameters (method signature) One of the method is parent class and other one is child class.

* **NOTE**

Subclass method signature hides (or) overrides the Super class method signature.

* **Syntax of Overriding**

```
class Parent
{ public void show()
    { // Statements
    }
}
class Child extends Parent
{ public void show()
    { // Statements
    }
}
```

Q: WAJP to demonstrate method overriding

```
class Animal
{ public void eating()
  { System.out.println("Animal eating\n");
  }
}
class Dog extends Animal
{ public void eating()
  { System.out.println("Dog eating\n");
  }
}
class Override1
{ public static void main(String args[])
  { Dog d = new Dog();
    d.eating();
    d.eating();
  }
}
```

Output

```
Dog eating
Dog eating
```

• NOTE:
The solution of method overriding is "super" keyword.

* Super Keyword
Super Keyword in java is a reference variable (or keyword) that is used to reference the parent class objects (parent class properties (variables), parent class methods).

→ The super keyword came into the picture with the concept of Inheritance.

• Syntax of super keyword
```
super.methodname();
super.variable;
```

→ Super keyword is not allowed in static area.

Q WAJP to demonstrate super keyword using methodname
```
class Animal
{ public void eating()
  { S.o.println("Animal eating\n");
  }
}
class Dog extends Animal
{ public void eating()
  { S.o.println("Dog eating");
  }
  public void display()
  { eating();
    super.eating();
  }
}
class override-2
{ p s v main(String args[])
  { Dog d = new Dog();
    d.display();
  }
}
```
Output
```
Dog eating
Animal eating
```

Q: WAJP to demonstrate super variable class property using super variable
```
class Vehicle
{ int speed = 100;
}
class Car extends Vehicle
{ int speed = 180;
  public void displaySpeed()
  { S.o.println("Car speed=" + speed); //180
    S.o.println("Vehicle speed=" + super.speed); //100
  }
}
class Override3
{ p s v main(String arg[])
  { Car c = new Car();
    c.displaySpeed();
  }
}
```
o/p
```
Car speed = 180
Vehicle speed = 100
```

7/9/19 ** Difference b/w Overloading & Overriding ** V.V.V Imp

| Overloading | Overridding |
|---|---|
| 1) It occurs when 2 or more methods in a class have method name but different parameters. | (1) Overridding means having 2 methods with the same method signatures. One of the method is in parent class & other one is derived class. |
| 2) It is possible within the class. | (2) It is possible in inheritance concept. |
| (3) Method overloading is compile time polymorphism | (3) Method overridding is run time polymorphism |
| 4) Method overloading parameters are unique (different) | (4) Method overriding parameters are same. |
| 5) class A { Methodname() { } Methodname(parameters) { } } | (5) class A { Methodname() { } } class B extends A { methodname() { } } |
| 6) Eg: class A { sum(int a) sum(int a, int b) } | (6) Eg: class A { void show() { } } class B extends A { void show() { } } |

* Final Keyword

The final keyword in java is used to restrict the user. (modification)

→ The final keyword used for variables, methods and classes only.

(1) Final Variable

→ Final can be variable.

→ If you make any value as final, you cannot change the value of final variable. (it will be const)

Q. WAJP to demonstrate final variable.

```
class A
{ final int value = 5000;
  public void change()
  { value = 3000;
    S o println("Value = " + value);
  }
}
class final
{ p s v main(String args[])
  { A obj = new A();
    S o println("Value = " + obj.value);
    obj.change();
  }
}
```

× while executing

static area.

× while executing

• while executing o/p

Value = 5000

o/p

Compile time error.
Cannot assign a value to final variable value

* In static area, we should not use any keywords like final, super etc.

(i) Final Method

→ If you make any method as final, you cannot override it

Eg:

```
class A
{
final void run()
  { S.o.println("Parent Method");
  }
}
class B extends A
{
  void run()
    { S.o.println("Child Method");
    }
}
class final2
{
p s v main(String a[])
  {
    B obj = new B();
      obj.run();
  }
}
```

O/P:

Compile time error.
Cannot override.

→ Compile time error because the final method cannot override.

(ii) Final class

If you make any class as final, you cannot or inherit it.

Eg

```
final class Parent
{ public void parentShow()
  { S.o.println("Parent property & method");
  }
}
class Child extends Parent
{ public void childShow()
  { S.o.println("Child property & method");
  }
}
class final3
{ p s v main(String a[])
  { Child c = new Child();
    c.childShow();
    c.parentShow();
  }
}
```

O/P

Compile time error.
Cannot inherit from final Parent

9/9/19

* Compile time polymorphism (or) Static method Dispatch

→ Compile time polymorphism (or) Static method dispatch is a process in which a call to an overloading method is resolved at compile time, rather than run time.

We done overloading of method is called through the reference variable of a class. Here, no need super class.

Ex:
```
Class Cal
{ void sum (int a, int b)
  { S.o.println("Sum of two numbers = "+(a+b));
  }
  void sum (int a, int b, int c)
  { S.o.println("Sum of three numbers = "+(a+b+c));
  }
  public static void main (String args[])
  { Cal obj = new Cal();
    obj.sum(10,20);
    obj.sum(10,20,30);
  }
}
```

Obj is resolved at compile time before execution

O/P
Sum of two numbers = 30
Sum of three numbers = 60

* Run time polymorphism (or) Dynamic Method Dispatch

Overridden method is resolved at runtime is called DMD.

→ The overridden method is called through a super class reference.

→ A super class reference (or) super class variable refered to be subclass methods (or) subclass objects

→ The Syntax is ~~Superclass obj = new~~

```
Superclass obj1 = new Superclass();
        Superclass ref;
Subclass obj2 = new Subclass();
        ref = obj1;
        ref = obj2;
```

• Upcasting
When parent class reference reffered to the child class object, it is known as Upcasting.

Eg Example for DMD

```
class A
{ public void m1()
  { S.o.p("A's method");
  }
}
class B extends A
{ public void m1()
  { S.o.p("B's method");
  }
}
```

```
class C extends A
{ public void m1()
  { S.O.P("C's method");
  }
}
class RP
{ p s v m (String args[])
  { A obj1 = new A();
    A ref;
    ref = obj1; ref.m1();
    B obj2 = new B();
    ref = obj2; ref.m1();
    C obj3 = new C();
    ref = obj3;
    ref.m1();
  }
}
```

o/p

A's method
B's method
C's method

Q. WAP to demonstrate multilevel inheritance using DMD.

```
class A
{ public void m1()
  { S.O.P("A's method");
  }
}
class B extends A
{ public void m1()
  { S.O.P("B's method");
  }
}
class C extends B
{ public void m1()
  { S.O.P("C's method");
  }
}
class RunP2
{ p s v m (String args[])
  { A obj1 = new A();
    A ref1;
    B obj2 = new B();
    ref1 = obj1;
    ref1.m1();
    ref1 = obj2;
    ref1.m1();
    B ref2;
    C obj3 = new C();
    ref2 = obj3;
    ref2.m1();
  }
}
```

o/p

A's method
B's method
C's method.

11/9/19

## * Abstract class

It is a class that generally contains some abstract class methods & non-abstract methods.

→ Abstract class declaring with "abstract" keyword.

→ Each abstract class should have one abstract method.

- Abstract Method

It is declared with abstract keyword does not contain any body or implementation.

- Syntax for abstract class

```
abstract class classname
{
  abstract Acc_spe returntype Methodname(parameters);
  Acc_spe returntype methodname(parameters)
  {
    // Method body
  }
}
```

(Acc_spe → Access specifier)

## * NOTE

(i) For abstract class, object creation is not possible

(ii) Abstract class allows static methods, final methods & constructors.

(iii) We cannot create object to the abstract class, but we can create reference variable to the abstract class

Q. WAJP to implement Square, cube, Squareroot using abstract class.

```
abstract class Calculate
{ abstract public void square(int x);
  public void cube(int x)
  {
    System.out.println("Cube="+(x*x*x));
  }
  public void sqroot(int x)
  {
    System.out.println("Sqrt(square root)="+Math.sqrt(x));
  }
}
class der extends Calculate
{ public void square(int x)
  { S.o.println("Square="+(x*x));
  }
}
class Abstract1
{ p s v m(String args[])
  { der obj=new der();
    obj.square(4);
    obj.cube(4); //16
    obj.sqroot(4); //9
  }
}
```

o/p

Square=16
Cube=125
Squareroot=3.0

1 → Abstract method can access more than one da...

Ex:

```
abstract class Sup1
{ abstract public void calculate (int x);
}
class sub1 extends Sup1
{ public void calculate (int x)
   { S.o.println ("Square = "+(x*x));
   }
}
class sub2 extends Sup1
{ public void calculate (int x)
   { S.o.println ("Cube = "+(x*x*x));
   }
}
class Abstract2
{ public static void main (String args[])
   { Sub1 S1 = new sub1();
     S1.calculate (4);
     Sub2 S2 = new sub2();
     S2.calculate (3);
   }
}
```

o/p

Square = 16
Cube = 125

Abstract class reference variable

Ex:

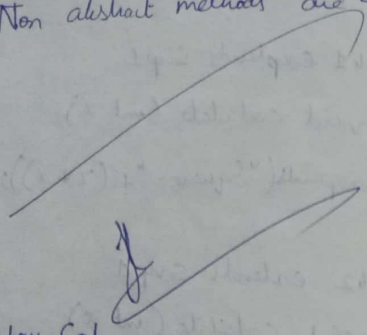
```
abstract class Sup sup1
{ abstract public void calculate (int x);
}
class sub1 esptends Sup1
{ public void calculate (int x)
   { S.o.println("Square = "+(x*x));
   }
}
class sub2 extends sup1
{ public void calculate (int x)
   { S.o.p ("Cube = "+ (x*x*x));
   }
}
class Abstract3
{ p s v main(String s[])
   { Sup1 ref;
     Sub1 S1 = new Sub1();
     Sub2 S2 = new sub2();
     ref = S1;
     ref.calculate (4);
     ref = S2;
     ref.calculate (5);
   }
}
```

o/p

Square = 16
Cube = 125

6. WAJP to implement Arithmatic operations using Abstract class

* NOTE:- The abstract methods are -, *.
Non abstract methods are +, /.



```
abstract class Cal
{ abstract public void Sub(int x, int y);
  abstract public void mul(int x, int y);
  public void sum(int x, int y)
  { S.o.p("Sum=" +(x+y));
  }
  public void div(int x, int y)
  { S.o.p("Division=" +(x/y));
  }
}
class Child extends Cal
{ public void Sub.(int x, int y)
  { S.o.println("Subtraction=" +(x-y));
  }
  public void mul(int x, int y)
  { S.o.println("Multiplication=" +(x*y));
  }
}
```

```
class Abstract4
{ pub s v m (String[] args)
  { Child c = new Child();
    c.Sub(15,3);
    c.mul(12,76);
    c.Sum(19,50);
    c.div(8,4);
  }
}
```

o/p

Subtraction = 12
Multiplication = 912
Sum = 69
Division = 2

16/6/14

## (ii) INTERFACES

Unit-3

→ Interfaces contain pure abstract methods and final static variables (or) fields.

[OR]

→ We should not declare "abstract" keyword for abstract methods. Therefore, by default interface methods are abstract methods.

→ All variables defined in the interface are by default static & final.

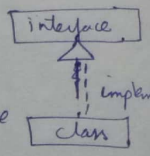
• Syntax for Interface

```
interface interfacename
{
  [static & final] datatype variable;
                   Returntype Methodname();
}
```

by default it exists

with dashed line & hollow arrow head

→ Class implements interface




• Syntax

```
class classname implements interfacename
{
    // body
    Returntype Interfacemethod()
    {
      // body
    }
}
```

→ In interface, we cannot create object for interface but we can create object to the class

Q: WAJP to demonstrate interface.

```
interface Inter1
{ int a=10;          abstract method
  void display();
}
class C1 implements Inter1
{
  public void display()
    { S.o.p("Interface Method");
    }
}
class Interface1
{ p s v m (String args[])
  {
    C1 obj1 = new C1();
      obj1.display();
    S.o.println("a=" + Inter1.a);
  }
}
```

* Interface variables are accessed by using "public" only.

Save: Interface1.java

o/p:

```
Interface Method
  a=10
```

→ We do not create objects for interfaces & abstract methods, but we create objects only for classes.

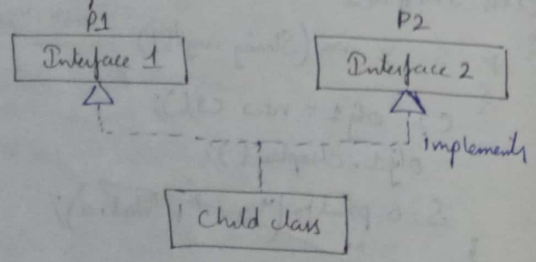
## To Overcome multiple inheritance through interfaces

→ Java doesnot support multiple inheritance that is using classes.

→ A large no. of real time applications require the use of multiple inheritance.

→ Java provides an alternative approach known as interfaces to support the concept of multiple interfaces.

- Structure of




- Syntax

```
interface Inter1
{ // abstract methods & final static variables
}
interface Inter2
{ // abstract methods & final static variables
}
class Child1 implements Inter1, Inter2
{ // Development of Interface methods
  &
  // Class methods
}
```

16/9/19

Q- WAJP to demonstrate multiple inheritance through interfaces.

Eg: [Interface 1 is father class, father variable are height (ht) father method is display father height.
Interface 2 is mother class, mother variable are height mother method is disp mother height.
Child class implements to mother & father. In child class use method which display child height which is equal to avg of mother height & father height]

for static variable

```
interface Father
{ float htf = 6.6f;
  void fatherHt();
}
interface Mother
{ float htm = 5.6f;
  void MotherHt();
}
class Child implements Father, Mother
{ public void fatherHt()
  { S.o.println("Father height =" + Father.htf);
  }
  public void MotherHt()
  { S.o.p("Mother height =" + Mother.htm);
  }
  public void ChildHt()
  { float htc = (Father.htf + Mother.htm)/2;
  } S.o.p("Child height =" + htc);
}
```

static variable

local variable

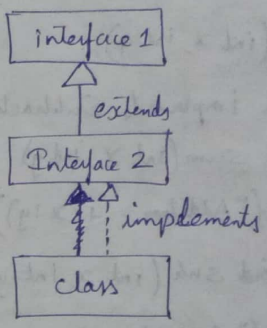
```
class Interface2
{ p s v m (String args)
  { Child c = new Child();
      c.FatherHt();
      c.MotherHt();
      c.ChildHt();
  }
}
```

o/p

Father height = 6.6
Mother height = 5.6
Child height = 6.1

* Interface extending another Interface

Similar to classes an interface can extend to another interface but only single inheritance is possible.




- Syntax

```
interface Inter1
{ // abstract methods & static - final variables
}
interface Inter2 extends Inter1
{ // A.M & S,F variables
}
class Classname implements Inter2
{ // Data members & Member functions
}
```

Q) WAJP to demonstrate +, − using interface

```
interface Addition
{ void sum(int x, int y);
}
interface Subtraction extends Addition
{ void sub(int x, int y);
}
class Addsub implements Subtraction
{ public void sum(int x, int y)
  { S.o.p("Addition = "+(x+y))
  }
  public void sub(int x, int y)
  { S.o.p("Subtraction = "+(x-y));
  }
}
class Interface3
{ p s v main(String args[])
  { Addsub obj = new Addsub();
    obj.sum(10,20);
    obj.sub(20,10);
  }
}
```

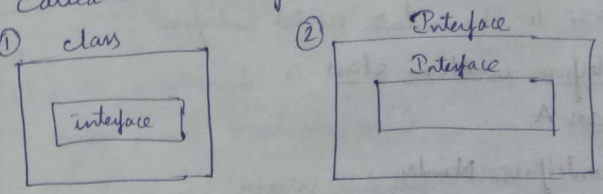
O/P

Addition = 30
Subtraction = 10

* Nested Interface

An interface can be declared as a member of a class (or) other interface. Such an interface is called Nested interface.

Ex:




Syntax

① i) class classname
{ access specifier (public/private) interface interfacename
  {
  }
}

② i) interface Interface name1
{ acc_spec interface interface name 2
  {
  }
}

17/9/19

ii) class der implements interfacename1, interfacename2
{ //implements
}

PTO

1(ii) class der implements classname, interfacename
{
// Implements
}

Zoblia

Q: WASP to demonstrate nested interface

- Interface within class

```
class A
{ interface Ninter
  { void display();
  }
}
class B implements A.Ninter
{ public void display()
  { System.out.println("Nested Interface Method");
  }
}
class Interface4
{ p s v m(String args[])
  { B obj = new B();
    obj.display();
  }
}
```

O/P
Nested Interface Method

Q WASP to demonstrate nested interface (interface within class) using Dynamic Method Dispatch.

```
class A
{ interface Ninter
  { void display();
  }
}
class B implements A.Ninter
{ public void display()
  { S.o.p("Nested Interface Method");
  }
}
class Interface5
{ p s v m (String args[])
  { A.Ninter ref;
    B obj = new B();
    ref = obj;
    ref.display();
  }
}
```

O/P
Nested Interface Method

→ The above program is runtime polymorphism ie different classes are called at different times ie for every runtime, different classes are called.

Q. WAJP to demonstrate Interface within interface

a) nested interface

```
interface A
{
  interface Nested
  {
    void display();
  }
}
public class B implements A.Nested
{
  public void display()
  {
    S.o.println("Nested Interface Method");
  }
}
class InterfaceB
{
  p s v main(String args[])
  {
    A.Nested ref;
    B obj = new B();
    ref = obj;
    ref.display();
  }
}
```

Public is optional may not be there it will execute

o/p

Nested Interface Method

Compare & contrast Abstract & Interface

V.V.Imp

| Abstract class | Interface |
|---|---|
| 1) Abstract is a class which contains abstract methods & non-abstract methods. | 1) It contains pure abstract methods and static, final variables. |
| 2) Ex: class abstractclass { abstract void display(); // Abstract method void show() { S.o.p("NAM"); // non abstract method } } | 2) Ex: interface inter1 { void show1(); void show2(); } |
| 3) Abstract methods are developed in derived classes without "abstract" keyword. → above programs: class der extends abstractclass { void display() { S.o.p("Abstract method"); } } | 3) Interface method is accessed used "Public" keyword. → above programs: class A implements inter1 { public void show1() { S.o.p("Interface method"); } } |
| 4) Abstract variables are static, final & local variables (non static, non-final, non-local) interface. class abstract class { int a; ✓ int a=10; ✓ } | 4) Interface variables are static & final variables. interface inter1 { int a; X int a=10; ✓ } |

| | Abstract class | Interface |
|---|---|---|
| 5) | Class extends to abstract class | 5) Class implements to interface (or) Interface extends to interface. It allows Nested interfaces |
| 6) | Using abstract class we cannot overcome multiple inheritance | 6) Using interface we can overcome multiple inheritance. |
| 7) | For creating abstract class we use "abstract" keyword | 7) "interface" keyword is used to declare interfaces |
| 8) | It allows public, private, protected access specifiers | 8) It allows only public |

21/9/19

## (iii) PACKAGES

→ Java is a group of similar classes, interfaces and sub packages.

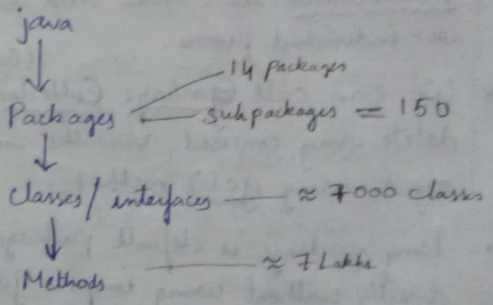
→ "Package" is a container of class.

→ Package is directories in our system (directories of system folders).

→ Packages provides a way to hide classes.

→ Lot of inbuilt classes, put in in one folder is called "Packages"

* Hierarchical structure of Packages



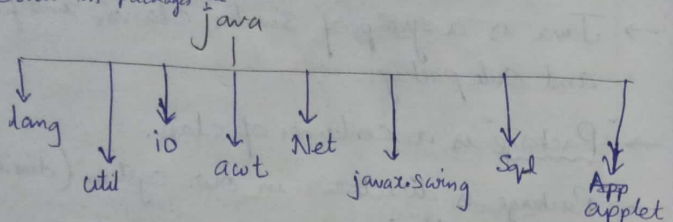

→ Packages are categorised into 2 types

(i) Builtin packages (or) System defined packages (or) Java API packages (API - Application programming Interface)

(ii) User defined packages

(i) Built In Packages & Using builtin packages

Built in packages are




(a) lang package

→ It consist of Wrapper classes which used to convert primitive datatypes into objects.

→ There are classes like String, StringBuffer, String Tokenzer .. etc.

→ Thread classes to create and execute tiny proces (or) individual proces.

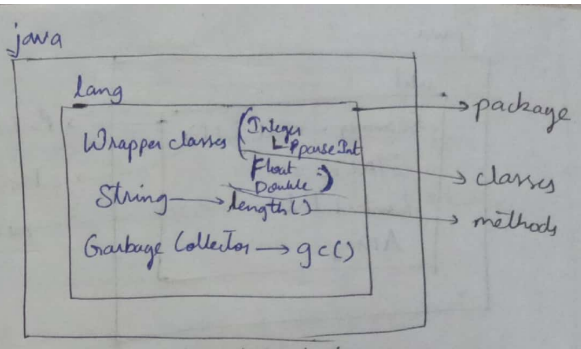
→ We can call Garbage Collector (GC) of JVM to delete any unused variables and unreferenced objects using gc() method.

→ lang package is default package, we can use directly without using import java.lang.*

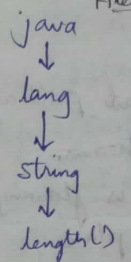
• Using lang package

import java.lang.Integer; ← for accessing Wrapper class

import java.lang.String; ← for accessing String class




Hierarchical structure



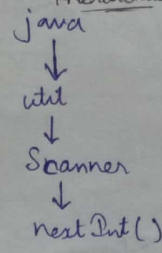

(b) util package [Utility package]

→ util stands for utility.

→ The package contain, useful classes, interfaces.

Ex:- Sk-STACK, Linked List, Arrays, Hash Table, Scanner.

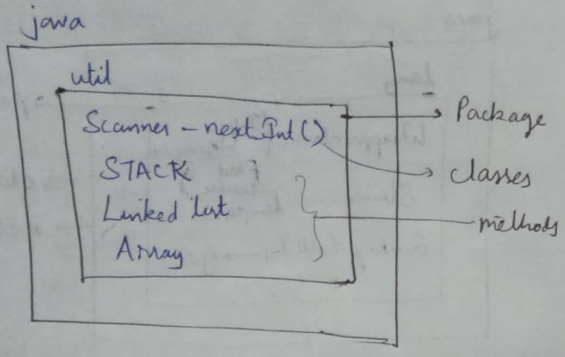
Hierarchical structure




import java.util.*;

import java.util.Scanner;

import java.util.Stack;




(c) io package

→ io stands for input output package.

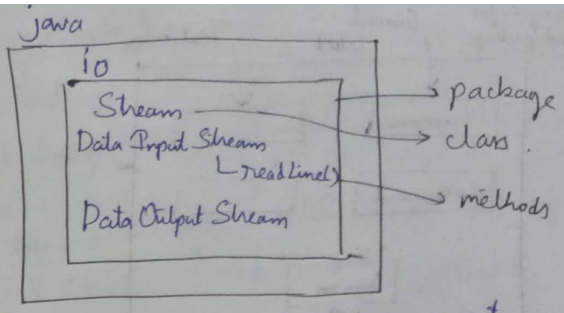
→ This package contains Streams.

→ A Stream representation is flowing the data from one place to another place.

→ Streams are very useful in file reading, file writing file modification

- import java.io.*;
- import java.io.DataInputStream;
- Hierarchical structure

java
↓
io
↓
Data Input Stream
↓
readLine()



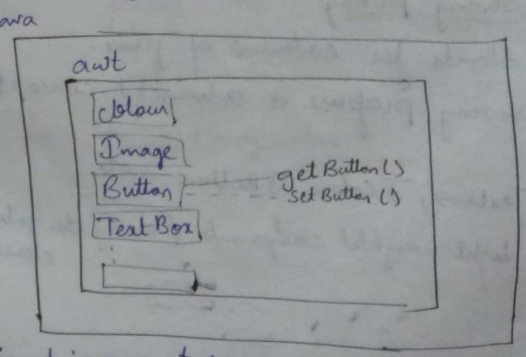

(d) awt package → imp subpackage : event

→ awt stands for Abstract Window toolkit.

→ This packages helps to develop GUI programs with colourful. (GUI - Graphical User Interface).

- Hierarchical structure

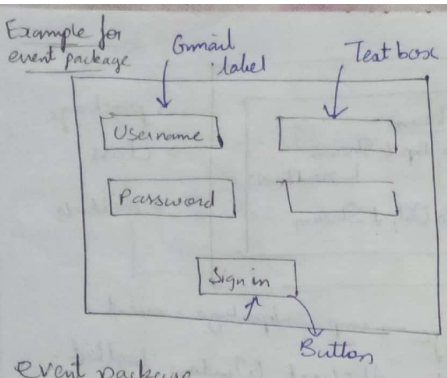
java
↓
awt
↓
Image
↓
SetImage()
setIcon()




→ import java.awt.*;

→ import java.awt.Image;

→ Example for event package




• Event package

→ Event package is a subpackage of awt package.

→ This package have classes & interfaces.

import java.awt.event.*;

↓

Mouse Motion Listener [Interface]

↓

(Mouse event)

mouse Pressed();
mouse Release();
mouse Dragged();
mouse Click();
} [methods]

(e) javax.Swing Package

→ javax stands for extension of java.

→ javax.swing purpose is advanced concepts of awt.

Ex: J Textbox, J Colour, J Button ... etc

→ It is light weighted components (easily gets develop).

(f) java.net package

→ net stands for Networking

→ Client & Server to establish communication through sockets.

Ex: Client is a system & Server maintaining all client details (Facebook)

→ Examples for network is socket.

• Hierarchical structure

java
↓
net
↓
Socket
↓
Socket Stream()

(g) java.sql package

→ Sql stands for Structured Query Language.

→ Purpose: to establish the connection b/w database

→ This package helps to connect database.

Ex: java
↓
Sql
↓
Driver Manager class

ODBC
JDBC

(h) java.applet package

→ The programs are executed in web browser.

→ All programs are available in HTML (Hyper Text Mark up Language)

java
↑
applet package
↑
Applet class
↑ init the package (create)
init() → create
Run() → execute
Stop()
destroy()

23/9/19

* Naming Conventions in Packages.

→ Package name should be in lower case letters.

→ Packages can be named using the standard java naming rules.

→ Packages begins with lower case letters

→ Package names letters are in lower case letters.
Ex: lang, util, imports

→ We know that all class names begin with Upper case letter.

Ex: double a = java.lang.Math.sqrt(2);
(lang: package, Math: class)

* User-defined Packages

Package → keyword

→ Creating our own packages.

(1) create package

→ We must first declare the name of the package using "package" keyword followed by a package name.

package <Directing>; (01)

package packagename; (01)

package pack1[.pack2][.pack3]....;

→ Advantage of package avoiding name space collision (or) naming collisions.

. Name space collisions

→ Two / more class files have same names and this is known as name-space collisions. In this process older class files deleted, new class files are stored.

. package syntax

```
package packagename;
public class A
  { // body Data members & Member functions (MF)
  }                    (DM)
class B
  { // DM & MF
  }
```

- Following steps

(1) Declare the package at beginning of a file using package keyword.

(2) Define the class that is in the package and declare public keyword.

3) One public class should be in each package.

4) Save the program as public classname.java. (classname where public is present import of it) Ex: A.java

5) Compile this file using javac

Ex: javac -d . A.java

(spaces)

javac -d . classname.java

6) Javac is a java compiler, -d is creating directory (or) folder structure, "." is place the folder (or) it is place the ".class" files in current directory.

Pack1
A.class
B.class

(2) Accessing package

→ Accessing a package is a shortcut approach through the "import" keyword / statement.

→ The "import" statement can be used to search a list of packages for a particular class.

- Syntax

import pack1.*;

(or)

import pack1.classA; (or) import pack1.classB;

(or)

import pack1[.pack2][.pack3] .... .classname;

→ We are importing all packages in main class.

E:
└→ B18EC137
└→ java program
Pack1
A.class
B.class

25/9/19

Q: WAJP to demonstrate Packages.

creation of Packages

```
Pa-package pack;
public class ClassA
{ public void display()
  { System.out.println("Welcome to user defined packages");
  }
}
```

java
↓
lang — pack
↓
String → classA
↓
length() → displays

- Save: ClassA.java
- Compile: javac -d . ClassA.java

WAJP to display msg using packages with import statement

```
import pack.ClassA;
class Package1
{ public static v m (String args[])
  { ClassA obj = new ClassA();
    obj.display();
  }
}
```

- Save Package1.java
- Compile javac Package1.java
- Run - java Package1
- O/p: Welcome to user defined packages.

To demonstrate simple package program without import statement

```
class Package2
{ p s v m(String[] args)
  {
    Pack.ClassA obj = new Pack.ClassA();
    obj.display();
  }
}
```

- Save: Package2.java
- Compile: javac Package2.java
- Run: java Package2

o/p - Welcome to user defined packages

WAJP to perform addition of 2 numbers using packages with constructors.

```
package pack;
public class Addition
{ public double d1,d2;
  Addition(double x, double y)
  { d1=x
    d2=y
  }
  public void sum()
  { S.o.p("Addition="+(d1+d2));
  }
}
```

- Save: Addition.java
- compile: javac -d . Addition.java

```
import pack.*;Addition;
class Package3
{ p s v m(String[] args)
  { Addition a = new Addition(10.0, 20.0);
    a.sum();
    //ClassA b = new ClassA();
    // b.display();
  }
}
```

- Save: Package3.java
- Compile: javac Package3.java
- o/p: Addition = 30.0

WAJP to add, sub, multiplication & division to pack and display / evaluate Arithematic operations classes +, -, *, /.

# Access Specifiers in Java

| | | Public | Private | Default | Protected |
|---|---|---|---|---|---|
| 1) | Same class | Yes | Yes | Yes | Yes |
| 2) | Same package & non-subclass | Yes | No | Yes | Yes |
| 3) | Same package & subclass | Yes | No | Yes | Yes |
| 4) | Different package & non-subclass | Yes | No | No | No |
| 5) | Different package & subclass | Yes | No | No | Yes |

Q: WAJP to implement area of circle class, area of rectangle class in area package.

```
package area;
public class Areaofcir
{ double r;
  public Areaofcir(double rad)
    { r = rad;
    }
public void cir()
    { S.o.p("Area of circle = "+(Math.PI * r * r));
    }
}
```

• save: Areaofcir.java
• compile: javac -d . Areaofcir.java

```
package area;
public class Areaofrect
{ double l,b;
  public Areaofrec(double len, double bre)
    { l = len;
      b = bre;
    }
public void rec()
    { S.o.p("Area of Rectangle = "+(l * b));
    }
}
```

• save: Areaofrec.java
• Compile: javac -d . Areaofrec.java

Structure 1: D:
↳ BIECB37
↳ area
Areaofcir.class
Areaofcir.java

Structure 2: D:
↳ BIECB37
↳ area
Areaofcir.class
Areaofcir.java
Areaofrec.class
Areaofrec.java