OBJECT ORIENTED PROGRAMMING

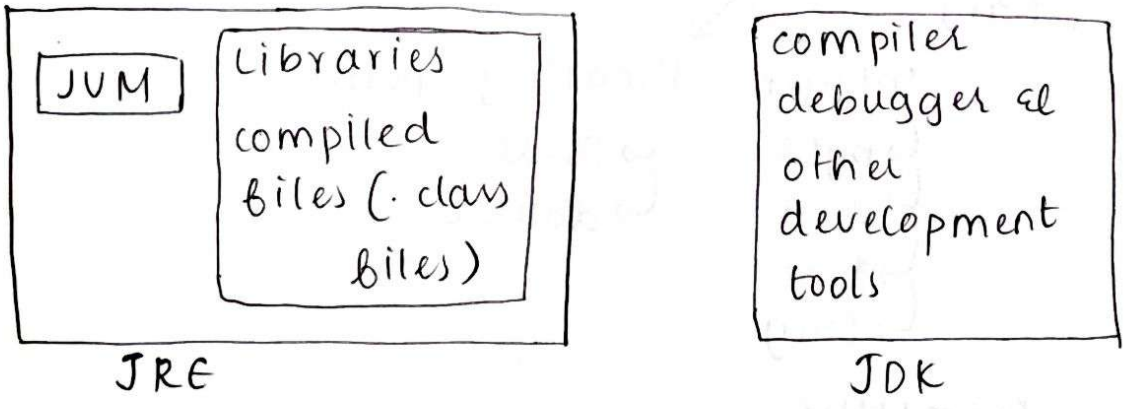
Java

Features / Buzzwords

1. Simple
2. Robust : check error in both compile and run time.
3. Platform independent : write once and run anywhere
4. Multithreading : multi procesing at a time.
5. Portable
6. ~~Byte code~~ security : source code converted into byte code during compilation. If shared, data is not directly ~~act~~ accessible.
7. Object oriented programming : Relates real world entities and it makes the data secure.

Java Development Environment

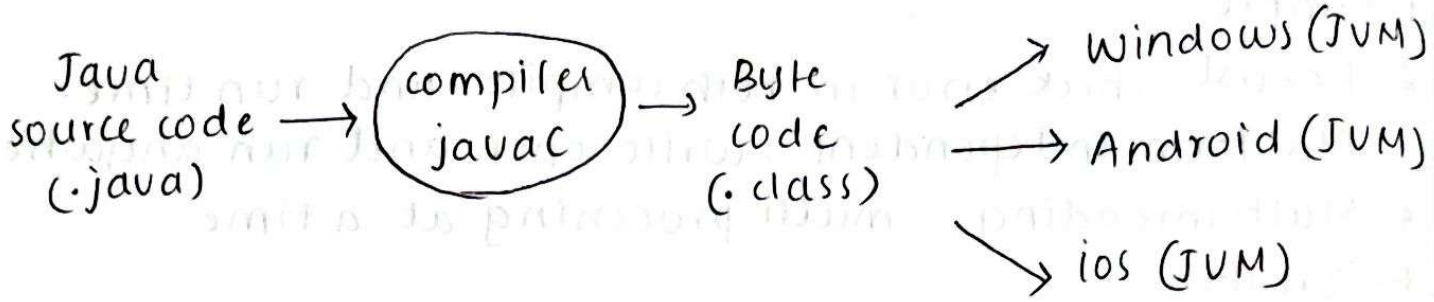
1. JDK : Java development kit has the tools for developing Java
2. JRE : Java run time environment is only a run time environment
   - ↳ subset of JDK
   - ↳ JVM (Java virtual machine)
   - ↳ Libraries



- class files : extension for byte code (compiled files)
- java : extension for java program.

JVM : Java virtual machine

- runs java program
- does memory management
- It uses JIT compiler (Just in time)



- when JVM runs the byte code, it act as an interpreter. Since it is time consuming, JVM uses JIT to compile loops etc.

19/7 Interpreter : line by line
Compiler : whole prgm.

Data types in Java

1. Primitive
   - Boolean (boolean)
   - Numeric
     - character (char)
     - Integral
       - Integer
         - byte
         - short
         - int
         - long
       - Floating point
         - float
         - double

2. Non-Primitive

Type	Default value	Size
byte	0	1 byte (8 bits)
short	0	2 bytes (16 bits)
int	0	4 bytes
long	0L	8 bytes
float	0.0f	4 bytes
double	0.0d	8 bytes
boolean	false	1 ~~byte~~ bit
char		2 bytes

Variable Types

1. Local variable : used within a method (function)

```
eg: add()
    {
    int a=5; // local variable
    }
```

2. Instance variable : used outside of a method and inside of a class.

```
eg: class A
    {
      int b=7; //instance variable
      add(){
          int a=5; // local variable
      }
    }
```

3. Static variable : keyword : static

```
eg: static int i=10;
```

Type Casting & Type Conversion

- when you asign a value of one primitive to another is type casting / conversion

Two types : 1. widening casting / Type conversion

- convert a small type to large type

byte → short → int → long → float → double

2. Type casting / Narrow casting
   - convert large type into small type.

eg: 1) int num = 10;
       double b = num;
   } Type conversion

   2) double a = 5.99;
      int b = (int) a;
   } Type casting

IMP Operators in Java

Bitwise operator

1. Bitwise AND

eg: int a = 10;
    int b = 3;
    int y = a & b;

∴ y = 2

10 : 1010
3 : 0011
& 0010 = 2
| 1011 = 11

2 | 10 | 0
2 | 5 | 1
2 | 2 | 0
1
2 | 3 | 1
1

2. Bitwise OR

eg: int x = a | b;

∴ x = 11

3. Bitwise compliment

eg: ~1010 = 0101

4. Left shift

eg: int a = 9;
    int b = <<a;

∴ b = 2

discard
a 9 : 1001
<<9 : 0010 = 2
>>9 : 0100 = 4

5. Right shift

eg: c = >>a;

∴ c = 4

Arithmetic operation
Relational operator
Assignment operator
Ternary operator

18/7

	Precedence order	Associativity
1.	Parenthesis ( ), [ ], ·	L → R
2.	++, -- (post increment)	R → L
3.	++, -- (pre), +, - (unary) ! (logical negation), ~ (compliment)	R → L
4.	*, /, %	L → R
5.	+, - (add^n, subtrac^n)	L → R
6.	<<, >>	L → R
7.	~~& (Bitwise AND)~~ <, <=, >, >=	L → R
8.	==, !=	L → R
9.	& (Bitwise and)	L → R
10.	^ (Bitwise exclusive OR)	L → R
11.	| (Bitwise or)	L → R
12.	&& (logical and)	L → R
13.	|| (logical OR)	L → R
14.	? : (ternary)	R → L
15.	=, +=, -=, *=, /=, %=	R → L

Q. Evaluate i) i = 2 * 3/4 + 4/ 4 + 8 - 2 + 5/8

2) ++a + a++ + a [int a = 5]
(+ addition)

3) 10 + 20 * 3 - 5/5

4) truee && false || true.

→ 1) i = 1+1+8-2+0 = 8

2) ~~16~~ 19

3) 10+60-1 = 69

4) true

3) Selection statements

- Using label

```
Label 1 : for (      ) {
              :
          for (      ) {
                                  } inner loop
            break Label 1;
          }                       } outer loop
        }
```

→ using Label 1 at break jumps outside the outer loop (not just the inner loop).

Array :- collection of similar data types
- the elements are ~~stored in~~ stored in consecutive bases.
- size of array is fixed, once declared cannot be increased (static memory allocation).

Declaration of array in Java

eg: `int [] arrayname;` / one dimensional
`int[][] array;` / 2-D array

Initialise array size

eg: • `int[] arr = new int[10];`

or

`int [] arr = {1,2,3,4,5}` / automatically size = 5.

• `int[][] arr = new int[10][10];`

17/1 **OOP CONCEPT**

1. Class
   - Blueprint / template from where the objects are created.

- class: Logical concept
- properties/attributes and ~~barriers~~ behaviors

Syntax:

```
class Student
    {
    String name;
    int age;
    String addren;

    void SubmitAssignment()
        {
        System.out.println("Submitting
                            Assignment")
        }
    void View-Marks()
        {
        System.out.println("Veiwing mark")
        }
    void ParticipateInEvents()
        {
        System.out.println("Participate
                            in events")
        }
    }

class main
  {
  public static void main(String[] args)
    {
    Student str1=new Student();

    str1.name = "Hari";
    str1.age = 20;
```

```
    str1.address = "...";
    str1.SubmitAssignment();
    str1.ViewMark();
    str1.ParticipateInEvents();
      }
  }
```

File name : Main.java
Compilation : javaC Main.java
Run : java Main

```
o/p : Submitting assignment
      View marks
      Participating in events.
```

Command Line Arguments in Java

- Command line arguments are the values that you can pass to a prgm in terminal.

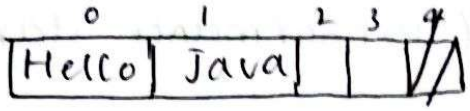
- This arguments are used to provide input to the program at run time from the terminal or command prompt

```
public class CommandLineDemo
  {
  public static void main(String[] args)
  System.out.println("Number of arguments : "
                          + args.length);
  }
}
```

filename : commandlinedemo.java

Compile : javac commandlinedemo.java

Run : java commandlinedemo Hello java



o/p : Number of arguments : 2

⇒
```
public class CommandLineDemo
  {
  public ~~voi~~ static void main (String[] args)
    {
    int i = Integer.parseInt (args[0]);
    int j = Integer.parseInt (args[1]);
    int sum = i+j;
    System.out.println (sum);
      }
  }
```

18[7]

Abstract Class

- An abstract class in java is a class that cannot be instantiated (cannot make objects for the particular class) of its own.
- It ~~cannot~~ is meant to be inherited by other classes
- Used an 'abstract' keyword to declare the abstract class.
- It contains
  1. Abstract methods (method without body)
  2. Concrete methods (regular methods with body).
  3. Field / Variable

Eg:
```
abstract class Musician
  {
   void performance()
     {
      system.out.println("Performance started");
     }
   abstract void playInstrument();
  }
class Guitarist extends Musician
  {
void playInstrument()
   {
    system.out.println("Playing guitar");
   }
 }
class Drummer extends Musician
    {
     void playInstrument()
       {
        system.out.println("Playing Drums");
       }
    }
class Main{
   public static void main(String[] args){
   Guitarist G1 = new Guitarist();
   G1.performance();
   G1.playInstrument();
   Drummer D1 = new Drummer();
   D1.performance();
   D1.playInstrument();
}}
```

to show Guitarist is a child class

file name : Main.java
compile : javac Main.java
Run : java Main.

o/p : Performance started
Playing Guitar
Performance started
Playing Drums

Q. Create a class, named person in which 3 method are used ie, Display role, show details and welcome.
Display role is abstract. show details gives id and name. ~~Class str~~ Welcome method displays Welcome to the institution.
Class, student and teacher are two child classes of person. Display role is implemented in these two classes, as the role of teacher and student are different.

→
```
abstract class Person {
    String name;
    int id;
    abstract void displayRole();
    Void showDetails() {
        system.out.println("Name = " + name);
        system.out.println("id = " + id);
        }
    void welcome() {
        system.out.println("Welcome to the
                Institution");
    }
}
```

```
class Main{
    public static void main(String[] args){
class Student extends Person{
    void displayRole(){
        System.out.println("I am a student");
        }
    name = "Manu";
    id = 123;
    }
class Teacher extends Person{
    void displayRole(){
        System.out.println("I am a teacher");
        }
    name = "Anoop";
    id = 3483;
    }
class Main{
    public static void main(String[] args){
        Student S=new Student();
        S.welcome();
        S.showDetails();
        S.displayRole();

        Teacher t=new Teacher();
        T.welcome();
        T.showDetails();
        T.displayRole();
```

Q. a) Create an abstract class named Vehicle with:
- Attributes: brand, model, year
- Abstract method: startEngine()
- Method: displayInfo() to print brand, model and year.

b) Create two subclasses:
- Car that implements startEngine() by printing "Car engine started"
- Bike that implements startEngine() by printing "Bike engine started"

c) In the main method
- Create one car and one Bike object
- Assign values to their attributes
- Display their details and start their engines.

→
```
abstract class Vehicle{
    String brand;
    String model;
    int year;
    abstract void startEngine();
    void displayInfo(){
        System.out.println("Brand: "+ brand);
        system.out.println("Model: "+model);
        system.out.println("Year: "+ year);
    }
}
class Car extends Vehicle{
    void startEngine(){
        system.out.println("Bike engine started");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        car brand = "Toyota";
        car model = "Corolla";
        car year = 2020;
class Bike extends Vehicle {
    void startEngine() {
        System.out.println("Bike engine started");
class Main {
    public static void main(String[] args) {
        Car C1 = new Car();
        car.brand = "Toyota";
        car.model = "Corolla";
        car.year = 2020;

        Bike B1 = new Bike();
        bike B1.model = "Yamaha";
        B1.model = "FZ";
        B1.year = 2022

        System.out.println("Car details:");
        car.displayInfo();
        car.startEngine();

        System.out.println("Bike Details");
        bike.displayInfo();
        bike.startEngine();
    }
}
```

21/7 Interfaces in Java

- An interface is a reference type similar to a class. But it is a collection of abstract methods (methods without body).
- Interfaces are used to achieve abstraction and multiple inheritance in java.
- Interfaces can contain abstract methods and variables that are constants and static.
- The keywords we use are interface, implements.

```
eg: Interface Shape{
        double area();
        double perimeter();
    }
    class Circle implements Shape{
        double radius=5;
        double area(){
            double return 3.14 x radius x radius;
        }
        double perimeter(){
            return 2x3.14xradius;
        }
    }
    class Main{
        public static void main(String[] args){
            Circle C=new Circle();
            double area=C.area();
            double perimeter=C.perimeter();
        System.out.println("Area is"+area);
```

```
        System.out.println("Perimeter is" + perimeter);
    }
}
```

Inheritance in Java

- Inheritance in Java is a mechanism where one class inherits the properties and behaviours of another class.
- It allows you to reuse code (avoid rewriting the same logic).
- To create a logical parent child relationship
- To implement run time polymorphism.

Syntax

```
class Parent {
    // fields and methods
}
class child extends Parent {
    //additional fields and methods
}
```

- Child class is known as the sub class or derived class
- Parent class is known as the super class or base class.
- Keyword used : extends
- Using the object of child class/sub class, it is possible to access the parent class properties and behaviour in any other classes.

Types of Inheritance

1. Single Inheritance
2. Multilevel "
3. Hierarchial inheritance
4. Multiple Inheritance

- Java doesnot supports multiple inheritance but is possible through interfaces

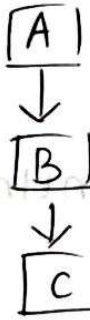
1. Single Inheritance

A parent

B child

eg: class parent{
    // Properties and behaviours
}
class child extends parent(){
    //all properties & behaviours of parent
    and additional properties and behaviours.
}

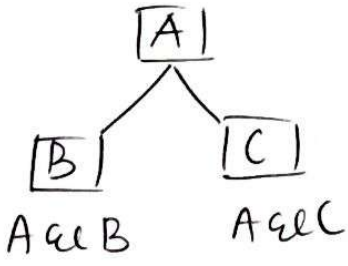
2. Multilevel inheritance



B all properties of A + additional

C all properties of A & B + additional.

3. Hierarchial inheritance



Eg: Single Inheritance:

```
class Animal{
    void eat{
        system.out.println("Eating");
    }
}
```

```
class Dog extends Animal {
    void make sound {
        system.out.println("Barking");
    }
}
```

Object declaration

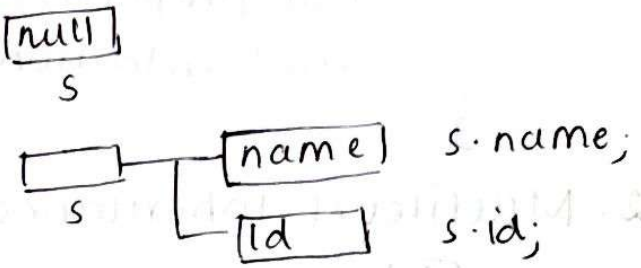
```
int a;
a = 10;
```

```
int a = 10;
```

```
student s; // reference
s = new student(); // allocation
```

```
student s = new
            student();
```

```
class student {
    string name;
    int id;
}
```



Two types of methods

a) Predefined methods eg: system.out.println

b) user defined methods

- Static and non static

⇒
```
class MathOp {
    int add(int a, int b)
    {
      return a+b;
    }
}
class Main {
    psvm (string[] args) {
        MathOp m = new MathOp();
        int sum = m.add(2,5);
        system.out.println(sum);
```

```
    }
}
```

Q. Create a class myClass. Initialise an integer value in it. In the main method access the attribute.

→
```
class myClass{
    int x=10
    public static void main(string[] args)
    {
    Myclass myobject=new Myclass();
    myobject.x=20;
    system.out.println(myobject.x);
    }
}
```
o/p : 20

To avoid the ability to overwrite the existing value, we use 'final' keyword.

```
final int x=10;
```

Static Method → no object creation

```
class Myclass{
    static void myStaticMethod(){
        sop("static methods can be accessed
            without creating an object");
    }
    public void myPublicMethod(){
        sop("can be accessed only by creating object");
    }
    psvm(string[] args){
        myStaticMethod();
```

```
Myclass.myobject= new myclass(); } object is
myobject.mypublicMethod();          created
}
}
```

25/7

Constructors

- same as a method and its name is same as its class.
- it is used to create objects.
- when objects are created, constructor is called.

Consider a class 'student'

```
class student
  {
    student(){
      }
  }
class Main{
    psvm (string[] args){
        student s=new student();
    }
}
```

- Constructors have no return type (not even void)

2 types : a) Default constructors
b) Parametrised constructor

```
class Bike{
  Bike(){
      SOP ("Bike is created");
  }
  psvm (string[] args){
      Bike b=new Bike();
  }
}
```

- Default constructors are used to provide default value to the object like 0, NULL etc.

```
class myclass{
    int x;
    myclass(int y) ⟶ parameterised constructor
    {
        x=y
    }
    p.s.v.m(string[] args)
    {
        myClass myobject=new myClass(5);
        s.o.p("myObject.x");
    }
}
```

* Difference blw constructors and methods.

'This' Keyword

```
public class MyClass{
    String name;
    int id;
    MyClass(string name, int id)
    {
        this.name = name; // current object name
        this.id = id;
    }
    p.s.v.m(string args[]){
        MyClass m=new MyClass("Arun", 123);
        s.o.p("name="+m.name+", id="+m.id);
    }
}
```

Inner class in Java (enhances encapsulation)

→ A class within a class

```
class OuterClass {
    int x = 10;
    class InnerClass {
        int y=5;
    }
}
p.s.v.m (string args[]) {
    OuterClass out = new OuterClass();
    OuterClass.InnerClass in = out.new InnerClass()
    S.O.P ("Outer class value x = " + out.x);
    S.O.P ("Inner class value y = " + in.y);
}
```

eg: Add two numbers : one integer pair and one double pair value.

→

```
class Add {
    int a, b, c;
    double d, e, f;
    void addinteger() {
        c = a+b;
    }
    void adddouble() {
        f = d+e;
    }
    p.s.v.m (string[] args) {
```

```
class Polymorphism{
    void addition(int a, int b){
        int c=a+b;
        system.out.println(c);
    }
    Void addition(double a, double b){
        double c=a+b;
        system.out.println(c);
    }
}
```

Polymorphism

- Has the ability of a menage to be displayed in more than one form.

2 types

1. Compile time polymorphism (method overloading)
   (same name for method but parameters different)
2. Run time polymorphism (method overriding)
   (same name for method and parameters but diff. class)

Q. Write a java program that calculate the area of diff. shape using method overloading. Class name is area calculator and shapes are square, rectangle, circle.

→
```
class AreaCalculator{
    void area(int a){
        int area = a*a;
        S.O.P("Area of square = " + area);
    }
    void area(int a, int b){
        int area = a*b;
        S.O.P("Area of rectangle =" + area);
    }
```

```
    void area(float a){
        int area = 3.14 * a * a;
        S.O.P("Area of circle = " + area);
    }
}
```

28/7 Access Specifiers / Access Modifiers

4 types → 1. Private 3. Protected
2. Default 4. Public

1) Private

Access level of a private modifier is only within the class. It cannot be accessed from outside the class.

```
class student{
    private String name = "ABC";
}
class Main(){
    p.s.v.m (String[] args){
        student s = new student();
        S.O.P (s.name) // error
    }
}
```

2) Default

only within package. It cannot be accessed from outside the package. If you donot specify any access level, it will be default.

3) Protected

Access level of a protected modifier is within the package and outside the package through child class. It cannot be accessed from

outside the package.

4) Public

Access modifier of public modifier is everywhere. It can be accessed from within the class, outside class, within package and outside package.

Q. write a java program to add two numbers. Numbers should be taken as user input.

→
```
import java.util.Scanner;
public class addition{
    p.s.v.m(string[] args){
        Scanner sc=new Scanner(System.in);
        S.O.P("Enter first number:");
        int a = sc.nextInt();
        S.O.P("Enter second number");
        int b = sc.nextInt();
        int c = a+b;
        S.O.P(c);
    }
}
```

Eg:
```
class Employee{
    int basicPay = 30,000;
}
class ProgramDeveloper extends Employee{
    int bonus=5000;
    Public void role(){
        S.O.P("I'm developer");
    }
}
```

```
class ProgramTester extends Employee {
        int bonus = 3000;
        public void role() {
            S.O.P ("I'm a tester");
        }
}
Public static void main (string[] args) {
```

30/7 'Super' keyword

```
clas Animal {
    String color = "Black";
}
class Dog extends Animal {
    String color = "White";
    void printData() {
        S.O.P (color);
        S.O.P (super.color);
    }
}
clas Main {
    p.s.v.m (string[] args) {
        Dog d = new Dog();
        d.printData();
    }
}
```

o/p : White
Black

→ Using "super" keyword helps in accessing parent class. Used to refer immediate parent class.
→ instance variable, methods, constructors.

For method

```
class Animal {
    void eat(){
        s.o.p("eating");
    }
}
class Dog extends Animal{
    void eat(){
        s.o.p("Dog is eating bread");
    }
    void work(){
        super.eat();
    }
}
class Main{
    p.s.v.m(string[] args){
        Dog d=new Dog();
        d.work();
    }
}
```

o/p : eating

For constructor

```
class Animal{
    void Animal(){
        s.o.p("Animal is created");
    }
}
```

```
class Dog extends Animal {
        Dog(){
            super();
            S.O.P ("Dog is created");
        }
}
class Main {
      p.s.v.m(String[] args){
            Dog d=new Dog();
            d.Dog();
      }
}
```

o/p : Animal is created
Dog is created.

Calling Order of a Constructor

```
class A {
      A() {
         S.O.P ("A is calling");
}}
class B extends A{
      B() {
         S.O.P("B is calling");
}}
class C extends B{
      C(){
         S.O.P("C is calling");
}}
class Main{
      p.s.v.m (String[] args) {
            C c=new C();
}}
```

o/p : A is calling
B is calling
C is calling

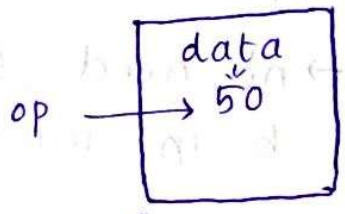
Calling order of a constructor in inheritance.

- The order of execution of constructor in inheritance relationship is from base class to derived class.
- When we create an object of a class, then the constructor get called automatically.
- In inheritance relationship, when we create an object of a child class then first base class constructor and then derived class constructor will call implicitly.

31/7 Using object as a parameter

```
class Operations{
    int data=50;
void change(Operations op){
    op.data = op.data+100;
}
p.s.v.m(string[] args){
    Operations op=new Operations();
    s.op(change "value of data before operation="
                    + op.data);
    op.change(op);
    s.o.p("value of data after operation ="
                    + op.data);
}
}
```

- Type of op is 'Operations'



o/p : 50
150

* All members of a class can be static but a class itself cannot be static.

Q.

```
class outer {
        int a = 10;
static class inner{
                public static
                int b = 20;
 }}
 class Main{
      p.s.v.m (string[] args){
          Outer out=new outer();
        //Outer.Inner in=out.new Inner();
          Outer.Inner in=new Outer.Inner;
      }
 }
```

→ no need of using outer class object if since b in inner class is static.

Q. ⓐ
```
class staticDemo{
      static int x =10;
      sout
      p.s.v.m (string[] args) {
            s.o.p (x);
      }
}
```

ⓑ
```
class staticDemo{
      static int x=10;
}
```

```
class Main{
    p.s.v.m(string args[]){
        s.o.p(staticDemo.x);
}}
```

Returning an Object

Eg:-
```
class Book{
    String title;
    String author;
    Book (String title, String author){
        this.title = title;
        this.author = author;
}}
class MyClass{
Book createBook(){
    return new Book ("WOF", "APJ");
}
class Meyclass{
    p.s.v.m(string[] args){
        Book b=new Book().createBook();
}}
```

1/8

Method Overwriting

- A subclass provides its own implementation of a method defined in its superclass.
- The method signature must match.

Dynamid method dispatch

- when you call a overwritten method via that reference, JVM determines which implementation to execute based on the actual object time is, at run time not reference time.

'Final' keyword

- Final keyword is used to restrict the user.
- You can declare a variable as final, a class as final and a method as final.
- Final with keyword : cannot be overwritten
  " " method : cannot be extended (child class)

Recursion

Sum of 10 numbers

```
~~import java.util.Scanner;~~
public class Sum{
    int count=10;
    int sumRecurse(int count){
        if (count==0)
            return 0;
        else
            ~~return~~ sum=sum+x;
            return sum+sumRecurse(x--);
}}
class Main{
    p.s.v.m(s.A){
        ~~Add~~ Sum a=new ~~A~~Sum();
        a.sumRecurse(10);
        s.o.p(a.sum);
}}
```