Static methods:-

→ This methods can read and act upon static variables.
→ Static methods cannot read and act upon instance variables.
→ Static variable is a variable whose single copy is shared by all other objects.
→ From any objects if static variable is modified it effects all the objects.

person and company details:-

```
Class person {
    String name;
    String permanentAddress;
    int age;
    void set_permanentDetails(String name,
        String permanentAddress, int age){
    this.name=name;
    this permanentAddress=permanentAddress
    this.age=age;
    }
    void get_permanentDetails(){
        S.o.p("name:"+name);
        S.o.p("permanent Address:"+permanent
                                    Address);
        S.o.p("Age:"+age);
    }}
Class Employ extends person {
    int id;
    String CompanyName;
    String CompanyAddress;
    Employ(int id, String name, String permanent
                                    Address, int age,
```

```
Sbing CompanyName, sbing CompanyAddress)
{
this.id = id;
set-permanentDetails(name, PermanentAddress, age);
this.CompanyName = CompanyName;
this. CompanyAddress = CompanyAddress;
}
void get_EmployDetails() {
    S.O.P("Employ Id :" + id);
    get-Permanet Details();
    S.O.P("company Name :" + CompanyName);
    S.O.P("Company address :" + CompanyAddress);
}
}
Class InnerDemo {
    Public static void main (String args[]) {
    Employ e1 = new Employ (101, "suresh kumar"
            "118 - Madhura - Nagara - Tirupathi", 24
            "Cente", "20-RVS nagar");
        e1.get_Employee Details();
    }
}
```

When the programmer does not initialize the instance variables, java compiler will write code and initializes the variables with default values.

Int - 0, Float - 0.0, Double - 0.0
Char - space, String - null
Boolean - False.

Initializing instance Variables:-

We can initialize instance variables directly in the class using assignment operator. In this type every object is initialized with the same data.

```
int rollNo = 101;
String name = "Kiran";
Class student {
    int rollNo=101;
    String name = "soriya"
    void display() {
        System.out.println("student Roll number is:
            rollNo);
        S.o.P("student name is:"+name); }}
Class studentDemo {
    public static void main(String[] args) {
        Student s1 = new student();
        S.o.P("First student detail"); s1.display
```

```
Student s2 = new Student();
s.o.p("second student details:");
s2.display();
}
}
```

Q) We can initialize one class instance variables in another class using reference variables.

```
class Student {
    int rollNo;
    String name;
    void display() {
        s.o.println("student roll no is" + rollNo);
        s.o.println("student name is :" + name);
    }
}
class StudentDemo {
    public static void main(String args[]) {
        Student s1 = new Student();
        s.o.print("First details:");
        s1.rollNo = 101;
        s1.name = "Suresh";
        s1.display()
        Student s2 = new Student();
        s.o.p("second student details");
        s2.rollNo = 102;
        s2.Name = "Ramesh";
        s2.display(); }}
```

Initialize student details using parameterized constructor:

```
class student {
  int rollNo;
  String name;
  Student (int r, String n) {
    rollNo = r;
    name = n;
  }
  void display() {
    S.O.P("student Roll number is:" + rollNo);
    S.O.P("student name is:" + name);
  }
}

class studentDemo {
  public static void main (String[] args) {
    Student S1 = new student (101, "suresh");
    S.O.P("S1 object contains");
    S1.display();
    Student S2 = new student (102, "ramesh");
    S.O.P("S2 object contains");
    S2.display();
  }
}
```

→ We can use this keyword we can use this inside any method for referring the current object.

→ Generally memory is allocated to objects by using new operator and deleting an allocated memory is uncommon. This deletion of memory is supported by delete operator in C++ but this deletion of allocated memory works automatically in Java.

→ This automatic deletion of already allocated but unused memory is called a garbage collection.

→ This operation of garbage collection is accompolished by method named gc().
→ This method is used for garbage collection.

→ It is an important feature which provides security for class members.

→ public, private, protected and default.

Variable:- access-specifier data-type variable-name =data;

method:- access-specifier return-type method-name (parameters)

```
Class A {
Public int x=10;
Private string y="Pupa";
Protected int z=5;
boolean b=true;
public void m1() {
Private int x=30;
}
}
```

public:- we can access anywhere, inside the class, outside class or any other package.

default specifier.

→ we should not apply access specifier to local variables.

```
sum=0
sum of digits
n=121
remainder=121%10
sum=sum+digits
```

default → we can access only inside that package

Protected:- can access in that entire package, can access in the sub class of other package.

Method overriding:-

```
Class M {
public void m1() {
S.o.p("Hi");
}
}
Class N extends M {
public void m1() {
S.o.p("Hello");
}
}
```

→ method overriding

abc→package

```
Class A
Protected int x=10
Class
```

↓ Here we can use

xyz another package

```
import abc.*;
Class C extends A
```

↓ Here also we can't able to access

→ protected

→ In method overriding if we change access specifier then also it will work. with one condition:-

in method overriding child class visibility (private) should be more than parent class then method overriding is acceptable whereas vice versa is not acceptable. ↓ is nothing but public

Members of Class:-

Anything created inside the class is called member of class

Types → Instance members → Instance variables
                         → Instance methods
      → Static members → static variables
                       → static methods

Instance methods: - we can access instance variables, static variables and can call after static method.

Static methods: - we can access only static variables & static methods

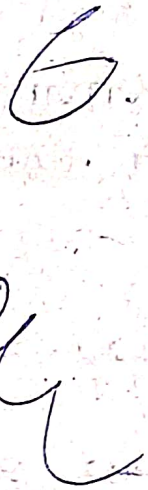
```
class student {
    public string name;
    public int age;
    public static string institute = "palle";
    public student (string name, int age) {
        this.name = name
        this.age = age; }
    public void display() {
        S.O.println(name + age + institute);
    }
    public static void display() {
        S.O.P(name + age + institute);
        S.O.P(institute) → error.
    public static void m1() {
        display(); → error.
        display2(); ✓ → static method
    }
}
```

How to access instance members & static members outside the class?

Instance members:- can be access with the object reference variable

Static:- can be access using class name or obj reference variable.

```
class Student{
  public String name="Bala";
  public int age=21;
  public String course="Java";
public static String institute="palle";
  public void display(){
    System.out.println(name+age
        +course);}
  public void display2(){
    S.O.println(institute);}}
public class main(){
  public static void main(String[] args){
    Student s= new student();
    S.O.println(s.name);
    S.O.println(Student.name)
        // Error because instance
variables only use obj not
    reference to access
      S.O.println(Student.institute);
      S.O.println(s.institute);
    }  // Here we can use
    either static classname
  or obj reference to
  access static variable
    Bala
    palle
    palle
```



Instance members

Create without static keyword inside class

access instance variables, instance methods, static methods, static variable.

access outside class by creating object and using object reference variable

loaded in memory after creating object

static members

Create with static keyword inside class.

access only static variables, static methods

access outside the class by using class name

loaded in memory after loading class

can't access using Classname	can't access using obj ref variable.
stored in heap memory.	It will be stored in data segment (static variable)
object creation is Compulsory to access	object creation is not Mandatory access