Class & object & Method

Method:

*) A set of code which perfoms a particular task

Advantages

→ Code Reusability
→ Code optimization.

Syntax:- Access modifier Return type Name (list of args)
{
// body.
}

object

*) It is an instance of class
*) object is Real world entity
*) " occupies Memory.
*) consists of
i) identity (Name)
ii) State/Attribute (color, breed, age)
iii) Behaviour (i.e. Methods e.g. eat, Run)

e.g. Animal → class
Dog, Cat → object.

Creati an object

i) By using new keyword
ii) " " newInstance() Method
iii) " " clone() method.
iv) " " deserialization() Method
v) " " factory method.

*) We can create object by

→ Declaration
→ Instantiation
→ Initialization

1) Declaration (Declaring a Variable name with an object type).

e.g. Animal Dog;

Animal is the class

Dog → variable of type object (Reference variable)

Note:- Simply declaring a reference Variable doesnot create an object.

2) Instantiation (Using new keyword)

```
Animal Dog = new Animal();
```

Animal → class
Dog → object
new → memory
Animal() → constructor

Now through . operator we accesses all methods

e.g.

```
class Animal
{
    public void eat()
    {
        S.o.p("I am eating");
    }
    public static void main(String args[])
    {
        S.o.p("");
        Animal dog = new Animal();
        dog.eat();
    }
}
```

Initialize an object.

i.e. How to put value to the object.

1). By Reference Variable
2) By using Method
3) " " constructer.

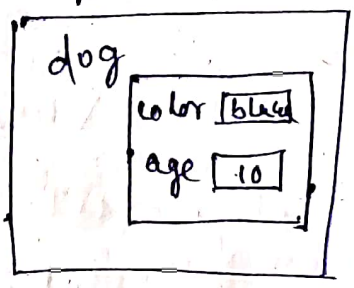
By using Ref. Variable.

e.g.
```
class Animal
{ String color;
  int age;
  p.s.v.m(String args[])
  {
  Animal dog = new Animal();

dog.color = "black";
dog.age = 10;
s.o.p(dog.color + " " +
        dog.age);

}
}
```

memory:



By using method

e.g.
```
class Animal
{ String color;
  int age;
  void display(String c, int a)
  {
    color = c;
    age = a;
  }
  P.s.v.m(String args[])
  { Animal dog = new Animal();
```

```
dog.display("black",
            10);
s.o.p(dog.color
  + " " + dog.age);
```

dog.

Static Variable.

Access Modifiers

→ public
→ Private
→ Protected
→ default

Non Access Modifiers

→ Static
→ final
→ abstract
→ Synchronized
→ volatile

*) Static is a Non Access Modifier.

*) We can use Static Keyword with

i) — Variable ✓ (class level Variable)
X local Variable)

ii) — Method

iii) — block

iv) — inner class (X outer class)
or nested class.

Variable ✓ (static variable belongs to class Not object)

```
class test
{
  static int a=10;
  void m1()
  {
    static int b=20; X //error due to local Variable
  }
  p.s.vm(String args[])
  {
    s.o.p(a);
    s.o.p(test.a);
    test e1 = new test();
    s.o.p(e1.a);
  }
```

X //error.
Not for object

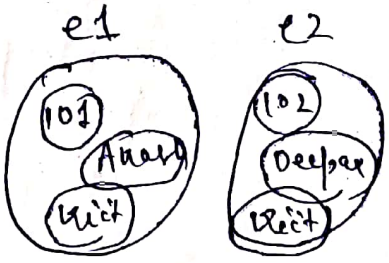
```
class test
{
    static int a=10;
}

class demo
{
    p.s.v.m()
    {
        System.out.println(a);  X
        s.o.p(test.a);          ✓ // (due to 'a' is static in test class).
        test ob= new test();
        s.o.p(ob.a);            X
    }
}
```

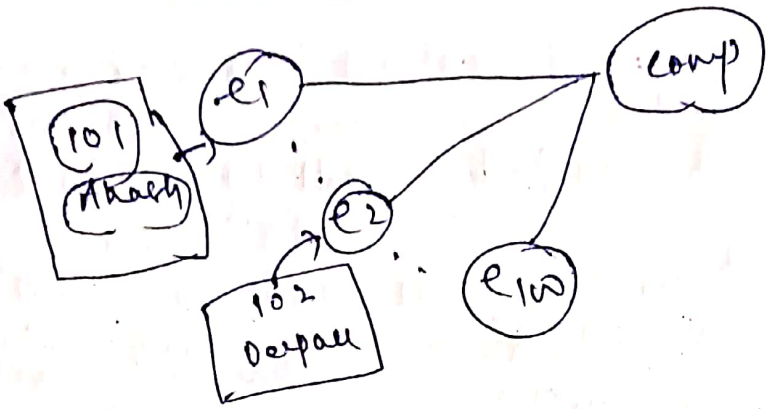
*) Static variable is used for memory management i.e. efficient use of memory or we can save memory.

e.g.

```
class emp
{ int empid;
  String name;
  String comp;
void getdata(int a, String b, String c)
{ empid = a;
  name = b;
  comp = c;
}
void display()
{ s.o.p(empid+" "+name+" "+comp);
}
p.s.v.m( )
{ emp e1 = new emp(101, Akash, Kiit);
  e1.display();
emp e2 = new emp(102, Deepak, Kiit);
  e2.display();
   :
emp e100 = new .....
```

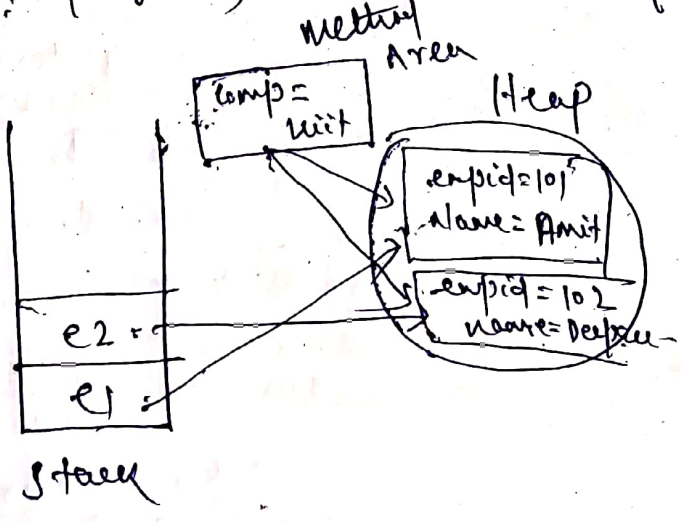


*) Each employee has company name = Kiit. So if we are creating 100 objects then 100 time same memory is created. So if we make 'comp' as static then no need to give its value Repeatedly.



eg.

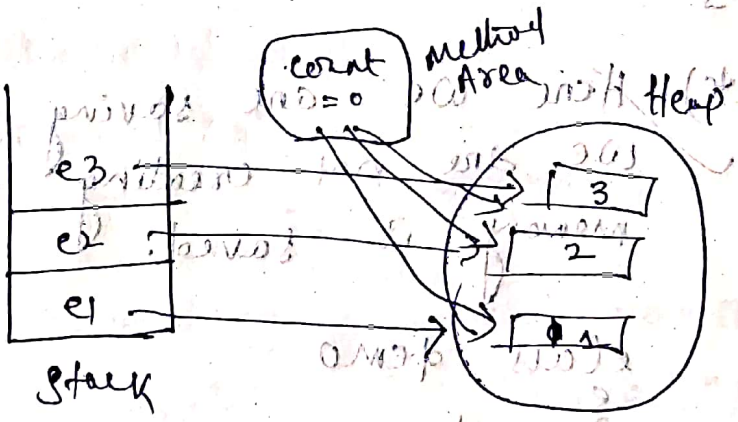
```
class emp
{   int empid;
    String name;
Static String comp = "kiit";
void getdata (int a, String b)
  {  empid = a;
     name = b;
  }
void display ()
  {
 S.o.p(empid + " " + name + " " + comp);
  }
 P.S.V.M()
 {
emp e1 = new emp ();
e1.getdata(101, "Amit");
e1.display();
emp e2 = new emp ();
e2.getdata(102, "Deepak");
e2.display();
 :
 :
}
```



*) Static Variable gets Memory only once in the class area in the time of class loading.

Counter example — Using Static

```
class Counter
{ int count=0;  ------> static int count = 0
  void getdata()
  {
    count++;
    p.o.p(count);
  }
p.s.v.m()
{ Counter c1 = new Counter();   o/p        o/p
    c1.getdata();  ---- 1 ------> 1
  Counter c2=new Counter();
    c2.getdata();  ---- 1 ------> 2
  Counter c3 = new Counter();
    c3.getdata();  ---> 1 ------> 3
}
```



Static Methods

1. *) Static Method belongs to class Not object.
2. *) We can call the method through class name not through object.

classname . method name ;

e.g.

```
class staticmethod
{
    void disp()
    {
    s.o.p ("1");
    }
p.s.v.m ( )
{
staticmethod e1 = new staticmethod ();
e1. display();
}
}
```

Static void disp()
{
s.o.p ("1");
}

staticmethod e1.display();

*) Here we are saving memory. we are not creating object so memory is saved.

```
class demo
{
static void disp( )
    { s.o.p("1");
    }
p.s.v.m ( )
    { disp(); -OR demo.disp();
      xyz . show();
    }
class xyz
    { static void show()
        { s.o.p ("2");
        }
```

③ *) Static method can access only static data. It cannot access nonstatic data (instance data).

e.g.

```
class demo
{
    int i=10;        ⟶ static int i=10
    static void disp()
    {
        s.o.p("i");   X   ✓
    }
    p.s.v.m()
    {
        disp();
    }
}
```

4) *) Static method can call only other static methods & cannot call a non-static method.

e.g.

```
class demo
{
    static void disp()
    {
        s.o.p("1");
        show();
    }
    void show()  ⟶  static void show()
    {
        s.o.p("2");     s.o.p("2")
    }
}
```