]

.1.2020

this() as a constructor :

→ Used to invoke current class constructor

→ Reuse constructor

→ Constructor chaining

- method of calling one constructor with the help of another while considering present object.

```
class a
{
    a()
    {
        System.out.println("Hello");
    }
    a(int x)
    {
        this();
        System.out.println(x);
    }
}
class testthis
{
    public static void main(String a[])
    {
        a a1 = new a(10);
    }
}
```

⊛

// Whenever you are going to use this() to invoke a constructor, it has to be the first statement in the method (or) constructor.

(Eg) 2

```
temp(int x, int y)
{
   this(5);                          // calls temp(int x)
   System.out.println(x+y);          Also called as constructor reuse
}
temp(int x)
{
   this();                           // calls temp()
   System.out.println(x);
}
temp()
{
   System.out.println("Default");
}
```

this to pass as an argument in method:

```
class c2
{
   void m(c2 obj)
   {
      System.out.println("Method");
   }
   void p()
   {
      m(this);                       // Here, this is replaced with object c.
   }
   public static void main(String a[])
   {
      c2 c = new c2();
      c.p();
   }
}
```

Constructor reuse :

```
class student
{
   int rollno ;
   String name, course ;
   float fee ;
   Student ( int rollno, String name, String course )
   {
      this. rollno = rollno ;
      this. name = name ;
      this. course = course ;
   }
   student ( int rollno, String name, String course, float fee )
   {
      this ( rollno, name, course) ;
      this. fee = fee ;
   }
   void display ()
   {
      System. out. println ( rollno" "+name+" "+course+" "+fee ) ;
   }
}
class testthis
{
   public static void main ( String a[] )
   {
      student s1 = new student ( 1, "a", "WT" ) ;
      student s2 = new student ( 2, "b", "Java", 6000f ) ;
      s1. display ( ) ;
      s2. display ( ) ;
   }
}
```

Array of objects :

Arrays are capable of storing objects.

```
class student
{
    int marks;

    public static void main (String a[])
    {
        student std[] = new student[3];
        for (i=0; i<std.length; i++)
        {
            std[i] = new student();
        }
        std[0].marks = 40;
        std[1].marks = 50;
        std[2].marks = 60;
        S.O.P (std[0].marks);
        S.O.P. (std[1].marks);
        S.O.P. (std[2].marks);
    }
}
```

// Here it is just a reference variable. This statement shows it is an array holding 3 values. It is not yet instantiated as an object.

// Accessing class variables

// std[i] = new student(); Here, instantiation takes place.

static keyword :

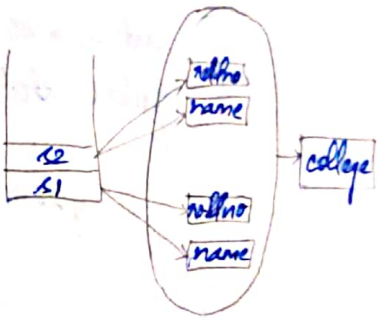
static variable :

Advantages : Memory efficient

Purpose of using static keyword :

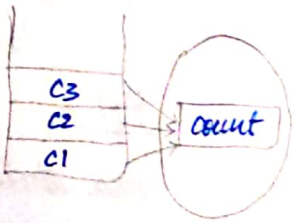
- It is going to be common for the entire class.
- Not separate for each object.
- Since it is going to be common, you can initialise it.

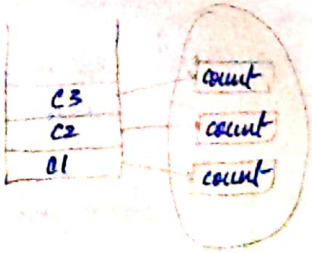
```
class student
{
   int rollno ;
   String name ;
   static String college = "MIT" ;
   student ( int r, String n )
   {
      rollno = r ;
      name = n ;
   }
   void display ( )
   {
      System.out.println (rollno+" "+name+" "+college ) ;
   }
   public static void main ( String a[ ] )
   {
      student s1 = new student (1, "a") ;
      student s2 = new student (2, "b") ;
      s1.display ( ) ;
      s2.display ( ) ;
   }
}
```



Eg.) 2

```
class counter
{
   int count = 0 ;     // If it had been static :
   counter ( )
   {
      count++ ;
      System.out.println ( count) ;
   }
   public static void main ( String a[ ] )
   {
      counter c1 = new counter ( ) ;
      counter c2 = new counter ( ) ;
      counter c3 = new counter ( ) ;
   }
}
```





static method :

→ Belongs to class
→ Can be invoked w.r.t. object
→ Can access static data member and change value

Restrictions :

→ Cannot use non-static data member (or) call non-static method directly
→ this and super cannot be used

```
class A
{
   int a1 = 40;
   public static void main(String a[])
   {
      System.out.println(a1);   // Compiler error
   }
}
```

If a1 had been static, this would be correct.

(All instance variables need to be accessed through objects)

(Eg) 2

```
class student
{
    int rollno;
    String name;
    static String college = "MIT";
    student(int r, String n)
    {  rollno = r;
       name = n;
    }
    static void change()
    {
       college = "CEG";
    }
```

```
    void display()
    {
      System.out.println(rollno+" "+name+" "+college);
    }
    public static void main(String a[])
    {
      student s1 = new student(1,"a");
      student s2 = new student(2,"b");
      s1.display(); s1.change();
      s2.display();
    }
}
```

(Eg) 2

```
class cal
{
  static int cube(int x)
  {
    return x*x*x;
  }
  public static void main(String a[])
  {
    int n = cube(5);          // Since it is a static method, this is correct.
    System.out.println(n);
  }
}
```

static block :

→ Used to initialize static data member
→ Executed before main method

Try in Lab.

```
class a2
{
  static
  {
    System.out.println("static block");
  }
  public static void main(String a[])
  {
    System.out.println("Hello");
  }
}
```

// Can we give only static block and no main ?

Till version 1.7, it worked. But now, main() block missing error occurs.