Types of Variable

Division - I

Based on type of value represented by a variable All variables are divided into two types:

(1) Primitive variable: Can be used to represent primitive values

```
int x = 10;
```

(2) Reference variables: Can be used to refer objects

```
Student s = new Student();
```

s pointing object

~~Part II~~

Division - II

(1) Instance Variable
(2) Static Variable
(3) Local Variable

⇒ Based on position of ~~Variable~~ declaration and Behaviour all Variable are divided into 3 type:

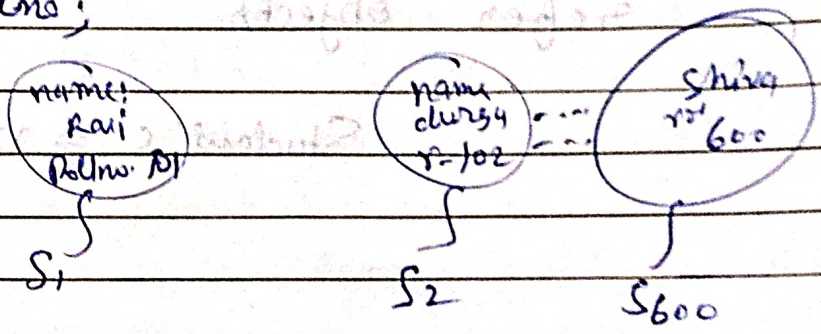
Date

Instance Variable:

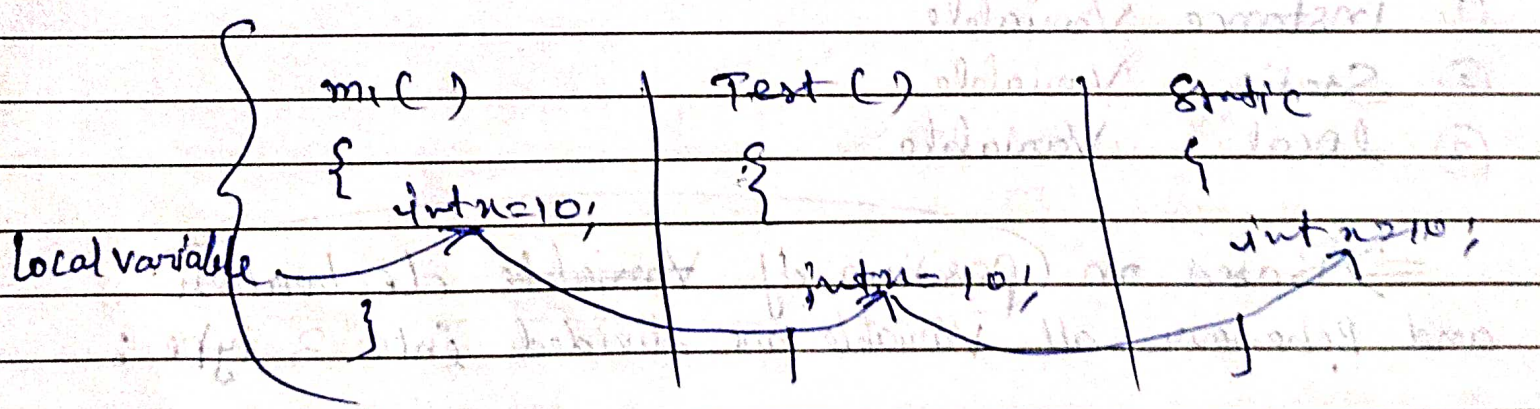
I. If the value of variable is varied from object to object such type of variables are called "instance variable".

II. For every object a seperate copy of Instance variable will be created.

```
Class Student
{
    String name;
    int rollno;
}
```



⇒ Instance variables should be declared within the class directly but outside of any method/block or constructor;



III. Instance variable will be created at the time of object creation and destroyed at the time of object destruction,

Hence the scope of instance variable is exactly same as scope of object

Q. Where object will get Stored Date

A. Heap memory

Q. Where instance Variable will get stored

A. Heap memory

★ instance Variable is Saved/stored in Heap memory as a part of object.

```
Class Test
{ int x=10;
  Public static void main(String[] arg) {
      SOP(x); //Compile Error
  }
}
```

~~Eg: Variable of CE~~

~~If~~ main method main() is Static method Static method means no where related to object but instance Variable is always part of object. And without having an object Main method main() Can never be executed.

[without having an object instance Variable there is no chance.]

C.E → Non Static Variable x Cannot be referenced from a static context

Conclusion ~~#~~ WE CAN'T ACCESS INSTANCE VARIABLE DIRECTLY FROM STATIC AREA,

BUT WE CAN ACCESS BY USING OBJECT REFERE

BUT WE CAN ACCESS ~~VARI~~ INSTANCE VARIABLE DIRECTLY FROM INSTANCE AREA.

Date

```
class Test
{
    int x=10;
    p s v main (String[] arg)
    {
        // Sopln (x); // CE
        Test t = new Test();
        Sopln (t.x); //10 out put
    }
    public Void m1()
    {
        Sopln (x);
    }
}
```

Non static Variable x Cannot be referenced from a Static Context

using instance variable

need to use of creation of object

```
class A
{
    public int m1 (int x, int y)
    {
        return (x+y);
    }
}
```

Here

This is instance method because Static keyword missing

NOT using any instance Variable

No need to declare m1() as instance method

(Static keyword ka upyog karke) main method ko class level par define karne se direct class name se main method ko access kar sakte hain bina kisi object ke.

Spiral

Eg:

```
class Test
{
    int x;
    double d;
    boolean b;
    String s;
    pe v main (String[] args)
    {
        Test T1 = new Test();
        Sopln(T1.x);    // 0
        Sopln(T1.d);    // 0.0
        Sopln(T1.b);    // false
        Sopln(T1.s);    // Null
    }
}
```

All default Value provided by JVM

Static methods can call and direct access instance method but non static method need object creation to directly access

for instance Variable J.V.M is always provide default Value and we are not required initialization explicitly.

Instance Variable is also known as object level Variables or attributes

// Yahan ek Java program diya gaya hai, jisme main() ko static keyword ke saath class level par define kiya gaya hai

```
Public class Greetings {
    Public static void main (String[] args)
        System.out.println ("Hellow orld in Greetings Class");
        greet();
    }
    Public static void greet(){
        System.out.println ("Greetings from greet()");
    }
}
```

Spiral

Static Variables.

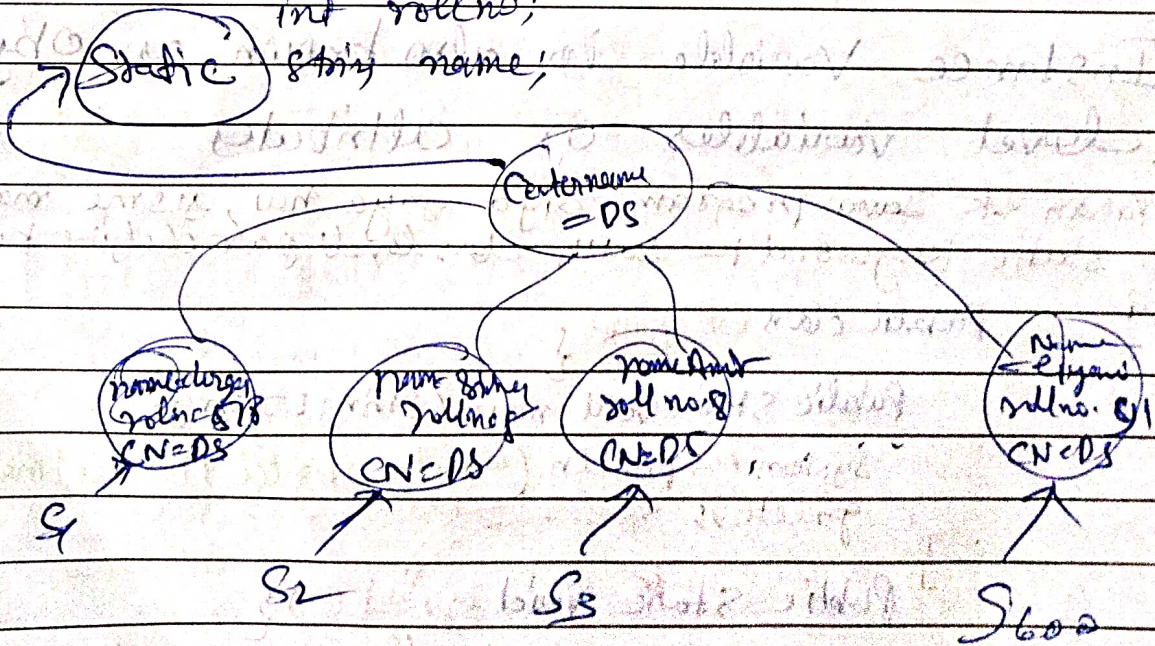
Date

1. If the value of a variable is not varied from object to object then it is not recomended to declare variable as Instance variable.

   We have to declare such type of variable at class level by using "static" modifier.

2. In the case of instance variables for every object a seperate copy will be created but in the case of static variable, a single copy will be created at class level and shared by every object of the class

```
class Student
    String name
    int rollno;
    String name;
```



Date

Q. Where you have to declare Static Variable ?

Ans Static Variable should bee declare within the class directly. (and not in the block, method or Constructor.)

Q. When Static Variable will be Created or destroyed and what is the Scope of Static Variable ?

Ans At the time of class loading - Creating
" " " class unloading - destroying

Hence, Scope of Static Variable is exactly same Scope of ~~obje~~ .Class file

Q. ~~Creation~~ & ~~destruction~~ of Static Variable.

Java Test

① Start JVM
② Created & Start Main thread
③ Located Test. Class file
④ Load Test. Class ← Static Variable Creation
⑤ Execute main( ) method
⑥ Unload Test. Class ← Static Variable destruction
⑦ Terminate main Thread
⑧ Shut down JVM

Static Variable will be stored in method area.

Shiral

Class Test

{ static int x=10;

psv main (String[] args)

{ Test t = new Test();

Sop(t.x); // Valid

Sop(Test.x); // Valid.

Sop(x);

3. Static Variable can be accessed either by using object reference or by class name, but recommended to use class name because (by using object reference we usually call instance variable but on static Variable) Unnecessary readibility of code is there

However recommend to use
classname.static variable
Sop(Test.x);

But we can also access directly
Sop(x);

As x is Static Variable and no Object reference i/p or class reference i/p required within the class same access directly and not required to use class name.

Spiral

4) We can access static Variables directly from both instance and static areas

```
class Test
{
    static int x=10;
    psv main(String[] args)
    {
        Soln(x);
    }
    public void m1()
    {
        Soln(x);
    }
}
```

Static area ←

Instance area ←

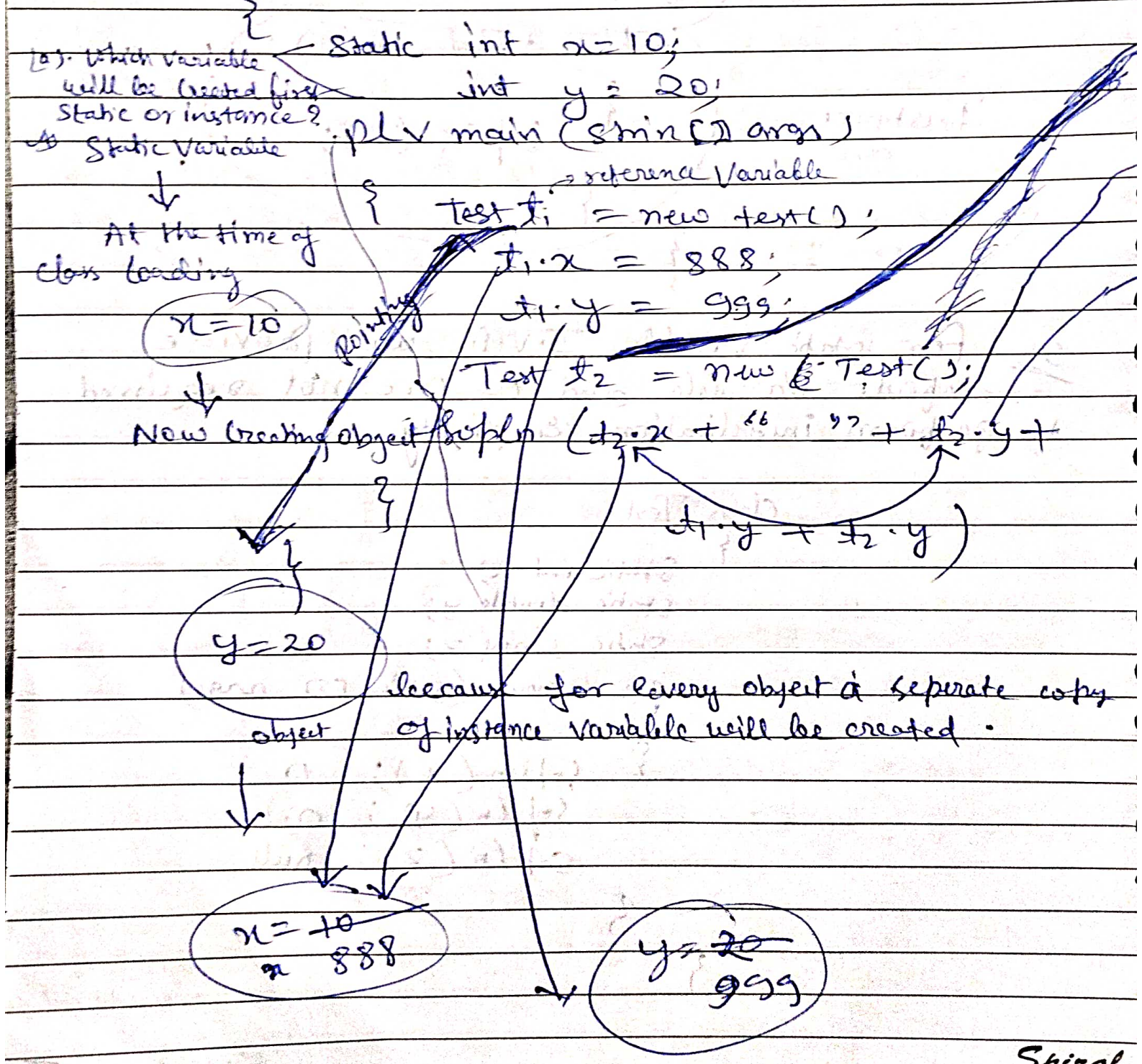
5) For static Variable J.V.M will provide default Variable and we are not required to perform initialisation explicitly

```
class Test
{
    static int x;
    static double y;
    static String z;
    psv main(String[] args)
    {
        Soln(x);   0
        Soln(y);   0.0
        Soln(z);   null
    }
}
```

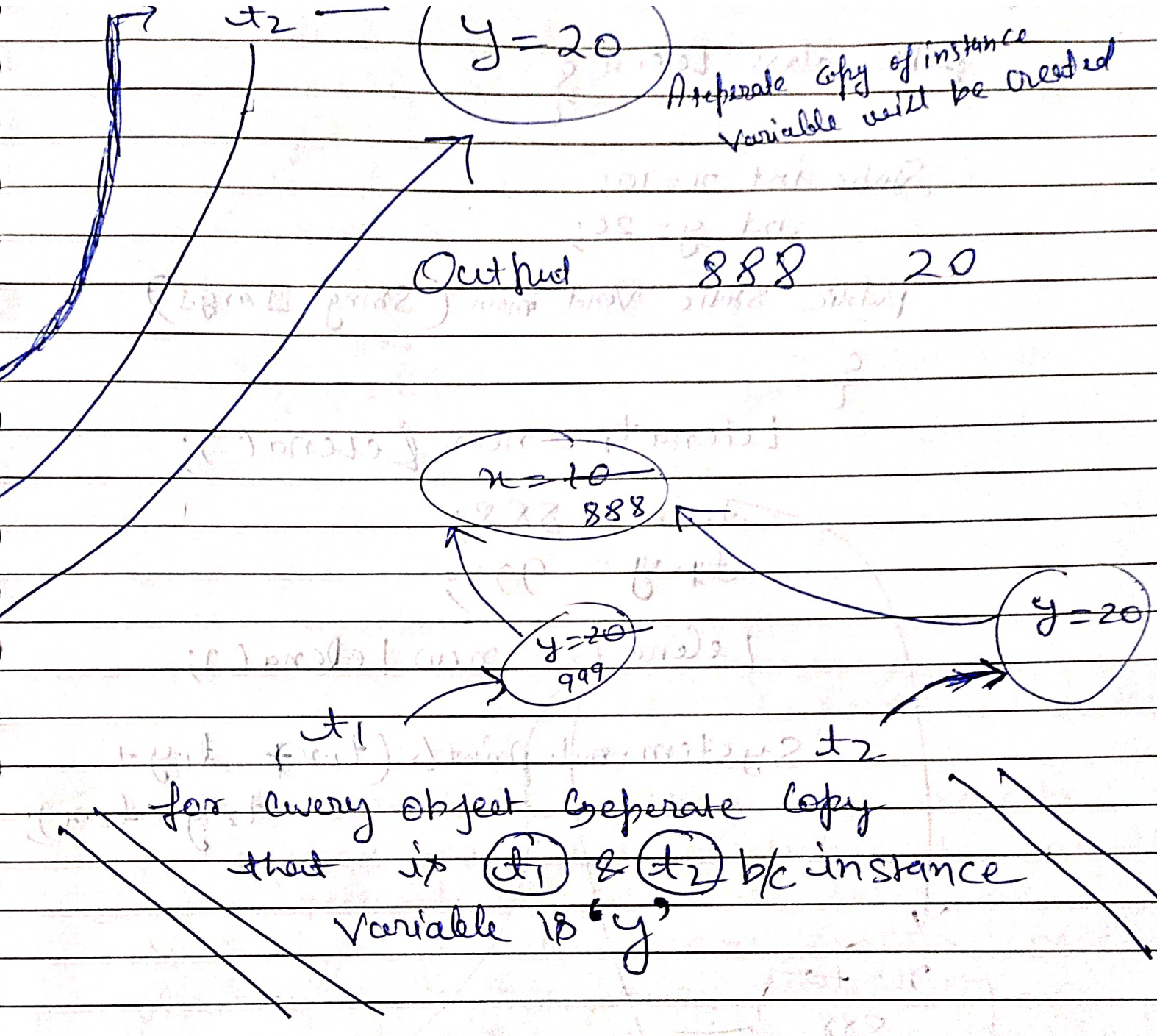
Date

6. Static Variables are also known as "class level Variable" or "fields"

7. —: Very important · Example :—



Spiral



for every object seperate copy that is t_1 & t_2 b/c instance variable is 'y'

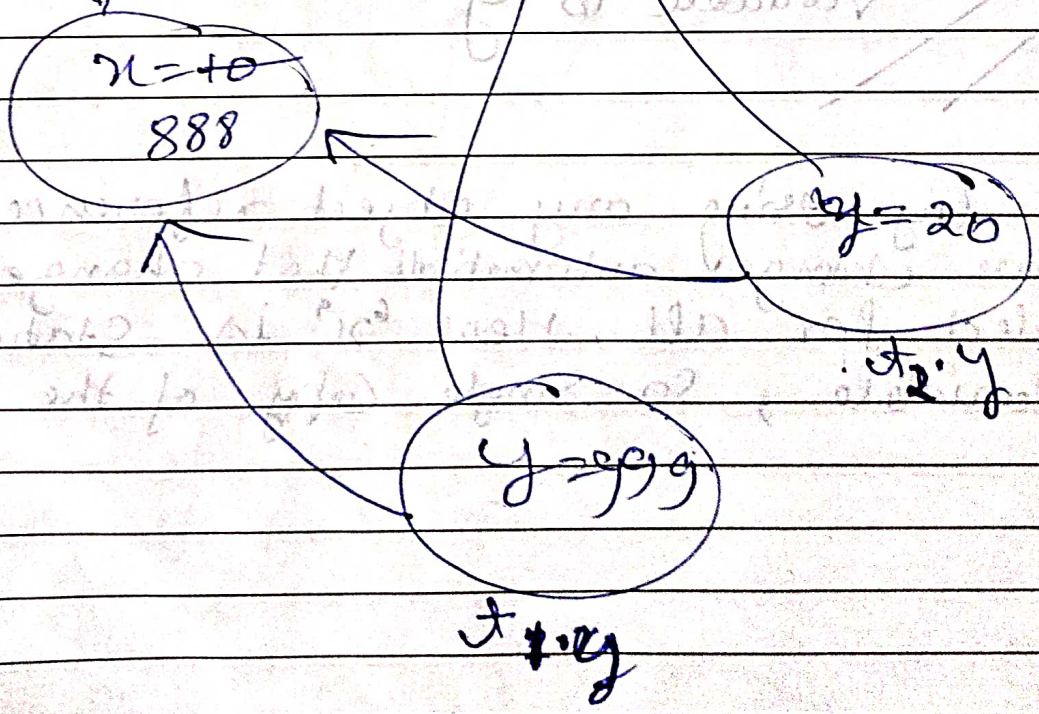
By using any object reference if we change automatically that change will reflect for all. Here 'x' is static variable, so single copy of the object

Date

Example

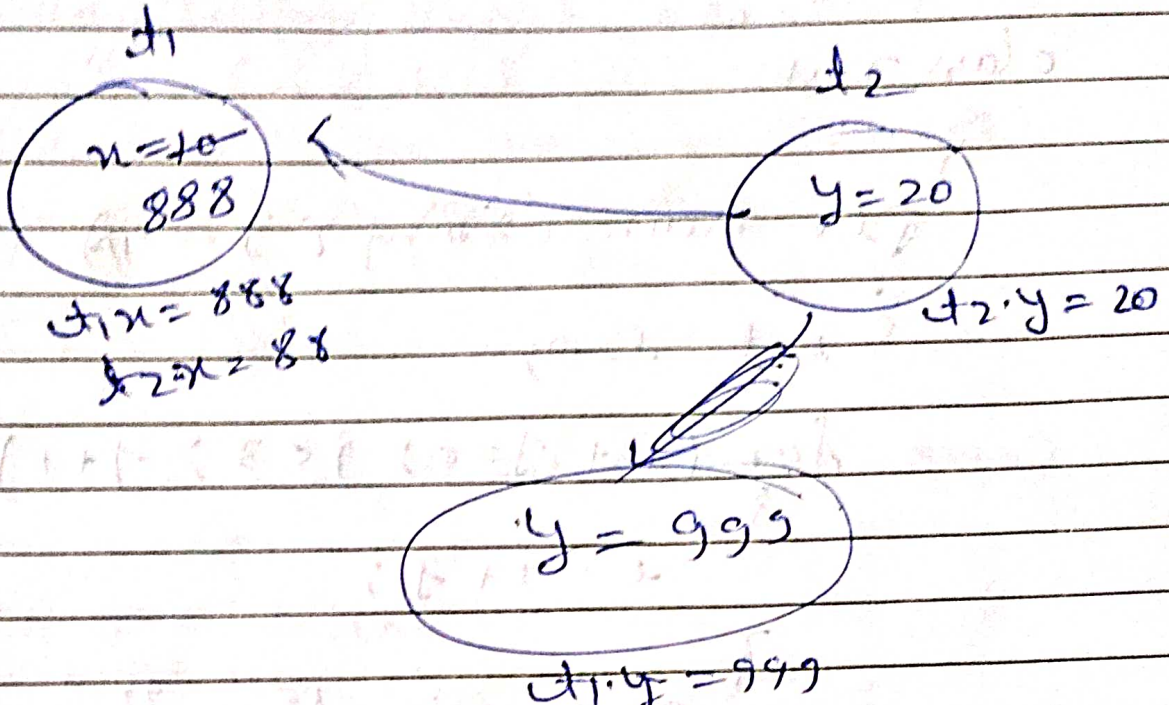
```
Public class Lelena {

Static int x=10;
    int y=20;
Public Static Void main(String[] args)
{
    Lelena t1 = new Lelena();
    t1.x = 888;
    t1.y = 999;
    Lelena t2 = new Lelena();
    System.out.println(t1.x+ t1.y+
                       t2.y+t2.x);
```



Spiral

Date



LOCAL VARIABLE

① Sometimes to meet the temporary requirements of the programmer we can declare the Variable inside a method, or block or Constructor. Such type of variables are called "Local Variable" or "temporary Variable" or "Stack Variable" or "Automatic Variables"

② Local variables will be stored inside "Stack memory".

③ Local Variable will be created while execution in which we declared it, once the block completed local variable destructed Automatically.

Spiral

Date

Hence the Scope of Local Variable is the block in which we declared it.

```
class Test
{
  pblv main (String [] args)
  { int i=0;
    for (int j=0; j<3; j++)
    {
      i = i+j;
    }
    Sopln (i + "  " + j)
  }
```

C:\ Time error find symbol:
Variable J
Location: Class Test

```
Class Test
{
  pblv main (String [] args) {
    try
    { int j= Integer.parseInt ("ten");
    }
    Catch (Number format exception e)
    { j=10;
    }
    Sopln (j);
  }
```

Spiral

Date

Ex: For local variable JVM don't provide ~~local values~~ default values
Compulsory we should perform initialization explicitly before using the variable.

that is if we are not using then it's not required perform initialisation

* Class Test
{
p s v main(String[] args)
{
int x;
Sopln("Hello");
}
}
output: Hello

* Class Test
{
p s v main(String[] args)
{ int x=0 output = 0
Sopln(x);
}

* Class Test
{ p s v main(String[] args) C.E Variable x might not have been initialized
{ int x
sopln(x);
{

Spiral

Date

(2)

```
class Test
{
  p s v main (String[] args)
  {  int x;
     if (args.length > 0)
     {
        x=10;
     }
     System.out.println(x);
  }
}
```

variable x might not have been initialised

(1)

```
class Test
{
  p s v main (String[] args)
  {
     int x;  ← recommended int x=0;
     if (args.length > 0)
     {
        x=10;   → not recommended
     }
     else
     {
        x=20
     }
     System.out.println(x);
  }
}
```

Passing command line argument:

Java Test
XYZ
O/P = 10

Java Test
O/P = 20

Spiral

Note for ② & ①

Date

initialisation for

① it is not recomended to perform local variables inside logical blocks. (if, for ...) because there is no guarantee for the execution of these block always at Run time.

② it is highly recomended to perform initialisation for local variable at the time of declaration at least with the default values.

The only applicable modifier for local variable is Final by mistake if you are trying to apply any other modifier. then you will get compile time error.

```
class Test
{
    p s v main (String[] args)
    {
        Public int x=10;
        private int x=10;
        Protected int x=10;
        Static int x=10;
        transient int x=10;
        Volatile int x=10;
        Final int x=10;
    }
}
```

C.E illegal start of Expression

Perfectly valid

Date

if we are not declaring with any modifier then by default it is default but this rule is applicable only for instance & Static Variable but not for "Local Variable

Conclusions:

① For instance & static Variable J.V.M will provide default value and we are not required to perform initialisation explicitly

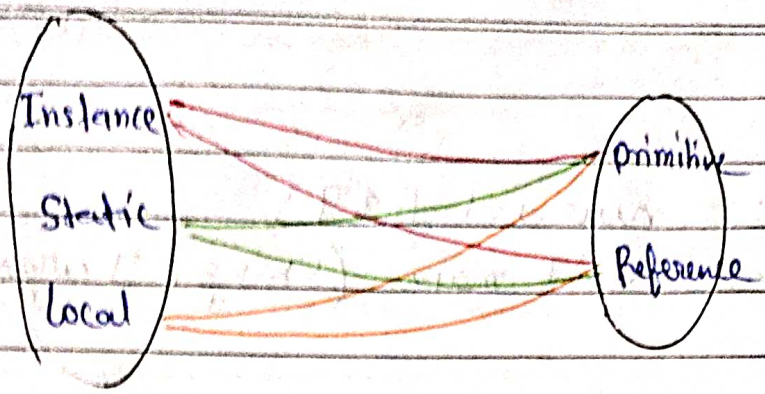
② But for local Variables J.V.M won't provide default value, compulsory we should perform initialisation explicitly before using the Variable.

③ Instance and Static Variable can be accessed by multiple thread simultaneously and Hence these are not thread safe,

but in case of local Variable for every thread, a separate copy will be created and Hence local Variables are thread safe.

Type of Variable	is thread safe
① instance Variable	✗
② Static Variable	✗
③ local Variable	✓

Date _____




```
Class Test
{                                          → Instance primitive
    int x=10;
Static reference variable → Static String s="durga";

    P S V main (String [] args)
Local reference → {  int [] y = new int [3];

    }
}
```

A every array in java is object only

(1) Every Variable in java should be either Instance or Static or local

(2) Every Variable in java should be either primitive or reference

(3) Hence various possible combination of variable in JAVA

Uninitialized Array

```
Class test
{  int [] x;
    P S V main (String [] args)
    {  Test t = new Test();
       Sopln (t.x);      Null
       Sopln (t.x[0]);   R.E → No pointer Exception
    }
}
```

default value of array for int type

Date


```
Class Test
{                                          → Instance primitive
    int x=10;
Static reference variable → Static String s="durga";

    P S V main (String [] args)
local reference → {  int [] y = new int [3];

    }
}
```

★ every array in java is object only

(1) Every Variable in java should be either Instance or Static or local

(2) Every Variable in java should be either primitive or reference

(3) Hence various possible combination of variable in JAVA

Uninitialised Array

```
Class test
{   int[] x;
    P S V main (String[] args)
    {   Test t = new Test();
        Sopln (t.x);      Null
        Sopln (t.x[0]);   R.E → No pointer Exception
    }
}
```

default value of array for int type

Spiral

```
Class Test
{
    int [] x = new int [3];
    Public static Void main (String [] args)
    {
        Test t = new test();
        System.Out.println (t.x);
        System.Out.println (t.x[0]);
    }
}
```

output [I@8d00c6
0

Instance level

① int [] x;

Sopln (obj.x); null

Sopln (obj.x[0]); R.E - NPE

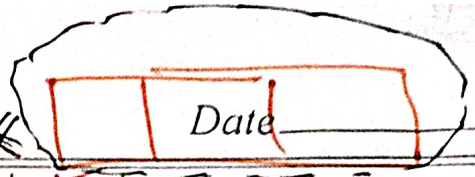
② int [] x = new int [3]

[0 | 0 | 0]

Sopln (obj.x); [I@ 3e25a5

Sopln (obj.x[0]); 0

II. Static level



① Static int[] x;

x is not pointing to memory location where an array is located

Hence the default value for 'x' is NULL

Sopln(x); null

Sopln(x[0]); R.E NPE

(ii) Static int[] x = new int[3]

Sopln(x); [I@302EF

Sopln(x[0]); 0

Prints the reference to the array x in memory

[I indicate array of integer

@302EF is the memory address of array

In java, when you declare a static array without initialising it, all the elements of the array will have their default values. For an array of integers (int[]), the default value is 0 for each element. However if you try to access an element of an array that has not been initialised (in this case "x"), you will get a Null pointer exception

III. Local level

①
```
int[] x;
Sopln(x);
Sopln(x[0]);
```
CE: Variable x might not have been initialised

② int[] x = new int[3]; [I@3a25a3

Sopln(x[0]); 0

Once we creates an array every array element by default initialised with '0' value. Whether it is instant or static or Local array.