A closer look at Methods and classes

Controlling Access to class Members

Rule

→ Within a class the instance variable should be made as private.

→ Behaviours or methods should be always made as public.

Restricting access to a class members is a fundamental part of object-oriented programming because it helps prevent the misuse of an object that by using following access specifiers

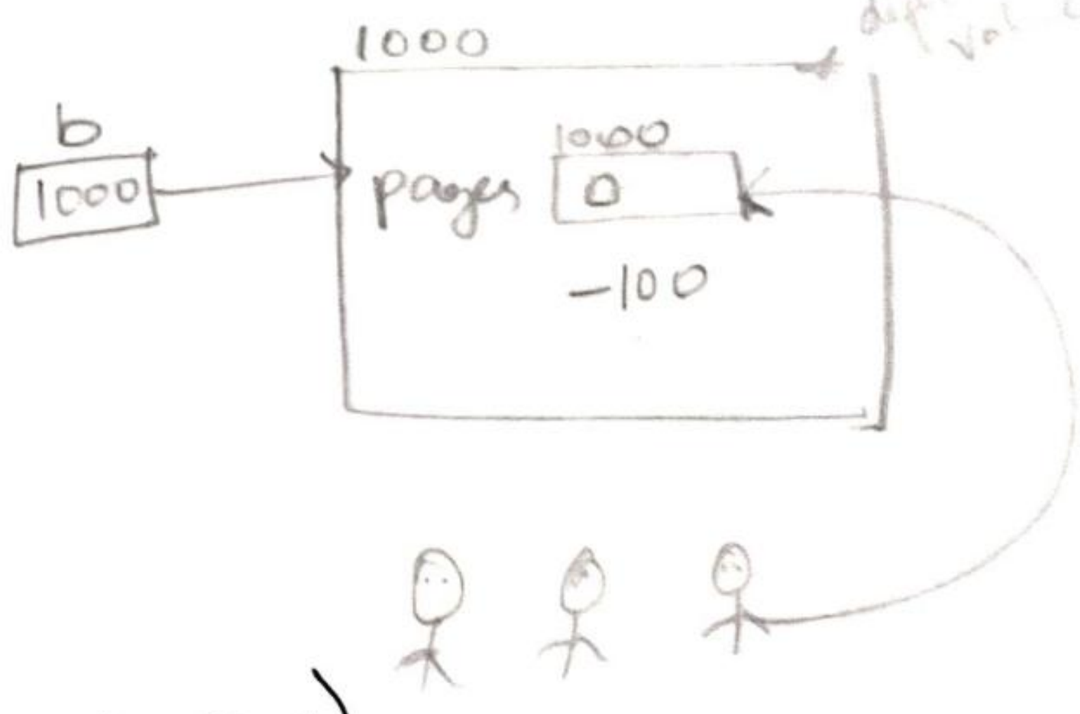
→ private
→ public
→ protected
→ Default

Modifiers	Same package			other package	
	Same class	Other class	Subclass (Inheritance)	other class	Subclass (Inheritance)
Private	Yes	No	No	No	No
Default	Yes	Yes	Yes	No	No
Public	Yes	Yes	Yes	NO	Yes
protected (Inheritance)	Yes	Yes	Yes	Yes	Yes

Note: if no access modifier is used, the default access setting is assumed.

Eg) class Demo
{
Private int a=10;
Public void access()
{
System.out.println(a);
}
}
Public class Firstpgm
{
Public static void main(String[] args)
{
Demo d = new Demo()
d.access();
// System.out.println(d.a); // error
}
}
O/P : 10

Eg2: class Book1
{
int pages;
}
public class BookApp1
{
Public static void main(String[] args)
{
Book1 b = new Book1();
b.pages = -100;
System.out.println("The page count is : "+b.pages);
}
}
O/P: -100



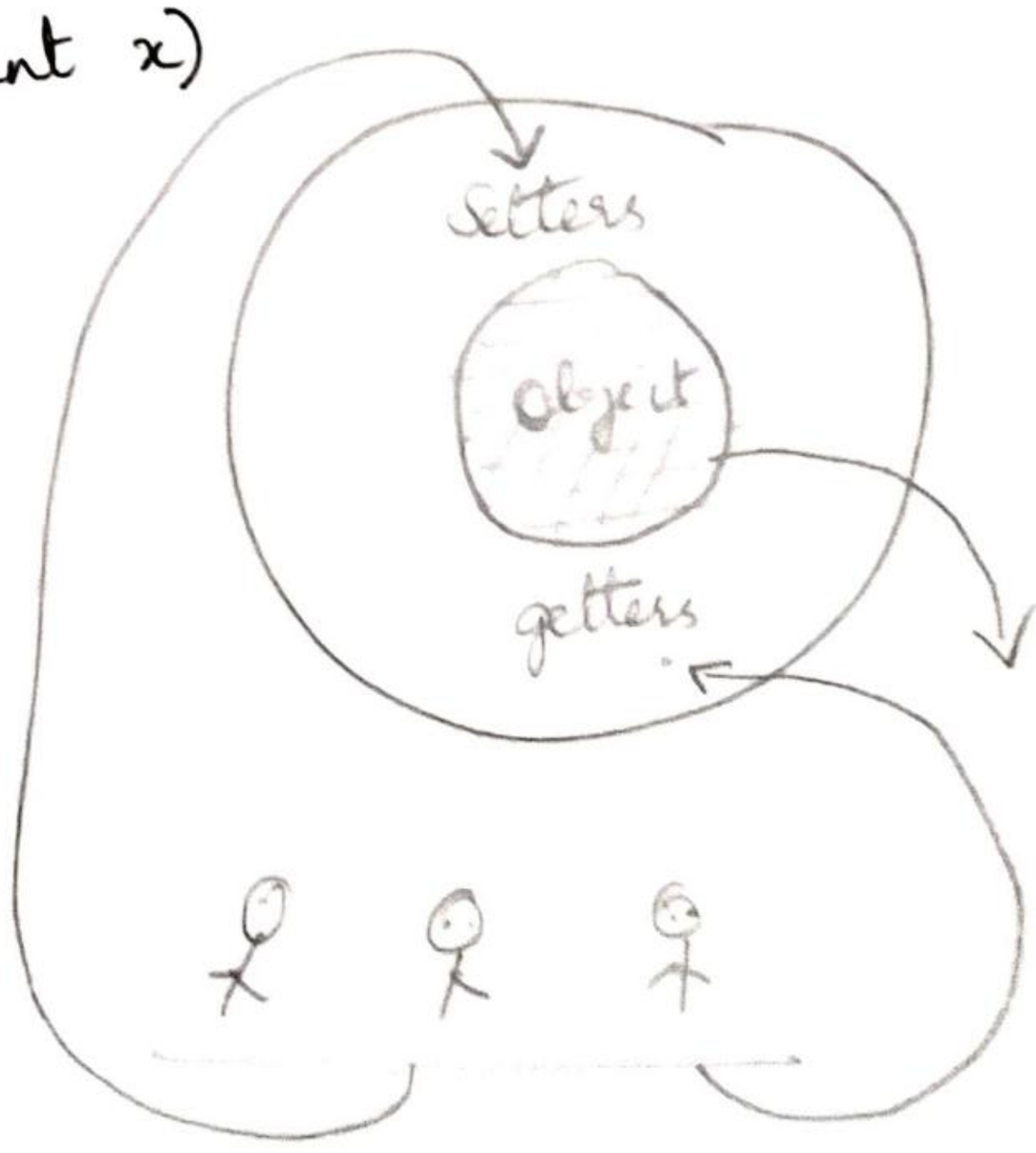
In the above program since pages is made accessible outside the class, the user may misuse the access and can give a negative value to the application

if it is made as private it cannot be accessible outside the class as result of which the application can become unused.

In order to have a controlled access over the instance variable of a class we use setters and getters

Encapsulation refers to a mechanism of providing a controlled access over the instance variable of a class.

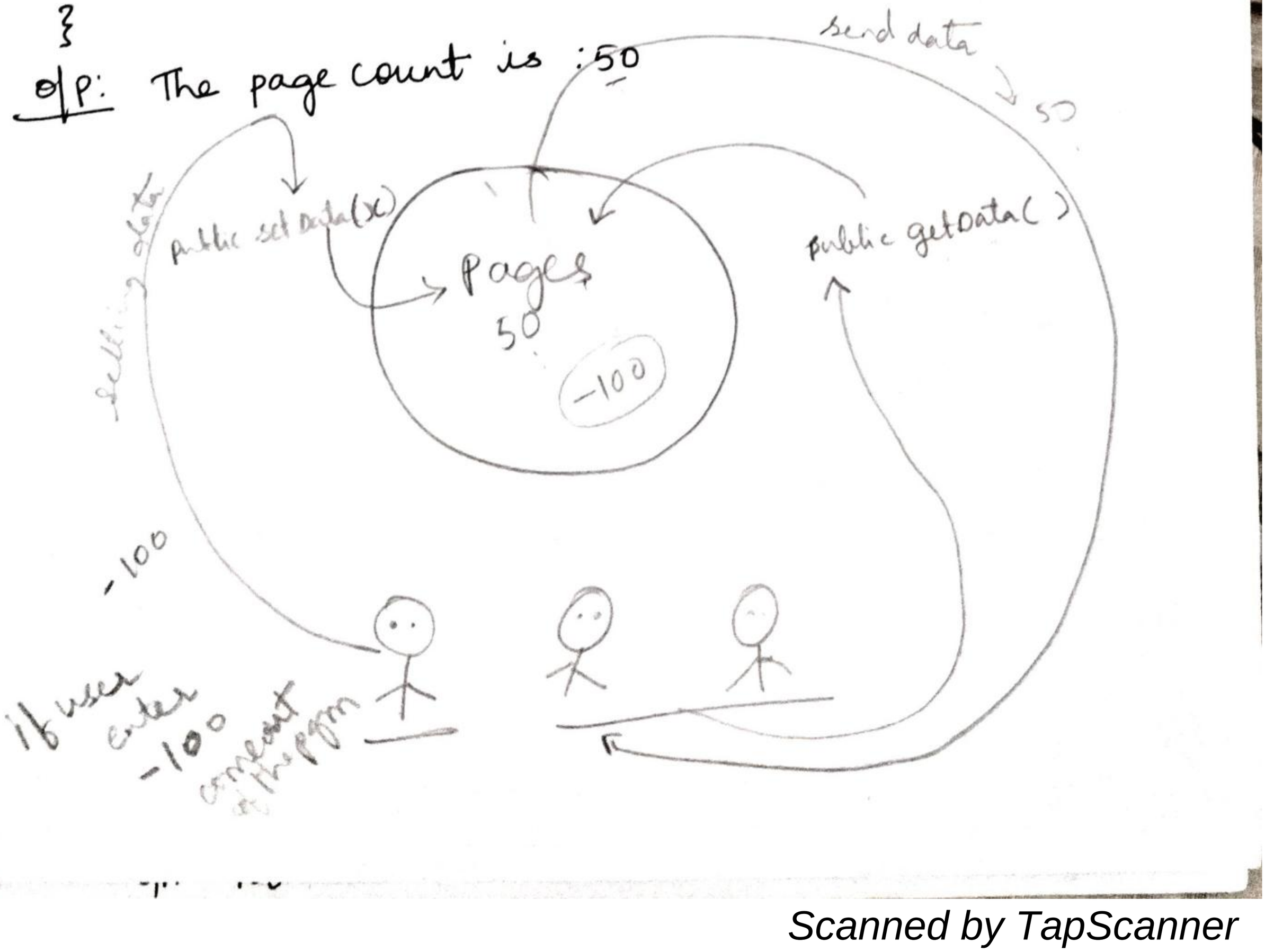
Eg: class Book
{
Private int pages;
Public void setData(int x)
{
if (x>=1)
{
pages=x;
}
else
{
;
}
}



```
  Public int getData()
  {
    return pages;
    }
 }

Public class BookApp
{
  Pnblic static void main(String[] args)
  {
    Book b = new Book();
    b.setData(50);
    int x = b.getData();
    System.out.println("The page count is :"+x);
  }
}
```

o/p: The page count is :50



Method Overloading

A method is said to be overloaded if a method has the same name with the following properties.

In order to resolve the call it make use of the following 3 rules.

1. it checks for number of parameters
2. it checks for the type of parameters
3. it checks for order of parameters.

Eg: class Addition

```
{
 void add (int x, int y)
 {
  System.out.println(x+y);   // 30
 }
 void add (int x, int y, int z)   // 60
 {
  System.out.println(x+y+z);
 }
 void add (int x, float y)
 {
  System.out.println(x+y);   // 20.0
 }
}
```

```
void add(float x, float y, float z)
{
  System.out.println(x+y+z);   //
}
 void add (double x, double y)
 {
   System.out.println(x+y);          //77.1
   }
}

 class Demo
 {
  Public static void Main(String[ ] args)
  {
  Addition a= new Addition();
  int x=10;
  int y=20;
  int z=30;
  float m=10.f;
  float n=20.1f;
  float o=30.1f;
  double p=31.333;
  double q=45.777;
```

```
a.add(x,y);
a.add(x,y,z);
a.add(x,m);
a.add(o,m,n);
a.add(p,q);
    }
}
```

O/P:
```
30
60
20.0
50.1
77 11
```

5 → 3 → ①

5 → 2 → ①

5 → 3 → ①

5 → 2 → ①

5 → 3 → ①

how many addition parameters

Same datatype based on parameters

Eg 2 Method Overloading

```
class Overload
{
    void test()
    {
      System.out.println("No parameters");
    }
    //overload test for one integer parameter
    void test(int a)
    {
      System.out.println("a:"+a);
    }
```

```
// overload test for 2 integer parameters
void test (int a, int b)
{
 System.out.println("a & b: " +a+ " " +b);
}
// overload test for double parameter
double test (double a)
 {
  System.out.println ("double a: "+a);  123.25
  return a*a;  123.25 * 123.25 = 15190.3625
 }
}
 class overloading
{
Public static void main (String[]args)
{
    Overload d = new overload ();
    double result;
// call all versions of test()
 d.test();
 d.test(10);
 d.test(10,20);
 result =d.test (123.25);
 System.out.println ("Result of d.test(123.25): "
                                    +result);
 }
}
```

o/p: no parameters
a: 10
a & b : 10 20
double a : 123.25
result of d. test (123.25) : 15190.5625

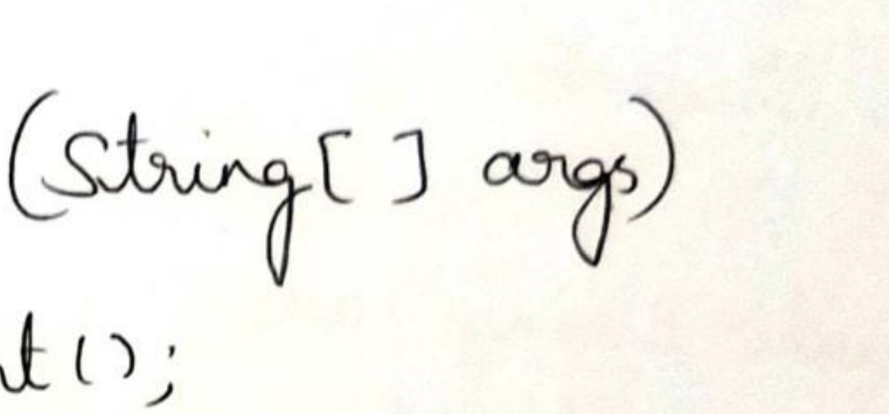
Constructor overloading

Constructor overloading is done to create the object in multiple states.

```
class Student
{
  Private String name;
  Private int id;
  Private float height;
  Public Student ()
  {
  }
  Public Student (String name, int id, float height)
  {
     this.name = name;
     this.id = id;
     this.height = height;
  }
```

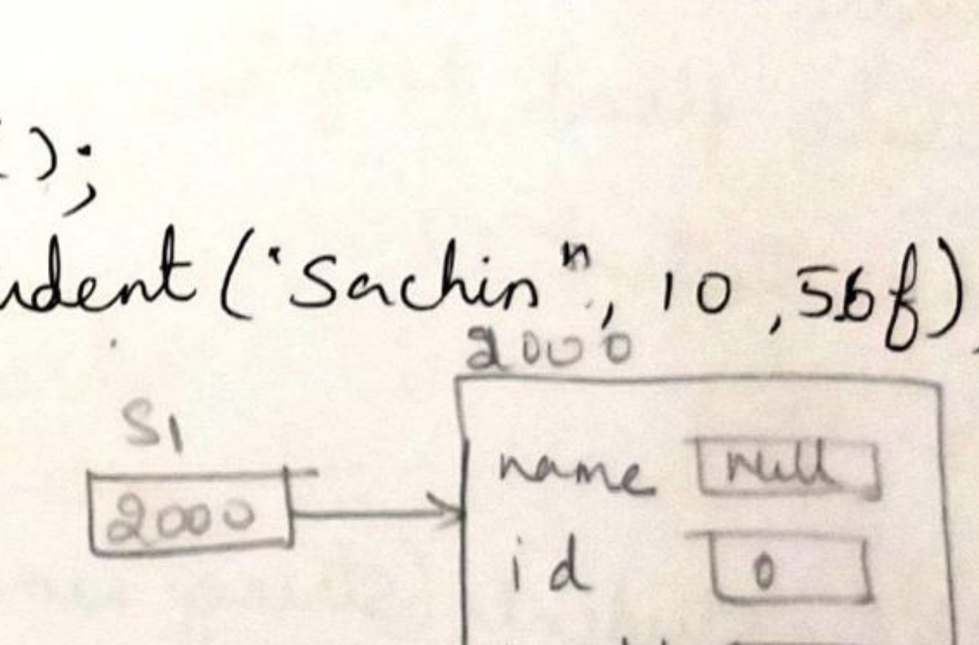
```
Public void disp()
{
 System.out.println(name);  // null   // sachin
 System.out.println(id);    // 0      // 10
 System.out.println(height); // 0.0   // 5.6f
}

Public class StudentApp
{
  Public static void main(String[] args)
  {
    Student S1 = new student();
    S1.disp();
    System.out.println();
    student S2 = new student("Sachin", 10, 56f);
    S2.disp();
  }
}
```



O/P

```
null
0
0.0
Sachin
10
5.6
```



Introducing Nested and Inner classes

In java, it is possible to define a class within another class, such classes are known as nested classes.

(or)

In java, a class can be written inside one more class such mechanism are refered as nested classes or it is also refered as inner class.

Syntax:
```
class Enclosingclass
{
   ==
   class Nestedclass
   {
      ==
   }
}
```

A nested class does not exist independently of its enclosing class. Thus, the scope of a nested class is bounded by its outer class.

There are 2 types of Nested class:

→ Static class

→ Non-static class or Inner class.

1. Static Nested class

Syntax:
```
class Enclosingclass
{
   ==
  Static class static Nested
  {
    ==
  }
}
```
Creating instance of static nested class

Enclosingclass.StaticNested ob = new EnclosingClass.StaticNested();

Eg:

2. Non - static class

→ They are known as inner class

→ it has access to all of the variables and methods of its outer class.

→ Members of the inner class are known only within the scope of the inner class and may not be used by the outer class.

Syntax:
```
class OuterClass
{
   ==
  Class InnerClass
  {
    ==
  }
}
```
Creating instance of inner class

outerclass ob = new outerclass();

1. b) write a java program to implement Innerclass and demonstrate in Access protection (10)

```
class Outer
{
   int out_x = 10;
   Public void test ()
   {
   Inner in = new Inner();
   in.disp();
   System.out.println("The outer value of x is :"
                                    + out.x);
}
class Inner
{
   int in_y = 20;
   Public void disp()
   {
    System.out.println("The outer value of x is :"
                                    +out_x);
    System.out.println("The inner value of y is :"
                                    +in_y);
   }
}
```

```
Public class Lab1b
{
  Public static void main (String[] args)
  {
     Outer O = new Outer(); // create object
     O.test();
  }
}
```

O/P: The outer value of x is : 10
The inner value of y is : 20
The outer value of x is : 10

Recursion

Recursion in java is a process in which a method calls itself continously

A method in java that calls itself is called recursive method.

why recursion

→ it requires the least amount of code to perform the necessary functions.

→ Solves complicated problems in simple steps

when?

→ The problem must be expressed in recursive manner.

→ There must be a condition which stops the recursion, otherwise there would be a stack overflo

Eg:

$$fact(n) = \begin{cases} \text{return } 1 & \text{if } n==0 \text{ or } n==1 \\ \text{return } n & n * fact(n-1) \text{ otherwise} \end{cases}$$

```
import java.util.Scanner;
Public class FactorialApp
{
  Public static void main (String[] args)
  {
```

```
Scanner scan = new Scanner(System.in);
System.out.println("Enter the value of n:"); 5
 int n = scan.nextInt();
 int ans = fact(n);
System.out.println("The fact of "+n+" is: "+ans);
Scan.close();
}
Public static int fact(int n)
 {
  if (n==0 || n==1)
  {
    return 1;
  }
   else
   {
     return (n* fact(n-1));
    }
  }
}
```

O/P: Enter the value of n:

5

The fact of 5 is: 120

Fibonacci

$$fib(n) = \begin{cases} \text{return } n \text{ if } n==1 \text{ or } n==0 \\ \text{return } fib(n-1) + fib(n-2) \text{ otherwise} \end{cases}$$

```
import java.util.Scanner;
Public static void main (String[ ] args)
{
   Scanner scan = new Scanner (System.in);
   System.out.println("Enter the value of n :");
   int n = scan.nextInt();
   int ans = fib(n);
   System.out.println("The fibonacci series is : "+ans);
   scan.close();
}

Public static intfib (int n)
{
   if (n==0 || n==1)
   {
      return n;
   }
   else
   {
      return fib(n-1) + fib(n-2);
   }
}
}
```

O/P: Enter the value of n

8

The fibonacci series is : 21

Variable length Argument : Varargs

Varargs: stands for variable - length argument.
A method that takes a variable number of arguments is called varargs method

Eg: Public class SampleApp

```
{
    Public static void main (String[] args)
    {
        add(10);
        add (10, 20);
        add (10, 20, 30);
    }
    Public static void add (int ... i)
    {
        for (int j=0; j < i.length ; j++)
        {
            System.out.print(i [j]+ "\t");
        }
        System.out.println();
    }
}
```

O/P 10

10 20

10 20 30

Understanding static

```
void disp()
{
  S.o.p("Inside non-static disp");
}
static void staticDisp()
{
  S.o.p("Inside static Disp");
}
}
class simple
{
Psvm(String[] args)
{ Demo d1 = new Demo();
  d1.disp();
  d1.staticDisp();
}
}
```

```
static
{
S.o.p("static block executed");
}
```

JVM → class loader → static variable, static block, static method

Static

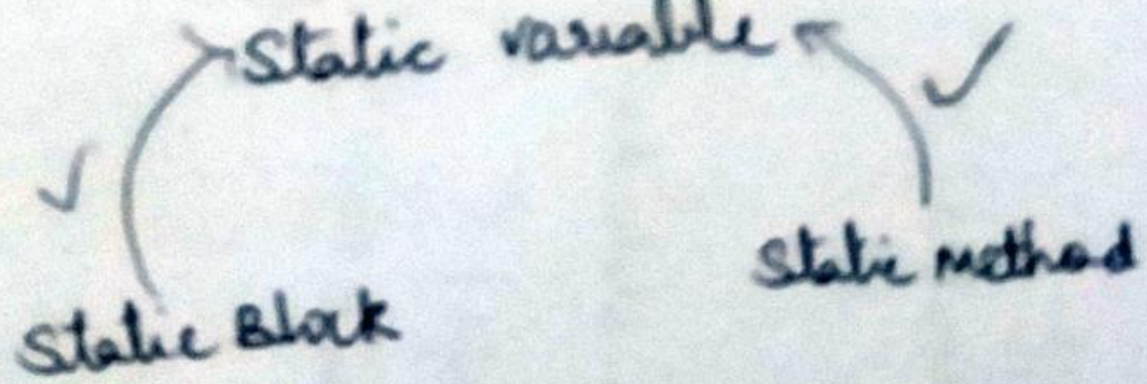
Steps followed during the creation of an object.

1. New operator would allocate a block of memory on the object space (heap)
2. Non-static block would be executed if at all if it is present within the class.
3. constructor of the class would be executed.
4. Finally the address would be returned.

O/P

Static block executed
Non-static block executed
Constructor executed
Inside non-static disp()
Inside static disp()

Rules of using static variables



1. Static method can access static variables.
2. Static block can access static variables.

Public void displayInterest ()

```
Class Demo
{
 Static int x, y, z;

Static
{
 S.o.p(x);
 S.o.p(y);
 S.o.p(z);
}
Public static void disp()
{
  S.o.p(x);
  S.o.p(y);
  S.o.p(z);
 }
}
```

Non-static variable

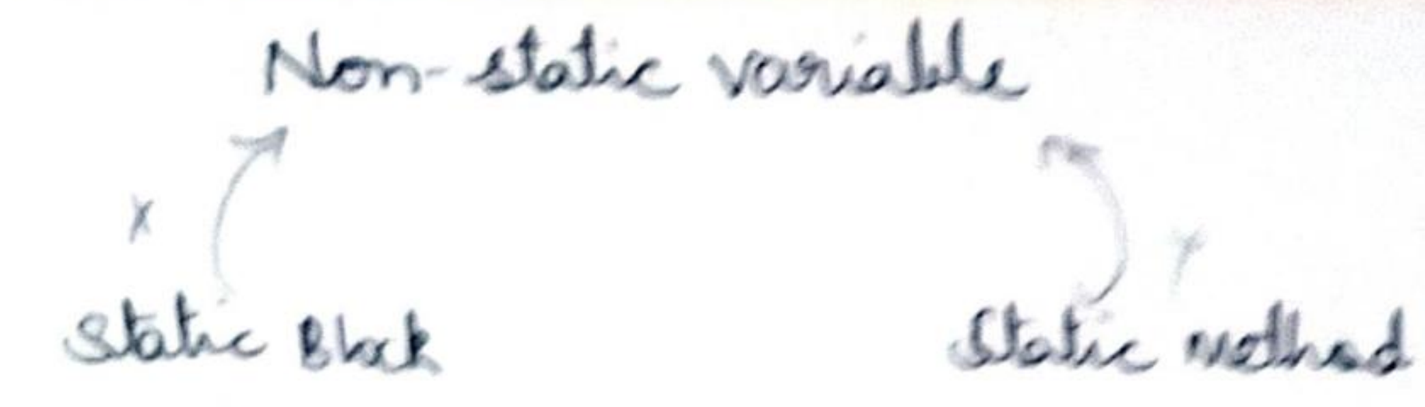
Non-static Variable

Non-static Method

3. Non-static block can access non-static variable
4. Non-static Method can access non-static ~~method~~ Variable

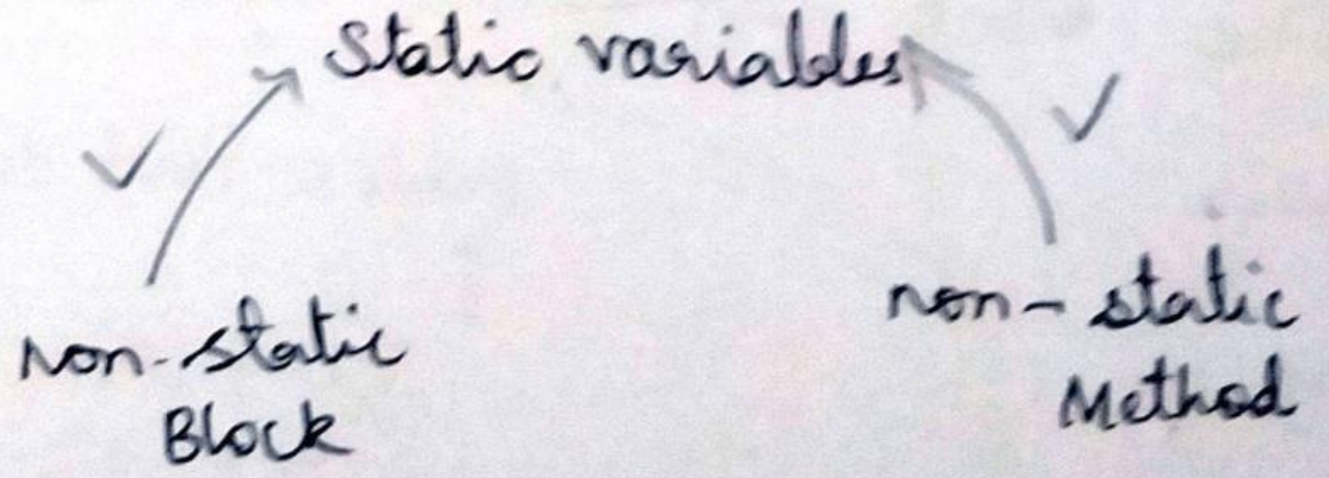
```
Class Demo
{
  int x, y, z;
  {
   S.o.p(a);
   S.o.p(y);
   S.o.p(z);
}
```

```
public void disp()
{
  S.o.p(x);
  S.o.p(y);
  S.o.p(z);
}
}
```



5. Static block cannot access non-static variables
6. Static method cannot access non-static variables

```
Class Demo
{
 int x, y, z;
 Static
 {
  S.o.p(x);
  S.o.p(y);
  S.o.p(z);
 }
 Public static void disp()
 {
  S.o.p(x);
  S.o.p(y);
  S.o.p(z);
 }
}
```



7. Non-static block can access static variable.
8. Non-static method can access static variable.

```
class Demo
{
 static int x, y, z;
 {
   S.o.p(x);
   S.o.p(y);
   S.o.p(z);
 }
   Public void disp()
   {
     S.o.p(x);
     S.o.p(y);
     S.o.p(z);
   }
}
```

Note: Static method can be called in 2 ways.

1. using object
2. using class name.

calling static method using class name.

Application of static variable

whenever the variable value remains constant throughout the application such type of applications should be made as static.

```
import java.util.Scanner;
Class Farmer
{
   float si;
    float t;
    Static float r;
    float pa;

public void input()
{
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter the principal amount");
    Pa = scan.nextfloat();
    System.out.println("Enter the time period");
    t = scan.nextfloat();

    r = 8.5f;

}
public void calculateInterest()
{
    si = (pa * t * r)/100;
}
```

```
Public void displayInterest ()
{
  System.out.println("The interest is :" + CI);
}

Public class Farmerapp
{
Public static void main (String[] args)
{
   Farmer f1 = new Farmer();
   f1.input (); //method
   f1.calculateInterest ();
   f1.displayInterest ();

   Farmer f2 = new Farmer();
   f2.input();
   f2.calculateInterest();
   f2.displayInterest();

  Farmer f3 = new farmer();
   f3.input ();
   f3.calculateInterest();
   f3.displayInterest ();
}
}
}
```