OBJECT ORIENTED PROGRAMMING USING

Java

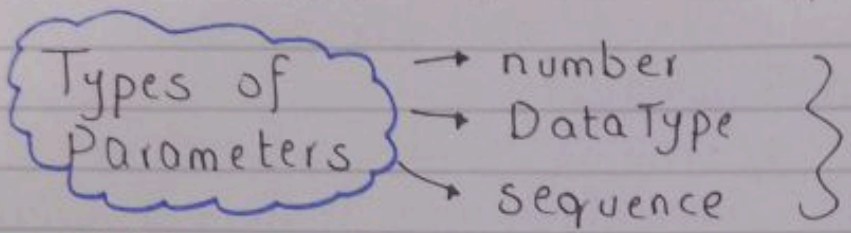
Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

Method / Function Overloading:

When a class have more than one methods with : Same name and different parameters



=> "Method Name" must be same in this case.

First, we'll see the syntax of method:

<access-modifier> <returntype> <Method Name> (list of parameters)

```
void show ()
 { System.out.println ("Hello");
 }
 void show (double d);
   { System.out.println ("Double:"+d);
   }
 void show (int i, float f);
   { System.out.println ("Float:"+f);
     System.out.println ("Integer:"+i);
   }

 void show (double d, int i);
   { System.out.println ("Double:"+d);
     System.out.println ("Integer:"+i);
```

```
public class Student {
    public static void main (String [] a)
    {
        Student S = new Student ();
        S.show ();  // call show having no parameters.
        S.show (1.1); // calling show with double datatype.
        S.show (3, 9.5); // calling show() having int and
        S.show (4.9, 6); double datatypes but we have
                         float as second parameter in func.
    }                    def. so compiler shows an error
                         and we have to do explicit type casting
                         here. double -> float
                               (bigger)  (smaller)
```

Function Calling:

Here, we're calling the method using obj.method() (name)

Compiler will see S.show() dot operator ('.') which shows that 'S' is an object of 'Student' class and then compiler will find the members of "Student" class i-e show method.

```
static int plusMethod (int x, int y) {
    return x+y; }
static double plusMethod (double x, double y){
    return x+y; }

public static void main (String [] args){
    int myNum1 = plusMethod (8, 5);
    double myNum2 = plusMthod ( 4.3, 6.26);
    System.out.println ("int: " + myNum1);
    System.out.println ("double:" + myNum2);
```

21/9/23

Java Scope - In Java, variables are only accessible inside the region they're created. This is called "scope".
In Java, a class have two members:
- Variables
- Methods.

Method Scope: Variables declared directly inside a method are available in the method following line of code in which they were declared.

```
public class Main {
  public static void main (Strig [] args)
  {   // code here cannot use x
      int x = 100;
       // Code here can use x.
      System.out.println(x);
  }
}
```

Block Scope: A block of code refers to all of the code b/w curly braces { }.
This block can't be called again bcz we need a method name for calling it.
Variables declared inside the blocks of code are only accessible by the code between the curly braces.

```
public static void main (String [] args)
  { // code here can't use x
      { int x; // code here can use x
      }
```

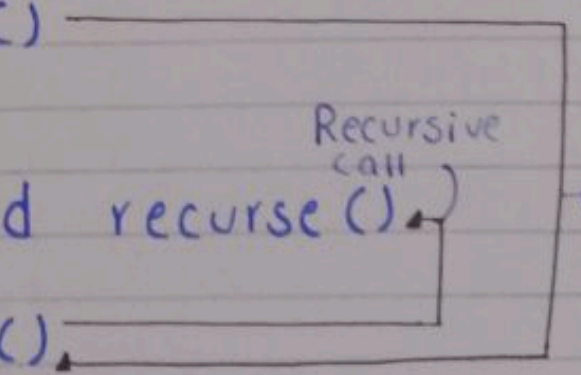
```
public class Main {
  public static void main (String[] args){

      int x;
      {
        x; // is accessible
        int y;
         y; // is accessible
      }
       y; // is not accessible
  }
```

Recursion in Java:

Recursion is a process in which a function calls itself directly or indirectly and corresponding function is called a recusive function.

```
public static void main (String[] args)
  {
     recurse()
  }
  static void recurse(){
     recurse();
  }
```



Recursion Example:

Recursion is the technique of making it easy to "Add range of numbers" together by breaking it down into the simple task of adding two numbers.

Use recursion to add all numbers upto 10.

```
public class Main {
 public static void main (String [] args)
   {
      int result = sum(10);
      System.out.println (result) ;

   }
  public static int sum (int k)
   {
      if (k > 0)
      {
        return k+sum(k-1);
      }
      else
      return 0;
   }
}}}
```

Output:
55

```
10 + sum (9);
10 + (9 + (sum(8));
10 + (9 + (8 + sum(7)));
:
:
10+9+8+7+6+5+4+3+2+1+
                    sum(0)
10+9+8+7+6+5+4+3+2+1+0
```

Calling a non-static method:

Every member in java defaults to a non-static without a 'static' keyword preading i
We can either call a non-static method by use of "this reference" or "object name".

Java this reference example: In Java, this is a reference variable that refers to current object on which the method is being involked.
It canbe used to access instance varit and methods of the "current object".
Itcan be used within a class only. can't used to access methods outside the clas

```
Class Student
{
    void show()
      System.out.println("Hello");
        this.show();
}
public static void main (String []a)
    {
      Student S = new Student;
        this.s.show();
    }
```

'this.show()' will move the cursor to show method i-e "void show()".
This will run lifetime unitil the memory is full.

If you want to use "Recursion" then you should restrict it to some limit by passing parameters to methods.

```
class Student
{   void show(int x);
    {
        System.out.println("Hello");
          if (x < 30000)
            this.show(x++);
    }
  public static void main(String[] a)
  {   Student s= new Student();
      s.show(9);  // we've called
                  show method here
                  using object 's'.
                  bcz we can't
                  use this.show()
                  outside the class
                  to access members
                  i-e methods.
  }
}

pulic class GFG
  {  int a=5;

    void f()  // non-static method
    {  System.out.println("Non-static method");
    }
    public static void main(String[] a)
    {   GFG obj = new GFG;
        obj.f(); // calling non-static
                    method / data members
    }
```

"this" keyword can also be used to change or modify the value of variables/ attributes of a class

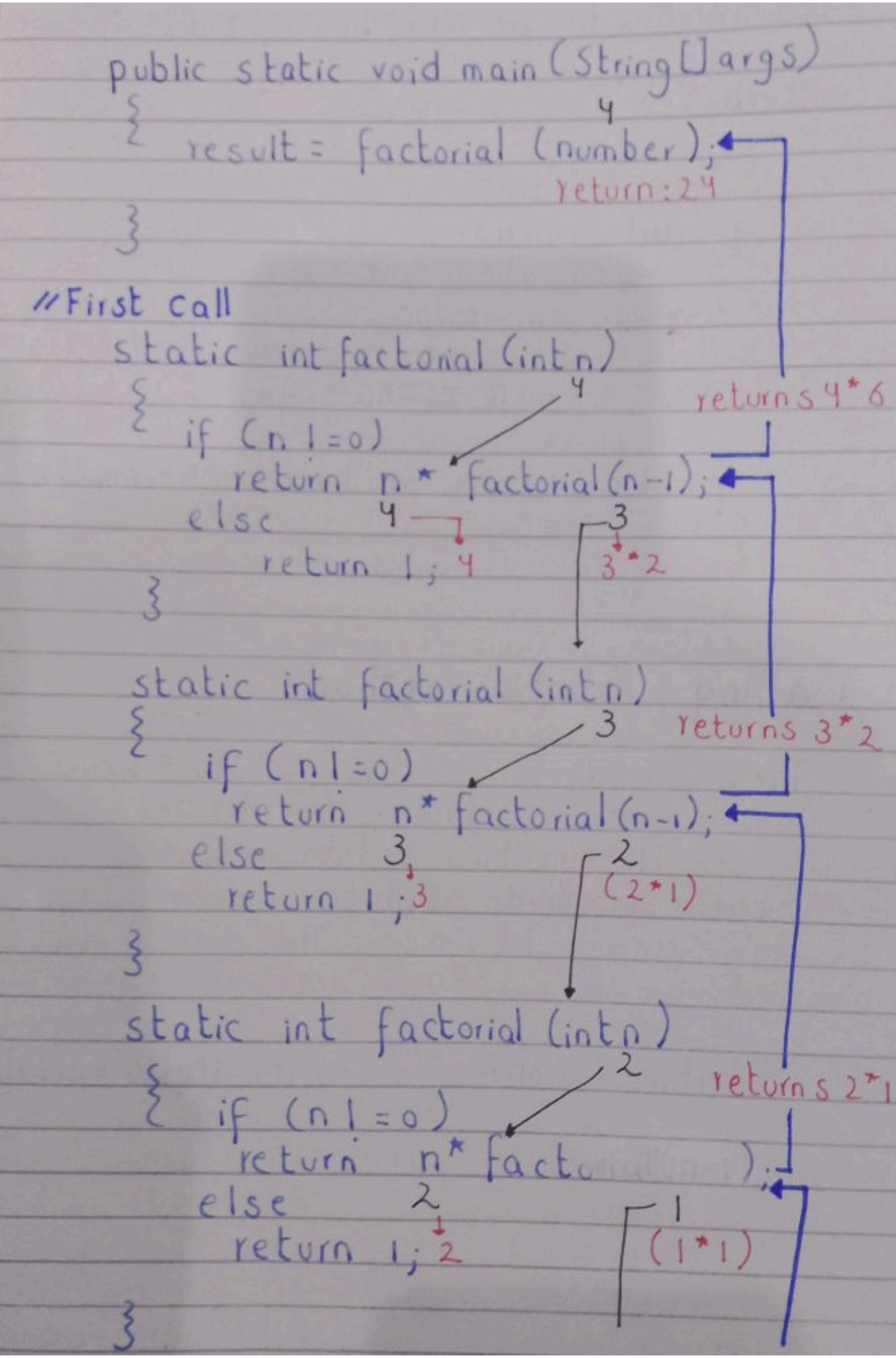
```
Class Student
{  int rollno;
   void show (int x);
   {  this.rollno = 33;
      Student n1 = new Student();
      n1.rollno = 99;
   }
   public static void main (String[]x)
{
}
```

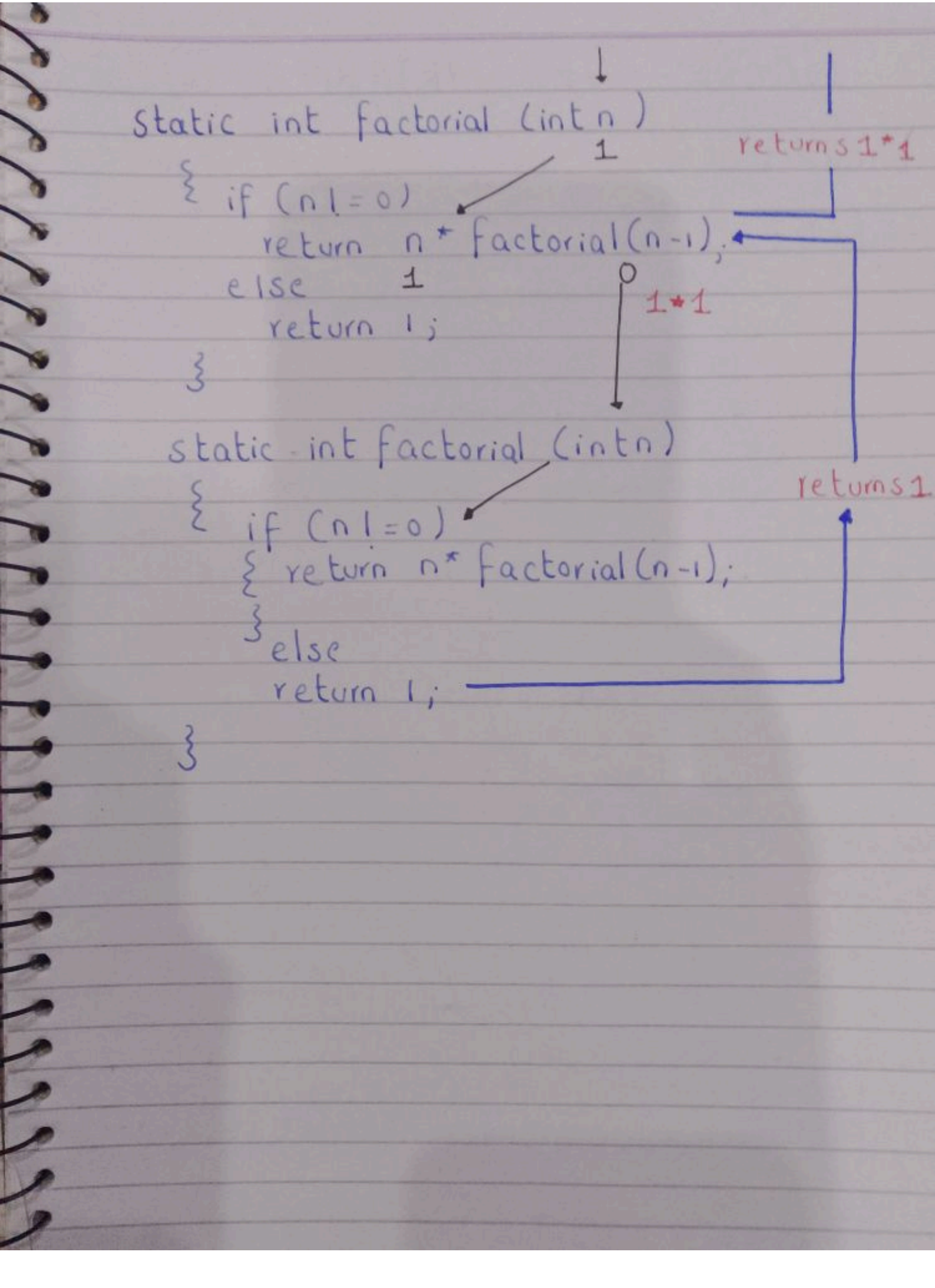
Modification of rollno attribute

Calling Static method:

They are declared using "static" keyword.
All objects of class share the same copy of static methods / data members.
Static methods denote current class.
They can be accessed using class name.

Static Variable	Non static variable
int uniName; it's shared among all students, all students have same uniName.	int rollno; it's dedicated for each student, each student has a different rollno.


public static void main (String[] args)
{
result = factorial (number);
4
return: 24
}
// First call
static int factorial (int n)
{
if (n != 0)
return n * factorial (n-1);
else
return 1;
4
4
3
3 * 2
returns 4 * 6
static int factorial (int n)
{
if (n != 0)
return n * factorial (n-1);
else
return 1;
3
3
2
(2 * 1)
returns 3 * 2
static int factorial (int n)
{
if (n != 0)
return n * factorial (n-1);
else
return 1;
2
2
1
(1 * 1)
returns 2 * 1


Static int factorial (int n)
{ if (n != 0)
return n * factorial(n-1);
else
return 1;
}
1
returns 1*1
1*1
static int factorial (int n)
{ if (n != 0)
{ return n * factorial(n-1);
}
else
return 1;
}
returns 1

SCOPE OF VARIABLES

Local: declared inside a method visible only to that method.

Global: declared outside a method but within a class visible to all parts of class.

```
import java.util.Random;
public class Main {
  public static void main (String [] args)
  {
    DiceRoller diceRoller = new DiceRoller ();

  }
  public class DiceRoller
{
  DiceRoller () → constructor
  {
    Random random = new Random ();
    int number = 0;
    roll ();
  }
  void roll ()
  {
    number = random.nextInt (6) + 1;
    System.out.println (number);
  }
```

Creating an instance of Radom class

* instance of random class and variable number is visible only within our constructor.

roll method doesn't have access to instance of random class and variable number.

Solution:

1) Passing random and number as argument to the roll() method.

```
import java.util.Random;
public class DiceRoller
{
  DiceRoller()
  {
    Random random = new Random();
    int number = random.nextInt(6)+1;
    roll(random, number);
  }
  roll(Random random, int number)
  { number = random.nextInt(6)+1;
    System.out.println(number);
  }
}
```

2) Declaring random and number before constructor.

```
import java.util.Random;
public class DiceRoller
{
  Random random = new Random();
  int number = 0;

  DiceRoller()
  { roll();
  }
  roll()
  { number = random.nextInt(6)+1;
    System.out.println(number);
```

Variable Type	Scope	Lifetime
Instance Variable	Throughout the class except in the static methods	Until the object is available in the memory.
Class variable	Throughout the class	Until the end of program.
Local variable	Within the block in which it's declared	Until the control leaves the block in which it's created

```
public class Demo
{
    // instance variable
    String name = "Andrew";
    // class/static variable
    static double height = 5.9;
    public static void main (String[] args)
    { // local variable
        int marks = 72;
}}
```

Local Variables	Instance Var.	Static Var.
• Variables declared within a method.	• Declared inside a class but outside method.	• Declared inside class outside metho with keyword static.
• Scope is limited to method in which it's declared.	• Accessible throughout the class.	• Accessible throughout the class.
• A local var. starts lifetime when method is invoked.	• Lifetime is decided by the object associated.	• Lifetime is same as that of program.
• Used to store values that are required for a particular method	• Used to store values that are needed to be accessed by different methods of class.	• Used for storing constants.

Constructors in Java:

Constructor is a "block of codes similar to the method." At the time of calling the constructor, memory for the object is allocated in the memory.
Special type of method that's used to initialize the object. Everytime an object is created using new() keyword, one constructor is called.

example

```
Class Geeks
{  Geeks() // constructor.
   { System.out.println("Constructor Called);
   }
   public static void main(String[]args)
   { Geeks geek = new Geeks();
   }
}
```

When Constructor is Called?

Each time an object is created using a new() keyword, at least one constructor (it could be a default constructor) is called to assign initial values to the data members of same class.

RULES

- The constructor must have same name as class name in which it resides.
- A constructor cannot be
  - final
  - static
  - abstract
- Access modifiers can be used with its decleration to control its access.
  public, private, protected, default

1) No-Arg Constructors:

If a constructor doesn't accept any parameters, it's is known no-Argument constructor.

```
class Main
{ int i;
  Main()
  { i = 5;
    System.out.println("Constructor is called");
  }
  public static void main (String [] args)
  { Main obj = new Main();
    System.out.println("Value of i" + obj.i);
  }}
```

2) Parameterized Constructor:

A constructor can accept one or more parameters.

```
class Main
{ String languages;

  Main(String lang)
  { languages = lang;
    System.out.println(languages + "Progr
                                    amming");
  }
  public static void main(String[] args)
{   Main obj1 = new Main("Java");
    Main obj2= new Main("C++");
}
```

3) Default Constructor:

If we don't create any constructor, the Java compiler will automatically create a no-arg constructor during execution of program. This constructor is called default constructor.
If we call a constructor with no parameters then compiler doesn't create default constructor.

```
class Main
{  int a;
   boolean b;
   public static void main (String [] args)
   {
      Main obj = new Main();
      System.out.println ("Default Value:");
      System.out.println ("a = " + obj.a);
      System.out.println ("b = " + obj.b);
}}
```

Default Value:
a = 0
b = false

Note: The default constructor initializes instance variable with default values.