14-MAR-2021

STRUCTRUE OF JAVA PROGRAM:

Java instruction are always written inside the class

```
class ClassName
{
    public static void main(String[] args)
    {
        // statements
    }
}
```

File Name: ClassName.java

NOTE:

Every class in java must have a name, it is known as Class Name

Every class has a blak, it is known as class block.

In class Block we can create

* Variables
* Methods
* Initializers

These are said to be a members of a class

Variables:

Variable is a container which is used to store data

Methods

It is a block of instructions which is used to perform a task.

Initializers:

Initializers are used to execute the start-up instructions

NOTE:

A class in java can be executed only if main method is created as follows.

Syntax to create main method:

```
public static void main(String[] args)
{
    //statements.
}
```

NOTE:

we can create a class without main method. It is compile time success. and class file is generated but we can't execute that class. Because, execution starts from main method and ends at main method.

DIFFERENCE BETWEEN PRINT AND PRINTLN STATEMENT.

Println statements:

```
System.out.println(data)
```

* println statement is used to print data as well as create a new line
* we can use the println statement without passing any data, it is just used for printing new line

eg:

```
System.out.println("Hi");
System.out.println("Laila");
System.out.println();
```

OUTPUT SCREEN:

```
Hi
laila
(no data)
```

Print Statement:

System.out.print(data)

* Print statement is used only to print the data.
* We can't use the print statement without passing any data, if we use then we will get compile time error (CTE)

eg:

```
System.out.print("Hi");
System.out.print("Laila");
// System.out.print(); //CTE
```

OUTPUT SCREEN:

```
HiLaiLa
```

Executing Hello world Program

```
class Program1
{
    public static void main(String[] args)
    {
        System.out.println("Hello world"); -> Print statement
    }
}
```

OUTPUT:

```
Hello world
```

TOKENS :

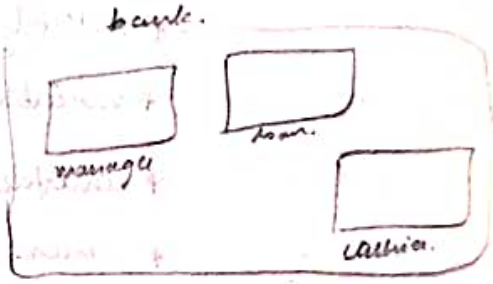
The smallest unit of programming language which is used to compose instruction is known as tokens

(OR)

tokens are the smallest unit of programming language. with the help of tokens we can built the program.

Types :

* Keywords
* Identifiers
* Literals / Data / values.



Keywords :

* A predefined words which the java compiler can understand is known as keyword
* Every keyword in java is associated with a specific task
* A programmer can't change the meaning of keyword (can't modify the associated task)

Eg :

we have 50+ keywords in java

class, public, static, void etc.,

RULE :

keywords are always written lower case.

eg: Class
while
take
give
→ cannot change the meaning if it is already defined

Identifiers:

The name given to the components of java by the Programmer is known as identifiers.

eg: to identify some us we are having name.

List of components:

* class
* method
* variable
* interface
* enum
* constructor
* Package etc.,

NOTE:

A Programmer should follow the rules and conventions of an identifier

Rules of an identifier:

* Identifiers should never starts with a number
* Identifier should not have special characters except $ and _
* character space is not allowed in identifier
* we can't use keyword as an identifier

conventions:

The coding or industrial standard to be followed by the Programmer is known as convention.

NOTE:

* Compiler doesn't validate the convention, therefore if convention is not followed then we won't get compile time error.

* It is highly recommended to follow the convention.

Convention for Class Name / Interface:

Single word - The first character should be in upper case remaining in lower case

eg: Addition, Calculator, Sum, etc,.

Multiword - The first character of every word should be in upper case remaining in lower case.

eg: SuquareRoot, PowerOfDigit, etc,.

Convention for Method

Single word - should be in lowercase:

eg: addition, calculator, sum, etc,.

Multi word - First word should be in lowercase remaining words should start with uppercase.

eg: squareRoot, powerOfDigit, factorialOfDigits, etc.,

Literals :

Literals are also called as Data or Values

Types :

* Number
* Character
* boolean
* String

The data is generally categorized into two types

* Primitive values
* Non-primitive values

Primitive values :

* Single valued data is called as primitive value.
  eg: number, character, boolean

Non-primitive values :

* Multi valued data is called as non-primitive value.
  eg: String, object reference

Primitive value :

Number Literals :

Integer number literals

eg: 1, 4, 67, 28, 37, etc,.

Floating number literals

eg: 1.5, 2.8, 37.25, ...

Character Literals:

Anything which is enclosed within a single quote (' ') is considered as a character Literal.

The length of character literals should be one.

eg: 'a', 'G', '1', '$', etc.,

Boolean Literals:

Boolean literals are used to write logical values. We have two Boolean literals

- true
- false

Non-primitive values:

Reference of an object / address is known as non-primitive value (group of data)

eg: String, details of objects, etc.,

String Literals:

Anything enclosed within a double quote (" ") is known as String literals. The length of the String literal can be anything.

They are case sensitive

eg: "Hello", "true", "a", "hello@", " ", "1.1" etc,.

Session 2: Activity:

1. Write a Java program with a class name starting with a number

2. WAJP with a class name as a keyword.

3. WAJP with a class name starting with $

4. WAJP with a class name starting with _

5. WAJP with a class name as multiword and the first letter of each word should be in upper case

6. WAJP to print the account no, IFSC, branch name, bank name and available balance.

15/3/22

Variables:

* Variable is a container which is used to store a ~~single~~ value.

* We have two types of variables. They are:

→ Primitive variable

→ Non primitive variable

Primitive Variable:

* The variable which is used to store a primitive values such as numbers, characters, boolean is known as primitive Variable.

* We can create primitive variable with the help of primitive data type

Syntax to create primitive Variable:

data type identifier1, identifier2, ... ;

Primitive datatype identifier1, identifier2, ...;

Eg:

int a; ⟶ Primitive Variable of int type

boolean b; ⟶ Primitive Variable of boolean type.

Non Primitive Variable:

* The Variable which is used to store a reference is known as Non-primitive Variable.

* It is also known as reference Variable

Syntax to create Non-primitive Variable:

Non primitive datatype identifier1, identifier2, ... ;

Eg:

String s = new String();

Datatypes:

* Datatypes are used to create variables of specific type.

* In java datatypes are classified into two types.

→ Primitive datatype

→ Non primitive datatype

eg: bottel can be plastic, glass, copper etc, based on its type.

Primitive datatype:

The datatype which is used to create a variable to store primitive value such as numbers, characters, boolean is known as primitive data type.

NOTE:

All primitive data types are keyword in java.

PRIMITIVE VALUES		PRIMITIVE DATA TYPES	DEFAULT VALUE	SIZE
Number	Integer	byte	0	1 byte
		short	0	2 byte
		int	0	4 byte
	floating	long	0l/L	8 byte
		float	0.0f/F	4 byte
		double.	0.0d/D	8 byte
Character		char	/u0000	2 byte
Boolean		boolean	false	1 bit

NOTE:

The number data type in increasing order of the capacity

byte < short < int < long < float < double

byte < short < char < int < long < float < double.

Non-primitive data type :

The data type which is used to create a non-primitive variable to store the reference is known as non-primitive data type.

NOTE:

Every class name in java is non-primitive datatype

Scope Of Variables :

* The visibility of a variable is known as scope of a variable

* Based on scope of a variable we can categorize variable in three types

→ Local variable

→ Static variable

→ Non Static variable (instance)

Local Variable:

The variable declared inside a method block or any other block except class block is known as local Variable

characteristics of Local Variable:

* We can't use local variables without initialization, if we try to use local variable without initialization then we will get compile time error

* Local variables will not be initialized with default Values

* The scope of the local variable is nested inside the block whenever it is declared, hence it can't be used outside the block.

Type Casting:

* The process of converting one type of data into another type is known as Type casting

* There are two types of type casting:

- Primitive type casting
  - widening (implicitly)
  - Narrowing (Explicitly)
- Non-primitive type casting
  - Upcasting (implicit)
  - downcasting (Explicitly)

eg: is we are taking initially byte container to store the distance travelled by you if the container is full with the size we cant store the size in the [illegible] [illegible] change the size [illegible]

Primitive type casting:

The process of converting one primitive value into another primitive value is known as primitive type casting

widening:

* The process of converting smaller range of primitive data type into large range of primitive data type is called widening

* In widening process there is no data loss

* Since, there is no data loss, compiler can implicitly do widening hence it is also known as auto widening

Narrowing:

* The process of converting larger range of primitive data type into smaller range of primitive data type is known as Narrowing

* In narrowing process there is a possibility of data loss

* Since there is a possibility of data loss, compiler does not do narrowing implicitly

* It can be done explicitly by the programmer with the help of type cast operator

Type Caste operator:

* It is a unary operator

* Type caste operator is used to explicitly convert one datatype into another data type.

16-3-21

Operators:

* Operators are predefined symbol which is used to perform specific task on the given data.
* The data given as a input to the operator is known as operand.

Based on the number of operands operator are further classified into the following

* Unary operator
* Binary operator
* Ternary operator

Unary operator:

The operator which can accept only one operand is known as unary operator

Binary operator:

The operator which can accept only two operand is known as Binary operator

Ternary operator:

The operator which can accept three operand is known as Ternary operator.

Types of operator: (based on function)

The operators can also be classified based on the task.

* Arithmetic operator (binary operator)
* Assignment operator (unary operator)
* Relational operator (binary operator)
* Logical operator (binary operator)
* Increment/decrement operator (unary operator)
* conditional operator (ternary operator)
* Miscellaneous (TYPE caste (unary) & instanceOf (binary)) (unary operator)
* Bitwise operator

Arithmetic Operator:

These are the operator which can accept two operands. so, arithmetic operator are binary operators

+	addition
-	Subtraction
*	Multiplication
/	division (Quotient)
%	modulus (Remainder)

Type1	Type 2	RESULT
Byte	Byte	int
Short	short	int
int	int	int
long	long	long
float	float	float
double	double	double
char	char	int

NOTE:

If operation done with combination of different type the result must be higher data type only.

Except for byte and short.

eg:

int + float = float.

Assignment Operator:

This operator is used to assign / give / store the value inside the variable.

eg: a=5; // here 5 is stored inside a

Compound assignment operator:

* There are the operator which perform arithmetic operation and store the data in the existing / current containner. i.e., it just update the value of the containner with new value.

* The combination of arithmetic and assignment operator is called compound assignment operator.

Operator	Example	equivalent to
+=	a+=b	a=a+b
-=	a-=b	a=a-b
*=	a*=b	a=a*b
/=	a/=b	a=a/b
%=	a%=b	a=a%b

Example 1:

```
class AssignmentOperator
{
   public static void main(String[] args)
   {
       int a=20;
           a+=10;
       System.out.println(a);
    }
}
```

O/P: 30

example 2:

```
class AssignmentOperator2
{
    public static void main(String[] args)
    {
        int a = 20;
        a+=20; // 20+20 = 40
        a-=1; // 40-1 = 39
        a*=3; // 39*3 = 117
        a/=2; // 117/2 = 58
        a+=17; // 58+17 = 75
        System.out.println(a);
    }
}
```

a

~~20~~ ~~40~~ ~~39~~ ~~117~~ ~~58~~ 75

o/p: 75

Relational operator:

* It is binary operator
* The return type of relational operator is boolean i.e, it returns either true or false

operators	Example
>	a>b
<	a<b
>=	a>=b
<=	a<=b
==	a==b
!=	a!=b

Example 1:

```
class RelationalOperator
{
    public static void main (String [] args)
    {
        int a = 5;
        int b = 2;
        boolean c = a>b;
        System.out.println (c);
    }
}
```

o/p: true

Logical Operator:

These are the operator which can accept boolean type of data. i.e, the expression or data given to the logical operator is of boolean type.

eg: 5>2 && 2>1

(5>2 → exp (returns boolean); 2>1 → exp (return boolean))

OPERATOR	OPERATION
&& (Logical AND)	If all the expression returns true. then this operator will return true
|| (Logical OR)	If any of one of the expression return true then this operator will return true
! (Logical NOT)	If the result is true then it will return false & vice versa.

example 1: (for AND &&)

case (i)

```
class LogicalOperator
{
  public static void main(String[] args)
  {
     int a=80;
     int b=30;
     int c=70;
     boolean res = a>b && b>c;
     System.out.println(res);
  }
}
```

(F) ← a>b: 80>30 (T); b>c: 30>70 (F); (T) && (F) → (F)

o/p: false

case (ii)

```
class LogicalOperator
{
  public static void main(String[] args)
  {
     int a=80;
     int b=30;
     int c=70;
     boolean res = a<b && b>c;
     System.out.println(res);
  }
}
```

80<30 → false → it won't check 2nd operand because it is && operator

(false)

o/p: false

Case (iii)

```
class LogicalOperator
{
    public static void main(String[] args)
    {
        int a=80;
        int b=30;
        int c=70;
        boolean res = !(a<b && b>c);
        System.out.println(res);
    }
}
```

80<30 → false → !false → true

→ won't check 2nd operand because it is &&.

o/p: true.

example 2: (for OR ||)

Case(i)

```
class LogicalOperator
{
    public static void main(String[] args)
    {
        int a=80;
        int b=30;
        int c=70;
        boolean res = a<b || b<c ;
        System.out.println(res);
    }
}
```

80<30 → false || 30<70 → true → true

o/p: true

Case (ii)

```
class LogicalOperator
{
  public static void main(String[] args)
  {
    int a=80;
    int b=30;
    int c=70;
    boolean res = a>b || b<c ;
    System.out.println(res);
  }
}
```

(80>30 (T)) → straightly print true because it is OR operator

o/p: true

Example 3: (for both && and ||)

```
class LogicalOperator
{
  public static void main(String[] args)
  {
    int a=80;
    int b=30;
    int c=70;
    boolean res = (a>b || b<c) && (a>b && b<c);
    System.out.println(res);
  }
}
```

(80>30 (T) won't check) True → go and check (80>30 (T) → go check → 30<70 (T)) True

o/p: true.

conditional operator:

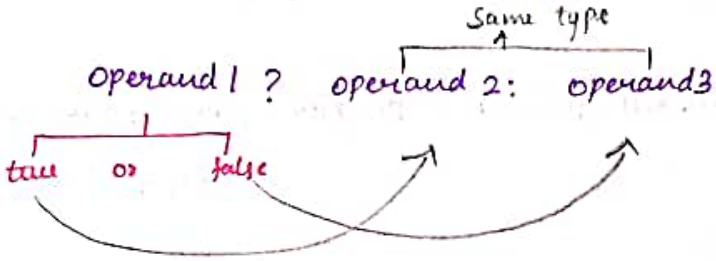
It is a ternary operator.

Syntax to create conditional operator:

```
operand 1 ? operand 2 : operand 3
condition ? statement 1 : statement 2
```

operation:

* The return type of operand 1 must be Boolean
* If the condition returns true, statement 1 will get executed else statement 2 will get executed.



example:

WAJP to find greatest of two numbers.

```
class GreatestOfTwoNumbers
{
  public static void main(String[] args)
  {
    int a=20;
    int b=10;
    int res = a>b ? a : b ;
              20>10  20  10
              T
    System.out.println(res);
                       20
  }
}
```

O/P: 20

example 2:

WAJP to find the given number is even or odd

```
class EvenOrOdd
{
    public static void main(String[] args)
    {
        int given_num = 7;
        String res = (given_num%2==0)? "Even" : "odd";
        System.out.println("The given_num "+given_num+" is "+
                           res + " number");
    }
}
```

7%2==0
1==0 (F)

o/p: The given_num 7 is odd number.

example 3:

WAJP to find the given char is alphabet or not.

```
class Alphabet
{
    public static void main(String[] args)
    {
        char ch = 'S';
        String res = (ch>=65 && ch<=90) || (ch>=97 && ch<=122)?
                     "Yes" : "No";
        System.out.println("The given character is "+ch+" is alphabet");
    }
}
```

o/p: Yes The given character S is alphabet

WAJP to find the greatest of three number:

```
class GreatestNumber
{
   public static void main(String[] args)
   {
      int num1 = 7;
      int num2 = 9;
      int num3 = 2;
      // first check num1 is greather than num2 if true then check num1 is
         greather than num3 if both are true print num1 else check num2 and num3.
      int res = ((num1 > num2) && (num1 > num3))? num1 :
                                  (num2 > num3)? num2 : num3;

      System.out.println("The given number "+res+" is greatest
                                                    number");

   }
}
```

o/p: The given number 9 is greatest number

WAJP to find the smallest of three numbers:

```
class SmallestOfThree
{
   public static void main(String[] args)
   {
      int num1 =7;
      int num2 =9;
      int num3 =2;
      int res = num1 < num2 ? (num1 < num3 ? num1 : num3) : (num2 < num3
                7 < 9 (T)      7 < 2 (F)                    num2 : num3);

      System.out.println("The given number "+res+" is smallest
                                                    number");

   }
}
```

o/p: The given number 2 is smallest number

Increment and decrement operator: (It is unary operator)

Increment operator:

* It is used update the variable by 1
* They are two types of Increment operator
    * Pre - Increment
    * Post - Increment

Pre - Increment:

* It is denoted as ++i

Operation:

→ Increase the value by 1 and update
→ Substitute the updated value
→ Use the substited value

Post - Increment:

* It is denoted as i++

Operation:

→ Substitute
→ Increase the value by 1 and update
→ Use the substituted value

Decrement operator:

* It is used to ~~decrease~~ update the variable by decreasing 1.
* They are two types of decrement operator
    * Pre - decrement
    * Post - decrement

Pre - decrement :

* It is denoted as --i

Operation :

→ Decrease the value by 1 and update
→ Substitute the updated value
→ use the substituted value

Post - Decrement :

* It is denoted as i--

Operation :

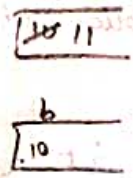
→ Substitute
→ Decrement the value by 1 and update
→ use the substituted value.

NOTE :

each time the value get updated while doing increment / decrement.

Example :

```
class Increment
{
   public static void main(String[] args)
   {
      int a=10;
      int b=a++;
      System.out.println(a);
      System.out.println(b);
   }
}
```



o/p : 11
10

Example :

```
class IncrementAndDecrement
{
    public static void main(String [] args)
    {
        int a=7;
        int b=8;
        int res = a+ --a + ++b - a++;
        System.out.println(res); //16
        System.out.println(a); //7
        System.out.println(b); //9
    }
}
```

a: 7 8 7

b: 8 9

res: 16

7 + 6 + 9 - 6
22 - 6 = 16

o/p : 16
7
9

Example :

```
class IncrementAndDecrement
{
    public static void main(String [] args)
    {
        int a=7;
        int b=7;
        a = a++; //7 // here first 7 is used and the
                     value is stored in a then
        b = ++b; //8   it update the value
        System.out.println(a);
        System.out.println(b);
    }
}
```

a: 7 8

b: 7 8

o/p : 7
8

17/3/22

Decision Statements :

Decision statements help the programmer to skip the block of instructions from the execution if the condition is not satisfied.

Types of Decision Statements :

* if statement
* if-else statement
* if- else if statement
* switch

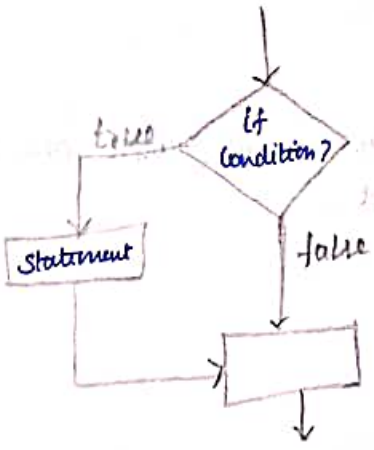
If Statement :

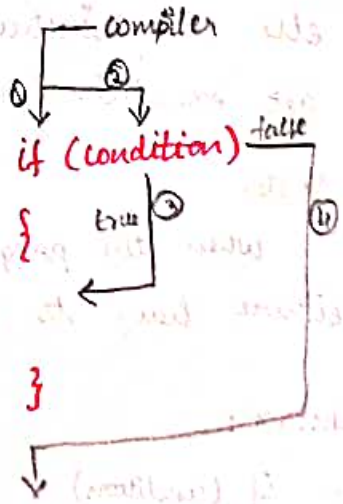
The instruction written inside the if block will execute if the condition returns true. else it will not execute the instruction inside the block.

Syntax :

```
if (condition)
{
}
```

work flow :





WAJP to print the even number if the number is even.

```
class EvenNumber
{
    public static void main(String[] args)
    {
        int num=7;
        if (num%2==0)
            7%2=1 //1==0 (F)
        {
            System.out.println(num);
        }
    }
}
```

o/p: nothing

if else Statement:

The instruction written inside the if block will get executed only if the condition is satisfied else the instruction written inside the else block will get executed.

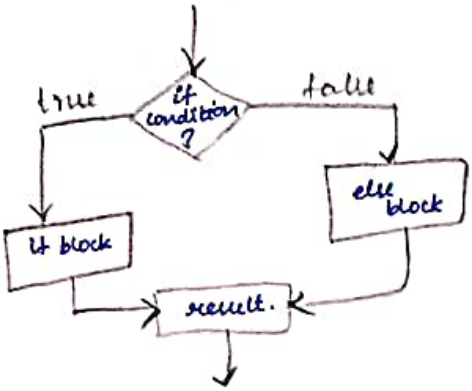
NOTE:

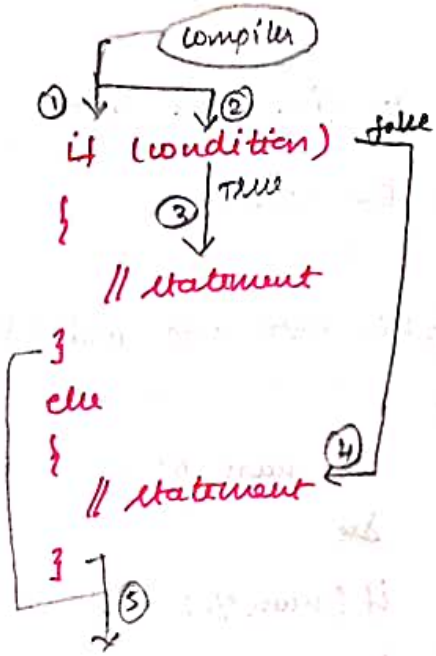
When the program wants the result with one condition either have to execute or not.

Syntax:

```
if (condition)
{
}
else
{
}
```

workflow :





WAP two find the greatest of two numbers.

```
class GreatestOfTwo
{
   public static void main(String[] args)
   {
      int num1 = 55;
      int num2 = 17;
      if (num1>num2)
      {
        System.out.println("The greatest number is "+num1);
      }
      else
      {
        System.out.println("The greatest number is "+num2);
      }
   }
}
```

WAJP to find the given number is even or odd.

```
class EvenOrOdd
{
    public static void main(String[] args)
    {
        int num = 6;
        if (num % 2 == 0)
        {
            System.out.println("The given number " +num+ " is even");
        }
        else
            System.out.println("The given number " +num+ " is odd");
    }
}
```

o/p: The given number 6 is even

WAJP to find the given number is divisible of 2 & 3.

```
class Divisible
{
    public static void main(String[] args)
    {
        int num = 6;
        if (num % 2 == 0 && num % 3 == 0)
        {
            System.out.println("The given number is divisible of 2 & 3");
        }
        else
            System.out.println("The given number is not divisible of 2 & 3");
    }
}
```

o/p: The given number is divisible of 2 & 3

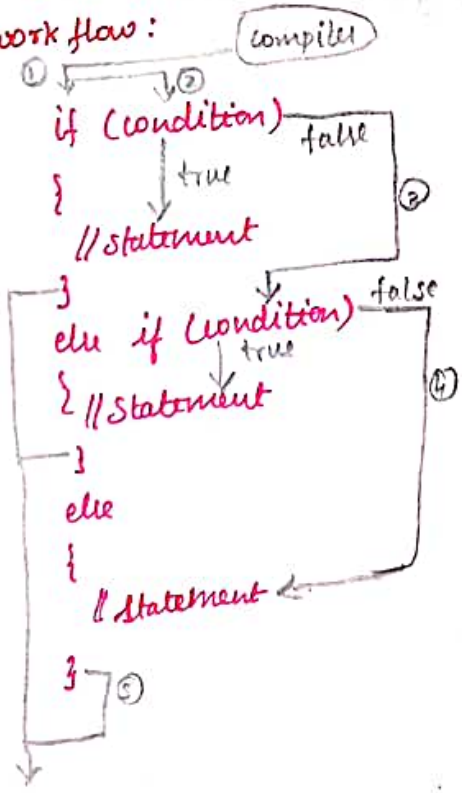
if - else if ladder :

If the condition is satisfied then the instruction written inside the if block gets executed if not satisfied, condition is checked in the else if block from top to bottom order and if the condition is satisfied in any of the else if block then, only that else if block is gets executed. if not satisfied else block gets executed remaining blocks are skipped.

Syntax :

```
if (condition)
{
}
else if (condition)
{
}
else if (condition)
{
}
.
.
else
{
}
```

work flow :



WAJP to find the greatest of Three numbers.

```
class GreatestOfThree
{
   public static void main(String [] args)
   {
      int num1 = 366;
      int num2 = 34;
      int num3 = 568;
      if( num1 > num2 && num1 > num3)
      {
        System.out.println("The greatest number is "+num1);
      }
      else if ( num2 > num3)
      {
        System.out.println("The greatest number is "+num2);
      }
      else
      {
        System.out.println("The greatest number is "+num3);
      }
   }
}
```

O/P:

The greatest number is 568

WAP to find the smallest of 5 numbers:

```
class SmallestOfFive
{
   public static void main(String [] args)
   {
      int n1= 7 ;
      int  n2=8 ;
      int  n3 = 1;
      int  n4 = -3;
      int  n5 =0;
     if (n1<n2 && n1<n3 && n1<n4 && n1<n5)
     {System.out.println("The smallest number is " +n1 ); }
     else if (n2 <n3 && n2 < n4 && n2 <n5)
      {System.out. println("The smallest number is " +n2 );}
     else if (n3 <n4 && n3 <n5)
      {System. out.println ("The smallest number is " + n3 );}
     else if (n4 <n5)
      { System. out. println ("The smallest number is " + n4 );
      }
     else
      {System.out.println ("The smallest number is " + n5 );
      }
   }
}
```

o/p:

The smallest number is -3

WAJP to find the type of character.

```
class TypeOfChar
{
    public static void main(String[] args)
    { char ch = '$';
      if (ch >= 'A' && ch <= 'Z' || ch >= 'a' && ch <= 'z')
        System.out.println("The given char is a Alphabet");
      else if (ch >='0' && ch <= '9')
        System.out.println("The given char is a number");
      else
        System.out.println("The given char is a special character");
    }
}
```

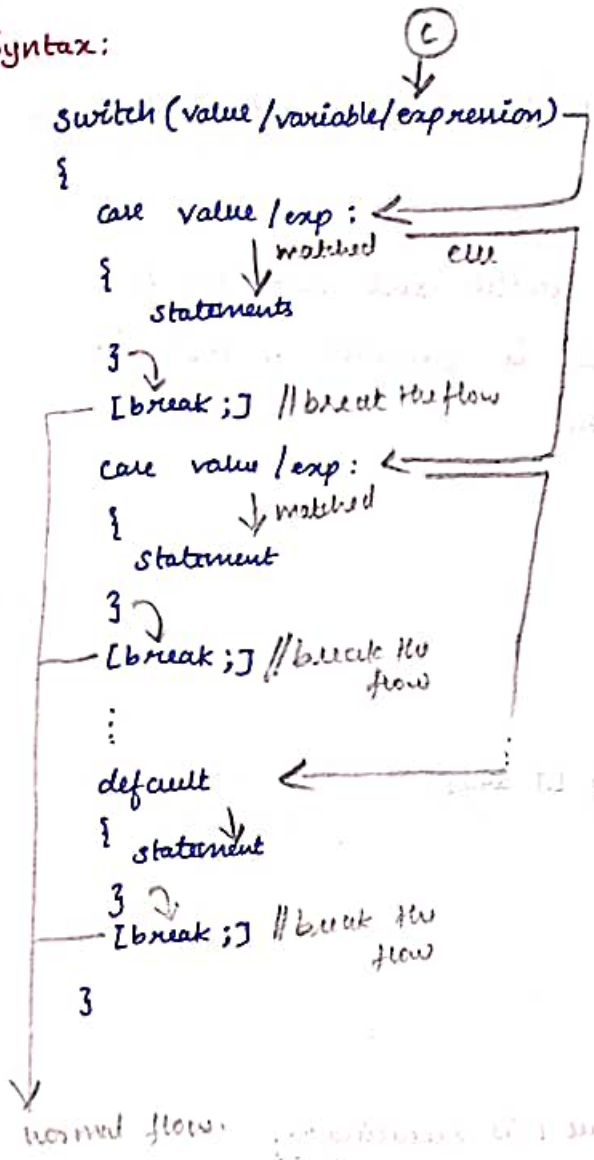
O/P:

The given char is a special character

Switch statement:

* Switch Statement is used for "pattern Matching"
* If the value is matching with the case means it will execute the block and control is transfer to the below cases also.
* In order to stop the flow of execution we have to use "break".

Syntax:



work flow :

* The value / variable / expression passed in the switch gets compared with value passed in the case.
* If any of a case is satisfied, the case block is executed and all the blocks present below gets executed
* If no case is satisfied then default gets executed
* For a case we can use a break statement which is optional

NOTE :

→ For a switch we can't pass long, float, double, boolean

→ For a case we can't pass variable.

BREAK :

*break is a keyword and it is a control transfer Statement

* break is used inside the switch and loop block
* when the break statement is executed control is transfered outside the block.

Example :

```
class SwitchCase
{
  public static void main(String [] args)
  {
    int v=2;
    switch(v)
    {
      case 1:
        System.out.println("case 1 is executing");
      break;
      case 2:
        System.out.println("case 2 is executing");
      break;
      case 3:
        System.out.println("case 3 is executing");
      default:
        System.out.println("default case is executed");
    }
  }
}
```

o/p :

case 2 is executing.

TC 1 : V=1

with break :

case 1 is executing

without break :

case 1 is executing
case 2 is executing
case 3 is executing
default case is executed

TC 2 : V=3

with break :

case 3 is executing

without break :

case 3 is executing
default case is executed.

WAJP to design a Game: (by grouping)

```
class Game
{
   public static void main(String[] args)
   {
      int task = 3;
      switch (task)
      {
         case 1:
         case 4:
         case 6:
            System.out.println("sing");
         break;
         case 2:
         case 7:
            System.out.println("Dance");
         break;
         case 3:
         case 5:
            System.out.println("Truth");
         break;
         case 8:
            System.out.println("story");
         break;

      }
   }
}
```

O/P: Truth

Example :

```
class LogicalFinder
{
   public static void main(String[] args)
   {
      int num1=5;
      int num2 = 68;
      int choice = 2 ;
      System.out.println (" you have choosen " +choice+ " function");
      switch (choice)
      {
         case 1:
         { System.out.println (" you have choosen addition");
           int c= num1+num2;
           System.out.println ("The addition of two number is "+c);
         }
         break;
         case 2:
         {
           System.out.println (" you have choosen subtraction");
           int d = num1-num2;
           System.out.println ("The subtraction of two number is "+d);
         }
         break ;
         case 3:
         {
           System.out.println (" you have choosen multiplication");
           int c= num1 * num2 ;
           System.out.println ("The multiplication of two number
                                is " +c );
         }
         break
```

```
case 4:
{
  System.out.println("you have choosen division");
  int c = num1/num2;
  System.out.println("The division of two number is " +c);
}
break;
default
   System.out.println("choose the correct function");
}
}
}
```

o/p:

T. Case(i) choice = 2

you have choosen 2 function
you have choosen Subtraction
The substraction of two number is -63

T. case (ii) choice = 1

you have choosen 1 function
you have choosen Addition
The addition of two number is 73

T. case (iii) choice = 4

you have choosen 4 function
you have choosen division
The division of two number is 0

T. case (iv) choice = 7

you have choosen 7 function.
Choose the correct function

Example:

WAJP to find the given year is leap year not

```
class LeapYearOrNot
{
   public static void main(String [] args)
   {
      int year = 1700;
      if((year % 4 ==0 && year % 100 !=0) || year % 400==0)
         System.out.println (year + " is a leap year");
      else
         System.out.println (year + " is not a leap year");
   }
}
```

O/P:

1700 is not a leap year

Example:

WAJP to find the given number is positive or negative or 0

```
class CheckInteger
{
   public static void main(String [] args)
   {
      int num =5;
      if (num >0)
         System.out.println ("The given number " +num+ " is a
                                                positive number");
      else if (num < 0)
         System.out.println ("The given number " +num+ " is a
                                                negative number");
      else
         System.out.println ("The given number is 0");
   }
}
```

o/p:

The given number 5 is a positive number

Example:

WAP to find the given character is vowel or consonant

```
class Vowel
{
   public static void main (String[] args)
   {
      char ch = 's';
      switch (ch)
      {
         case 'A':
         case 'E':
         case 'I':
         case 'O':
         case 'U':
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':
            System.out.println ("The given character is vowel");
         break;
         default:
            System.out.println ("The given char is consonent");
      }
   }
}
```

o/p:

The given char is consonent

18/3/22 Scanner (Read value from User)

Dynamic Read :

The process of reading a data from the user during execution of a process is known as dynamic read.

Step to Achieve Dynamic Read :

step 1: Import Scanner class from java util package

```
import java.util.Scanner;
```

Step 2: Create an object for the Scanner class.

```
Scanner input = new Scanner(System.in)
```

step 3: call the method of Scanner class to read the data from the user

```
input.method();
```

Methods in a Scanner class :

Types of Data	Method Signature	Return type
byte	nextByte()	byte
short	nextShort()	short
int	nextInt()	int
long	nextLong()	long
float	nextFloat()	float
double	nextDouble()	double
boolean	nextBoolean()	boolean
char	next().charAt(0)	char
String (s.w)	next()	String
String (n.w)	nextLine()	String

WAJP to perform multiplication of two numbers by using Scanner.

```
import java.util.Scanner;
class Multiplication
{
    public static void main(String[] args)
    {
        Scanner s= new Scanner(System.in);
        System.out.println("enter the number");
        int num1 = s.nextInt();
        System.out.println("enter the 2nd number");
        int num2 = s.nextInt();
        System.out.println("The multiplication of two number is " +
                                                    num1 * num2);
    }
}
```

O/P:

```
enter the number
4
enter the 2nd number
10
The multiplication of two number is 40
```

WAJP to display name, age and gender by reading value from the user.

```
import java.util.Scanner;
class Details
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Please enter your name");
        String name = sc.nextLine();
```

```
        System.out.println("Please enter your age");
        int age = sc.nextInt();

        System.out.println("Please enter the gender");
        char gender = sc.next().charAt(0);

        System.out.println("The details are:");
        System.out.println(name + "  " + age + "  " + gender);
    }
}
```

o/p:

```
Please enter your name
Lavanya L
Please enter your age
23
Please enter the gender
female
The details are :
Lavanya L  23  f
```

WAJP to find the addition of two character.

```
import java.util.Scanner;
class AdditionChar
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("enter the 1st char");
        char ch1 = s.next().charAt(0);
        System.out.println("enter the 2nd char");
        char ch2 = s.next().charAt(0);
        int sum = ch1+ch2;
        System.out.println("The addition of two char is "+sum);
    }
}
```

O/P:

enter the 1st char
a
enter the 2nd char
A
The addition of two char is 162

Loop Statement

* Loop statements help the programmer to execute the set of instruction repeatedly.

* In java we have different types of loop statement, they are:

  * while loop
  * do while loop
  * for loop
  * for each/ advanced loop
  * nested loop

eg: If the programmer wants to print the ~~sta~~ same statement more than one time or many time we will go for looping.

NOTE:

To perform looping we have to follow three steps

They are:

→ declaration & initialization
→ condition
→ updatation

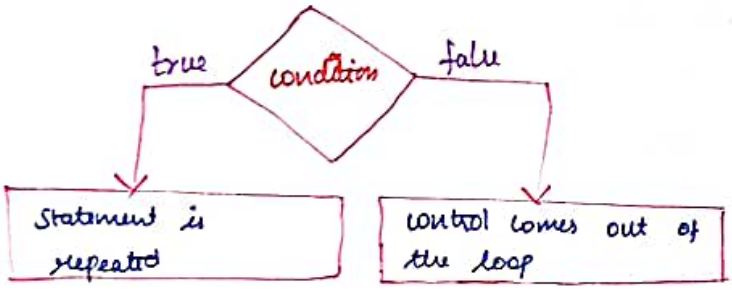
While Loop:

When the programmer do not know the number of iteration. Then he go for while loop.

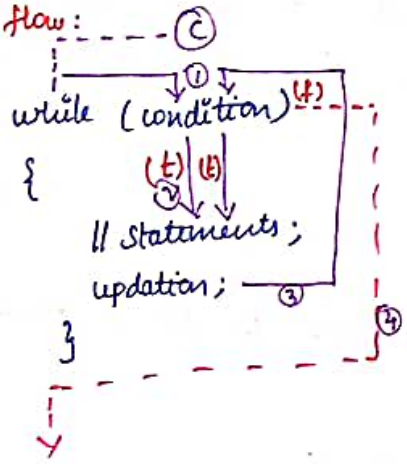
Syntax:

```
while (condition)
{
  //Statements ;
}
```

flow chart:



work flow:



Example:

```
class WhileLoop
{
  public static void main(String [] args)
  {
    int i = 1;
    while ( i <= 5)
    {
      System.out.println("Hi ");
      i++;
    }
  }
```

Tracing:

i

1 2 3 4 5 6

I i=1

1<=5 (T)

S.o.pln ("Hi");

i++;

II i=2

2<=5 (T)

S.o.pln ("Hi");

i++;

III i=3

3<=5 (T)

S.o.pln ("Hi");

i++

IV i=4

4<=5 (T)

S.o.pln ("Hi");

i++

V i=5

5<=5 (T)

S.o.pln ("Hi");

i++

VI i=6

6<=5 (F)

comes out of the loop

OUTPUT SCREEN

Hi

Hi

Hi

Hi

Hi

WAJP to print the first 10 Natural numbers.

```
class NaturalNumber
{
    public static void main(String[] args)
    {
        int i = 1;
        while (i<=10)
        {
            System.out.println(i);
            i++;
        }
    }
}
```

Tracing:

i.

1 2 3 4 5 6 7 8 9 10 11

(i) i<=10 1<=10 (T) S.O.pln(i) i++	(v) i<=10 5<=10 (T) S.O.pln(i) i++	(ix) i<=10 9<=10 (T) S.O.pln(i) i++	OUTPUT SCREEN 1 2 3 4 5 6 7 8 9 10
(ii) i<=10 2<=10 (T) S.O.pln(i) i++	(vi) i<=10 6<=10 (T) S.O.pln(i) i++	(x) i<=10 10<=10 (T) S.O.pln(i) i++	
(iii) i<=10 3<=10 (T) S.O.pln(i) i++	(vii) i<=10 7<=10 (T) S.O.pln(i) i++	(xi) i<=10 11<=10 (F) comes of loop	
(iv) i<=10 4<=10 S.O.pln(i) i++	(viii) i<=10 8<=10 (T) S.O.pln(i) i++		

WAJP to print the even numbers in the given Range

```
import java.util.Scanner;
class EvenNumber
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("enter the initial number");
        int m = s.nextInt();
        System.out.println("enter the ending number");
        int n = s.nextInt();
        System.out.println("the even numbers are");
        while(m <=n)
        {
            if (m%2 ==0)
                System.out.println(m);
            m+=2;
        }
    }
}
```

m	n
~~2,4,6,8~~	7

tracing:

(i) m<=n
2 <=7 (T)
2%2==0 (T)
S.O.pln(m)
m+=2

(ii) m<=n
4 <=7 (T)
4%2 ==0 (T)
S.O.pln(m)
m+=2

(iii) m<=n
6 <=7 (T)
6%2 ==0 (T)
S.O.pln(m)
m+=2

(iv) m<=n
8 <=7 (F)
comes out of loop

OUTPUT SCREEN:

```
enter the initial number
2
enter the ending number
7
the even numbers are
2
4
6
```

do-while:

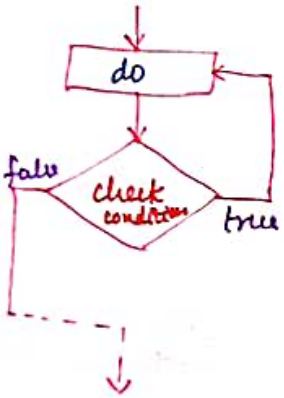
When the programmer wants to execute the instruction at least one time without checking the condition.

we can go for do-while loop.

Syntax:

```
do
{
 // statements
}
while (condition);
```

flow chart:

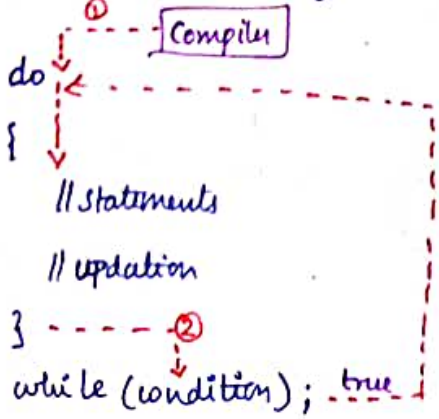


work flow:

Case (i): when the condition is true.

* control goes to the loop block directly, execute the instruction

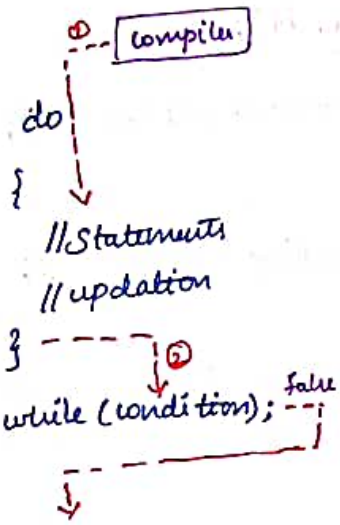
* Then control goes to the condition, if the condition is true the control goes back to the do block.



case (ii) : When the condition is false

* control goes to the loop block directly, execute the instruction

* Then control goes to the condition, if the condition returns false the loop stops and control goes to the next statement.



Difference between while And do-while loop

WHILE

In while loop first condition being checked. if condition is satisfied then the task will be done

→ Syntax:

```
while (condition)
{
  statements;
}
```

→ eg:

```
class wl
{
  public static void main(String [] args)
  {
    int count =1;
    while (count <6)
    {
      S.o.pln (count);
      count ++;
    }
  }
```

DO-WHILE

In do-while loop first it execute the task and then it will check the condition

→ Syntax:

```
do
{
  statement;
}
while (condition);
```

→ eg:

```
class Dwl
{ public static void main(String [] args)
  {
    int count =1;
    do
    {
      S.o.pln (count);
      count ++;
    } while (count<6);
  }
}
```

Example: WAJP to print the Alphabets in the given Range

```
import java.util.Scanner;
class PrintAlphabets
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the starting alphabet");
        char ch1 = s.next().charAt(0);
        System.out.println("Enter the ending alphabet");
        char ch2 = s.next().charAt(0);
        System.out.println("The alphabets in the given range are:");
        do
        {
            System.out.print(ch1+" ");
            ch1++;
        }
        while(ch1<=ch2);
    }
}
```

Tracing:

ch1: A B C D E

ch2: D

(i) S.o.p(A+" "); ch1++; ch1<=ch2 → 66<=68 (T)

(ii) S.o.p(B+" "); ch1++ ch1<=ch2 → 67<=68 (T)

(iii) S.o.p(C+" "); ch1++ ch1<=ch2 → 69<=68 (T)

(iv) S.o.p(D+" "); ch1++ ch1<=ch2 → 69<=68 (F) comes out of loop

OUTPUT SCREEN:

```
Enter the starting alphabet
A
Enter the ending alphabet
D
The alphabets in the given range are:
A B C D
```

WAJP to print the sum of natural numbers in the given range

```
import java.util.Scanner;
class SumOfNaturals
{
    public static void main(String[] args)
    {
        Scanner s= new Scanner(System.in);
        System.out.println("Enter the starting Number");
        int num1= s.nextInt();
        System.out.println("Enter the ending number");
        int num2 = s.nextInt();
        int sum=0;
        System.out.println("The sum of natural numbers :");
        do
        {
            sum = sum+num1;
            num1++;
        }
        while(num1<=num2);
        System.out.println(sum);
    }
}
```

Tracing:

num1	num2	sum
~~1 2 3 4 5 6~~	5	~~0 1 3 6 10 15~~

(i) sum = sum+num1;
sum = 0+1;
sum = 1
num++;
2<=5 (T)

(ii) sum = 1+2;
sum = 3
num++;
3<=5 (T)

(iii) sum = 3+3
sum = 6
num++;

(iv) sum = 6+4
sum = 10
num++;
5<=5 (T)

(v) sum = 10+5
sum = 15
num++;
6<=5 (F)
Comes of the loop

OUTPUT SCREEN:

```
Enter the starting Number
1
Enter the ending Number
5
The sum of natural numbers:
15
```

for loop :

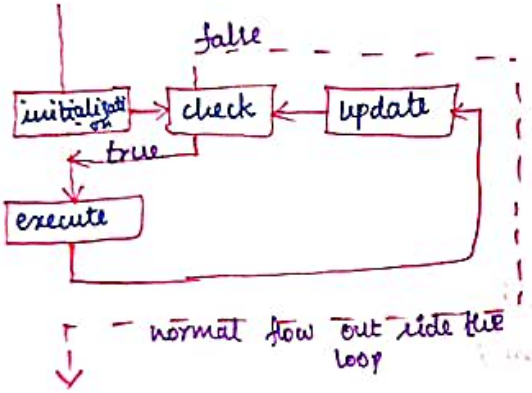
* When the program knows the number of iteration we can go with for loop.

* In for loop the basic & important steps three steps (ie) declaration & initialization , condition , updation are given in the same line.

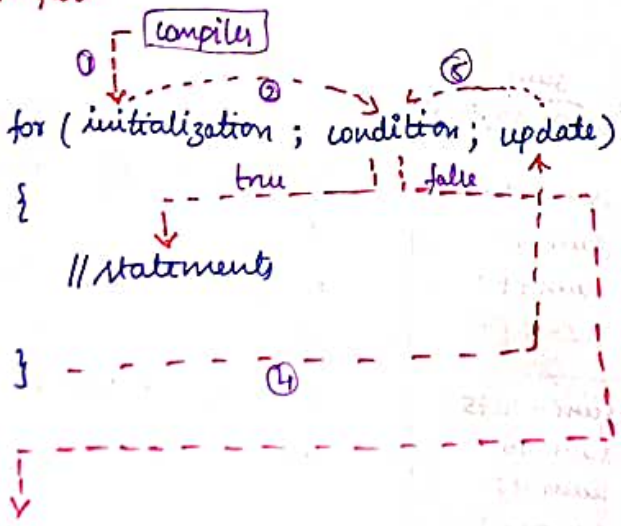
Syntax :

```
for (initialization ; condition ; update)
{
   // statement
}
```

flow chart :



work flow :



WAJP to check the given number is prime or not.

```
import java.util.Scanner;
class Prime
{
   public static void main(String[] args)
   {
      Scanner s = new Scanner(System.in);
      System.out.println("Enter the number ");
      int num = s.nextInt();
      int count =0, i;
      for(i=1; i<=num; i++)
      {
         if (num%i == 0)
         {
            count++;
         }
      }
      if (count ==2)
         System.out.println("The given number "+num+ "is prime");
      else
         System.out.println("The given number "+num+ " is not prime");
   }
}
```

Tracing:

num	count	i
3	0 1 2	1 2 3 4

(i) i=1
1<=3 (T)
if(3%1==0) (T)
count++

(ii) i=2
2<=3 (T)
if(3%2==0) (F)

(iii) i=3
3<=3 (T)
if(3%3==0) (T)
count++

(iv) i=4
4<=3 (F)
comes out of loop

if (count==2)
2==2 (T)
S.o.pln("The given number" num+"is prime");

OUTPUT SCREEN

```
Enter the number
3
The given number 3 is prime
```

19/3/22 Programming Session.

1) WAJP to check whether the given number is prime or not.

```
import java.util.Scanner;
class PrimeOrNot
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the number");
        int num = s.nextInt();
        int count = 0;
        for(int i=1; i<=num; i++)
        {
            if(num%i==0)
            {
                count++;
            }
        }
        if(count==2)
        {
            System.out.println("The given number "+num+" is a prime");
        }
        else
            System.out.println("The given number "+num+" is not a prime");
    }
}
```

Tracing

num 5 i 1 2 3 4 5

count 0 1 2

(i) i=1 1<=5 (T)
if(5%1==0) (T)
count++

(ii) i=2 2<=5 (T)
if(5%2==0) (F)

(iii) i=3 3<=5 (T)
if(5%3==0) (F)

(iv) i=4 4<=5 (T)
if(5%4==0) (F)

(v) i=5 5<=5 (T)
if(5%5==0) (T)
count++

(vi) i=6 6<=5 (F)
comes out of for loop

```
if(count==2)
    2==2 (T)
S.O.ln("Prime");
```

OUTPUT :

Enter the number
5
The given number 5 is a prime

2) WAJP to find the factorial of the given number.

```
import java.util.Scanner;
class Factorial
{
    public static void main(String [] args)
    {
        Scanner s= new Scanner(System.in);
        System.out.println(" Enter the number");
        int num = s.nextInt();
        int fact = 1;
        for (int i=1; i<=num; i++)
        {
            fact = fact * i;
        }
        System.out.println("The factorial of the given number "+num+
                                                    "is "+fact);
    }
}
```

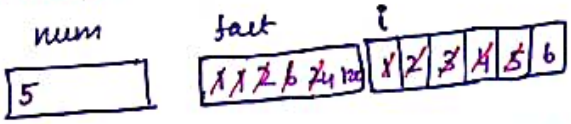
OUTPUT:

Enter the number

5

The factorial of the given number 5 is 120

Tracing:



(i) i=1 1<=5(T)
fact = 1*1
fact=1

(ii) i=2 2<=5(T)
fact = 1*2
fact = 2

(iii) i=3 3<=5(T)
fact = 2*3
fact = 6

(iv) i=4 4<=5(T)
fact = 6*4
fact = 24

(v) i=5 5<=5(T)
fact = 24 *5
fact = 120

(vi) i=6 6<=5 (F)
comes out of
for loop

3) WAJP to print the factors of the given number

```
import java.util.Scanner;
class Factors
{
    public static void main(String [] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the number : ");
        int num = s.nextInt();
        System.out.print("The factors of the given number is : ");
        for(int i=1; i<=num; i++)
        {
            if(num%i==0)
            {
                System.out.println(i+" ");
            }
        }
    }
}
```

OUTPUT :

Enter the number
4
The factors of the given number is : 1 2 4

Tracing

num	i
4	1 2 3 4 5

(i) i=1 1<=4(T) if(4%1==0)(T) S.O.P(1)

(ii) i=2 2<=4(T) if(4%2==0)(T) S.O.P(2)

(iii) i=3 3<=4(T) if(4%3==0)(F)

(iv) i=4 4<=4(T) if(4%4==0)(T) S.O.P(4)

(v) i=5 5<=4(F) comes out of for loop

4) WAJP to print the number of digits in the given number.

```
import java.util.Scanner;
class CountDigits
{
    public static void main(String[] args)
    {
        Scanner s= new Scanner(System.in);
        System.out.println("Enter the number to find the number of digits");
        int num =s.nextInt();
        int count =0;
        System.out.println("The number of digits present in the given
                                                    number are");
        while (num>0)
        {
            num=num/10;
            count++;
        }
        System.out.println(count);
    }
}
```

OUTPUT:

Enter the number to find the number of digits

58127

The number of digits present in the given number are 5

Tracing:

num	count
~~58127~~ 0	~~0~~ 1 2 3 4 5

(i) 58127 > 0 (T)
num = 58127/10
num = 5812
count++

(ii) 5812 > 0 (T)
num = 5812/10
num = 581
count++

(iii) 581 > 0 (T)
num = 581/10
num = 58
count++

(iv) 58 > 0 (T)
num = 58/10
num = 5
count++

(v) 5 > 0 (T)
num = 5/10
num = 0
count++

(vi) 0 > 0 (F)
comes out of the loop

numbers.

```
import java.util.Scanner;
class SumOfEvenNumber
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the number");
        int num = s.nextInt();
        int store, sum = 0;
        while (num != 0)
        {
            store = num % 10;
            if (store % 2 == 0)
            {
                sum = sum + store;
            }
            num /= 10;
        }
        System.out.println("The sum of even number is :" + sum);
    }
}
```

OUTPUT :

Enter the number

5183414

The sum of even number is 16

Tracing

num 5183414 0 store 4 1 4 3 8 1 5

sum 0 4 8 16

(i) 5183414 != 0 (T)
store = 5183414 % 10
store = 4
if (4 % 2 == 0) (T)
sum = 0 + 4
sum = 4
num = 5183414 / 10
num = 518341

(ii) 518341 != 0 (T)
store = 518341 % 10
store = 1
if (1 % 2 == 0) (F)
num = 518341 / 10
num = 51834

(iii) 51834 != 0 (T)
store = 51834 % 10
store = 4
if (4 % 2 == 0) (T)
sum = 4 + 4
sum = 8
num = 51834 / 10
num = 5183

(iv) 5183 != 0 (T)
store = 5183 % 10
store = 3
if (3 % 2 == 0) (F)
num = 5183 / 10
num = 518

(v) 518 != 0 (T)
store = 518 % 10
store = 8
if (8 % 2 == 0) (T)
sum = 8 + 8
sum = 16
num = num / 10
num = 51

(vi) 51 != 0 (T)
store = 51 % 10 => store = 1
if (1 % 2 == 0) (F)
num = 51 / 10 => num = 5

(vii) 5 != 0 (T)
store = 5 % 10
store = 5
if (5 % 2 == 0) (F)
num = 5 / 10 => num = 0

(viii) 0 != 0 (F)
comes out of the loop

6) WAJP to print the reversing order of the given number (without storing)

```
import java.util.Scanner;
class ReversingNumber
{
    public static void main(String[] args)
    {
        Scanner s= new Scanner(System.in);
        System.out.println("Enter the number to reverse");
        int num = s.nextInt();
        int temp = num, rev;
        while (num !=0)
        {
            rev = num %10;
            System.out.print (rev);
            num /=10;
        }
    }
}
```

OUTPUT:

Enter the number to reverse

1523

3251

Tracing

num	rev
1523 0	3251

(i) 1523 !=0 (T)
rev = 1523 %10
S.O.P(rev)=3
num = 1523/10
num = 152

(ii) 152 !=0 (T)
rev = 152 %10
rev = 2
S.O.P(2)
num = 152/10
num = 15

(iii) 15 !=0 (T)
rev = 15%10
rev = 5
S.O.P(5)
num = 15/10
num = 1

(iv) 1 !=0 (T)
rev = 1%10
rev = 1
S.O.P(1)
num = 1/10
num = 0

(v) 0 !=0 (F)
comes out of loop

7) WJP to print the reversing order of the given number (with storing)

```
import java.util.Scanner;
class ReversingNumber
{
   public static void main(String[] args)
   {
      Scanner s = new Scanner(System.in);
      System.out.println("Enter the number to reverse");
      int num = s.nextInt();
      int temp = num, rev = 0, store;
      while (num != 0)
      {
         store = num % 10;
         rev = rev * 10 + store;
         num /= 10;
      }
      System.out.println("The reversing order of the given number " temp
                         " is " + rev);
   }
}
```

OUTPUT :

Enter the number to reverse

5237

The reversing order of the given number 5237 is 7325

Tracing :

	num	temp	rev	store
	5237 0	5237	0 7 73 732 7325	7 3 2 5

(i) num != 0 (T)
5237 != 0
store = 5237 % 10
store = 7
rev = 0 * 10 + 7
rev = 7
num = 5237 / 10
num = 523

(ii) 523 != 0 (T)
store = 523 % 10
store = 3
rev = 7 * 10 + 3
rev = 73
num = 523 / 10
num = 52

(iii) 52 != 0 (T)
store = 52 % 10
store = 2
rev = 73 * 10 + 2
rev = 732
num = 52 / 10
num = 5

(iv) 5 != 0 (T)
store = 5 % 10
store = 5
rev = 732 * 10 + 5
rev = 7325
num = 5 / 10
num = 0

(v) 0 != 0 (F)
Comes out of the loop

8) WAJP to check the given number is palindrome or not.

```
import java.util.Scanner;
class Palindrome
{
  public static void main(String[] args)
  {
    Scanner s = new Scanner(System.in);
    System.out.println("Enter the number to reverse");
    int num = s.nextInt();
    int temp = num, rev = 0, store;
    while (num!=0)
    {
      store = num % 10;
      rev = rev * 10 + store;
      num = num/10;
    }
    if (rev == temp)
      System.out.println("The given number is palindrome");
    else
      System.out.println("The given number is not palindrome");
  }
}
```

OUTPUT:

Enter the number to reverse

12521

The given number is palindrome.

Tracing

num: ~~12521~~ ~~1252~~ ~~125~~ ~~12~~ ~~1~~ 0 — temp: 12521

rev: 0 1 12 125 1252 12521 — s: ~~1~~ ~~2~~ ~~5~~ ~~2~~ ~~1~~

(i) 12521 != 0 (T)
store = 12521 % 10
store = 1
rev = 0 * 10 + 1
rev = 1
num = 12521 / 10
num = 1252

(ii) 1252 != 0 (T)
store = 1252 % 10
store = 2
rev = 1 * 10 + 2
rev = 12
num = 1252 / 10
num = 125

(iii) 125 != 0 (T)
store = 125 % 10
store = 5
rev = 12 * 10 + 5
rev = 125
num = 125 / 10
num = 12

(iv) 12 != 0 (T)
store = 12 % 10
store = 2
rev = 125 * 10 + 2
rev = 1252
num = 12 / 10
num = 1

(v) 1 != 0 (T)
store = 1 % 10
store = 1
rev = 1252 * 10 + 1
rev = 12521
num = 1 / 10
num = 0

(vi) 0 != 0 (F)
comes out of loop

if (rev == temp)
12521 == 12521 (T)
S.O.pln ("palindrome");

9. WAJP to print the prime numbers in the given Range

```
import java.util.Scanner;
class PalindromeInRange
{
  public static void main(String[] args)
  {
    Scanner s= new Scanner(System.in);
    System.out.println("Enter the starting number");
    int num1 = s.nextInt();
    System.out.println("Enter the ending number");
    int num2 = s.nextInt();
    System.out.print("The prime numbers in the given range are: ");
    for(int i=num1; i<=num2; i++)
    { int count=0;
      for(int j=1; j<=i; j++)
      { if(i%j==0)
          count++;
      }
      if(count==2)
      System.out.print(i+" ");
    }
  }
}
```

OUTPUT:

Enter the starting number
3
Enter the ending number
7
The prime numbers in the given range are: 3 5 7

Traing.

num1 = 3, num2 = 7

i: 3 4 5 6 7 8

(1) 3<=7 (T) count 0 1 2 j 1 2 3 4
(i) 1<=3 (T) 3%1==0 (T) count++
(ii) 2<=3 (F) 3%2==0 (F)
(iii) 3<=3 (T) 3%3==0 (T) count++
if(count==2) (T) 2==2 S.O.P(3)

(2) 4<=7 (T) count 0 1 2 3 j 1 2 3 4 5
(i) 1<=4 (T) 4%1==0 (T) count++
(ii) 2<=4 (T) 4%2==0 (T) count++
(iii) 3<=4 (T) 4%3==0 (F)
(iv) 4<=4 (T) 4%4==0 (T) count++
(v) 5<=4 (F)
if(count==2)(F) 3==2

(3) 5<=7 (T) count 0 1 2 j 1 2 3 4 5 6
(i) 1<=5 (T) 5%1==0 (T) count++
(ii) 2<=5 (T) 5%2==0 (F)
(iii) 3<=5 (T) 5%3==0 (F)
(iv) 4<=5 (T) 5%4==0 (F)
(v) 5<=5 (T) 5%5==0 (T) count++
(vi) 6<=5 (F) inner loop end
if(count==2)(T) 2==2 S.O.P(5)

(4) 6<=7 (T) count 0 1 2 3 4 j 1 2 3 4 5 6 7
(i) 1<=6 (T) 6%1==0 (T) count++
(ii) 2<=6 (T) 6%2==0 (T) count++
(iii) 3<=6 (T) 6%3==0 (T) count++
(iv) 4<=6 (T) 6%4==0 (F)
(v) 5<=6 (T) 6%5==0 (F)
(vi) 6<=6 (T) 6%6==0 (T) count++
(vii) 7<=6 (F) inner loop end
if(count==2)(F) 4==2

(5) 7<=7 (T) count 0 1 2 j 1 2 3 4 5 6 7 8
(i) 1<=7 (T) 7%1==0 (T) count++
(ii) 2<=7 (T) 7%2==0 (F)
(iii) 3<=7 (T) 7%3==0 (F)
(iv) 4<=7 (T) 7%4==0 (F)
(v) 5<=7 (T) 7%5==0 (F)
(vi) 6<=7 (T) 7%6==0 (F)
(vii) 7<=7 (T) 7%7==0 (T) count++
(viii) 8<=7 (F) inner loop end
if(count==2) 2==2 (T) S.O.P(7)

(6) 8<=7 (F) outer loop end

22/3/22

Methods :

what is method ?

* method is a block of instruction which is used to perform a specific task

* It is used to transfer a data.

method Syntax :

[access modifier] [modifier] return type method name([data type var1, ... data type vn])

{

}

Technical words of method definition:

* method signature
* method declaration
* method definition

method signature :

* method name
* formal argument

method declaration:

* Access modifier
* modifier
* Return type
* Signature

method definition :

* method declaration
* method body / implementation / block.

Access modifiers :

Access modifiers are the keywords which is responsible for the accessibility of the method. ie, Access modifier is used to modify the accessibility of the method.

We have 4 levels of acess modifiers.

* private
* public
* protected
* default.

modifiers:

modifiers are the keywords which are responsible to modify the characteristics of the method.

eg:

* static
* abstract
* final
* synchronized
* volatile
* transient.

Return type:

* The method after execution can return a value back to the caller.

* Therefore it is mandatory to specify what type of data returned by the method in the method declaration statement, this is done with the help of return type.

Return type definition:

Return type is a data type which specify what type of data is returned by the method after execution.

types:

* void
* primitive data type
* Non-primitive data type

void:

* void is a data type which is used as a return type when the method returns nothing.
* It is a keyword in java.

NOTE:

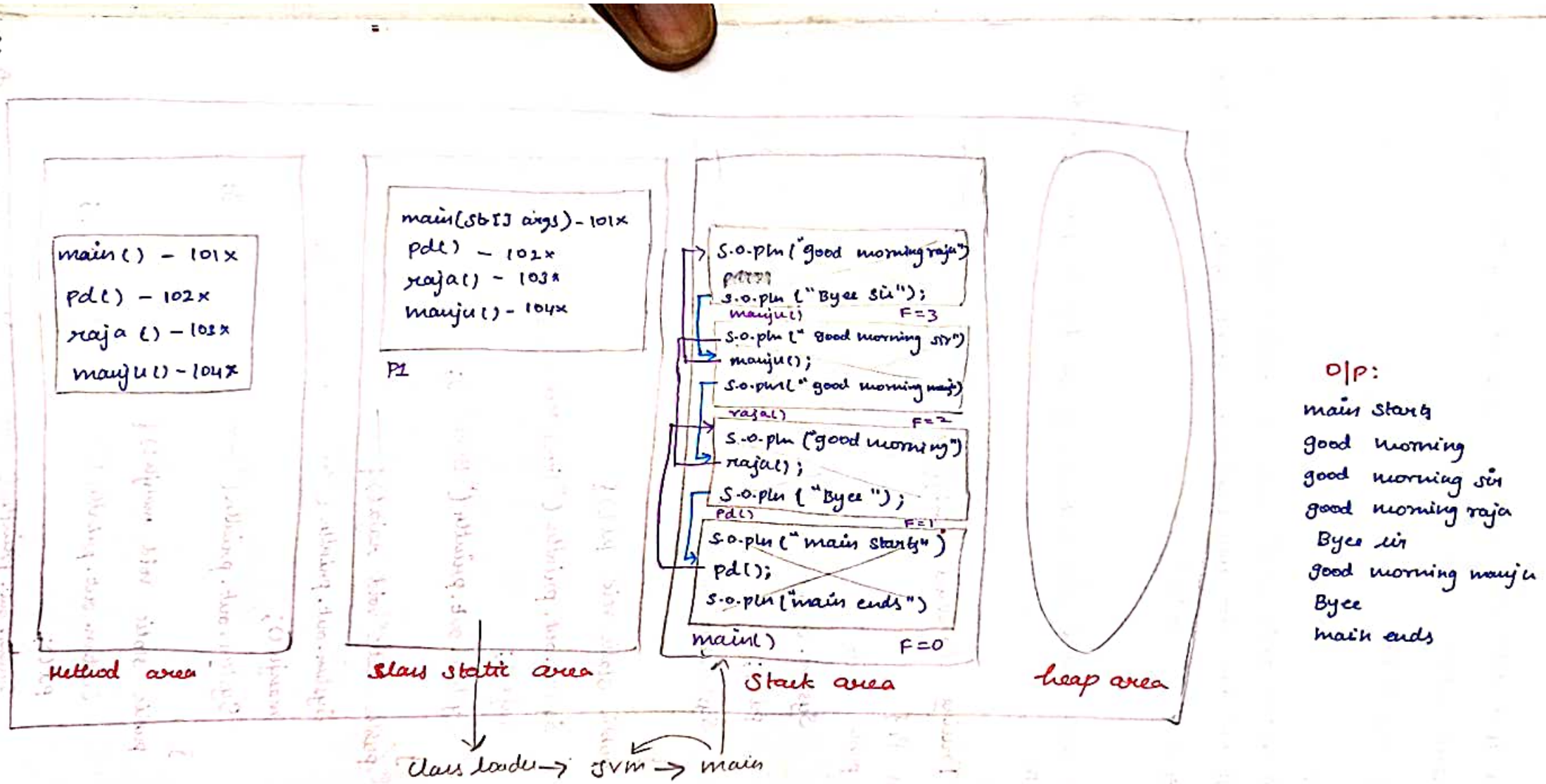
* A method can't create inside a another method.
* A class can have any number of methods. But, a class have only one main method.

NOTE:

we can't print void method. if we try to print we will get CTE.

Tracing (method calling)

```
class P1 {
    public static void main(String[] args) {
        System.out.println(" main starts ");
        pd();
        System.out.println(" main ends ");
    }
    public static void pd() {
        System.out.println(" Good Morning ");
        raja();
        System.out.println(" Byee ");
    }
    public static void raja() {
        System.out.println("Good morning sir ");
        manju();
        System.out.println(" Good morning manju ");
    }
    public static void manju() {
        System.out.println("Good morning raja");
        Pd();
        System.out.println(" Byee Sir ");
    }
}
```



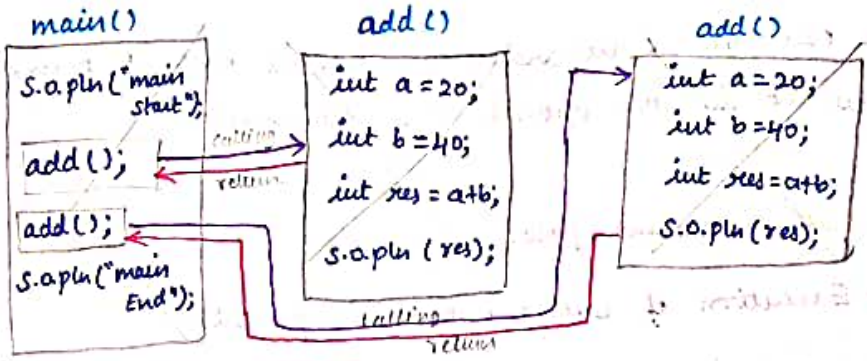
O|P:

main starts
good morning
good morning sir
good morning raja
Byee sir
good morning manju
Byee
main ends

```
class P2 {
    public static void main(String [] args) {
        System.out.println("Main start");
        add();
        add();
        System.out.println("Main End");
    }
    public static void add()
    {
        int a = 20;
        int b=40;
        int res = a+b;
        System.out.println(res);
    }
}
```

Tracing:

JVM calling main method.



OUTPUT:

```
Main start
60
60
Main End
```

Method Call Statement:

The statement which is used to call a method is known as method call statement.

Syntax:

```
method name([actual argument]);
```

NOTE:

A method will get executed only when it is called, we can call a method with the help of method call statement.

Types of methods Based on arguments:

The methods are classified into two based on arguments.

* no argument method
* Parameterized method.

NOTE:

We can call a no argument method without passing actual argument in the method call statement.

Method call statement flow:

* Execution of calling method is paused
* control is transferred to the called method
* Execution of called method is begins
* Once the execution of called method is completed the control is transferred back to the calling method.
* Execution of calling method resumes.

Calling Method Vs Called Method.

Calling Method:

The method is which is trying to call another method is known as calling method (caller)

Called method:

The method which is being called by the caller is known as called method.

Main Method

main method:

[The execution of the program always starts from main method and ends at main method only]

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

}
```

Purpose of main method:

Start the execution
control the flow of the execution
End of execution

NOTE:

* A method can be executed only when it is called, we can call a method any number of times, therefore it is said to be code reusability.

* Main method is always called by JVM

Example :

WAJP to create an application called calculator

```
class Calculator {
    public static void add(int a, int b) {
        int res = a+b;
        System.out.println("The addition of two number is " + res);
    }
    public static void sub(int a, int b) {
        int res = a-b;
        System.out.println("The difference of two number is" + res);
    }
    public static void multiply(int a, int b) {
        int res = a*b;
        System.out.println("The product of two number is " + res);
    }
    public static void division(int a, int b) {
        int res = a/b;
        System.out.println("The division of two number is " + res);
    }
}
import java.util.Scanner;
class CalculatorDriver {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("1. Addition \n2. Subtraction \n3. Multiplication
                            \n4. division ");
        System.out.println("choose the operation which you want to
                            perform");
        int choice = s.nextInt();
        switch(choice)
        {
```

```
case 1 :
{ System.out.println("Enter the first number");
  int a = S.nextInt();
  System.out.println("Enter the second number");
  int b = S.nextInt();
  Calculation.add(a,b);
}
 break;
case 2 :
{
  System.out.println("Enter the first number");
  int a= S.nextInt();
  System.out.println("Enter the second number");
  int b=S.nextInt();
  Calculation.sub(a,b);
}
 break;
 case 3:
 {
  System.out.println("Enter the first number");
  int a = S.nextInt();
  System.out.println("Enter the second number");
  int b= S.nextInt();
  Calculation.multiply(a,b);
 }
 break;
 case 4:
 {
  System.out.println("Enter the first number");
  int a = S.nextInt();
  System.out.println("Enter the second number");
  int b= S.nextInt();
  Calculation.division(a,b);
 }
}
}
}
```

java class CalculatorDriver (executing)

OUTPUT :

1. Addition
2. Subtraction
3. Multiplication
4. Division.

2

Enter the first number

20

Enter the second number

9

The difference of two number is 11

Types of Method :

Based on number of arguments, methods can be classified into two types,

No argument method
Parametrized method

No argument method :

A method which does not have formal argument is known as no argument method.

Example :

```
public static void demo()
{
  System.out.println("demo - no argument method");
}
```

Parameterized method :

* The method which has formal argument is known as parameterized method.

* Parameterized methods are used to accept the data.

Formal argument :

A variable which is declared in a method declaration is known as formal argument.

Actual argument :

The values passed in the method call statement is known as actual argument.

Rules to call the parameterized method :

* The number of actual argument should be same as the number of formal argument.

* The type of corresponding actual argument should be same as the type of formal argument, if not compiler tries implicit conversion if it is not possible then we will get CTE

Session 5 : Activity

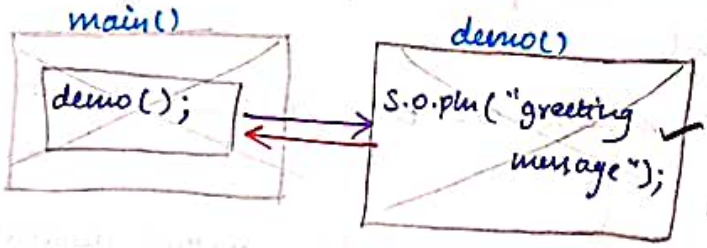
1. Design a method which is used to print "Greeting message" when it is called.

```
class E1 {
    public static void main(String[] args) {
        demo();
    }
    public static void demo() {
        System.out.println("Greeting message");
    }
}
```

OUTPUT:

Greeting message

Tracing:



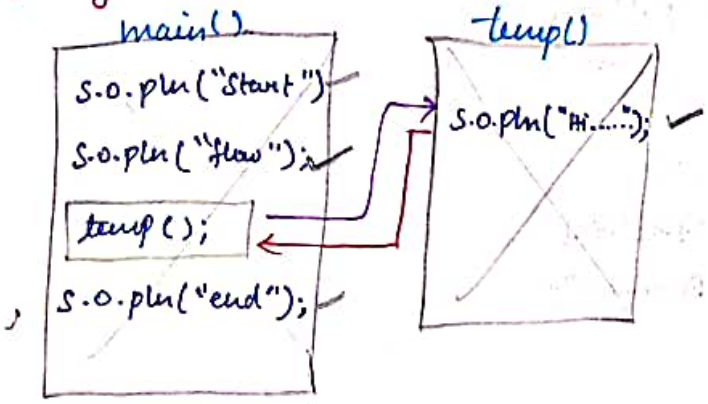
2. Do tracing on book for the given code.

```
class Demo
{
    public static void temp() {
        System.out.println("Hi..... ");
    }
    public static void main(String[] args) {
        System.out.println("start");
        System.out.println("flow");
        temp();
        System.out.println("end");
    }
}
```

OUTPUT :

Start
flow
Hi.....
end

Tracing :



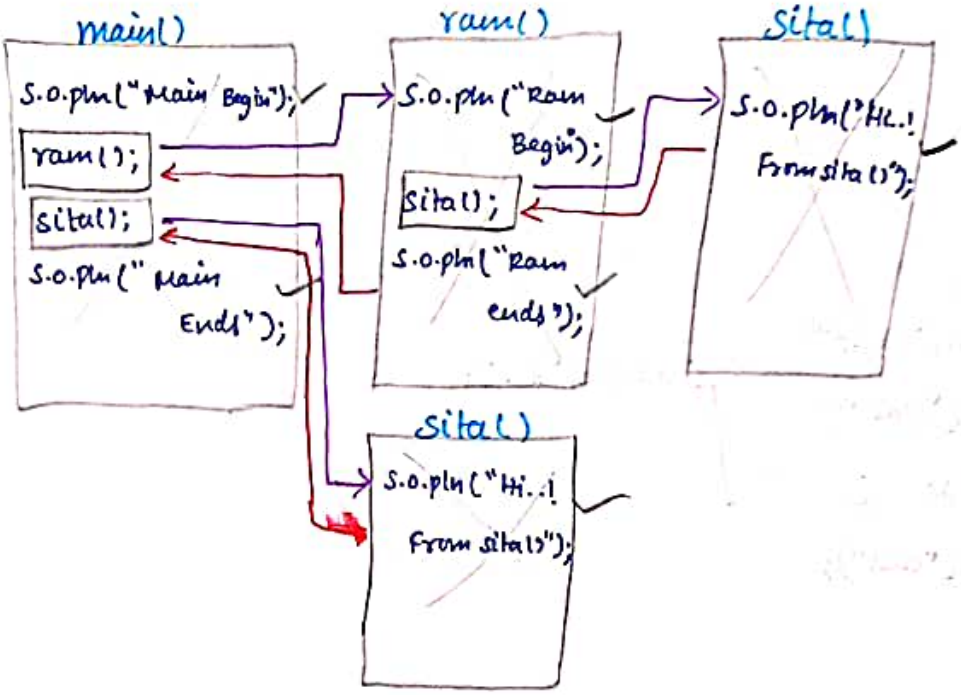
3. Do tracing for the code.

```
class Demo
{
   public static void sita(){
      System.out.println("Hi..! From sita()");
   }
   public static void ram(){
      System.out.println("Ram Begins");
      sita();
      System.out.println("Ram Ends");
   }
   public static void main(String[] args){
      System.out.println("Main Begins");
      ram();
      sita();
      System.out.println("Main Ends");
   }
}
```

OUTPUT:

Main Begin
Ram Begin
Hi..! From sita()
Ram Ends
Hi..! From sita()
Main Ends

Traing:



22-HAE

Return Statement:

A method after execution will return a data back to the caller with the help of return statement.

return:

* return is a keyword
* It is a control transfers statement.
* When the return statement is executed, the execution of the method is terminated and control is transferred to the calling method.

Steps to use return statement:

Step 1: provide a return type for a method (It should not be void)

Step 2: Use the return statement in the value to be returned.

Rule:

The type specified as return type should be same as the type of value passed in a return statement.

Method Overloading

If more than one method is created with the same name but different formal arguments in the same class are known as method overloading.

Example:

```
java.lang.Math;
abs(double d)
abs(float f)
abs(int i)
abs(long l)
```

} overloaded method

There are some of the overloaded method (Method with same name different formal arguments) implemented in java.lang.Math class.

Example:

```
test();          ---> void test();
test(10);        ---> int test(int i);
test(10.5f);     ---> float test(float f);
test(10,10.5f)   ---> void test(int i, float f)
test(10.5f,10);  ---> void test(float f, int i)
test('a');
test(10,10);     //CTE
```

STATIC AND STATIC MEMBERS

STATIC : eg: building the house.

* Static is a keyword.
* It is a modifier
* Any member of a class is prefixed with static modifier then it is known as static member of a class.
* Static members are also known as class members.

NOTE:

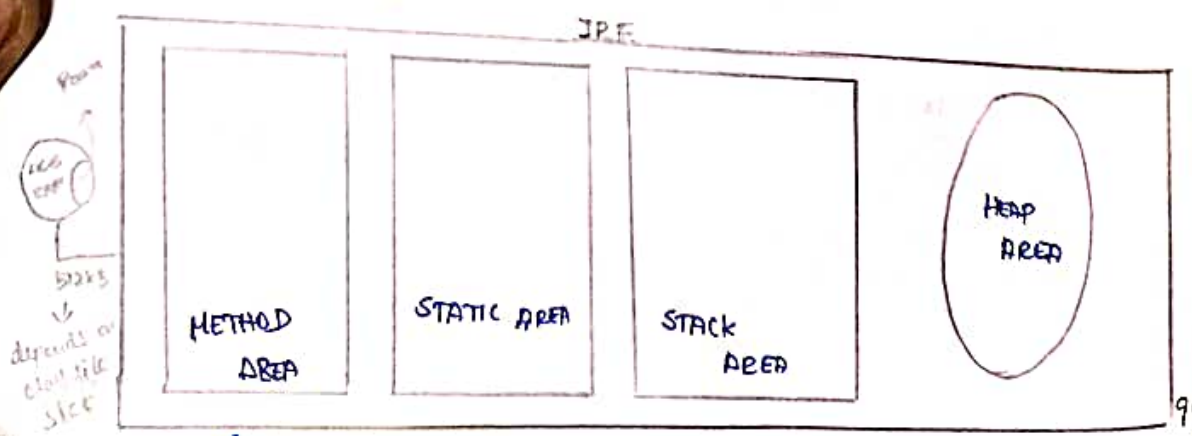
Static members can be prefixed only for a class members (members declared in a class).

STATIC MEMBERS :

* Static method
* Static variable
* Static initializer

JAVA RUNTIME MEMORY

* To execute the java program a portion of memory in RAM is allocated for JRE
* In that portion of memory allocated, we have different range of memory, hence they are classified as follows,
  * Method area
  * class static area
  * Stack area
  * Heap area



METHOD AREA:

All the methods of a class will be stored in a method area (instruction of the methods).

CLASS STATIC AREA:

* For every class there is a dedicated block of memory is created in the class static area

* The static members of the class will be allocated inside the memory created for the class.

STACK AREA:

* Stack area is used for execution of instruction

* For every method that is under execution a block of memory is created in this stack area which is known as frame.

* Once the execution of a method is completed the fram is removed.

HEAP AREA:

* In a heap area a block of memory is created for the instance of class

* Every block of memory created with the help of reference

* All the non static member of a class will be allocated inside this block of memory.

* Therefore we can access the non-static member with the help of reference.

STATIC METHOD:

A method prefixed with static modifier is known as static method.

CHARACTERISTICS:

* Static method block is stored in the method area and reference of the static method is stored inside the class static area

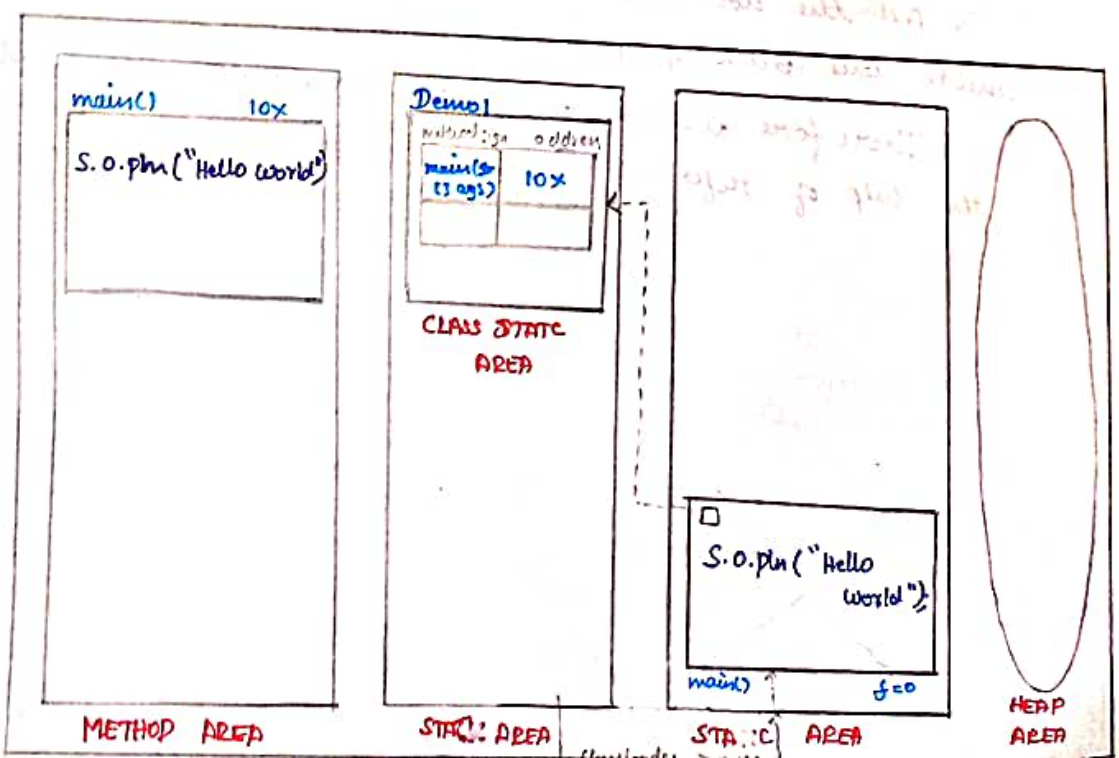
* We can use the static method with or without creating object of the class.

* We can use the static method with the help of class name

* A static method of the class can be used in any class with the help of class name.

Example:

```
class Demo1{
    public static void main(String [] args){
        System.out.println("Hello world");
    }
}
```



O/P:

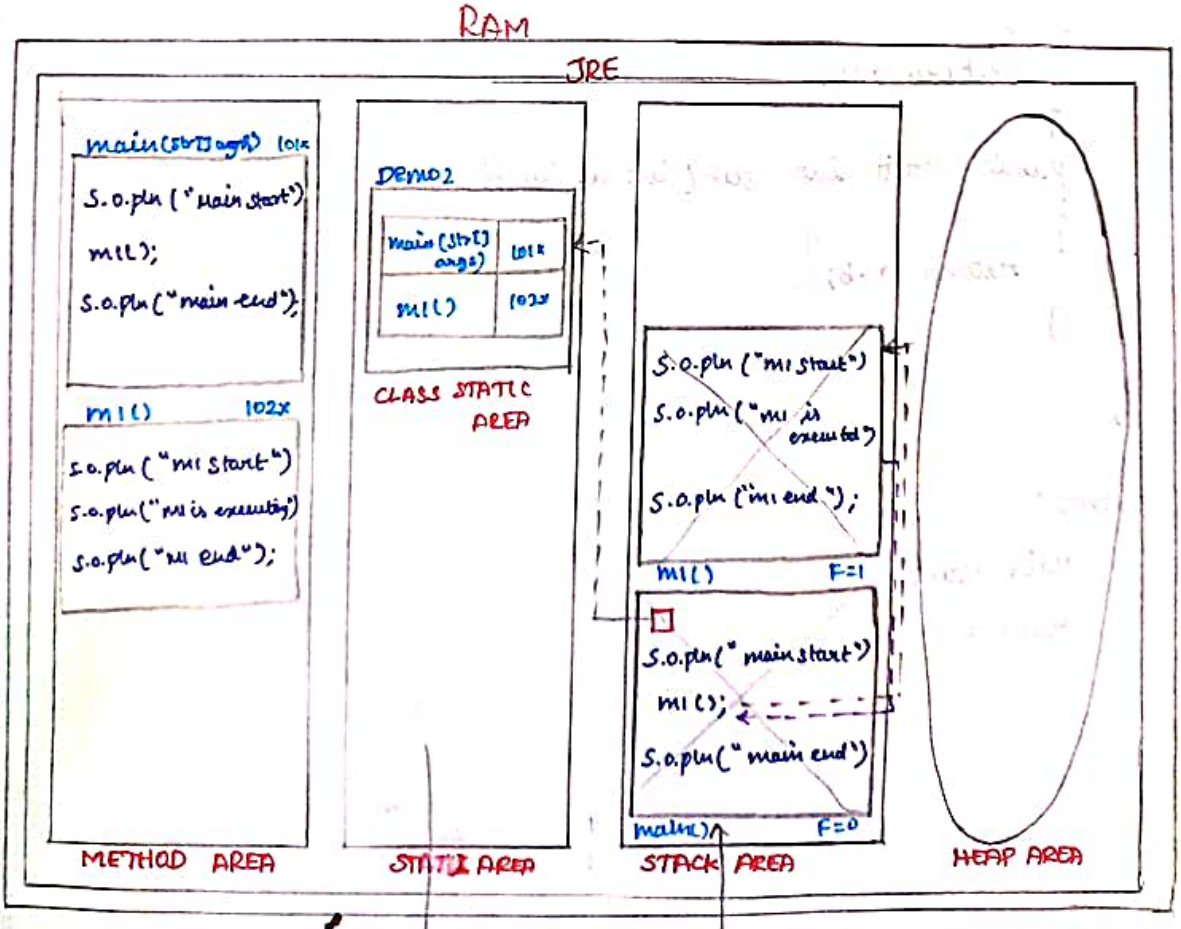
Hello world

Example:

```
class Demo2
{
    public static void main (String[] args)
    {
        System.out.println (" Main start");
        m1();
        System.out.println ("Main end ");
    }
    public static void m1()
    {
        System.out.println("M1 start");
        System.out.println("M1 is executing");
        System.out.println("M1 End");
    }
}
```

→ main method

→ Static method m1()



OUTPUT:

Main start

M1 start

M1 is executing

M1 End

Main end.

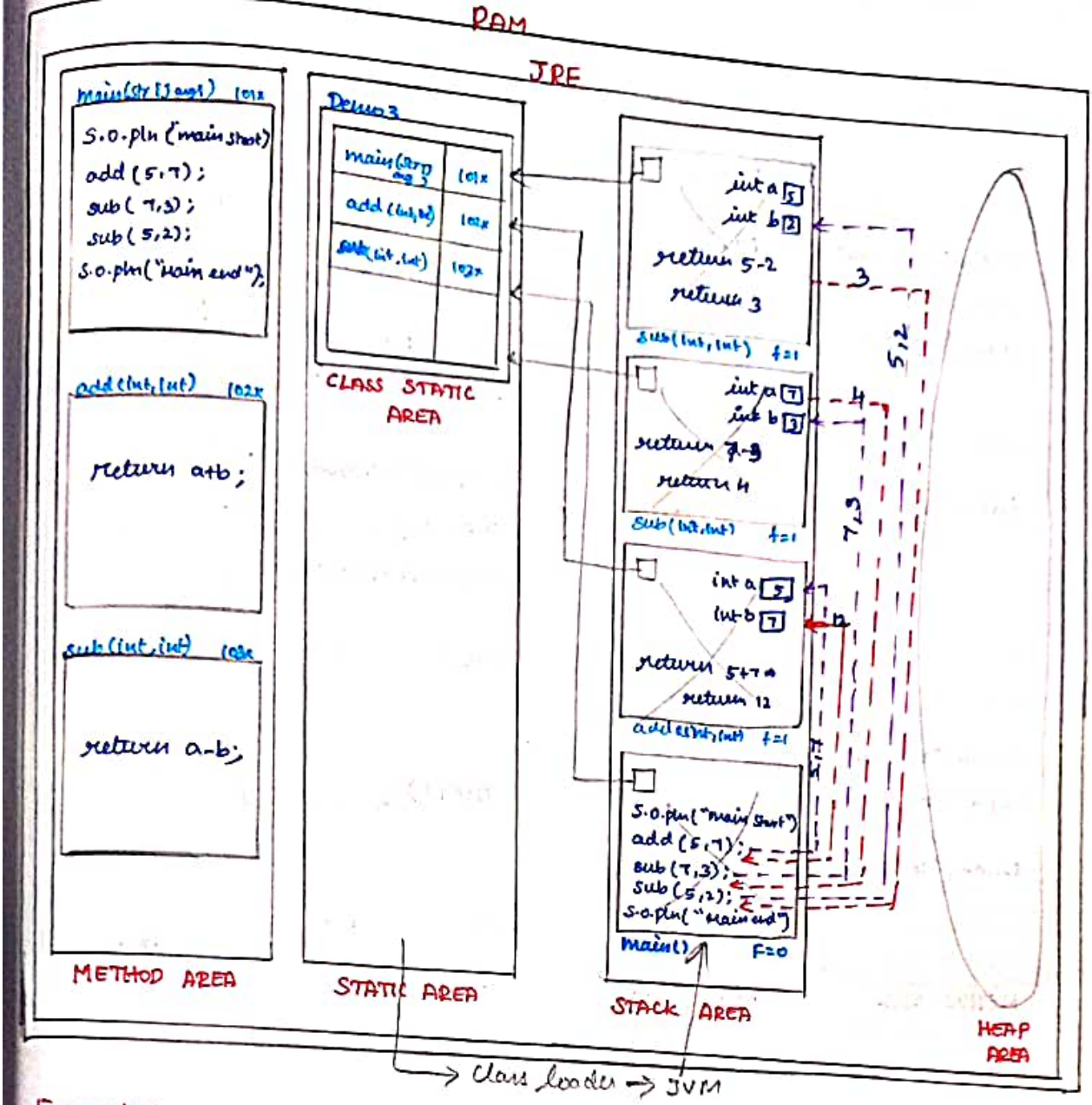
Example:

```
class Demo3
{
  public static void main(String[] args)
  {
    System.out.println("Main Start");
    add(5,7);
    sub(7,3);
    sub(5,2);
    System.out.println("Main End");
  }
  public static int add(int a, int b)
  {
    return a+b;
  }
  public static int sub(int a, int b)
  {
    return a-b;
  }
}
```

OUTPUT:

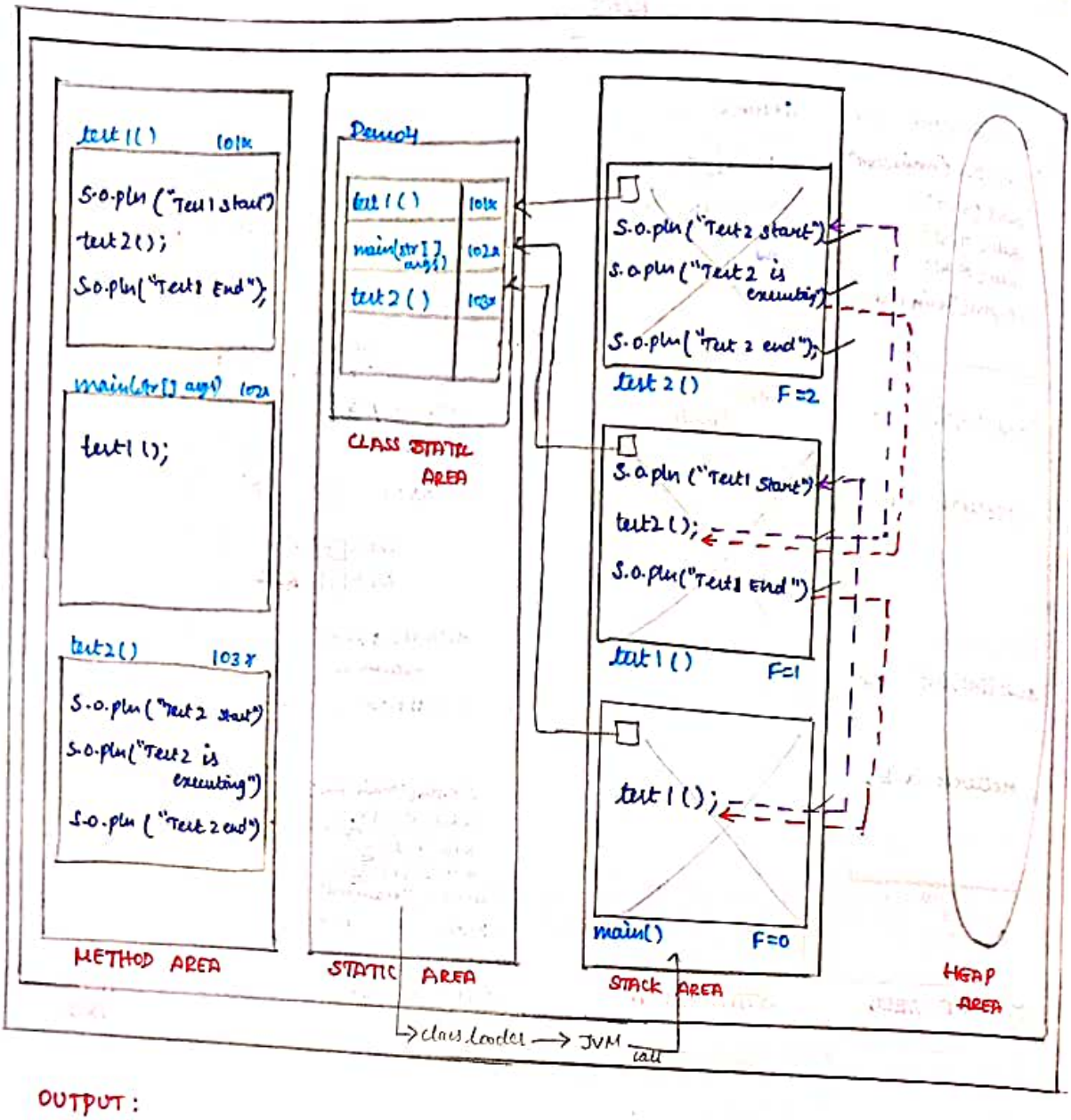
Main Start

Main End



Example:

```
class Demo4
{
   public static void test1():
   {
      System.out.println("Test 1 start");
      test();
      System.out.println("Test 1 End");
   }
   public static void main(String[] args)
   {
      test1();
   }
   public static void test2()
   {
     System.out.println("Test 2 start");
     System.out.println("Test 2 is executing");
     System.out.println("Test 2 End");
   }
}
```



OUTPUT :

Test1 start

Test2 start

Test2 is executing

Test2 is end

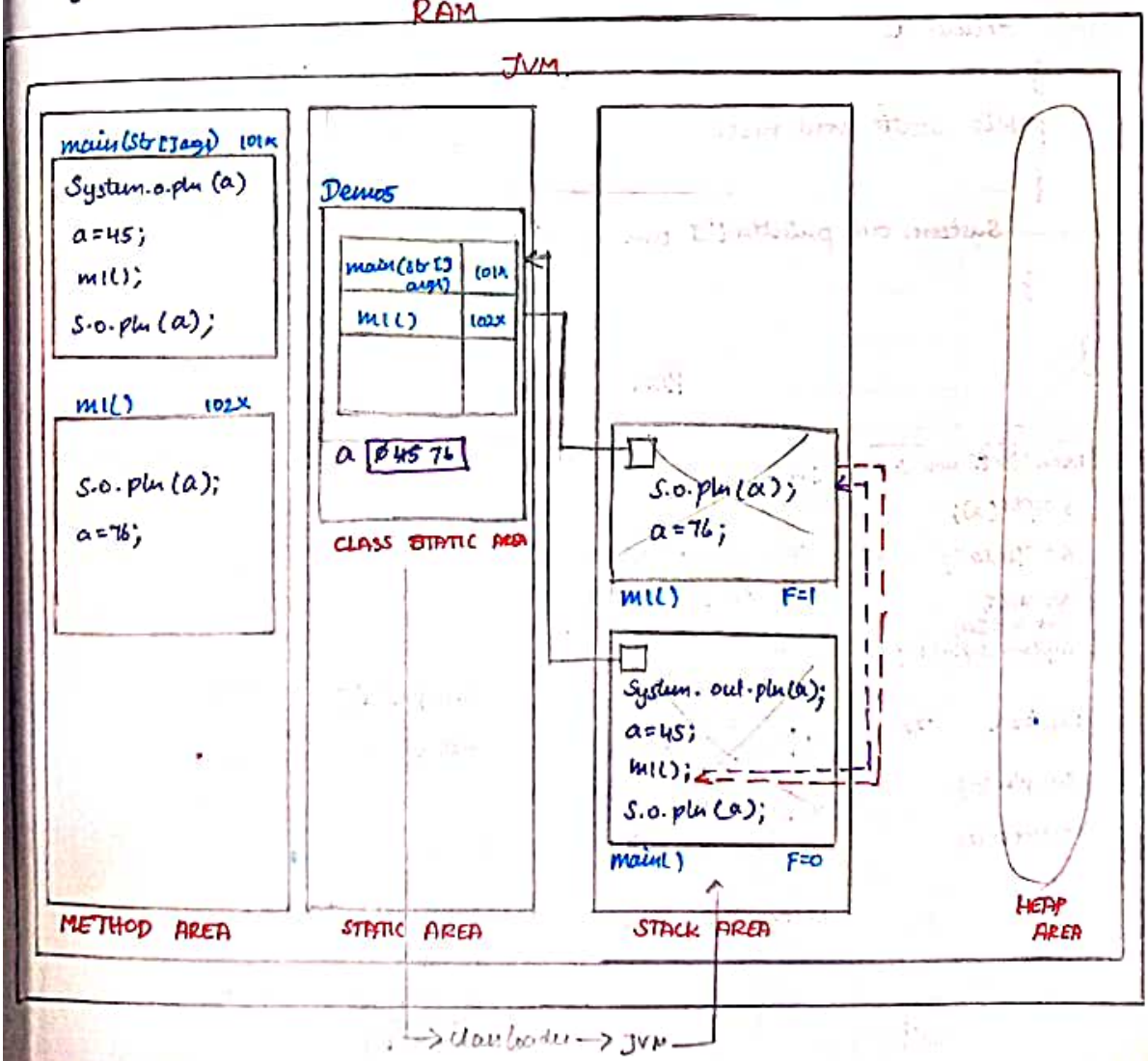
Test1 is end

Example:

```
class Demo5
{
   static int a;
   public static void main(String[] args)
   {
      System.out.println(a);
      a=45;
      m1();
      System.out.println(a);
   }
   public static void m1()
   {
     System.out.println(a);
     a=76;
   }
}
```

OUTPUT:

0
45
76

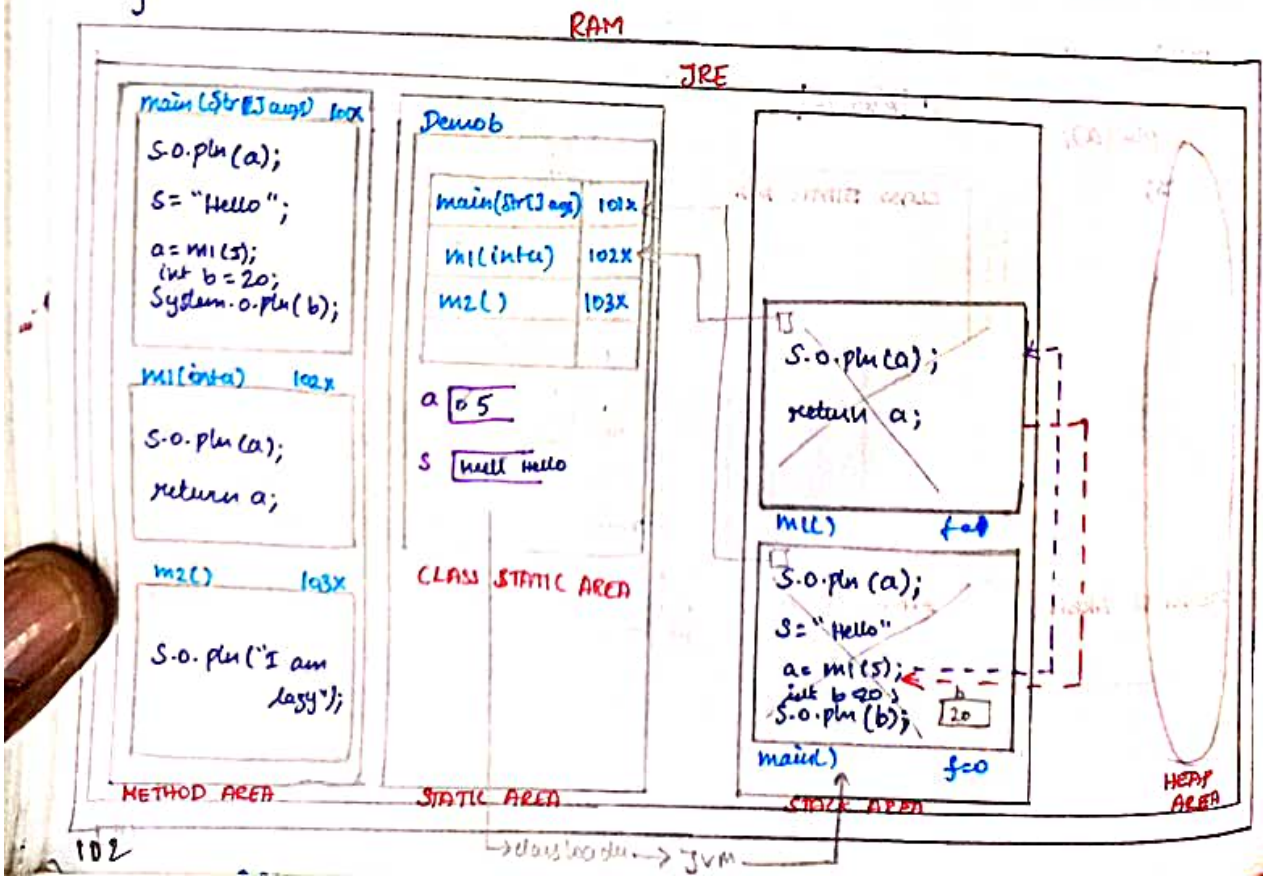


Example:

```
class Demo6
{
  static int a;
  static String s;
  public static void main(String[] args)
  {
    System.out.println(a);
    s = "Hello";
    a = m1(5);
    int b = 20;
    System.out.println(b);
  }
  public static int m1(int a)
  {
    System.out.println(a);
    return a;
  }
  public static void m2()
  {
    System.out.println("I am lazy");
  }
}
```

OUTPUT:

0
5
20

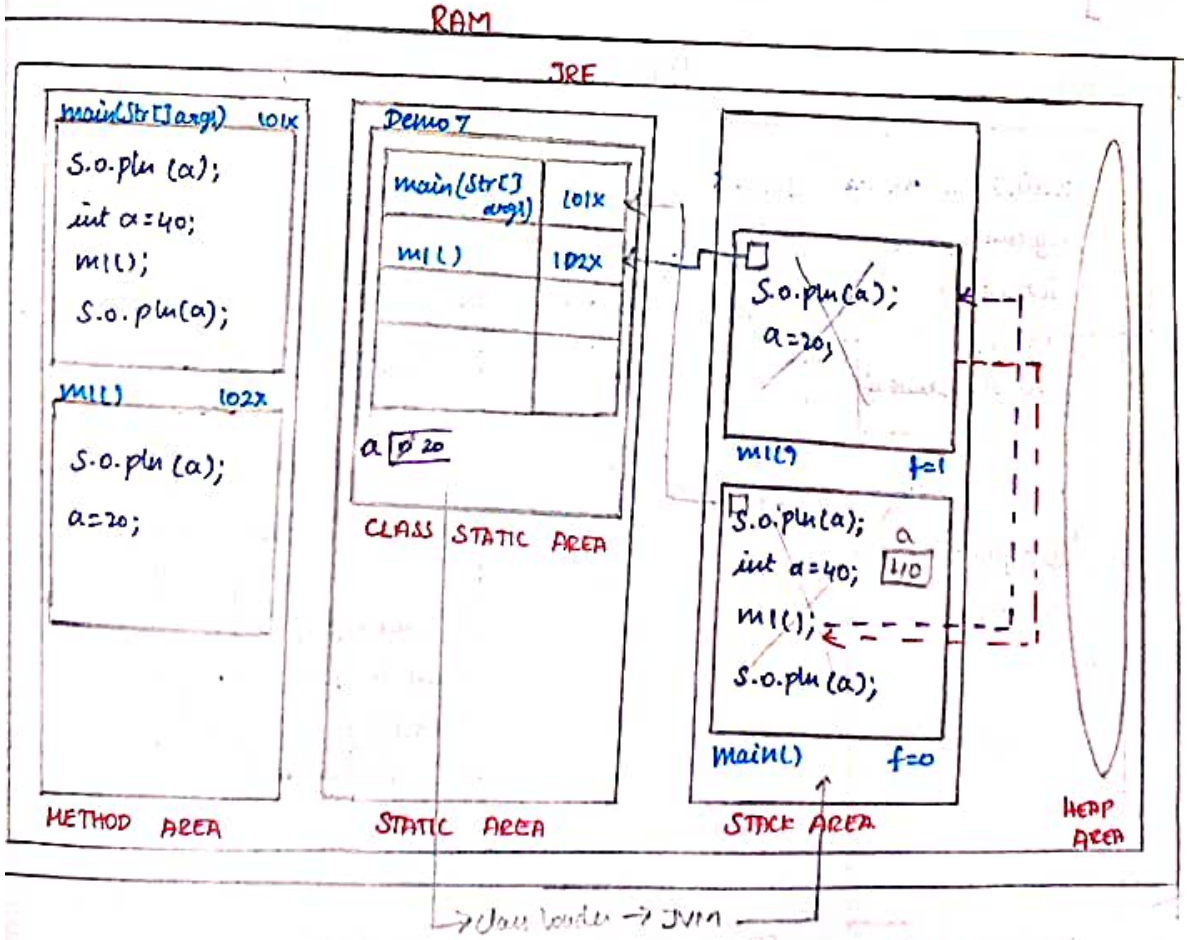


EXAMPLE:

```
class Demo7
{
    static int a;
    public static void main(String[] args)
    {
        System.out.println(a);
        int a=40;
        m1();
        System.out.println(a);
    }
    public static void m1()
    {
        System.out.println(a);
        a=20;
    }
}
```

OUTPUT:

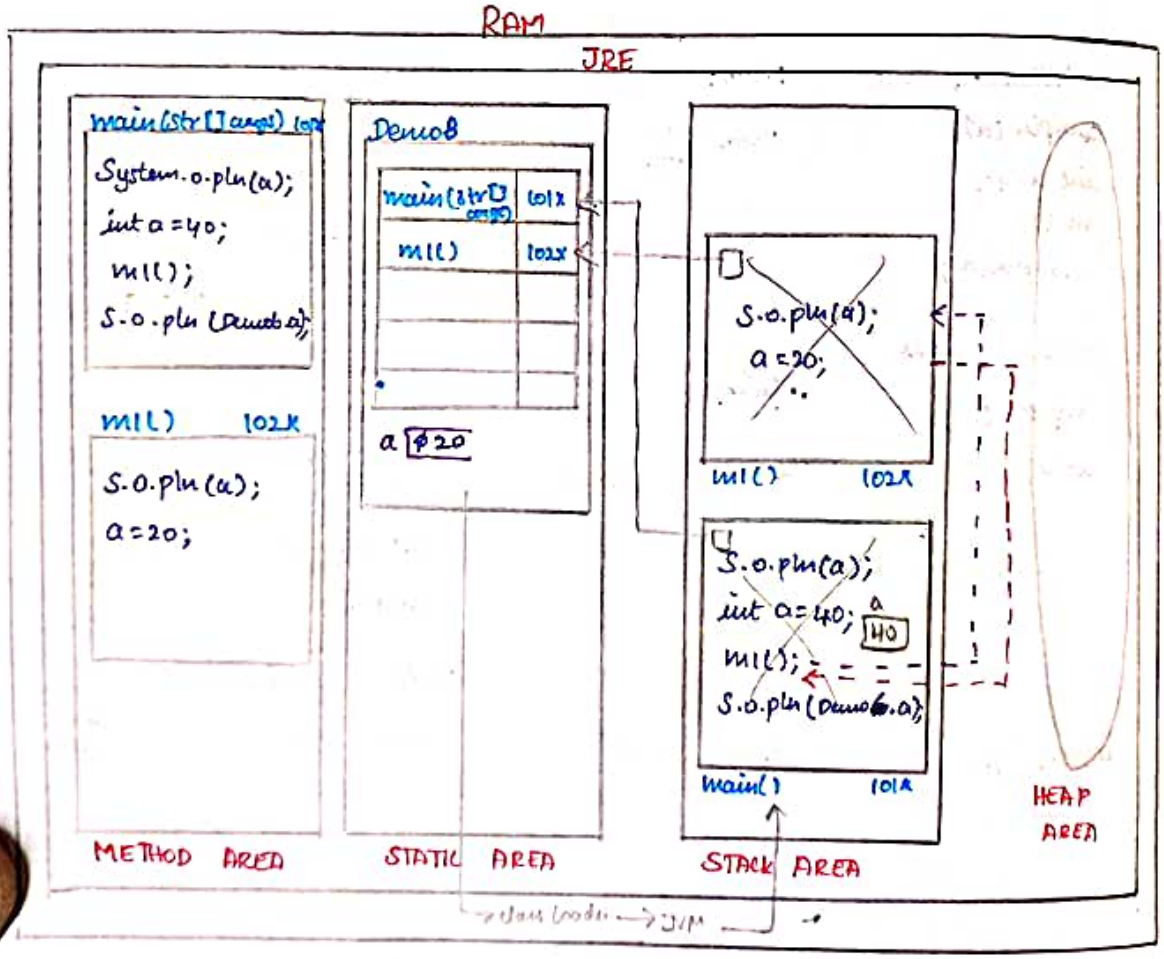
0
0
40



EXAMPLE:

```
class Demo8
{
  static int a;
  public static void main(String[] args)
  {
    System.out.println(a);
    int a=40;
    m1();
    System.out.println(Demo8.a)
  }
  public static void m1()
  {
   System.out.println(a);
   a=20;
  }
}
```

OUTPUT:
0
0
20



STATIC VARIABLE:

Variable declared in a class block and prefixed with static modifier is known as static variable.

CHARACTERISTICS:

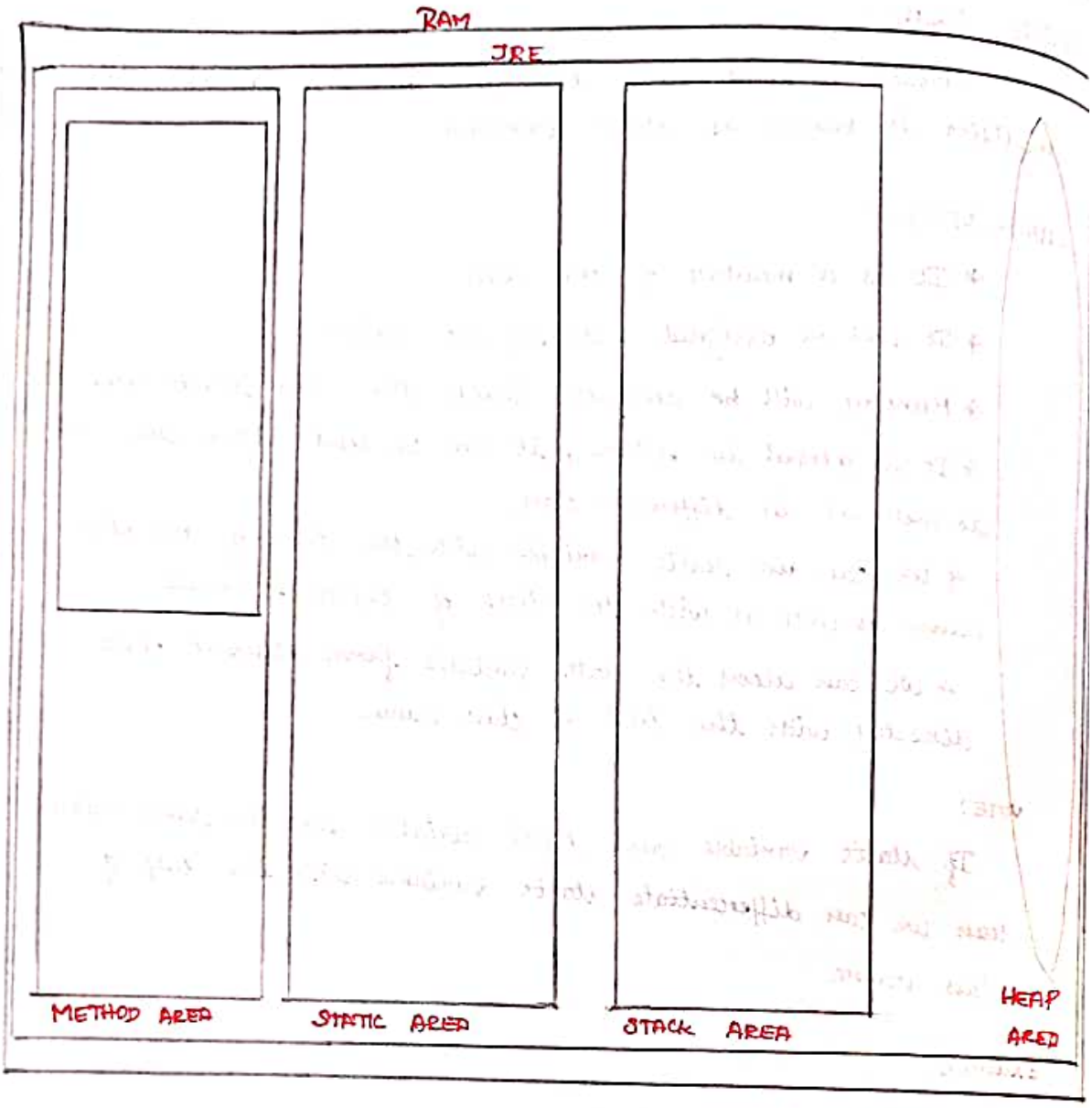
* It is a member of the class.
* It will be assigned with default value.
* Memory will be allocated inside the class static area.
* It is global in nature, it can be used within the class as well as in different class.
* We can use static variable with the help of the class name as well as with the help of object reference.
* We can access the static variable from different class directly with the help of class name.

NOTE:

If static variable and local variable are in same name then we can differentiate static variable with the help of class name.

Example:

```
class Demo7
{
  static int a;
  public static void main(String[] args)
  {
    int a=20;
    System.out.println(a);
    Demo6.a=70;
    m1();
    a=29;
    System.out.println(a++);
    System.out.println(a);
    System.out.println(Demo7.a);
  }
  public static void m1()
  {
      a++;
  }
}
```


RAM
JRE
METHOD AREA
STATIC AREA
STACK AREA
HEAP AREA

1/3/22

STATIC INITIALIZER:

we have two types of static initializers. They are,

→ Single line static initializer

→ Multi line static initializer

Single line Initializers:

Syntax:

```
static datatype variable = value/Expression;
```

Multiline static Initializers:

Syntax:

```
static
{
  //statements;
}
```

characteristics:

* Static initializers gets executed implicitly during the loading process of the class.

* A class can have more than one static initializers they execute top to bottom order.

purpose of static initializers:

* Static initializers are used to execute the startup instructions

* As the static blocks get executed before the actual execution

Loading process of the class:

* A block is created for a class in the static pool, it can be accessed with the help of class name.

* All the method definition are loaded in method area and if the method is static then the reference of that method is stored inside the class block.

* If class has any static variable they are loaded in the class static area with default value

* If the class has any static initializers they are executed from top to bottom order.

* The loading process of the class is completed then JVM will call the main method of initial loading class.

JVM: NOTE:

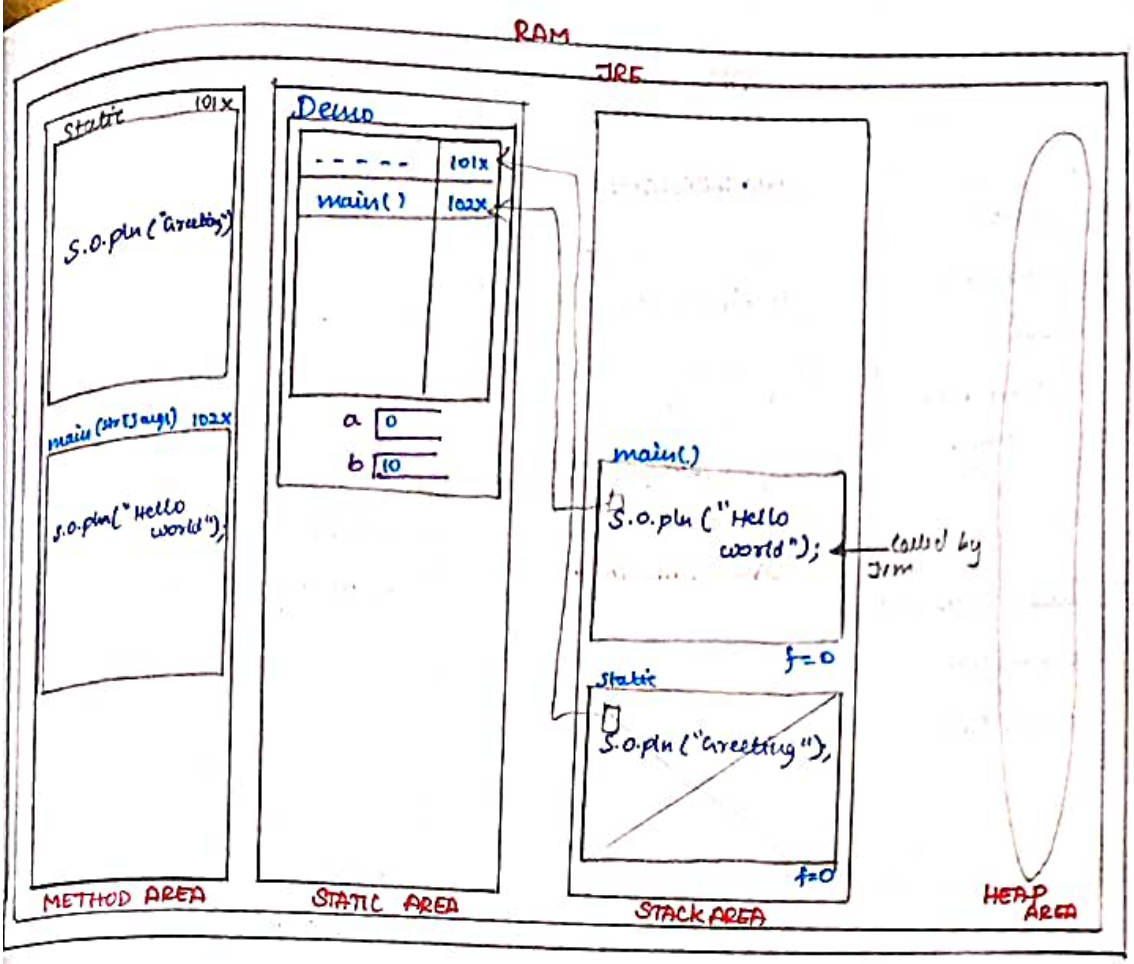
JVM will call only the main method of initial loaded class.

Example:

```
class Demo{
  static int a;
  static int b=10;
  static {
     System.out.println(" Greetting ");
  }
  public static void main(String[] args)
  {
    System.out.println("Hello world");
  }
}
```

OUTPUT:

```
Greetting
Hello word
```

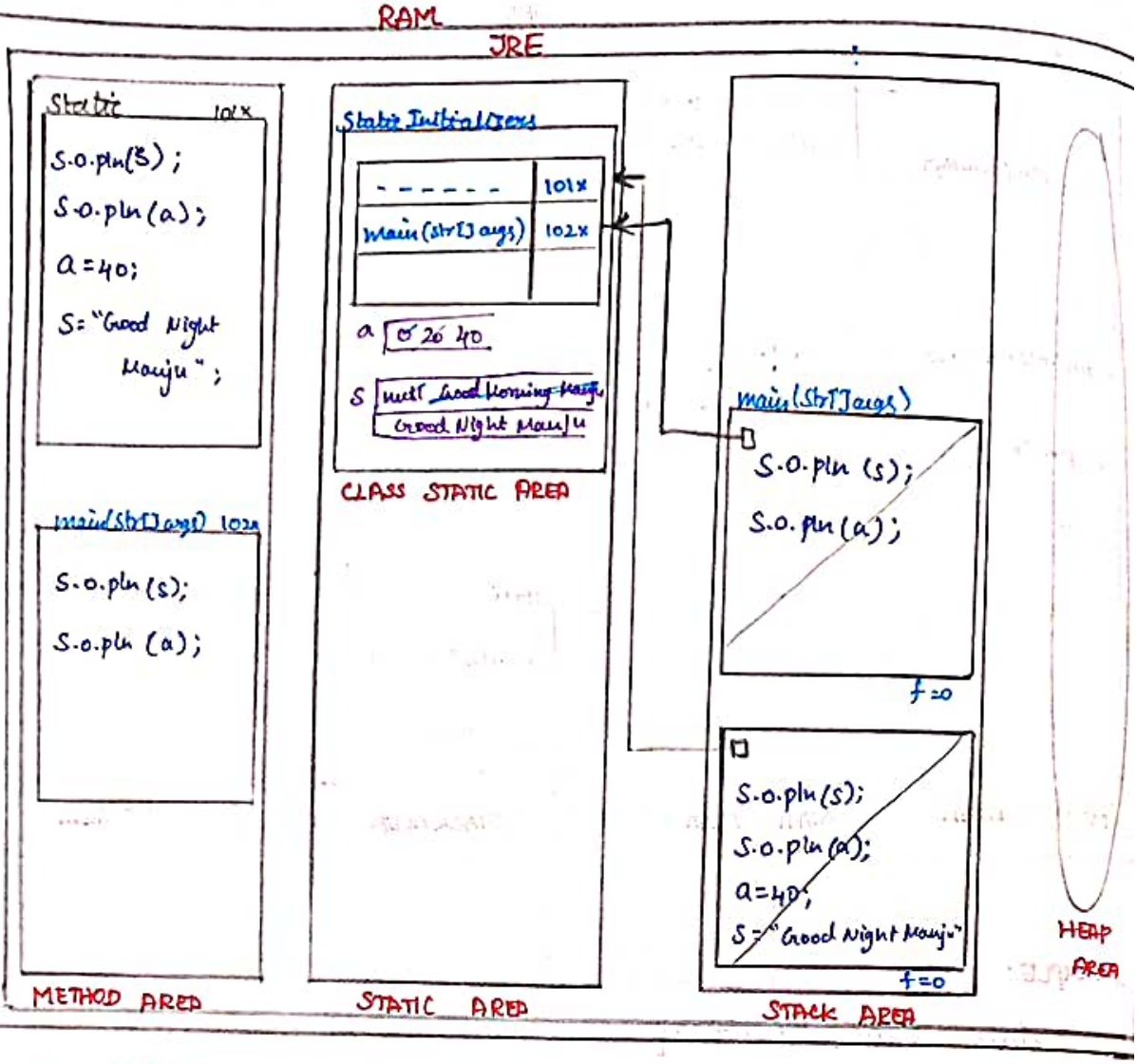


EXAMPLE:

```
class StaticInitializers{
    static int a=20;
    static String s="Good Morning Manju";
    static
    {
        System.out.println(s);
        System.out.println(a);
        a=40;
        s="Good Night Manju";
    }
    public static void main(String[] args)
    {
        System.out.println(s);
        System.out.println(a);
    }
}
```

OUTPUT:

```
Good Morning Manju
20
Good Night Manju
40
```

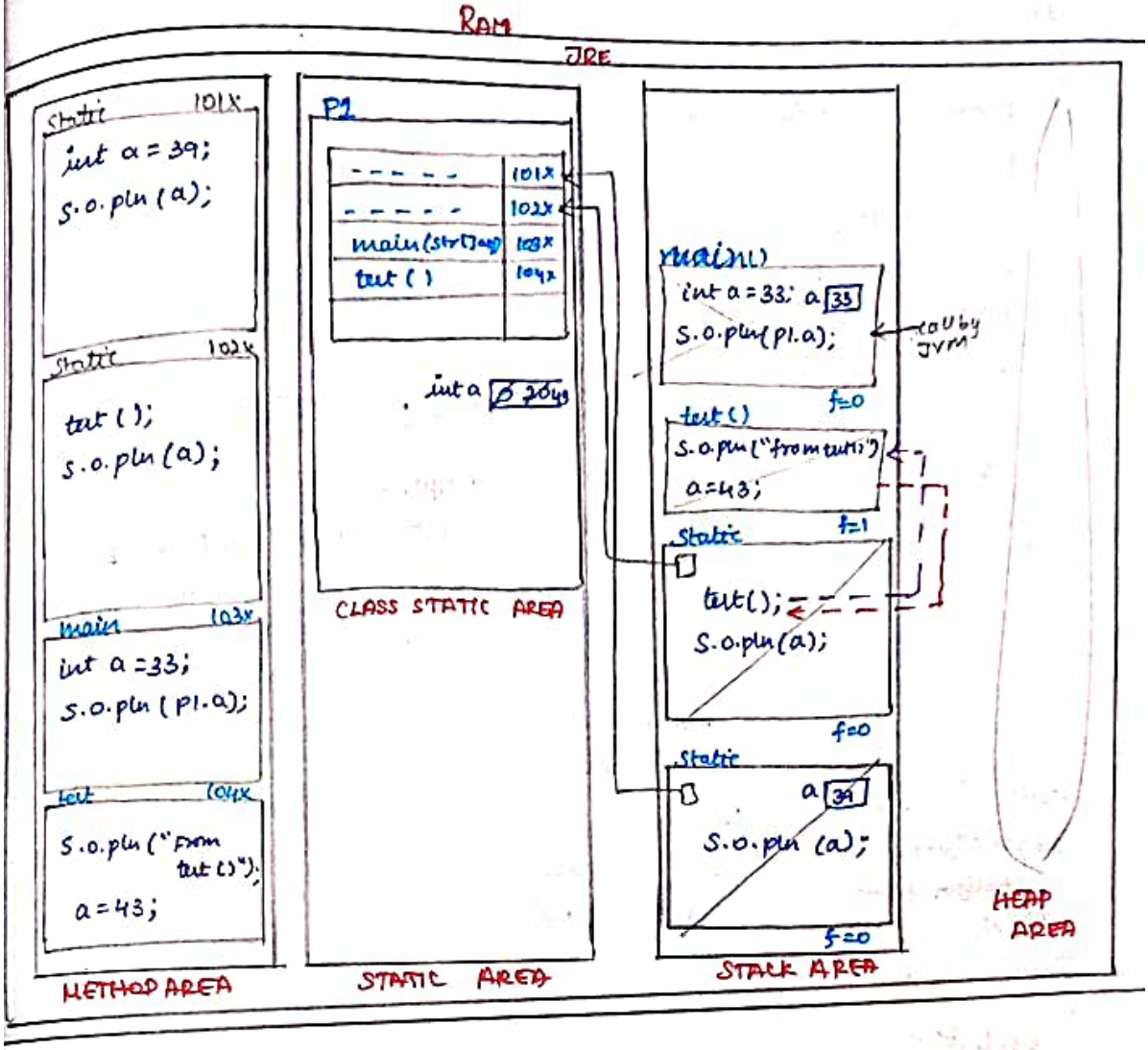


EXAMPLE:

```
class P1
{
  static {
      int a = 39;
      System.out.println(a);
  }
  static int a=20;
  public static void main (String[] args)
  {
      int a = 33;
      System.out.println (P1.a);
  }
  public static void test ()
  {
    System.out.println ("From test()");
    a=43;
  }
  static {
     test();
     System.out.println (a); } }
```

OUTPUT:

```
39
From test()
43
43
```



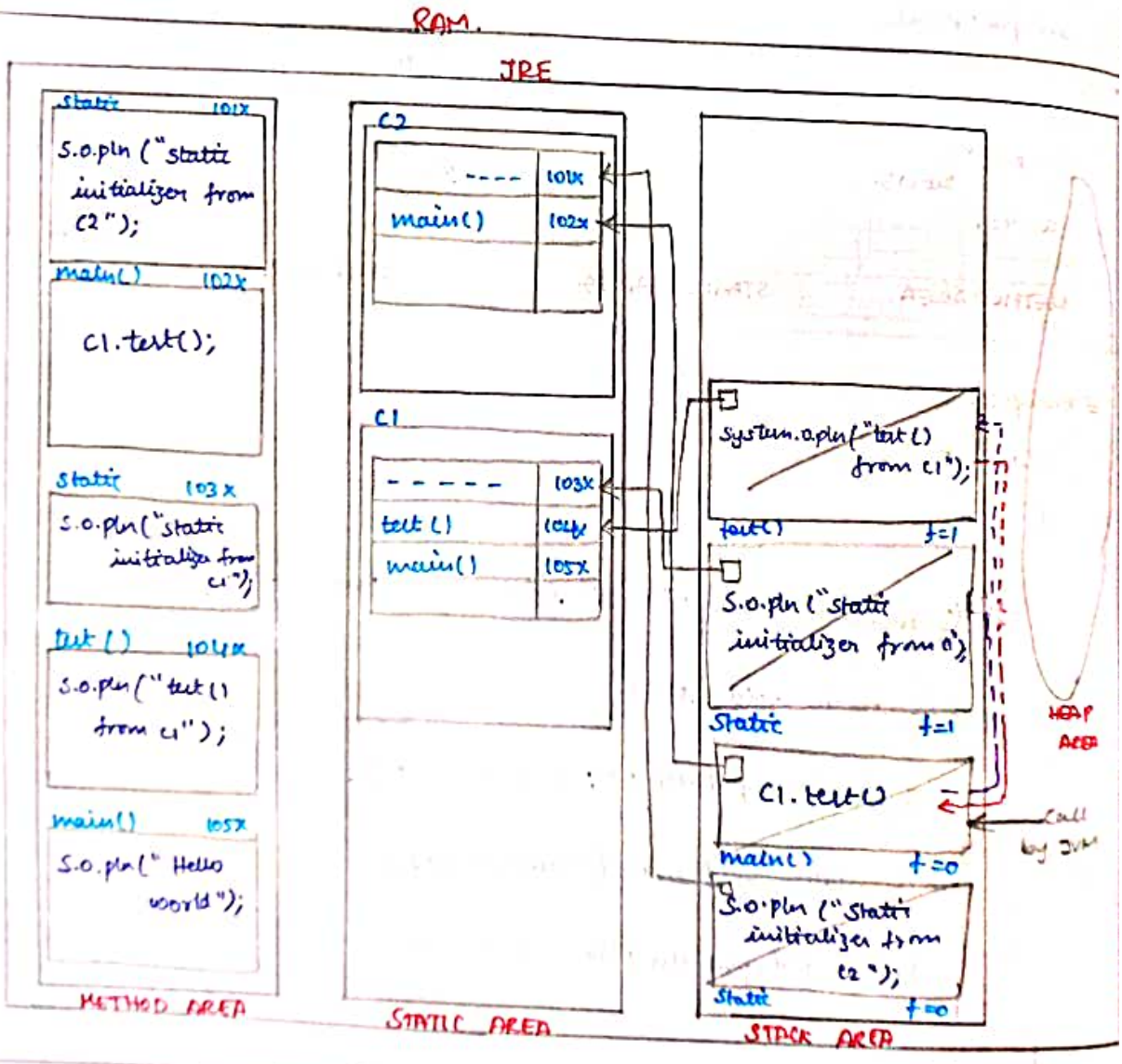
Example :

```
class C1
{
   static
   {
     System.out.println ("Static initializer from C1");
   }
   public static void test()
   {
      System.out.println ("test() from C1");
   }
   public static void main (String[] args)
   {
      System.out.println ("Hello world");
   }
}
```

```
class C2
{
    public static void main(String[] args)
    {
        C1.test();
    }
    static
    {
        System.out.println("Static initializer from C2");
    }
}
```

Output :

Static initializer from C2
Static initializer from C1
test() from C1

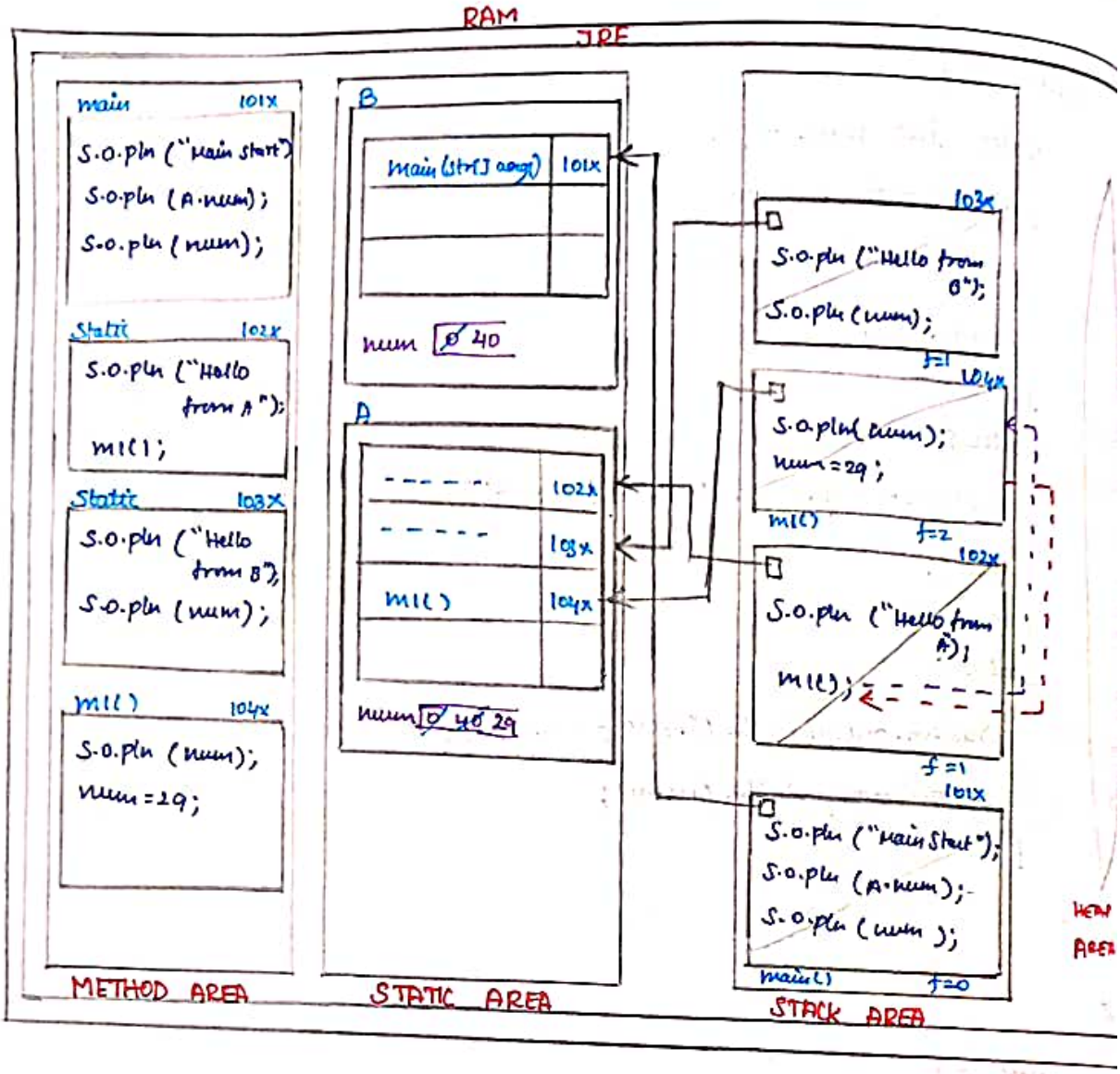


```
class A {
  static int num = 40;
  public static void m1 {
    System.out.println(num);
    num = 29;
  }
  static {
    System.out.println("Hello from A");
    m1();
  }
  static {
    System.out.println("Hello from B");
    System.out.println(num);
  }
}

class B {
  static int num = 40;
  public static void main(String[] args) {
    System.out.println("Main Start");
    System.out.println(A.num);
    System.out.println(num);
  }
}
```

output :

```
main Start.
Hello from A
40
Hello from B.
29
29
40
```


RAM
JRE
main 101x
S.O.pln ("main start")
S.O.pln (A.num);
S.O.pln (num);
Static 102x
S.O.pln ("Hello from A");
m1();
Static 103x
S.O.pln ("Hello from B");
S.O.pln (num);
m1() 104x
S.O.pln (num);
num = 29;
B
main (Str[] args) 101x
num 40
A
102x
103x
m1() 104x
num 0 40 29
103x
S.O.pln ("Hello from B");
S.O.pln (num);
f=1
104x
S.O.pln(num);
num=29;
m1()
f=2
102x
S.O.pln ("Hello from A");
m1();
f=1
101x
S.O.pln ("main Start");
S.O.pln (A.num);
S.O.pln (num);
main()
f=0
METHOD AREA
STATIC AREA
STACK AREA
HEAP AREA

OBJECT

OBJECT:

* Any substance which has existence in the real world is known as an object.

* Every object will have attribute and behaviours.

OBJECT IN JAVA:

* According to object oriented programming object is a block of memory created in the heap area during the runtime it represents a real world object

* A real world object consists of attributes and behaviour

* Attributes are represented with the help of non-static variables.

* Behaviours are represented with the help of non-static method.

CLASS

CLASS:

* According to real world situation before constructing. an object blueprint of the object must be designed, it provides specification of the real world object.

* Similarly in object oriented programming before creating an object the blue print of the object must be designed which provides the specification of the object, this is done with the help of class.

DEFINITION OF CLASS:

* It is user defined non primitive data type, it represent the blueprint of the real world object.

* class provides specification of real world object.

NOTE:

we can create any number of object for a class, it is known as instance of a class.

NON STATIC :

* Any member declared in a class and not prefixed with a static modifier is known as a non-static member of a class.

* Non-static members belong to an instance of a class. Hence it is also known as an instance member or object member.

* The memory for the non-static variable is allocated inside the heap area (instance of a class).

* we can create any number of instance for a class

* Non-static members will be allocated in every instance of a class.

NON STATIC MEMBERS:

* Non Static variable
* Non Static method
* Non static initializer
* constructors.

DESIGNING AN OBJECT :

Example :

```
class Pen
{
  String brand;
  String color;
  double price;
  double length;
}
```

brand
color
price
length
} → Properties

write()
draw()
} → behaviour

```
class Bottle
{
 String brand;
 String color;
 double price;
 int capacity;
 String shape;
}
```

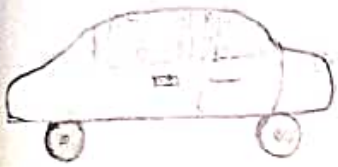
brand, color, price, capacity, shape → properties representing by non static variable

drink(), fill() → behaviour representing by non static method


```
class Book
{
 String title;
 int noOfPages;
 String author;
 double price;
}
```

title, noOfPages, author, price → properties

read(), return() → behaviour



```
class Car
{
 String brand;
 String color;
 double price;
 int millage;
 String model;
}
```

drive(), travel() → behaviour

```
class Chair
{
 double height;
 String color;
 String typeOfMaterial;
}
```

sit(), stand() → behaviour

```
class Bag
{
 int price;
 String brand;
 String color;
 String Material;
 int capacity;
 int warranty;
}
```

store(), carry() → behaviour

STEPS TO CREATE AN OBJECT:

STEP 1: Create a class or use an existing class if already created

STEP 2: Instantiation.

INSTANTIATION:

The process of creating an object is known as instantiation

Syntax to create an object:

```
new className();
```

NEW:

* new is a keyword.
* It is a unary operator.
* It is used to create a block of memory inside a heap area during Runtime.
* Once the object is created it returns the reference of an object.

CONSTRUCTOR:

Constructor is a special member of the class whose name is same as the class name.

Constructor is used to load the non static members of a class into the object.

Example:

STEP 1: Designing a class

```
class Employee
{
  String ename;
  int eid;
}
```

STEP 2: Instantiation

```
new Employee();
```

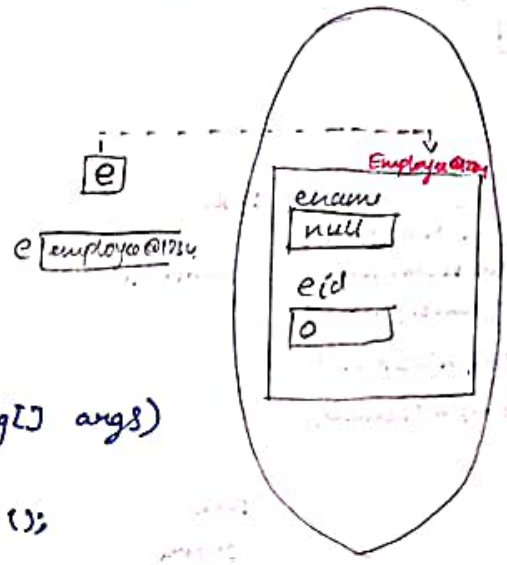
NON-PRIMITIVE DATA TYPE:

Every class name in java is a non primitive type.

Non primitive data type are used.

Example:

```
class Employee
{
  String ename;
  int eid;
}
class EmployeeDriver
{
  public static void main(String[] args)
  {
    Employee e = new Employee();
    System.out.println(e);
  }
}
```



output:

Employee@1234

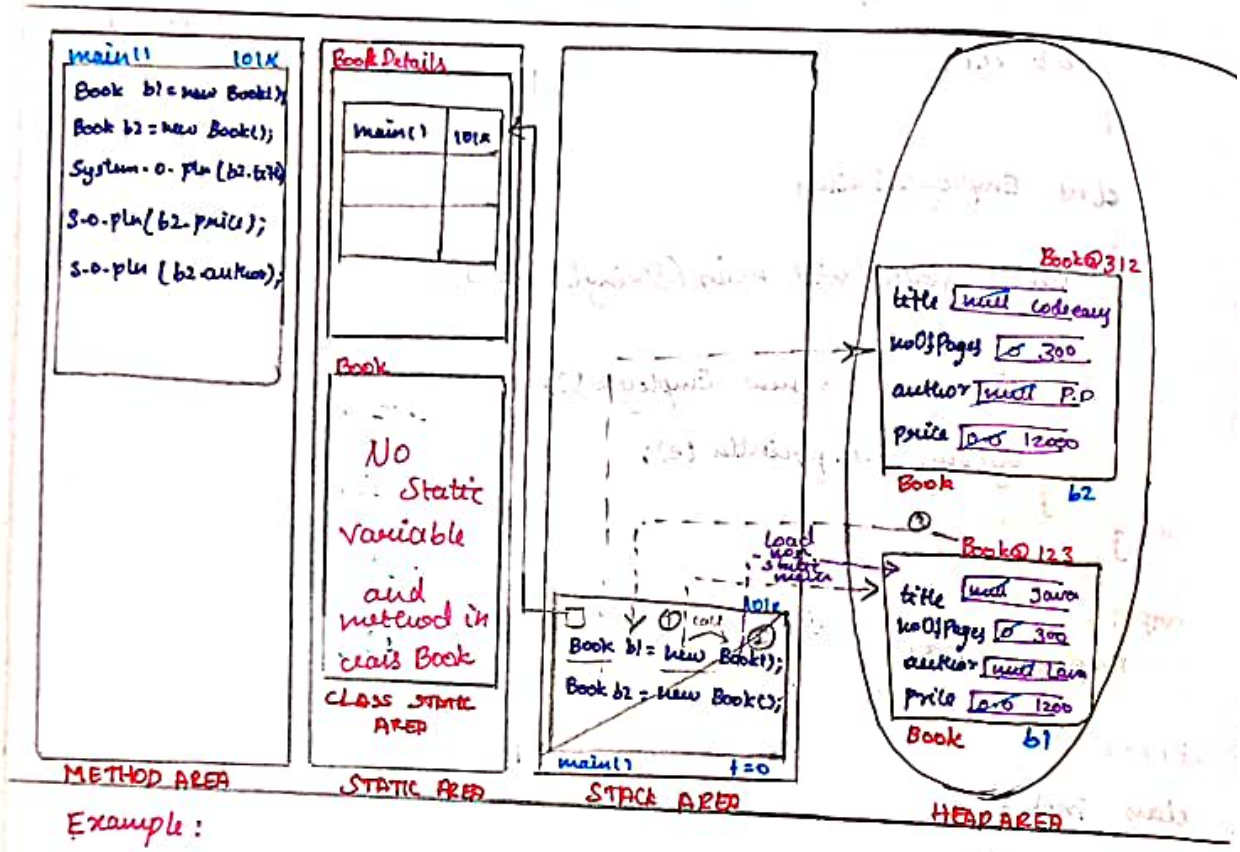
Example:

```
class Book {
  String title;
  int noOfPage;
  String author;
  double price;
}
class BookDetails {
  public static void main(String[] args) {
    Book b1 = new Book();
    Book b2 = new Book();
    b1.title = "Java";
    b1.noOfPages = 800;
    b1.author = "Java";
    b1.price = 1200;
    b2.title = "code Easy";
    b2.noOfPages = 300;
```

```
b2.author = "P.D";
b2.price = 12000;
System.out.println(b2.title);
System.out.println(b2.price);
System.out.println(b2.author);
}
}
```

OUTPUT :

Code easy
12000
P.D



Example :

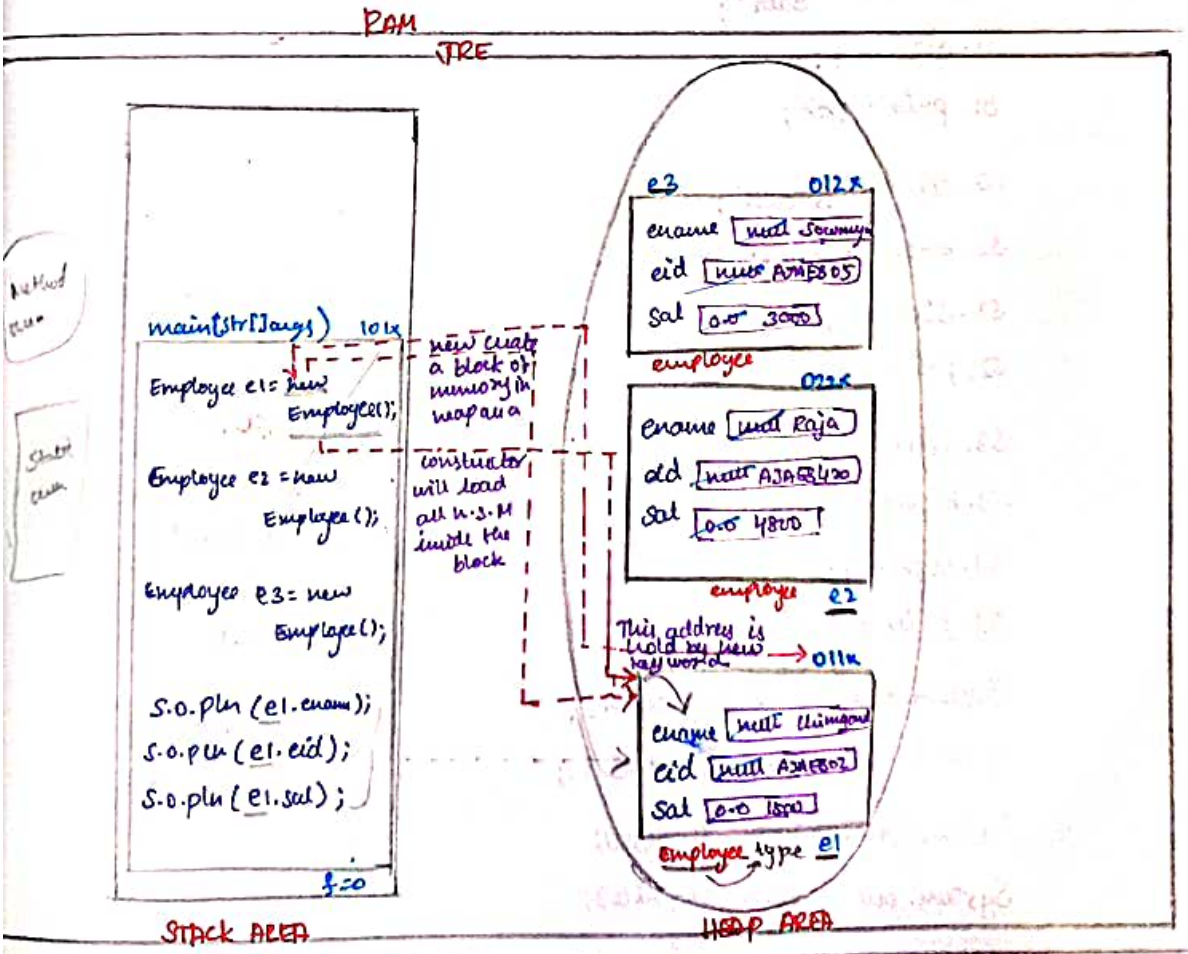
```
class Employee
{
    String ename;
    String eid;
    double sal;
}

class EmployeeDriver
{
    public static void main(String[] args)
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee();
```

```
        Employee e3 = new Employee();
        e1.ename = "chimgond";
        e1.eid = "AJAEB02";
        e1.sal = 1500d;
        e2.ename = "Raja";
        e2.eid = "AJAEB420";
        e2.sal = 4800;
        e3.ename = "Soumya";
        e3.eid = "AJAEB05";
        e3.sal = 3000;
        System.out.println(e1.ename);
        System.out.println(e1.eid);
        System.out.println(e1.sal);
    }
}
```

OUTPUT:

```
chimgond
AJAEB02
1500.0
```



28-MAR-22

Example :


```
class Shoe
{
    String color;
    String brand;
    int size;
    double price;
}
class ShoeDriver
{
    public static void main(String[] args)
    {
        Shoe s1 = new Shoe();
        Shoe s2 = new Shoe();
        Shoe s3 = new Shoe();

        s1.color = "Black";
        s1.brand = "Bata";
        s1.size = 9;
        s1.price = 1800;

        s2.color = "Brown";
        s2.brand = "woodland";
        s2.size = 8;
        s2.price = 2500;

        s3.color = "white";
        s3.brand = "puma";
        s3.size = 7;
        s3.price = 3000;

        System.out.println(s2.color);
        System.out.println(s2.brand);
        System.out.println(s1.size);
        System.out.println(s3.price);
    }
}
```

OUTPUT :

```
Brown
woodland
9
3000.0
```

NON STATIC VARIABLE :

A variable declared inside a class block and not prefixed with a static modifier is known as non-static variable.

Characteristics :

* we can't use the non-static variable without creating an Object.
* we can only use the non-static variable with the help of object reference.
* Non-static variable are assigned with default during the object loading process.
* Multiple copies of non-static variables will be created (once for every object).

Example :

```
class Slipper
{
   int size;
   String brand;
   public void slipperDetails()
   {
      System.out.println(size);
      System.out.println(brand);
   }
}
class SlipperDriver
{
   public static void main(String[] args)
   {
      Slipper slipper1 = new Slipper();
      Slipper slipper2 = new Slipper();
      slipper1.size = 8;
      slipper1.brand = "PUMA";
      slipper2.size = 9;
      slipper1.slipperDetails();
      slipper2.slipperDetails();
   }
}
```

OUTPUT :

```
8
PUMA
9
NULL
```