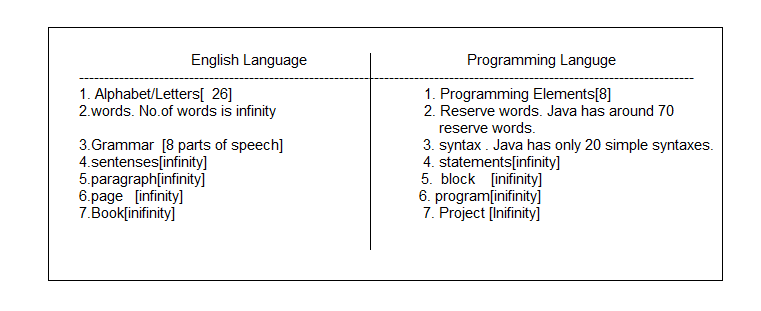
1. **Programming Elements.**
2. **Class**
3. **Instance**
4. **Object**
5. **Static Variables.**
6. **Non-Static Variables.**
7. **Static Blocks.**
8. **Non-static Blocks.**
9. **Methods.**
10. **Static methods.**
11. **Non-Static Methods.**
12. **Constructors.**
13. **Execution flow of static members.**
14. **Execution flow of Non-static Members.**
15. **Setter & Getter Methods.**

**Basic Java Programming Elements**

**1. Basic Java programming elements**:-

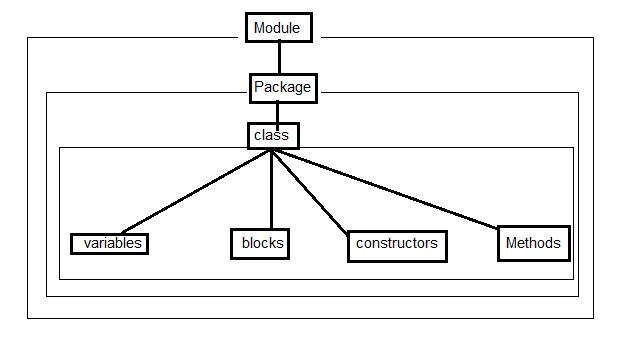


The programming language(java) is very easy than English language.

The Java programming language has only 8 basic elements.

1. Module b. Package c. class d. variable e. Block f. Constructor

g. method h. Inner class.



1.1 Module:- It is for packing/grouping the packages.

1.2. Package:- It is for grouping the classes.

1.3 class:- It is for representing Real world object in programming world. It groups, variables, blocks, methods, constructors and inner class as the unit.

Note:- Only common noun has to be takes as name of class in java world.

1.4 variable:- It represent the property of object. It hold value of object property and operations.

1.5 block:- It is for initializing variables common to all constructors.

1.6 Methods:- It is for representing an operation and implementing logic to perform this operation.

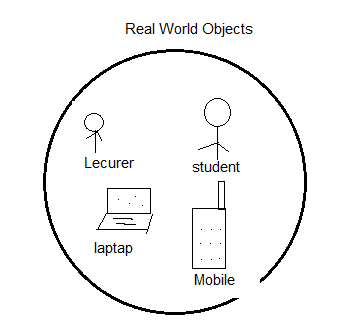
1.7 constructors:- It is for initializing variables with different logic.

1.8. Inner class:- it is to represent inner object to outer object.

Note:- Among all above 8 elements, class is mandatory and remaining elements are optional. The class is mandatory for creating java program . That is why java is oop language.

According OO Programming , we must develop all programs around objects. The object is real-world entity that u can see and use. We must bring real world objects into programming language(java) to perform their operation by using computer as part of business application.

Example:- Bring the following Real world objects into java world using basic programming elements.

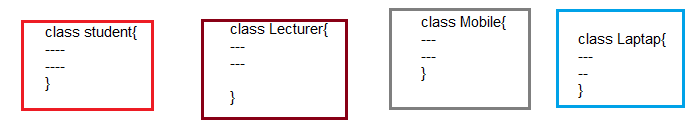


As per above diagram,

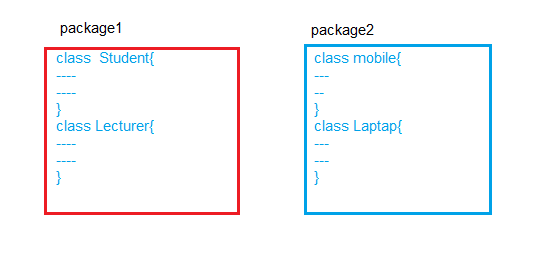
we should create the four different classes for representing the four objects in Java world.

In each class, We must create multiple variables for representing multiple properties of object.

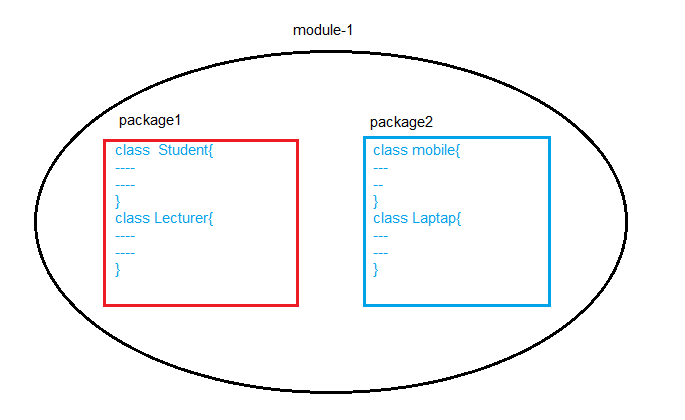
In each class, we must create multiple methods based on multiple operation which is performed by object.



We must make a groups based on relativity among objects.



We must make group the modules based on relation among the packages.



**CLASS**

**2.Class:**  we defines the class in several ways.

Definiton-1: The class is Referenced-defined data type. It is used for creating the user-defined Data type. When we declare variable of a user defined Data Type, that variable is reference variable and it holds address of object. That object can contains the different types of values.

Definition-2: The class is model of object.

Definition-3: The class is blue print of object.

Definition-4: The class is specification of object. The class

Definition-5: The class is logical construct of object.

Class provides convient method for packing group of related data items and methods that work on data items. The class specifies properties and methods of real world object.

Basic form of class is:

Class class-name

{ [Static Variables]

[Non-Static Variables]

[static blocks]

[non-static blocks]

[Constructors]

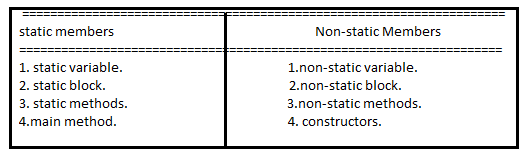
[static Methods]

[non-static Methods]

}

We can define 8 types of memebers in a single class. They have been divided into two categories.

1. Static members.
2. Non-static members.



Everything inside square brackets is optional. This means following class definition is also valid.

Class Name { }

We can access properties and methods of object using following syntax:

Syntax:

Reference DT VariableName. propertyName/methodName

**INSTANCE**

**3. Instance:** One memory copy created from class for representing one individual object by storing its values is called instance.

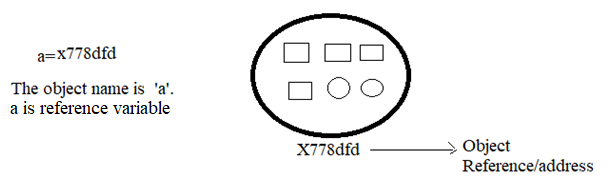
Syntax:

New classname();

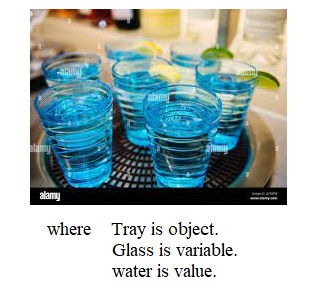
The memory copy does not hold values of real world object. It have empty memory locations.

**OBJECT**

**4. Object:-** The object is unnamed memory location. It contains the multiple values or objects. If object reference / address is stored in variable then variable name is considered as object name and that variable is called ‘reference variable’.



Real Time Example by which we can easily understand the ‘variable’ and ‘object’.



A Java object is **Instance of a Java class**. Each object has an identity(address/reference), a behavior and a state. The state of an object is stored in fields (variables), while methods (functions) display the object's behavior. Objects are created at runtime from templates, which are also known as classes.

( or )

When we place real world object value in instance,the instance is said to be a Object. Now java object represents real world object. In another way, we say that real world object is brought into java world.

Syntax:

1.className var-name=new ClassName(argument1,..etc);

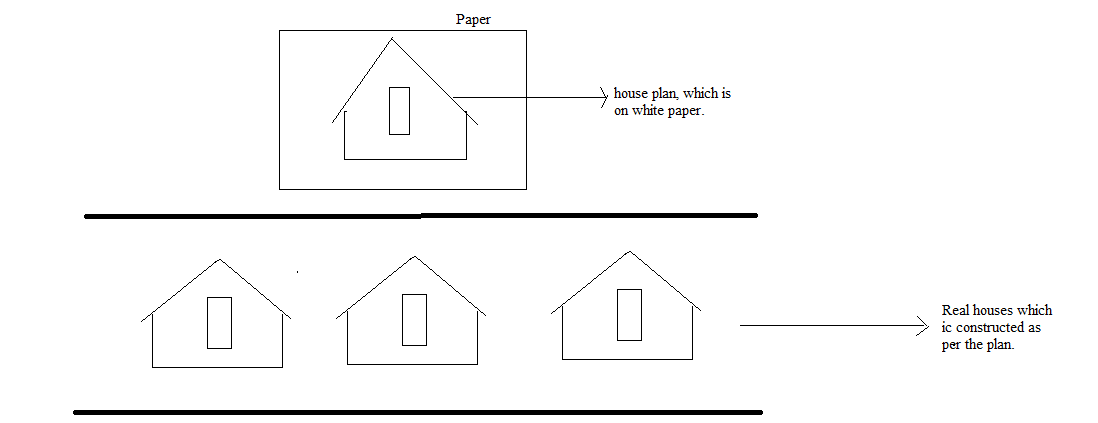
2. className var-name=new ClassName();

Var-name.propertyName=v1;

Var-name.propertyName=v2;

…etc.

Real Time Example to Class,Instance and Object:-



The house plan on paper is class.

Every real house is object.

**4.1 Steps to create real world object in the programming world**

a. Identify type of object, create it by using a class . class name must be object type name.

b. Identify its values, create them by using variables. Variables names must be same as value name.

c. Identify its operations, implement those operation by using methods. The method names must be same as operation names.

d. Identify individual objects. Create them by using instances.

e. Store the values of each object in their respective instances.

**4.2. Important Points:**

**a.** during desing phase, class depends on object. Design means deciding name of class, number of variables and their name, number of methods and their names.

**b.** In creation phase, object depends on class. Creation means memory allocation & storing object values in program and performing its operation in program.

Example:1 Develop a class for student with rno,branch,marks.

Class Student{

Int rno;

String branch;

Float Marks[]=new Float[3];

Boolean pass;

}

Example:2 Develop a class for Employee with eno, ename, salary.

Class Employee{

Int eno;

String ename;

Float salary;

}

Example: 3 Develop a class for Employee with eno, ename, worked\_locations salary and create two instances for representing employees ‘sukumar’,’srinivasa’.

Class Employee{

Int eNo;

String eName;

Float salary;

String workedLocations[]=new String[3];

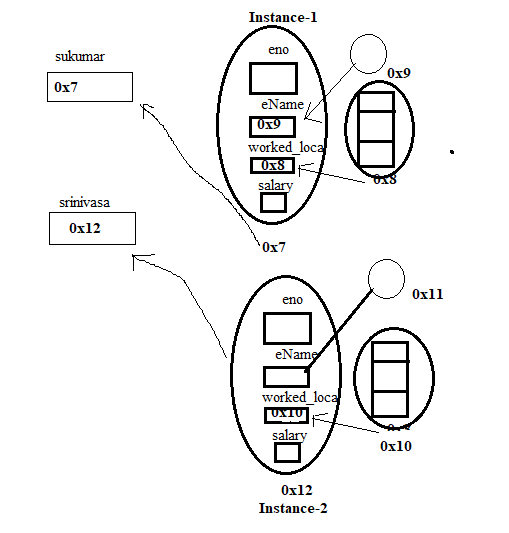
}

Employee sukumar=new Employee();

Employee srinivasa=new Employee();

Where sukumar and srinivasa are referenced variables. Their data type is Employee. These two variables will hold the reference of two different Employee instances.

Memory Representation Diagram:



Example: 4 Develop a class for Employee with eno, ename, worked\_locations salary and create two objects for representing employees ‘sukumar’,’srinivasa’.

Class Employee{

Int eNo;

String eName;

Float salary;

String workedLocations[]=new String[2];

}

Employee sukumar=new Employee();

Employee srinivasa=new Employee();

Sukumar.eno=7;

Sukumar.eName=’sukumar’;

Sukumar.salary=7000f;

Sukumar.workedLocation[0]=’nlr’;

Sukumar.workedLocation[1]=’sing’;

srinivasa.eno=8;

srinivasa.eName=’veena’;

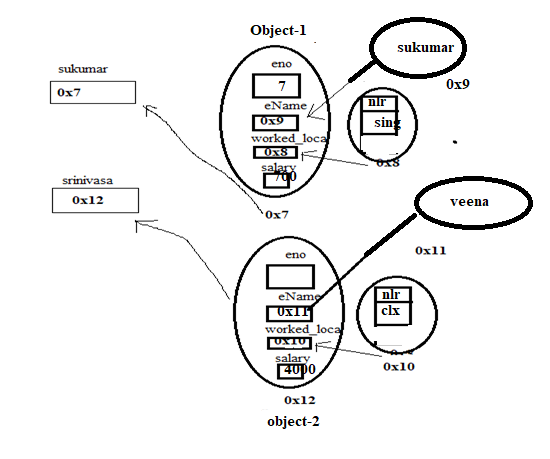
srinivasa.salary=4000f;

srinivasa.workedLocation[0]=’nlr’;

srinivasa.workedLocation[1]=’clx’;

Where sukumar and srinivasa are referenced variables. Their data type is Employee. These two variables will hold the reference of two different Employee objects.

Memory Representation Diagram:



Example: 6 Develop program to create RWO Bank A/c with its properties bankname, branchName, ifsc, accnum, balance & address. Create two new A/c for customers suku and sv with their values display their values on console.

class Address

{

String aptName;

String city;

int pin;

String streetName;

}

class BankAccount

{

String bankName;

String ifsc;

long acno;

String accHName;

double balance;

Address addr;

public static void main(String arg[]){

BankAccount suku=new BankAccount();

suku.bankName="SBI";

suku.ifsc="001ASBI";

suku.accHName="A.sukumar";

suku.balance=7000;

suku.acno=74735634252L;

suku.addr=new Address();

suku.addr.aptName="srinu";

suku.addr.city="nlr";

suku.addr.pin=520001;

suku.addr.streetName="kamati Street";

System.out.println("BankName:"+suku.bankName);

System.out.println("ifsc:"+suku.ifsc);

System.out.println("acno:"+suku.acno);

System.out.println("Acc Holder Name:"+suku.accHName);

System.out.println("Balance:"+suku.balance);

System.out.println("ApartmentName:"+suku.addr.aptName);

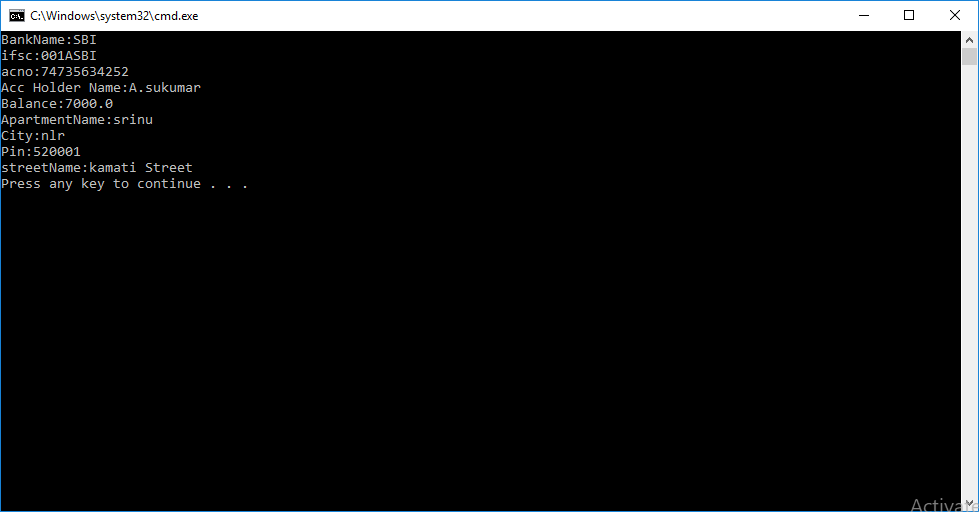
System.out.println("City:"+suku.addr.city);

System.out.println("Pin:"+suku.addr.pin);

System.out.println("streetName:"+suku.addr.streetName);

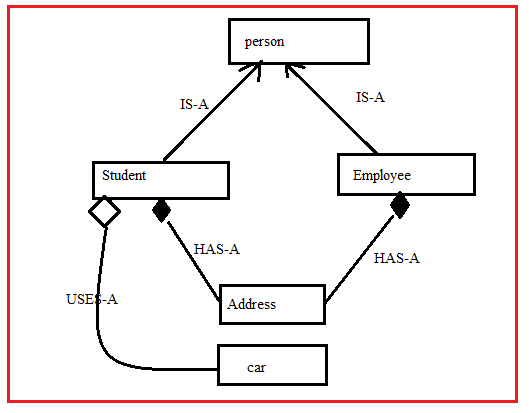
}

}



**4.3.Object Relations:-** A business has many objects. All these objects must be related to each other. Java supports we have three types of relations between objects.

1. IS-A (Inheritance)
2. HAS-A (Composition)
3. USES-A (Aggregation)



**4.3.1.HAS-A (Composition)**: We should establish HAS-A relation between classes if one object can not exist with out another object.

Example: a student or Employee objects can not exist without address.So we must establish HAS-A relation between these objects. This is the reason we call them as:

Student HAS-A Address.

Employee HAS-A Address.

Implementation:

Class Address{}

Class Student{

Address add= new Address();

}

Class Employee{

Address add =new Address();

}

**4.3.2.USES-A relation:** we should establish USES-A relation between classes if one object uses another object for performing one of its operations. For example Employee uses car for traveling . So we must establish USES-A relation between these two objects. This is the reason we call them as:

Employee USES-A car for travelling.

To implement this relation we must create one class referenced variable as parameter of another class method.

Class Bike{}

Class student {

Void m1(Bike b){}

}

**STATIC VARIABLES**

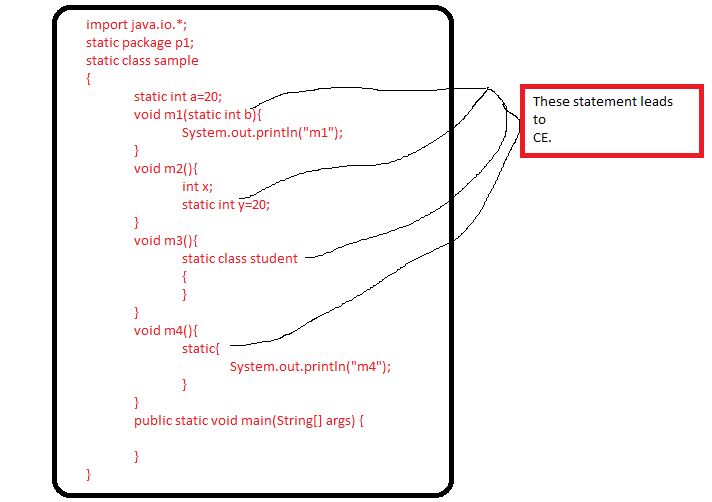
The static is a modifier.The static keyword is only allowed to following class members.

1. Fields
2. Blocks
3. Methods
4. Main method
5. Inner class.

Note:- Java5 onwards , the static keyword is allowed in “**import”** statement.

The static keyword is not allowed with

1. Module
2. Package
3. Outer class
4. Constructor
5. Parameter
6. Local variable
7. Local block
8. Local inner classes.



**5.Static Variables:** A variable which is declared inside a class , outside of all methods and It’s declaration/definiton has static keyword is said to be a static variable.

Syntax:

Static datatype var-name[=value];

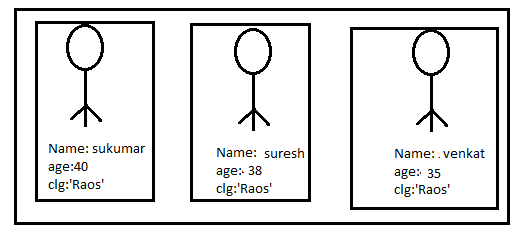
The static variable value can be accessed using following syntax:

Syntax:

className.var-name;

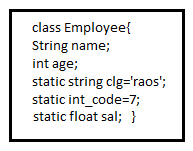
or

ObjectName.var-name;



When propertyname will has same value for all objects,such property has to be declared as static variable. In above diagram, clg is having same value for all object. Thereforeclg has to be declared as static variable.

**5.1.LifeTime of Static variable**:- The static variable gets memory in **method area** of JVM when class is loaded into JVM. The static variable is destroyed either if class is unloaded from JVM or JVM is destroyed.



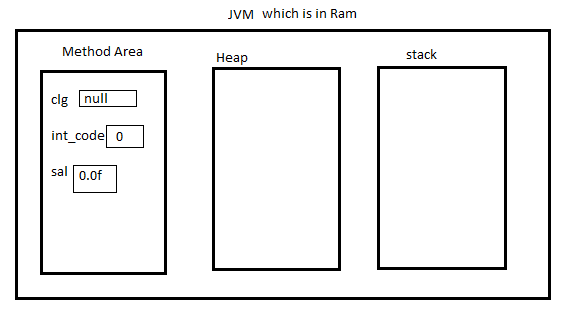
Stpe1: >javac Employee.java

The java compiler createsemployee.class and it place in hard disk.

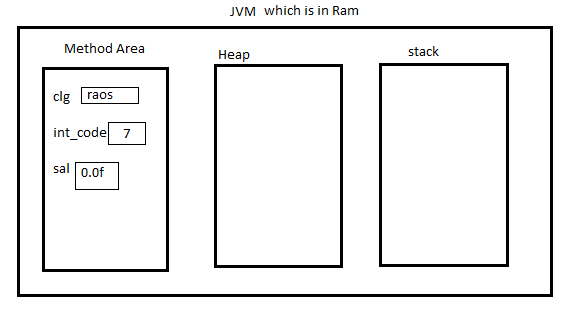
Step2: > java Employee

Java command starts execution of JVM. JVM starts loading of Employee.class (byte code)from hard disk into JVM memory.

Step3: while loading, JVM first creates all static variables from top to bottom of class in method area. JVM places default value based on data type in static variable.



Step4. Finally JVM stores assigned value in static variable,if any.



**5.2.Scope of Static Variable:-** Its scope is throughout class(i.e the variable can be accessed in all methods of same class) and where ever class is available there static variable is also available if it is public.

Note:\*\* The static variable can be got using “null Reference Varibale”.

Example:1

class Sample1

{

static byte x=10;

static short y=20;

static int a=10;

static long b=2345;

static float c=10.5f;

static double d=25.78;

static String e="A.sukumar";

static int[] f= new int[]{10,20,30};

static boolean g=true;

}

class Sample

{

public static void main(String[] args)

{

System.out.println("============================================================================");

System.out.println("Static Variables of All datatypes is accessed using class Name");

System.out.println(Sample1.x);

System.out.println(Sample1.y);

System.out.println(Sample1.a);

System.out.println(Sample1.b);

System.out.println(Sample1.c);

System.out.println(Sample1.d);

System.out.println(Sample1.e);

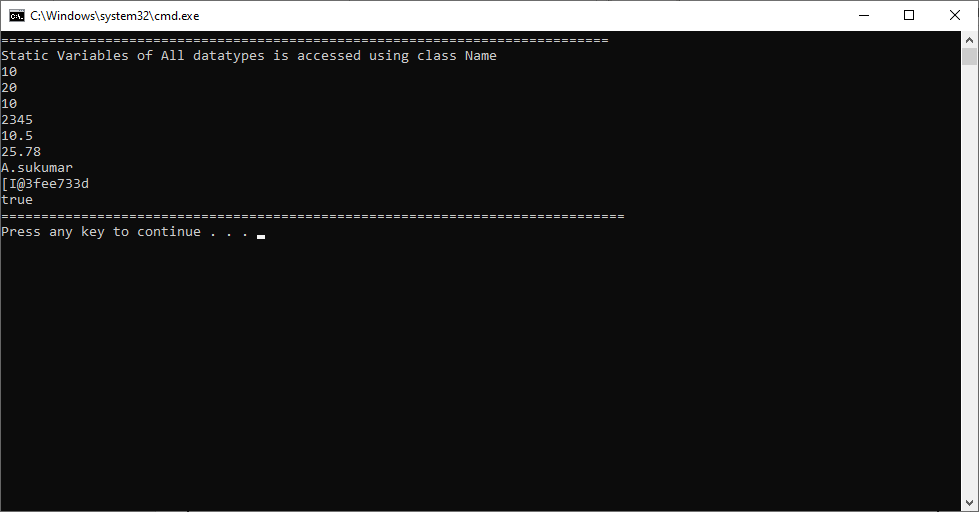
System.out.println(Sample1.f);

System.out.println(Sample1.g);

System.out.println("==============================================================================");

}

}



Example:2

class Sample1

{

static byte x=10;

static short y=-20;

static int a=30;

static long b=4567;

static float c=12.3f;

static double d=23.4;

static String e="A.sukumar";

static int[] f= new int[]{};

static boolean g=false;

}

class Sample

{

public static void main(String[] args)

{

Sample1 ab=new Sample1();

System.out.println("============================================================================");

System.out.println("Static Variables of All datatypes is accessed using object Name");

System.out.println(ab.x);

System.out.println(ab.y);

System.out.println(ab.a);

System.out.println(ab.b);

System.out.println(ab.c);

System.out.println(ab.d);

System.out.println(ab.e);

System.out.println(ab.f);

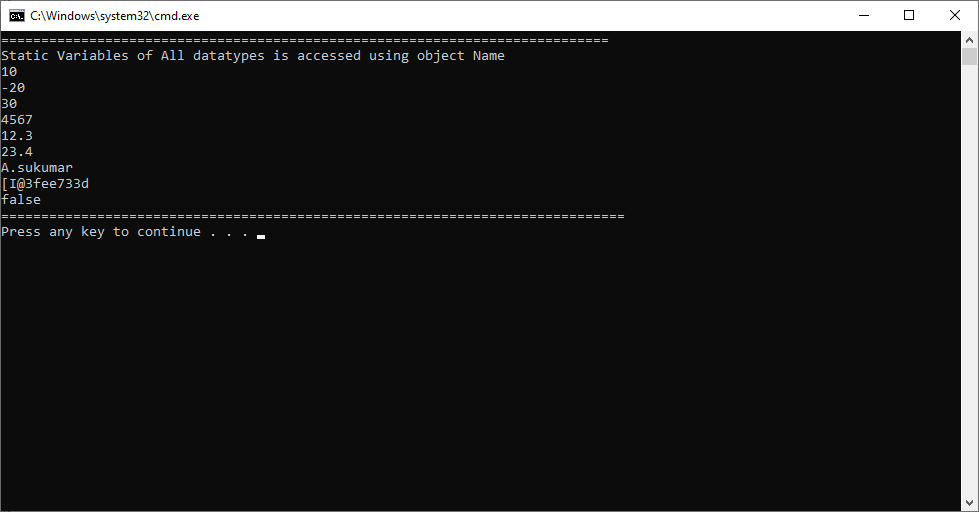
System.out.println(ab.g);

System.out.println("==============================================================================");

}

}

Output:



Example:3

class Sample1

{

static byte x;

static short y;

static int a;

static long b;

static float c;

static double d;

static String e;

static int[] f;

static boolean g;

}

class Sample

{

public static void main(String[] args)

{

Sample1 ab=new Sample1();

System.out.println("============================================================================");

System.out.println("Default Values of Static variables ");

System.out.println("Byte:"+ab.x);

System.out.println("Short:"+ab.y);

System.out.println("Integer:"+ab.a);

System.out.println("Long:"+ab.b);

System.out.println("Float:"+ab.c);

System.out.println("Double:"+ab.d);

System.out.println("String:"+ab.e);

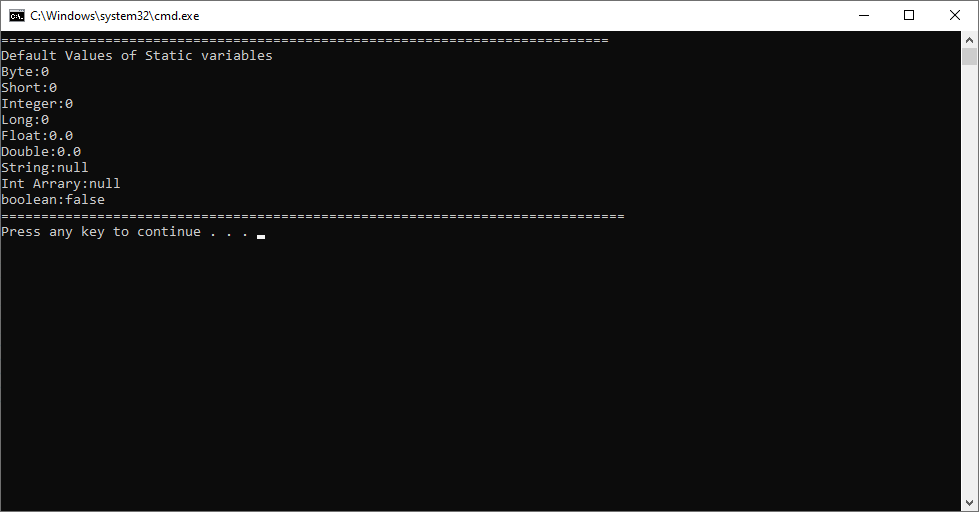
System.out.println("Int Arrary:"+ab.f);

System.out.println("boolean:"+ab.g);

System.out.println("==============================================================================");

}

}



Example:4 This program demonstrates that static variables can be accessed using “Null Reference Variable”.

class Sample1

{

static int a=20;

}

class Sample

{

public static void main(String args[])

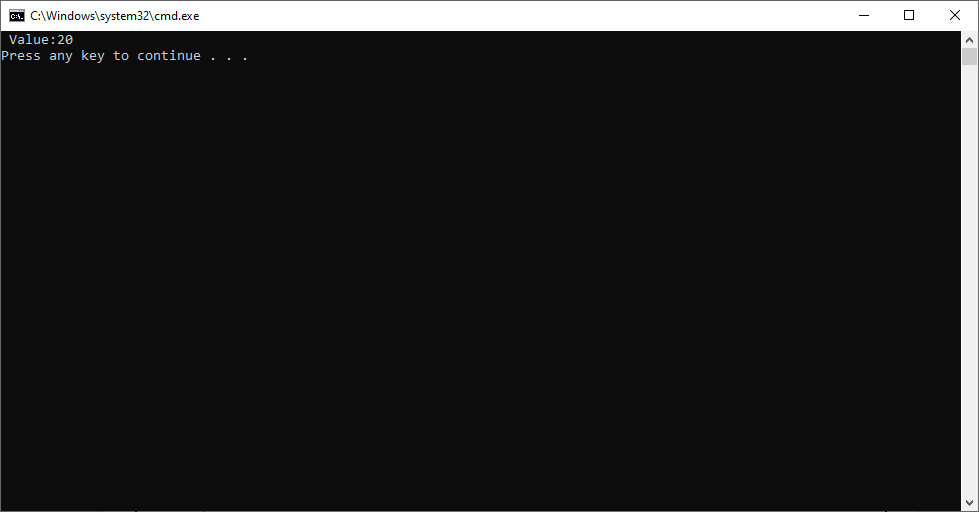
{

Sample1 ab=null;

System.out.println(" Value:"+ ab.a);

}

}

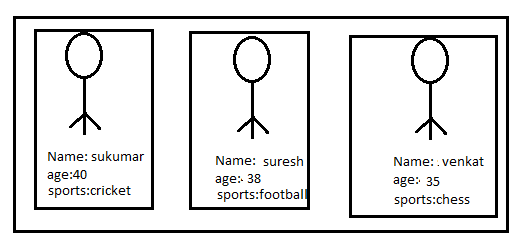


**Non-Static Variables**

**6.Non-Static Variable:** A variable which is declared outside of all methods and inside a class is said to be non-static variable. This variable declaration/defintion does not has static keyword.

Syntax:

Datatype var-name[=value];



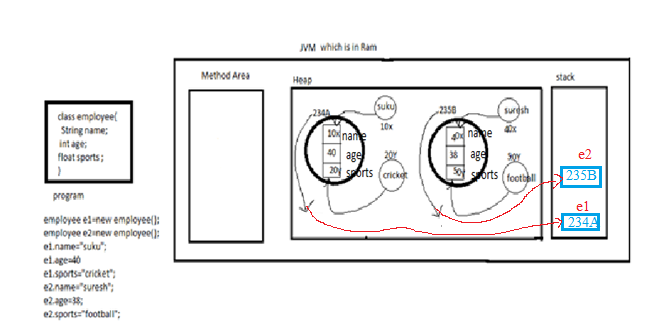
when we provide different value for same property of all objects of same class, that variable has to be declared as non-static variable. In above picture, all variables are taken as non-static variables.

**6.1.LifeTime of non-static variable:-** The non-static variables are continuously created in heapby JVM when object is created.When object is deleted, non-static variable of object is destructed.

Note:- The non-static variable gets independent memory location for each object.

**6.2. Scope of non-static variable:-** The scope of non-static variable is scope of its object. It means the non-static variable can be accessed in all methods of same class.

Memory Representation to employee Class:



Example:1

class Sample1

{

String eName;

float sal;

boolean working;

}

class Sample

{

public static void main(String args[])

{

Sample1 ab=new Sample1();

ab.eName="suku";

ab.sal=7000;

ab.working=false;

Sample1 xy=new Sample1();

xy.eName="rock";

xy.sal=9000;

xy.working=true;

System.out.println("=============FirstEmployee Details=================");

System.out.println("Ename:"+ ab.eName);

System.out.println("Salary:"+ab.sal);

System.out.println("Working Status:"+ab.working);

System.out.println("=============SecondEmployee Details=================");

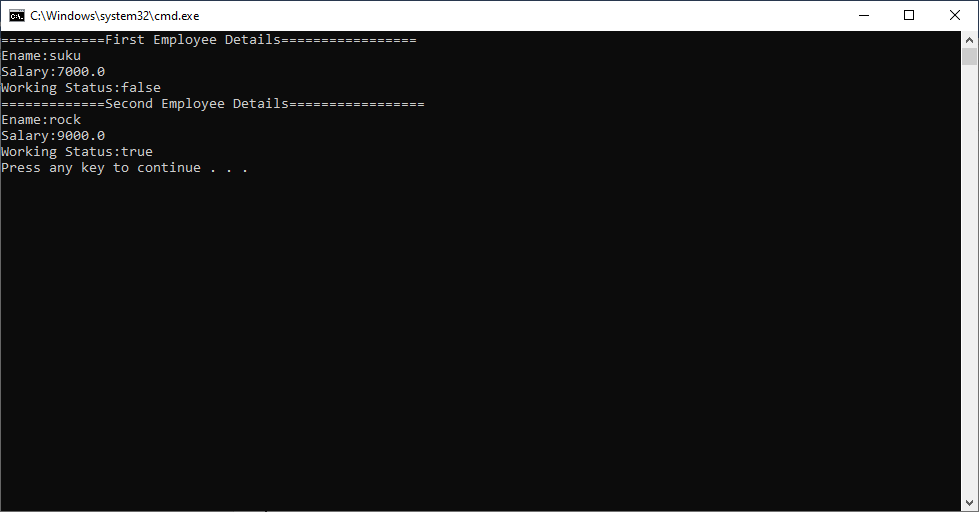
System.out.println("Ename:"+ xy.eName);

System.out.println("Salary:"+xy.sal);

System.out.println("Working Status:"+xy.working);

}

}



Note:\*\* The non-static variable can’t be accessed from static context.

Example:2 This program demonstrates that non-static variable can’t be accessed inside static block and static method. Such Accessing leads to CE.

class Sample1

{

String eName="suku";

float sal=7000f;

boolean working=true;

static{

System.out.println("=============Employee Details Inside a Static block=================");

System.out.println("Ename:"+ eName);

System.out.println("Salary:"+sal);

System.out.println("Working Status:"+working);

}

static void m1(){

System.out.println("=============Employee Details Inside a Static block=================");

System.out.println("Ename:"+ eName);

System.out.println("Salary:"+sal);

System.out.println("Working Status:"+working);

}

}

class Sample

{

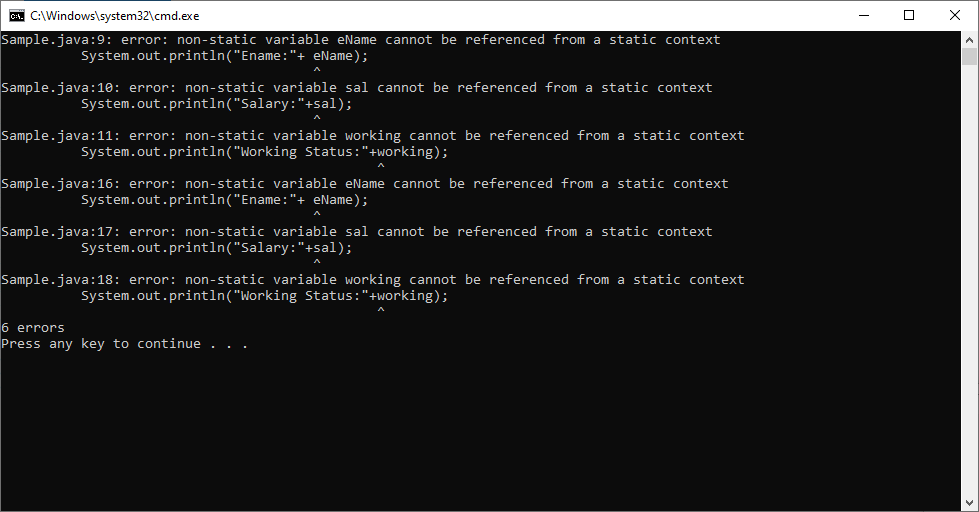
public static void main(String args[])

{

Sample1 ab=null;

}

}



**STATIC BLOCKS**

**7.static Block:**The static block is nameless block . The static block is written inside a class and outside the all methods of same class. In single class, we can define zero,one or more than one static block.

Syntax:

Class class-Name{

Static{

---

}

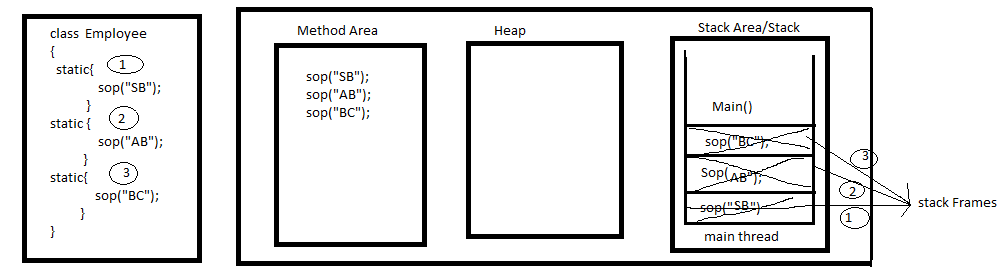
}

Usually, we write a logic inside static block for initializing **static variables, Registering libraries, connecting db, conneting to network**,..etc. we can write any java legal statement in static block except ‘**return and thrown**’.

The static blocks are stored in method area of JVM at the time of class loading in the order they defined from top to bottom. The static blocks are executed **automatically** by JVM at the time of class loading in order they defined from top to bottom.

Before start static block execution, JVM creates **stack frame** in main thread in **stack area**. JVM copies entire static block into stack frame and do exection. Once execution completed , immediately JVM deletes the stack frame.

Memory Representation of Employee Class:



Note: we should not write one static block inside the another static block.

Example:1

class Sample

{

static{

System.out.println("static block-1");

}

static{

System.out.println("static block-2");

}

static{

System.out.println("static block-3");

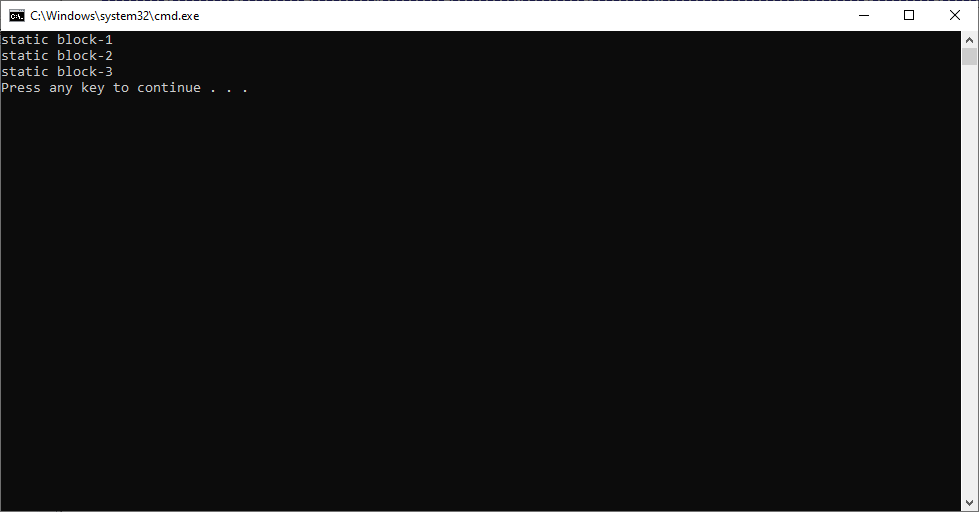
}

public static void main(String[] args)

{

}

}



Example:2 This program demonstrates that static variables can be accessed inside a static block.

class Sample

{

static int a=10;

static float b=20.4f;

static String c="sukumar";

static{

System.out.println("static block-1:"+a);

}

static{

System.out.println("static block-2:"+b);

}

static{

System.out.println("static block-3:"+c);

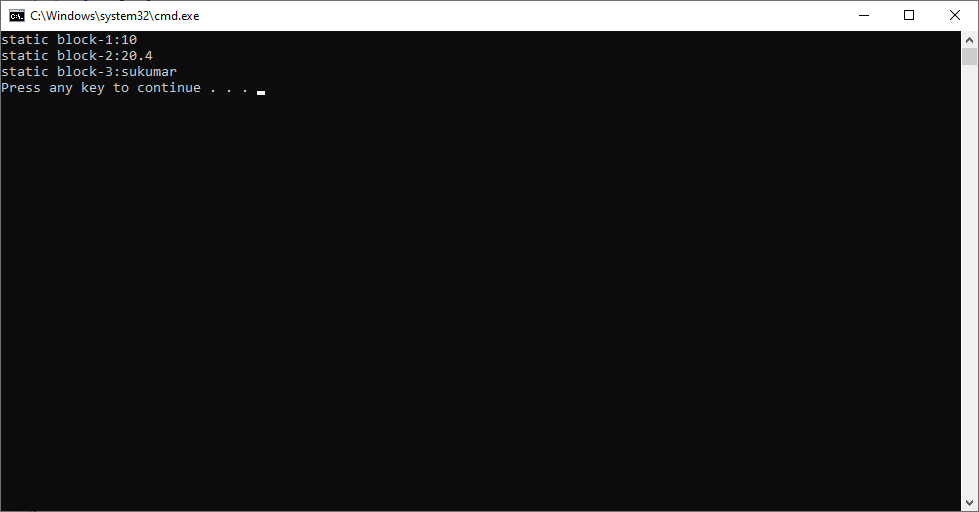
}

public static void main(String[] args)

{

}

}



**NON-STATIC BLOCKS**

**8.Non-Static blocks:** The block which is inside a class and which does not has prototype(head/static keyword) is called non-static block.

Syntax:

Class classname{

{ // non-static block started.

---

---

}//non-static block ended.

}

In single class, we can write zero, one or more than one non-static blocks.Non static blocks are automatically called by JVM for every object creation in java stack area. All static blocks are executed by JVM in the order they defined from top to bottom. Non-static blocks are always executed before constructor execution.

Example:1

class Sample1

{

String eName="suku";

float sal=7000f;

boolean working=true;

{

System.out.println("=============Employee Details Inside a Static block=================");

System.out.println("Ename:"+ eName);

System.out.println("Salary:"+sal);

System.out.println("Working Status:"+working);

}

}

class Sample

{

public static void main(String args[])

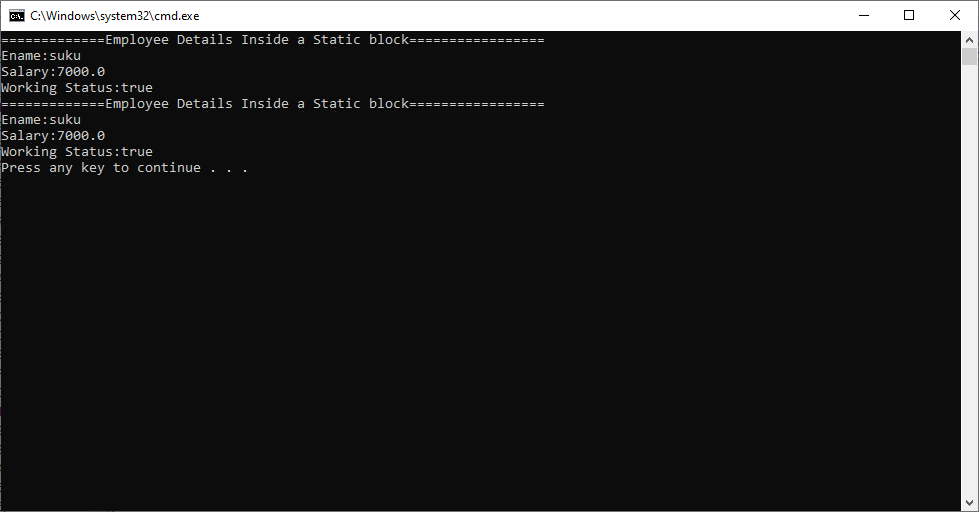
{

Sample1 ab=new Sample1();

Sample1 xy=new Sample1();

}

}



Example:2 This program demonstrates that static variables can be accessed inside a non-static-block and static methods can be invoked from non-static block.

class Sample1

{

static String eName="suku";

static float sal=7000f;

static boolean working=true;

{

m1();

}

static void m1()

{

System.out.println("============= Inside a non-Static block=================");

System.out.println("Ename:"+ eName);

System.out.println("Salary:"+sal);

System.out.println("Working Status:"+working);

}

}

class Sample

{

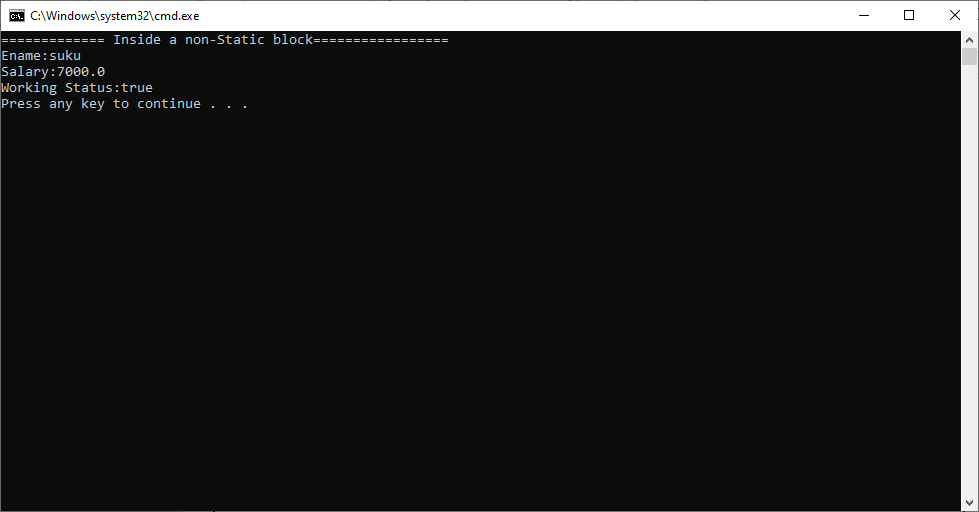
public static void main(String args[])

{

Sample1 ab=new Sample1();

}

}

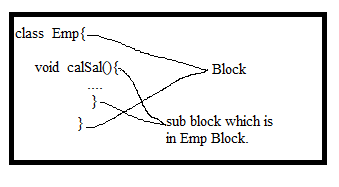


**METHODS**

**9. Method :**

Definition1: sequence of statements written as one group with name and return type is called a method. The sequence of statements are written in between the { and }.

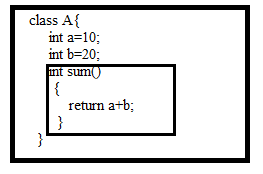
Definition2: The method is named sub block of class.



The method is invented for reusing same logic again and again in program. The method has logic for performing the mathematical operation or business operation.

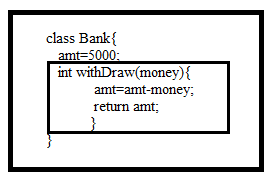
1.1.Mathematical Operation:The addition, subtraction, multiplication, division, flooring, ceiling, …etc.

Example:



1.2. Business operation: If mathematical operation is performed for particular realworld object(business,person,..etc) then mathematical operation is said to be business operation. Depositing ,withdrawing ,recharging..etc.

Example:



**9.2. Method Syntax:**

Generally when we visit the e-commerce website, we see the textbox along with search icon. We enter ‘mobile phone’ in textbox and we press the search icon. Immediately some logic will be executed. It will show either result(phone images or no products available)or exception message. To search, user must has some permissions to search products.

From above paragraph, we can know following points.

1.Every operation has Name.

Operation Name: search.

2. Operation may or may not take inputs.

Input is ‘mobile phone’.

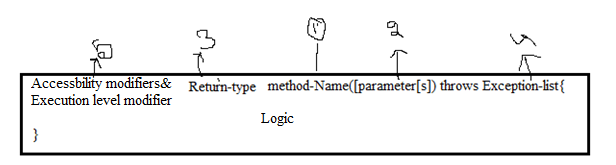
3. operation will give some result.

Result is either ‘mobile phone images’ or ‘no products available’.

4. while executing logic, some error may happens.(exceptions).

5. The user must has some permission to perform search.

Syntax:



Where Accessbility modifier, execution level modifier, parameters , Exception-list and logic are optional parts.

Return-type,method-name is mandatory elements.

**9.3.Method Terminology:**

3.1. Prototype.

3.2. body and logic.

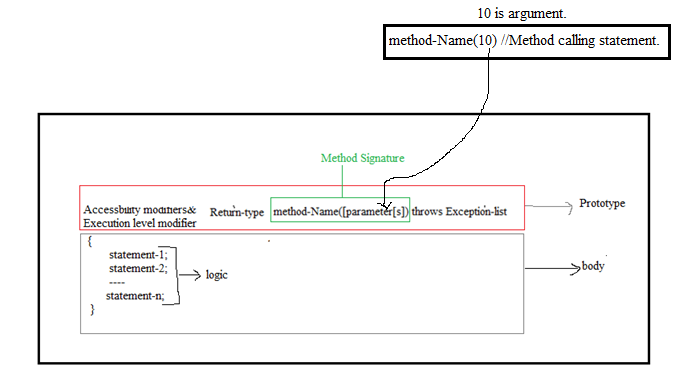
3.3. Signature.

3.4. Parameter and argument

3.5. Method Name.

3.6. Return type.

3.7. Modifiers.



9.3.1.Prototype: The modifiers,return type, method-name , parameters and exception list all combinedly is said to be prototype.

9.3.2.Body: The region between { and } brace is said to be body.

9.3.3.Logic: The statements which are written in a body is called a logic.

9.3.4.Signature: The method name and parameters combinedly is called method signature.

9.3.5.Parameter:- The variable which is declared between ‘(‘ and ‘)’ is said to be parameter. The parameter is used to receive input from outside a method.

9.3.6.Argument:- The value which is passed from method calling statement to method definition is called argument.

9.3.7. Return Type:- It specifies that what type of value returned by method.

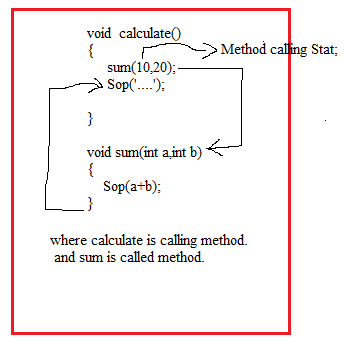
**9.4.Method Invocation:**calling a method to execute a method logic is called method invocation.

**9.4.1.Calling method:-**The method which invokes/calls the another method is called calling method.

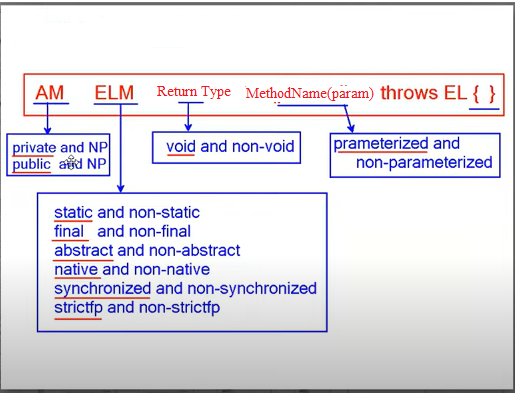
**9.4.2. Called method:-**The method which is invoked/called by another method is called ‘called method’.

When we call a method currently running method execution is passed, control is sent to this called method, this called method logic is executed. After method execution is completed, control is returned to calling method, the rest of statements placed in the calling are executed.

Example:



**9.5.Types of Methods:**

****

9.5.1.Void Method:**-** The method which don’t’ return a value is called void method.

9.5.2.non-voidMethod:- The method which returns a value is called non-void method.

9.5.3.Parameterized Method:- The method contains parameter list which receives value[s] from calling method.

**Limitation of Parameterized Method having more than 3 arguments:**

1. While calling a method, we should remember the no.of arguments.
2. While calling a method, We should remember the type of each argment.
3. While calling a method, We should remember the order of the arguments.
4. While calling a method, we can’t ignore pass un know value.

**Solution:-** Design the method which has Java bean class parameter type.

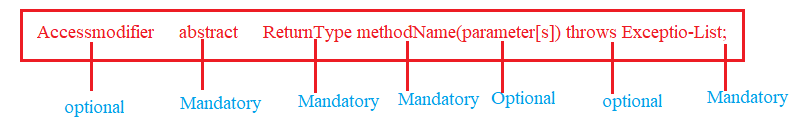


9.5.4 non-parameterized Method:- The method does not contains parameter list. The calling method invokes non-parameterized method without sending arguments.

9.5.5 abstract method:- If method Logic is vary from sub-type object to sub-type object then that method should be implemented as abstract method. The abstract method should always be written inside the interface.

The abstract method does not have body, just have prototype. The abstract method must ends with semicolon(;) otherwise it leads to CE.

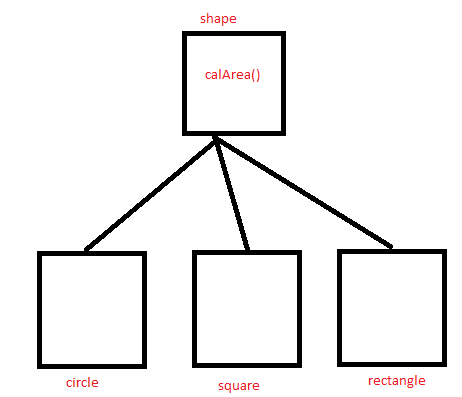
Syntax:



The abstract method allows only public, protected and default access modifiers.

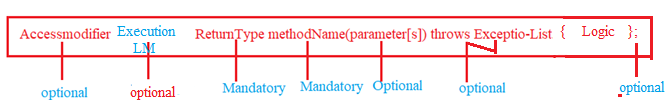
It does not allows private access modifier.

Example:



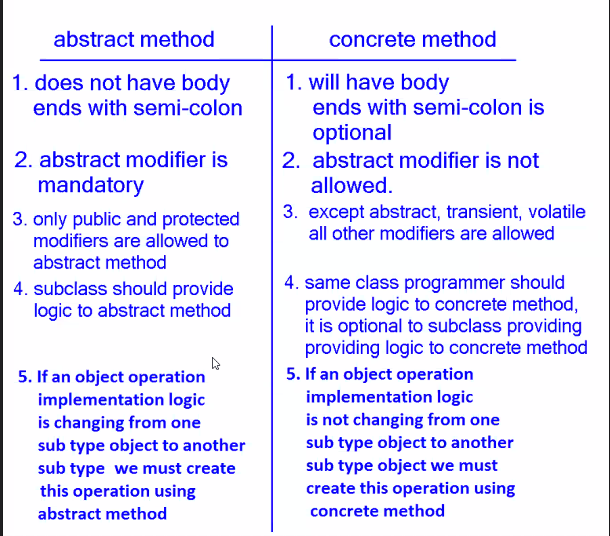
9.5.6.Non Abstract Method/ concrete Method: If method logis is same to all sub-types objects , then method should be implemented as concrete method. The method which has both prototype and body is called concrete method.

Syntax:



Note:- In method prototype, we should not write the abstract keyword.

Diff between abstract and concrete method:



**STATIC METHODS**

**10.static Methods:** A method which is inside a class and Its definition has static key word is called static method.

Syntax:

Static return-type method-name([parameter[s] definition]) {

----

}

The static methods are invoked using classname or object Name.

Syntax:

Classname.method-name;

(or)

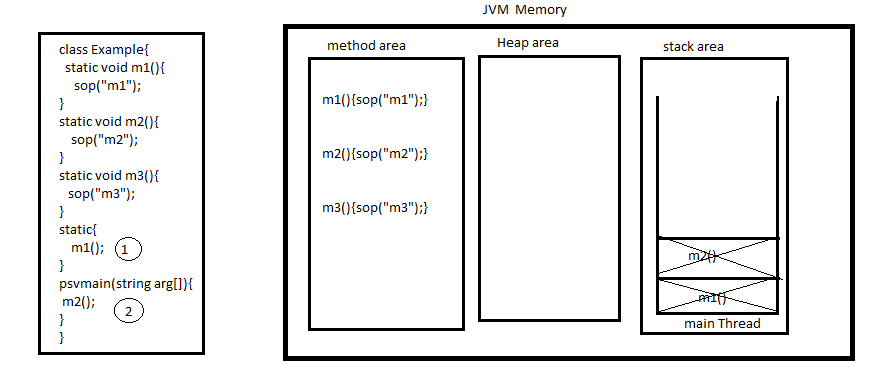
Objectname.method-name

The static methods are stored in method area of JVM at the time of class loading in the order they defined from top to bottom. JVM does not execute static blocks automatically itself.They are executed only if they are invoked from **main method**, from **static variables** as its assignment statement or from **static block** or from another **static method.**

Before start static method execution, JVM creates **stack frame** in main thread in **stack area**. JVM copies entire static method into stack frame and do exection. Once execution completed , immediately JVM deletes the stack frame.

Note:-1.\*\* The static methods can also be invoked with “Null reference variable”.

Memory Representation to Example class:



Example:1 This program demonstrates that static methods can be invoked using class name.

class Sample1

{

static void m1(){

System.out.println("This is static Method:1");

}

static void m2(){

System.out.println("this is static Method:2");

}

}

class Sample

{

public static void main(String args[])

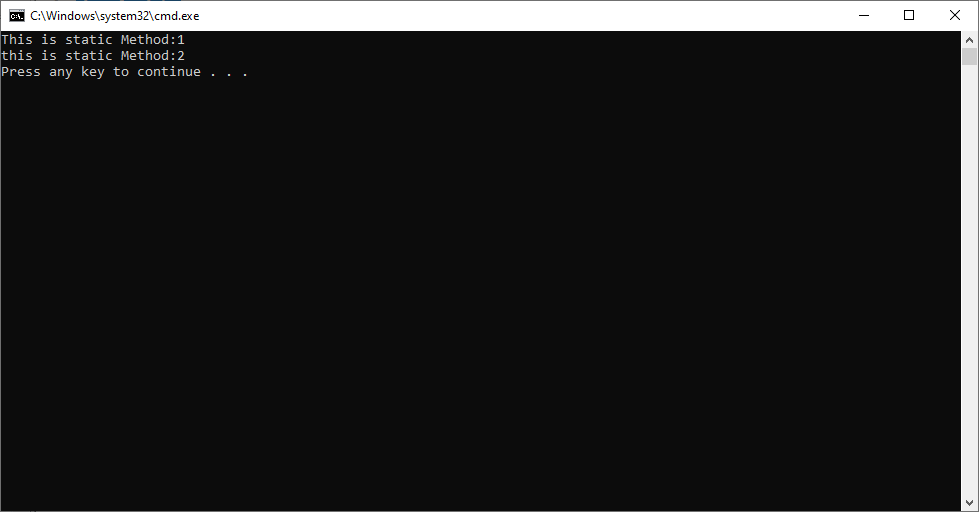
{

Sample1.m1();

Sample1.m2();

}

}



Example:2 this program demonstrates that static methods can be invoked using object name.

class Sample1

{

static void m1(){

System.out.println("This is static Method:1");

}

static void m2(){

System.out.println("this is static Method:2");

}

}

class Sample

{

public static void main(String args[])

{

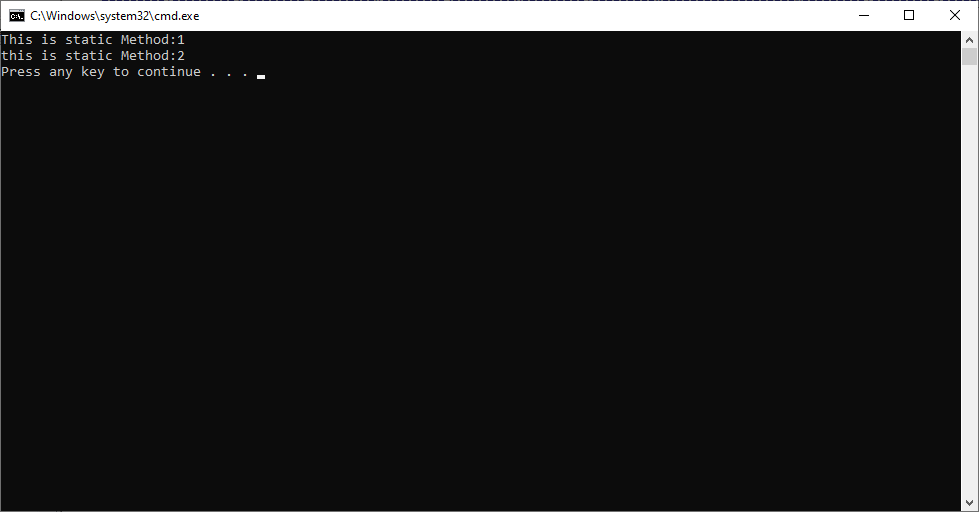
Sample1 s1=new Sample1();

s1.m1();

s1.m2();

}

}



Example:3 This program demonstrates that static methods can access the static variables and static methods can be invoked from static blocks.

class Sample

{

static int a=20;

static float b=20.3f;

static{

m1();

m2();

}

static void m1(){

System.out.println("This is static Method-1 and static variable:"+ a);

}

static void m2(){

System.out.println("this is static Method:2 and and static variable:"+b);

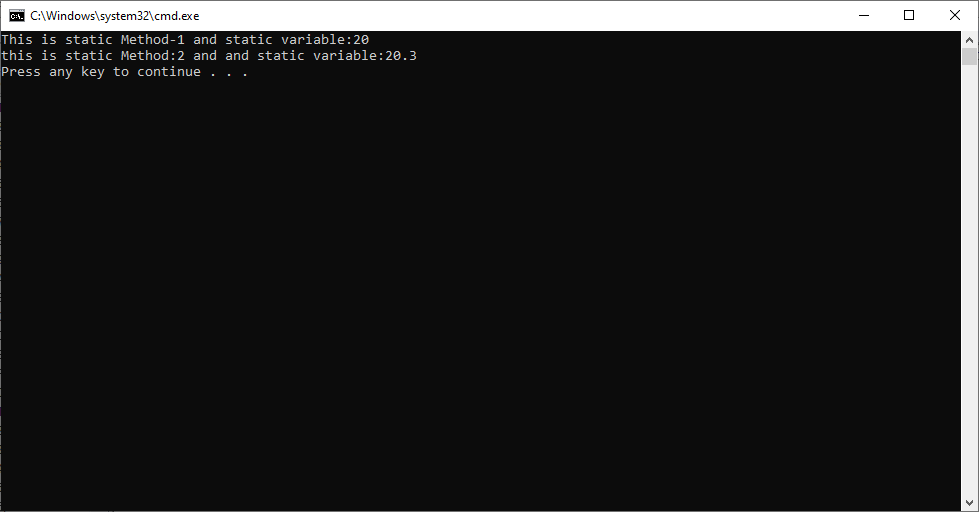
}

public static void main(String args[])

{

}

}



Example:4 This program demonstrates that static methods can also be invoked using “null reference variable”.

class Sample1

{

static int a=20;

static void m1(){

System.out.println(" Value:"+ a);

}

}

class Sample

{

public static void main(String args[])

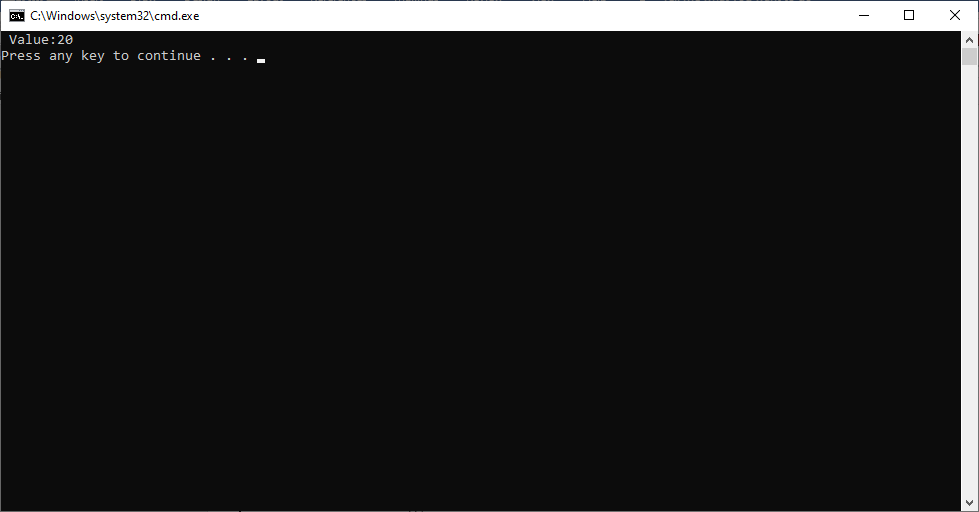
{

Sample1 ab=null;

ab.m1();

}

}



**NON-STATIC METHODS**

**11.Definition:** The method which does not have static keyword in its prototype is called non-static method.

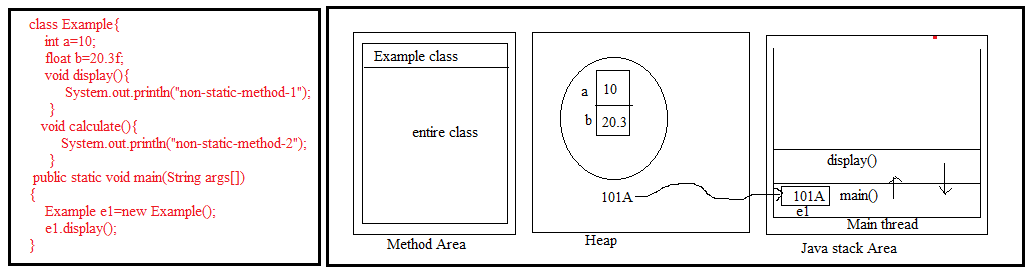
The JVM does not execute the non-static method automatically. The JVM execute it when it is invoked. We invoke non-static method using object name.

Syntax:

Objectname.methodName([parameter[s]]);

Usually, the non-static methods logic is stored in Method Area. When it is called its logic is loaded and executed in java stacks Area by creating separate stack frame in main thread. After execution completed , the stack frame is removed from java stack area.

Memory Representation to Example Class:



Note:-1. The non-static method can invoke from another non-static method directly by its name with in same class.

2. The non-static method can invoke from static context using object name.

Example:1

class Sample

{

void calculate(){

System.out.println("This is Non-static Mehtod-1");

display();

}

void display(){

System.out.println("This is non-static Mehod-2");

}

public static void main(String args[])

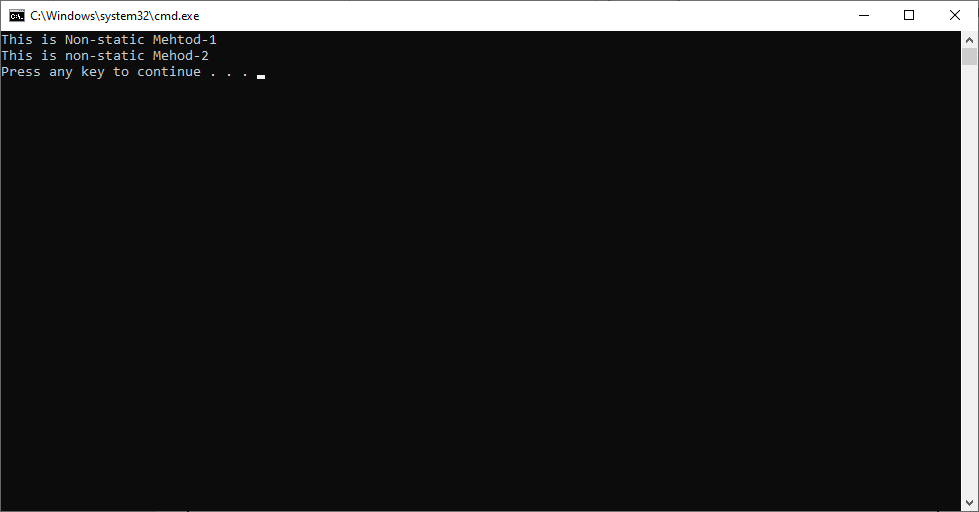
{

Sample ab=new Sample();

ab.calculate();

}

}



Example:2 This method demonstrates thatThe non-static method can be invoked from static context using objectName.

class Sample

{

static void calculate(){

Sample s=new Sample();

s.display();

}

void display(){

System.out.println("Rock");

}

static{

Sample s=new Sample();

s.display1();

}

void display1()

{

System.out.println("Rock1");

}

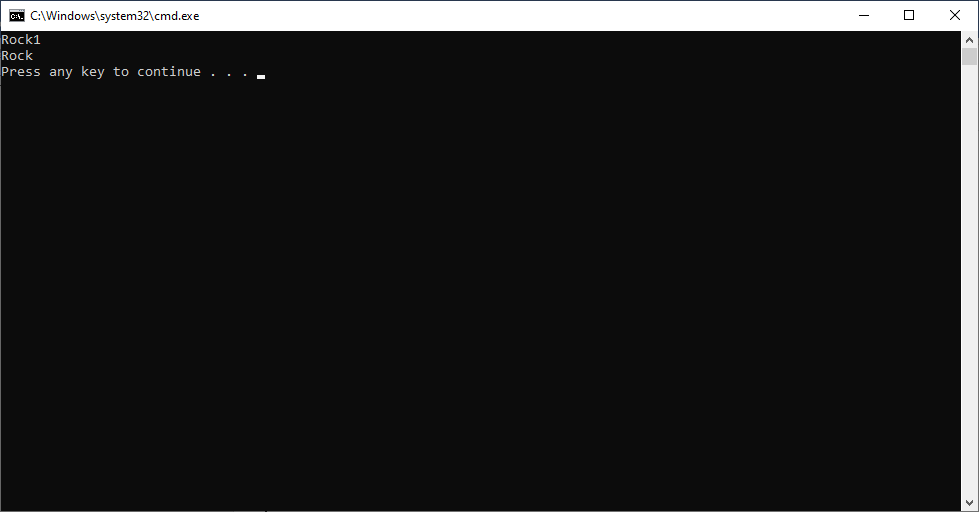
public static void main(String args[])

{

Sample.calculate();

}

}



**CONSTRUCTORS**

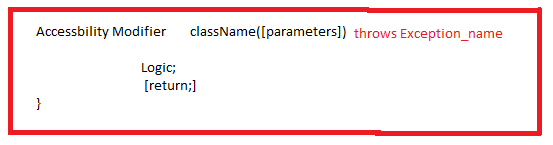
**12.Constructor**: The constructor is a special method. The constructor role is only initializing object. Initializing object means placing values in the class variables at the time of object creation.

If we want to execute some logic at the time of object creation, that logic may be object

Rules in Defining Constructor:

1. Constructor name should be same as class name.
2. It can has one of four accebility modifier. If accebility modifier is private then we can’t create object for that class.
3. It should not contain the return type. If it contains return type then compiler will not treat it as constructor and compiler will treat it as normal method.
4. It can have parameters.
5. The constructor body can have return statement with out value.
6. The constructor body has logic. It can have all java legal statement except return statement with value.
7. In a class we can define multiple constructors , but every constructor must have different parameters type or parameters order. So in class we can define one no-argument constructor + ‘n’ number of parameterized constructors.

Syntax:



1. Employee(){

….

}

Valid constructor.

2. int Employee(){

…

}

Compiler will not thorow CE. It treat it as method.

3.Employee(int a, float )

{

Return;

}

Valid constructor.

4. Employee(int a,float b)

{

Return 10;

}

Invalid constructor.

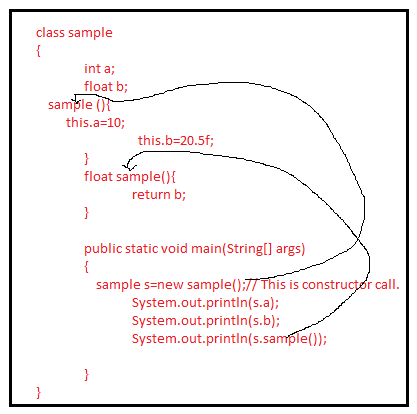
5. private Employee(){

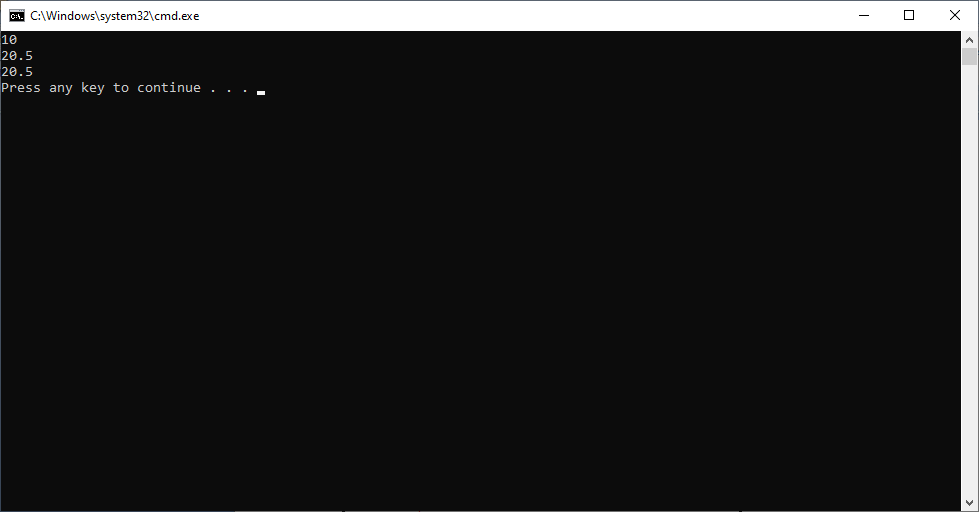
…

}

Valid constructor but we can’t create object to Employee class Because it’ constructor is private constructor.

Example:6





Note:-1. The constructor must be called along with ‘new ‘ keyword. Else it leads to compile time error.

Example:1

class sample

{

int a;

float b;

sample (){

this.a=10;

this.b=20.5f;

}

public static void main(String[] args)

{

**sample s=new sample();//** This is constructor call.

System.out.println(s.a);

System.out.println(s.b);

}

}

Example:2

class sample

{

int a;

float b;

sample (){

this.a=10;

this.b=20.5f;

}

public static void main(String[] args)

{

sample s=new sample();// This is constructor call.

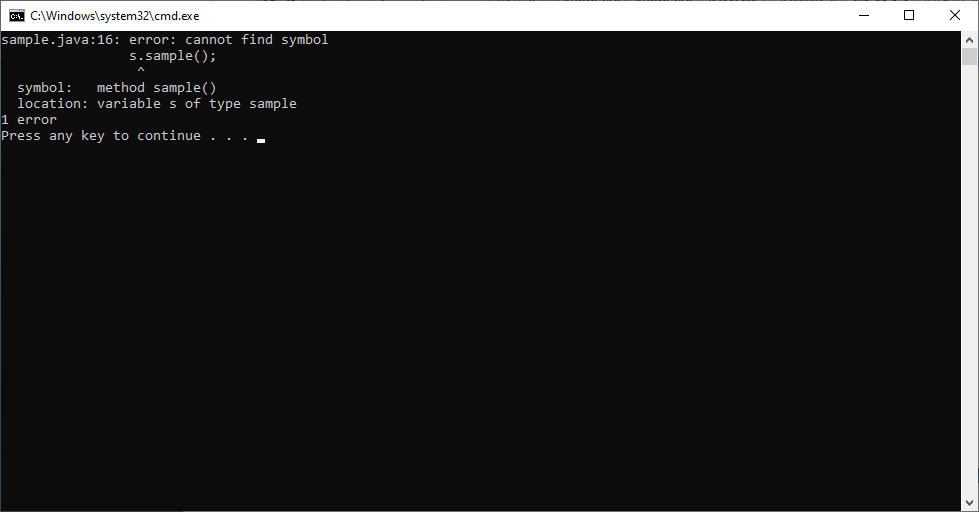
System.out.println(s.a);

System.out.println(s.b);

s.sample(); // This statement leads to CE.

}

}



**12.1.Types of Constructors:-** Java supports 3 types of constructors.

a. Default Constructor.

b. No/Non-parameterized constructor.

c. Parameterized constructor.

**a) Default Constructor:-** In a class , constructor is optional. If programmer does not define a constructor in class then compiler places constructor definiton. The compiler defined constructor is said to be default constructor.

The default constructor does not have parameters and logic except **super()**call. Its accessibility modifier is same as class accessibility modifier. So the only allowed accessibility modifier are default or public.

Example:-

class sample

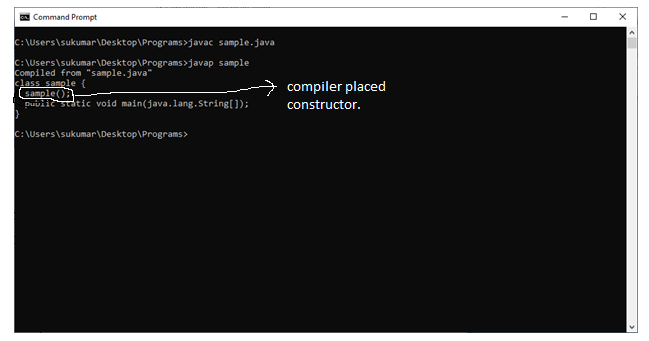
{

public static void main(String[] args)

{

}

}



**b) non-parameterized constructor:-** The developer defines constructor without parameter is called no- arg |non-parameterized constructor.

Example:1

class sample

{

float f;

sample(){

}

public static void main(String[] args)

{

sample s=new sample();

}

}

Example:2

class sample

{

float f;

sample(){

f=20.5f;

}

public static void main(String[] args)

{

sample s=new sample();

}

}

**c) parameterized constructor:-** The developer defines constructor with parameters is called parameterized constructor.

Example:

class sample

{

float f;

sample(float f){

this.f=f;

System.out.println(this.f);

}

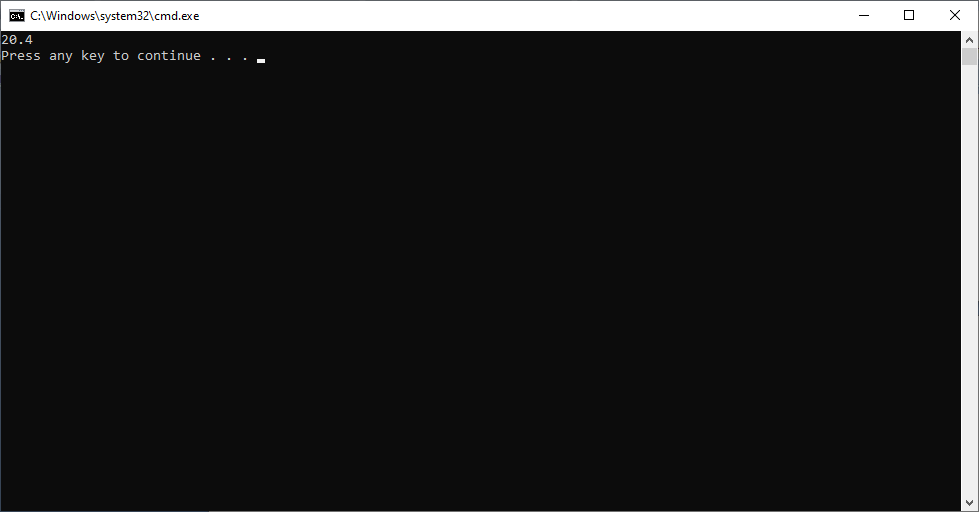
public static void main(String[] args)

{

sample s=new sample(20.4f);

}

}



Important Rule:- 1.If user has to write super() call in constructor then it should be as first statement in constructor. Otherwise it leads to CE.

Example:

class sample

{

float f;

sample(float f){

this.f=f;

System.out.println(this.f);

super();

}

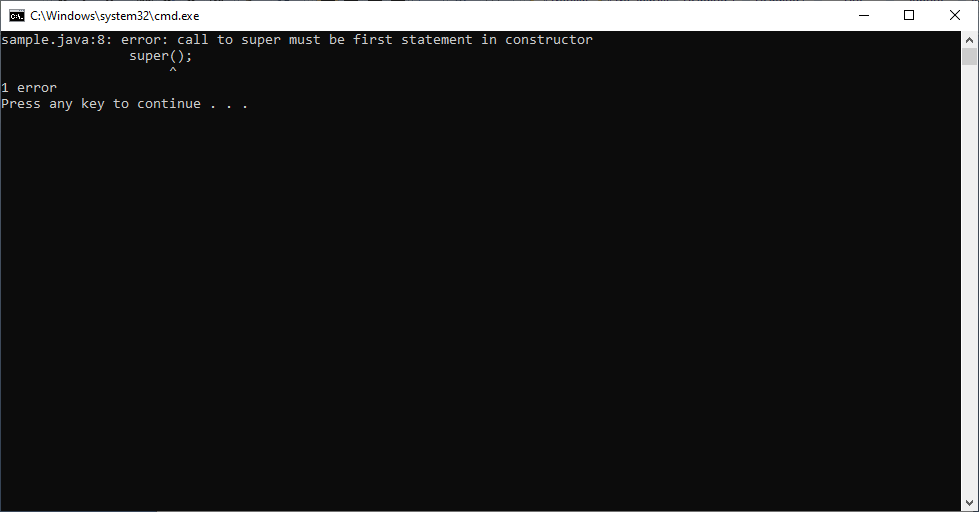
public static void main(String[] args)

{

sample s=new sample(20.4f);

}

}



2.If user does’t write super() call in constructor logic then compiler automatically places super() call as first statement.

**EXECUTION FLOW OF STATIC MEMBERS**

**13.** **JVM executes static members of class in two phases.**

1. Identifying phase.
2. Execution phase.

**A.Identifying phase:-**

a. JVM creates a static variables(creates memory) in the order they have been declared/defined from top to bottom and JVM keep default value in them based on their data types.

b. JVM analyze and remember the static blocks in the order they defined from top to bottom.

c. JVM remembers prototypes of static methods.

**B.Execution phase:-**

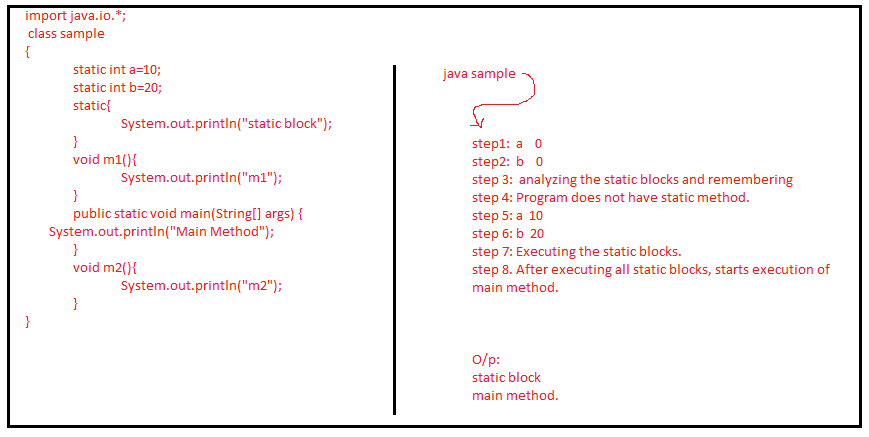
a. JVM stores assigned values in static variables in the order they defined from top to bottom.

b. JVM executes all static blocks in the order they are defined from top to bottom.

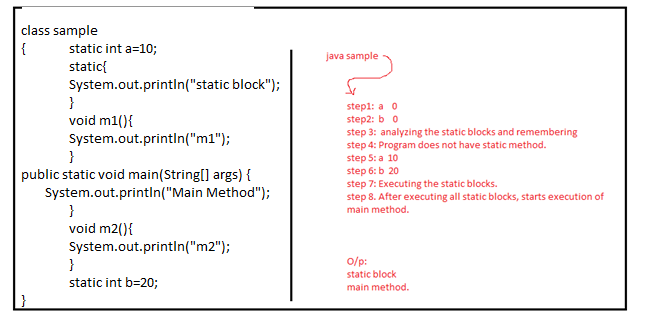
c. JVM executes static methods ,if they are invoked . otherwise JVM does not executes them.

d. Finally main() is invoked by JVM.

Example:-1



Example:2



**13.2.Illegal Forward Reference Error:** The class level static variable value should not be accessed directly before its declaration statement either from other variables or from static blocks.

It leads to CE: illegal forward reference.

Note:- From static block, we can’t read default value of static variable. Because it is locked.

To avoid CE: illegal forward reference error, That variable must be accessed using class name. now we get the default value.(go to example 4)

But It is possible to access those variables from methods and constructors.

Example:1

class sample

{

static int a=50;

static int c=b;

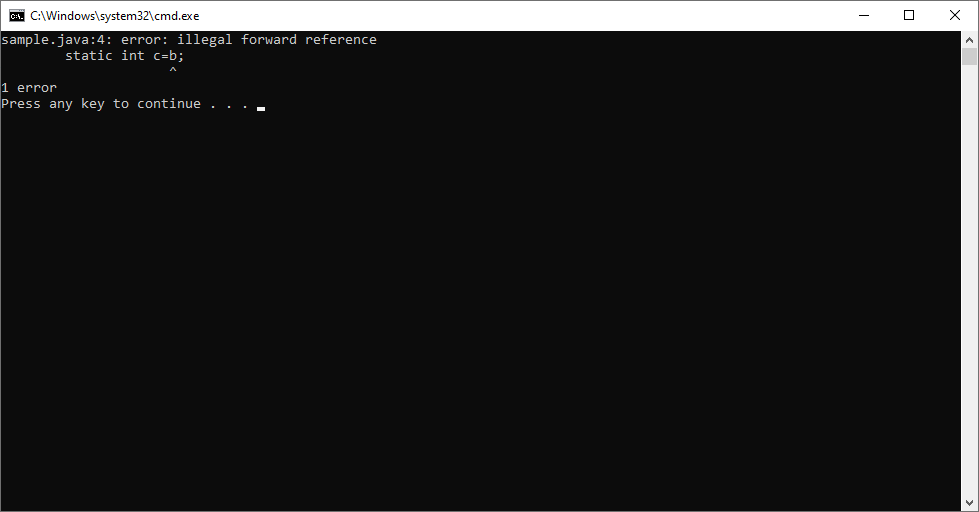
public static void main(String[] args)

{

}

static int b;

}



Example:2

class sample

{

static int a=50;

static{

int c=b;

}

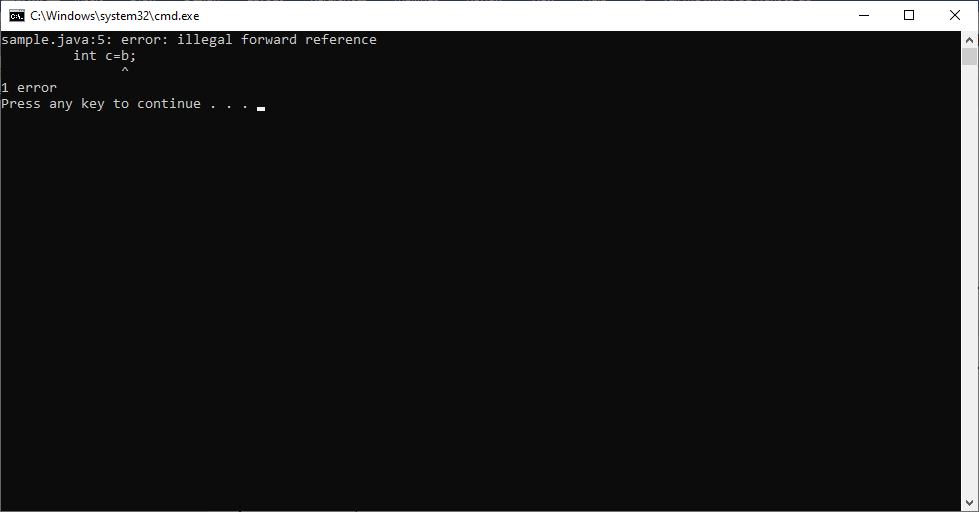
public static void main(String[] args)

{

}

static int b=60;

}



Example:3

class sample

{

static int a=50;

static void display(){

int x=b;

}

public static void main(String[] args)

{

}

static int b=60;

}

Output: no error.

Example:4

import java.io.\*;

class sample

{

static int a=50;

static{

int c=sample.b;

System.out.println(c);

}

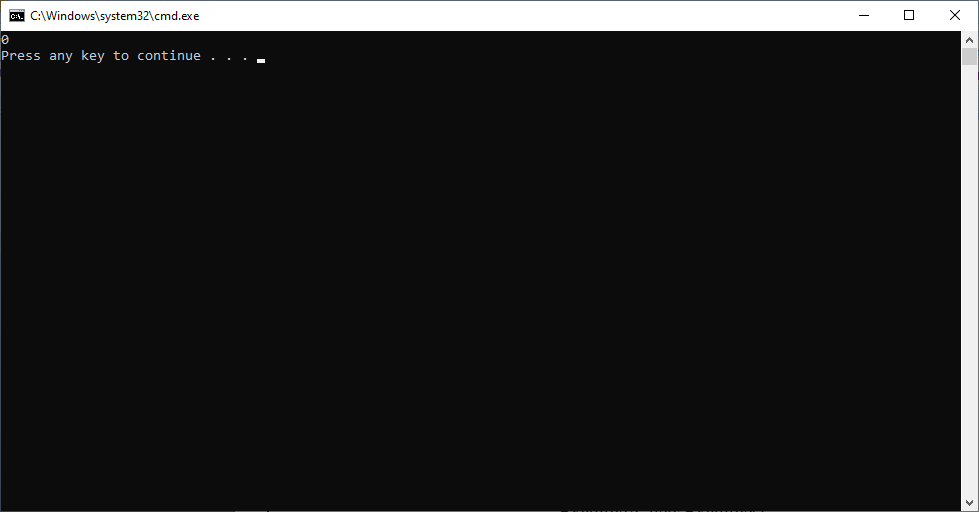
public static void main(String[] args)

{

}

static int b=60;

}



import java.io.\*;

class sample

{

static int a=50;

static int b=a;

//static int c=d; this statement leads to CE: illegal forward reference.

static int c=sample.d;

static int d =40;

public static void main(String[] args)

{

System.out.println(a);

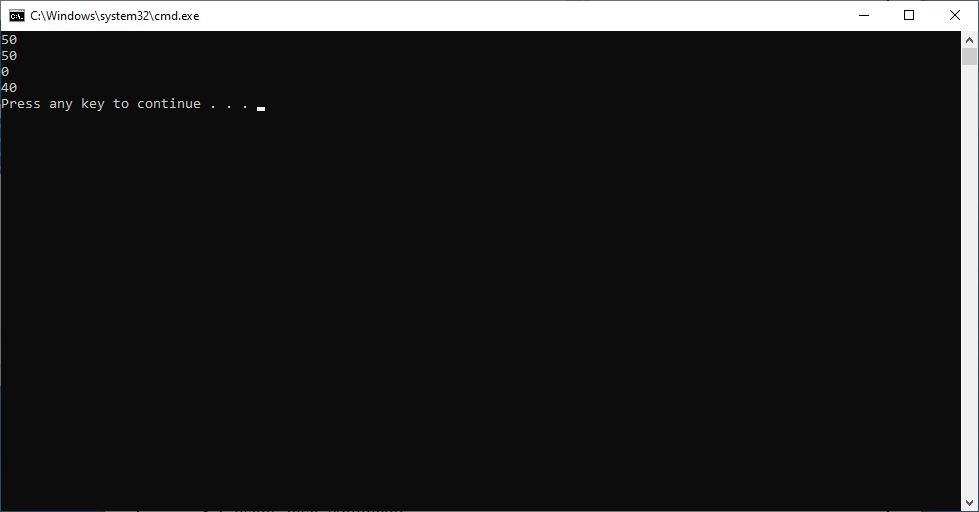
System.out.println(b);

System.out.println(c);

System.out.println(d);

}

}



**13.3Ways to Load the Class into JVM:**  There are 4 ways to load class into JVM.

13.1. using ‘Java’ command.

Example: java ClassName.

13.2 From another class, by calling either static variable/static method.

13.3 From another class, by creating class

Example:

Class sample{

Public static void main(Stirng args[])

{

Example e=new Example();

}

}

13.4. Using Reflection API.

Class.forName(“<<className>>”);

Note:1. Class loaded loads class only once into JVM.

2.JVM executes main() of class,when only it is loaded by “java” command.

**EXECUTION FLOW OF NON-STATIC MEMBERS**

There are two phases while creating the object:

Phase1(Identification Phase): In this phase, non-static variables are created in heap area in continuous memory locations and also identify the non-static blocks in the order they defined from top to bottom.

Phase2(Execution phase):In this phase, the non-static variable are assigned with given values and executes the non-static block logic in java stack area by creating separate stack frame. The non-static variable and block are executed in order they defined from top to bottom.

Phase3: The new keyword invokes the constructor of class.

Note.1.we can create object at class level using static referenced variable and we can’t create object using non-static referenced variable it leads to “java.lang.StackOverflowError.

Example:1 this program demonstrates object is created with static variable.

class sample

{

static sample a=new sample();

public static void main(String[] args)

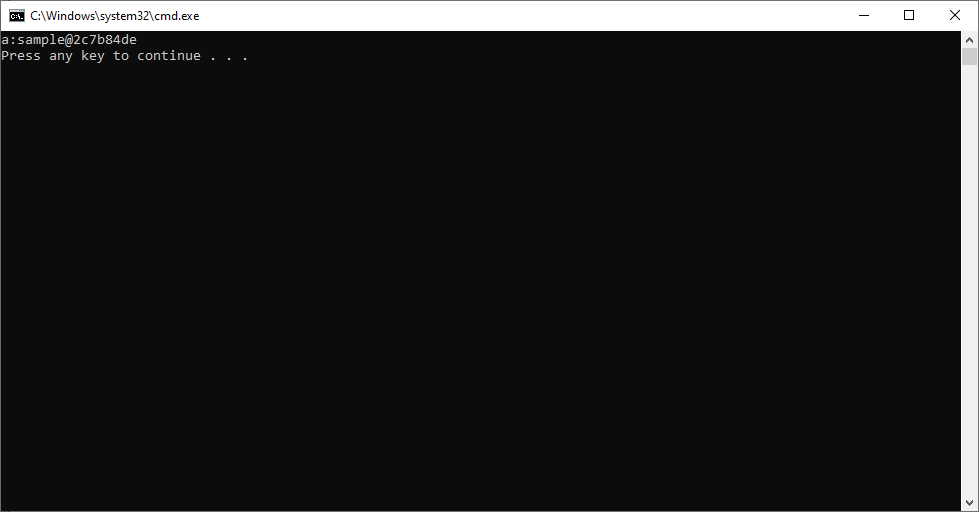
{

System.out.println("a:"+a);

}

}

Output:-



Example:2 This program demonstrates object can’t be create with non-static variable. It leads to “java.lang.stackOverflowError”.

class sample

{

sample s=new sample();

public static void main(String[] args)

{

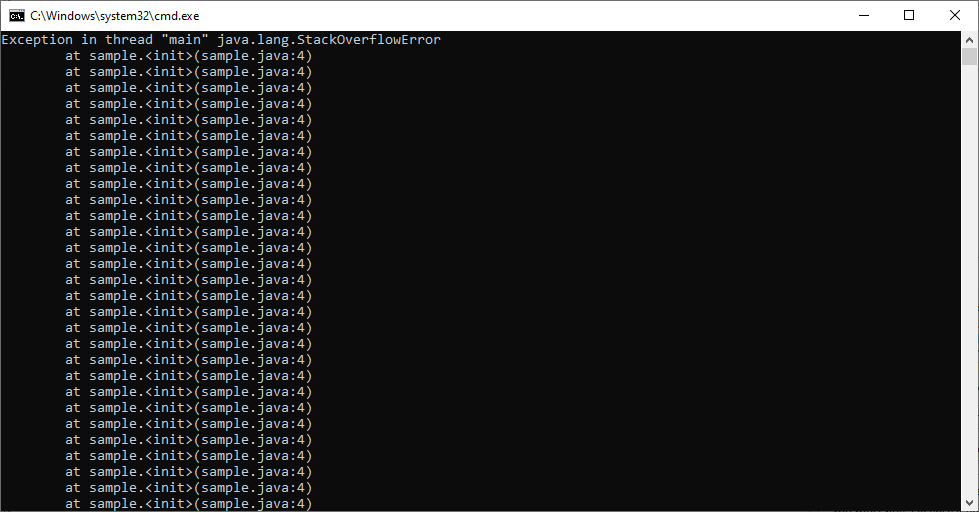
sample x=new sample();

}

}

It does not leads CE.

It leads to Runtime error:java.lang.StackOverflowError.



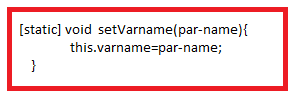
Note:2 we can create same class object in static block. If you create object in non-static block, it Leads to error “java.lang.stackoverflowError.”.

**SETTER AND GETTER METHODS**

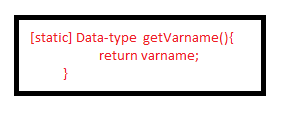
The private variables can only be accessed within the same class(an outside class has no access to it). However , it is possible to access them outside the class if we provide public **get(accessor)** and **set(mutator)** methods. The get method returns the variable value, and the set mehod sets the value.

The getter function starts with “get” followed by variable name where the first letter of the variable is in upper case. The same goes with setters, it starts with “set” followed by variable name where first letter is in uppercase.

Syntax 0f Setter Method:



Syntax Of getter Method:



Example:

import java.io.Console;

class Example

{

int x;

static float y=20;

void setX(int a){

x=a;

}

void setY(float y)

{

this.y=y;

}

int getX(){

return x;

}

float getY(){

return y;

}

}

class sample

{

public static void main(String[] args)

{

Example e=new Example();

e.setX(10);

e.setY(20.4f);

int a=e.getX();

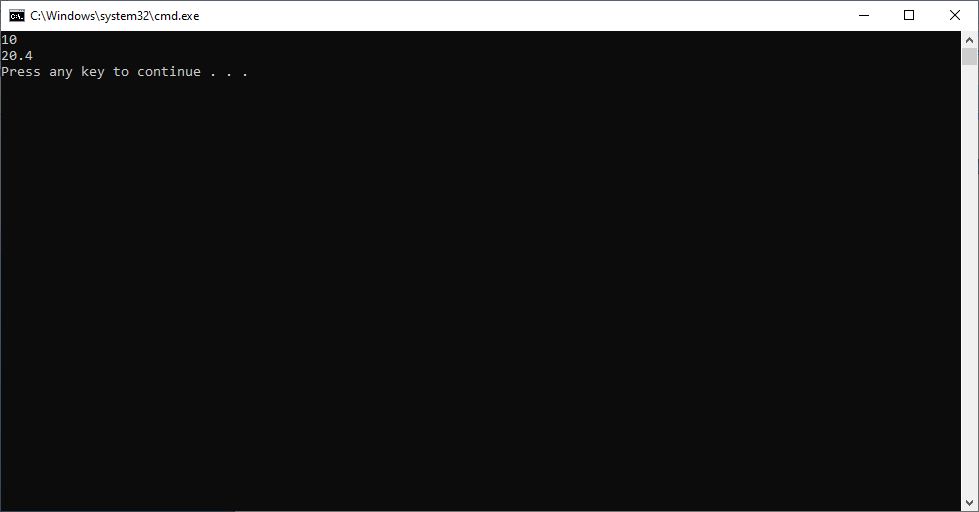
float b=e.getY();

System.out.println(a);

System.out.println(b);

}

}



If the property is boolean type, its getter method must not be started with get, its must be started with **is, can or has**.

For equiry, we use methods. That getter methods should start with ‘is’.

Ex: isActive(), isAlive()…etc.

For taking Permissions , we use getter methods. That getter methods should start with ‘can’.

Ex: canRead(),canWrite(),canChange(),…etc.

For knowing Availability , we use getter methods. That getter method should start with ‘has’.

Ex: hasBalance(),hasNext(),hasMoreElements(),…etc.