Object Oriented Programming using Java
Unit-2

Subject Teacher : Dr. K. V. Metre

Object oriented programming

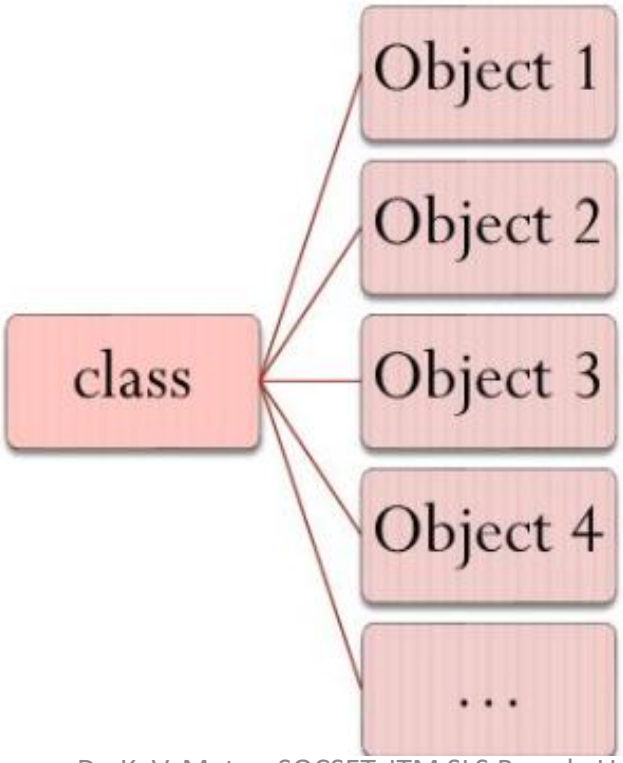
- It is a method of programming that involves the creation of intellectual objects that model a business problem we are trying to solve.(e.g. a bank account)

- With each object, we wodel data associated with it (i.e. its status) and the behaviour associated with it (what our program should allow that object to do).

Object oriented programming

- In creating an object oriented program, we define the properties of a class of objects (e;g; all bank accounts) and then create indivudual objects from this class (e.g. your bank account).

A class is a software design which describes the general properties and bihaviours of something



Individual objects are created from the class design for each actual thing.

Object oriented programming

Benefits of OOP

- Better abstractions(modelling information and behaviour together)
- Better maintainability (more comprehensible , less fragile)
- Better reusability (classes as encapsulated components)

Java - Overview

- A high-level programming language
- Originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).
- Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
- With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications.
- Java is guaranteed to be **Write Once, Run Anywhere.**

Java- Key benefits

- Object oriented
- Platform independant
- Simple
- Secure
- architectural-neutral
- Portable
- Robust
- Multithreaded
- Interpreted
- High performance
- Distributed
- dynamic

Java- Key benefits

- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

- **Platform independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is interpreted by virtual Machine (JVM) on which ever platform it is being run.

- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master.

Java- Key benefits

- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously.

- **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.

Java- Key benefits

- **Architectural-neutral :**Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.

- **Portable:** Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable.

- **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Java- Key benefits

- **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.

- **Distributed:** Java is designed for the distributed environment of the internet.

- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Classes are stored in separate files and are loaded into the Java interpreter only when needed. This means that an application can decide as it is running what classes it needs and can load them when it needs them.

Classes and Objects

- The ***class*** is the unit of programming
- A Java program is a ***collection of classes***
  - Each class definition (usually) in its own **`.java`** file
  - *The file name must match the class name*
- A class describes ***objects (instances)***
  - Describes their common characteristics: is a *blueprint*
  - Thus all the instances have these same characteristics
- These characteristics are:
  - ***Data fields*** for each object
  - ***Methods*** (operations) that do work on the objects

Class syntax:

```
class classname{
datatype instance_variable1;

Datatype instance_variable;

datatype methodname1(parameter_list)
{
//body of mehod
}
datatype methodname2(argument_list)
{
//body of method
}
}
```

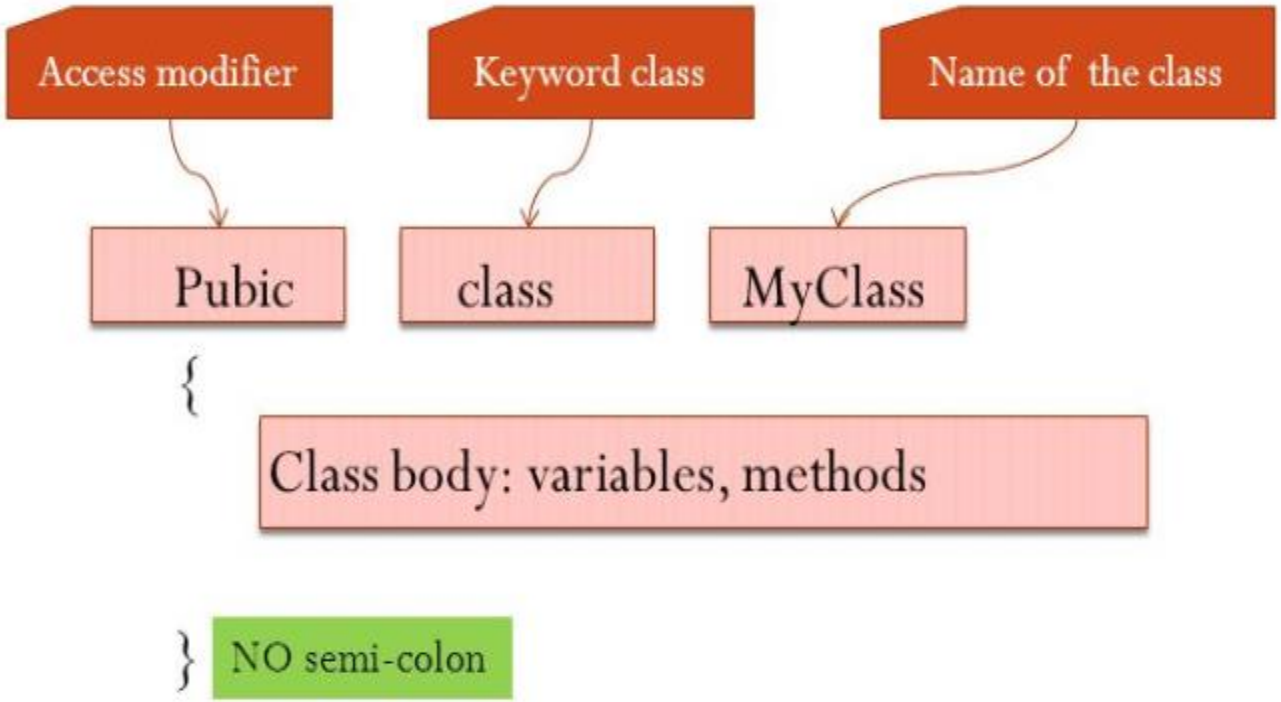
Java : Objects & Classes

- A Java program can be defined as a collection of objects that communicate via invoking each other's methods.

- **Object**
  - Objects have states and behaviors.
    - Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating.
  - If you compare the software object with a real world object, they have very similar characteristics.
  - A software object's state is stored in fields and behavior is shown via methods.
  - An object is an instance of a class.

Java : Objects & Classes

- **Class**
  - A class can be defined as a template that describes the behaviors /states that object of its type support.
  - Anatomy of a Java Class:



Java : Objects & Classes

- **Class**
  - Example:

```
public class Dog{
String breed;
 int age;
String color;
void barking(){ }
 void hungry(){ }
 void sleeping(){ }

}
```

variables

methods

- What is an Object?

Real world objects are things that have:

1) state 2) behavior

Example: your dog:

state – name, color, breed,

behavior – sitting, barking, running

A software object is a bundle of variables (state) and methods (operations).

Class :

Is a group of data and methods (functions).

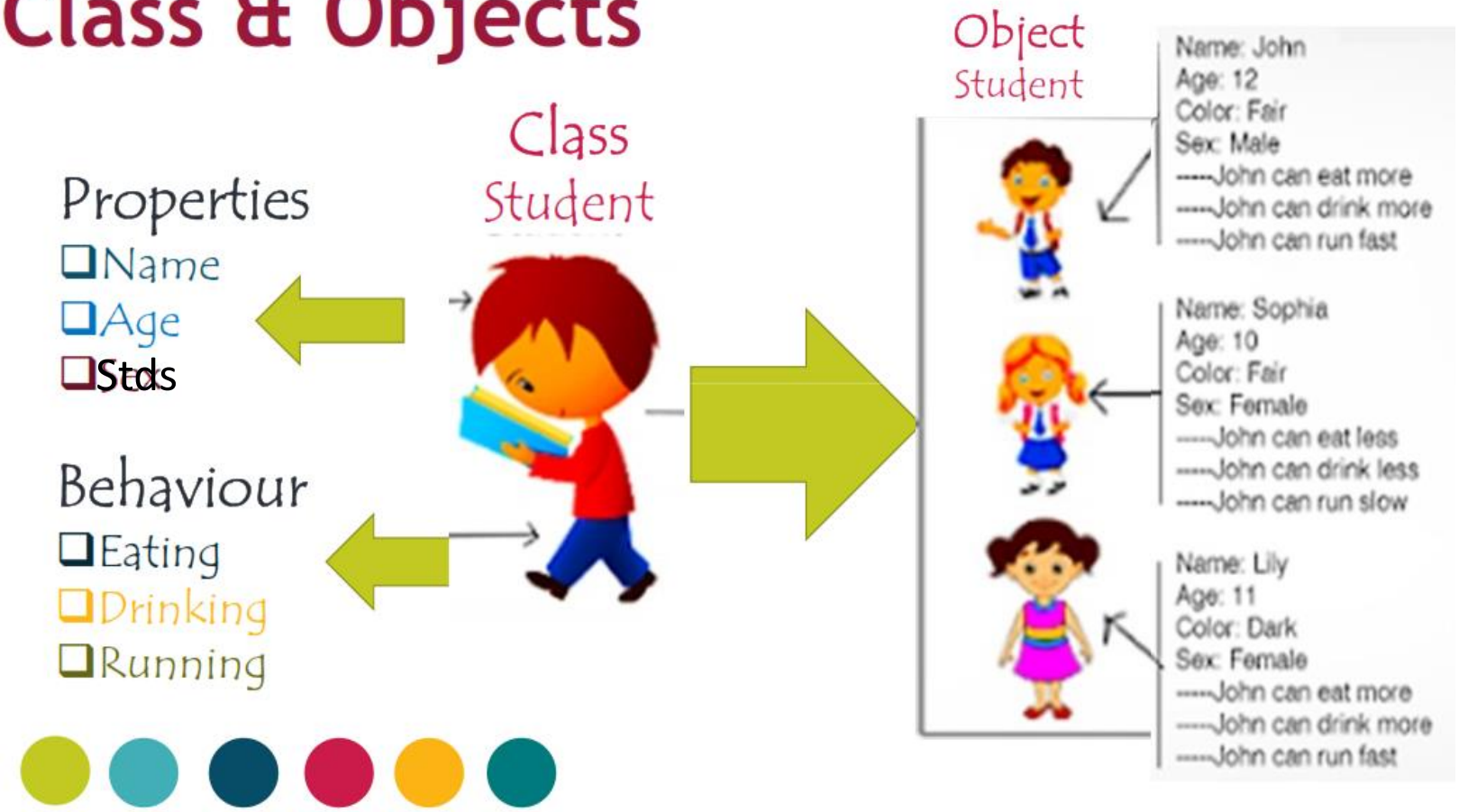
Object :

Is an instance of a class, which is similar to a variable, defined as an instance of a type. An object is what you actually use in a program since it contains values and can be changed.

Method :

Is a function contained within the class. You will find the functions used within a class often referred to as methods in programming literature.

Class & Objects



Creating object:

- To access the properties and methods of a class, we must declare a variable of that class type.
- This variable does not define an object. Instead, it is simply a variable that can refer to an object.
- We must acquire an actual, physical copy of the object and assign it to that variable.
- We can do this using new operator.
- The new operator dynamically allocates memory for an object and returns a reference to it.
- This reference is, more or less, the address in memory of the object allocated by new.
- This reference is then stored in the variable.
- Thus, in Java, all class objects must be dynamically allocated.

New operator

- Using the new keyword in java is the most basic way to create an object. This is the most common way to create an object in java.
- By using this method we can call any constructor we want to call (no argument or parameterized constructors).

Example:

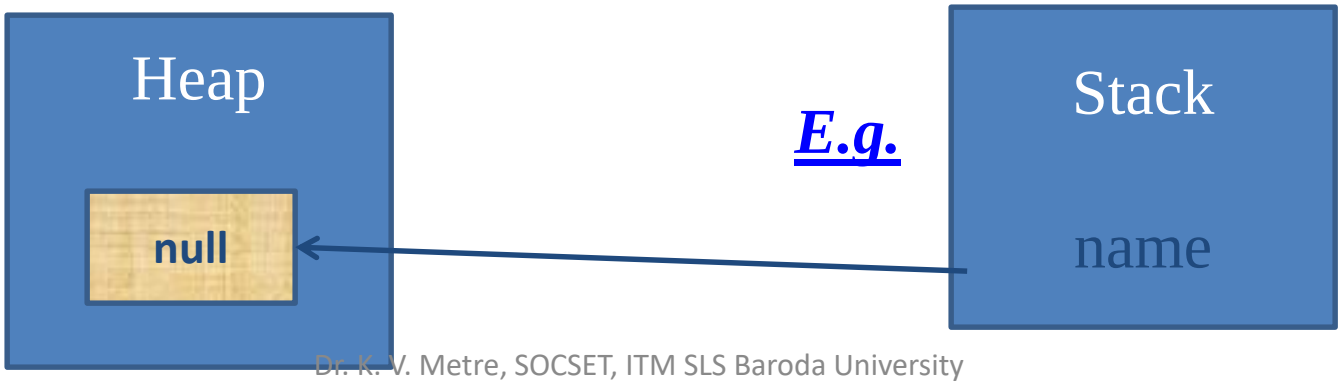
- mybox = new Box();

Object Creation Process

Object Creation Process involves following steps

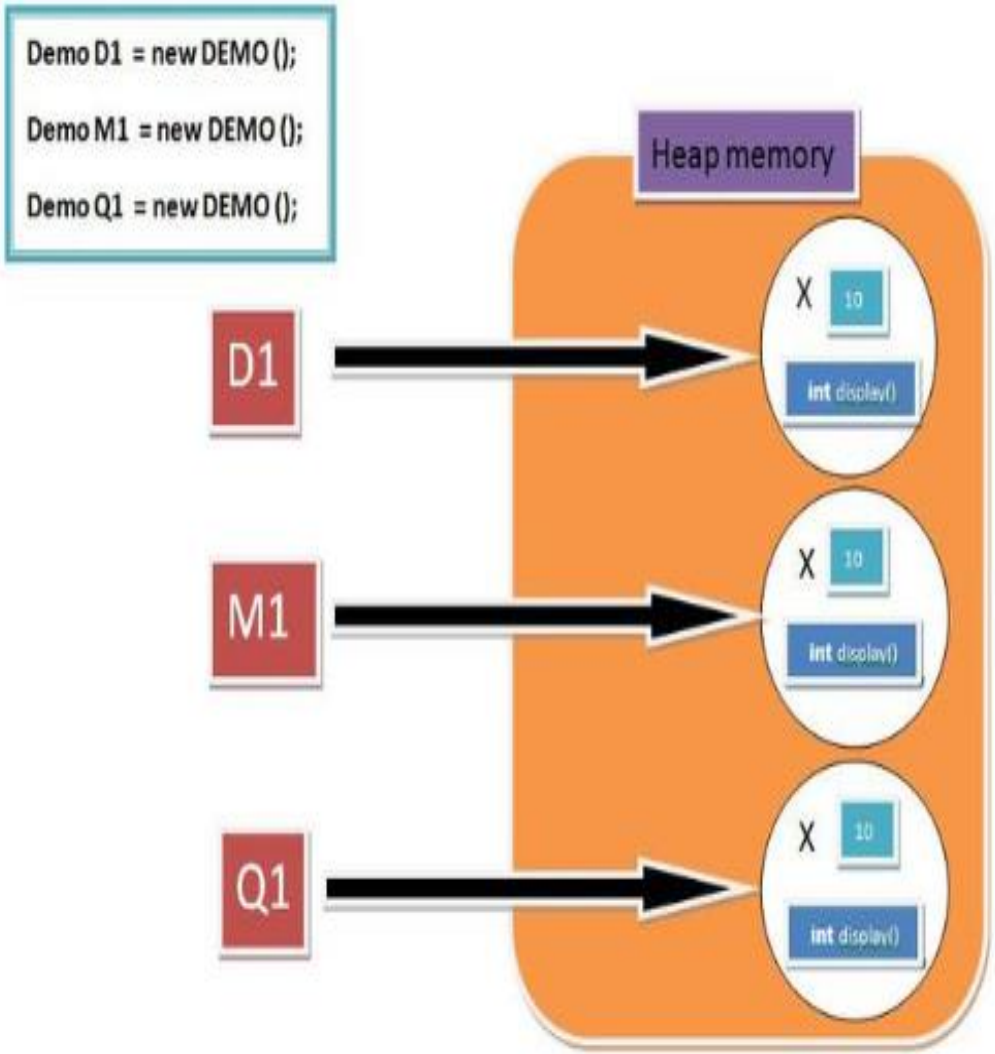
E.g. Box name = new Box( );

1. new operator causes construction of the object Dynamically
2. JVM gives default values to the instance variables of Object
3. All static and non-static blocks are executed
4. Constructor is Executed
5. new operator binds the object to reference variable



```
class Box //define a class
 { private :
      int length;
      int breadth;
      int height;
public :
void setdata(int l, int b, int h)     //member function to set data
    { length = l; breadth = b;  height = h;  }
void showdata() //member function to display data
   { System.out.println("length ="+length+ "breadth ="+breadth+"height
     ="+height);}
public static void main( String args[])
{   Box b1 = new Box();
    b1.setdata(10,20,30);
    b1.showdata();
}
}
```

```
import java.io.*;
class Demo {

    int x = 10;
    int display()
    {
        System.out.println("x = " + x);
        return 0;
    }
}
class Main {
    public static void main(String[] args)
    {
          // create instances
        Demo D1 = new Demo();

        Demo M1 = new Demo();

        Demo Q1 = new Demo();
    }
}
```

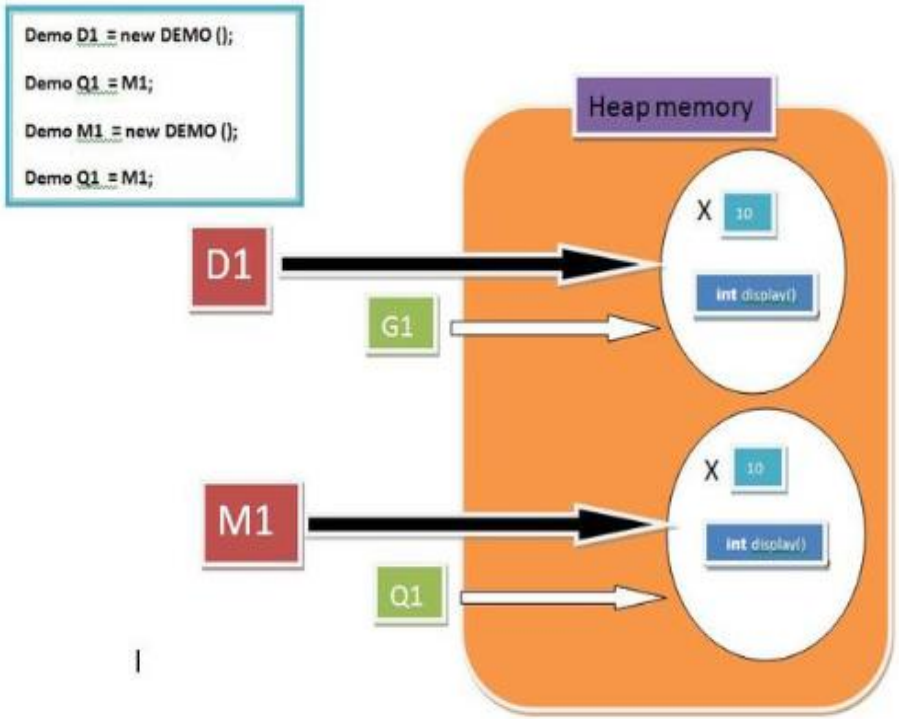

Demo D1 = new DEMO ();
Demo M1 = new DEMO ();
Demo Q1 = new DEMO ();
Heap memory
D1
M1
Q1
X
10
int display()
X
10
int display()
X
10
int display()

```
// Accessing instance methods

// Pointing to same instance memory

import java.io.*;
class Demo {
    int x = 10;
    int display()
    {
        System.out.println("x = " + x);
        return 0;
    }
}
class Main {
    public static void main(String[] args)
    {
        // create instance
        Demo D1 = new Demo();

        // point to same reference
        Demo G1 = D1;

        Demo M1 = new Demo();

        Demo Q1 = M1;

        // updating the value of x using G!
        // reference variable
        G1.x = 25;

        System.out.println(G1.x); // Point 1

        System.out.println(D1.x); // Point 2
    }
```

ity

Output

```
25
25
```


Demo D1 = new DEMO ();
Demo Q1 = M1;
Demo M1 = new DEMO ();
Demo Q1 = M1;
Heap memory
D1
G1
M1
Q1
X
10
int display()
X
10
int display()

Constructors

Constructors .

- It is set of Instructions designed to perform initial activities or task after an object creation is completed
- Constructor is a special method in java
- Constructor must not have any return type
- The name of the constructor is same as the name of the class
- Constructor can have any access modifier
- If Constructor have private modifier, object can't be created from other classes.
- Every constructor has, as its first statement, either call to other constructor of same class or call to its super class constructor

Types of Constructors

Constructors are of two type

1) Default Constructor

This constructor is generated by the compiler when the user don't write any constructor

2) User Defined Constructor

This is the constructor written by the user in his program

There are two types of User defined Constructor

1. No-Argument Constructor

2. Parameterized Constructor

Constructor Overloading

- Constructor overloading, in this the name remains the same just the arguments change
- In this the return type may or may not change
- To call the super class constructor use the *"super"* keyword
- ***This Mechanism is similar to Method Overloading***

```
class Game{
public Game( ){ }
public Game(int i) { }
public Game( String a, String b){ }
```

```
class Box //define a class
 { private
     int length;
     int breadth;
     int height;
public
      Box() {length = 10; breadth = 10;  height = 10};
      Box(int l, int b, int h ) {length = l; breadth = b;  height = h;  }
      void setdata(int l, int b, int h)     //member function to set data
                  { length = l; breadth = b;  height = h;  }
      void showdata() //member function to display data
        { System.out.println("length ="+length+ "breadth ="+breadth+"height ="+height);}
      public static void main( String args[])
          {   Box b1 = new Box();
               // b1.setdata(10,20,30);
              System.out.println("\nNo argument Constructor values: \n");
              b1.showdata();

              Box b2 = new Box(10,20,30);
              System.out.println("\nParameterized Constructor values: \n");
              b2.showdata();
          }
}
```

Using Object as a Parameters

- When we pass a primitive type to a method, it is passed by value.
- But when we pass an object to a method, objects are passed by call-by-reference.
- Java does this interesting thing that's sort of a hybrid between pass-by-value and pass-by-reference.
- While creating a variable of a class type, we only create a reference to an object.
- Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
- This effectively means that objects act as if they are passed to methods by use of call-by-reference.
- Changes to the object inside the method do reflect in the object used as an argument.

Parameterized Constructor-2

```
// Java program to demonstrate one object to
    initialize another
class Box
{   double width, height, depth;
// It takes an object of type Box. This constructor
    use  one object to initialize another
    Box(Box ob)
    {       width = ob.width;
            height = ob.height;
            depth = ob.depth;
    }
// constructor used when all dimensions specified
    Box(double w, double h, double d)
    {       width = w;
            height = h;
            depth = d;    }
    double volume() // compute and return volume
    {  return width * height * depth;     }
}
```

```
// driver class
public class Test {
    public static void main(String args[])
    {
    // creating a box with all dimensions
    specified
        Box mybox = new Box(10, 20, 15);
// creating a copy of mybox
        Box box1 = new Box(mybox);
        double vol;
            // get volume of mybox
         vol = mybox.volume();
System.out.println("Volume of mybox is " +
                                    vol);
            // get volume of box1
        vol = box1.volume();
System.out.println("Volume of box1 is " +
    vol);   }
}
```

Constructor Chaining

- Constructor Chaining is calling of the other constructor of same class
- Constructor Chaining is possible with ***this*** keyword
- ***this*** must be the first statement of the constructor body
- ***this*** can be parameterized or non parameterized
- ***super*** keyword is used to call super class constructor

Method Overloading/ Polymorphism

Method Overloading

- Method Overloading is the mechanism in which the methods have same name but different signature
- Method Overloading happens in the same class
- Advantage of method overloading
  Method overloading *increases the readability of the program*.

- There are 3 ways to overload the method in java
  1) By changing number of arguments
     add(int, int)
     add(int, int, int)
  2) By changing the data type of arguments
     add(int, int) add(2, 5)
     add(int, float) add(5, 10.5)
  3) By changing Sequence of Data type of parameters.
     add(int, float)
     add(float, int)

- Signature of the Overloaded method should be Different
- Overloaded methods are not required to have the same

  return type

  if two methods have same name, same parameters and have

  different return type, then this is not a valid method

  Overloading

  int add(int, int)

  float add(int, int)
- This is similar to constructor overloading
- Subclass can Overload a inherited method of its base class
- **Method overloading** is an example of **Static Polymorphism**.

1. Static Polymorphism is also known as compile time binding or early binding.
2. **Static binding** happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

1) Method Overloading: changing no. of arguments

```
class Adder{
    static int add(int a, int b)
    {       return a+b;
    }
    static int add(int a, int b, int c)
    {       return a+b+c;
} }
class TestOverloading1{
    public static void main(String[] args){

            System.out.println(Adder.add(11,11));
            System.out.println(Adder.add(11,11,11));
    } }
```

Output :

22
33

2) Method Overloading: changing data type of arguments

```
class Adder{
static int add(int a, int b){
      return a+b;
  }
static double add(double a, double b){
     return a+b;
   }
}
class TestOverloading2{
     public static void main(String[] args){
              System.out.println(Adder.add(11,11));
              System.out.println(Adder.add(12.3,12.6));
     } }
```

Output

22

24.9

3) Method Overloading: changing sequence of arguments

```
class Adder{
static float add(int a, float b){
      return a+b;
  }
static float add(float a, int b){
     return a+b;
   }
}
class TestOverloading2{
     public static void main(String[] args){
            System.out.println(Adder.add(11,11.0));  //method is defined as static
            System.out.println(Adder.add(12.3,12));
     } }
```

Output

22.0

24.3

- Can we overload java main() method?
- Yes, by method overloading. You can have any number of main methods in a class by method overloading.
  But JVM calls main() method which receives string array as arguments only.

```
class TestOverloading{
    public static void main(String[] args){
        System.out.println("main with String[]");}
    public static void main(String args){
        System.out.println("main with String");}
    public static void main(){
        System.out.println("main without args");}
     }
```

- Output:

  main with String[]

Returning a value

- Method returns value

```
class Box //define a class
{ private
    int length;
    int breadth;
    int height;
public
     Box() {length = 10; breadth = 10;  height = 10;};
     Box(int l, int b, int h ) {length = l; breadth = b;  height = h;  }
     void setdata(int l, int b, int h)     //member function to set data
                  { length = l; breadth = b;  height = h;  }
     void showdata() //member function to display data
       {
         System.out.println("length ="+length+"breadth ="+ breadth+"height ="+height);}
   int volume()
  {   return length*breadth*height;}

     public static void main( String args[])
         {   Box b1 = new Box();
            // b1.setdata(10,20,30);
  System.out.println("\n No argument Constructor values: \n");
  b1.showdata();
  int v = b1.volume();
  System.out.println("\n Volume = “+v);
   Box b2 = new Box(10,20,30);
  System.out.println("\nParameterized Constructor values: \n");
             b2.showdata();
         }
}
```

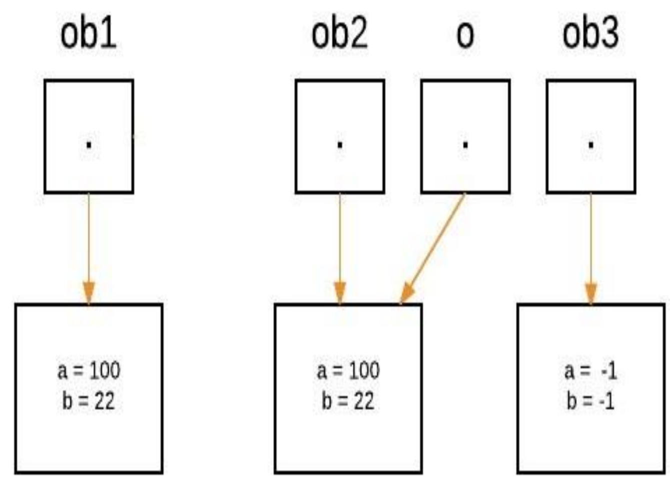
passing objects to methods

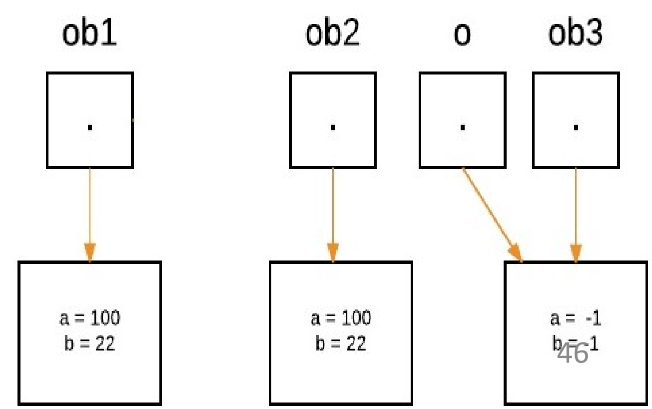
```
// Java program to demonstrate objects  passing to
    methods.
class Demo
{    int a, b;
     Demo(int i, int j)
     {
          a = i;   b = j;
     }
     // return true if o is equal to the invoking  object
     boolean equalTo(Demo o)
     {
          return (a == o.a  && b == o.b );           }
}
public class Test
{
     public static void main(String args[])
     {
     Demo ob1 = new Demo(100, 22);
     Demo ob2 = new Demo(100, 22);
     Demo ob3 = new Demo(-1, -1);
     System.out.println("ob1 == ob2: " + ob1.equalTo(ob2
     System.out.println("ob1 == ob3: " + ob1.equalTo(ob3
     }   }
```

- Output:

ob1 == ob2: true

ob1 == ob3: false





```
class Box
  {   double width, height, depth;
     Box(Box ob)          {
                width = ob.width;
                height = ob.height;
               depth = ob.depth;
          }
     Box(double w, double h, double d)
     {         width = w;
               height = h;
               depth = d; }
     void add( Box b1)
     {         double w, h, d;
                w = width + b1.width;
               h = height + b1.height;
               d = depth + b1.depth;
            Box  ans = new Box(  w, h, d);
            ans.show();    }
     void show()
     {         System.out.println("After add  " );
               System.out.println("Width = " + width);
               System.out.println("Height =  " + height);
               System.out.println("Depth =  " + depth);
    } }
```

```
// driver class
public class Main
    {
     public static void main(String args[])
     {
          // creating a box with all dimensions
         Box b1 = new Box(10, 20, 15);
         Box b2 = new Box(10, 20, 30);
    Box t = new Box();
      t= b1.add(b2);
     t.show();                                    }
}
```

- **Output :**

```
After add
Width = 20.0
Height =  40.0
Depth =  45.0
```

Methods returning objects

```
class Box
{   private
       int length;
       int breadth;

public
     Box() {length = 0; breadth = 0;  }
     Box(int l, int b ) {
      length = l; breadth = b;  }
     void setdata(int l, int b)     //member function to set data
     { length = l; breadth = b;   }
     void showdata() //member function to display data
     {

      System.out.println("length="+length+"\tbreadth
                ="+ breadth);
     }
      int area()
                {   return length*breadth;    }

    Box  add( Box b2)
 {
        Box temp = new Box();
        temp.length = length + b2.length;
        temp.breadth = breadth + b2.breadth;
        return temp;
}
```

```
Class Test  {
public static void main( String args[])
 {   Box b1 = new Box(20,20);
     b1.showdata();
      int a = b1.area();
      System.out.println("\n Area ="+ a);

      Box b2 = new Box(10,20);
          b2.showdata();

      Box b3 = b1.add(b2);
       b3.showdata();
}
}
```

Output :

```
length=20          breadth   =20
 Area =400
length=10          breadth   =20
length=30          breadth   =40
```

Returning Objects

- In java, a method can return any type of data, including objects.
- the **incrByTen( )** method returns an object in which the value of a (an integer variable) is ten greater than it is in the invoking object.

```
// Java program to demonstrate returning  of objects
class Demo
{  int  a;
     Demo() {
              a = 0; }
     Demo(int  i) {
               a = i; }
     Demo incrByTen() // This method returns an object
   {
       Demo temp = new Demo();
       temp.a = a + 10;
       return temp;
   }
}
```

```
// Driver class
public class Test
{
   public static void main(String args[])
   {
      Demo ob1 = new Demo(2);
      Demo ob2;
      ob2 = ob1.incrByTen();

      System.out.println("ob1.a =" + ob1.a);
      System.out.println("ob2.a = " + ob2.a);
   }
}
```

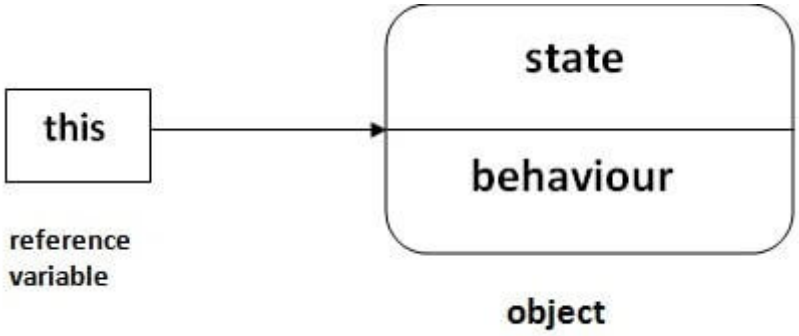
Output:

ob1.a =2

ob2.a = 12

this keyword in Java

- There can be a lot of usage of **Java this keyword**.
- In Java, this is a **reference variable** that refers to the current object.



Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01 this can be used to refer current class instance variable.

02 this can be used to invoke current class method (implicity)

03 this() can be used to invoke current class Constructor.

04 this can be passed as an argument in the method call.

05 this can be passed as argument in the constructor call.

06 this can be used to return the current class instance from the method

```
class Student{
    int rollno;
    String name;
    float fee;
    Student(int rollno, String name, float fee)
    {     rollno = rollno;
           name = name;
           fee = fee;          }
void display(){
    System.out.println(rollno+" "+name+" "+fee);}     }
class TestThis1{
    public static void main(String args[]){
    Student s1=new Student(111,"ankit",5000f);  Stude
    nt s2=new Student(112,"sumit",6000f);
    s1.display();
    s2.display();
}}
```

- OUTPUT

0 null 0.0

0 null 0.0

parameters (formal arguments) and instance variables are same.

Using this keyword

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
    this.rollno=rollno;
    this.name=name;
    this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);
    }
}

class TestThis2{
    public static void main(String args[]){
        Student s1=new Student(111,"ankit",5000f);
        Student s2=new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
}
}
```

- Output :

111 ankit 5000.0

112 sumit 6000.0

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword

```
Student(int r, String n, float f )
{
rollno=r;
name=n;
fee=f;
}
```

2) this: to invoke current class method

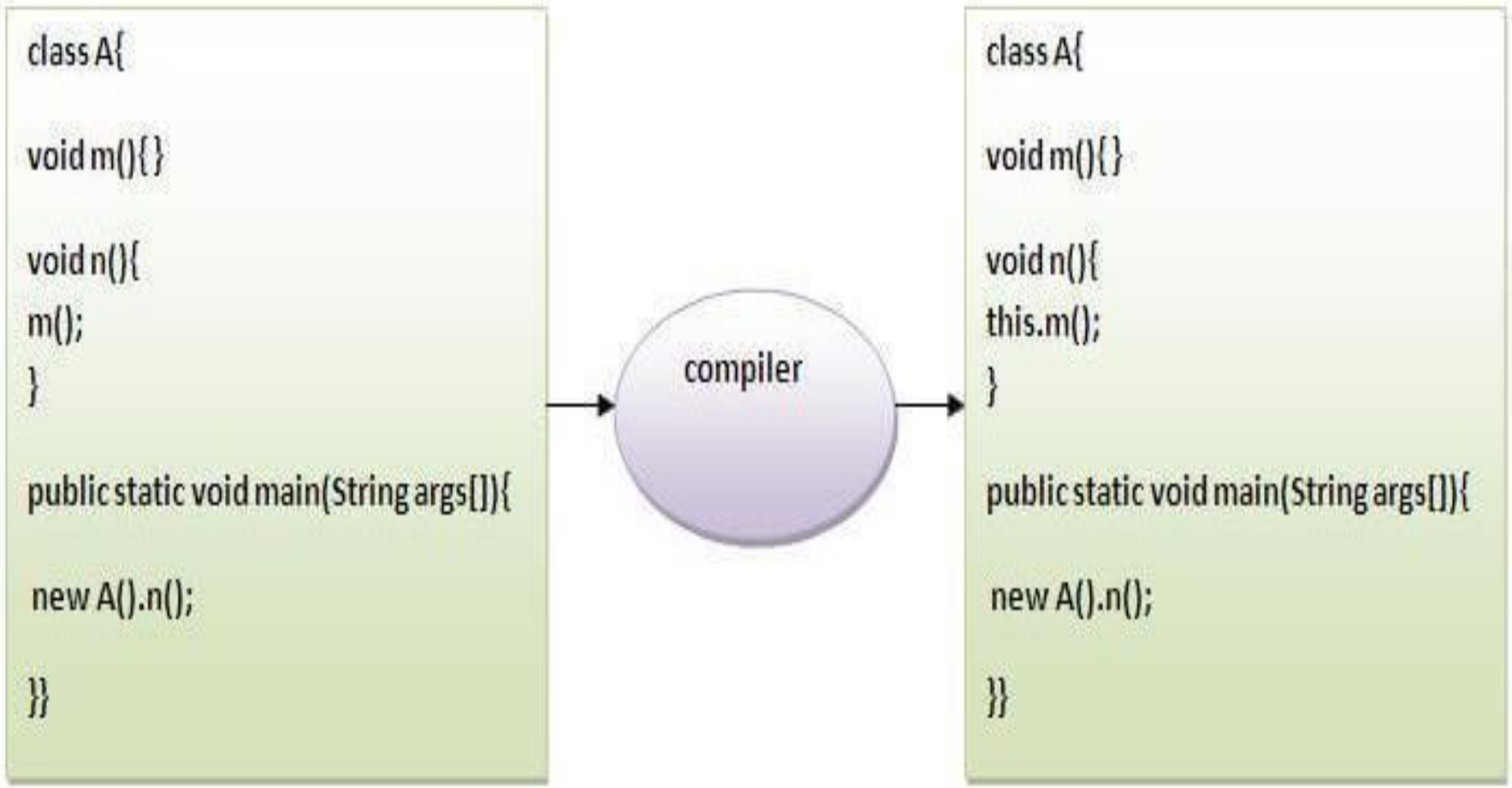
invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method

```
class A{
    void m(){System.out.println("hello m");}
    void n(){
    System.out.println("hello n");
                    //m();
    this.m();
}
}
class TestThis4{
    public static void main(String args[]){
    A a=new A();
    a.n();
    }}
```

- Output

hello n hello m

- **2) this: to invoke current class method**



3) this() : to invoke current class constructor

- this() constructor call can be used to invoke the current class constructor.
- It is used to reuse the constructor. it is used for constructor chaining. **Calling default constructor from parameterized constructor:**

```
class A{
      A( ){
          System.out.println("hello a");}
      A(int x){
        this();
       System.out.println(x);
       }
        }
class Test{
  public static void main(String args[]){
    A a=new A(10);
}}
```

output : hello a
10

Calling parameterized constructor from default constructor:

- call to this() must be the first statement in constructor

```
class A{
    A(){
        this(5);
       System.out.println("hello a");
         }
     A(int x){
        System.out.println(x);
       }
      }
class Test{
    public static void main(String args[]){
      A a=new A();
}}
```

Output : 5

hello a

Static keyword:

- **static keyword** in Java is used for memory management mainly.
- We can apply static keyword with variables, methods, blocks and nested classes.
- The static keyword belongs to the class than an instance of the class.

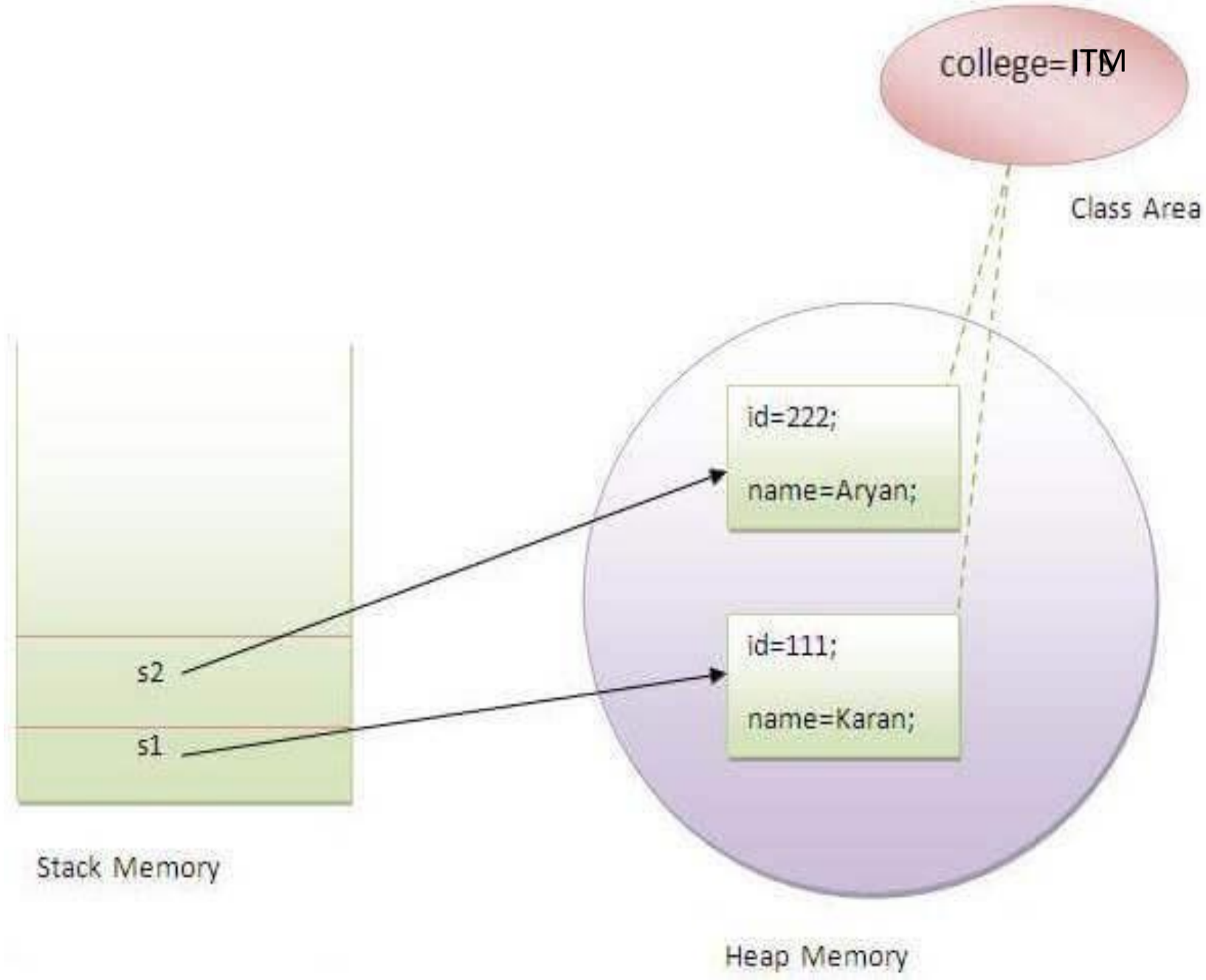
The static can be:

- Variable (also known as a class variable)
- Method (also known as a class method)
- Block
- Nested class

1) Java static variable

- If you declare any variable as static, it is known as a static variable.
- The static variable can be used to refer to the common property of all objects (which is not unique for each object),
- the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

```
class Student{
    int rollno;
    String name;
    static String college = "ITM";
```


college=ITM
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

Java static variable

```
class Student{
  int rollno;//instance variable
  String name;
  static String college ="ITM";//sta
tic variable

  //constructor
  Student(int r, String n){
  rollno = r;
  name = n;
  }
  //method to display the values
  void display (){System.out.printl
n(rollno+" "+name+" "+college);}
}
```

```
public class Test{
  public static void main(String args[]){
  Student.change();//calling change met
hod
  //creating objects
  Student s1 = new Student(111,"Karan"
);
  Student s2 = new Student(222,"Aryan"
);
  Student s3 = new Student(333,"Sam");

  //calling display method
  s1.display();
  s2.display();
  s3.display();
  }
}
```

Output: *111 Karan ITM*
222 Aryan ITM
333 Sam ITM

static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to demonstrate the use of an instance va
      riable
//which get memory each time when we create an objec
      t of the class.
class Counter{
int count=0;
      //will get memory each time when the instance is cr
      eated

Counter(){
                  count++;//incrementing value
                  System.out.println(count);
      }

public static void main(String args[]){
//Creating objects
      Counter c1=new Counter();
      Counter c2=new Counter();
      Counter c3=new Counter();
}
}
```

Output:

```
1
1
1
```

```
//Java Program to demonstrate the use of an instance va
      riable
//which get memory each time when we create an objec
      t of the class.
class Counter{
int count=0;
      //will get memory each time when the instance is cr
      eated

Counter(){
                  count++;//incrementing value
                  System.out.println(count);
      }

public static void main(String args[]){
//Creating objects
      Counter c1=new Counter();
      Counter c2=new Counter();
      Counter c3=new Counter();
}
}
```

Output:

```
1
2
3
```

2) Java static method

- If you apply static keyword with any method, it is known as static method.
- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

2) Java static method

```
//Java Program to get the cube of a given number using the static method
  class Calculate{
  static int cube(int x){
  return x*x*x;
  }
  public static void main(String args[]){
  int result=Calculate.cube(5);
  System.out.println(result);
  }
}
```

Output : 125

2) Java static method

Restrictions for the static method:

- static method can not use non static data member or call non-static method directly.
- this and super cannot be used in static context.

```
class A{
 int a=40;//non static
 public static void main(String args[]){
  System.out.println(a);
 } }
```

Output : Compile Time Error

2) Java static method

If you apply static keyword with any method, it is known as static method.
A static method belongs to the class rather than the object of a class.
A static method can be invoked without the need for creating an instance of a class.
A static method can access static data member and can change the value of it.

```
//Java Program to demonstrate the use of a static method.

class Student{
    int rollno;
    String name;
    static String college = “ITM";

      //static method to change the value of static variable
    static void change(){
      college = “ITM SLS";
       }

    //constructor to initialize the variable
    Student(int r, String n){
        rollno = r;
       name = n;
    }
    //method to display values

    void display(){
      System.out.println(rollno+" "+name+" "+college);}
  }
```

```
//Test class to create and display the values of
      object
public class TestStaticMethod{

   public static void main(String args[]){

   Student.change();//calling change method
   //creating objects

   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method

   s1.display();
   s2.display();
   s3.display();
   }
|
```

Restrictions for the static method

There are two main restrictions for the static method.

1) The static method can not use non static data member or call non-static method directly.

2) this and super cannot be used in static context.

```
class A{
 int a=40;//non static

 public static void main(String args[]){
  System.out.println(a);
        }
}
```

Output:

Compile Time Error

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of class loading.

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Output : static block is invoked

Hello main

Why is the Java main method static?

- It is because the object is not required to call a static method.
- If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Access Modifier

Access Modifiers in Java

- As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor, variable, method, or data member.

There are four types of access modifiers available in java:

- Default – No keyword required
- Private
- Protected
- Public

Java has 3 Access Modifier and 4 Accessibility Modes

- **Public**

  The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

- **Private**

  The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

- **Protected**

  The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

- **Default**

- When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier by default.
- The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.

Visibility Control

Modifier / Access Control	public	Protected	default (friendly)	Private
Same Class	Yes	Yes	Yes	Yes
Subclass in Same Package	Yes	Yes	Yes	No
Other Class in same Package	Yes	Yes	Yes	No
Subclass in other Package	Yes	Yes	No	No
Non-subclasses in other package	Yes	No	No	No

Example of private access modifier

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

If you make any class constructor private, you cannot create the instance of that class from outside the class

Example of public access modifier

```
/save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

- Output : Hello

Example of protected access modifier

```
//save by A.java
package pack;
public class A{
protected void msg()
    {   System.out.println("Hello"); }
}
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

Output : Hello

Example of default access modifier

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
    }
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[])
    {
         A obj = new A();  //Compile Time Error
        obj.msg();   //Compile Time Error
       }
}
```

since A class is not public, so it cannot be accessed from outside the package.

Java Object finalize() Method :

- Finalize() is the method of Object class.
- This method is called just before an object is garbage collected.
- finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.
- The primary purpose of the finalize() method is to allow an object to perform cleanup operations, such as releasing resources or closing connections, before it is removed from memory.
- While the use of finalize() has been somewhat deprecated in modern Java in favor of other mechanisms (such as try-with-resources and AutoCloseable), it's still worth understanding its importance.

- **Syntax :**

  protected void finalize() **throws** Throwable

Throwable - the Exception is raised by this method

```
public class Test {
    public static void main(String[] args)
   {
      Test obj = new Test();
      System.out.println(obj.hashCode());
      obj = null;
      // calling garbage collector
      System.gc();
      System.out.println("end of garbage collection");

   }
   @Override
   protected void finalize()
   {
      System.out.println("finalize method called");
   } }
```

Output :

```
2018699554
end of garbage collection
finalize method called
```

Recursion

- Each time a method is called, a new space is allocated on the internal stack for all the variable in the method, which means there is no reason you can't call the same method again – a new set of variables will be allocated on the stack automatically
- Recursion is a technique in which a method calls itself.
- This technique provides a way to break complicated problems down into simple problems which are easier to solve.
- Examples of recursion are:
  - Factorial of number
  - Fibonacci series

Recursion

```
static int factorial(int n){
if (n == 1)
return 1;
else
return(n * factorial(n-1));

}

public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
}
}
```

Recursion

- Each time a method is called, a new space is allocated on the internal stack for all the variable in the method, which means there is no reason you can't call the same method again – a new set of variables will be allocated on the stack automatically
- Recursion is a technique in which a method calls itself.
- This technique provides a way to break complicated problems down into simple problems which are easier to solve.
- Examples of recursion are:
  - Factorial of number
  - Fibonacci series

End of Unit-2