Classes and Objects: Basic concepts of OOPS, Classes and Objects, Modifiers, Passing arguments, Constructors, Overloaded Constructors, Overloaded Operators, Static Class Members, Garbage Collection.

Inheritance: Basics of inheritance, Inheriting and Overriding Superclass methods, Calling Superclass Constructor, Polymorphism, Abstract Classes, Final Class.

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc.
Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- ❖ Object
- ❖ Class
- ❖ Inheritance
- ❖ Polymorphism
- ❖ Abstraction
- ❖ Encapsulation

Objects and Classes in Java

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

What is an object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (**tangible** and **intangible**). The example of an intangible object is the banking system.

An object has three characteristics:

State: represents the data (**value**) of an object.

Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- ❖ An object is *a real-world entity*.
- ❖ An object is *a runtime entity*.
- ❖ The object is *an entity which has state and behavior*.
- ❖ The object is *an instance of a class*.

What is a class in Java?

A Class is a user defined data type which contain data member , member functions with different access specifiers like private, protected, public & default. A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

Fields

Methods

Constructors

Blocks

Nested class and interface

Syntax to declare a class:

```
class <class_name>
{
   field;      // data members
   method;   // members fucntiosn
}
```

Instance variable(data member) in Java

A variable which is created inside the class but outside the main method is known as an instance variable.

Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created.

That is why it is known as an instance variable.

```
Class    Rectangle
{
          int length, width;  // instance variable

}
```

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

Code Reusability
Code Optimization

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in **Heap** memory area.

=>Write any Example to create any class to print object values

3 Ways to initialize object

There are 3 ways to initialize object in Java.

- By reference variable (with new keyword)
  - z
- By constructor

Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, an anonymous object is a good approach.

For example:

```
Calculation obj = new Calculation();

new Calculation();      //anonymous object

new Calculation().fact(5);
```

Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

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

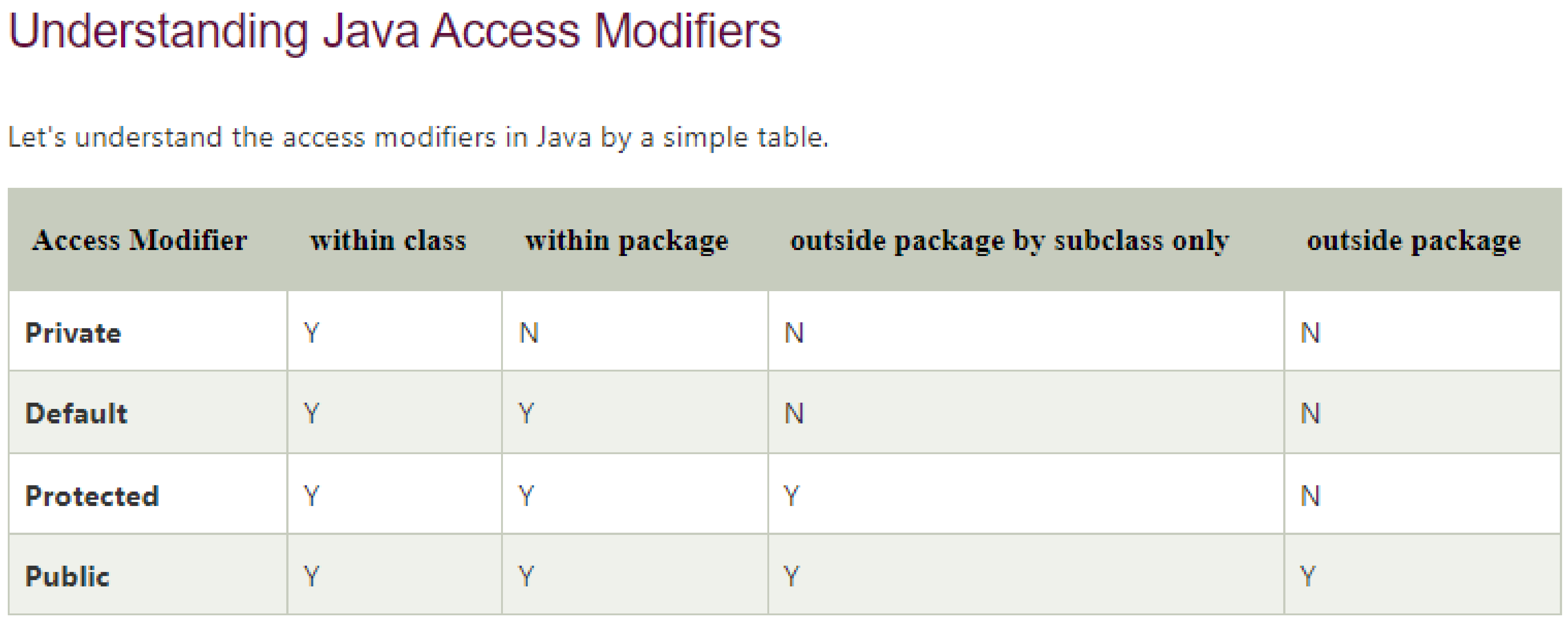
Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

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.

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.

Access Modifiers in Java

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.



Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Access Modifiers in Java

Private: The private access modifier is accessible only within the class.

```
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
            }
}
```

Access Modifiers in Java

2) Default

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

```
//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
          }
}
```

Access Modifiers in Java

3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

```
//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

Access Modifiers in Java

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
//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
```

Passing Arguments

```
class V
{
        int a,b;  //States or Fields or Instance variables or data members
        void print() //Behaviour or member functions
        {
                System.out.println("The value of a is "+a+" b is "+b);
        }

}
class K
{
        double c,d;
        //By Intilizing through method
        public void set(double a,double b) {
                c=a;
                d=b;
        }
        void print() //Behaviour or member functions
        {
                System.out.println("The value of c is "+c+" d is "+d);
        }

}
```

```
public class ObjectVal {

        public static void main(String[] args) {


                V t2=new V();
                V t3=new V();
                t2.a=20;
                t2.b=100;
                t2.print();

                 V t1=new V();  // Creating an object
                  t1.a=100;
                  t1.b=200;  //By reference variable
                 t1.print();

                K a1=new K();
                a1.set(100.78,    200.89);
                a1.print();
        }

}
```

Constructors

Constructors are special member functions of the class. It is called when an **instance** of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

```
    Class   Rectangle
   {
        public Rectangle()  //  Constructors
        {


        }
  }
```

Rules:-

1. Constructor has the same name as class name.
2. A Constructor must have no explicit return type.
3. Constructor are called when we create the object of the same class.
4. A Java constructor cannot be abstract, static, final, and synchronized

It is a special type of method which is used to initialize the object.

Every time an object is created using **the new()** keyword, at least one constructor is called.

It calls a **default constructor** if there is no constructor available in the class. In such case, Java compiler provides a default constructor by **default**.

There are two types of constructors in Java: no-arg constructor, and **parameterized** constructor.

Note: It is called constructor because it constructs the values at the time of object creation.

It is not necessary to write a constructor for a class.

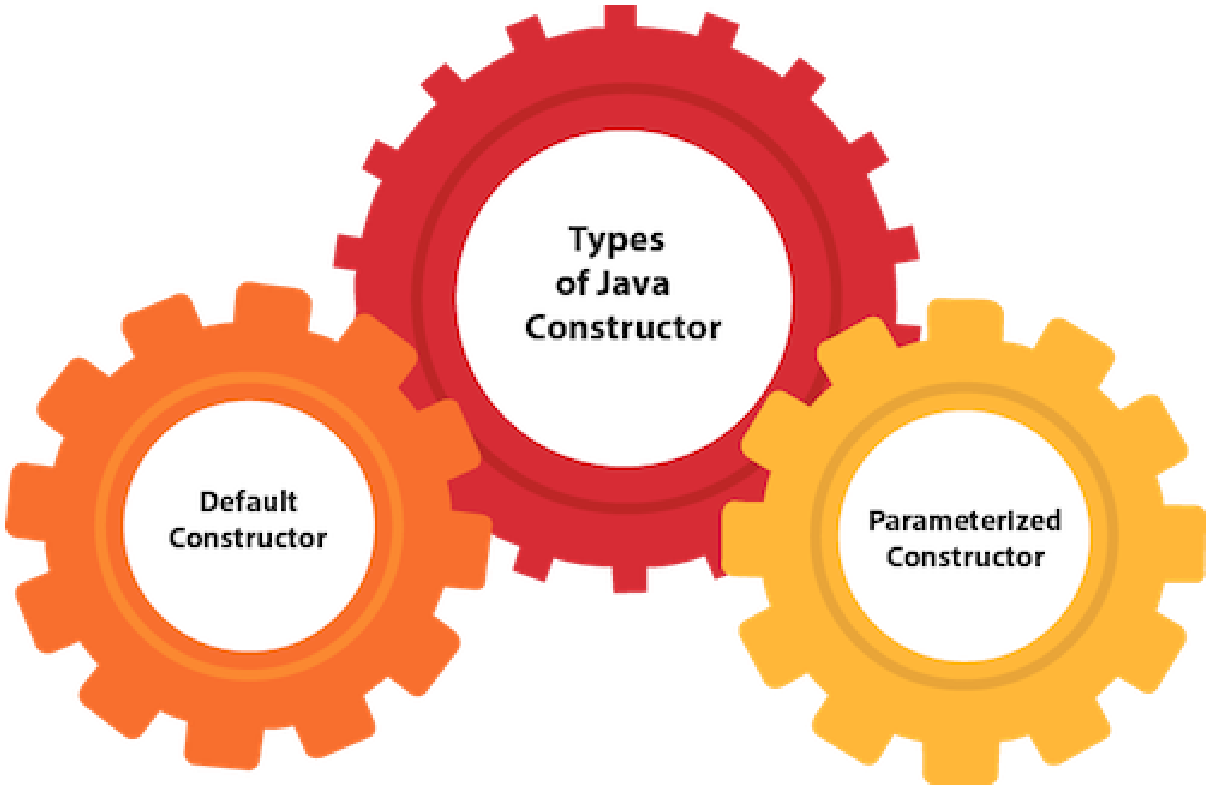
It is because java compiler creates a default constructor if your class doesn't have any.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)

2. Parameterized constructor



What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example to demonstrate the default values of constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
Class   class_name()
{


}
```

Example of Default Constructor

```
class Bike1
{
    //creating a default constructor
    Bike1()
    {
        System.out.println("Bike is created");
    }
    //main method
    public static void main(String args[])
    {
        //calling a default constructor

        Bike1 obj2 = new Bike1();
    }
}
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

=> Program to demonstrate the use of the parameterized constructor.

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

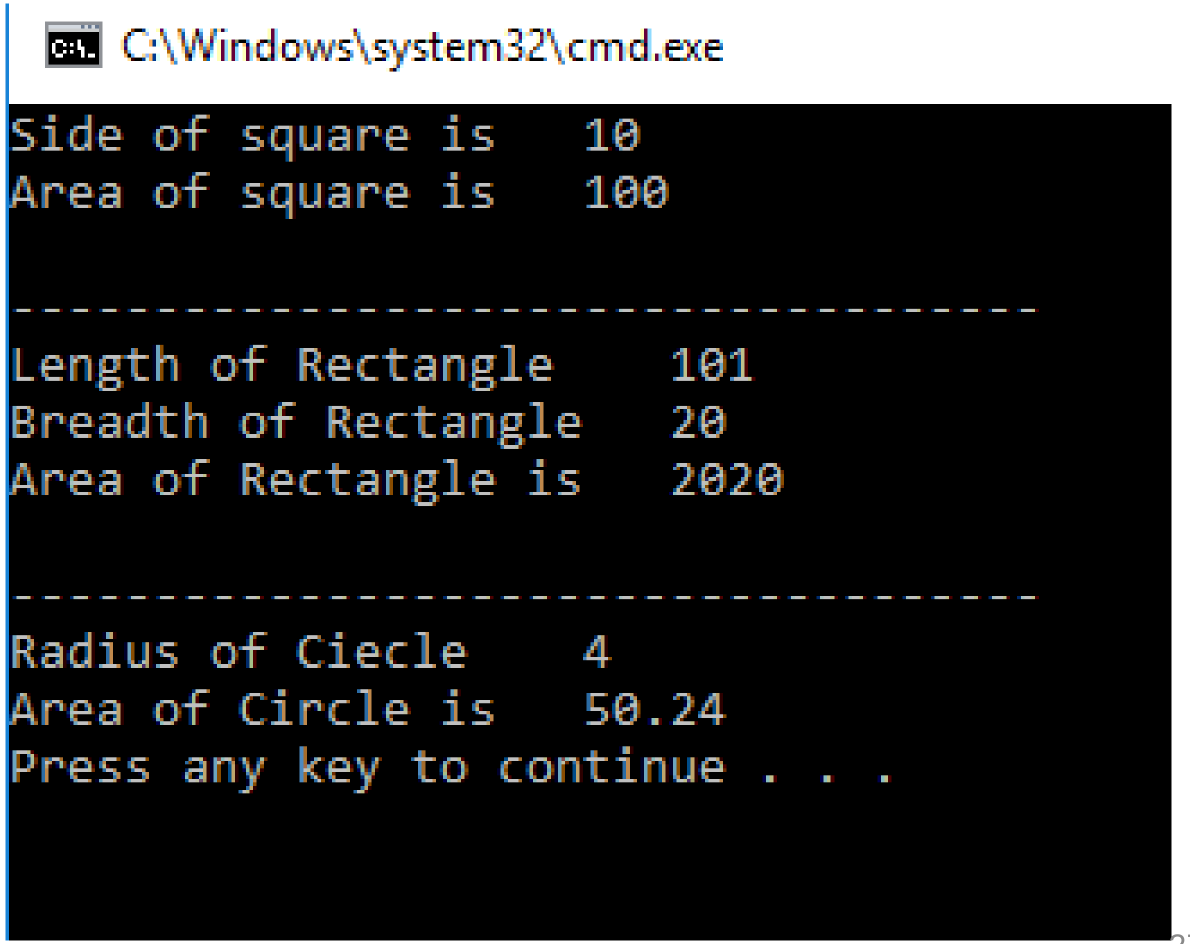
Constructor overloading in Java is a technique of having more than one constructor with **different parameter** lists.

They are arranged in a way that each constructor performs a different task.

They are differentiated by the compiler by the number of parameters in the list and their types.

```
class    Area
{
    // Area of Square = side*side
                   public   Area(int a)
                   {
                        int area= a*a;
                        System.out.println("Side of square is    "+a);
                        System.out.println("Area of square is    "+area);
                   }
                   // Area of Rectangle
                   public  Area(int l,int b)
                   {
                        System.out.println("\n-----------------------------------");
                       int area = l*b;
                        System.out.println("Length of Rectangle     "+l);
                        System.out.println("Breadth of Rectangle    "+b);
                           System.out.println("Area of Rectangle is    "+area);

                   }
                   // Area of Circle
                   public  Area(double pi, int r)
                   {
                                          System.out.println("\n-----------------------------------");
                                                         double area = pi*r*r;
                                                          System.out.println("Radius of Ciecle     "+r);

                              System.out.println("Area of Circle is    "+area);
                   }
}
```

```
class   ConstrcutorOverloading
{
                public static void main(String args[])
                {
                        Area obj1 = new Area(10);
                        Area obj2 = new Area(101,20);
                        Area obj3 = new Area(3.14, 4);

                }
}
```



C:\Windows\system32\cmd.exe

```
Side of square is   10
Area of square is   100


------------------------------------
Length of Rectangle    101
Breadth of Rectangle   20
Area of Rectangle is   2020


------------------------------------
Radius of Ciecle    4
Area of Circle is   50.24
Press any key to continue . . .
```

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4
{
    int id;      // instance variable
    String name;    // instance variable
                                         //creating a parameterized constructor
    Student4(int id,String name)  // Here, I and n are local variables
    {
        this.id = id;
        this.name = name;
    }

    //method to display the values
    void display()     //  Member function
     {
         System.out.println(id+" "+name);
      }
```

//Java Program to demonstrate the use of the parameterized constructor.

```
    public static void main(String args[])
    {
        //creating objects and passing values
        Student4 s1 = new Student4(111,"Karan");

        Student4 s2 = new Student4(222,"Aryan");
        //calling method to display the values of object

        s1.display();

        s2.display();
   }
}
```

The **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

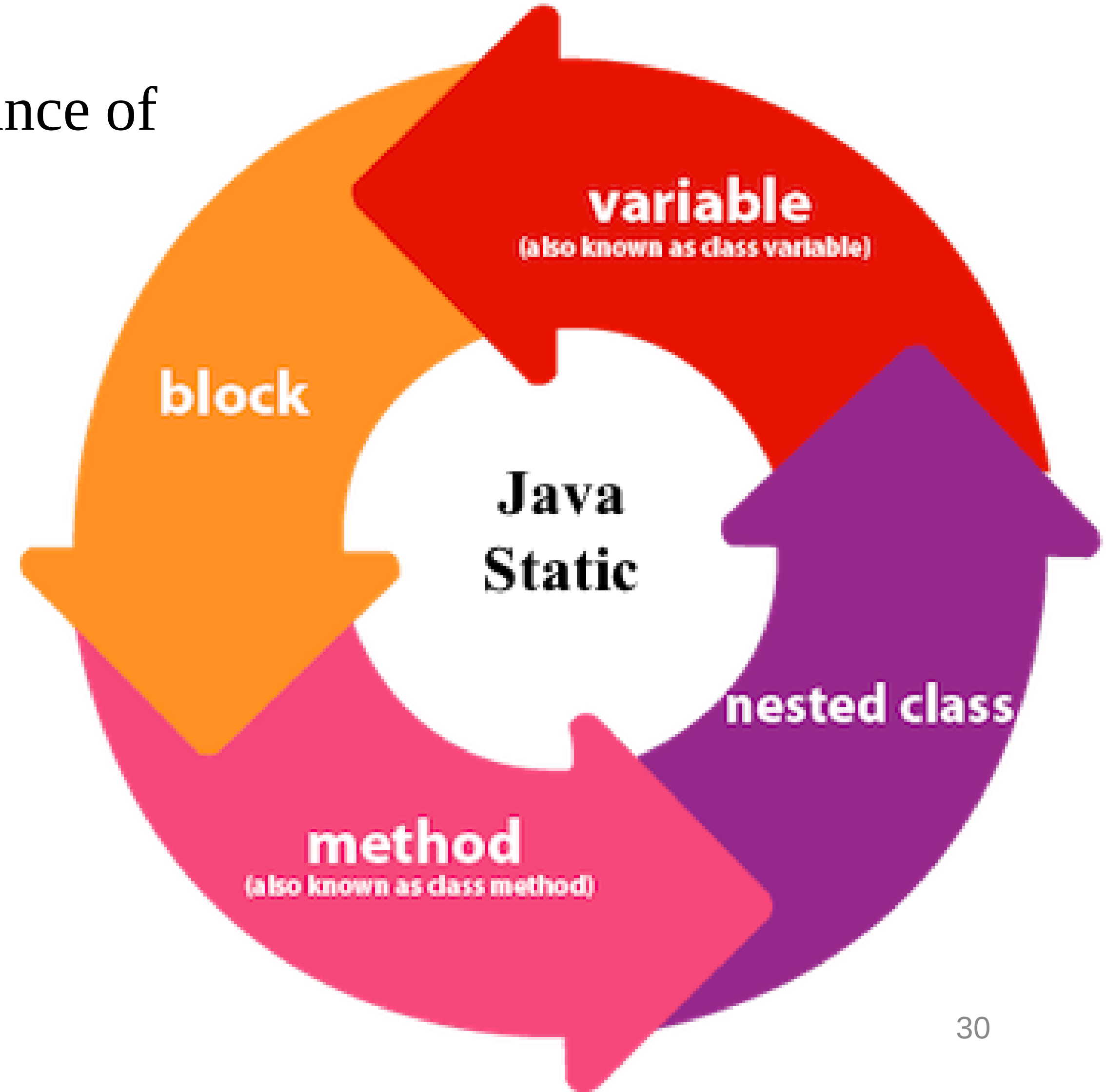
the class.
The static can be:

❖ 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**), for example, 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.

Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

Understanding the problem without static variable

```
class Student{
     int rollno;
     String name;
     String college="ITS";
}
```

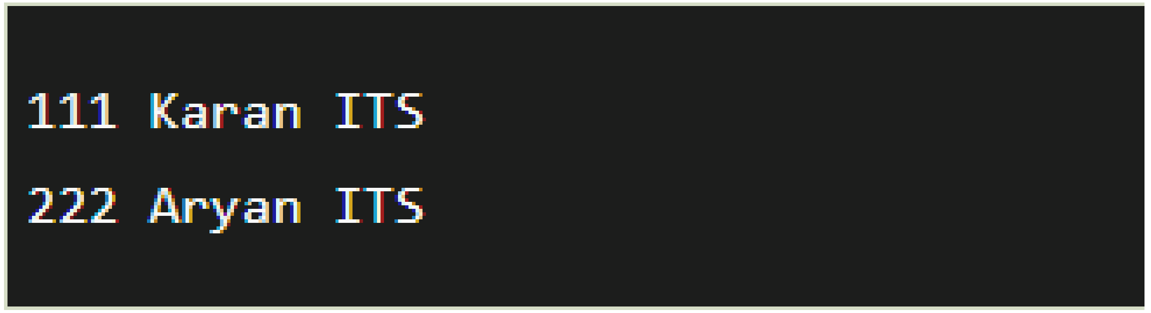
Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Java static property is shared to all objects.

```
//Java Program to demonstrate the use of static variable
class Student
{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
   //constructor
   Student(int r, String n) // Local variable
   {
         rollno = r;
         name = n;
   }
   //method to display the values
   void display ()
   {
               System.out.println(rollno+" "+name+" "+college);
    }
}
```

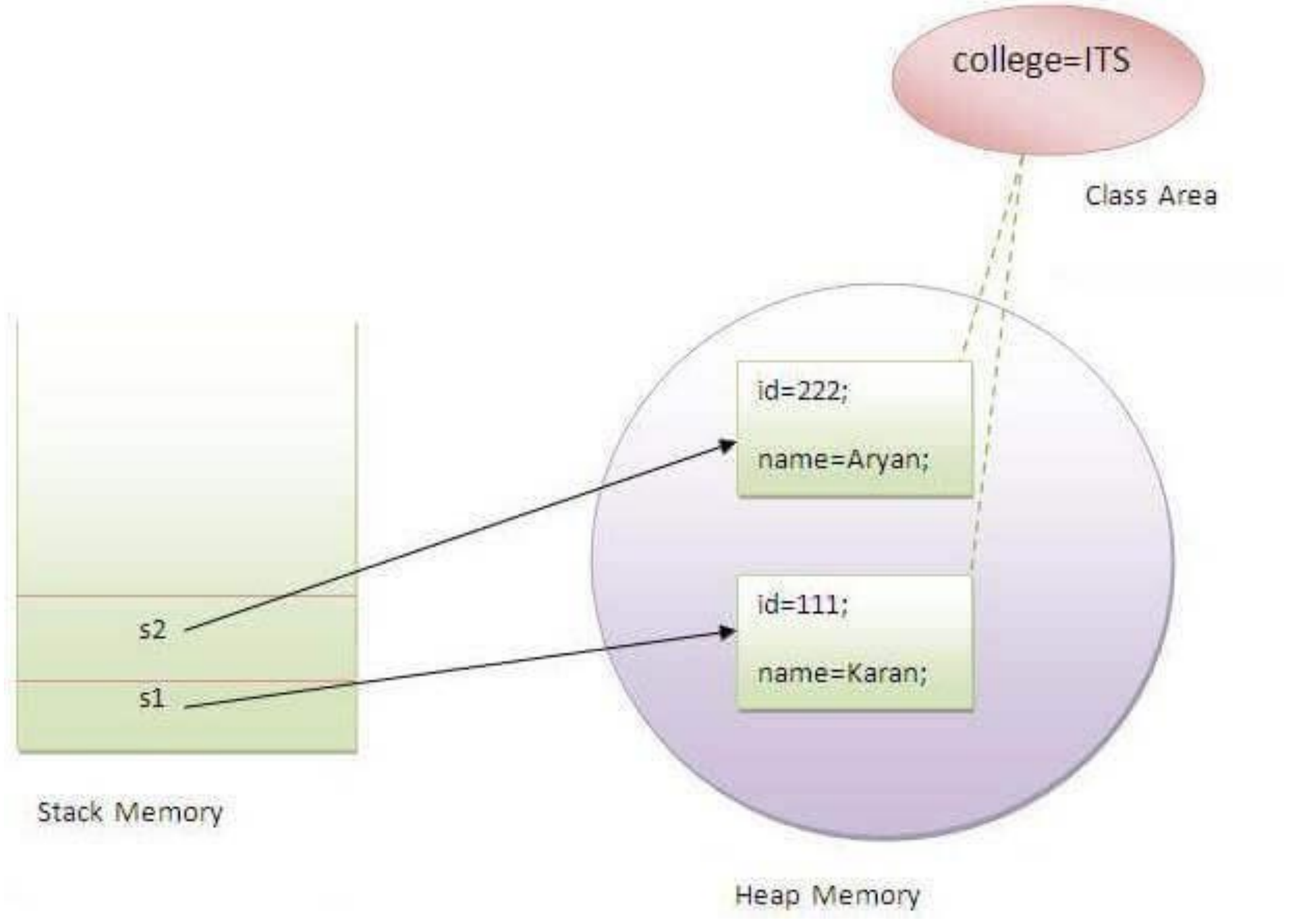
```
//Java Program to demonstrate the use of static variable
public class TestStaticVariable1
{
      public static void main(String args[])
      {
            Student s1 = new Student(111,"Karan");
            Student s2 = new Student(222,"Aryan");

            //we can change the college of all objects by the single line of code

           //  Student.college="BBDIT";

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

Output:



```
111 Karan ITS
222 Aryan ITS
```

Static class members, Garbaze collection



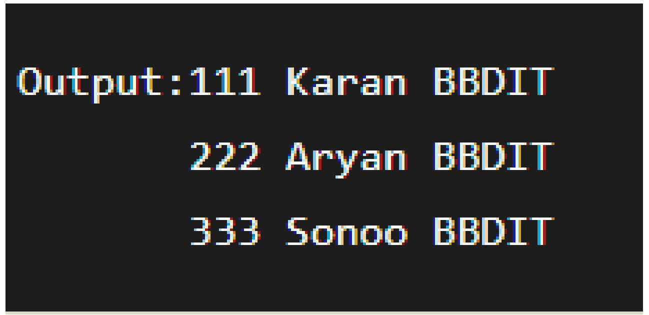
2) Java static method

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 = "ITS";
     //static method to change the value of static variable
     public static void change()
     {
            college = "BBDIT";
     }
     //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();
    }
}
```



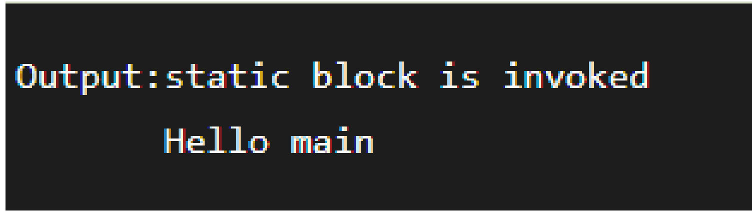
```
Output:111 Karan BBDIT
       222 Aryan BBDIT
       333 Sonoo BBDIT
```

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.

Example of static block

```
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
```

Garbage Collection

n java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using **free()** function in C language and delete() in C++.

But, in **java** it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Garbage Collection

n java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using **free()** function in C language and delete() in C++.

But, in **java** it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
protected void finalize(){}
```

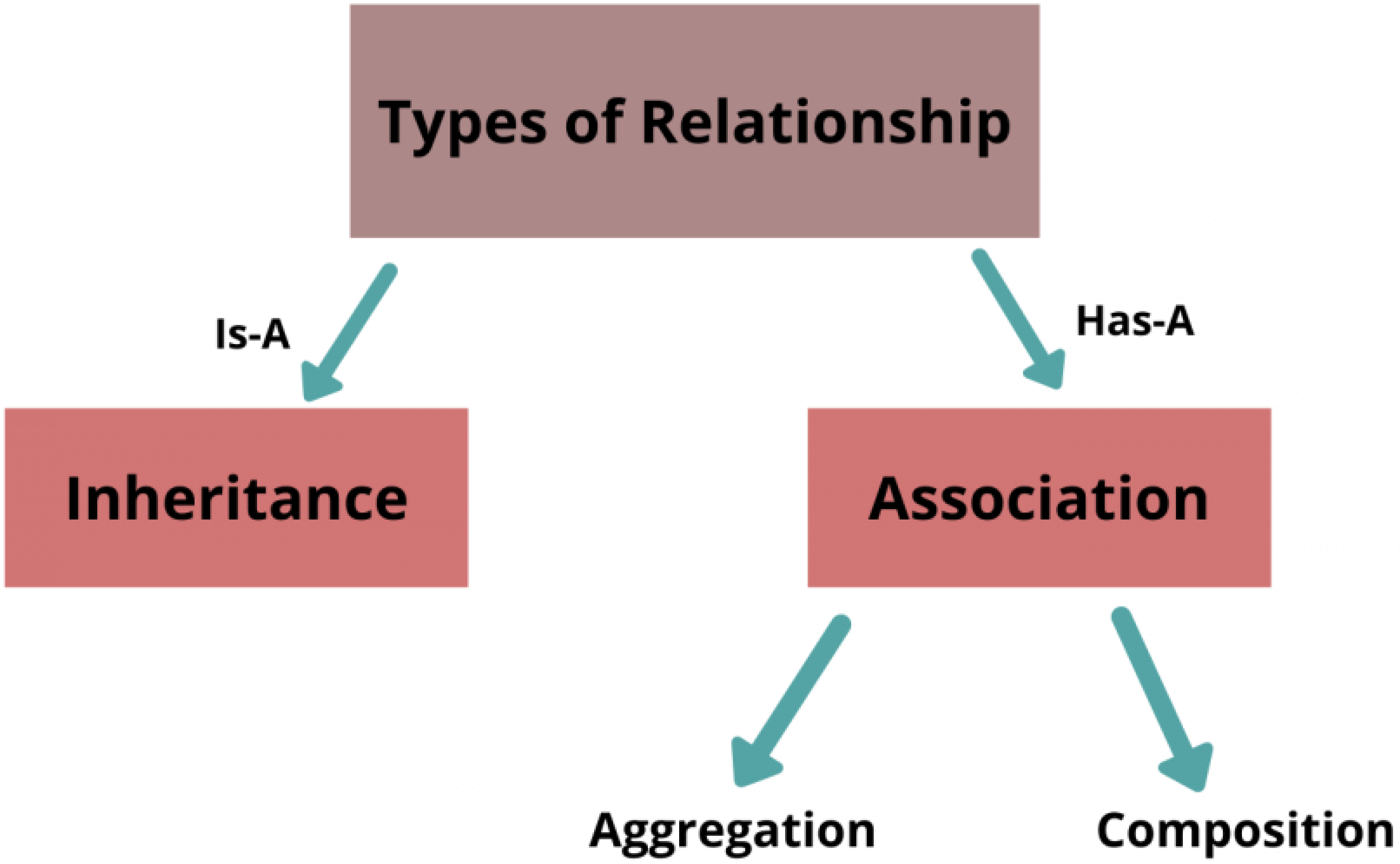
Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

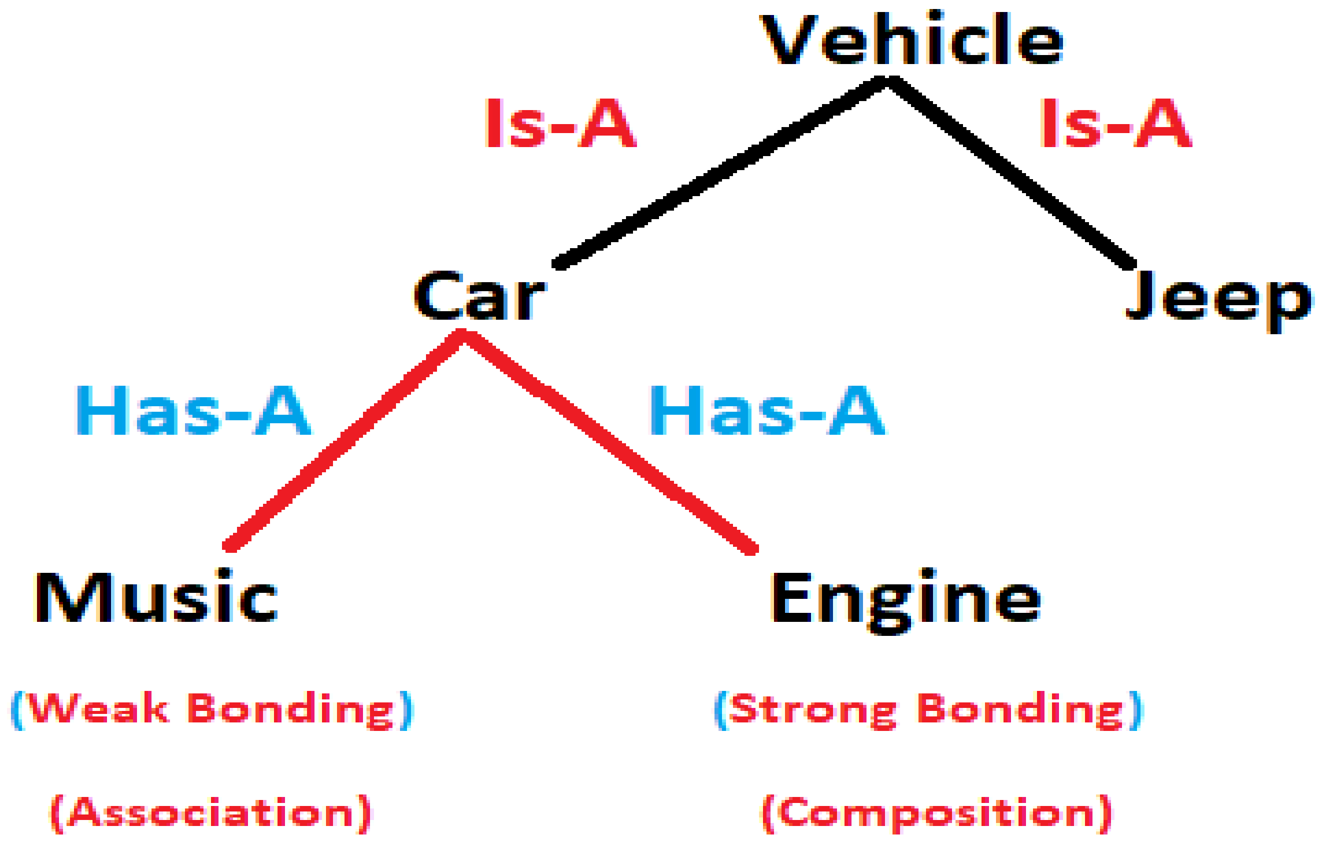
gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
```

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.


Types of Relationship
Is-A
Has-A
Inheritance
Association
Aggregation
Composition


Vehicle
Is-A
Is-A
Car
Jeep
Has-A
Has-A
Music
Engine
(Weak Bonding)
(Association)
(Strong Bonding)
(Composition)

Inheritance:

- ❖ Basics of inheritance,
- ❖ Inheriting and Overriding Superclass methods,
- ❖ Calling Superclass Constructor,
- ❖ Polymorphism,
- ❖ Abstract Classes,
- ❖ Final Class.

Inheritance in Java

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

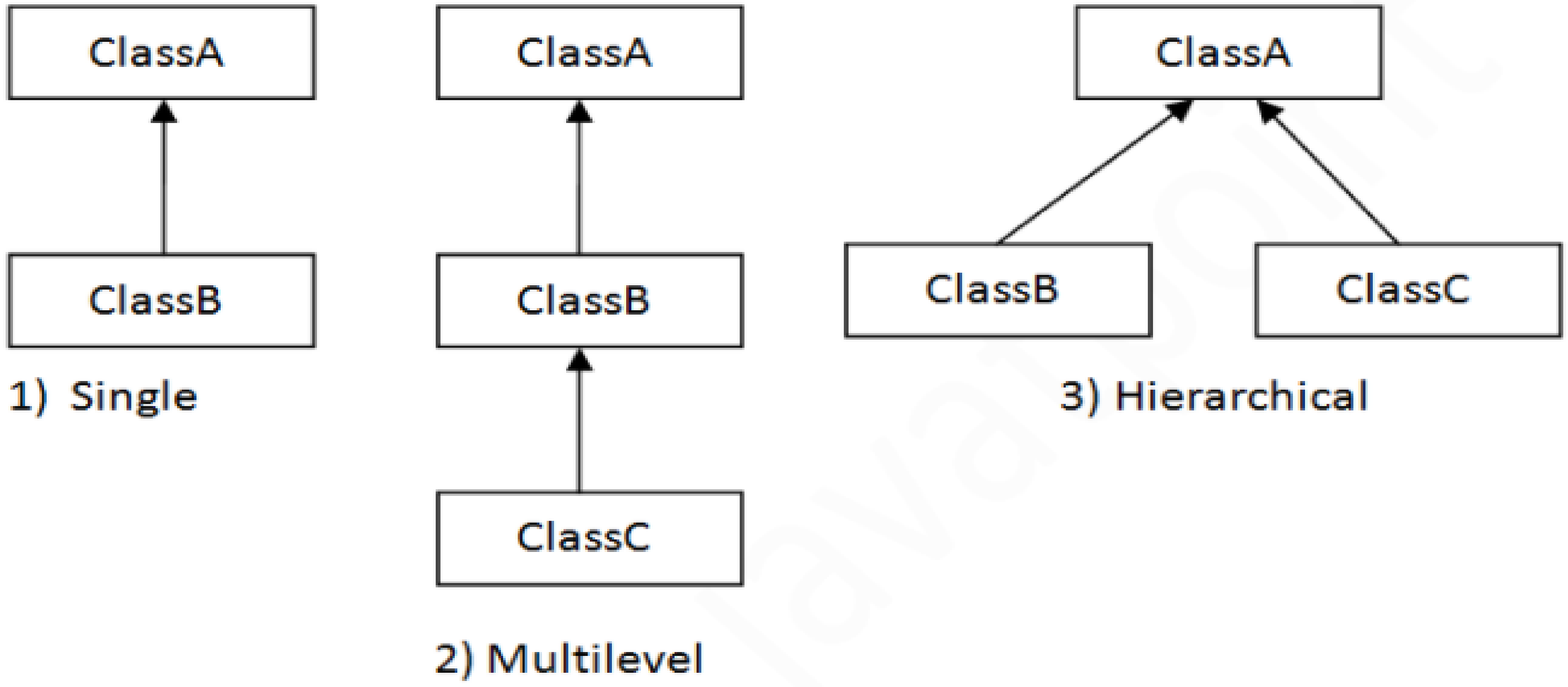
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

- When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

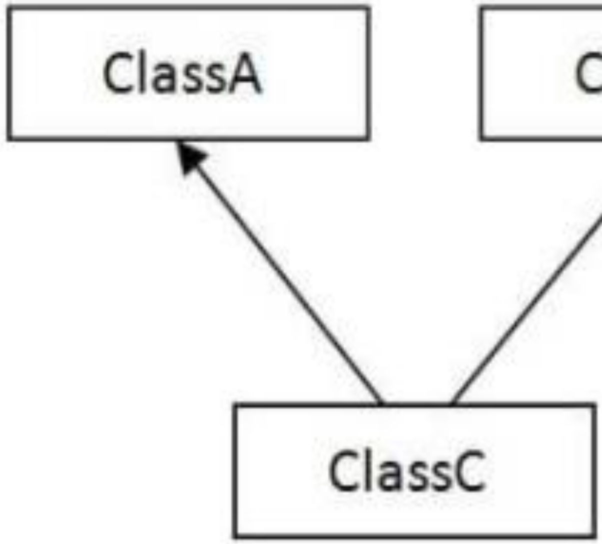
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

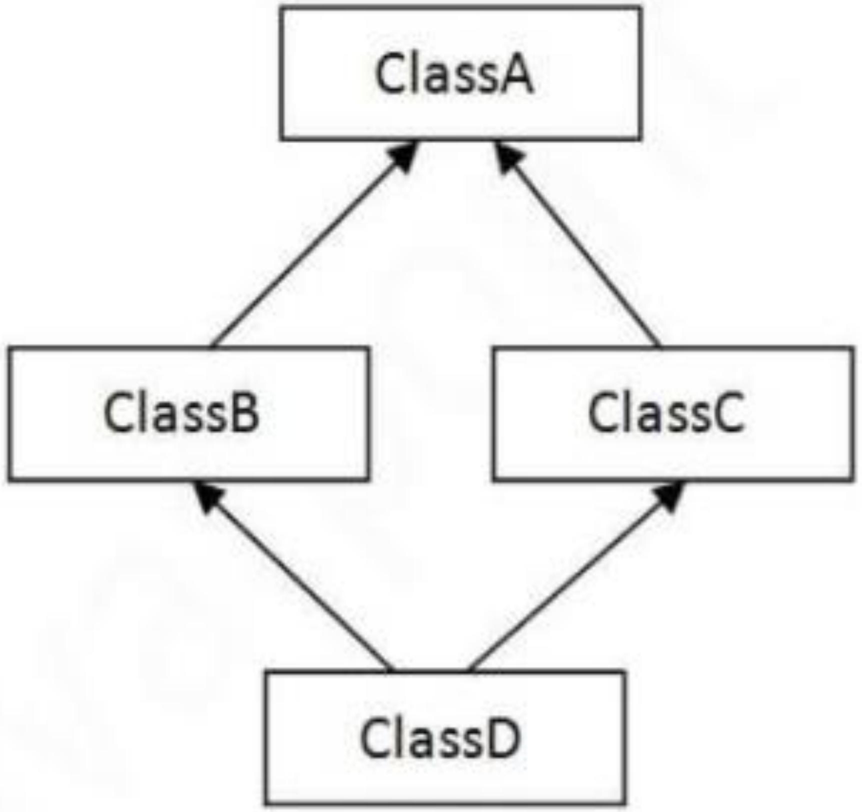
In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



When one class inherits multiple classes, it is known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Sem2 class inherits the Sem1 class features, so there is the single inheritance.

Single Inheritance

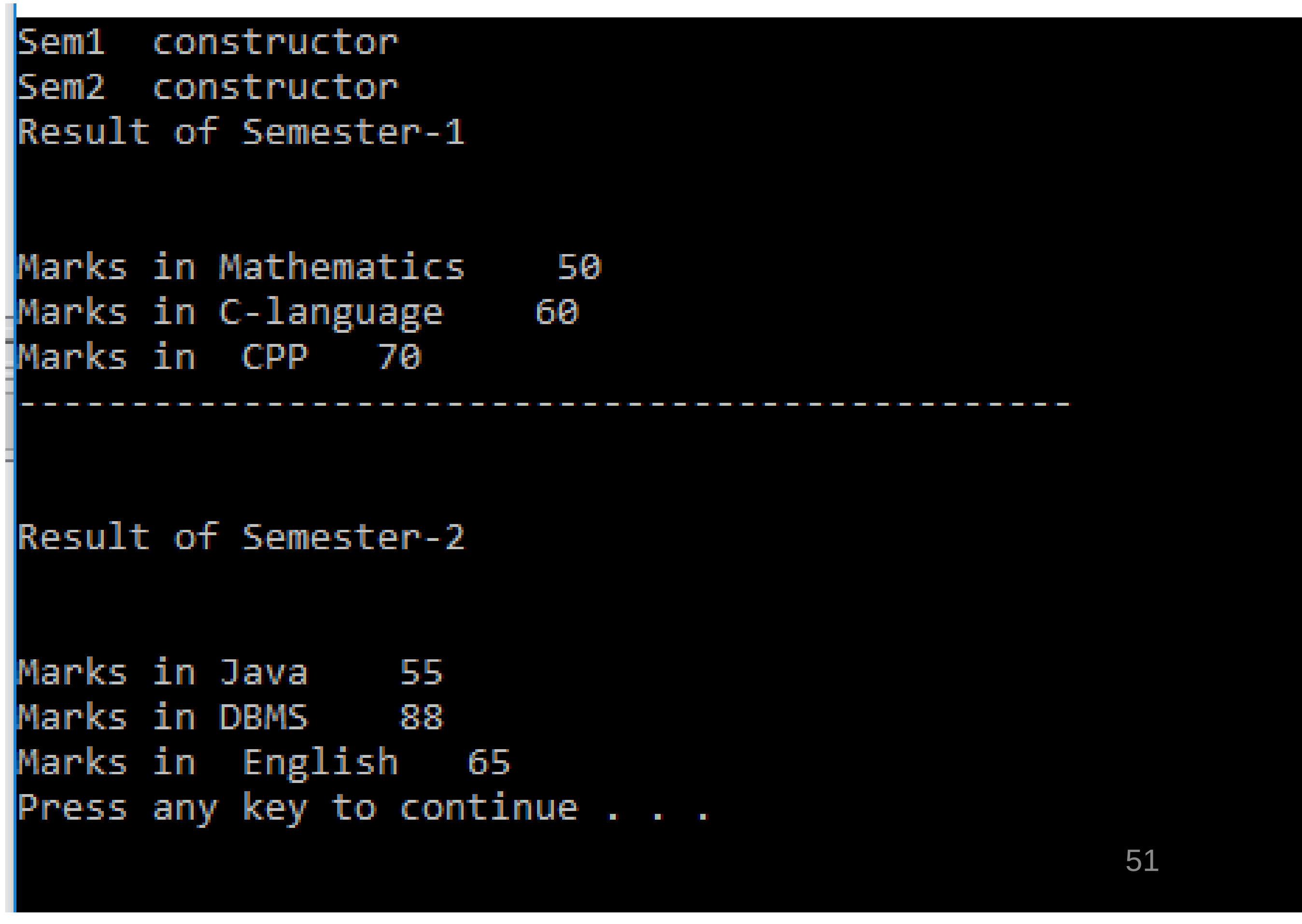
```
class    Sem1
{
          public Sem1()
          {
                    System.out.println("Sem1  constructor     ");
          }
          int math, c, cpp;   // instance variable
           void input()   // Member function
          {
               math=50;
               c=60;
               cpp=70;
          }
          public void output()  // Member function
          {
                  System.out.println("Result of Semester-1  \n\n");
                    System.out.println("Marks in Mathematics    "+math);
                    System.out.println("Marks in C-language    "+c);
                    System.out.println("Marks in  CPP   "+cpp);

                    System.out.println("---------------------------------------------  \n\n");

          }
}
```

Single Inheritance

```
class   Sem2  extends Sem1
{
            int java, dbms, eng; // instance variable

            public Sem2()
            {
                    System.out.println("Sem2  constructor     ");
            }

            public void input2()  // Member functions
            {
                       java =55;
                       dbms=88;
                       eng=65;
            }
            public void output2()
            {
                      System.out.println("Result of Semester-2  \n\n");
                      System.out.println("Marks in Java    "+java);
                      System.out.println("Marks in DBMS    "+dbms);
                      System.out.println("Marks in  English   "+eng);

            }
}
```

Single Inheritance

```
class   SingleInheritance
{
        public static void main(String args[])
        {
                Sem2 obj = new Sem2();

                obj.input();
                obj.input2();
                obj.output();
                obj.output2();


        }
}
```

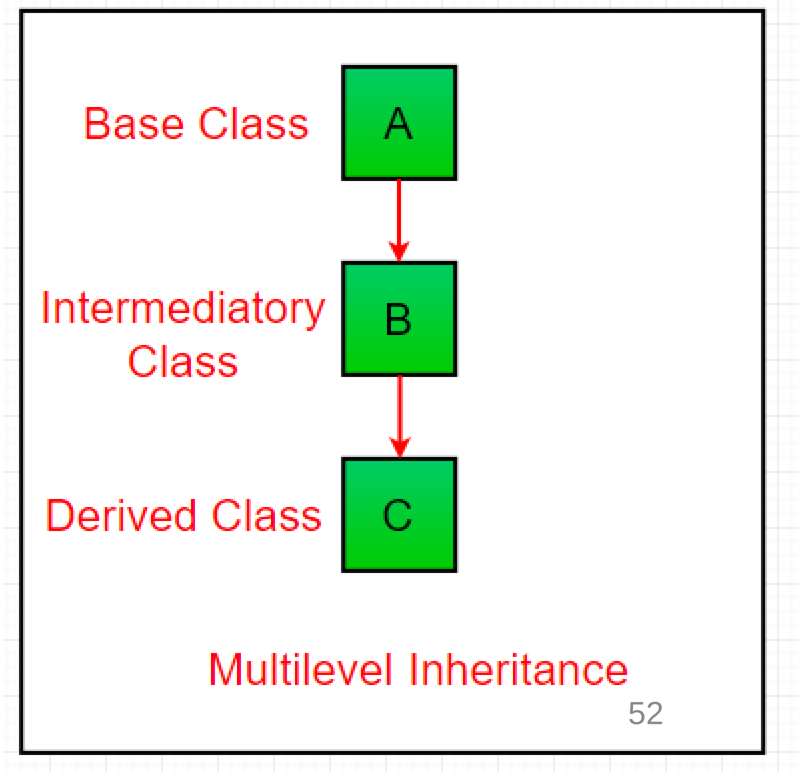


```
Sem1  constructor
Sem2  constructor
Result of Semester-1


Marks in Mathematics    50
Marks in C-language    60
Marks in  CPP   70
-----------------------------------------------



Result of Semester-2


Marks in Java    55
Marks in DBMS    88
Marks in  English   65
Press any key to continue . . .
```

Multilevel Inheritance

When a class extends a class, which extends anther class then this is called **multilevel inheritance**.

For example class C extends class B and class B extends class A then this **type of inheritance** is known as multilevel inheritance



Multilevel Inheritance

```
class    Sem1
{
             public Sem1()
             {
                             System.out.println("Sem1  constructor     ");
             }
             int math, c, cpp;   // instance variable
              void input()   // Member function
             {
                  math=50;
                  c=60;
                  cpp=70;
             }
             public void output()  // Member function
             {
                          System.out.println("Result of Semester-1  \n\n");
                             System.out.println("Marks in Mathematics    "+math);
                             System.out.println("Marks in C-language    "+c);
                             System.out.println("Marks in  CPP   "+cpp);

                             System.out.println("--------------------------------------------  \n\n");

             }
}
```

Multilevel Inheritance

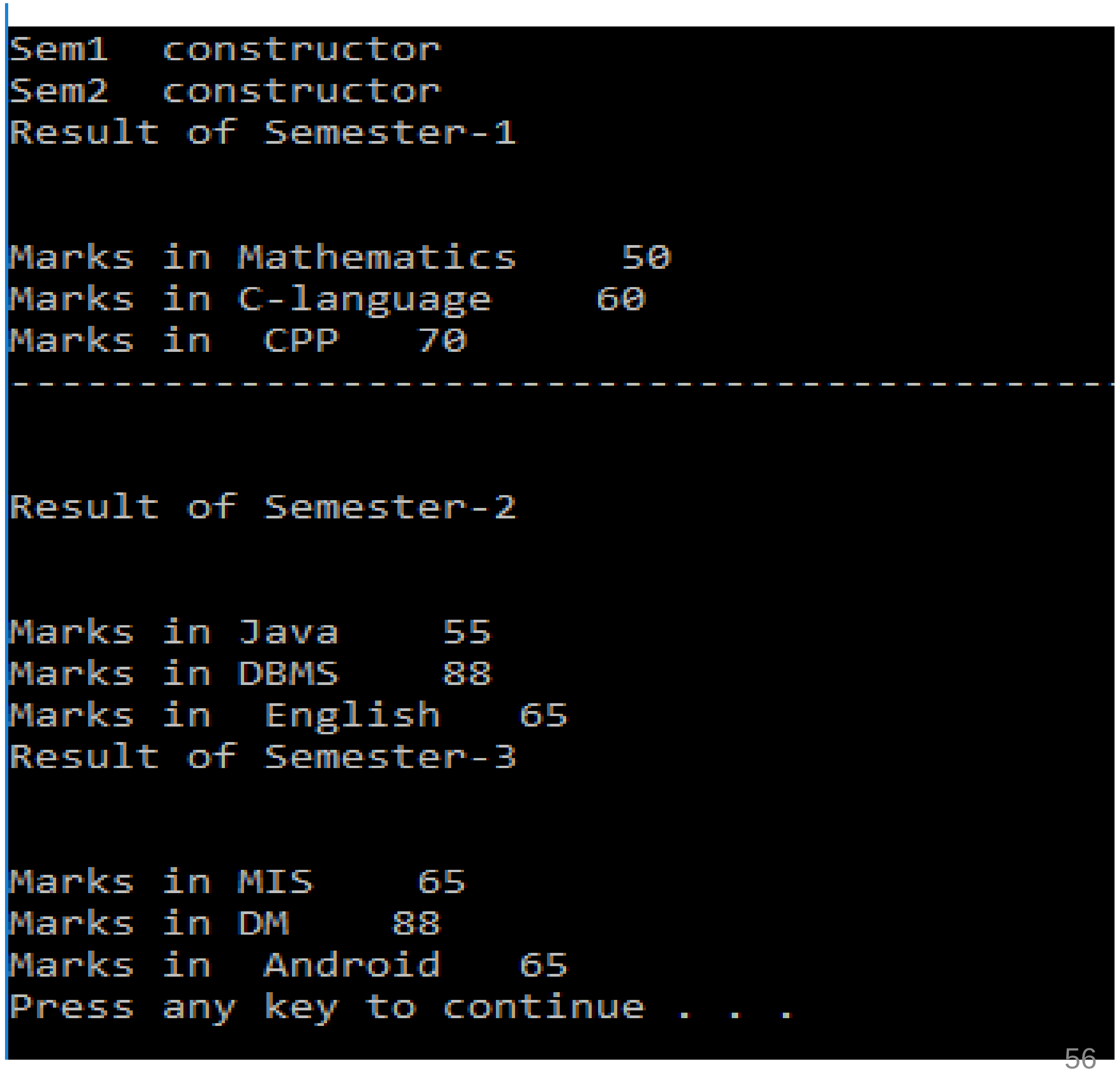
```
class   Sem2  extends Sem1
{
            int java, dbms, eng; // instance variable

            public Sem2()
            {
                    System.out.println("Sem2  constructor     ");
            }

            public void input2()  // Member functions
            {
                       java =55;
                       dbms=88;
                       eng=65;
            }
            public void output2()
            {
                      System.out.println("Result of Semester-2  \n\n");
                      System.out.println("Marks in Java    "+java);
                      System.out.println("Marks in DBMS    "+dbms);
                      System.out.println("Marks in  English   "+eng);

            }
}
```

Multilevel Inheritance

```
class Sem3 extends Sem2
{
            int mis, dm, android;
            public void input3()  // Member functions
            {
                                    mis =65;
                                    dm=88;
                                    android=65;
             }
                    public void output3()
                     {
                                   System.out.println("Result of Semester-3  \n\n");
                                   System.out.println("Marks in MIS    "+mis);
                                   System.out.println("Marks in DM    "+dm);
                                   System.out.println("Marks in  Android   "+android);

            }


}
```

Multilevel Inheritance

```
class   MultievelInheritance
{
            public static void main(String args[])
            {
                    Sem3 obj = new Sem3();

                    obj.input();
                    obj.input2();
                    obj.input3();
                    obj.output();
                    obj.output2();
                    obj.output3();


            }
}
```



```
Sem1  constructor
Sem2  constructor
Result of Semester-1


Marks in Mathematics    50
Marks in C-language    60
Marks in  CPP   70
-------------------------------------------



Result of Semester-2


Marks in Java    55
Marks in DBMS    88
Marks in  English   65
Result of Semester-3


Marks in MIS    65
Marks in DM    88
Marks in  Android   65
Press any key to continue . . .
```

Multiple Inheritance

Interfaces in java

- It is an alternative for multiple inheritance.
- Conceptually like abstract classes.
- More than one interfaces can be implemented without the problem of multiple inheritance.
- Similar to class, having variable declaration and methods but all the methods are abstract and variables are constants (final).

Interface

- Interface is used when some task is to be performed but exact definition of that task is not predefined.
- Interface is that user defined data type, which contains abstract method and final constant. It is declared with the keyword interface followed by its name.
- Methods can’t be defined directly in the interface.
- By default all the methods declared in interface are public & abstract.
- class is derived from the interface with the keyword implements.
- Therefor single inheritance & Multilevel inheritance can be created can be implemented using interface.

Interfaces in java (Cont.)

- Defining Interfaces

Syntax:

```
interface interface_name
{
        //variable declaration;
        //Methods() Declaration;
}
```

- Methods having no implementation inside the interface

Interface

- Format for interface is as follow:-

```
interface    interface_Name
{
                  variable declaration;

                  method declarartion;
}
```

Here interface is the keyword and interface name is any valid java variable

- Example:-

```
interface    abc
{
                  final int a=40;

                  abstract public void input();
}
```

Interfaces in java (Cont.)

- Extending Interfaces
  - An interface can be sub interfaced, inherits all the properties of super interface.
  - Eg:

```
interface one
{
        //variable declaration;
        //Methods() Declaration;
}
interface two extends one
{
        //variable declaration;
        //Methods() Declaration;
}
```

Interfaces in java (Cont.)

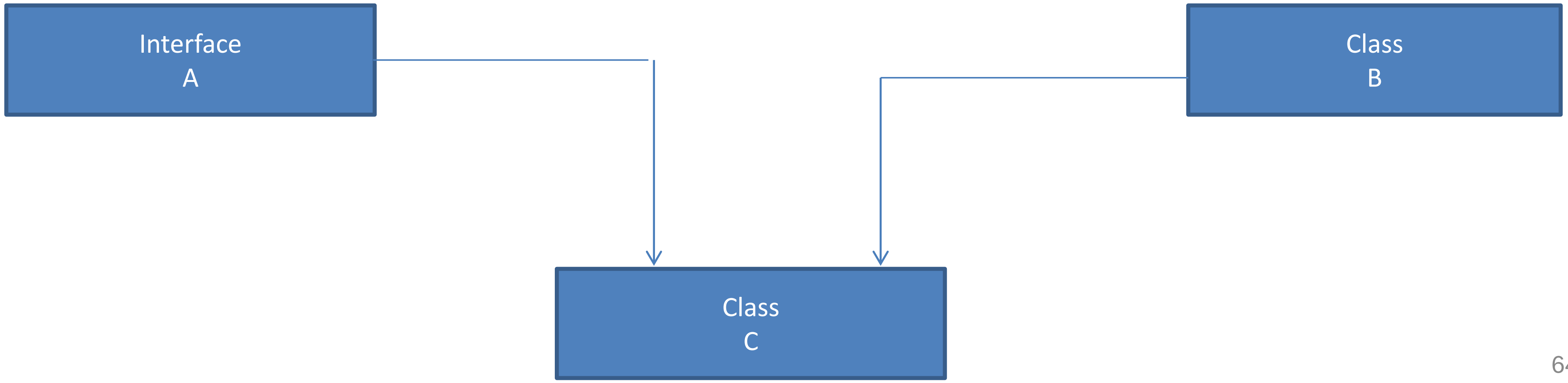
- Implementing Interfaces
  - Class can implement an interface
  - A class can also extends another class while implementing interface.
  - This can be done as follows:

```
Class one extends superclass implements interface1, interface2,...
{
        //Body of the class
        // implement all methods of interfaces.
}
```

  - A class which will not implement all the methods of interface will become abstract class

Multiple Inheritance

- Multiple inheritance is not available in java. But it can be constructed by using both interface and classes.
- Suppose there is an interface A and there is independent class B.
- There is one more independent class C derived from interface A and class B.
- The derived class C will inherit all the features of interface as well as all the features of class B.
- In interface data members should be declare as public along with the methods and data members of the class should be also public.



```
interface    Test
{
                 final int d=22;
                 abstract public int area();
                  abstract public  int volume();
}
Interface Test2
{
     final int d=25;
      public   int area2();

}
Class     B
{
      public void input()
      {
                  System.out.println("This is an input method  ");

      }
      public void output()
      {
                  System.out.println("This is an outputmethod  ");


      }


}
```

```
class           CProg        extends  B  implements Test,Test2
{
                int i,j;
                 public int area()
                {
                            i=10;
                            j=20;
                                return i*j;
                }
                public int volume()
                {
                                return (i*j*22) ;
                }
               public int area2()
              {
                   System.out.println("Area 2 is a method of Test2 interface   ");
                   return i*j;
              }
}
```

```
class         CProg        extends  B  implements Test,Test2
{
              int i,j;
               public int area()
              {
                         i=10;
                         j=20;
                            return i*j;
              }
              public int volume()
              {

                             System.out.println(“Value of d in test interface   “+Test.d);
                             System.out.println(“Value of d in test interface   “+Test2.d);
                              return (i*j*22) ;
              }
             public int area2()
            {
                 System.out.println(“Area 2 is a method of Test2 interface   “);
            }
}
```

```
class          Demo3
{
               public static void main(String args[])
               {
                     Cprog    obj = new Cprog();
                     obj.input();
                     obj.output();
                  int area=    obj.area();
                  int area2=    obj.volume();

                   System.out.println(area);
                  System.out.println(area2);
                     obj.area2();

               }
}
```

Abstract Class

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

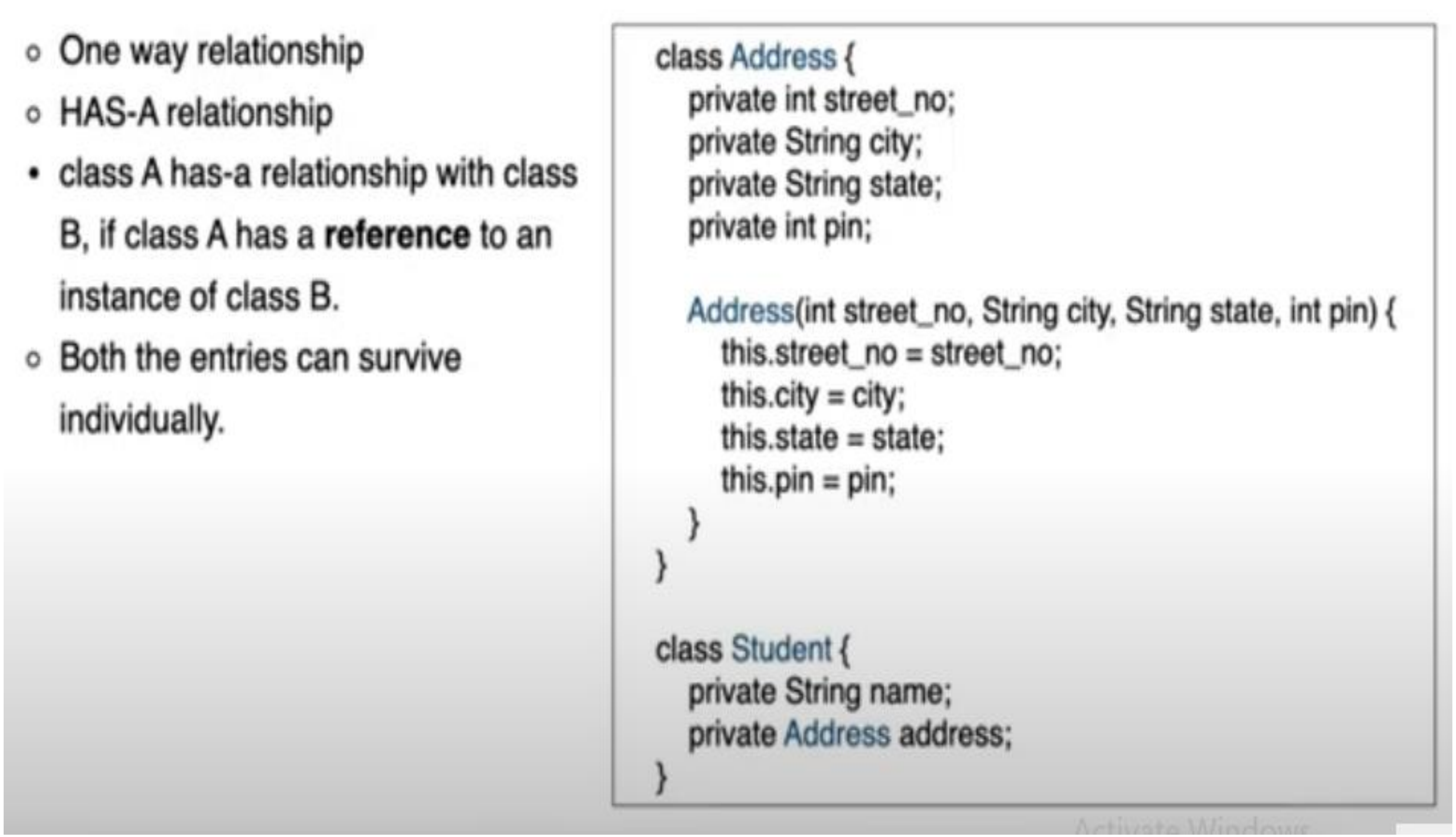
- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.(We can not create the object)
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Abstract Class

```
abstract class Bike
{
 abstract void run();
}
class Honda4 extends Bike
{
        void run()
        {
             System.out.println("running safely");
          }
          public static void main(String args[])
          {
                Honda4 obj = new Honda4();
                 obj.run();
           }
}
```

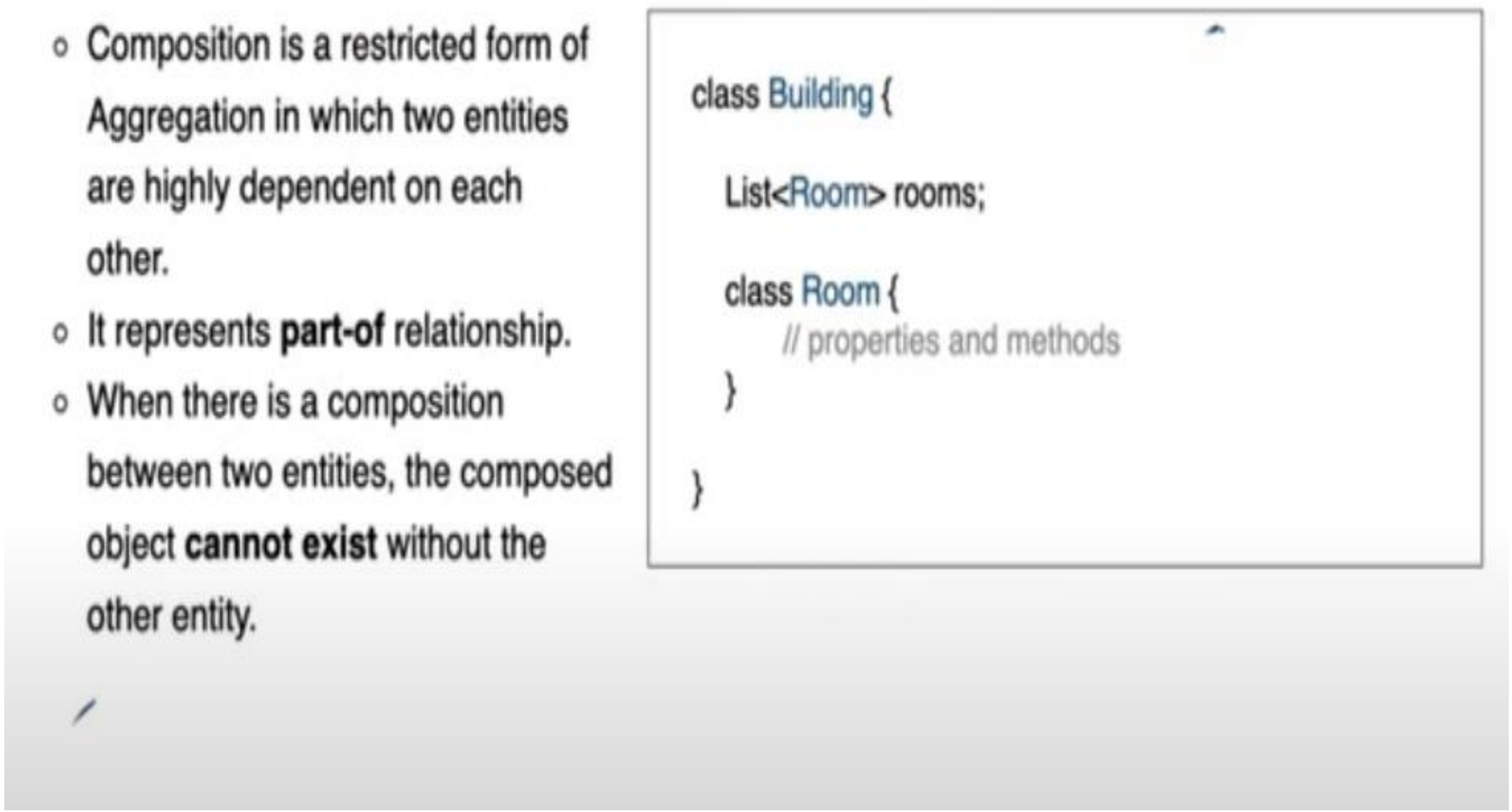
Difference between abstract class & interface

❖ Abstract class contains abstract and non abstract methods.	❖ Interface is to be created when all the method have abstract.
❖ Abstract class to be inherited using extends keyword.	❖ Interface is to be implemented using implements keyword.
❖ Contains variable field.	❖ Contains only final field.



- One way relationship
- HAS-A relationship
- class A has-a relationship with class B, if class A has a **reference** to an instance of class B.
- Both the entries can survive individually.

```
class Address {
    private int street_no;
    private String city;
    private String state;
    private int pin;

    Address(int street_no, String city, String state, int pin) {
        this.street_no = street_no;
        this.city = city;
        this.state = state;
        this.pin = pin;
    }
}

class Student {
    private String name;
    private Address address;
}
```



- Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.
- It represents **part-of** relationship.
- When there is a composition between two entities, the composed object **cannot exist** without the other entity.

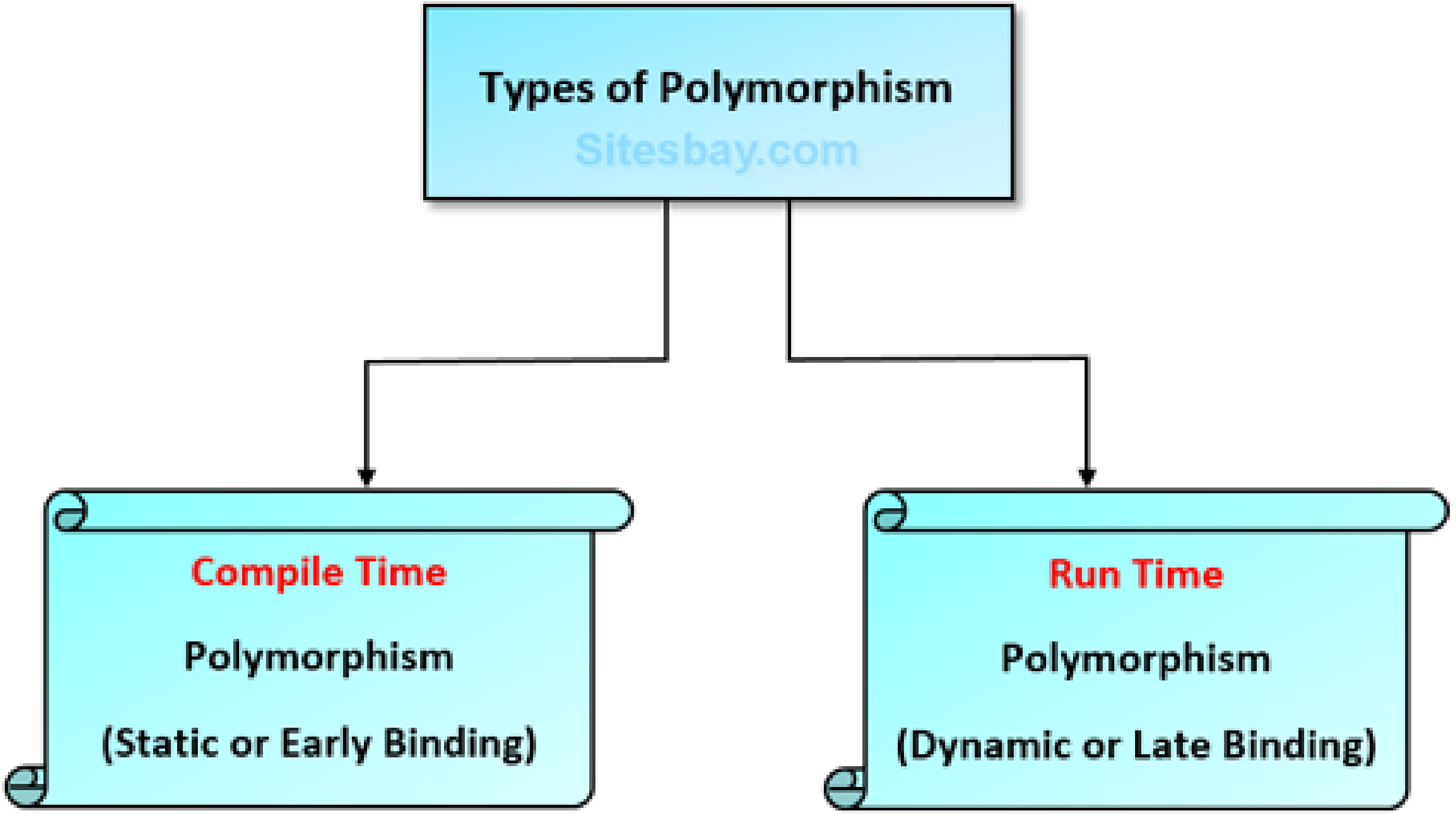
```
class Building {

   List<Room> rooms;

   class Room {
         // properties and methods
   }

}
```

Polymorphism

The word polymorphism means having **many forms**. In simple words, we can define polymorphism as the ability of a message to be displayed in **more than one form**.

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways.

In other words, polymorphism allows you to define one interface and have multiple implementations. The word "**poly**" means **many** and "**morphs**" means **forms**, So it means **many forms**.



Method Overloading

Method Overriding

Method Overloading: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**.

Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

Example: By using different types of arguments

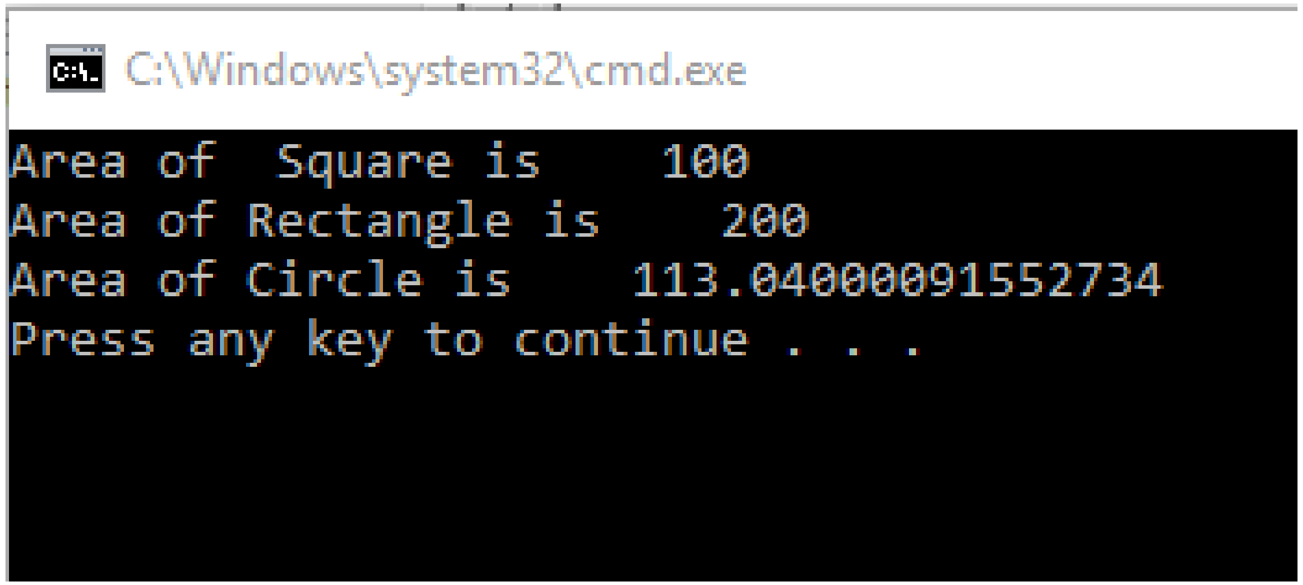
Advantage of method overloading

Method overloading *increases the readability of the program*.

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

```
// WAP to implement Method Overloading
Class     Shape
{



                public void  area(int  side)   // Calculate the area of  Square
                {
                            int result  = side*side;
                            System.out.println(“Area of  Square is    “+result);

                  }

                  public void area(int length, int breadth)  //  Area of Rectangle
                {

                            int result  = length*breadth;
                            System.out.println(“Area of Rectangle is    “+result);

                }
                  public void area(double pi, int r)
                {
                                double result = pi*r*r;
                                System.out.println(“Area of Circle is    “+result);



                  }
public static void main(String args[])
    {
            Shape obj = new Shape();

            obj.area(10);
            obj.area(10,20);
            obj.area(3.14, 6);

            // There are two types of literal
            //    integer    44          4 bytes
            //    Double     6.3         8 bytes



     }
}
```



C:\Windows\system32\cmd.exe

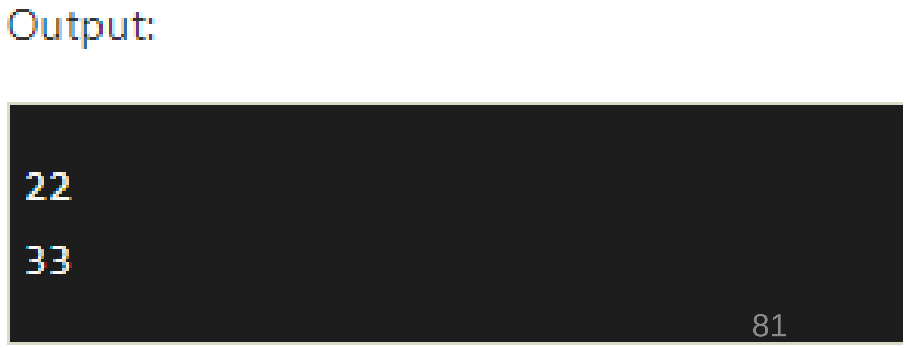
```
Area of  Square is    100
Area of Rectangle is    200
Area of Circle is    113.04000091552734
Press any key to continue . . .
```

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
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
```

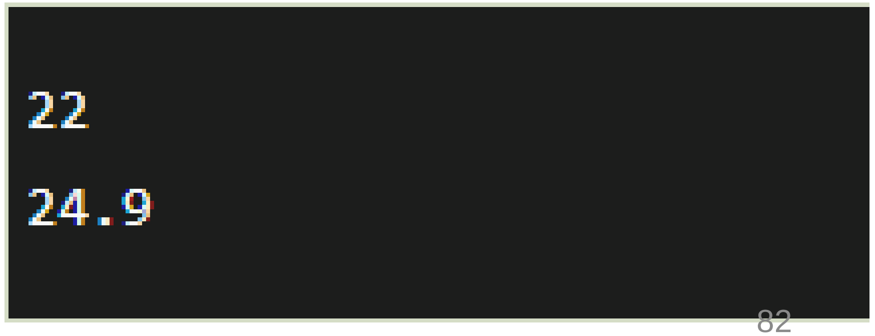
2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type.

The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder
{
        static int add(int a, int b)
        {
            return a+b;
         }
        static double add(double a, double b)
        {
                return a+b;
         }
}
```

Output:

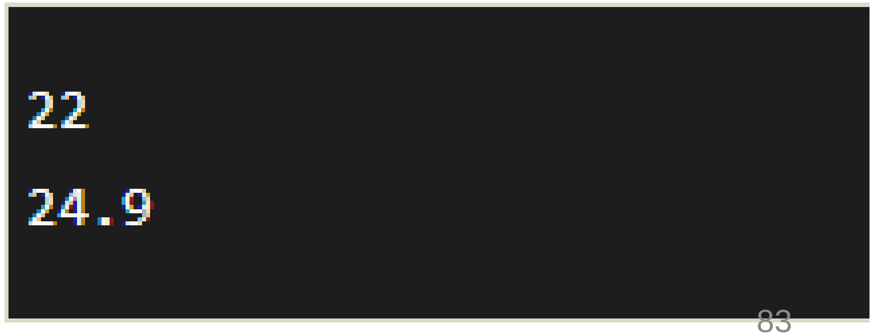


```
22
24.9
```

2) Method Overloading: changing data type of arguments

```
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
```

Method Overriding

- Method Overriding is a feature that allows us to redefine the method in the subclass or derived class which is already defined in its parent class or superclass.

- In any object-oriented programming language, we can implement Method Overriding only when two classes have ‘Is-a’ relationship of inheritance between them.

- Using Method Overriding, a derived class or child class can provide a specific implementation of a function or method that is already defined in one of its parent classes.

- When a method of a derived or sub-class has the same name, same return type or signature and with the same arguments as a method in its parent class, then we say that the method in the superclass is being overridden by the method in the subclass.

```
class Animal
{
       public void move()
       {
                System.out.println("Animals can move");
       }
}

class Dog extends Animal
{
      public void move()
      {
           System.out.println("Dogs can walk and run");
      }
}

public class TestDog
{

        public static void main(String args[])
        {
             Dog  b = new Dog();   //
           Animal obj2= new Dog();

              b.move();   // runs the method in Dog class
        }
}
```

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

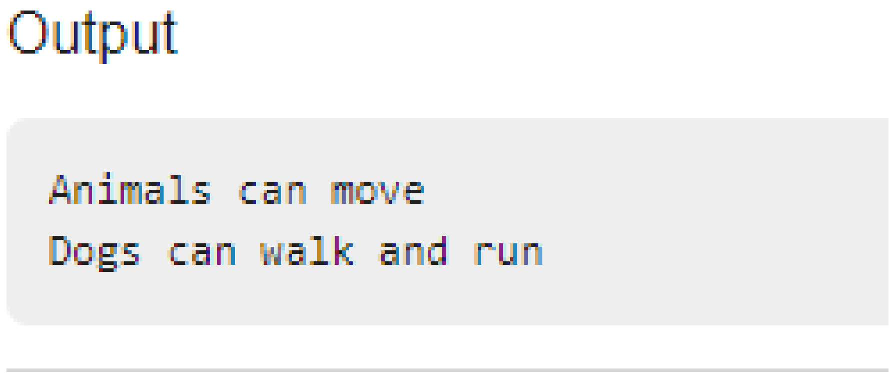
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

- ❖ super can be used to refer immediate parent class instance variable.
- ❖ super can be used to invoke immediate parent class method.
- ❖ super() can be used to invoke immediate parent class constructor.

Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

```
class Animal
{
      public void move()
      {
           System.out.println("Animals can move");
      }
}

class Dog extends Animal
 {
      public void move()
      {
           super.move();   // invokes the super class method
           System.out.println("Dogs can walk and run");
      }
}

public class TestDog
 {

   public static void main(String args[])
    {
                   Animal b = new Dog();   // Animal reference but Dog object
                   b.move();   // runs the method in Dog class
   }
}
```



Output

```
Animals can move
Dogs can walk and run
```