# Day-4 JA-111 Batch

## Topics

- **Constructor & Constructor Overloading**
- **The this keyword**
- **The static keyword**
- **Object collaboration with Has-a Relationship**

## Constructor

***What is difference between assignment and initialization?***

❖ Initialization refers to assigning value at the time of variable/object creation.
❖ Initialization can be done only one time for a variable.

✓ Assignment refers to assigning value to an existing variable/object.
✓ Assignment can be done only any number of times for a variable.

```
int a = 10;          //initialization
//int a  = 15;       Error, variable 'a' is already initialized
a = 20;              //Assignment
a = 30;              //okay
```

Now let us consider a situation, we have a database that have information of 50 students of a class. we have to create objects, each representing a Student. We have to open connection with database and read information of each student then create objects and then call a parameterized method that assign value to variables. In the whole processing we have to call parameterized method for all 50 objects. Java provides us a better way to do so using constructor. Constructor provides us ability to initialize object when they are created.

1. A constructor has name same as of its class name.
2. It should be defined inside the class hence It can access instance member directly without using object.
3. It is called automatically after object is created, before new operator completes.
4. A constructor does not have an return type because implicit return type of a constructor is object of class type itself.
5. A constructor is used for initialization and resource allocation to be done at the time object creation.

General form of constructor is

```
class-name(parameter-list){
  //body of constructor
}
```

An Example

```
//filename: Rectangle.java
package com.masai.p1_class_obj_demo;

public class Rectangle {
  double length;
  double breadth;
```

```
  Rectangle() {
    System.out.println("Inside constructor")
    length = 10.0;
    breadth = 15.0;
  }

  double getArea() {
    double area = length * breadth;
    return area;
  }
}

//filename: RectangleDemo.java
package com.masai.p1_class_obj_demo;

public class RectangleDemo {
  public static void main(String[] args) {
    Rectangle rectOne = new Rectangle();
    System.out.println("Area of rectOne is " + rectOne.getArea());

    Rectangle rectTwo = new Rectangle();
    System.out.println("Area of rectTwo is " + rectTwo.getArea());
  }
}
```

Output

```
Inside constructor
Area of rectOne is 150.0
Inside constructor
Area of rectTwo is 150.0
```

Following operation takes place

- Memory is allocated for an object
- Constructor is called and instance variables of object created in step 1, are initialized
- The reference is returned by new operator and stored in reference variable.

Before the example that we have just discussed we didn't use constructor in any example. If we don't create constructor in a class then java itself provides a constructor that initializes all instance variable to their default values. But if we have provided constructor to a class, Java remove its default constructor. Default values are as follows:

| Data Type | Default Value |
|---|---|
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0 |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' |
| References | Null |
| boolean | False |

**An Example**

```
//filename: Student.java
package com.masai.p1_class_obj_demo;

class Student{
  String name;
  int rollNo;
  boolean isPrefect;

  public static void main(String args[]){
    Student student = new Student();
    System.out.println(student.name + " " + student.rollNo + " " +
student.isPrefect);
  }
}
```

Output

```
null 0 false
```

In the Example discussed above we are using constructor that do not accept any parameter hence we are unable to initialize instance variable for each object uniquely. To address that problem we can use parametrized constructor. passing argument to constructor is similar to passing argument in method. Just we have to specify arguments at the time of object creation.

An Example

```
//filename: Rectangle.java
package com.masai.p1_class_obj_demo;

public class Rectangle {
  double length;
  double breadth;

  Rectangle(double l, double b) {
    length = l;
    breadth = b;
  }

  double getArea() {
    double area = length * breadth;
    return area;
  }
}

//filename: RectangleDemo.java
package com.masai.p1_class_obj_demo;

public class RectangleDemo {
  public static void main(String[] args) {
    Rectangle rectOne = new Rectangle(10.5, 20.5);
    System.out.println("Area of rectOne is " + rectOne.getArea());

    Rectangle rectTwo = new Rectangle(11.5, 22.5);
    System.out.println("Area of rectTwo is " + rectTwo.getArea());
  }
}
```

Output

```
Area of rectOne is 215.25
Area of rectTwo is 258.75
```

You can have more than one constructor in a class such that they must be separated by the parameter list. This is called **constructor overloading.**

An Example

```
//filename: Rectangle.java
package com.masai.p1_class_obj_demo;

public class Rectangle {
  double length;
  double breadth;

  Rectangle() {
    length = 10.5;
    breadth = 20.5;
  }

  Rectangle(double l, double b) {
    length = l;
    breadth = b;
  }

  double getArea() {
    double area = length * breadth;
    return area;
  }
}

//filename: RectangleDemo.java
package com.masai.p1_class_obj_demo;

public class RectangleDemo {
  public static void main(String[] args) {
    Rectangle rectOne = new Rectangle();
    System.out.println("Area of rectOne is " + rectOne.getArea());

    Rectangle rectTwo = new Rectangle(11.5, 22.5);
    System.out.println("Area of rectTwo is " + rectTwo.getArea());
  }
}
```

Output

```
Area of rectOne is 215.25
Area of rectTwo is 258.75
```

## this keyword

- It is a reference variable that points to the calling object of method & constructor
- It is maintained by the system, you cannot perform assignment operation on this keyword

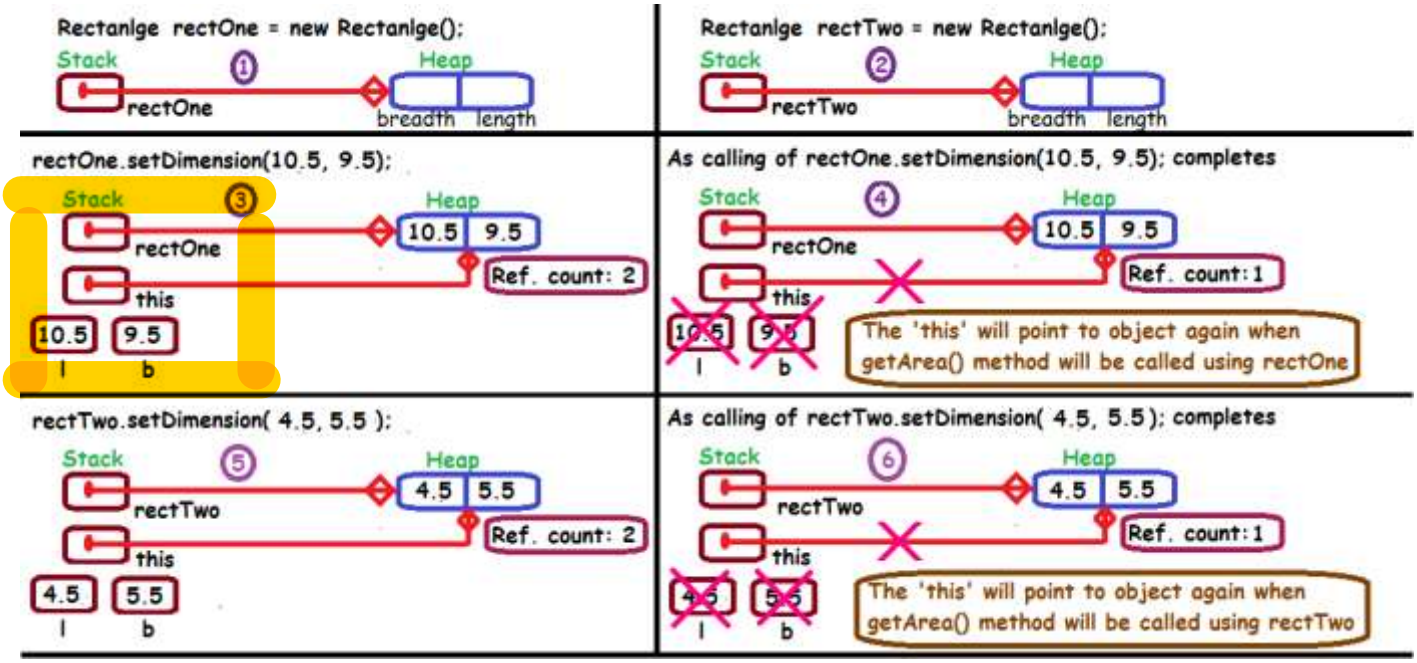
```
//filename: Rectangle.java
package com.masai.p1_class_obj_demo;
```

```
public class Rectangle {
  double length;
  double breadth;

  void setDimension(double l, double b) {
    length = l;
    breadth = b;
  }

  double getArea() {
    double area = length * breadth;
    return area;
  }
}

//filename: RectangleDemo.java
package com.masai.p1_class_obj_demo;

public class RectangleDemo {
  public static void main(String[] args) {
    Rectangle rectOne = new Rectangle();
    Rectangle rectTwo = new Rectangle();

    rectOne.setDimension(10.5, 9.5);
    rectTwo.setDimension(4.5, 5.5);

    System.out.println("Area of rectOne is " + rectOne.getArea());
    System.out.println("Area of rectOne is " + rectTwo.getArea());
  }
}
```

Output

```
10 45.25
15 65.25
```

Look at the diagram for the execution



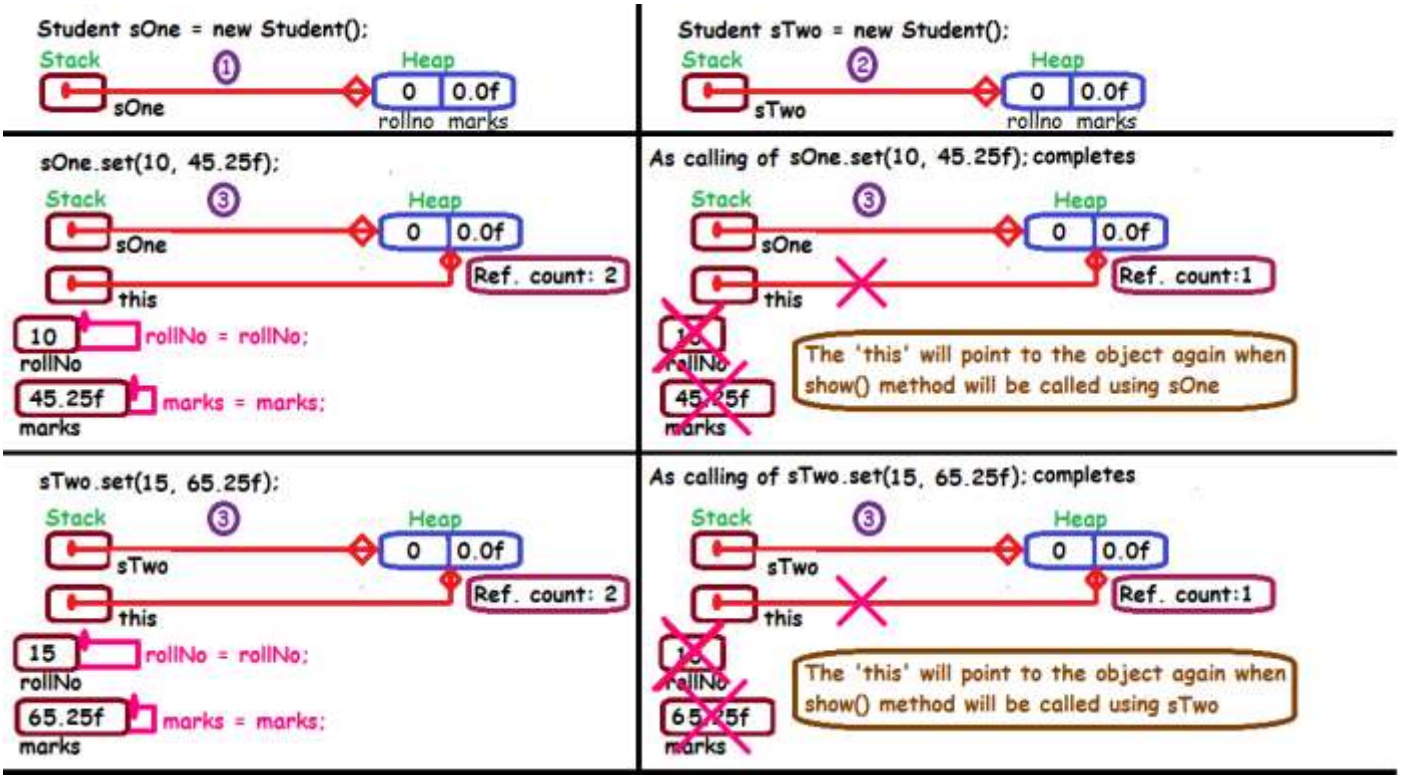
**Application of this keyword**

1. If local variables and instance variables of a class has same name then in the local scope (where local variables are created), local variables will hide the instance variables i.e. every time local variables will be accesses

```
//filename: Student.java
package com.masai.p1_class_obj_demo;

class Student{
  int rollNo;  //instance variables
  float marks;

  void set(int rollNo, float marks) {  //local variables
    rollNo = rollNo;  //local variable assigns value to itself
    marks= marks;  //local variable assigns value to itself
  }

  void show(){
    System.out.println(rollNo + " " + marks);
  }

  public static void main(String... args){
    Student sOne = new Student();
    Student sTwo = new Student();
    sOne.set(10, 45.25f);
    sTwo.set(15, 65.25f);
    sOne.show();
    sTwo.show();
  }
}
```

Output

```
0 0.0
0 0.0
```



Solution: use this keyword to access instance variable in presence of local variable with same name

```
  void set(int rollNo, float marks) {  //local variables
    this.rollNo = rollNo;
    this.marks= marks;
  }
```

Output

```
10 42.25
15 65.25
```

2. To call constructor of class from another constructor, make sure that the constructor calling must be the first statement.

```
//filename: Student.java
package com.masai.p1_class_obj_demo;

class Student{
  String name;
  float marks;
  int rollNo;

  Student(){
    name = "kapil";
  }

  Student(float m, int r){
    this();  //will call default constructor of Student class
    marks = m;
    rollNo = r;
  }

  void display(){
    System.out.println("Name = " + name + " marks = " + marks + " rollno = " +
rollNo);
  }

  public static void main(String args[]){
    Student sTwo = new Student(55.05f, 12);
    sTwo.display();
  }
}
```

Output:

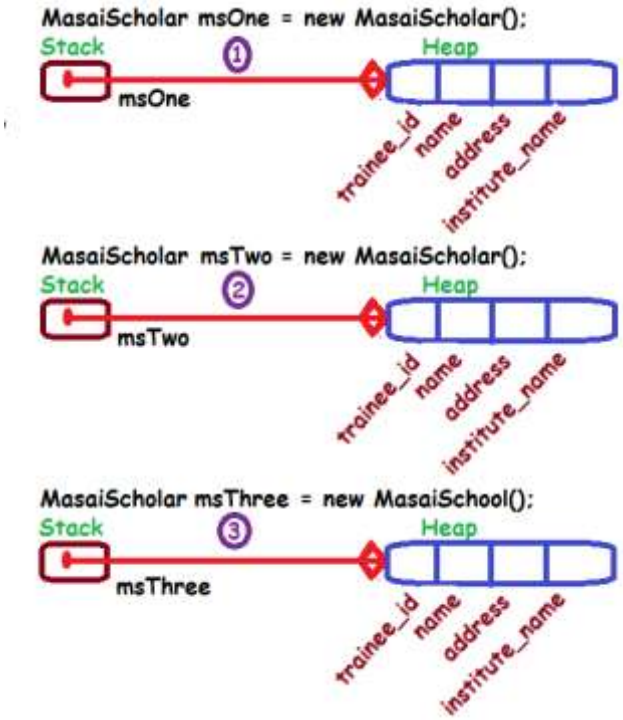
```
Name = kapil marks = 55.05 rollno = 12
```

## static keyword

Consider a situation for the institute Masai school, where want to maintain information about all the scholars such that the information is-

```
class MasaiScholar{
  int trainee_id;
  String name;
  String address;
```

```
  String institute_name;
}

MasaiScholar msOne = new MasaiScholar();
MasaiScholar msTwo = new MasaiScholar();
MasaiScholar msThree = new MasaiScholar();
```




here in the above example, we can see that the first three properties are different for every trainee but the fourth property that is institute_name is going to be common for all scholars so there is no logic of maintaining separate copy of this variable for all the scholars it will be good to maintain only one copy that is to be shared among all object.

***How to achieve this?***
**Solution:** use static with variables

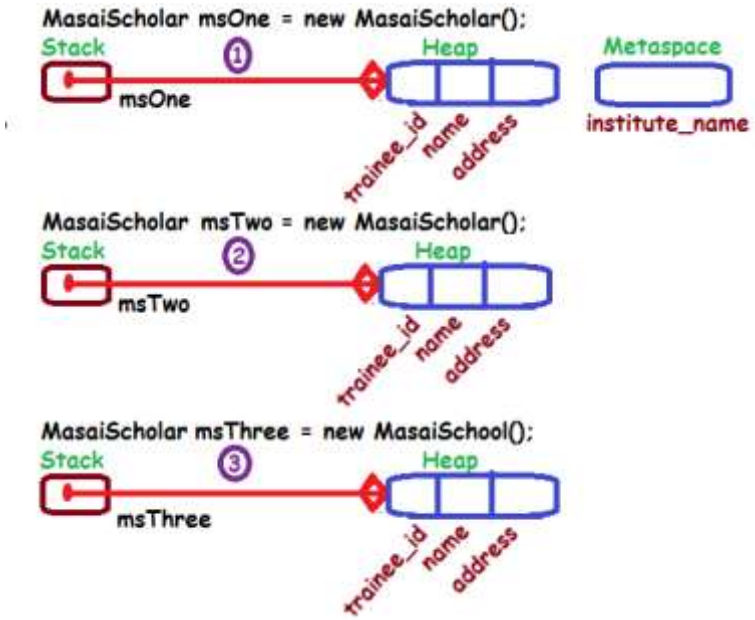
### static variable

1. Only a single copy of the static variable is maintained and this copy is shared among all the objects.
2. The static variable does not belong to object actually it belong to the class. It is also referred as class variable.
3. Because the static variable is for class, so to access the static variable, class-name can be used. Although accessing using the object name is also okay. Both (class-name and object name) will access the same variable
   class-name.variable-name; (Preferred way)
   obj-name.variable-name;
4. java does not allow the local static variable
5. local variables: stack; dynamic variable (i.e. object and array): heap; static variable: metaspace (before JDK 1.8 the space was PermGen)
   ***Tip:*** Metaspace is a memory area where static variables are created
6. Memory allocation for such variables happens only once when the class is loaded in the memory.

So the improvement for the above program is-

```
class MasaiScholar{
```

```
  int trainee_id;
  String name;
  String address;

  static String institute_name;
}

MasaiScholar msOne = new MasaiScholar();
MasaiScholar msTwo = new MasaiScholar();
MasaiScholar msThree = new MasaiScholar();
```




***We Activity***

An Example

```
package com.masai.p1_class_obj_demo;

class A{
  int i;    //instance variable
  float f;  //instance variable

  static double d;  //class variable

  public static void main(String.. args){
    System.out.println(A.d);  //0.0

    A aOne = new A();
    aOne.i = 10;
    aOne.f = 10.5f;
    aOne.d = 1000;

    System.out.println(aOne.i + " " + aOne.f + " " + aOne.d);  //10 10.5 1000.0
    System.out.println(A.d);  //1000.0

    A aTwo = new A();
    aTwo.i = 15;
    aTwo.f = 15.5f;
    aTwo.d = 2000;
```

```
    System.out.println(aOne.i + " " + aOne.f + " " + aOne.d);  //10 10.5 2000.0
    System.out.println(aTwo.i + " " + aTwo.f + " " + aTwo.d);  //15 15.5 2000.0
    System.out.println(A.d);  //2000.0

    A.d = 3000;
    System.out.println(aOne.d + " " + aTwo.d + " " + d);  //3000.0 3000.0
3000.0

    //static double k = 10;  Error, local variable cannot be static
  }
}
```

**static method**

1. To create a static method, we have to precede the method prototype (i.e. return-type method-name(para-list)) with static keyword
   static return-type method-name(para-list){
   ...body of the method...
   }
2. The static method belongs to the class, not to the object. So it can be called using class-name as well as using object-name. Both will call the same method
   class-name.method-name(argument-list); (Preferred way)
   object-name.method-name(argument-list);
3. Because the static method belongs to the class so this keyword (and super keyword also) is not accessible in the static method and due to absence of this keyword, non-static (i.e. instance variables + methods) are not accessible in the static context (i.e. method or block)
4. The non-static method (i.e. instance methods) can access the static variable directly because they are for class and can be accessed using the class name and the object name
5. The static method of a class can access all static variables and methods of the same class directly.
6. If programmer is creating local object/taking the object as parameter then it can access the instance variable using that local object. (because this keyword will not be involved)

***We Activity***

An Example

```
package com.masai.p1_class_obj_demo;

class A{
  int i;   //instance variable
  static int j; //class variable

  void set(){//instance method
    i = 10;      //instance method can access instance members
    j = 20;      //static variable can be accesses in non static method
  }

  static void show(){//class method
    //System.out.println("i = " + i); Error non-static is not accessessible in
static method
    System.out.println("j = " + j);   //static  variable  can  be  accesses  in
static method
  }

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

```
    A a = new A();
    a.set();
    a.show();    //ok... static memebers can be accessed using class name and
object-name

    //i = 100;  Error non-static is not accessessible in static method
    j = 200;

    show();      //static member can be accesses in static method

    a.j = 2000;
    A.show();    //static member can be accesses in static method
  }
}
```

**static block**

Consider the following program-

```
package com.masai.p1_class_obj_demo;
class P{
  int i;   //instance variable
  static int j; //class variable

  P(){
    i = 10;
    j = 20;
  }

  void show(){
    System.out.println(i + " " + j);
  }

  public static void main(String args[]){
    P p1 = new P();
    p1.show();

    j = 200;
    p1.show();

    P p2 = new P();
    p2.show();

    A.j = 200;
    p1.show();
  }
}
```

From the above example it is clear that the value of static variable is getting reset to 20 after every instantiation (i.e. object creation). so constructor is not a good place to provide value to a static variable. We have a better place that is static block

```
static{
  body of static block
```

```
}
```

- The static block of a class is executed first.
- It is executed even before main() method.

```
package com.masai.p1_class_obj_demo;
class P{
  int i;
  static int j;

  void show(){
    System.out.println(i + " " + j);
  }

  static{
    System.out.println("Inside static block");
    j = 20;
  }

  P(){
    System.out.println("Inside constructor");
    i = 10;
  }

  public static void main(String args[]){
    System.out.println("Inside main")
    P p1 = new P();
    p1.show();

    j = 200;

    p1.show();

    P p2 = new P();
    p2.show();

    A.j = 20;
    p1.show();
  }
}
```

Output

```
Inside static block
Inside main
Inside constructor
10 20
10 200
Inside constructor
10 200
10 20
```

### Using static to generate unique identifier

```
class Student{
```

```
  int rollNo;
  String name;
  static int counter;

  static{
    counter = 1;
  }

  Student(String name){
    rollNo = counter++;
    this.name = name;
  }

  public static void main(String args[]){
    Student s1 = new Student("Akshay");
    Student s2 = new Student("Chaitanya");
    Student s3 = new Student("Devendra");
    System.out.println(s1.rollNo + ": " + s1.name);
    System.out.println(s2.rollNo + ": " + s2.name);
    System.out.println(s3.rollNo + ": " + s3.name);
  }
}
```

Output

```
1: Akshay
2: Chaitanya
3: Devendra
```

***You activity***

Say we have a class whose objects are created in main method, other method of same class and methods of other class also. You need to write program to count total objects created of that class.

## The has-a relationship (Composition/Aggregation)

- It means that an instance of the one class has a reference to the instance of another class or the other instance of the same class.
- This relationship helps to minimize the duplication of code as well as the bugs.

An Example

```
package com.masai.p1_class_obj_demo;

class A{
      int i;
      int j;
}
```

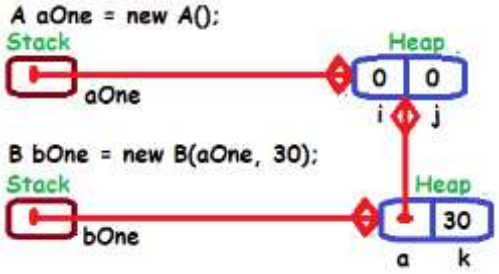
```
package com.masai.p1_class_obj_demo;
class B{
      A a;
      int k;

      B(A a, int k){
            this.a = a;
            this.k = k;
```

```
	}
}
```

```
package com.masai.p1_class_obj_demo;
class Demo{
	public static void main(String args[]){
		A aOne = new A();
		B bOne = new B(aOne, 30);
		System.out.println(bOne.aOne.i + " " + bOne.aOne.j + " " + bOne.k);
	}
}
```

Output

```
0 0 30
```




Another example

```
package com.masai.p1_class_obj_demo;

public class Address {
  String doorNo;
  String street;
  String city;
  int zipcode;

  Address(String doorNo, String street, String city, int zipcode) {
    this.doorNo = doorNo;
    this.street = street;
    this.city = city;
    this.zipcode = zipcode;
  }
}
```

```
package com.masai.p1_class_obj_demo;
class Customer {
  String customerId;
  String customerName;
  long contactNumber;
  Address address;

  Customer(String customerId, String customerName, long contactNumber, Address
address) {
    this.customerId = customerId;
    this.customerName = customerName;
    this.contactNumber = contactNumber;
    this.address = address;
  }
```

```
  void showCustomerDetails() {
    System.out.println("Displaying customer details");
    System.out.println("Customer Id: " + this.customerId);
    System.out.println("Customer Name: " + this.customerName);
    System.out.println("Contact Number: " + this.contactNumber);
    System.out.println("Address: " + ("Door No: " + address.getDoorNo() + "," +
address.getStreet() + ", " + address.getCity() + ", " + address.getZipcode()));
    System.out.println();
  }
}

package com.masai.p1_class_obj_demo;
public class MainDemo {
  public static void main(String[] args) {
    Address address = new Address("D101", "Park Street", "Mumbai", 400404);
    Customer customer = new Customer("C101", "kapil", 9865326598L, address);
    customer.showCustomerDetails();
  }
}
```

Output

```
Displaying customer details
Customer Id: C101
Customer Name: kapil
Contact Number: 9865326598
Address: Door No: D101,Park Street, Mumbai, 400404
```