

# JAVA CONSTRUCTORS

## CONSTRUCTOR

- A special method in java, which is used to initialize the data members of a class.
- It will be called automatically, when an object of a class is created.
- Constructors have the same name as the class name itself.
- It accepts four types of modifiers such as public, private, protected and default.
- It is [an optional](#). (Alternative to methods)

## Syntax

```
class-name(arg1, arg2,...argn)
{
    // instance variable '1'=arg'1'
    // instance variable '2'=arg'2'
    ...
    // instance variable 'n'=arg'n'
}
```

## Example

```
class JTest
{
    JTest()                // default constructor
    {
        // user code
    }
}
```

## Characteristics

- Constructor name and class name **should be same**
- It has **no return type** and **no dot operator**
- It will be called automatically, when the object is created
- Constructors are executed from **top to bottom order**.
- It can be overloaded
- A class **can have more than one constructor**
- **Only one default constructor** is allowed per class.

## Usage

- It is used to initialize the fields of a class (variables)
- It can be used to allocate the memory to the fields during object creation.

## DIFFERENCE BETWEEN CONSTRUCTOR AND METHOD

S.N	Constructor	Method
1.	Special method	Normal method
2.	<b>Implicit Calling</b> <ul style="list-style-type: none"><li>• It will be called automatically, when the object is created.</li></ul>	<b>Explicit Calling</b> <ul style="list-style-type: none"><li>• Method must be called by using object / class.</li></ul>
3.	It has no return type	It has return type

## Types

- Like c++, java provides three types of constructors. They are:
  1. Default Constructor (Parameter less Constructor)
  2. Parametrized Constructor (Constructor Overloading)
  3. Copy Constructor.

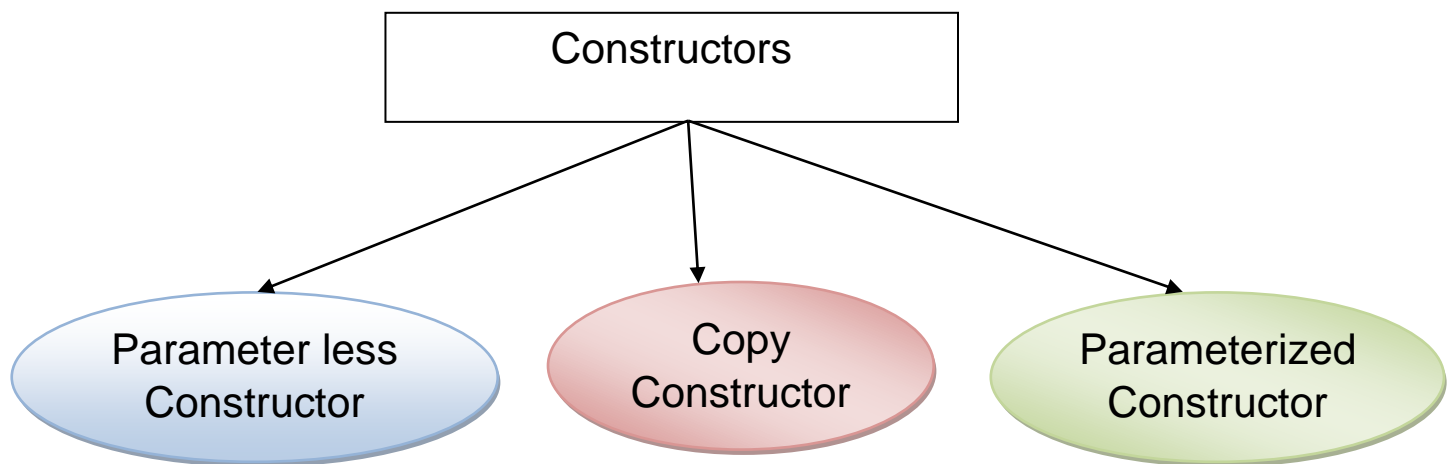


Figure: Types of Constructors

## Types of constructor

1. Parameter less constructor (Default constructor)
2. Parameterized constructor
3. Copy constructor

### 1. Parameter Less Constructor (Default Constructor)

#### Default Constructor

- If a constructor **doesn't take any arguments**, then it is called as default constructor / parameter less constructor.
- Only one default constructor is allowed per class.
- If no constructors are defined explicitly, then JVM will automatically add the default constructor to class

#### Use

- It is used to initialize the data members (variables) of a class.

## Calling

- It will be called automatically, when an object of a class is created.
- Implicit calling (No need to use an object to call constructor)

## I. USAGE OF DEFAULT CONSTRUCTOR

(DefaultConstructor.java)

### 1. SOURCE CODE

// main class

```
public class DefaultConstructor
```

```
{
```

```
    String name;
```

```
    int id;
```

// default constructor

```
public DefaultConstructor()
```

```
{
```

```
    name="Raman";
```

```
    id=339;
```

```
}
```

// instance method

```
public void disp()
```

```
{
```

```
    System.out.println("Name \t: "+name);
```

```
    System.out.println("Id \t: "+id);
```

Variables Initialization through  
default constructor

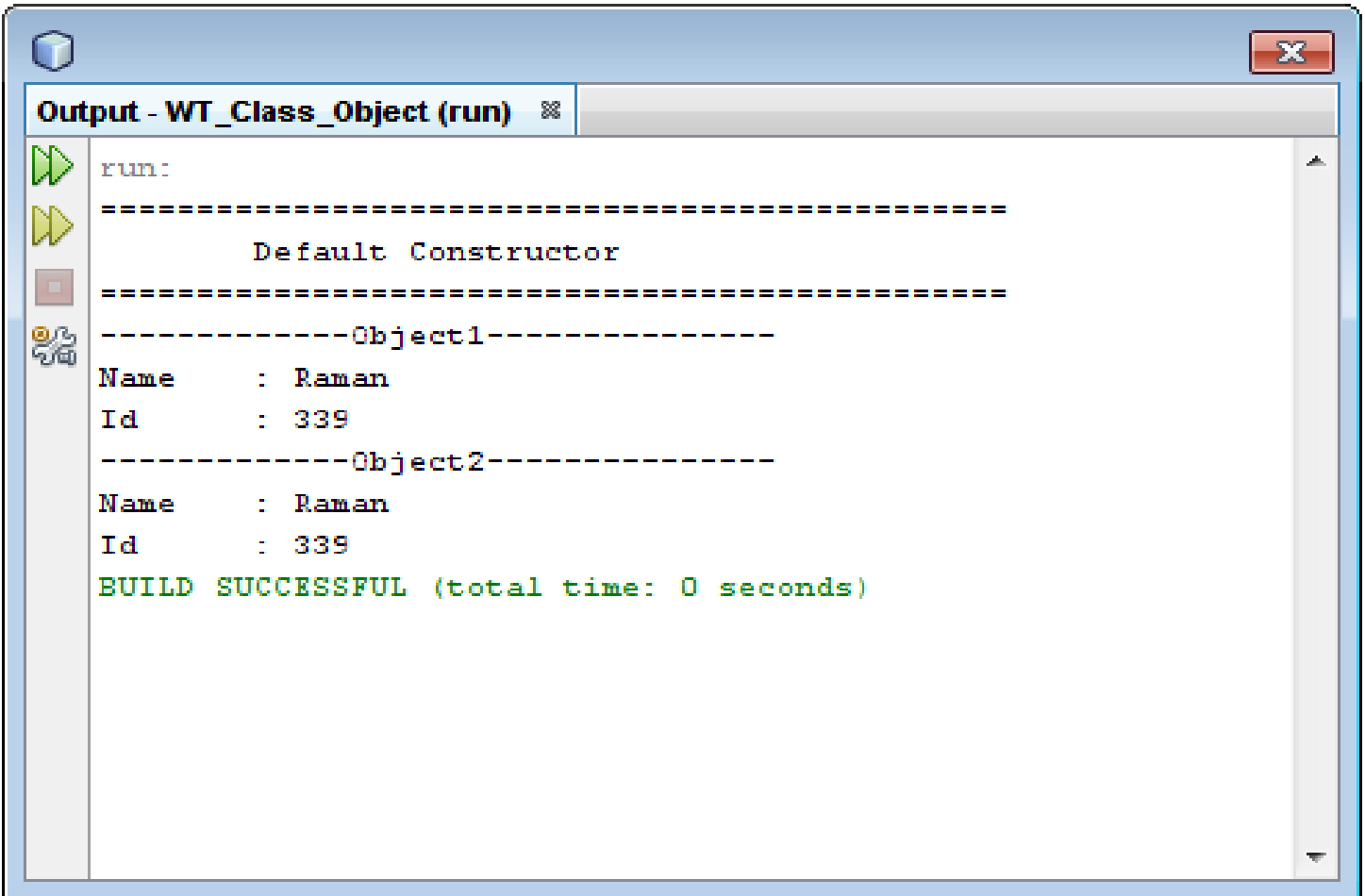


```
}  
  
// main method()  
public static void main(String[] args)  
{  
    // object creation  
    DefaultConstructor obj1=new DefaultConstructor();  
    DefaultConstructor obj2=new DefaultConstructor();  
    System.out.println("-----Object1-----");  
    obj1.disp();  
    System.out.println("-----Object2-----");  
    obj2.disp();  
}  
}
```

## Note

- It is important to note that, **each object has separate copy of their instance variables.**
- If there are any changes in an object **will not reflect / update** the members (instance variables) of other objects.

## 2. OUTPUT



```
run:
=====
      Default Constructor
=====
-----Object1-----
Name      : Raman
Id        : 339
-----Object2-----
Name      : Raman
Id        : 339
BUILD SUCCESSFUL (total time: 0 seconds)
```

### 2. Parameterized Constructor (Constructor Overloading)

- If a constructor takes the arguments, then it is called as parameterized constructor / Argument constructor
- The parameterized constructors are called by [using its signatures](#).
- Signature can be number of arguments / type of arguments
- The constructor supports [only argument based overloading](#). Because constructor [does not return any value \(no return type\)](#).

## Constructor Overloading

- If a same method (single method) performs different operations with **different set of signatures** (number of arguments), then it is called as constructor overloading
- This overloading is called with help of signatures.

## II. EXAMPLE OF PARAMETERIZED CONSTRUCTOR

(Empolyee.java)

### 1. SOURCE CODE

// main Class

```
public class Empolyee
{
```

// instance variable declarations

```
    String name;
```

```
    int id;
```

// public constructor

```
    public Empolyee(String str, int i)
```

```
    {
```

```
        name=str;
```

```
        id=i;
```

```
    }
```

// instance method

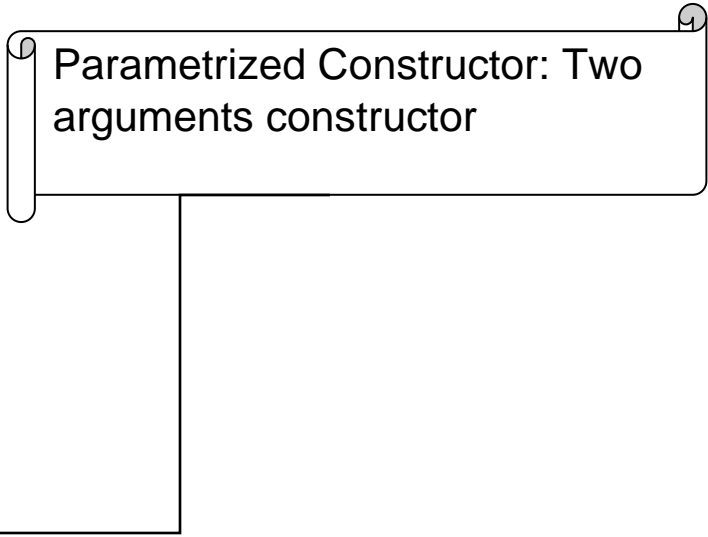
```
    public void disp()
```

```
    {
```

```
        System.out.println("Name \t: "+name);
```

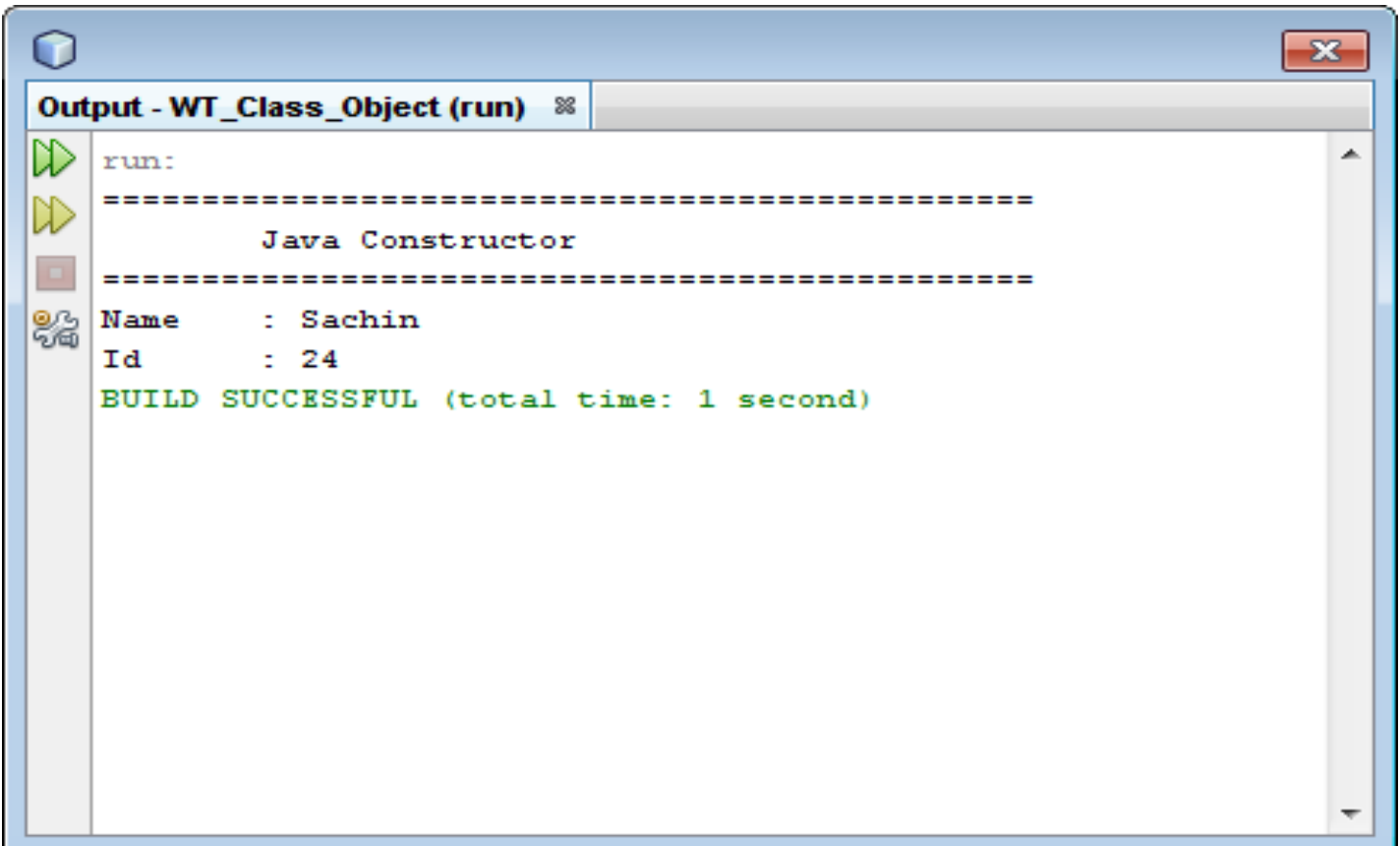
```
        System.out.println("Id \t: "+id);
```

Parametrized Constructor: Two arguments constructor

A callout box with a rounded rectangle and a small circle at the top right corner. It contains the text "Parametrized Constructor: Two arguments constructor". A line extends from the bottom left of the box, turns left, and then points to the constructor signature "public Empolyee(String str, int i)" in the source code.

```
}  
  
// main method()  
public static void main(String[] args)  
{  
    System.out.println("=====");  
    System.out.println("\tJava Constructor");  
    System.out.println("=====");  
    // object creation  
    Empolyee obj=new Empolyee("Sachin",24);  
    obj.disp();  
  
}  
}
```

## 2. OUTPUT



```
run:  
=====  
        Java Constructor  
=====  
Name      : Sachin  
Id        : 24  
BUILD SUCCESSFUL (total time: 1 second)
```



### III.CONSTRUCTOR OVERLOADING

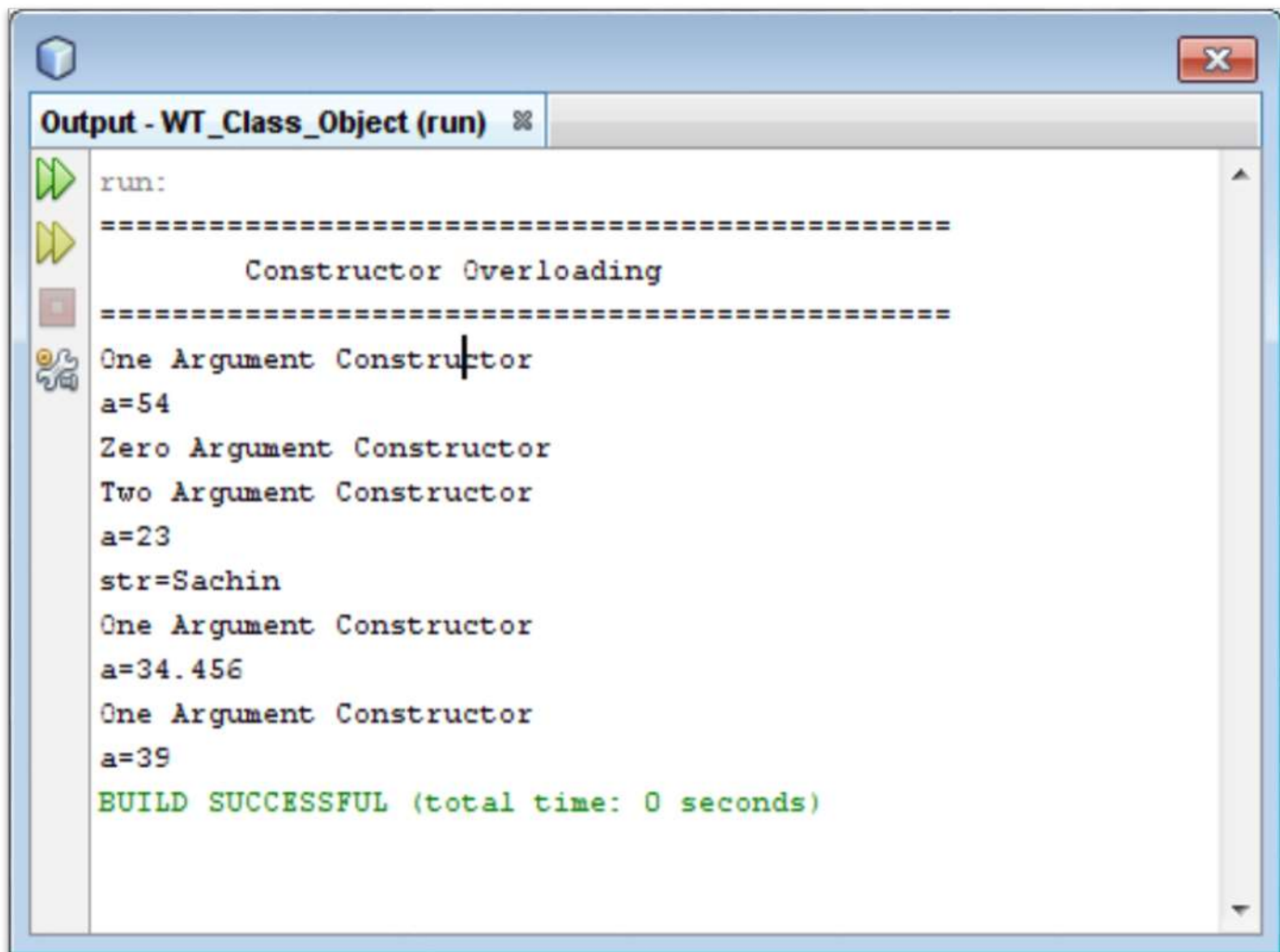
(Argu\_Overloading.java)

#### 1. SOURCE CODE

```
public class Argu_Overloading
{
// zero argument constructor
    Argu_Overloading()
    {
        System.out.println("Zero Argument Constructor");
    }
// one argument constructor
    Argu_Overloading(int a)
    {
        System.out.println("One Argument Constructor");
        System.out.println("a="+a);
    }
// one argument constructor
    Argu_Overloading(float a)
    {
        System.out.println("One Argument Constructor");
        System.out.println("a="+a);
    }
// two arguments constructor
    Argu_Overloading(int a, String str)
    {
        System.out.println("Two Argument Constructor");
        System.out.println("a="+a);
        System.out.println("str="+str);
    }
// main method
    public static void main(String[] args)
    {
```

```
System.out.println("=====");
System.out.println("\tConstructor Overloading");
System.out.println("=====");
// calling constructor overloading using its argument signatures
Argu_Overloading a1=new Argu_Overloading(54);
Argu_Overloading a2=new Argu_Overloading();
Argu_Overloading a3=new Argu_Overloading(23,"Sachin");
Argu_Overloading a4=new Argu_Overloading(34.456f);
Argu_Overloading a5=new Argu_Overloading(39);
}
}
```

## 2. OUTPUT



```
run:
=====
    Constructor Overloading
=====
One Argument Constructor
a=54
Zero Argument Constructor
Two Argument Constructor
a=23
str=Sachin
One Argument Constructor
a=34.456
One Argument Constructor
a=39
BUILD SUCCESSFUL (total time: 0 seconds)
```

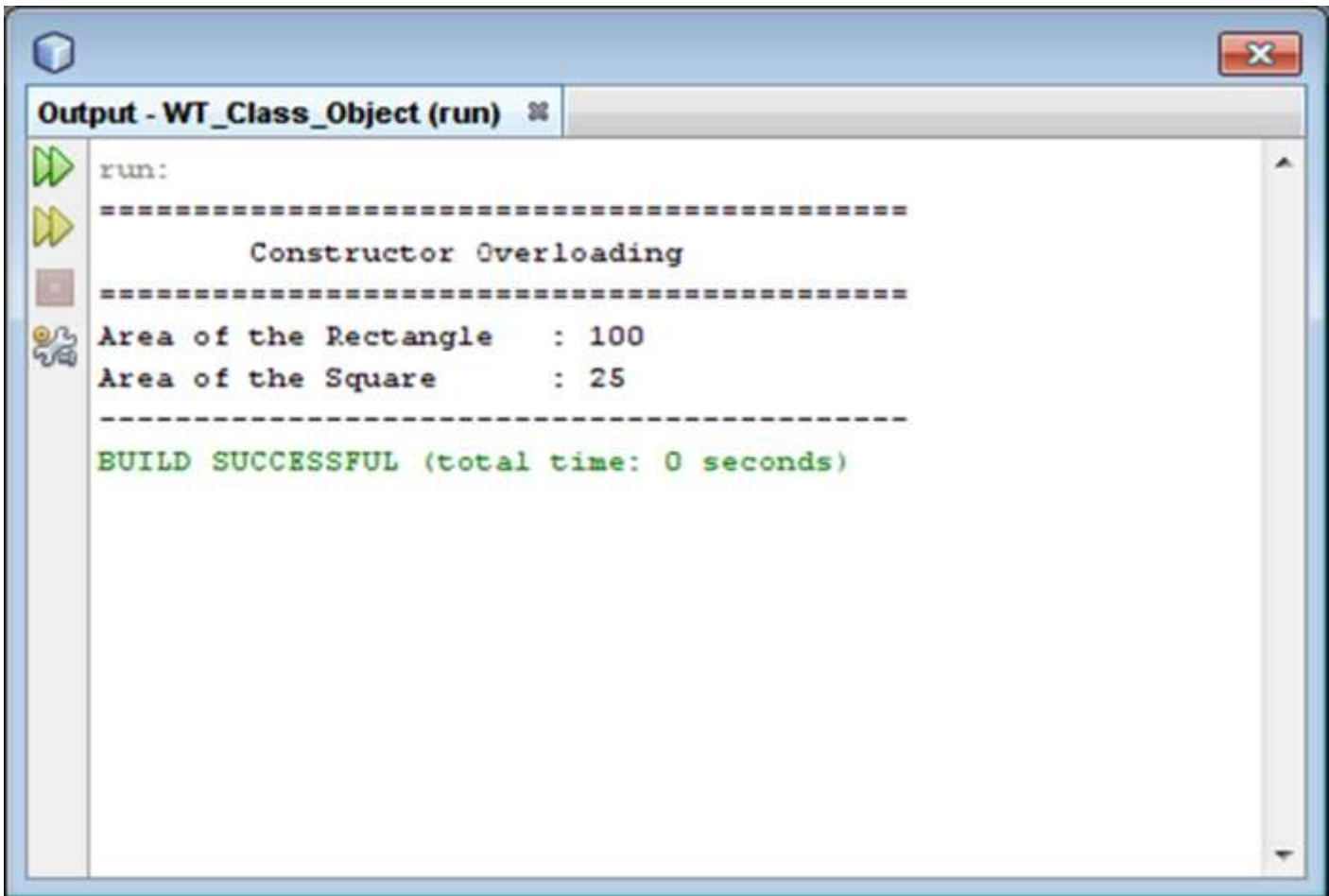
## IV.CONSTRUCTOR OVERLOADING (EXAMPLE 2)

(Argu\_Overloading2.java)

### 1. SOURCE CODE

```
public class Argu_Overloading2
{
// one argument constructor
    public Argu_Overloading2(int a)
    {
        int s=a*a;
        System.out.println("Area of the Square \t: "+s);
    }
// two arguments constructor
    public Argu_Overloading2(int l,int b)
    {
        int rs=l*b;
        System.out.println("Area of the Rectangle \t: "+rs);
    }
// main method
    public static void main(String[] args)
    {
        System.out.println("=====");
        System.out.println("\tConstructor Overloading");
        System.out.println("=====");
// calling constructor overloading using its argument signatures
        Argu_Overloading2 a1=new Argu_Overloading2(10,10);
        Argu_Overloading2 a2=new Argu_Overloading2(5);
        System.out.println("-----");
    }
}
```

## 2. OUTPUT



```
run:
=====
        Constructor Overloading
=====
Area of the Rectangle    : 100
Area of the Square      : 25
=====
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 3. Copy Constructor

- Process of copying the values of one object into another object, that is called as copy constructor
- In c++, **objects are value types**. So that, we should mention the copy constructor definition.
- But in java, objects are reference types (object types). So no need to explicitly define the copy constructor. Simply assign the original object to another object.

## V. USAGE OF COPY CONSTRUCTOR

(Copyconstructor.java)

### 1. SOURCE CODE

```
// main class
public class Copyconstructor
{
    // instance variable declarations
    String name;
    int id;
    // instance constructor
    Copyconstructor(String s, int i)
    {
        name=s;
        id=i;
    }
    // copy constructor Definition: optional
    /* Copyconstructor(Copyconstructor tmp)
    {
        name=tmp.name;
        id=tmp.id;
    }
    */
    // print output data
    void printResult()
    {
        System.out.println("Name \t:" + name);
        System.out.println("Id \t:" + id);
    }
}
```

## // main method

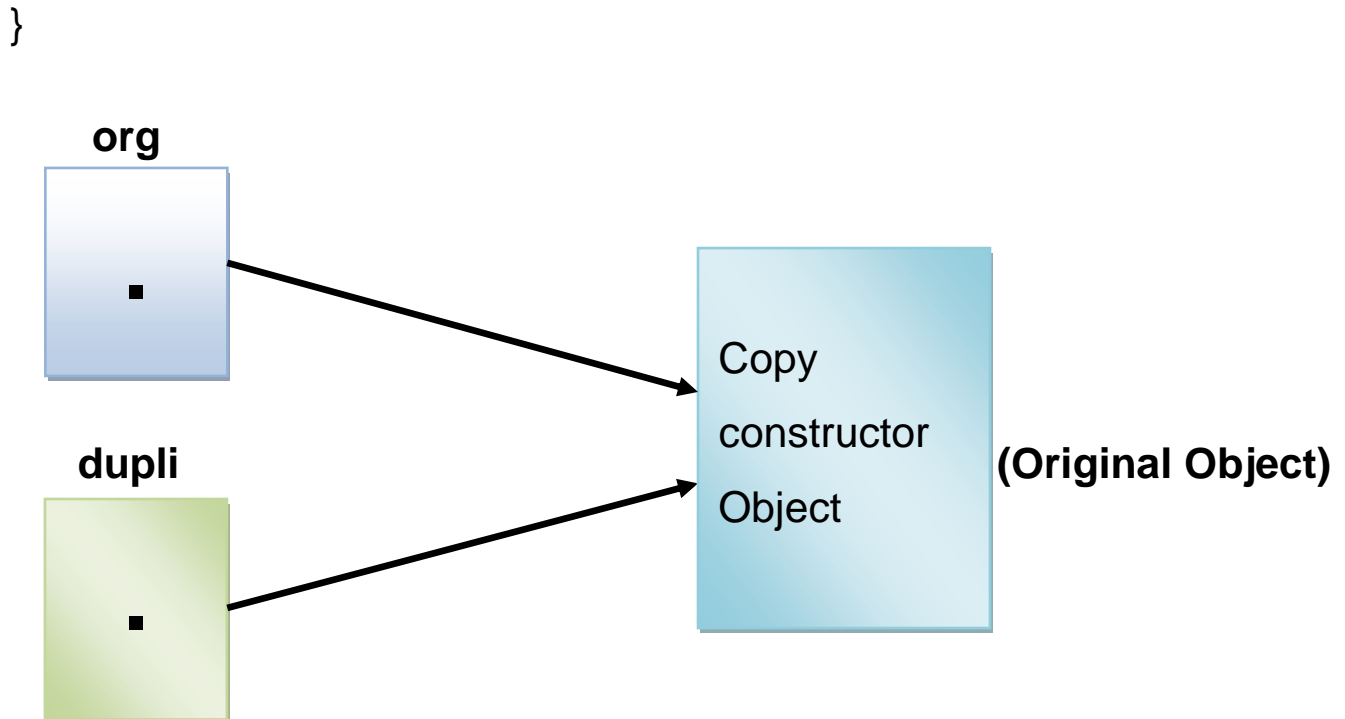
```
public static void main(String[] args)
{
    System.out.println("=====");
    System.out.println("\tCopy Constructor ");
    System.out.println("=====");
```

## //object creation using new modifier

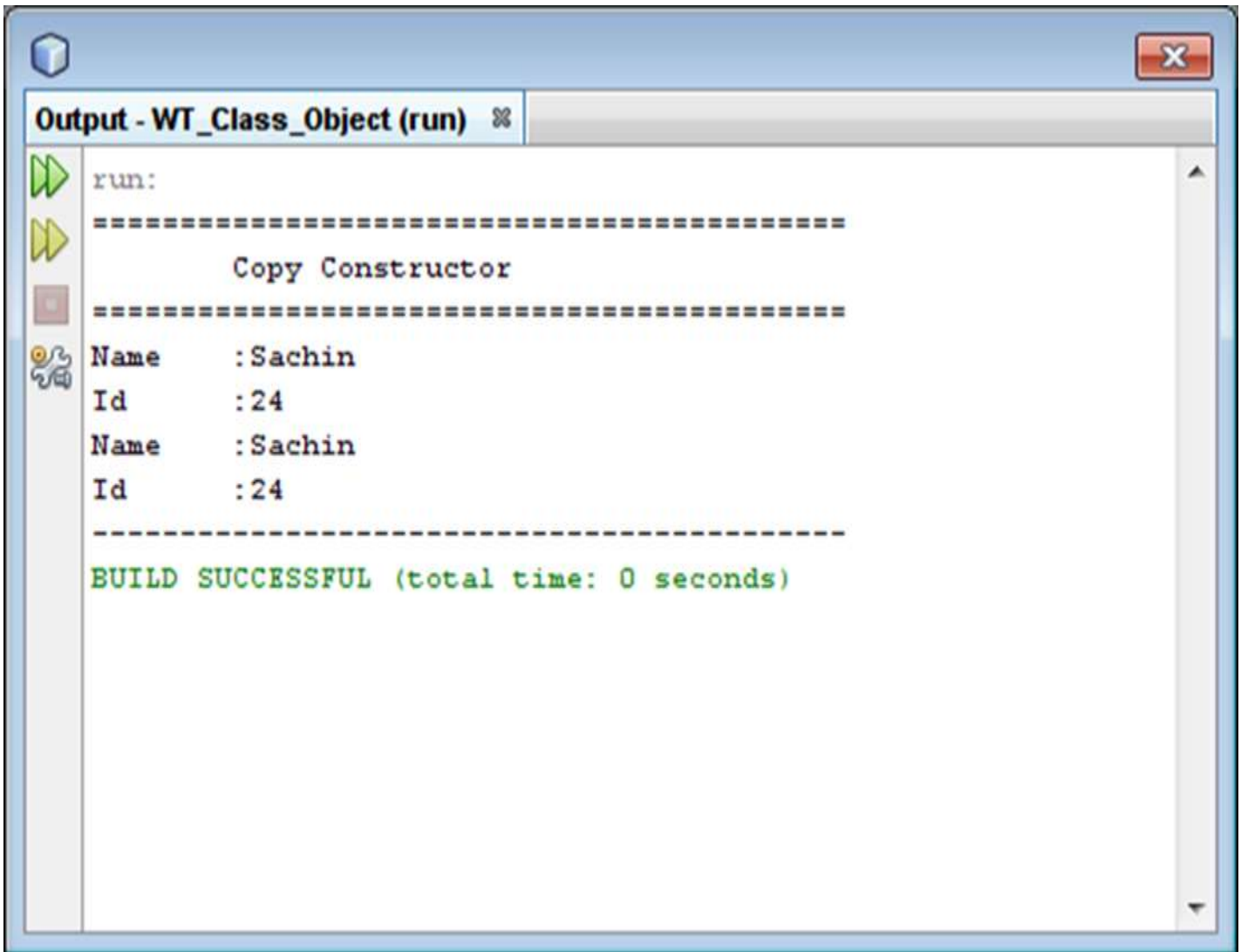
```
Copyconstructor org=new Copyconstructor("Sachin",24);
```

## // set object reference 'org' to object alias 'dupli' (Copy Constructor)

```
Copyconstructor dupli=org;
org.printResult();
dupli.printResult();
System.out.println("-----");
```



## 2. OUTPUT



```
run:
=====
      Copy Constructor
=====
Name    : Sachin
Id      : 24
Name    : Sachin
Id      : 24
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Constructor Chaining

- Process of [calling one constructor from another constructor](#), that is called as constructor chaining
- This is implemented with help of [this](#) keyword and [super](#) keyword
- This keyword is used to call the same class constructors in the constructor chaining
- Super keyword is used to call the super class constructor in the constructor chaining

## This

- It refers the instance (object) of current class
- It is used to [call the same class constructors](#)

## Importance of constructor chaining

- Instead of creating multiple objects for each constructor definition, we can create single object for all constructor definitions with help of this constructor chaining.

## Rules for constructor chaining

1. This() expression must be the [first statement of the constructor definition](#)
2. [Class must include at least one constructor without the use of this\(\) expression](#)
3. This() will call the default constructor and this([parameter](#)) will call the parameterized constructors
4. Like this(), super() expression will call the default constructor of super class and super([parameter](#)) will call the parameterized constructors of the super class
5. Constructor chaining can be done in any order.



## VI. CONSTRUCTOR CHAINING

(JTest.java)

### 1. SOURCE CODE

```
public class JTest
{
// default constructor
    JTest()
    {
// calling one argument constructor
        this(17);
        System.out.println("Default Constructor");
    }
// one argument constructor
    JTest(int id)
    {
// calling two arguments constructor
        this("Vijay","Salem");
        System.out.println("Id \t: "+id);
    }
// two arguments constructor
    JTest(String name, String place)
    {
        System.out.println("Name \t: "+name);
        System.out.println("Place \t: "+place);
    }
// main method
    public static void main(String[] args)
    {
        System.out.println("-----");
        System.out.println("\t Constructor Chaining");
        System.out.println("-----");
    }
}
```

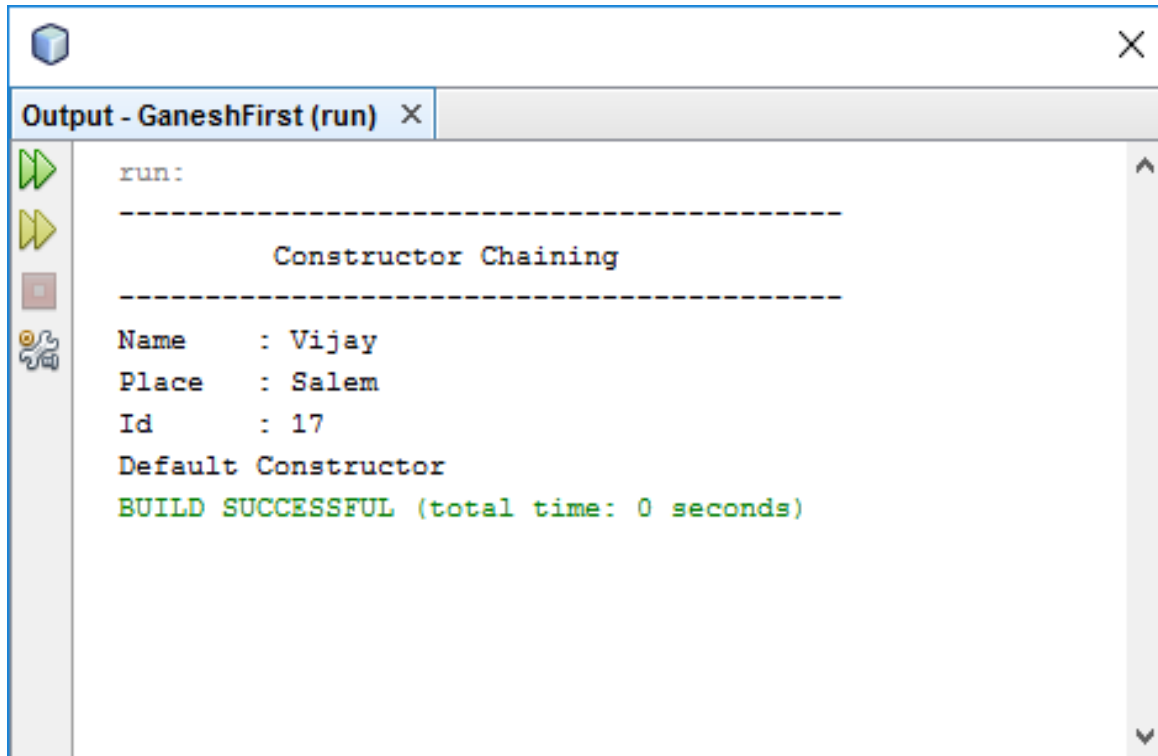
```
// object creation for current class
```

```
JTest obj=new JTest();
```

```
}
```

```
}
```

## 2. OUTPUT



## CONSTRUCTORS IN INHERITANCE

### Sub class Constructor

- Constructors are **not inherited in sub class (derived class)**. But a sub class constructor can call the super class constructor.
- This is done by **using super keyword**.
- A subclass constructor is used to **initialize both the sub class and super class variables (instance & static variables)**.

### Syntax

```
class-name(arg1, arg2,...argn)  
{  
    super(a1, a2, ... an)      ← calling super class constructor  
    // variable1=arg1;  
    // variable2=arg2;  
    ...  
    // variablen=argn;  
}
```

### Rules

- Super keyword is **used only in a sub class constructor**.
- Super is used to **invoke only super class constructor** (not sub class constructor)
- The parameters in the super call must match the order and type of parameters defined in the super class constructor.**
- The call to super call (super class constructor) must appear as the first statement within the subclass constructor.** Otherwise it will provide the error message.

## I. CONSTRUCTOR IN INHERITANCE

(SubConstructor.java)

Inheritance Type : Single Inheritance

### 1. SOURCE CODE

// super class definition

```
class Person
```

```
{
```

// instance variable declaration

```
    String name;
```

```
    int id;
```

// super class constructor

```
    Person(String n, int i)
```

```
    {
```

```
        name=n;
```

```
        id=i;
```

```
    }
```

```
}
```

// sub class definition

```
class Employee extends Person
```

```
{
```

```
    double sal;
```

```
    String loc;
```

// subclass Constructor

```
    Employee(String s, int i, double d, String l)
```

```
    {
```

// calling super class constructor

```
        super(s,i);
```

```
        sal=d;
```

```
        loc=l;
```

```
    }
```

It must be a **first statement** in the sub class constructor.

```
// print output data
```

```
void disp()
{
    System.out.println("Name \t: "+name);
    System.out.println("Id \t: "+id);
    System.out.println("Salary \t: "+sal);
    System.out.println("Place\t: "+loc+"\n");
}
}
```

```
// main class
```

```
public class SubConstructor
{
```

```
// main method
```

```
public static void main(String[] args)
{
    System.out.println("=====");
    System.out.println("\tSub Class Constructors in Inheritance");
    System.out.println("=====");
}
```

```
// sub class object creation & calling sub class constructor
```

```
Employee obj=new Employee("Sachin",73,95000.75,"Chennai");
```

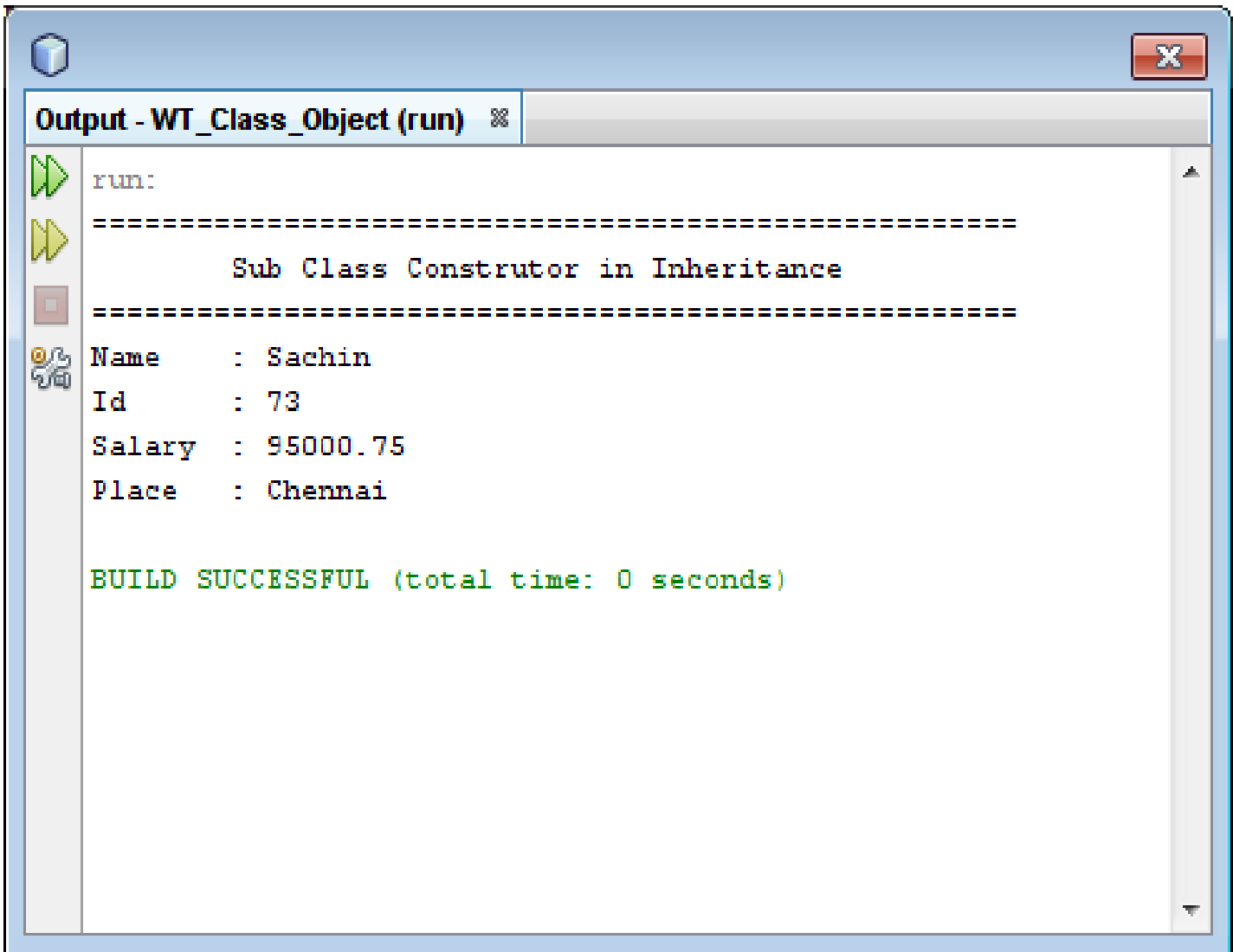
```
// calling sub class instance method
```

```
obj.disp();
```

```

}
}
```

## 2. OUTPUT



```
run:
=====
      Sub Class Construtor in Inheritance
=====
Name      : Sachin
Id        : 73
Salary    : 95000.75
Place     : Chennai

BUILD SUCCESSFUL (total time: 0 seconds)
```

### Important Points about constructors in Inheritance

1. If you did not call any constructors of super class in inheritance, then the compiler will automatically call the default constructor of a super class (parent class)
2. If you did not call any constructors of super class and did not create default constructor of the super class in inheritance, then the compiler will provide the error message. So we must define the default constructor of the super class (base class) in inheritance.

**Case 1:**

1. If you did not call any constructors of super class in inheritance, then the compiler will automatically call the default constructor of a super class (parent class) for every object creation of sub class.

**II. CONSTRUCTOR IN INHERITANCE (Example 2)**

(D.java)

Inheritance Type : Single Inheritance

**1. SOURCE CODE**

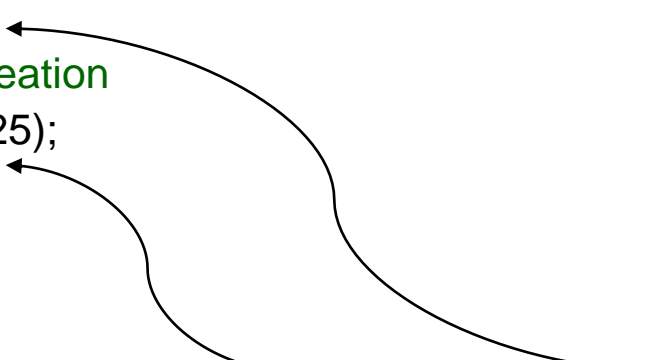
// super class definition

```
class S
{
    S()
    {
        System.out.println("Super class");
    }
    S(int k)
    {
        System.out.println("k="+k);
    }
}
```

// sub class definition

```
class D extends S
{
    D()
    {
        System.out.println("Derived class");
    }
    D(int g)
    {
        System.out.println("g="+g);
    }
}
```

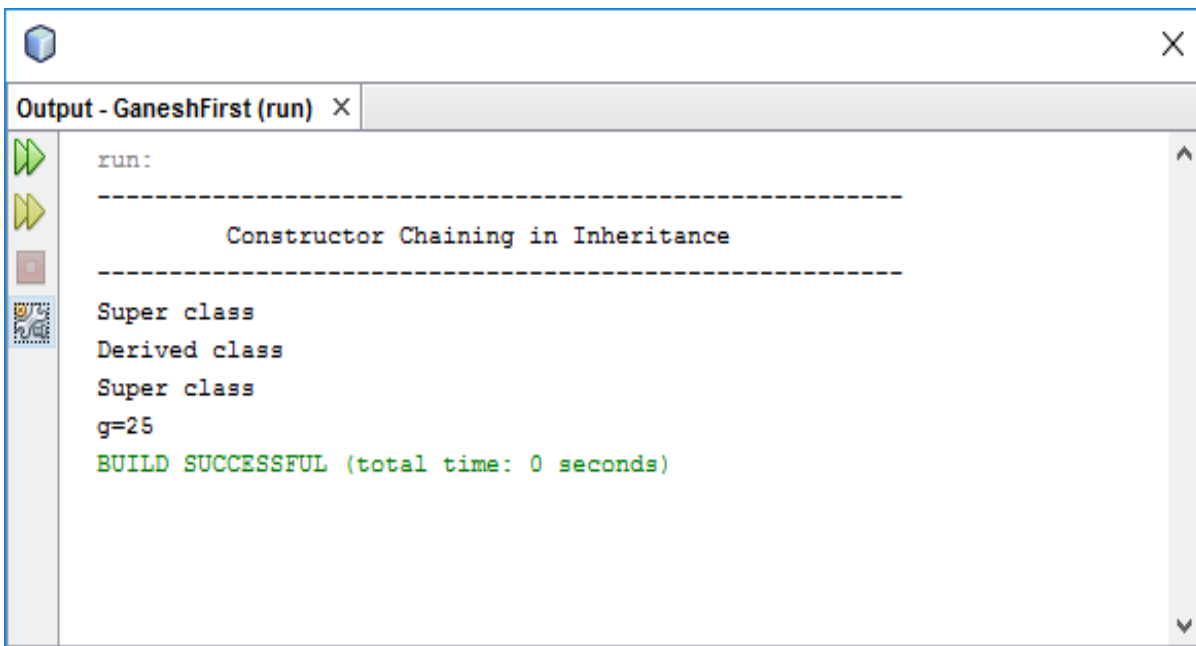
```
    }  
}  
// main class  
public class JTest1  
{  
    // main method  
    public static void main(String[] args)  
    {  
        System.out.println("-----");  
        System.out.println("\t Constructor Chaining in Inheritance");  
        System.out.println("-----");  
        // sub class object creation  
        D obj=new D();  
        // sub class object creation  
        D obj1=new D(25);  
    }  
}
```



Here, compiler called the default constructor of super class **two times**. Because, **this program did not call any constructors of the super class.**



## 2. OUTPUT



```
run:
-----
      Constructor Chaining in Inheritance
-----
Super class
Derived class
Super class
g=25
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Case 2:

1. If you did not call any constructors of super class and you **did not create default constructor of the super class** in inheritance, then the **compiler will provide the error message**. So we must define the default constructor of the super class (base class) in inheritance.

## III. CONSTRUCTOR IN INHERITANCE (Example 2)

(D.java)

Inheritance Type : Single Inheritance

### 1. SOURCE CODE

// super class definition

```
class S
{
    S(int k)
    {
        System.out.println("k="+k);
    }
}
```

In this inheritance, there is no default constructor in the super class. So compiler will provide the error message.

// sub class (derived class) definition

```
class D extends S
```

```
{
    D()
    {
        System.out.println("Derived class");
    }
    D(int g)
    {
        System.out.println("g="+g);
    }
}
```

// main class

```
public class JTest1
```

```
{
```

// main method

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

```
    {
```

```
        System.out.println("-----");
```

```
        System.out.println("\t Constructor Chaining in Inheritance");
```

```
        System.out.println("-----");
```

// sub class object creation

```
        D obj=new D();
```

```
    }
```

```
}
```

## 2. OUTPUT (ERROR MESSAGE)

Exception in thread "main" java.lang.Error: Unresolved compilation problems:

Implicit super constructor S() is undefined. Must explicitly invoke another constructor, etc, ...

## DESTRUCTOR

- It is also member of a class.
- Destructor is called automatically during the object destruction
- It has same name as the class name, preceded by a tilde (~) symbol.
- It is always executed from bottom to top (reverse order)
- It does not support overloading. Because it does not take any arguments.

### Java Destructor

- Java does not support destructor. Because java is a garbage collected language
- Garbage Collection
  - Process of removing the object from a running program is called as garbage collection.
  - This is automatically handled by JVM in java
- The JVM handles automatic memory deallocation (destruction of objects is called as garbage collector)
- There is no syntax for java destructor. Objects are destructed but there is no destructor
- We cannot predict when an object will be destroyed. This is automatically handled by JVM.